LLMATCH: a Unified Schema Matching Framework with Large Language Models

No Author Given

No Institute Given

Abstract. Schema matching is crucial for data integration and involves finding matching columns across schemas. Traditional tools focus on simple single table mappings while multi-table matching remains a challenging task. We present LLMATCH, a solution for complex schema matching tasks. LLMATCH decomposes the problem into three modular components: schema preparation, table selection, and column matching, each supporting multiple configurable options to allow flexible customization of various method. On top of the framework, we introduce Rollup and Drilldown, a multi-level schema matching mechanism that first aggregates semantically related columns (Rollup) before high-level matching and later expands them back into their original components (Drilldown) for detailed alignment. To support this effort, we created a benchmark dataset with seven multi-table schema pairs from three industries. We used this dataset to test configurations and improve state-of-the-art performance. The experiments provided insights for schema matching and LLM applications. Tests with human data engineers showed significant productivity improvements in business scenarios.

Keywords: Schema Matching · LLM · Data Management

1 Introduction

Schema matching is an important task in database systems. It supports applications like data integration, warehousing, and exchange. The process identifies semantic correspondences between columns of source and target schemas, allowing data from various systems to be transformed into a unified format. Industries such as healthcare [39], energy [35], and agriculture [6] are adopting standardized data formats to improve interoperability. In healthcare, a unified data model [47] can facilitate cross-institutional research and speed up new treatment discovery [33,29,30]. Each country performs schema matching to convert their healthcare data into this unified format [21,17,31,15,22]. In the business world, companies match and migrate data during mergers and acquisitions (M&As). Errors in this process can cause significant disruptions. For example, Deutsche Bank's 2023 merger with Postbank resulted in account lockouts due to data migration issues, leading to regulatory scrutiny and reputational damage [14].

Classical schema matching tools and benchmarks are designed for tabular data in CSV files, but their accuracy drops sharply on real-world schemas due

to inconsistent naming conventions and hierarchical structures [51]. For instance, in the toy matching task shown in Fig.1, fields like *PersonID*, *CustomerID*, and *ContactID* differ lexically but are considered matches. Consequently, complex matching scenarios often depend on domain experts to manually create and maintain mappings[9], a process that can take up to 500 hours for two experts to map a single schema [37] in some cases. This approach is not only time-intensive but also prone to errors [2,44].

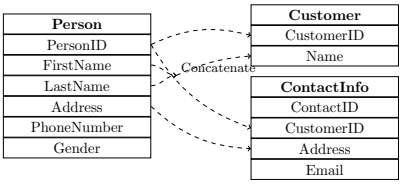


Fig. 1: Toy schema matching example.

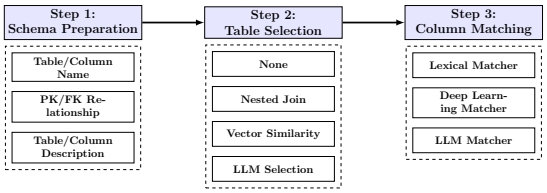


Fig. 2: Workflow design for schema matching task.

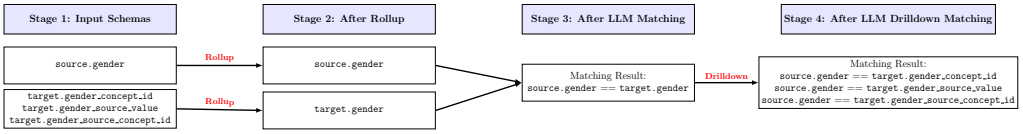


Fig. 3: LLMatch: A Multi-Level Schema Matching Approach with Rollup and Drilldown

Recently, Large language models (LLMs) have shown great potential in tackling complex database tasks, including entity resolution, data lake discovery, and query synthesis [49,12]. While several LLM-based schema matching tools [43,18] have demonstrated promising capabilities, fair comparison remains challenging due to the lack of a standardized evaluation framework and appropriate datasets. Existing methods address tasks with varying complexities: *table-to-table* matching involves aligning columns in isolated tables, while *schema-to-schema* matching requires understanding relationships across entire schemas, such as primary key-foreign key mappings. Most evaluation datasets focus on *table-to-table* matching and fail to capture the broader scope of schema-level tasks. This disparity in evaluation settings obscures the strengths and weaknesses of different approaches and emphasizes the need for a unified framework with datasets that address both simple and complex matching tasks.

In this paper, we introduce a unified framework that divides the schema matching process into three phases: *schema preparation*, *table selection*, and *column matching* (Fig. 2). Schema preparation identifies the components of the source and target data, including pre-processing transformations such as Rollup, where semantically related columns are merged to facilitate high-level matching. Table selection determines the most relevant target tables for each source table, reducing the context size for the next phase. After selecting target tables, the column matching phase aligns columns between each source table and its corresponding target tables, generating the final mappings. Once a high-level match is established, the Drilldown process expands the rolled-up columns back into their original components, ensuring a fine-grained alignment of schema elements.

To evaluate our framework, we curated seven benchmark datasets spanning finance, healthcare, and entertainment domains. These schemas average 14 tables and 135 columns, in sharp contrast to simpler benchmarks, which typically include just one table and 15 columns (Table 1).

Using the proposed framework and benchmark, we conducted extensive experiments with state-of-the-art methods and uncovered several key insights:

- **Curating High-Quality Candidates:** Presenting all target tables in a single prompt with a large context size or processing them individually across multiple prompts with a small context size often leads to suboptimal performance. Instead, selecting a subset of the most relevant target tables while maintaining a moderate context size yields more effective results.
- **Target Table Selection Methods Are Critical:** Embedding vector similarity, commonly used in the RAG framework, is not ideally suited for schema matching tasks. Our experiments indicate that allowing the LLM to directly handle target selection often produces better results, highlighting the need for more adaptive and context-aware approaches.
- **Hierarchical Matching with Rollup and Drilldown for Complex Schema Alignment:** The Rollup and Drilldown operations are essential for managing complexity at different stages of the matching process. By first simplifying the schema through Rollup and then applying Drilldown once a general term is matched, this approach effectively improves performance on complex datasets.

These findings render the interplay between model capabilities, context quality, and schema complexity, offering insights for advancing schema matching solutions. We release our framework as well as the new benchmark datasets[5].

2 Problem Statement

A database schema S consists of a set of tables T, columns A, descriptions of tables and columns D, and relationships R, including primary keys (PK) and foreign keys (FK). Given two database schemas S_s (source) and S_t (target), the objective is to map tables and columns from S_s to those in S_t.

Motivation for n:m Mapping. In many real-world data integration scenarios, especially those involving relational databases with multi-table structures, matching often requires mapping multiple columns from the source schema to multiple columns in the target schema. We refer to this as an n:m mapping. This pattern, while expansive, helps ensure that important relationships are captured, even if it occasionally results in higher false positives. For most industrial applications, prioritizing recall (minimizing missed matches) is more desirable than optimizing precision, as it reduces the risk of missed correspondences [36].

