LLMATCH: a Unified Schema Matching Framework with Large Language Models

No Author Given

No Institute Given

Abstract. Schema matching is a foundational task in enterprise data integration, aiming to align disparate data sources. While traditional methods handle simple one-to-one table mappings, they often struggle with complex multi-table schema matching in real-world applications. We present LLMatch, a unified and modular schema matching framework. LLMatch decomposes schema matching into three distinct stages: schema preparation, table-candidate selection, and column-level alignment, enabling component-level evaluation and future-proof compatibility. It includes a novel two-stage optimization strategy: a *Rollup* module that consolidates semantically related columns into higher-order concepts, followed by a *Drilldown* module that re-expands these concepts for fine-grained column mapping. To address the scarcity of complex semantic matching benchmarks, we introduce SchemaNet, a benchmark derived from real-world schema pairs across three enterprise domains, designed to capture the challenges of multi-table schema alignment in practical settings. Experiments demonstrate that LLMatch significantly improves matching accuracy in complex schema matching settings and substantially boosts engineer productivity in real-world data integration.

Keywords: Schema Matching · LLM · Data Management

1 Introduction

Schema matching is a database system task that identifies semantic correspondences between columns of source and target schemas and transforms them into a unified format. Industries such as healthcare [30], finance [17], energy [27], and agriculture [6] are increasingly adopting standardized data formats to improve interoperability. In healthcare, unified data models such as OMOP [38] have demonstrated benefits for cross-institutional research and treatment discovery, driving efforts to migrate existing schemas to standardized targets [26,24]. However, flawed data migration can have serious consequences, as illustrated by Deutsche Bank's 2023 account lockouts due to integration failures with Postbank [17,14]. Schema matching in real-world scenarios is complex; while research has primarily addressed basic table-to-table alignment, industrial applications often involve mapping multiple source and target tables, requiring a comprehensive understanding of schema semantics, inter-table relationships, and structural dependencies such as primary–foreign keys and hierarchies.

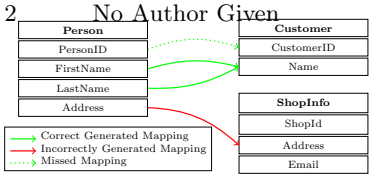

Fig. 1: Toy matching task using traditional methods

Before the advent of large language models (LLMs), complex multi-table schema matching was largely intractable [42]. Traditional tools struggled with challenges such as inconsistent naming conventions, data heterogeneity, and the lack of standardized structural representations. As shown in Fig. 1, traditional embedding-based NLP methods are unable to distinguish minute differences among attributes such as *Person.address* and *ShopInfo.address*, leading to incorrect mappings. Resolving such mappings often demands extensive, error-prone manual effort by domain experts, with a single schema alignment taking up to 500 hours for two experts. [9,29,2,35].

The emergence of LLMs has enabled a new class of schema matching methods with strong performance across diverse tasks [40,12,34,16]. However, two key limitations persist: (1) prevailing approaches treat schema matching as a monolithic process and evaluate it end-to-end, obscuring the contribution of individual components; (2) they largely focus on pairwise table matching, neglecting the more complex many-to-many mappings common in real-world applications.

In this paper, we propose a unified, modular framework for schema matching that decomposes the process into three stages: *schema preparation*, *table selection*, and *column matching* (Fig. 2). *Schema preparation* organizes the information of source and target schemas into standard categories including names, relationships and metadata. *Table selection* narrows the candidate target tables for each source table, reducing the context length for downstream processing. Finally, *column matching* aligns columns between each source table and its selected target tables to produce the final mappings. This modularization promotes compatibility with existing and future methods while enabling fine-grained analysis of component-level contributions to overall performance.

