Towards Scalable Schema Mapping using Large Language Models

Technical Report

Christopher Buss*
bussch@oregonstate.edu
Oregon State University
Corvallis, Oregon, USA

Mahdis Safari*
safarim@oregonstate.edu
Oregon State University
Corvallis, Oregon, USA

Arash Termehchy
termehca@oregonstate.edu
Oregon State University
Corvallis, Oregon, USA

Stefan Lee
leestef@oregonstate.edu
Oregon State University
Corvallis, Oregon, USA

David Maier
maier@pdx.edu
Portland State University
Portland, Oregon, USA

Abstract

The growing need to integrate information from a large number of diverse sources poses significant scalability challenges for data integration systems. These systems often rely on manually written schema mappings, which are complex, source-specific, and costly to maintain as sources evolve. While recent advances suggest that large language models (LLMs) can assist in automating schema matching by leveraging both structural and natural language cues, key challenges remain. In this paper, we identify three core issues with using LLMs for schema mapping: (1) **inconsistent outputs** due to sensitivity to input phrasing and structure, which we propose methods to address through sampling and aggregation techniques; (2) the need for **more expressive mappings** (e.g., GLaV), which strain the limited context windows of LLMs; and (3) the **computational cost** of repeated LLM calls, which we propose to mitigate through strategies like data type prefiltering.

CCS Concepts

• **Information systems → Mediators and data integration**; • **Computing methodologies → Natural language processing**.

Keywords

Data Integration, Generative AI

1 Introduction

There is a recognized need to collect and connect information from a variety of data sources [11, 13, 14]. As an example, we have recently worked in a large-scale NIH-funded project to augment the information of biomedical entities by querying other biomedical data sources [35]. The main focus of this project is to *repurpose* current drugs to treat or mitigate the symptoms of diseases for which there is insufficient time or resources to develop effective treatments (e.g., new or rare diseases) [4]. To support such a system, its developers must find and combine a patchwork of data sources to get a full picture of a drug (e.g., clinical trials, research literature, and adverse effects). Collecting all this information is resource-intensive and can be a barrier to important discoveries.

Data integration systems are complex and often ingest data from a large number of diverse sources. For each source, programmers must manually write mappings that reconcile structural differences. Writing the correct mappings often requires understanding the semantics of the source. Thus, programmers must cross-reference natural-language descriptions (e.g., database documentation), the logical model, and the actual representation of data.

Due to the complexity of mappings, the large number of sources, and the fact that sources evolve over time, integration systems have major scalability problems. Often, mappings are source-specific and cannot be reused. Systems not only become more complex as mappings are added, but with each new source, there is a higher probability that any one source will change, disrupting the system until developers repair the affected mappings. This results in high maintenance costs. To contend with the growing number of sources, we must develop new tools to reduce the manual effort required to build and maintain data integration systems.

Due to the complicated nature of data integration, it requires human-in-the-loop approaches. Many older works have focused on rule-based tools [6, 10, 28], which lack semantic understanding, limiting their usefulness. More recent work has proposed training models to expand their ability beyond basic rules [39]. However, doing so requires preparing labeled data specific to certain domains.

Even more recently, large language models (LLMs) have shown promise in automating a broad range of data processing tasks [3, 25], including schema alignment [26]. Importantly, LLMs are effective on many of these tasks with little-to-no additional training. Additionally, in our own work, we have found LLMs to show promise in adapting to different data domains while requiring little human attention [7, 8]. However, to our knowledge, little-to-no work has been done in investigating how LLMs could help support schema mapping and its related tasks.

Recent studies have explored how large language models (LLMs) can be used to generate schema mappings [15, 21, 22, 27, 29, 30, 36]. Since LLMs can incorporate a wide range of supporting information, including schema metadata and natural language descriptions, they are especially suited to generating mappings.

In this paper we describe three challenges with using LLMs for generating schema mappings.

- **Inconsistent Outputs**: LLMs are highly sensitive to input phrasing and structure, leading to unpredictable and diverse results. We propose techniques for sampling and combining multiple outputs, increasing our mapping coverage and providing effective ways to filter out unlikely mappings. We

*Both authors contributed equally to this research.

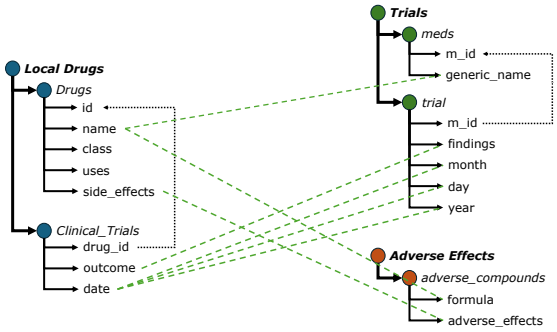

Figure 1: Example integration scenario with a target (left) and two sources (right). Green dashed lines represent semantic correspondences between attributes (schema alignment). Dotted represent inter-schema references (foreign keys).

provide preliminary results that illustrate the effectiveness of our methods.

- **GLaV with Limited Contexts**: Existing LLM-based methods focus on limited mapping types, which are often insufficient for many real-world integration scenarios. Towards supporting more integration systems, we consider the challenges associated with generating more expressive mappings (GLaV). Supporting such mappings would increase complexity, requiring more sophisticated representations and careful design to avoid overwhelming the LLM's context.
- **Challenge: Efficient Prompting**: LLM-based schema mapping is computationally expensive due to repeated model calls, especially with large datasets. This, in large part, is thanks to its large input, which only becomes more difficult to manage as we consider more expressive mappings.

2 Schema Mapping Assistant

Our discussion is inspired by tools [6, 10, 28], meant to assist the user in the schema mapping process. Generally speaking, these systems take the full set of possible mappings and filter them down to a candidate set. The candidate set is then shown to the user so they may verify them, selecting which mappings to implement. This work focuses on mappings between relational databases, but we assert that the challenges highlighted here apply to schema mapping broadly, regardless of the logical models used.

It is expected that users will need to verify the output mappings as the underlying model used to generate them will not know the user's latent intent. In essence, there is nearly always expectations (e.g., business rules not represented within the schema) for which the underlying model is not privy to.

2.1 Preliminaries

Data Integration System. Following previous works [19], we describe a data integration system $\mathcal{I}$ as a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where

- $\mathcal{G}$ is the *target schema*, which describes a unified view of sources.
- $\mathcal{S}$ is the *source schema*, which specifies the structure of the sources to integrate. For the sake of definitional simplicity, we do not distinguish between different sources; instead, we consider $\mathcal{S}$ to simply be the union of all source schemas.
- $\mathcal{M}$ is the mapping between $\mathcal{G}$ and $\mathcal{S}$, constituted by a set of *rules*, each describing how a subset of $\mathcal{G}$ semantically corresponds to a subset of $\mathcal{S}$.

Both $\mathcal{S}$ and $\mathcal{G}$ contain relations $S_1, S_2, ...S_n$ and $G_1, G_2, ...G_m$ respectively. In turn, each relation contains a set of attributes denoted as $attr(S_i) = \{s_1, s_2, ..., s_l\}$ and $attr(G_j) = \{g_1, g_2, ..., g_k\}$.

Example 1. Figure 1 represents an integration scenario within the drug domain. Specifically, to better understand whether certain drugs can be used to treat certain rare diseases, we want to integrate information about each drug's clinical trials and adverse affects, and likely many other sources not picture here. In our running example, we focus on integrating information from the Trials source: given $\mathcal{S} = \{meds, trial\}$ and $\mathcal{G} = \{Drugs, Clinic\_Trials\}$, we must define a mapping ($\mathcal{M}$).

Mapping Rules (st-tgds). We formally express rules as Source-to-Target Tuple-Generating Dependencies (st-tgds) [12],

$$\forall \vec{x} \ (\phi(\vec{x}) \rightarrow \exists \vec{y} \ \psi(\vec{x}, \vec{y}))$$

where

- $\phi(\vec{x})$ is a conjunction of atoms over the source $\mathcal{S}$.
- $\psi(\vec{x}, \vec{y})$ is a conjunction of atoms over the target $\mathcal{G}$.
- $\vec{x}$ are universally quantified variables.
- $\vec{y}$ are existentially quantified variables.

Both $\phi(\vec{x})$ and $\psi(\vec{x}, \vec{y})$ may include additional predicates (e.g., for filtering tuples). In essence, each rule asserts a pattern over the source, that, if matched, generates tuples adhering to the corresponding pattern in the target.