Definition 1. *In an n:m match, the task is to find a mapping* $\Psi : A_s \rightarrow A_t$*, where each column* $a \in A_s$ *is associated with a subset* $\Psi(a) \subseteq A_t$*.*

Metrics. We evaluate the performance of schema matching using the F1 score, which is calculated based on the precision and recall between the predicted mapping Ψ and the ground truth mapping $\hat{\Psi}$, defined as follows:

$$\text{precision}(\Psi, \hat{\Psi}) = \frac{|\{(a,b) \mid b \in \Psi(a) \cap \hat{\Psi}(a)\}|}{|\{(a,b) \mid b \in \Psi(a)\}|}, \quad \text{recall}(\Psi, \hat{\Psi}) = \frac{|\{(a,b) \mid b \in \Psi(a) \cap \hat{\Psi}(a)\}|}{|\{(a,b) \mid b \in \hat{\Psi}(a)\}|}$$

3 Dataset Development

Traditional schema matching benchmarks typically involve a single source and target table, as traditional NLP tools struggle to interpret PK/FK relationships in relational schemas. More recent works on complex schema matching primarily uses two healthcare datasets: *mimic-omop* and *synthea-omop*[43,18]. We expand this scope to include seven pairs of source and target schemas spanning healthcare, entertainment, and finance, creating a more diverse benchmark. For healthcare data, we leverage mappings from prior research [43,37] and official mapping [29,9]. For entertainment data, we use ground truth mappings from [50]. In the finance domain, we release two test schemas along with their ground truth mappings annotated by industry experts.

Table 1: Dataset statistics for traditional benchmarks (top) and the complex benchmark (bottom) we developed.

Mapping Task	Domain	Src # Tables/-Cols/PKs/FKs	Tgt # Tables/-Cols/PKs/FKs	# Total Map	Avg/Max # Tgt Tab. per Src Tab.	1:1 Map Ratio (%)	Src/Tgt Column Map Ratio (%)
mjs-mjt	Entertainment	1 / 13 / 0 / 0	1 / 13 / 0 / 0	6	1 / 1	100%	46.15% / 46.15%
msjs-msjt	Entertainment	1 / 14 / 0 / 0	1 / 14 / 0 / 0	8	1 / 1	100%	57.14% / 57.14%
mus-mut	Entertainment	1 / 20 / 0 / 0	1 / 20 / 0 / 0	20	1 / 1	100%	100% / 100%
mvs-mvt	Entertainment	1 / 13 / 0 / 0	1 / 13 / 0 / 0	6	1 / 1	100%	46.15% / 46.15%
imdb-sakila	Entertainment	7 / 39 / 2 / 7	16 / 90 / 12 / 22	22	1.00 / 3	26.67%	56.41% / 42.22%
bank1-bank2	Finance	9 / 27 / 5 / 11	9 / 36 / 4 / 12	11	0.44 / 1	100.00%	81.48% / 63.89%
cms-omop	Healthcare	5 / 96 / 1 / 4	39 / 432 / 12 / 58	157	7.40 / 11	3.70%	60.42% / 25.69%
synthea-omop	Healthcare	12 / 111 / 3 / 19	39 / 432 / 12 / 58	101	1.50 / 3	19.44%	49.55% / 31.02%
cprd_aurum-omop	Healthcare	8 / 76 / 5 / 21	39 / 432 / 12 / 58	42	1.75 / 3	48.00%	60.53% / 21.06%
cprd_gold-omop	Healthcare	9 / 123 / 4 / 21	39 / 432 / 12 / 58	52	2.44 / 4	25.00%	36.59% / 21.30%
mimic_iii-omop	Healthcare	26 / 324 / 6 / 55	39 / 432 / 12 / 58	189	1.54 / 5	14.73%	56.79% / 34.26%

Table 1 compares traditional single-table schema matching datasets, derived from [20], with our new complex datasets. Traditional datasets, designed for schema and instance matching, have fewer tables and columns and lack relational structures such as PK/FK relationships. In contrast, our datasets feature larger, more complex schemas with explicit FK/PK relationships, better reflecting real-world relational database designs. The schemas and ground truths are available [5] as a benchmark to support schema matching research and improve solutions for large-scale data integration.

4 Related Work

There are two primary types of information commonly used in the matching process: (1) schema information and (2) instance information. We first examine how each type is utilized to establish correspondences, then discuss recent approaches that leverage deep learning and LLMs.

Schema-based Approaches. Schema-based methods utilize three types of metadata: linguistic knowledge, constraint knowledge, and structural knowledge. Linguistic knowledge derives meaning from attribute names and descriptions to identify semantic similarities between schema elements [28,10,52,23]. Constraint knowledge uses metadata such as data types, primary keys, foreign keys, and uniqueness constraints to establish correspondences [52,7]. Structural knowledge examines table relationships, including one-to-one and one-to-many relationships, as well as their cardinalities [52,7,27]. These methods assume the availability of rich metadata, but real-world databases often include abbreviations or domain-specific acronyms, which limit their effectiveness. To address this, approaches such as those by [46,34,16] incorporate external resources like dictionaries and thesauri to enhance schema semantics.

Instance-based Approaches. When schema metadata is limited, instance-based methods analyze data values, column semantics, and distribution patterns to identify matches [24,28]. For example, numeric data distributions can reveal semantic relationships that schema-based methods might miss. Machine learning (ML)-based matchers, such as those in [41], leverage data patterns to predict correspondences between schemas. However, these methods require significant amounts of data and often need retraining for new domains, limiting their scalability [11]. Additionally, instance-based approaches are computationally expensive and prone to errors when applied to large datasets. Some solutions combine schema and instance information to improve correspondence accuracy [6,10,52,45].

Deep-learning and LLM-based Approaches. Recent methods leverage deep learning and LLMs to uncover deeper relationships between schema elements. SMAT [50] and SemProp [13] use pre-trained word embeddings to move beyond syntactic similarities [38,26], while EmbDI [8] incorporates instance-level information, and REMA [19] applies graph embeddings to capture structural relationships. LLM-based approaches further advance schema matching. LSM [51] combines LLMs with active learning and real-time user feedback. Our approach differs by operating without human-in-the-loop feedback. Parciak et al. [36] use LLMs as standalone matching tools but limit their scope to single-source and single-target table matching, which is less applicable to multi-table scenarios. Huang et al. [18] propose a schema matching tool involving offline PK/FK identification and sample generation, followed by online SQL transformation and iterative debugging. In contrast, our work avoids query generation and does not rely on strict SQL constraints like DISTINCT for correctness evaluation. Recent multi-table methods, such as ReMatch [43], show strong results with GPT-4 [1]

but struggle with models like GPT-3.5 [32], which often fail to capture deeper relational patterns.