To improve multi-table schema matching, we introduce an optimization strategy based on *Rollup* and *Drilldown* techniques, applied on *schema preparation* and *column matching*, respectively (Fig. 3). *Rollup* aggregates semantically related columns into a higher-level abstraction. For instance, time-related fields like `updated_at` and `created_at` are abstracted as `timestamp`. The abstracted schemas are then used to perform coarse-grained alignment. Upon identifying a high-level match, the *Drilldown* phase reverses the abstraction to perform fine-grained column alignment. This hierarchical approach greatly enhances performance on large and semantically rich schemas while reducing computational overhead and improving scalability.

Recognizing the lack of large-scale, semantically rich benchmarks for schema matching, we curated SchemaNet, a comprehensive benchmark comprising seven datasets across the finance, healthcare, and entertainment domains. Developed in collaboration with two leading financial institutions, SchemaNet captures real-world data integration challenges and encompasses a wide range of schema matching complexities. On average, each dataset includes 14 tables and 135 columns, substantially exceeding the scope of existing benchmarks, which

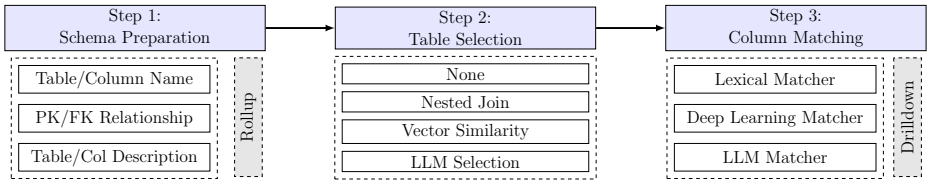

Fig. 2: Architecture diagram of LLMatch. *Schema preparation* organises information into three key categories for downstream processing. *Table selection* maps target tables to source tables. *Column matching* performs column-level mapping.

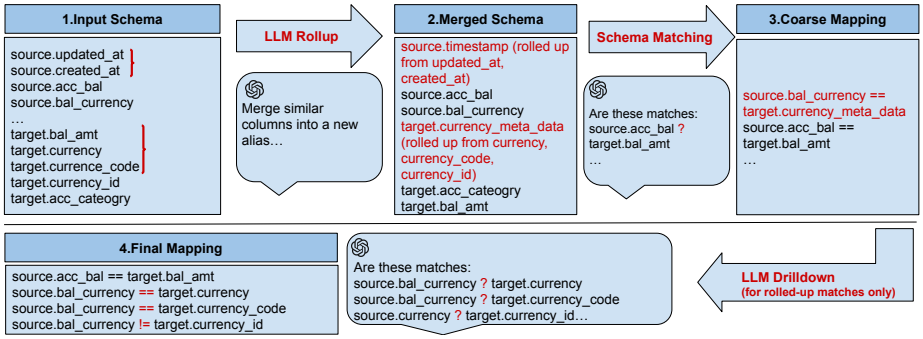

Fig. 3: Example of multi-level matching with Rollup and Drilldown

typically feature a single table with around 15 columns (Table 1). SchemaNet offers a more realistic and diverse foundation for evaluating schema matching methods in practical, multi-table settings.

In summary, this work makes the following key contributions:

- We present LLMatch, a unified and modular schema matching framework that supports existing and future methods while enabling fine-grained evaluation of component contributions.
- We develop an optimization strategy leveraging *Rollup* and *Drilldown* techniques to address the challenges of complex multi-table schema matching.
- We introduce SchemaNet, a multi-domain benchmark co-developed with leading financial institutions, designed to capture the complexity of real-world multi-table schema alignment.
- We conduct extensive experiments demonstrating state-of-the-art performance on 5 benchmarks including SchemaNet, with substantial gains on large, semantically rich datasets.
- Both LLMatch and SchemaNet are made public to facilitate further research in schema matching [5].

2 Related Work

Schema matching methods generally rely on two types of information: (1) schema metadata and (2) instance data. We briefly review both approaches, followed by recent developments using deep learning and large language models (LLMs).