Example 2. Continuing our example in 1, we indicate which tuples in Trials should trigger the generation of tuples in Local Drugs. Further, we indicate which attributes within the new target tuples should be populated, and how that population is determined by the attributes within the triggering source tuples.

$$\begin{aligned} \forall i, g, f, m, d, y \ (&\text{meds}(i, g) \wedge \text{trial}(i, f, m, d, y), y > 1990 \\ &\rightarrow \exists x_1, x_2, x_3, x_4 \ (\text{Drugs}(x_1, \tau_1[g], x_2, x_3, x_4) \wedge \\ &\quad \text{Clinical\_Trials}(x_1, f, \tau_2[m, d, y]))) \end{aligned} \tag{1}$$

Target attributes are populated based on their semantic counterparts in the source. In some instances, value-level transformations are necessary, such as translating a drug's generic name to its brand name (τ_1) and concatenating date-parts (τ_2). However, we make a distinction between value-level transformations and schema-level transformations. This work focuses on the latter, though, in the long-term, we foresee it being useful, especially for complex value-level transformations, to tackle the former problem as a separate step from that of schema-level transformations.

Referential Dependencies. Often, rules contain *referential dependencies* which condition the existence of rows in one relation upon the existence of join-able rows in another. In the case of rule 1, we specify referential dependencies over both the source and target. Over the source, the shared variable i in *meds* and $trial$ forces tuples

from these relations to fall within the same equi join on *m_id*. Over the target, the shared variable x_1 implies the creation of a surrogate key for each answer within the source, ensuring that rows in *Drugs* are connected to their corresponding rows in $Clinical\_Trials$. Importantly, referential dependencies are expressed via the presence of the same variable (i and x_1) within *multiple* relational predicates on the left-hand side (*meds* and $trials$) and right-hand side (*Drugs* and $Clinical\_Trials$) of the *same* rule.

2.2 Rule Expressiveness

Rules are commonly divided into three classes, each of which is defined by the number of relational predicates allowed over the source and target. Formally, these classes are called Global-as-View (GaV), Local-as-View (LaV), and Global-Local-as-View (GLaV). We refer interested readers to [19] for a more detailed comparison of these three classes. For the sake of our exposition, we simply make the distinction between those classes that limit either side of a rule to one-and-only-one relational predicate (i.e., GaV and LaV) and the class that does not (i.e., GLaV). Henceforth refer to the former class as *limited referential dependencies (LRD)* and the latter class as *full referential dependencies (FRD)*. The rule written in Equation 1 falls strictly within the FRD class as more than one relational predicate appears on both sides.

Example 3. To demonstrate the limitations of LRD, we translate rule 1 (written as one FRD rule) into a mapping containing only LRD rules,

$$\forall i, g \ \ (\text{meds}(i, g) \rightarrow \exists x_1, x_2, x_3, x_4 \ \text{Drugs}(x_1, \tau_1[g], x_2, x_3, x_4))$$

$$\forall i, f, m, d, y \ \big(\text{trial}(i, f, m, d, y), y > 1990 \rightarrow \exists x_5 \ \text{Clinical\_Trials}(x_5, f, \tau_2[m, d, y])\big)$$

Note that the translation from FRD to LRD requires two rules, isolating the source and target relations. This is problematic since the variables i and x_1 do not share the same scope across rules. In other words, the LRD mapping will result in a target instance which does not specify which clinical trials concern which drugs. Further, the instance will contain all drugs in *meds* regardless of their most recent clinical trial.

Schema Alignments. Rather than generating schema mappings directly, many works focus on the simpler task of generating schema alignments, which can eliminate many undesired mappings from consideration. A schema alignment is a set of pairs, $\{(s_l, g_k) \mid s_l \in attr(S_i), g_k \in attr(G_j)\}$ where a pair asserts that source attribute s_l semantically corresponds to target attribute g_k. In limited cases, algorithms can produce the exact correct mapping rules given the alignments as input [28]. Figure 1 includes schema alignments for our running example.

2.3 Problem Definition

Given a source schema $\mathcal{S}$, a global schema $\mathcal{G}$, and a set of *hints*, a *Schema Mapping Assistant* must produce a candidate set of mappings C of some class as described in Section 2.2. From the schemata itself, we are guaranteed to have certain information, including relation names, attribute names, and any constraints stated within the schema (e.g., primary keys, foreign keys, data types, etc,.).

2.3.1 Hints. In addition to the information given by the schemata, we may also have additional contextual information, which we call *hints*, that can be leveraged for determining C. In current work, we consider natural language descriptions of tables $tdesc(\cdot)$ and attributes $adesc(\cdot)$, as well as sample data values $val(\cdot)$.

Example 4. For example, the *Drugs* in Figure 1 might have the following hints,

$$\begin{aligned} name(T) &= \text{"}Drugs\text{"} \\ tdesc(T) &= \text{"}Information\ on\ class,\ uses,\ and\ side\ effects\text{"} \\ datatype(T) &= \{int, String, String..., \} \\ adesc(T) &= \{\text{"}Unique\ ID\ for\ Drug.\text{"}, \\ &\qquad \text{"}Brand\ name\ of\ drug.\text{"}, ..., \} \end{aligned}$$

Not pictured are other hints that are given by the schema definition.

The availability of additional hints depends on the sources themselves. Data values may not be available, either because the relations are empty or the data itself is restricted due to privacy concerns. In practice, natural language descriptions may be derived from documentation, but even this is not guaranteed to exist.

2.3.2 Candidate Set Quality. The goal is to provide users with a high-quality candidate set. Ultimately, what makes a candidate set "high-quality" depends on many factors, including user preference. However, we often prioritize recall over precision: maximize the number of true mappings while simultaneously minimizing the number of false mappings. This is because it often takes less effort for a human to confirm that a candidate mapping is wrong than it does for them to determine the correct mapping *on their own*.

3 Challenge: Inconsistent Outputs

The output of LLMs depend heavily on how their input is phrased. The space of possible outputs is often diverse and varies significantly in terms of quality. Further, it is difficult to predict the relative impact that different phrasings will have on the output. For example, minor adjustments in the ordering of content—such as rearranging rows or columns in a table—can impact accuracy, as LLMs are sensitive to the structure of the input data [32].

However, most existing schema matching approaches overlook these sensitivities. They typically rely on a single, static prompt and do not account for structural variations in the input. For example, Parciak et al [27] repeat the same prompt multiple times and aggregate the outputs using majority voting to approximate a high-confidence result, rather than varying the prompt itself. In contrast, we treat prompt variation as a key mechanism for exploring the model's output space more effectively and improving overall quality of the final candidate set.

3.1 Sampling Outputs

Instead of relying on a single prompt, we treat this as a sampling problem. More specifically, we view the model as a black-box from which we can sample mappings by providing different prompt phrasings. More formally, we start with a reasonably effective prompt template $P(\cdot)$, and then apply a set of transformations that exploit symmetric properties of our problem. We prompt the LLM n times to produce candidate sets $c_1, c_2, ...c_n$. We then combine

these sets using a function $A(\cdot)$ to produce the final candidate set $C' = A(c_1, c_2, ...c_n)$.

Sampling multiple outputs for the same input can benefit us in two key ways. First, it expands our coverage of the hypothesis space, increasing the chance of discovering high-quality alignments that might otherwise be missed with a single static prompt. Second, it provides insight into the relative likelihood of different mappings: if a particular output appears consistently across multiple samples, it may serve as a proxy for confidence in its quality.

3.1.1 Symmetric Transformations. Starting from a well-performing base prompt, we apply transformations that alter the input format without changing the underlying task. Specifically, we introduce variation by randomly permuting the order of columns in the prompt, sampling different data instances, and swapping the source and target tables. These transformations preserve the semantic equivalence of the task while encouraging the model to explore different parts of the output space.

3.1.2 Sample Merging. Chen et al. [9] propose a consistency sampling method where the LLM selects the most coherent response from multiple outputs generated using a single prompt. While effective for free-form tasks, this approach requires an additional LLM call over a long, concatenated prompt, which is problematic in schema matching due to the size and complexity of input schemas. In contrast, we generate outputs from multiple transformed prompts, applying symmetric variations to explore different parts of the alignment space. Rather than selecting a single response, we collect multiple candidate mappings per attribute for further analysis. To merge the results, we apply logical operations over structured JSON outputs, offering a scalable and inference-efficient alternative. The choice of operation depends on the desired trade-off between recall and precision: union includes all possible matches to maximize recall, majority vote selects the most frequent alignments, and intersection keeps only those consistently predicted across prompts.

3.2 Bidirectional Schema Matching

To account for alignments from both the original and swapped table perspectives, simple aggregation functions like majority vote or union are not ideal. These methods fail to consider the differing confidence levels that the LLM may have for matches from each perspective. Instead, we propose estimating a confidence score for each candidate match in both directions before merging the results, leading to a more reliable combination.

MatchMaker estimates the LLM's confidence in alignments by prompting it to score candidate matches for each attribute. However, these scores may not accurately reflect true probabilities. Inspired by [37], we approximate the LLM's confidence by first asking it to select the best match among the candidates, then using its output logits to compute confidence scores.

Combining results from the original and swapped table perspectives presents a challenge, as its not clear which merging method is most effective. We explore different techniques for this task. As a starting point, we test two simple approaches: averaging and multiplying confidence scores. Averaging reduces the confidence if an alignment appears in only one direction, while multiplication removes alignments that are missing from either direction.