5 LLMATCH Framework

Framework Design. Unifying different schema matching methods presents significant challenges due to the diversity in their design:

- **Task Scope Differences:** Some methods are designed for table-to-table matching, focusing on aligning columns between individual tables. Others address schema-to-schema matching, which involves capturing relationships across multiple interrelated tables. These differences in scope require a flexible framework that accommodates both granular and holistic approaches.
- **Integrating Large Language Models (LLMs):** LLMs are increasingly ubiquitous in schema matching. However, their integration into existing methods raises fairness concerns. Some methods may use LLMs extensively, while others rely minimally or not at all. A fair framework must standardize LLM usage to enable meaningful comparisons across diverse approaches.

To address these challenges, we propose a unified framework with a focus on structured evaluation and fair comparison. The processing framework includes three major steps: *Schema Preparation*, *Table Selection* and *Column Matching*.
Schema Preparation: In this step, we categorize schema information into table and column names (syntactic content), primary and foreign keys (PK/FK) (structural relationships), and table/column descriptions (semantic metadata), enabling our framework to assess their impact on matching quality and identify the most influential features across different contexts.Additionally, we introduce **Rollup**, a schema transformation that merges semantically related columns into a single higher-level representation to simplify matching. As shown in Fig. 3, the *Rollup phase* aggregates columns such as: `target.gender_concept_id`, `target.gender_source_value`, `target.gender_source_concept_id` into a single column, `target.gender`. This reduces schema complexity before the matching process begins.
Table Selection: This step narrows down the set of target tables for each source table T_s, ensuring the next stage focuses only on relevant candidates. We compare four candidate selection strategies:

- **None:** Matches all target tables at once.
- **Nested Join:** Matches each target table separately in distinct prompts.
- **Vector Similarity:** Uses embeddings to select the top-k most similar tables.
- **LLM Selection:** Lets the LLM determine relevant target tables based on schema context.

Column Matching with Drilldown: Once relevant tables are selected, column matching is performed. To enhance alignment, we introduce **Drilldown**, which refines matches by expanding rolled-up columns back into their original

components. As shown in Fig. 3, once `source.gender` is matched to `target.gender`, the *Drilldown phase* decomposes it back into: `target.gender_concept_id`, `target.gender_source_value`, `target.gender_source_concept_id`. This approach ensures a fine-grained alignment.

Our framework provides a systematic and flexible approach to evaluating schema matching methods. By categorizing schema information, standardizing LLM integration, and structuring the matching process into table candidate selection and column matching, we enable fair comparisons and deeper insights into the performance of diverse methods. This structured evaluation advances the understanding and design of LLM-enabled schema matching pipelines.

6 Evaluation

We evaluate the following baselines using our proposed LLMATCH framework.

- Coma [10]: A classical schema matching approach that combines multiple schema-based matchers, representing schemata as rooted graphs.
- SF [25]: Transforms schemas into directed graphs and propagates similarity scores through neighboring nodes.
- Cupid [23]: Represents schemas as hierarchical tree structures and calculates similarity as a weighted sum of linguistic (name-based) and structural (context-based) similarities.
- Unicorn [48]: A the-state-of-the-art deep learning model that learns from multiple datasets and tasks using a mixture-of-experts model.
- ReMatch [43]: An LLM-based method that serializes tables into documents, generates document embeddings, and uses vector similarity to pair the most similar target tables with the source table. The selected tables are then fed into an LLM to produce column mappings.
- LLMATCH (our approach): We leverage LLMs for both table candidate selection and column matching. Similar to ReMatch, we include schema information such as names, primary/foreign keys, and descriptions in the prompt. Details of the two prompt templates and schema serialization format are provided in the extended report [5].

Evaluation Metric. We use the **F1 score** as the primary metric for performance evaluation, as it is widely adopted in schema matching research [36,50,3,10]. While alternative metrics such as recall@k [48] and accuracy@k [43] are also used in related work, we prioritize the F1 score over rank-based metrics because our approach does not require the LLM to produce ranked results. Following the setup in [36], we ask the model to generate a list of mappings, treating each mapping with equal importance. In industrial applications, recall is prioritized over precision because identifying and addressing missing matches typically requires more human effort than correcting incorrect ones [36]. In this context, a high recall rate is generally more valuable than high precision, and the F1 score captures this balance effectively. For F1 score calculations, we treat mappings to foreign keys (FKs) as equivalent to their corresponding primary keys (PKs).

Strategy	Complex Schema Matching Task							Simple Schema Matching Task			
	imdb-sakila	**cms-omop**	**cprd_au-omop**	**cprd_gold-omop**	**mimic_iii-omop**	**synthea-omop**	**bank1-bank2**	**mjs-mjt**	**msjs-msjt**	**mus-mut**	**mvs-mvt**
Coma	0.18	0.01	0.13	0.03	0.04	0.08	0.76	0.91	0.71	0.57	**0.91**
SF	0.21	0.04	0.08	0.03	0.09	0.03	0.50	0.71	0.70	0.89	0.59
Cupid	0.00	0.01	0.00	0.00	0.00	0.00	0.32	0.33	0.80	0.86	0.33
Unicorn	0.00	0.05	0.05	0.01	0.01	0.03	0.38	**0.92**	0.80	0.87	0.80
ReMatch GPT-3.5	0.36	0.07	0.06	0.04	0.07	0.06	0.40	0.86	0.70	0.88	0.67
ReMatch GPT-4o-mini	0.64	0.16	0.20	0.19	0.20	0.19	0.74	**0.92**	0.84	0.81	0.71
LLMATCH GPT-3.5	0.47	0.20	0.15	0.16	0.13	0.26	0.67	**0.92**	0.82	0.95	0.83
LLMATCH GPT-4o-mini	**0.73**	**0.37**	**0.36**	**0.30**	**0.33**	**0.37**	**0.85**	**0.92**	**0.94**	**0.97**	0.83

Table 2: F1 scores of schema matching methods on complex and simple tasks. Traditional methods (Coma, SF, Cupid, and Unicorn) perform well on simple schema matching tasks but experience a significant drop in performance on complex schema matching problems. In contrast, the two LLM-based methods, ReMatch and LLMATCH, achieve top performance across both task types, demonstrating their effectiveness in handling complex schema matching.