Schema-based Approaches. These methods leverage metadata such as attribute names, data types, and table relationships. Linguistic features help identify semantic similarities [10,20,23], while constraints like keys and uniqueness provide structural cues [43,7]. More advanced techniques incorporate external resources (e.g., dictionaries, ontologies) to resolve abbreviations and domain-specific terms [37,15]. However, their effectiveness diminishes when metadata is sparse or inconsistent.

Instance-based Approaches. When metadata is insufficient, instance-based methods analyze data values and distributions to infer correspondences [23,21]. ML-based matchers learn from patterns in instance data [32], but often require large training sets and retraining for new domains [11]. These methods also struggle with scalability and may underperform on large, noisy datasets. Hybrid approaches combine schema and instance information to improve accuracy [6,36].

Deep Learning and LLM-based Approaches. Recent methods leverage embeddings and pre-trained models to capture deeper semantic and structural patterns. SMAT [41], SemProp [13], and EmbDI [8] use word and instance embeddings, while REMA [18] applies graph embeddings. LLM-based approaches go further: LSM [42] integrates active learning; Parciak et al. [28] use LLMs for single-table matching; and Huang et al. [16] combine offline PK/FK inference with iterative SQL-based refinement. Our work differs by avoiding query generation and human-in-the-loop feedback, and by supporting complex multi-table matching. ReMatch [34] shows strong results with GPT-4 [1], but performance degrades with smaller models like GPT-3.5 [25], which often fails to capture relational structures.

3 Problem Definition

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, ensures 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 [28].

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

4 LLMatch Framework

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

- **Task Scope Differences:** Some methods target table-to-table matching, aligning columns between isolated tables, while others tackle schema-to-schema matching, which captures relationships across multiple interrelated tables. These differing scopes highlight the need for a flexible framework that supports both fine-grained and holistic matching strategies.
- **Integrating Large Language Models (LLMs):** LLMs are becoming integral to schema matching, but their varying integration across methods raises fairness concerns. Some approaches leverage LLMs extensively, while others use them minimally or not at all. A fair framework should standardize LLM usage to facilitate meaningful comparisons between different approaches.

Inputs: source schema $S_s = (T_s, A_s, D_s, R_s)$ and target schema $S_t = (T_t, A_t, D_t, R_t)$; LLM $\mathcal{F}$; embedding model Φ; context limit L.
Output: A mapping $\Psi : A_s \to \mathcal{P}(A_t)$

1: $S_s^{roll}, S_t^{roll} \leftarrow$ Rollup$(S_s, S_t, \mathcal{F})$
2: $T_s^{sel}, T_t^{sel} \leftarrow$ TableSelection$(S_s^{roll}, S_t^{roll}, \mathcal{F}, L)$
3: $\Psi_{coarse} \leftarrow$ LLMColumnMatch$(T_s^{sel}, T_t^{sel}, \mathcal{F})$
4: $\Psi \leftarrow \emptyset$
5: **for all** $(a, b) \in \Psi_{coarse}$ **do**
6: **if** b is a rolled-up alias **then**
7: $C \leftarrow$ Drilldown$(a, b, \mathcal{F})$
8: $\Psi \leftarrow \Psi \cup \{(a, c) \mid c \in C\}$
9: **else**
10: $\Psi \leftarrow \Psi \cup \{(a, b)\}$
11: **end if**
12: **end for**
13: **return** Ψ

Algorithm 1: LLMatch

To address these challenges, we propose a LLMatch with a focus on structured evaluation and fair comparison. The processing framework includes three major steps: *Schema Preparation*, *Table Selection* and *Column Matching*, as outlined in Algorithm 1.