In addition, we apply the stable matching algorithm [2] to the ranked matches from both directions. This approach conceptually aligns with the table-swapping process, as it reflects the bidirectional nature of preferences and ensures a stable and consistent matching between attributes. We define Stable Schema Matching as follows:

Definition 3.1 (Stable Schema Matching). Given two schemas A and B, the input consists of two sets of attributes, $A[\text{attr}] = \{a_1, a_2, \ldots, a_n\}$ and $B[\text{attr}] = \{b_1, b_2, \ldots, b_m\}$, along with ranked preference lists for each attribute in $A[\text{attr}]$ over the attributes in $B[\text{attr}]$, and vice versa. The goal is to compute a stable matching between attributes in A and B such that no unmatched pair of attributes would prefer each other over their current matches. The output is a set $M = \{(a_i, b_j) \mid a_i \in A[\text{attr}], b_j \in B[\text{attr}]\}$ containing the stable matches between attributes in A and B, ensuring that the matching is mutually acceptable and stable. The number of matches can be constrained by a parameter K, which specifies the top K stable matches for each attribute.

An overview of our bidirectional matching design is shown in Figure 2. For more details on our prompts, how we compute the confidence score, and the stable matching algorithm, see Appendix B.

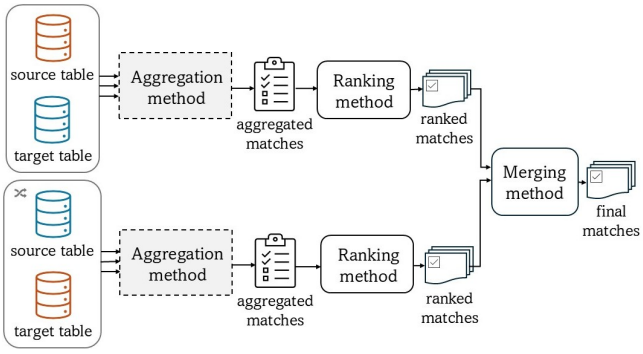

Figure 2: Bidirectional schema matching by swapping source and target tables. Results from both directions are aggregated, ranked, and merged.

3.3 Preliminary Results

In this section, we evaluate the impact of our proposed prompting strategies and bidirectional approach with symmetric transformations, demonstrating how these methods help an open-source model achieve competitive performance compared to proprietary models.

Problem Definition. We focus on the simplest form of mapping, which is nonetheless a difficult task. Specifically, given relations S_i and G_j, we want to to produce a set of pairs, each representing an alignment between a source attribute and a global attribute. Formally, $C = \{(s_l, g_k) \mid s_l \in attr(S_i), g_k \in attr(G_j)\}$ indicating that the attribute s_l semantically corresponds to the attribute g_k.

Prompt Template and Techniques. We use reasoning-based prompting strategies to improve LLM performance in zero-shot

settings [16]. Specifically, we prompt the LLM three times with transformed inputs using a fixed seed. To ensure consistent output formatting and enable efficient processing, we constrain the model to produce structured JSON output [34]. Each prompt follows an N-1 format, where N source attributes and one target attribute are serialized in JSON to improve structural understanding [31, 32]. This N-1 strategy has been shown to outperform other settings in terms of matching effectiveness [27]. We conducted experiments to evaluate the impact of different types of metadata. Our results show that natural language descriptions of attributes consistently provide the largest boost to schema matching performance. Even with basic schema details such as table names, attribute names, and data types, the LLM-based method outperforms traditional approaches such as COMA in the single-prompt setting. We included data values based on the idea that example values could help the model better understand attribute meaning. Interestingly, data values caused a slight drop in performance in single-prompt setups but proved more helpful when aggregating results across multiple prompts—each varying in column order and sampled values. For more details and results, please refer to Appendix E. To improve the semantic reasoning of the LLM, we include all available schema metadata in the prompt, including attribute names, data types, descriptions, ten randomly sampled unique data values, and relation descriptions.

Datasets. We evaluate our method on two widely used schema matching benchmarks: MIMIC-OMOP and Synthea-OMOP. MIMIC-OMOP aligns real-world clinical databases, MIMIC-III and the OMOP Common Data Model, across 26 schema pairs. It includes 268 source attributes, 203 target attributes, and 155 ground truth matches. To populate OMOP with sample values, we use data from MIMIC-IV, which avoids overlap with MIMIC-III while remaining conceptually aligned. However, some aligned columns lack sufficient sample data due to privacy restrictions or missing data. Many columns also lack descriptions, which adds complexity to schema matching. Synthea-OMOP is derived from synthetic healthcare records generated by Synthea and aligned to OMOP. It consists of 12 schema pairs, 101 source attributes, 134 target attributes, and 105 matches. Although synthetic, the dataset captures realistic variability and schema ambiguity, making it a strong benchmark for evaluating schema matching methods.

Baselines. We compare our methods with three baselines that require no training data. The first is COMA [10], a widely used schema-matching method known for its efficiency and flexibility, which has been refined through several iterations [5, 23]. COMA is a rule-based approach that lacks semantic understanding, with the schema-based version considering only schematic information, and the instance-based version incorporating both schematic information and data values. As a result, COMA can miss important semantic nuances and struggle with the complexities of real-world schemas like those in MIMIC-OMOP and Synthea-OMOP, which affects its overall matching performance. We evaluate both versions using Valentine's Python wrapper for COMA 3.0[1] [17].

For language model baselines, we use the N-1 prompting method from Parciak et al. [27]. This method is simple and effective. It works by aggregating multiple prompts, similar to our own aggregation approach. However, It does not provide ranked match suggestions, which are often useful in practice. It also restricts mappings to one-to-one correspondences, overlooking scenarios where an attribute may align with multiple counterparts. We also evaluate MatchMaker [29], which refines alignments with pre- and post-filtering steps. Although MatchMaker improves performance by filtering, its multi-step prompting pipeline introduces inefficiencies and can fail if intermediate language model outputs deviate from expected patterns. Both methods are implemented based on their respective papers, and for a fair comparison, we exclude the pre-filtering step in MatchMaker, as our method does not include it.

Metrics. We report the average Precision@k, Recall@k, and F1@k for both our approach and the LLM baselines, averaged across three different random seeds. For methods that do not rank the alignments, we report metrics at $k = \max$. For the bidirectional methods using stable matching and multiplication, the value of k is limited based on the pipeline, as the final alignments must be present in the aggregated candidates from both directions. Therefore, we report a limited k for these methods. For the method using averaging, we report the maximum k obtained from the MatchMaker method. In real-world data integration scenarios, automated matching candidates require manual validation. Therefore, we aim to achieve high precision and recall simultaneously to reduce manual effort while ensuring high-quality matches.

Many studies across various tasks, including schema matching, demonstrate that larger language models perform better [16, 27, 33, 38]. However, the most advanced models are typically available only via APIs, which raises significant privacy concerns. In these experiments, we use the Meta-*Llama-3.1-70B-Instruct-GPTQ-INT4* [2] , the largest model we could run on the server. More details on the evaluation setup can be found in Appendix E.

Method	Experiment	k	P@k	R@k	F1@k
Aggregation	Original tables	max	0.35	0.79	0.47
	Swapped tables	max	0.47	0.67	0.54
Bidirectional	Stable Matching	1	0.68	0.62	0.64
		2	0.66	0.63	0.64
	Average	1	0.37	0.78	0.49
		2	0.31	0.82	0.44
		3	0.30	0.83	0.43
	Multiply	1	0.67	0.62	0.64
		2	0.66	0.63	0.64
Baseline	MatchMaker	1	0.25	0.24	0.23
		2	0.15	0.30	0.19
		3	0.11	0.31	0.15
	COMA Sch.	max	0.14	0.11	0.10
	COMA Inst.	max	0.21	0.14	0.16
	Parciak et al	max	0.30	0.15	0.18

Table 1: Precision@k (P@k), Recall@k (R@k), and F1@k for different methods on the MIMIC dataset.

[1] https://github.com/delftdata/valentine

[2] The "70B" refers to the model's size, with 70 billion parameters. "Instruct" indicates that the model is fine-tuned for instruction-following tasks. "GPTQ" is a quantization method that optimizes memory efficiency and improves inference speed. "INT4" refers to 4-bit integer quantization, a specific technique used within GPTQ to further reduce memory usage while maintaining performance.

3.3.1 ***Evaluation Results.*** The results in this paper use the majority vote aggregation method, as it offers the best balance between recall and precision. Results for other aggregation methods are available in Appendix E. As shown in Tables 1 and 2, our aggregation method outperforms all baselines. In addition to our symmetric transformations, the key difference from Parciak et al. lies in the output format and schema serialization used in the prompt. Their method struggles with incomplete responses, where the model often skips the final decision[3]. MatchMaker also fails when the LLM does not follow the required format in intermediate steps. While both MatchMaker and Parciak et al. used GPT-4 in their studies, we employed a significantly smaller model in our experiments. As a result, our prompt is more effective for smaller models, which we attribute to its clear output format and structured JSON schema serialization.