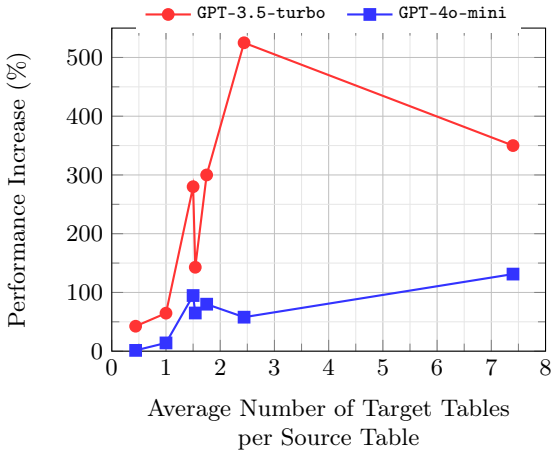
Implementation. We use the `gpt-3.5-turbo` and `gpt-4o-mini` models with their default settings, acknowledging the limited reproducibility of conversations due to the inherent non-determinism of these models [4]. We do not modify any model parameters or fine-tune the models. All embeddings in our experiments are generated using SBERT [40]. Since the specific wording of a prompt influences the output of an LLM [42], we include all prompt templates and schema serialization formats in the Appendix.

Evaluation Goals. The primary goal of our evaluation is to assess the effectiveness of LLMATCH for schema matching, particularly in complex, multi-table relational databases. To achieve this, we investigate several key questions: how different approaches perform on traditional and new benchmarks (Sec. 6.1), the impact of table selection methods on performance (Sec. 6.2), and the role of schema elements such as names, descriptions, and PK/FK relationships (Sec. 6.3). Additionally, we examine how multi-level schema matching with Rollup and Drilldown affects performance (Sec. 6.4), the implications of LLM context limits for large datasets (Sec. 6.5), and the productivity gains in schema matching with and without machine assistance (Sec. 6.6).

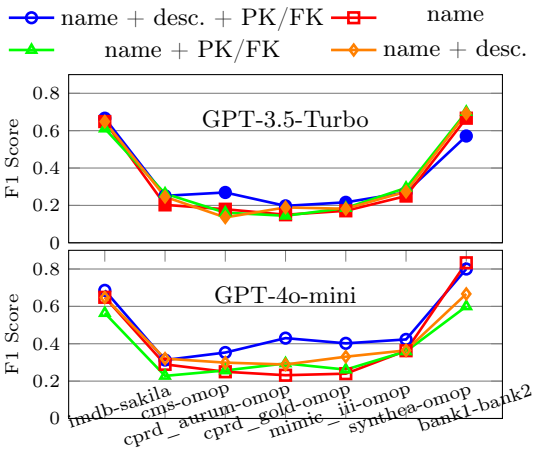
6.1 End-to-End Performance Evaluation

We conducted an extensive evaluation of our proposed method against several baseline strategies on both complex and simple schema matching tasks. The results are summarized in Table 2. For the simple schema matching tasks, both traditional methods and LLM-based methods performed well, achieving high scores on all datasets. However, traditional methods' performance dropped significantly on complex tasks, a phenomenon also reported by LSM [51].

For the complex matching problems, LLMATCH significantly outperforms all baselines. For instance, on the largest dataset, mimic_iii-omop, LLMATCH achieved a score of 0.4, doubling the performance of ReMatch, which scored



(a) Percentage performance increase of LLMATCH over SOTA (ReMatch) across datasets with varying complexity. Higher values indicate higher complexity.



(b) Schema element ablation study.: removing the PK/FK relationship and descriptions reduces the F1 score in the matching task, particularly for the more capable GPT-4o-mini model.

0.2. Traditional methods like Coma and SF performed considerably worse, with scores of 0.04 and 0.09, respectively. An exception was observed with the bank1-bank2 dataset, where traditional methods showed comparable performance. A detailed inspection revealed that the two schemas share a high degree of common vocabulary, making the task easier for lexical matching approaches.

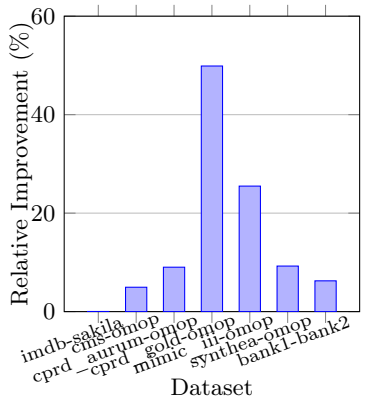
Our method also consistently outperforms the state-of-the-art LLM matcher ReMatch across both GPT-3.5-turbo and GPT-4o-mini models. The performance gain becomes more substantial as the complexity of the matching task increases. Fig. 4a shows the percentage improvement relative to the average number of target tables matched with each source table. For simpler tasks, the difference in performance is minimal. However, as the number of target tables approaches 2, the improvement from our method becomes more pronounced. Notably, the performance gap is even wider for GPT-3.5-turbo, indicating that our approach is effective even with less capable models where other methods struggle.

6.2 Table Selection Strategy Study

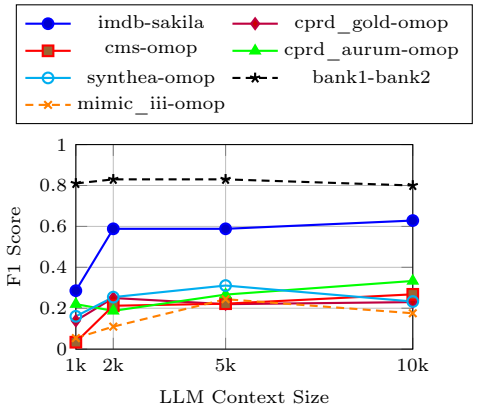
In this section, we evaluate the impact of various table selection strategies (Step 2 in Figure 2) on schema matching performance. We compare four distinct table selection methods while keeping all other settings constant. For each source table, target table candidates are selected using the following strategies. Firstly, in the *None* strategy, no specific table selection is applied; all target tables are considered relevant, so the output size matches the number of tables in the target schema. Secondly, the *nested_join* strategy matches each source table with a single target table in separate prompts during column matching. Thirdly, the *vector_similarity* strategy, a commonly used filtering method, selects the top-k target tables for each source table based on vector similarity scores, with

Table Selection Strategy	imdb-sakila		cms-omop		cprd_aurum-omop		cprd_gold-omop		mimic_iii-omop		synthea-omop		bank1-bank2	
	Score	Count	Score	Count	Score	Count	Score	Count	Score	Count	Score	Count	Score	Count
None	0.39	(16.0)	0.21	(39.0)	0.28	(39.0)	0.22	(39.0)	0.13	(39.0)	0.23	(39.0)	0.74	(9.0)
nested_join	0.32	(1.0)	0.10	(1.0)	0.20	(1.0)	0.18	(1.0)	0.07	(1.0)	0.10	(1.0)	0.64	(1.0)
vector_similarity (top5)	0.67	(5.0)	0.35	(5.0)	0.19	(5.0)	0.27	(5.0)	0.16	(5.0)	0.29	(5.0)	0.85	(5.0)
vector_similarity (top10)	0.67	(10.0)	0.36	(10.0)	0.33	(10.0)	**0.37**	(10.0)	0.25	(10.0)	0.29	(10.0)	**0.88**	(9.0)
vector_similarity (top15)	0.59	(15.0)	0.31	(15.0)	0.27	(15.0)	0.29	(15.0)	0.26	(15.0)	0.32	(15.0)	**0.88**	(9.0)
llm (gpt-4o-mini)	**0.73**	(4.0)	**0.37**	(5.4)	**0.36**	(6.3)	0.30	(6.9)	**0.33**	(6.9)	**0.37**	(4.7)	0.85	(4.6)