Schema Preparation with Rollup In this step, we organize schema information into three categories: table/column names (syntactic content), primary and foreign keys (structural relationships), and table/column descriptions (semantic metadata). This enables LLMatch to assess feature-level contribution to schema matching performance. To further reduce schema complexity and improve alignment performance, we introduce *Rollup* (line 1), a transformation technique that simplifies both source and target schemas by merging semantically related columns into a single, higher-level abstraction. Unlike traditional rule-based processing, *Rollup* relies on LLM to infer semantically related columns and determine meaningful merged aliases. For example, in Fig. 3, *Rollup* groups `currency`,`currency_code_id` and `currency_code` into a new alias `currency_meta_data`. This abstraction reduces the number of elements involved in matching and directs focus to broader semantic patterns rather than low-level features. *Rollup* is particularly useful in schemas where related information is distributed across multiple columns. Importantly, all rolled-up columns are recorded so that, after a high-level match is established, the original columns can be reintroduced in the *Drilldown* phase for fine-grained alignment.

Table Selection This step (line 2) narrows the set of target tables for each source table T_s, ensuring that subsequent matching focuses only on relevant candidates. We evaluate four strategies for candidate selection: (1) None: matching all target tables simultaneously without filtering, (2)Nested Join: processing each target table separately using isolated prompts, (3)Vector Similarity: selecting the top-k most similar tables based on embedding similarity, and (4)LLM: allowing

the LLM to identify relevant tables using schema context. Each strategy reflects a trade-off between context size, semantic filtering, and computational efficiency.

Column Matching with Drilldown After relevant target tables are selected, column matching (line 3) is performed. To enhance alignment precision, we introduce *Drilldown*, a refinement step that revisits the original components of previously rolled-up columns. As shown in Fig. 3, once `source.bal_currency` is matched to `target.currency_meta_data`, the *Drilldown* (line 7) phase triggers a second, focused matching process where the LLM reconsiders the original target columns—such as `target.currency`, `target.currency_code`, and `target.currency_id` as candidates for alignment. Unlike a simple expansion, *Drilldown* does not assume that all rolled-up columns are relevant; instead, the LLM performs a more targeted evaluation within this reduced context. In some cases, only a subset of the rolled-up columns are aligned, while others may be excluded entirely. This two-step process balances abstraction and detail, enabling high-level semantic alignment followed by selective refinement.

5 Dataset Development

Existing schema matching benchmarks often fail to capture the complexity of real-world industry scenarios. They typically consist of a single source and target table, with limited structural depth and no inter-table relationships. To address this gap, we collaborated with major financial institutions to design representative schema matching tasks. While confidentiality constraints prevent releasing of internal datasets, we propose SchemaNet by curating public schema pairs that reflect the structural and semantic complexity of enterprise schemas. Drawn from healthcare, finance, and entertainment domains, SchemaNet provides a foundation for evaluating schema matching in real-world applications.

Dataset Statistics Table 1 compares traditional benchmarks [19] with SchemaNet. SchemaNet contains multiple tables and rich PK/FK relationships compared to flat single-table structures of existing datasets. SchemaNet has higher mapping complexity with more frequent multiple-multiple table mappings and less 1:1 correspondences.

Mapping Development SchemaNet includes seven source–target schema pairs. *Mimic-omop* and *synthea-omop* are adapted from prior work [34,16]; *cms-omop*, *cprd_ aurum-omop*, *cprd_ gold-omop* are reverse-engineered from ETL code and verified by data scientists. The finance pair (*bank1-bank2*) was anonymized and approved for release with industry partners. The entertainment pair (*imdb-sakila*) was manually annotated by experts. All schemas and ground truth mappings are released at [5].

6 Evaluation

We evaluate the following baselines using our proposed LLMatch framework.

Table 1: Dataset statistics for traditional benchmarks (top) and the complex benchmark (bottom) we developed. Complex datasets have more mapped target tables per source table and a low ration of simple (1:1) mappings.