On the MIMIC dataset, LLM-based methods outperform COMA, highlighting that LLMs excel in domains where domain knowledge is crucial, thanks to their ability to process natural language descriptions. In contrast, on the Synthea dataset—where vocabulary and attribute names require less domain knowledge—Parciak et al.'s method does not outperform COMA. This suggests that when domain knowledge is less critical, the choice of pipeline and prompting method, especially for smaller models, becomes more important.

We also observe that the bidirectional method using multiplication outperforms the others, achieving the best F1@1 score. The bidirectional method using stable matching closely follows. The difference lies in our use of confidence score ranking in stable matching, while multiplication computes final alignment confidence based on the actual values of scores from each direction. These methods are effective for tasks where both precision and recall are important. The bidirectional method using averaging, though lower in precision, excels in recall and is preferred when high recall is prioritized.

[3]Example: *The first attribute to consider is* {$source\_attribute$}. *Does* {$source\_attribute$} *semantically match* {$target\_attribute$}*?*

Method	Experiment	k	P@K	R@K	F1@K
Aggregation	Original tables	max	0.57	0.95	0.70
	Swapped tables	max	0.52	0.56	0.51
Bidirectional	Stable Matching	1	0.78	0.52	0.60
		2	0.77	0.55	0.62
	Average	1	0.58	0.95	0.71
		2	0.50	0.95	0.65
		3	0.48	0.96	0.63
	Multiply	1	0.77	0.55	0.62
		2	0.77	0.55	0.62
Baseline	MatchMaker	1	0.45	0.21	0.27
		2	0.23	0.21	0.21
		3	0.15	0.21	0.17
	COMA Sch.	max	0.30	0.15	0.19
	COMA Inst.	max	0.30	0.16	0.20
	Parciak et al	max	0.53	0.11	0.17

Table 2: Precision@k (P@k), Recall@k (R@k), and F1@k for different methods on the Synthea dataset.

We compared our methods against LLM baselines originally designed for GPT-4. MatchMaker is directly comparable, as it also evaluates on the MIMIC and Synthea datasets. We assess performance by comparing our bidirectional method, based on an open-source model, with the full MatchMaker pipeline using their reported accuracy@1. As shown in Table 3, our bidirectional method using stable matching and multiplication achieves accuracy comparable to MatchMaker on the Synthea dataset. It also outperforms MatchMaker on the MIMIC dataset.

Dataset	Method	Accuracy@1
MIMIC	MatchMaker	62.20 ± 2.40
	Bidirectional (Stable Matching)	0.78 ± 0.00
	Bidirectional (Average)	0.49 ± 0.01
	Bidirectional (Multiply)	0.77 ± 0.01
Synthea	MatchMaker	70.20 ± 1.70
	Bidirectional (Stable Matching)	0.69 ± 0.01
	Bidirectional (Average)	0.64 ± 0.01
	Bidirectional (Multiply)	0.70 ± 0.01

Table 3: Comparison of our proposed method using Llama-3.1-70B-Instruct-GPTQ-INT4 against MatchMaker's using GPT-4.

Our bidirectional approach, combined with symmetric transformations, delivers strong results on clinical datasets like Synthea and MIMIC, achieving accuracy comparable to or exceeding GPT-4-based models such as MatchMaker. While their results were obtained using a significantly larger model, our approach, built on a smaller open-source model, performs competitively. This highlights that strong performance can be achieved not just through model scale or domain knowledge, but through careful pipeline design. In addition to improved prompting strategies, such as more effective schema serialization, our method introduces order variations at the table, column, and data value levels to encourage broader exploration of the alignment space. These design choices highlight the impact of pipeline structure in maximizing the effectiveness of LLMs for schema matching, even without access to proprietary models.

3.3.2 Sampling Techniques for FRD Mappings. Our evaluation indicates that, with the correct sampling techniques, open source LLMs are highly competitive with at least one major, proprietary model (GPT-4). However, these techniques depend on decomposing the LLM's responses into a fairly limited number of atomic elements. For example, the final answer in the LLM's response for schema alignment is pairs of source columns and target columns. It is easy to break the output into sets of these pairs, making the ordering irrelevant, Further, each pair is reflexive in the sense that (A, B) is identical to (B, A), allowing us to leverage bidirectional techniques. In essence, sampling techniques require outputs that are simple enough such that they can be broken down into atomic elements and overlap between responses can be calculated.

Extending these techniques to schema mappings is not as easy do to the increased complexity in the output language. A mapping is essentially a collection of queries, and it is not immediately clear how one might effectively partition such an output to test for overlap. One way to partition the output is on a per-query basis, testing for the repetition of queries within outputs. However, using exact

string comparison would be much too restrictive, likely leading to very little overlap, even if there are logically-equivalent queries. Simultaneously, the logical-equivalency of queries is not guaranteed to be decidable [1]. Further, if a given rule is not invertible, then the bidirectional technique cannot be used. This makes such techniques incorrect in a theoretical sense, but empirically, such techniques may still be useful if employed correctly. Clearly, this is an important subject for future research.

4 Challenge: Representation and Large Input

As discussed in Section 2.2, existing research does not consider FRD rules making the associated techniques insufficient for many common mapping scenarios. In this section, we consider two immediate challenges associated with Zero-Shot generation of FRD mappings using LLMs. First, we discuss the difficulties with representing FRD rules. Second, we discuss the overall complexity of generating a full FRD mapping. We close this section with an empirical study meant to help us better understand these challenges and establish future research.

4.1 Output Representation

LLMs have show great success at generating SQL [20], making it a promising candidate for representing rules. However, an SQL query can only represent a single GaV rule because it's output is always only a single table. In fact, most popular query languages can only produce individual GaV rules. That being said, multiple SQL statements can be used in tandem to represent a GLaV rule. Whether an LLM will produce such scripts is another issue. That being said, it is possible to translate SQL into other classes. For example, in the context of importing tabular data into a relational target, one work prompts an LLM to generate a query over the target and then inverts it to produce a LaV rule [15]. However, this approach only works if the GaV counterpart is invertible itself, a theoretically needless limitation.

4.2 Input

Given some unknown rule, the LLM can only generate it if it is provided, at minimum, the schema fragments appearing in the rule. On the other head, existing works have shown that filtering input to that which is relevant often improves performance. It is reasonable to believe that we would also benefit from filtering out irrelevant schema. However, such a task is not straightforward.

A mapping could be quite large, with rules covering the entirety of the source and target schemas. In such an instance, it perhaps makes more sense to break the full mapping down into parts. A sensible approach would be to segment the full mapping at the rule level. However, since we do not know the underlying rules, we would need to predict their contents related to source and target relations.

Relations within Rule k. As the same relation may appear multiple times within a conjunction, we define $\mathcal{S}^k$ and $\mathcal{T}^k$ to be the unique relations appearing in $\phi(\vec{x})$ (i.e., source relations appearing in left-hand side of the rule) and $\psi(\vec{x}, \vec{y})$ (i.e., target relations appearing in right-hand side of the rule), respectively, for the kth rule.

We do not know either $\mathcal{S}^k$ or $\mathcal{T}^k$ for any given rule (much less the rule itself). And finding $\mathcal{S}^k$ and $\mathcal{T}^k$ is, itself, a form of schema filtering. However, knowing neither set of relations makes filtering a potentially multi-step process where we must predict subgraphs of the target and the source and then pair those subgraphs across the source and target correctly. This may be quite hard, perhaps motivating solutions that are very relaxed in how they filter schema.

When developing a schema filter, we must consider its restrictiveness: we want a filter that maintains as many relevant schema parts as possible while also reducing the size of the context. This presents an optimization problem where we must decide how aggressively to filter such that the LLM's performance does not significantly degrade either due to a lack of context (information) or an overly noisy context (too much irrelevant information).

4.3 Preliminary Results

In this section, we present experiments meant to provide some preliminary insight into the two questions raised in this section: 1) in what ways is SQL sufficient (insufficient) for generating GLaV rules? and 2) How does the input size affect the final set of mappings? For the later question, we assume that we have already found $\mathcal{S}^k$ and $\mathcal{T}^k$ for each kth rule. The question is, how do we chunk these specifications into prompts.

Prompt Template and Techniques. Similar to our prompt discussed in Section 3.3, we assume a zero-shot setting and encourage the model to reason about its mapping prior to producing its final script. Each prompt can contain multiple source relations and target relations, each of which is serialized as JSON. We include all available and relevant metadata. For each relation, we include its name, primary key, and any foreign key relationships. For each attribute, we include its name, type, whether it is NULLable, and up to 10 instance data values, uniformly sampled without replacement. Each sampled data value is truncated to 100 characters.