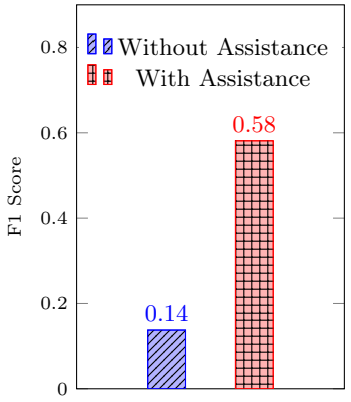
Table 3: Table Selection Strategy Study: F1 scores for various table selection methods, with the number of selected tables in parentheses. LLM-based methods select fewer tables but achieve higher quality. (Column matching uses LLM gpt-4o-mini).



(a) Effect of Rollup/Drilldown on Improving Matching Performance: The improvement is more significant for complex datasets.



(b) Scalability study: when the context window is too small to capture the entire schema, performance drops. Once the window size is sufficient, further increases have little effect on F1 score.



(c) Productivity Gain Study: machine assistance significantly boosts schema matching performance, with equal time per task by human.

Fig. 5: Comparison of Different Factors on Schema Matching Performance

k set to 5, 10, and 15 in our experiments. Lastly, the LLM strategy provides descriptions of the source and target tables to the LLM, which then identifies relevant tables, producing a variable number of outputs.

The experimental results are presented in Table 3. For each dataset, the first column displays the final F1 score of the column matching results, and the second column shows the average number of target tables selected for each source table. From the results, it is evident that the quality of table selection impacts the final matching performance. The LLM-based strategies, with smaller output size, out-perform vector similarity methods.

This insight is significant because vector similarity methods have been the popular choice in retrieval-augmented generation (RAG) frameworks. Our findings suggest that LLMs possess a deeper understanding of the semantic relationships between tables, leading to better schema matching performance even with fewer tables selected.

6.3 Schema Elements Ablation Study

To assess the impact of schema elements—*name*, *description*, and *relationship*—on matching performance, we conducted an ablation study. Figure 4b reports F1 scores for GPT-3.5-Turbo and GPT-4o-mini across different schema element combinations. The results show that incorporating *description* and *relationship* improves performance, with the *name* + *description* + *relationship* combination yielding the highest F1 scores. The additional context helps disambiguate schema elements with similar names but different roles. GPT-4o-mini consistently outperforms GPT-3.5-Turbo, particularly when enriched with all schema elements, indicating its superior ability to capture complex semantic relationships. However, the performance gap varies across datasets, suggesting that dataset characteristics influence how effectively each model leverages schema information.

6.4 Effect of Rollup and Drilldown

The results, as shown in Fig. 5a, demonstrate the impact of Rollup and Drilldown on schema matching performance across multiple datasets. The relative improvement varies significantly depending on the dataset's complexity and structure. The cprd_gold-omop dataset exhibits the highest improvement, suggesting that Rollup effectively consolidates fragmented schema elements, allowing the LLM to establish stronger high-level correspondences before Drilldown refines the column-level mappings. Similarly, mimic_iii-omop and synthea-omop show notable gains, indicating that multi-level schema matching is particularly beneficial for datasets with many-to-many schema relationships. However, datasets such as imdb-sakila and bank1-bank2 show relatively minor improvements, likely due to their simpler schema structures where traditional column-level matching is already effective. This suggests that Rollup and Drilldown offer the greatest benefits when schema fragmentation is high, reinforcing the importance of hierarchical schema alignment in complex integration scenarios. These findings highlight the strengths of multi-level schema matching, demonstrating that schema abstraction (Rollup) and fine-grained alignment (Drilldown) significantly improve performance on complex datasets while providing only marginal benefits in simpler cases.

6.5 Scalability Study

In this experiment, we limit input context size while keeping output unrestricted due to technical constraints. To standardize evaluation across LLMs, we approximate context size using word count. Tasks exceeding a single-prompt limit are split into multiple prompts with small contexts, ensuring the largest source and target tables fit together. This constraint applies to both table selection and column matching. As shown in Fig. 5b, overly small contexts fragment the task, reducing performance. This underscores the need for LLMs with sufficient context capacity to handle large schemas effectively.

6.6 Productivity Gain Study

To assess the impact of LLM-generated matching results on manual schema matching tasks, we conducted an experiment with two scenarios: one without machine-generated results and one with them. Each scenario was limited to two minutes. Four data engineers participated, and the results, shown in Fig.5c, indicate that machine assistance significantly enhances performance, even when LLM-generated matches are not entirely accurate. These findings align with prior research[51], which demonstrated that LLM assistance improves accuracy and reduces labeling costs by up to 81% compared to fully manual efforts.

7 Conclusion

This paper presents a framework that decomposes schema matching into three stages: schema preparation, table selection, and column matching. A key feature of our approach is the introduction of Rollup and Drilldown, a multi-level schema matching mechanism that simplifies complex schemas before matching and refines alignments afterward. This approach improves performance by first abstracting semantically related attributes and later restoring granularity for precise column-level alignment. Through this framework, we also found that vector similarity, a widely used method, is not the most effective choice for table selection, offering valuable insights for other LLM-based applications. Additionally, we developed and released a comprehensive multi-industry complex schema matching benchmark to support future research in this field.

References

1. Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al.: Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023)
2. Ackerman, R., Gal, A., Sagi, T., Shraga, R.: A cognitive model of human bias in matching. In: PRICAI 2019: Trends in Artificial Intelligence: 16th Pacific Rim International Conference on Artificial Intelligence, Cuvu, Yanuca Island, Fiji, August 26–30, 2019, Proceedings, Part I 16. pp. 632–646. Springer (2019)
3. Alwan, A.A., Nordin, A., Alzeber, M., Abualkishik, A.Z.: A survey of schema matching research using database schemas and instances. International Journal of Advanced Computer Science and Applications **8**(10) (2017)
4. Anadkat, S.: How to make your completions outputs consistent with the new seed parameter (2023), https://cookbook.openai.com/examples/reproducible_outputs_with_the_seed_parameter, openAI Cookbook
5. Anonymous Authors: Llmatch code and dataset. https://anonymous.4open.science/r/LLMatch-AC0E/README.md (2024), accessed: 2024-11-12
6. Asif-Ur-Rahman, M., Hossain, B.A., Bewong, M., Islam, M.Z., Zhao, Y., Groves, J., Judith, R.: A semi-automated hybrid schema matching framework for vegetation data integration. Expert Systems with Applications **229**, 120405 (2023)
7. Bernstein, P.A., Madhavan, J., Rahm, E.: Generic schema matching, ten years later. Proceedings of the VLDB Endowment **4**(11), 695–701 (2011)