Task	Domain	Source Stats[a]	Target Stats[a]	Mappings Stats[b]
mjs-mjt	Entertainment	1/13/0/0	1/13/0/0	6 / 100%/ 1/1
msjs-msjt	Entertainment	1/14/0/0	1/14/0/0	8 / 100%/ 1/1
mus-mut	Entertainment	1/20/0/0	1/20/0/0	20 / 100%/ 1/1
mvs-mvt	Entertainment	1/13/0/0	1/13/0/0	6 / 100%/ 1/1
imdb-sakila	Entertainment	7/39/2/7	16/90/12/22	22 / 26.7%/ 1.0/3
bank1-bank2	Finance	9/27/5/11	9/36/4/12	11 / 100%/ 0.4/1
cms-omop	Healthcare	5/96/1/4	39/432/12/58	157 / 3.7%/ 7.4/11
synthea-omop	Healthcare	12/111/3/19	39/432/12/58	101 / 19.4%/ 1.5/3
cprd_aurum-omop	Healthcare	8/76/5/21	39/432/12/58	42 / 48.0%/ 1.8/3
cprd_gold-omop	Healthcare	9/123/4/21	39/432/12/58	52 / 25.0%/ 2.4/4
mimic_iii-omop	Healthcare	26/324/6/55	39/432/12/58	189 / 14.7%/ 1.5/5

[a] Number of tables/columns/primary keys/foreign keys.
[b] Total number of mappings/percentage of 1:1 mappings/ average number of target tables per source table/max number of target tables per source table.

- COMA [10]: A classical schema matching approach that combines multiple schema-based matchers, representing schemata as rooted graphs.
- SF [22]: Transforms schemas into directed graphs and propagates similarity scores through neighboring nodes.
- CUPID [20]: Represents schemas as hierarchical tree structures and calculates similarity as a weighted sum of linguistic (name-based) and structural (context-based) similarities.
- UNICORN [39]: A the-state-of-the-art deep learning model that learns from multiple datasets and tasks using a mixture-of-experts model.
- REMATCH [34]: 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 **F1 score** as our primary metric, following prior schema matching works [28,41,3,10]. While alternatives like recall@k [39] and accuracy@k [34] exist, we prioritize F1 as our method treats each match equally and does not produce ranked outputs. During evaluation, we treat foreign key (FK) matches as equal to their corresponding primary keys (PKs).

Implementation We use the `gpt-3.5-turbo` and `gpt-4o-mini` models with default settings, without any parameter modifications or fine-tuning. As LLM outputs are inherently non-deterministic [4], we acknowledge the limited reproducibility of specific responses. All embeddings are generated using SBERT [31]. Since prompt phrasing can significantly affect LLM behavior [33], we release all prompt templates and schema serialization formats in the full report [5].

	Complex Matching Task							Simple Matching Task			
Strategy	1	2	3	4	5	6	7	8	9	10	11
Coma	0.74	0.18	0.13	0.03	0.08	0.01	0.04	0.91	0.71	0.57	**0.91**
SF	0.50	0.21	0.08	0.03	0.03	0.04	0.09	0.71	0.70	0.89	0.59
Cupid	0.32	0.00	0.00	0.00	0.00	0.01	0.00	0.33	0.80	0.86	0.33
Unicorn	0.38	0.00	0.05	0.01	0.03	0.05	0.01	**0.92**	0.80	0.87	0.80
ReMatch GPT-3.5	0.40	0.36	0.06	0.04	0.06	0.07	0.07	0.86	0.70	0.88	0.67
ReMatch GPT-4o-mini	0.74	0.64	0.20	0.19	0.19	0.16	0.20	**0.92**	0.84	0.81	0.71
LLMatch GPT-3.5	0.67	0.47	0.15	0.16	0.26	0.20	0.13	**0.92**	0.82	0.95	0.83
LLMatch GPT-4o-mini	**0.85**	**0.73**	**0.36**	**0.30**	**0.37**	**0.37**	**0.33**	**0.92**	**0.94**	**0.97**	0.83

Columns 1 bank1-bank2; 2 imdb-sakila; 3 cprd_au-omop; 4 cprd_gold-omop; 5 synthea-omop; 6 cms-omop; 7 mimic_iii-omop; 8 mjs-mjt; 9 msjs-msjt; 10 mus-mut; 11 mvs-mvt. Columns 1–7 correspond to Complex Schema Matching tasks, Columns 8–11 to Simple Schema Matching tasks.