Dataset. We use a subset of Amalgam [24], a commonly-cited schema mapping benchmark. Of Amalgam's four bibliography databases, we take S1 as our source and S2 as our target. The gold mapping contains 7 rules. Generally speaking, it involves decomposing publication-type relations (S1) into attribute-specific relations (S2). During our research, we have noticed a surprising lack of schema mapping benchmarks despite the prevalence of the problem. Unfortunately, many benchmarks have broken links. We have pieced Amalgam together from a few places: we use the schema definitions from the original source [24], the data provided by [18], and the ground truth as provided by the iBench scenario Github page[4]. Notably, the ground truth mappings are specified using a proprietary format. We translate these to SQL for our purposes. We hope that the inaccessibility of the original dataset helps circumvent any data leakage issues. Amalgam does not contain any natural language descriptions of attributes or relations. However, given its common domain, we hypothesize that an LLM should have a general understanding of the domain.

Metrics. We borrow a metric commonly used in Text-to-SQL called execution accuracy which measures the overlap between the

[4] https://github.com/RJMillerLab/ibenchScenarioCollection

results obtained from predicted queries and ground-truth queries. However, instead of only reporting either 1 (full overlap) or 0 (anything less than full overlap), we report the percentage of overlap. As discussed in Section 2, most data integration systems follow a human-in-the-loop approach, where a user can validate and fix predicted mappings. Thus, it is useful to report the proximity of a predicted mapping to that of the ground truth.

To test overlap, we need to compare the effect of both mappings given the same input target-instance. For evaluation data, we generate 100 rows for each table in the target database. When schematically valid, we insert NULLs into attributes for some rows and remove some foreign key references, resulting in some parent rows with no children (i.e., some authors with no publications and some publications with no authors). We apply both the gold and predicted mapping to produce a gold target instance I' and a predicted target instance I. We then apply an exhaustive set of test queries to both instances. For each test query q_i, we calculate false positive rows (FP), false negative rows (FN), and true positive rows (TP) as follows,

$$FP_i = q_i[I] - q_i[I'] \qquad FN_i = q_i[I'] - q_i[I] \qquad TP_i = q_i[I'] \cap q_i[I]$$

We then calculate recall (R), precision (P) as,

$$R_i = |TP_i| \;/\; (|TP_i| + |FN_i|) \qquad P_i = |TP_i| \;/\; (|TP_i| + |FP_i|)$$

Finally, F1-score is calculated as $F1_i = (2.0 * R_i * P_i) \;/\; (R_i + P_i)$.

We test for two kinds of overlap: Table Overlap and Join Overlap. In both cases, we project over all columns *except for arbitrary primary keys and foreign key references*. For our dataset, these keys have no semantic meaning. Thus, correctness depends on upon establishing the correct references between rows and not on the particular values used to do so. our only expectation is that, no matter the values assigned to these keys, the correct references are established between rows.

Table Overlap. Let $T_1(x_1, \bar{x}_1), T_2(x_2, \bar{x}_2), ..., T_n(x_n, \bar{x}_n)$ be relations in the target schema, where $x_1, x_2, ..., x_n$ is each relation's respective set of attributes (columns), and $\bar{x}_1, \bar{x}_2, ..., \bar{x}_n$ are the primary key columns and foreign key references for their respective relation. For each T_i, we test for overlap using $q_i(x_i) : -T_i(x_i \cap \bar{x}_i)$ if both $q_i[I']$ and $q_i[I]$ return no results, then we do not consider q_i in our final calculation. This prevents the overlap from being inflated by target tables that are not touched by any mapping.

Join Overlap. We indicate the target relations appearing in the kth gold mapping with $\mathcal{T}_k = T_1(x_1, \bar{x}_1), T_2(x_2, \bar{x}_2), ..., T_l(x_l, \bar{x}_l)$. For each $\mathcal{T}_k$, we test for overlap using $q_k(\bigcup_{j=1}^{l} x_j) : -T_1(x_1 \cup \bar{x}_1) \bowtie T_2(x_2 \cup \bar{x}_2) \bowtie ... \bowtie T_l(x_l \cup \bar{x}_l)$. In instances where $|\mathcal{T}_k| = 1$, we drop query q_k as it devolves to a query with no joins, which is already covered by Table Overlap. In instances where we have multiple, identical queries due to target-relation overlap in our gold mapping, we remove all but one of the queries. After calculating individual metrics, we take the average of all queries.

It is worth noting that perfect Join Overlap (i.e., F1 = 1.0) also implies perfect Table Overlap, but perfect Table Overlap does not imply perfect Join Overlap. In fact, low Table Overlap is likely connected with very low table overlap scores given the nature of the metrics. Producing high Join Overlap requires that the model effectively generate the correct references between rows, sometimes requiring the model to invent new keys.

4.3.1 Evaluation Results. We conduct an empirical evaluation to answer two questions. First, we want to know how does chunking affect performance; namely, are there benefits of producing more chunks at a time. What are the drawbacks of having the LLM do more work with fewer prompts? Second, we want to better understand the limitations of a baseline approach, which will help establish promising research directions.

Rules / Prompt	Input Tokens	Output Tokens
1	3910	1104
2	5024	1484
3	6425	1838
4	7596	2053
5	8839	2489
6	9983	2849
7	11259	2639

Table 4: The average number of input and output tokens according to the number of rules a prompt is given specification for (i.e., the underlying source and target relations). Each average is calculated over 20 random prompt/response pairs.

To understand the effect of input size on mapping quality, we vary the input size of prompts with respect to how many $(\mathcal{S}^k, \mathcal{T}^k)$ pairs we supply in the prompt. As more pairs are added, the model will not only need to parse more information, but will also need to generate more code. We treat the *Max. Rules per Prompt (MRPP)* as a hyperparameter and vary it from 1 to 7. For example, if MRPP is 5, we will prompt the model twice: once with a specification for 5 rules and again with a specification for 2 rules (one prompt will have fewer rules if the number of mappings is not divisible by MRPP). Rules are uniformly sampled without replacement, and as we add rules, we remove overlapping relations.

We use the same LLM here (Meta-Llama-3.1-70B-Instruct-GPTQ-INT4). For each MRPP, we prompt the model 20 times using different seeds, controlling the order in which relations and attributes within those relations are presented and which data values are sampled. Unlike the previous section, we do not combine the outputs of these different models, but rather use this as a way to get a more robust measurement of performance that is not dependent on a single representation of the input. As discussed, combining these outputs is a viable technique for producing better programs, but it is saved for future works. We report 95% confidence intervals for our metrics.

Figure 3 shows the performance averaged over seeds for each setting for MRPP. Though we see a decrease in performance overall, it is most drastic for recall, implying that the model omits parts the rules whenever more than one rule is included in the prompt. Table 4 gives us an idea of how the complexity of the input and output grows as more rules are added.

5 Challenge: Reducing Poor-Quality Mappings

In Section 3, we proposed methods for filtering the candidate set based on sampling multiple outputs from an LLM. Furthermore, we

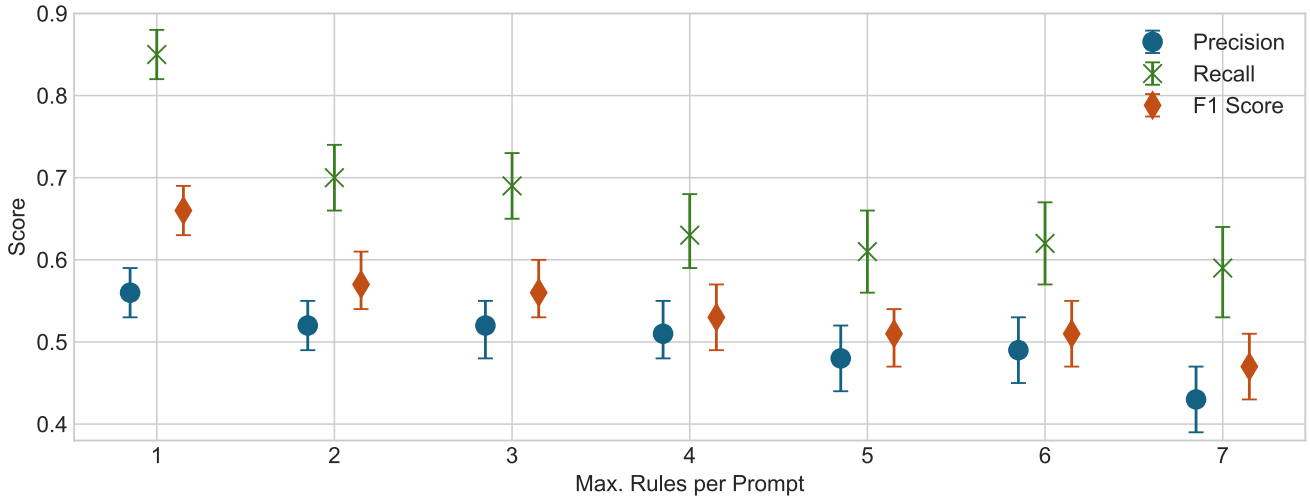

Figure 3: Average precision, recall, and F1-Score plotted against the maximum rules per prompt (MRPP). Error bars represent a 95% confidence interval

empirically showed how such methods can improve the candidate set of mappings. However, sample-based filtering will not eliminate false positives if they consistently appear in the output. Thus, it is worth considering other filtering strategies that can complement these sample-based methods. We propose additional methods for filtering the candidate set based on markers of mapping quality.