8. Cappuzzo, R., Papotti, P., Thirumuruganathan, S.: Embdi: generating embeddings for relational data integration. In: 29th Italian Symposium on Advanced Database Systems (SEDB), Pizzo Calabro, Italy (2021)
9. Development, J.R..: Etl lambdabuilder documentation (2024), https://ohdsi.github.io/ETL-LambdaBuilder/, accessed: 2024-10-09
10. Do, H.H., Rahm, E.: Coma—a system for flexible combination of schema matching approaches. In: VLDB'02: Proceedings of the 28th International Conference on Very Large Databases. pp. 610–621. Elsevier (2002)
11. Feng, J., Hong, X., Qu, Y.: An instance-based schema matching method with attributes ranking and classification. In: 2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery. vol. 5, pp. 522–526. IEEE (2009)
12. Fernandez, R.C., Elmore, A.J., Franklin, M.J., Krishnan, S., Tan, C.: How large language models will disrupt data management. Proc. VLDB Endow. **16**(11), 3302–3309 (Jul 2023). https://doi.org/10.14778/3611479.3611527, https://doi.org/10.14778/3611479.3611527
13. Fernandez, R.C., Mansour, E., Qahtan, A.A., Elmagarmid, A., Ilyas, I., Madden, S., Ouzzani, M., Stonebraker, M., Tang, N.: Seeping semantics: Linking datasets using word embeddings for data discovery. In: 2018 IEEE 34th International Conference on Data Engineering (ICDE). pp. 989–1000. IEEE (2018)
14. Financial Times: Deutsche Bank struggles with fallout after huge Postbank IT migration: Regulator makes unprecedented rebuke as many clients locked out of their accounts for weeks. https://www.ft.com/content/4138876c-5a10-46d4-b6c6-7d421cbd9df0 (2024), [Accessed: 12-Nov-2024]
15. Frid, S., Pastor Duran, X., Bracons Cucó, G., Pedrera-Jiménez, M., Serrano-Balazote, P., Muñoz Carrero, A., Lozano-Rubí, R.: An ontology-based approach for consolidating patient data standardized with european norm/international organization for standardization 13606 (en/iso 13606) into joint observational medical outcomes partnership (omop) repositories: Description of a methodology. JMIR Med Inform **11**, e44547 (Mar 2023). https://doi.org/10.2196/44547, https://medinform.jmir.org/2023/1/e44547
16. Giunchiglia, F., Shvaiko, P., Yatskevich, M.: S-match: an algorithm and an implementation of semantic matching. In: The Semantic Web: Research and Applications: First European Semantic Web Symposium, ESWS 2004 Heraklion, Crete, Greece, May 10-12, 2004. Proceedings 1. pp. 61–75. Springer (2004)
17. Haberson, A., Rinner, C., Schoberl, A., Gall, W.: Feasibility of mapping austrian health claims data to the omop common data model. Journal of Medical Systems **43**(10), 314 (2019). https://doi.org/10.1007/s10916-019-1436-9, https://doi.org/10.1007/s10916-019-1436-9
18. Huang, Z., Guo, J., Wu, E.: Transform table to database using large language models. Proceedings of the VLDB Endowment. ISSN **2150**, 8097 (2024)
19. Koutras, C., Fragkoulis, M., Katsifodimos, A., Lofi, C.: Rema: Graph embeddings-based relational schema matching. In: EDBT/ICDT Workshops (2020)
20. Koutras, C., Siachamis, G., Ionescu, A., Psarakis, K., Brons, J., Fragkoulis, M., Lofi, C., Bonifati, A., Katsifodimos, A.: Valentine: Evaluating matching techniques for dataset discovery. In: 2021 IEEE 37th International Conference on Data Engineering (ICDE). pp. 468–479. IEEE (2021)
21. Lamer, A., Abou-Arab, O., Bourgeois, A., Parrot, A., Popoff, B., Beuscart, J.B., Tavernier, B., Moussa, M.D.: Transforming anesthesia data into the observational medical outcomes partnership common data model: Development and usability study. J Med Internet Res **23**(10), e29259 (Oct 2021). https://doi.org/10.2196/29259, https://www.jmir.org/2021/10/e29259

22. Lamer, A., Depas, N., Doutreligne, M., Parrot, A., Verloop, D., Defebvre, M.M., Ficheur, G., Chazard, E., Beuscart, J.B.: Transforming french electronic health records into the observational medical outcome partnership's common data model: a feasibility study. Applied clinical informatics **11**(01), 013–022 (2020)
23. Madhavan, J., Bernstein, P.A., Rahm, E.: Generic schema matching with cupid. In: vldb. vol. 1, pp. 49–58 (2001)
24. Mehdi, O., Ibrahim, H., Affendey, L.: An approach for instance based schema matching with google similarity and regular expression. International Arab Journal of Information Technology (IAJIT) **14**(5) (2017)
25. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In: Proceedings 18th international conference on data engineering. pp. 117–128. IEEE (2002)
26. Mikolov, T.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
27. Mukherjee, D., Bandyopadhyay, A., Chowdhury, R., Bhattacharya, I.: Learning knowledge graph for target-driven schema matching. In: Proceedings of the 3rd ACM India Joint International Conference on Data Science & Management of Data (8th ACM IKDD CODS & 26th COMAD). pp. 65–73 (2021)
28. Munir, S., Khan, F., Riaz, M.A.: An instance based schema matching between opaque database schemas. In: 2014 4th International Conference on Engineering Technology and Technopreneuship (ICE2T). pp. 177–182. IEEE (2014)
29. OHDSI: The Book of OHDSI: Observational Health Data Sciences and Informatics. OHDSI (2019)
30. OHDSI: Omop common data model - version 5.4. https://ohdsi.github.io/CommonDataModel/cdm54.html (2024), accessed: 2024-10-09
31. Oja, M., Tamm, S., Mooses, K., Pajusalu, M., Talvik, H.A., Ott, A., Laht, M., Malk, M., Lõo, M., Holm, J., Haug, M., Šuvalov, H., Särg, D., Vilo, J., Laur, S., Kolde, R., Reisberg, S.: Transforming Estonian health data to the Observational Medical Outcomes Partnership (OMOP) Common Data Model: lessons learned. JAMIA Open **6**(4), ooad100 (12 2023). https://doi.org/10.1093/jamiaopen/ooad100, https://doi.org/10.1093/jamiaopen/ooad100
32. OpenAI: Gpt-3.5 turbo fine-tuning and api updates. https://openai.com/index/gpt-3-5-turbo-fine-tuning-and-api-updates/ (August 22 2023), retrieved October 12, 2024
33. Overhage, J.M., Ryan, P.B., Reich, C.G., Hartzema, A.G., Stang, P.E.: Validation of a common data model for active safety surveillance research. Journal of the American Medical Informatics Association **19**(1), 54–60 (2012)
34. Palopoli, L., Terracina, G., Ursino, D., et al.: The system dike: Towards the semi-automatic synthesis of cooperative information systems and data warehouses. In: ADBIS-DASFAA Symposium. pp. 108–117 (2000)
35. Pan, Z., Yang, M., Monti, A.: Schema matching based on energy domain pre-trained language model. Energy Informatics **6**(Suppl 1), 22 (2023)
36. Parciak, M., Vandevoort, B., Neven, F., Peeters, L.M., Vansummeren, S.: Schema matching with large language models: An experimental study. Joint Workshops at the 50th International Conference on Very Large Data Bases (VLDBW'24) — TaDA'24: 2nd International Workshop on Tabular Data Analysis (August 26–30 2024)
37. Paris, N., Lamer, A., Parrot, A.: Transformation and evaluation of the mimic database in the omop common data model: development and usability study. JMIR Medical Informatics **9**(12), e30970 (2021)

38. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). pp. 1532–1543 (2014)
39. Reich, C., Ostropolets, A., Ryan, P., Rijnbeek, P., Schuemie, M., Davydov, A., Dymshyts, D., Hripcsak, G.: Ohdsi standardized vocabularies—a large-scale centralized reference ontology for international data harmonization. Journal of the American Medical Informatics Association **31**(3), 583–590 (2024)
40. Reimers, N., Gurevych, I.: Sentence-bert: Sentence embeddings using siamese bert-networks. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics (11 2019), https://arxiv.org/abs/1908.10084
41. Rong, S., Niu, X., Xiang, E.W., Wang, H., Yang, Q., Yu, Y.: A machine learning approach for instance matching based on similarity metrics. In: The Semantic Web–ISWC 2012: 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I 11. pp. 460–475. Springer (2012)
42. Schulhoff, S., Ilie, M., Balepur, N., Kahadze, K., Liu, A., Si, C., Li, Y., Gupta, A., Han, H., Schulhoff, S., et al.: The prompt report: A systematic survey of prompting techniques. arXiv preprint arXiv:2406.06608 (2024)
43. Sheetrit, E., Brief, M., Mishaeli, M., Elisha, O.: Rematch: Retrieval enhanced schema matching with llms. arXiv preprint arXiv:2403.01567 (2024)
44. Shraga, R., Amir, O., Gal, A.: Learning to characterize matching experts. In: 2021 IEEE 37th International Conference on Data Engineering (ICDE). pp. 1236–1247. IEEE (2021)
45. Shraga, R., Gal, A., Roitman, H.: Adnev: Cross-domain schema matching using deep similarity matrix adjustment and evaluation. Proceedings of the VLDB Endowment **13**(9), 1401–1415 (2020)
46. Sorrentino, S., Bergamaschi, S., Gawinecki, M., Po, L.: Schema label normalization for improving schema matching. Data & Knowledge Engineering **69**(12), 1254–1273 (2010)
47. Stang, P.E., Ryan, P.B., Racoosin, J.A., Overhage, J.M., Hartzema, A.G., Reich, C., Welebob, E., Scarnecchia, T., Woodcock, J.: Advancing the science for active surveillance: rationale and design for the observational medical outcomes partnership. Annals of internal medicine **153**(9), 600–606 (2010)
48. Tu, J., Fan, J., Tang, N., Wang, P., Li, G., Du, X., Jia, X., Gao, S.: Unicorn: A unified multi-tasking model for supporting matching tasks in data integration. Proceedings of the ACM on Management of Data **1**(1), 1–26 (2023)
49. Wornow, D., Suh, G., Elmore, A.J., Krishnan, S., Parameswaran, A.: Automating the enterprise with foundation models. Proceedings of the VLDB Endowment **17**(12), 2805–2808 (2024)
50. Zhang, J., Shin, B., Choi, J.D., Ho, J.C.: Smat: An attention-based deep learning solution to the automation of schema matching. In: Advances in Databases and Information Systems: 25th European Conference, ADBIS 2021, Tartu, Estonia, August 24–26, 2021, Proceedings 25. pp. 260–274. Springer (2021)
51. Zhang, Y., Floratou, A., Cahoon, J., Krishnan, S., Müller, A.C., Banda, D., Psallidas, F., Patel, J.M.: Schema matching using pre-trained language models. In: 2023 IEEE 39th International Conference on Data Engineering (ICDE). pp. 1558–1571. IEEE (2023)
52. Zhao, H., Ram, S.: Combining schema and instance information for integrating heterogeneous data sources. Data & Knowledge Engineering **61**(2), 281–303 (2007)

A Appendix

Listing 1.1: Table Selection Prompt

```
You are an expert in matching database schemas. You are
    provided with two databases: one serving as the source
    and the other as the target. Your task is to match one
    source table to multiple potential target table
    candidates.

**Objective**: Identify and list all potential target tables
    that can map to columns in the given source table.

**Source Table Details**:
{{source_table}}

**Target Tables Details**:
{{target_tables}}

**Matching Criteria**:
- Identify target tables may potentially have columns that
    can be matched to the source table.
- One source column might be matched to multiple target
    columns. Redundant matches are allowed.

**Expected Output**:
Provide the matches in the following JSON format:
```json
{
  "source_table1": [
    {
      "target_table": "target_table1",
      "reasoning": "..."
    },
    {
      "target_table": "target_table2",
      "reasoning": "..."
    }
  ]
}
```
Return only the JSON object and no other text.
```

Listing 1.2: Column Matching Prompt

```
You are an expert in databases. Your task is to create
    matches between columns in two datasets: "Source␣Columns"
     and "Target␣Columns". One source column can be matched
    to multiple target columns. The matches should be based
    on the semantic similarity of the entities described by
    the columns, considering the context provided in their
    descriptions.
```

```
2
3  **Matching Criteria:**
4
5          -       Entity Similarity: The matched entries should
                    describe the same or very similar entities. The
                   source entry can be part of the target entry and
                   vice versa. e.g. full_name => first_name,
                   last_name. registration_date => registration_date
                   , registration_time
6          -       Contextual Alignment: each column represent
                   different types of entities. make sure that the
                   matched columns are of the same type. Negative
                   examples: bank.name != bank_branch.name, customer
                   .name != staff.name
7          -       Data Type Compatibility: Ensure that the data
                    types of the matched columns are compatible. A
                   single element can be matched with multiple
                   elements and vice versa. e.g. source_table.
                   language => target_table.languages.
8
9  **Instructions:**
10
11         1.      Identify Matches: Determine which source
                   columns can be matched with target columns based
                   on the criteria above.
12         2.      Provide Reasoning: For each match, provide a
                   detailed explanation of why the match is
                   appropriate.
13     3.  Review Matches: Ensure data belongs to the same
           domain and is semantically similar.
14
15 **Output Format:**
16
17         -       Provide the matches in the following JSON
                   format:
18 ```json
19 {
20     "source_table1.source_column1": [
21         {
22             "mapping": "target_table1.target_column1",
23             "reasoning": "explanation␣of␣the␣match"
24         },
25         {
26             "mapping": "target_table2.target_column2",
27             "reasoning": "explanation␣of␣the␣match"
28         }
29     ],
30     "source_table2.source_column2": [
31         {
32             "mapping": "None",
```

```
            "reasoning": "..."
        }
    ]
}
```
Source Tables:
{{source_columns}}

Target Tables:
{{target_columns}}

Return only the JSON object and no other text.