Table 2: F1 scores of schema matching methods on simple and complex tasks. Traditional methods (Coma, SF, Cupid, Unicorn) perform well on simple tasks but degrade significantly on complex ones. In contrast, LLM-based approaches (ReMatch and LLMatch) maintain high performance across both, highlighting their effectiveness on complex schema matching.

Evaluation Goals We evaluate the effectiveness of LLMatch by addressing following questions: how different approaches perform on traditional and new benchmarks (Sec. 6), the impact of table selection methods on performance (Sec. 6), and the role of schema elements such as names, descriptions, and PK/FK relationships (Sec. 6). Additionally, we examine how multi-level schema matching with Rollup and Drilldown affects performance (Sec. 6), the implications of LLM context limits for large datasets (Sec. 6), and the productivity gains in schema matching with and without machine assistance (Sec. 6).

End-to-End Performance Evaluation The performance of LLMatch are shown in Table 2. For simple tasks, both traditional and LLM-based methods perform well, with high scores across all datasets. However, traditional methods degrade sharply on complex tasks, consistent with prior findings [42]. On complex benchmarks, LLMatch outperforms all baselines. For example, on the largest dataset, *mimic_iii-omop*, LLMatch achieves an F1 score of 0.4, double that of ReMatch (0.2), and far above Coma (0.04) and SF (0.09).One exception is *bank1-bank2*, where traditional methods perform comparably due to extensive shared vocabulary in the schema, which is effectively picked up by lexical matchers. Fig. 4 shows the performance gap between LLMatch and ReMatch plotted against task complexity. As tasks become more challenging, LLMatch's advantage grows, especially with GPT-3.5-turbo, highlighting the method's robustness even with less capable models.

Table Selection Strategy Study To evaluate the impact of different table selection strategies (Step 2 in Fig. 2), we compare four methods: *None* includes all target tables for each source table in a single prompt. *Nested Join* processes each source–target pair independently using separate prompts. *Vector Similarity* selects the top-k most similar target tables using embedding-based scores. *LLM* prompts the model with source and target table descriptions, asking it to identify relevant candidates directly. Table 3 reports the F1 score and the average number of target tables selected per source table. *LLM* consistently outperforms *Vector*

Strategy	1	2	3	4	5	6	7
None	0.74/(9.0)	0.39/(16.0)	0.28/(39.0)	0.22/(39.0)	0.23/(39.0)	0.21/(39.0)	0.13/(39.0)
Nested Join	0.64/(1.0)	0.32/(1.0)	0.20/(1.0)	0.18/(1.0)	0.10/(1.0)	0.10/(1.0)	0.07/(1.0)
VS[a] (top5)	0.85/(5.0)	0.67/(5.0)	0.19/(5.0)	0.27/(5.0)	0.29/(5.0)	0.35/(5.0)	0.16/(5.0)
VS[a] (top10)	**0.88**/(9.0)	0.67/(10.0)	0.33/(10.0)	**0.37**/(10.0)	0.29/(10.0)	0.36/(10.0)	0.25/(10.0)
VS[a] (top15)	**0.88**/(9.0)	0.59/(15.0)	0.27/(15.0)	0.29/(15.0)	0.32/(15.0)	0.31/(15.0)	0.26/(15.0)
LLM[b]	0.85/(4.6)	**0.73**/(4.0)	**0.36**/(6.3)	0.30/(6.9)	**0.37**/(4.7)	**0.37**/(5.4)	**0.33**/(6.9)

[1] Column order: 1 bank1-bank2; 2 imdb-sakila; 3 cprd_aurum-omop; 4 cprd_gold-omop; 5 synthea-omop; 6 cms-omop; 7 mimic_iii-omop. [a] VS: `Vector Similarity` [b] LLM: gpt-4o-mini

Table 3: Results are 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.