5.1 Constraint-Based Filtering

Mappings must be consistent with both the explicit and implicit (semantic) constraints of the global schema. Explicit constraints include those stored in the schema and are often enforced by database management systems. For example, mappings must populate required attributes and respect attribute data types. Moreover, mappings must respect semantic constraints, which are often not enforced by the database, but are still generally followed by its users. Some semantic constraints are commonsense regardless of the domain. For example, each row within a relation must have some information content. This constraint can be violated when a relation has an arbitrary primary key, which is a common database design practice. This can create situations where the primary key is populated but all of the attributes of the row are NULL, functionally producing a row with no real information content.

Another source of semantic constraints are business rules, which are often documented (likely as an entity-relationship diagram) as part of the global schema's design. Some semantic constraints are commonsense given the domain (e.g., each child must have exactly two biological parents) while others may be esoteric and purely process-driven. Regardless, many such constraints are often not stored in the schema. Instead, they are very likely enforced by the end-user when they inspect the final mapping set. In fact, the existence of latent constraints is one important reason for why humans *must* validate candidate rules.

Specifying such constraints could be valuable in eliminating unfavorable mappings from the candidate set. Further, since such constraints specify how data should be organized within the target schema, theoretically, they could be specified once and then be used to produce higher-quality candidate sets for *all sources*, the very definition of a scalable technique. Of course, it is likely that even these constraints will need to be edited as the target itself evolves.

Using semantic constraints is a promising technique for enabling scalable data integration systems, but there are important questions: how would one explicitly define these constraints and how could they be used in practice? It would tedious and difficult to write out and manage all of these underlying constraints, so a more practical approach would be to derive them from other sources. For example, if we already have data in our target, we can use techniques to derive constraints from that data. Further, some such constraints could be learned through user preferences over candidate mappings.

5.2 Model-Based Filtering

Users may not want to semantic constraints into logical statements as it, admittedly, may require significant overhead. Alternatively, one can leverage models of what consistent data "looks like". When adding new sources or maintaining existing ones, we can materialize the data (i.e., run the candidate mapping) and check the quality of the instance.

Assuming that the current instance data in the global up to this point is a good representation of "high-quality data." We can derive characteristics of that data that end up being indicators of quality. In some way, building a model that, through unsupervised learning, constructs its own non-logical black-box constraints of data quality based on the current instance. Reward models could essentially measure the similarity of of existing data and new data (proposed data from candidate mapping)–how well do they "mix".

Max. Rules/Prmpt	Input	Output	Total	Reduction
1	27577	8135	35712	–
2	19204	5282	24486	1.46
3	16635	4633	21268	1.68
4	14087	3777	17864	2.00
5	14083	4068	18151	1.97
6	13884	4006	17890	2.00
7	11259	2639	13898	2.57

Table 5: How chunking affects the number of tokens processed. "Reduction" specifies efficiency relative the first row (i.e., using a separate prompt for each rule)

Of course, this raises questions of how we might prevent the model from becoming biased towards our current instance. For example, if our current instance only contains publications from VLDB, the model might associate publication[title]="VLDB" as an important indicator of quality, and conversely, would assume that instances where, for some publications, publication[title]!="VLDB" indicates a poor mapping.

6 Challenge: Efficient Prompting

A major challenge in LLM-based schema mapping is its high computational cost: for large datasets, many tokens need to be processed. Though computation is generally cheaper than human attention, it is necessary to consider the trade-off between performance and computational costs. imprecisely speaking, techniques should produced the desired results without excessive computation. This is especially important for those who must pay a third party for computational resources. We discuss three strategies for reducing computational costs (i.e., tokens processed).

Reducing Unnecessary Comparisons (Don't ask LLM trivial things). One way to reduce unnecessary comparisons is through data type prefiltering. By categorizing attributes into broad types—such as Numeric, Text, Date/Time, and Boolean—we can reduce the number of source attributes that need to be compared to a target attribute. In an N-1 matching setup, where each prompt compares one target attribute to multiple source attributes, prefiltering ensures that only source attributes with the same data type as the target attribute are included in the comparison. This reduces the pool of source attributes from N to a smaller subset, k, improving both the efficiency and accuracy of the model by eliminating irrelevant comparisons. Another approach is to use the results from the first round of predictions to assess confidence. If the top match has a much higher score than the others, there may be no need for additional rounds of comparison, further reducing the number of calls. By refining the selection process based on confidence and narrowing down the pool of candidate pairs, we can make the schema matching process more efficient and scalable.

Efficient Chunking. As discussed, one tunable parameter is the amount of work induced (i.e., code written) by each prompt. We observed in section 4 that the performance of our baseline approach does worsen as we ask it to generate more rules. However, as shown in Table 5, we also observe that the total number of tokens processed (input/output) is drastically reduced when we have the LLM produce two rules instead of one. Further, the total tokens processed is reduced by more than 50% when we ask the model to generate the full mapping within one prompt.

The non-linear nature of this reduction implies that we can be more efficient with our techniques while also maintaining good performance–basically, the problem is more nuanced than simply reducing the number of prompts. One reason for this non-linear relationship is that there is a constant overhead that must be paid with each prompt; namely, the static text which sets up our specific task for the LLM. The other major reason has to do with high rule overlap, where underlying rules share many source and target relations. Chunking two high-overlap rules together in one prompt has the affect of completing more work while adding relatively little to the input (i.e., the very few relations that do not appear in both rules). In other words, some rules can be chunked together without adding significant complexity (i.e., number of tokens) to either the input or the output rule. A simple technique, then, would be to chunk as many rules together under different prompts such that their overlap is high.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of databases*. Vol. 8. Addison-Wesley Reading.

[2] Menatalla Abououf, Shakti Singh, Hadi Otrok, Rabeb Mizouni, and Anis Ouali. 2019. Gale-Shapley Matching Game Selection—A Framework for User Satisfaction. *IEEE Access* 7 (2019), 3694–3703. doi:10.1109/ACCESS.2018.2888696

[3] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language models enable simple systems for generating structured views of heterogeneous data lakes. *arXiv preprint arXiv:2304.09433* (2023).

[4] Ted T. Ashburn and Karl B. Thor. 2004. Drug repositioning: identifying and developing new uses for existing drugs. *Nature Reviews Drug Discovery* 3, 8 (2004), 673–683.

[5] David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. 2005. Schema and ontology matching with COMA++. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data* (Baltimore, Maryland) *(SIGMOD '05)*. Association for Computing Machinery, New York, NY, USA, 906–908. doi:10.1145/1066157.1066283

[6] Angela Bonifati, Giansalvatore Mecca, Alessandro Pappalardo, Salvatore Raunich, and Gianvito Summa. 2008. Schema mapping verification: the spicy way. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*. 85–96.

[7] Christopher Buss, Jasmin Mosavi, Mikhail Tokarev, Arash Termehchy, David Maier, and Stefan Lee. 2023. Generating Data Augmentation Queries Using Large Language Models.. In *VLDB Workshops*.

[8] Christopher Buss, Jasmin Mousavi, Mikhail Tokarev, Arash Termehchy, David Maier, and Stefan Lee. 2023. Effective Entity Augmentation By Querying External Data Sources. *Proceedings of the VLDB Endowment* 16, 11 (2023), 3404–3417.

[9] Xinyun Chen, Renat Aksitov, Uri Alon, Jie Ren, Kefan Xiao, Pengcheng Yin, Sushant Prakash, Charles Sutton, Xuezhi Wang, and Denny Zhou. 2023. Universal Self-Consistency for Large Language Model Generation. arXiv:2311.17311 [cs.CL] https://arxiv.org/abs/2311.17311

[10] Hong-Hai Do and Erhard Rahm. 2002. Chapter 53 - COMA — A system for flexible combination of schema matching approaches. In *VLDB '02: Proceedings of the 28th International Conference on Very Large Databases*, Philip A. Bernstein, Yannis E. Ioannidis, Raghu Ramakrishnan, and Dimitris Papadias (Eds.). Morgan Kaufmann, San Francisco, 610–621. doi:10.1016/B978-155860869-6/50060-3

[11] AnHai Doan, Alon Halevy, and Zachary Ives. 2012. *Principles of Data Integration* (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[12] Ronald Fagin, Phokion G Kolaitis, Renée J Miller, and Lucian Popa. 2005. Data exchange: semantics and query answering. *Theoretical Computer Science* 336, 1 (2005), 89–124.