Listing 1.3: Example of target table serialization during table selection. It demonstrates a concise schema representation optimized to fit within a single prompt.

```
{
  "account": {
    "description": "Represents bank accounts, specifying type, balance, and associated branch.",
    "foreign_keys": "branch_id=>branch.branch_id",
    "non_foreign_key_columns": "account_balance,account_id,account_type"
  },
  "branch": {
    "description": "Represents a branch of the bank, containing basic information about each branch.",
    "foreign_keys": "",
    "non_foreign_key_columns": "assets,branch_address,branch_id,branch_name"
  },
  ...
}
```

Listing 1.4: Table Serialization Format. This is the full schema representation. It is used to serialize the source table during table selection and column matching. The same format is applied to the target table during column matching.

```
{
  "account": {
    "table": "account",
    "columns": {
      "acc_type": {
        "name": "acc_type",
        "description": "Type of account (e.g., savings, checking)"
      },
      "account_no": {
        "name": "account_no",
        "description": "Unique identifier for each account",
```

```
        "is_primary_key": true,
        "foreign_keys": [
          "hold_by.account_no",
          "maintain.account_no"
        ]
      },
      "balance": {
        "name": "balance",
        "description": "Current␣balance␣in␣the␣account"
      },
      "branch_id": {
        "name": "branch_id",
        "description": "Foreign␣key␣linking␣the␣account␣to␣
            the␣branch␣where␣it␣is␣maintained",
        "is_foreign_key": true,
        "linked_entry": "branch.branch_id"
      }
    },
    "table_description": "Account␣stores␣information␣about␣
        accounts␣held␣by␣customers␣at␣the␣bank.';"
  },
  "branch": {
    "table": "branch",
    "columns": {
      "address": {
        "name": "address",
        "description": "Address␣of␣the␣branch"
      },
      "bank_code": {
        "name": "bank_code",
        "description": "Foreign␣key␣linking␣the␣branch␣to␣its
            ␣bank",
        "is_foreign_key": true,
        "linked_entry": "bank.code"
      },
      "branch_id": {
        "name": "branch_id",
        "description": "Unique␣identifier␣for␣each␣branch",
        "is_primary_key": true,
        "foreign_keys": [
          "account.branch_id",
          "loan.branch_id",
          "maintain.branch_id",
          "offer.branch_id"
        ]
      },
      "name": {
        "name": "name",
        "description": "Name␣of␣the␣branch"
      }
```

```
    },
    "table_description": "Branch␣stores␣details␣about␣each␣
        branch␣of␣the␣bank.';"
  },
  ...
}
```

Listing 1.5: Rollup (column merge) Prompt

```
Task: Database Schema Pre-Processing for Data Migration

Objective:
You will be provided with a database schema in JSON format.
    Your goal is to identify columns within the same table
    that hold related or similar data (e.g., first_name and
    last_name, start_date and start_datetime, end_date and
    end_datetime). Then, merge these related columns into a
    single column in your output.

Steps:
        1.      Read the Input JSON Schema
        - Understand the structure of each table.
        - Look for columns in the same table that are similar
            or can be combined logically.
        2.      Identify Mergable Columns
        - If two or more columns represent "parts␣of␣a␣whole"
            (like first and last name, or address1 and
           address2), these can be merged.
        - If the columns are not logically related, do not
           merge them.
    - If two columns represent the same data but are named
       differently, they can be merged. e.g. table.address,
       table.address_id
        3.      Create a Merged Output
        - For each table with identified merges, create an
           entry in a JSON array named "merged_columns".
        - Inside each entry, show:
        - table_name: The name of the table.
        - merged_columns: An array of merged column
           definitions, each of which must include:
        1.      column_name: The new merged name.
        2.      column_description: A short explanation of
           why or how these columns are merged.
        3.      original_columns: A list of the fully
           qualified column references (e.g., "customer.
           first_name", "customer.last_name").
        4.      Output Format
        - Your final output must be valid JSON.
        - Use the following structure (with any necessary
           merges included):
```

```
{
  "merged_columns": [
    {
      "table_name": "<TABLE_NAME>",
      "merged_columns": [
        {
          "column_name": "<MERGED_COLUMN_NAME>",
          "column_description": "<WHY_MERGED>",
          "original_columns": [
            "<table.columnA>",
            "<table.columnB>"
          ]
        }
      ]
    }
  ]
}



Important:
        - Only list tables that actually have merged columns.
    - If a table has no columns that need merging, skip it.
        - Do not delete or rename any columns unless they are
            part of a merged set.

Sample Input

(The JSON schema from your database might be large, so here
    is a short version showing two tables as an example.)

{
  "actor": {
    "table": "actor",
    "columns": {
      "actor_id": {
        "name": "actor_id",
        "description": "Primary key"
      },
      "first_name": {
        "name": "first_name",
        "description": "Actor first name"
      },
      "last_name": {
        "name": "last_name",
        "description": "Actor last name"
      }
    }
  },
```

```
  "customer": {
    "table": "customer",
    "columns": {
      "customer_id": {
        "name": "customer_id",
        "description": "Primary key"
      },
      "first_name": {
        "name": "first_name",
        "description": "Customer first name"
      },
      "last_name": {
        "name": "last_name",
        "description": "Customer last name"
      }
    }
  }
}

Sample Output

Here is the kind of JSON output we expect after identifying
    mergable columns. Notice how first_name and last_name
    have been combined into one column called name. Adjust
    the language/description to fit your needs:

{
  "tables": [
    {
      "table_name": "actor",
      "merged_columns": [
        {
          "column_name": "name",
          "column_description": "Actor's full name (merged
              from first_name and last_name)",
          "original_columns": [
            "actor.first_name",
            "actor.last_name"
          ]
        }
      ]
    },
    {
      "table_name": "customer",
      "merged_columns": [
        {
          "column_name": "name",
          "column_description": "Customer's full name (merged
              from first_name and last_name)",
          "original_columns": [
```

```
                "customer.first_name",
                "customer.last_name"
            ]
          }
        ]
      }
    ]
}

Input:
```