Similarity, suggesting that while *Vector Similarity* is often the default option in semantic retrieval tasks, it may not be the best option for schema matching.

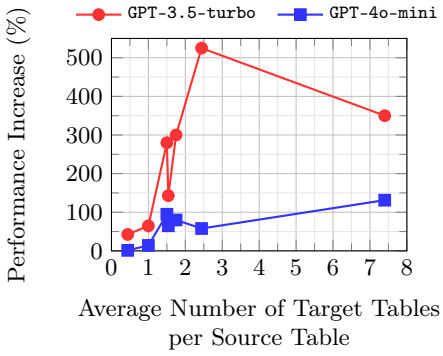

Fig. 4: Percentage improvement of LLMatch over the SOTA baseline (ReMatch) across datasets of varying complexity.

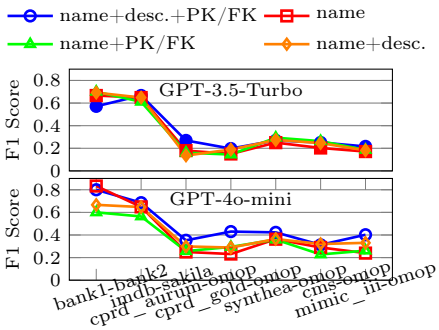

Fig. 5: Schema element ablation study: adding contextual information such as PK/FK and descriptions improves the F1 score.

Schema Elements Ablation Study To evaluate the contribution of schema elements to matching performance, we conducted an ablation study. Figure 5 shows F1 scores across different element combinations. Results indicate that adding *description* and *relationship* improves performance, with the full combination achieving the highest scores.

Effect of Rollup and Drilldown Fig. 6 illustrates the impact of Rollup and Drilldown across different datasets. The performance gains vary with schema complexity. Datasets with more fragmented or relational structures, such as *cprd_gold-omop*, *mimic_iii-omop*, and *synthea-omop*, show the largest improvements. In contrast, simpler datasets like *imdb-sakila* and *bank1-bank2* exhibit only modest gains. These results suggest that the benefits of Rollup and Drilldown are most pronounced in complex matching scenarios, where hierarchical alignment is critical for resolving multi-level relationships.

Scalability Study To assess scalability, we constrain input context size while leaving output unrestricted. For consistency across LLMs, we approximate context size by word count. Tasks exceeding a single-prompt limit are split into

multiple smaller prompts, ensuring the largest source and target tables fit together. This constraint applies to both table selection and column matching. As shown in Fig. 8, overly small contexts fragment the task and degrade performance. These results highlight the importance of sufficient context capacity for effective schema matching on large, complex datasets.

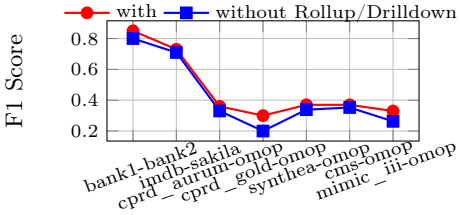

Fig. 6: Rollup/Drilldown improves matching accuracy, with gains up to 40% on some datasets.

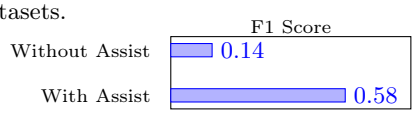

Fig. 7: Productivity Gain Study. Machine assistance significantly improves matching F1 score with equal time per task.

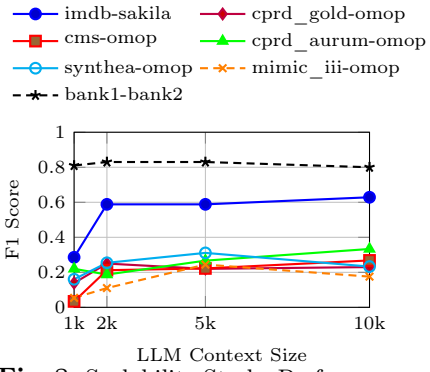

Fig. 8: Scalability Study. Performance drops with small context windows and plateaus once the schema fits entirely.