[13] National Science Foundation and National Institutes of Health. 2021. Smart Health and Biomedical Research in the Era of Artificial Intelligence and Advanced Data Science (SCH). https://www.nsf.gov/pubs/2021/nsf21530/nsf21530.htm

[14] Behzad Golshan, Alon Y. Halevy, George A. Mihaila, and Wang-Chiew Tan. 2017. Data Integration: After the Teenage Years. In *PODS*.

[15] Zezhou Huang, Jia Guo, and Eugene Wu. 2024. Transform Table to Database Using Large Language Models. *Proceedings of the VLDB Endowment. ISSN* 2150 (2024), 8097.

[16] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large Language Models are Zero-Shot Reasoners. arXiv:2205.11916 [cs.CL] https://arxiv.org/abs/2205.11916

[17] Christos Koutras, George Siachamis, Andra Ionescu, Kyriakos Psarakis, Jerry Brons, Marios Fragkoulis, Christoph Lofi, Angela Bonifati, and Asterios Katsifodimos. 2021. Valentine: Evaluating Matching Techniques for Dataset Discovery. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. 468–479. doi:10.1109/ICDE51399.2021.00047

[18] Sebastian Kruse, Paolo Papotti, and Felix Naumann. 2015. Estimating Data Integration and Cleaning Effort.. In *EDBT*. 61–72.

[19] Maurizio Lenzerini. 2002. Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 233–246.

[20] Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. 2024. A Survey of NL2SQL with Large Language Models: Where are we, and where are we going? *arXiv preprint arXiv:2408.05109* (2024).

[21] Xuanqing Liu, Runhui Wang, Yang Song, and Luyang Kong. 2024. GRAM: Generative Retrieval Augmented Matching of Data Schemas in the Context of Data Security. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Barcelona, Spain) *(KDD '24)*. Association for Computing Machinery, New York, NY, USA, 5476–5486. doi:10.1145/3637528.3671602

[22] Yurong Liu, Aecio Santos, Eduardo HM Pena, Roque Lopez, Eden Wu, and Juliana Freire. [n. d.]. Enhancing Biomedical Schema Matching with LLM-based Training Data Generation. In *NeurIPS 2024 Third Table Representation Learning Workshop*.

[23] Sabine Massmann, Salvatore Raunich, David Aumüller, Patrick Arnold, Erhard Rahm, et al. 2011. Evolution of the COMA match system. *Ontology Matching* 49 (2011), 49–60.

[24] Renée J. Miller, Daniel Fisla, Mary Huang, David Kymlicka, Fei Ku, and Vivian Lee. 2001. The Amalgam Schema and Data Integration Test Suite. (2001). www.cs.toronto.edu/ miller/amalgam.

[25] Avanika Narayan, Ines Chami, Laurel Orr, and Christopher Ré. [n. d.]. Can Foundation Models Wrangle Your Data? ([n. d.]).

[26] Marcel Parciak, Brecht Vandevoort, Frank Neven, Liesbet M Peeters, and Stijn Vansummeren. [n. d.]. Schema Matching with Large Language Models: an Experimental Study. *Proceedings of the VLDB Endowment. ISSN* 2150 ([n. d.]), 8097.

[27] Marcel Parciak, Brecht Vandevoort, Frank Neven, Liesbet M. Peeters, and Stijn Vansummeren. 2024. Schema Matching with Large Language Models: an Experimental Study. arXiv:2407.11852 [cs.DB] https://arxiv.org/abs/2407.11852

[28] Lucian Popa, Yannis Velegrakis, Renee J Miller, Mauricio A Hernandez, and Ronald Fagin. 2002. Translating web data. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 598–609.

[29] Nabeel Seedat and Mihaela van der Schaar. 2024. Matchmaker: Self-Improving Compositional LLM Programs for Table Schema Matching. In *NeurIPS 2024 Third Table Representation Learning Workshop*. https://openreview.net/forum?id=KCklcYUlLb

[30] Eitam Sheetrit, Menachem Brief, Moshik Mishaeli, and Oren Elisha. 2024. ReMatch: Retrieval Enhanced Schema Matching with LLMs. arXiv:2403.01567 [cs.DB] https://arxiv.org/abs/2403.01567

[31] Ananya Singha, José Cambronero, Sumit Gulwani, Vu Le, and Chris Parnin. 2023. Tabular Representation, Noisy Operators, and Impacts on Table Structure Understanding Tasks in LLMs. arXiv:2310.10358 [cs.CL] https://arxiv.org/abs/2310.10358

[32] Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table Meets LLM: Can Large Language Models Understand Structured Table Data? A Benchmark and Empirical Study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining* (Merida, Mexico) *(WSDM '24)*. Association for Computing Machinery, New York, NY, USA, 645–654. doi:10.1145/3616855.3635752

[33] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. arXiv:2203.11171 [cs.CL] https://arxiv.org/abs/2203.11171

[34] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. 2023. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. arXiv:2302.11382 [cs.SE] https://arxiv.org/abs/2302.11382

[35] E. C. Wood, Amy K. Glen, Lindsey G. Kvarfordt, Finn Womack, Liliana Acevedo, Timothy S. Yoon, Chunyu Ma, Veronica Flores, Meghamala Sinha, Yodsawalai Chodpathumwan, Arash Termehchy, Jared C. Roach, Luis Mendoza, Andrew S. Hoffman, Eric W. Deutsch, David Koslicki, and Stephen A. Ramsey. 2021. RTX-KG2: a system for building a semantically standardized knowledge graph for translational biomedicine. *bioRxiv* (2021). https://www.biorxiv.org/content/early/2021/11/01/2021.10.17.464747

[36] Yongqin Xu, Huan Li, Ke Chen, and Lidan Shou. 2024. KcMF: A Knowledge-compliant Framework for Schema and Entity Matching with Fine-tuning-free LLMs. arXiv:2410.12480 [cs.CL] https://arxiv.org/abs/2410.12480

[37] Yifan Zeng, Ojas Tendolkar, Raymond Baartmans, Qingyun Wu, Lizhong Chen, and Huazheng Wang. 2024. LLM-RankFusion: Mitigating Intrinsic Inconsistency in LLM-based Ranking. arXiv:2406.00231 [cs.IR] https://arxiv.org/abs/2406.00231

[38] Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. 2024. Large Language Models as Data Preprocessors. arXiv:2308.16361 [cs.AI] https://arxiv.org/abs/2308.16361

[39] Jing Zhang, Bonggun Shin, Jinho D Choi, and Joyce C Ho. 2021. 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*. Springer, 260–274.

A Related Work in Schema Alignment

Recent studies have explored improving schema matching with large language models (LLMs) by incorporating schema metadata, natural language descriptions, and various techniques to optimize candidate matches and reduce false positives. One common strategy involves *pre-prompt filtering* of matching candidates based on semantic similarity measures [21, 29, 30, 36**?**], which simplifies the LLM's task by narrowing down the set of candidates. Another strategy is *post-prompt filtering*, where candidate alignments are refined through techniques such as ensembling multiple prompts [27, 36] or prompting LLMs to assign confidence scores to candidates [29]. In the following, we summarize key methods from recent papers in this domain, with Tables 6 and 7 providing an overview of the techniques they employ and the metadata incorporated into their prompts.

Parciak et al. [27] present an experimental approach to schema matching that does not rely on pre-prompt filtering. They explore several prompting configurations, including 1-to-1[5], 1-to-M[6], N-to-1, and N-to-M setups. Each attribute is provided along with their name, description, and relational context (relation name and description). The authors ask the model to reason over this information before classifying each alignment pair into one of three different JSON-formated lists: "yes," "no," or "unknown." To reduce hallucinations, the authors employ a majority voting technique by prompting the LLM three times and selecting the most frequent alignments. They find 1-to-M and N-to-1 prompts to be most effective, and they recommend applying the union to the output matches of both configurations to improve recall. However, their evaluation reveals a notable gap between recall and precision, suggesting that further refinement of the approach is necessary to enhance matching accuracy.

ReMatch [30] introduces an efficient approach to LLM-based schema alignment by employing semantic similarity to significantly reduce the search space before prompting the LLM. Instead of evaluating all possible candidate alignments, this method uses embeddings generated from schema information and textual descriptions to pre-filter target tables, narrowing the candidates to only the most promising options. These embeddings identify the top J candidate target tables for each source attribute, which are then sent to the LLM for further refinement. The LLM subsequently selects the top K most relevant target attributes for alignment with each source attribute. This strategy, while effective in improving efficiency and alignment quality, relies on careful calibration of the parameters J and K to maintain robust performance across various domains.

Liu et al. [**?**] apply an initial filtering over all alignment pairs before prompting the LLM for a decision. However, their approach relies on an embedding model that is trained on data synthesized by the LLM itself. They introduce a contrastive learning framework to generate column embeddings that capture important semantic differences. The schema matching process is performed in two stages: first, they narrow down the pool of potential matches by identifying those most similar to the queried attribute. Then, they pass the top candidates along with the source attribute to the LLM to perform the final alignment. This filtering approach introduces additional complexity and may lead to biases, as it depends on the quality and domain-specific capabilities of the LLM used for data augmentation.