Productivity Gain Study To evaluate the impact of LLM-generated matches on productivity, we asked annotators to manually align two schemas under two conditions: with and without machine-generated assistance. Each task was limited to two minutes. As shown in Fig. 7, the average F1 across four participants was notably higher with machine assistance, demonstrating the helpfulness of LLM outputs. These findings align with prior findings from [42], who reported that LLM can reduce labeling costs by up to 81% compared to manual methods.

7 Conclusion

This paper presents a framework that decomposes schema matching into three stages: schema preparation, table selection, and column matching. Our key contribution is the introduction of Rollup and Drilldown, a multi-level schema matching addon that simplifies complex schemas prior to matching and refines alignments afterward. Using this framework, we also observed that vector similarity, commonly employed for table selection, is not the most effective method, offering valuable implications for other LLM-based applications. Furthermore, we developed and publicly released a comprehensive, multi-industry benchmark for complex schema matching to support future research in this domain.

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. 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 (2025)
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** (2023)
7. Bernstein, P.A., Madhavan, J., Rahm, E.: Generic schema matching, ten years later. Proceedings of the VLDB Endowment **4**(11) (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: 2025-03-01
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. 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. 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) (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). 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 (2025), accessed: 2025-03-01
15. 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. Springer (2004)

16. Huang, Z., Guo, J., Wu, E.: Transform table to database using large language models. Proceedings of the VLDB Endowment. ISSN **2150** (2024)
17. International Organization for Standardization: ISO 20022-1:2013 – Financial services – Universal financial industry message scheme – Part 1: Metamodel. https://www.iso.org/standard/55005.html, accessed: 2025-05-07
18. Koutras, C., Fragkoulis, M., Katsifodimos, A., Lofi, C.: Rema: Graph embeddings-based relational schema matching. In: EDBT/ICDT Workshops (2020)
19. 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: ICDE. IEEE (2021)
20. Madhavan, J., Bernstein, P.A., Rahm, E.: Generic schema matching with cupid. In: vldb. vol. 1 (2001)
21. 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)
22. 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. IEEE (2002)
23. 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). IEEE (2014)
24. OHDSI: Observational Health Data Sciences and Informatics (2019)
25. 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 Feb 12, 2025
26. 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) (2012)
27. Pan, Z., Yang, M., Monti, A.: Schema matching based on energy domain pre-trained language model. Energy Informatics **6**(Suppl 1) (2023)
28. 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 (2024)
29. 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) (2021)
30. 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) (2024)
31. 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)
32. 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. Springer (2012)
33. 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)

34. Sheetrit, E., Brief, M., Mishaeli, M., Elisha, O.: Rematch: Retrieval enhanced schema matching with llms. arXiv preprint arXiv:2403.01567 (2024)
35. Shraga, R., Amir, O., Gal, A.: Learning to characterize matching experts. In: 2021 IEEE 37th International Conference on Data Engineering (ICDE). IEEE (2021)
36. 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) (2020)
37. Sorrentino, S., Bergamaschi, S., Gawinecki, M., Po, L.: Schema label normalization for improving schema matching. Data & Knowledge Engineering **69**(12) (2010)
38. 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) (2010)
39. 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) (2023)
40. Wornow, D., Suh, G., Elmore, A.J., Krishnan, S., Parameswaran, A.: Automating the enterprise with foundation models. Proceedings of the VLDB Endowment **17**(12) (2024)
41. 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. Springer (2021)
42. 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: International Conference on Data Engineering (ICDE). IEEE (2023)
43. Zhao, H., Ram, S.: Combining schema and instance information for integrating heterogeneous data sources. Data & Knowledge Engineering **61**(2) (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:
```