GRAM [21] is a schema matching method that incorporates a filtering step inspired by named entity recognition (NER). They categorize attributes into predefined object types (e.g., "zip codes" vs. "cities") and train a BERT-base model using synthesized data to classify attributes accordingly. After clustering attributes by these categories, they limit the candidate matches passed to the LLM based on these clusters and their respective data types. To further enhance performance, the authors utilize in-context learning, dynamically generating examples based on the similarity of the source attribute and the object category of the examples. Although this approach shows promise, there are concerns about the quality of the training data. The method's heavy reliance on predefined object types requires deep domain knowledge to define these categories accurately, which may be difficult when applying the method to new domains. Moreover, the data synthesis process depends on random generation, publicly available datasets, and LLM-generated examples, which may not always provide high-quality, domain-specific training data.

MatchMaker [29] is a schema matching framework that narrows candidates into two groups: *semantically similar candidates*, generated using column-wise similarities from a pre-trained embedding model, and *reasoning-based candidates*, derived by prompting an LLM to suggest matches based on schema descriptions. These matches are further refined using an LLM with chain-of-thought reasoning to identify the most relevant alignments. For validation, the framework employs confidence scoring, where the LLM assigns confidence levels to alignments and includes a "None of the above" option to avoid forced matches. To enhance its performance, the framework self-optimizes by creating synthetic in-context examples from its outputs. These examples are generated by running the model on unlabeled data. The authors then prompt an LLM using chain-of-thought reasoning to rate input-output relevance for each example, keeping the highest-scored examples as in-context examples. The authors observe that *reasoning-based candidates* often outperform those from *semantic similarity* alone. We argue that relying on synthetic in-context examples could introduce risks of bias and error propagation, especially if comprehensive validation across domains is not conducted.

KcMF [36] is a schema matching framework that combines structured reasoning with knowledge retrieval techniques. The framework begins by using if-then-else pseudo-code rules to guide the alignment process, which requires manual configuration. Knowledge is represented in two forms: *Dataset as Knowledge (DaK)* and *Examples as Knowledge (EaK). DaK* employs metadata to describe database objects such as tables, while *EaK* utilizes language models to extract domain-specific keywords from metadata pairs and provides examples from external knowledge sources, including Snomed-CT, Wikidata, and Wikipedia. To improve alignment accuracy, KcMF incorporates summaries of schema descriptions and relevant knowledge entries into the prompts. The framework enhances reasoning through a k-shot in-context learning approach that follows the pseudo-code to generate reasoning steps and answers. For managing complexity, KcMF generates separate prompts

[5] In 1-to-1 prompts, the LLM evaluates candidate alignments one-at-a-time
[6] In 1-to-M prompts, the LLM produces alignment pairs for one source attribute and all target attributes at once

Table 6: Summary of the methods from recent papers. "# Target Table" specifies the number of target tables supported. "Match" describes the matching strategies, where 1:m means a single attribute from one schema can align with m attributes in another schema. The table also highlights the use of synthetic data (Synth.), chain of thought (CoT), incontext learning (ICL), and the prompting method, where N-M denotes that each prompt introduces N source attributes and M target attributes. Finally, the "Pre-filtering" and "Post-filtering" columns indicate whether candidate alignments are filtered before or after prompting the LLM.

Paper	# Target Table	Match	Synth.	Prompt	CoT	ICL	Pre-filtering	Post-filtering
Parciak et al.[27]	1	1:1	✗	N-1,1-M	✓	✗	✗	✓
ReMatch[30]	>1	1:m	✗	1-M	✗	✗	✓	✗
liu et al.[?]	1	1:m	✓	1-M	✗	✗	✓	✗
GRAM[21]	1	1:1	✓	1-M	✗	✓	✓	✗
MatchMaker[29]	>1	1:m	✓	1-M	✓	✓	✓	✓
KcMF[36]	1	1:m	✓	1-1	✓	✓	✓	✓

Table 7: Summary of the metadata used by each method in addition to attribute names. The table highlights the types of additional schema information utilized, including primary key/foreign key relationships, attribute and table descriptions, data types, data values, and the model used.

Paper	Primary key/Foreign key	Attribute description	Data type	Table description	Data value	Model
Parciak et al. [27]	✗	✓	✗	✓	✗	GPT-4
ReMatch [27]	✓	✓	✓	✓	✗	GPT-4
Liu et al. [21]	✗	✗	✗	✗	✓	GPT-4
GRAM [29]	✗	✗	✓	✗	✓	FLAN-T5
MatchMaker [30]	✗	✓	✓	✓	✗	GPT-4
KcMF [36]	✗	✓	✗	✓	✓	GPT-4

for each knowledge source and combines results through majority voting. However, the framework's reliance on manual selection of domain knowledge bases complicates implementation, and its focus on pairwise attribute matching makes it computationally expensive, limiting its scalability for larger database mappings.

B Prompt Templates

Similar to [27], we use an N-1 prompt template consisting of four sections: *Introduction*, *Source Schema Information*, *Target Schema Information*, and *Task Description*. Each prompt contains N attributes from the source schema and a single attribute from the target schema, where the source and target schema details are serialized in JSON format to improve LLM performance in structural table understanding tasks [31, 32]. For both the *Source Schema Information* and *Target Schema Information* sections, we use all the hints available given the dataset. The target attribute information is serialized in JSON format following the template shown in Figure 4. For the source information, we include the details for all attributes in the key *"columns"*.

In the *Introduction* section, we use the Persona Pattern [34] to encourage the LLM to act as a schema matcher. In the *Task Description* section, we include the phrase "Let's think step by step" to help the LLM produce clear, step-by-step reasoning [16]. We also use the Output Automater pattern shown in Equation 2 to guide the model towards generating its output in JSON format, allowing easier processing [34].

{"matches": ["<source attribute>, target_attribute", ...]} (2)

```
{
  "relation name": "relation_name",
  "relation description": "relation_description",
  "column": {
    "name": "attribute_name",
    "type": "data_type",
    "description": "attribute_description",
    "sample data instances": "random_unique_samples"
  }
}
```

Figure 4: JSON Serialization of the target attribute, with all hints available.

Where *target_attribute* is the queried target attribute in the prompt. We also instruct the LLM not to mention the the source attributes for which there is not enough information to conclude whether they align with the target attribute. Additionally, if there is no source attribute matching the target attribute, we ask the LLM to report `"None, target_attribute"` instead. For an overview of our N-1 prompt see Figure 5.

C Confidence Scores computed using LLM Logits

In this approach, we prompt the LLM to identify the best match among the candidates for each attribute as shown in Figure 6. Then, we use the LLM logits to calculate the confidence score for each candidate. The process is broken down into several steps:

Act as a schema matcher for relational schemas. Your task is to create semantic matches that specify how the elements of the source schema and the target schema semantically correspond to one another. I will first provide the information of a single relation from the source schema, including the relation name and description, as well as the name, type, description and sample data instances of all its attributes. Next, I will provide the same information for a single relation and a single attribute from the target schema.
The information about the relation from the source schema is as follows:
source_schema_serialization
The information about the relation from the target schema is as follows:
target_attribute_serialization
Explain which of the target attributes semantically match to *target_attribute* from *target_relation*. Let's work this out step by step to make sure we get it correct. After your explanation, give a final decision formatted like this: '{"matches": ["<source attribute>, target_attribute", ...]}'. Do not mention an attribute if there is not enough information to decide. If there is no source attribute matching the target attribute, return "None, *target_attribute*".

Figure 5: N-1 Prompt

Act as a schema matching expert. Given the attribute from the source schema, which of the following target attributes is the best match? Provide only the attribute name of the best match.
Both the input query and the schema options are formatted as 'attribute name (data type) : description'.
Question:
Target attributes:
1. target_attribute1(data type1): description1
2. target_attribute2(data type2): description2
3. target_attribute3(data type3): description3
Input query: source_attribute_name (data type): description
Answer:

Figure 6: Best Match Prompt

The model generates raw scores (logits) for each word option. Let the logits be represented as:

$$\text{logits} \in \mathbb{R}^{(\text{batch size} \times \text{sequence length} \times \text{vocab size})}$$

We apply the Softmax function to transform these logits into token probabilities. The Softmax function converts the logits into probabilities that sum to 1 across all tokens:

$$P(t_i) = \frac{e^{\text{logit}(t_i)}}{\sum_j e^{\text{logit}(t_j)}}$$

where $P(t_i)$ is the probability of token t_i, and the summation is over all tokens in the vocabulary.

Next, we break each candidate into tokens. The total confidence score for a candidate is the product of the probabilities of all its tokens:

$$P(\text{candidate}) = P(t_1) \times P(t_2) \times \cdots \times P(t_n)$$

where $t_1, t_2, \ldots, t_n$ are the tokens of the candidate. However, multiplying small values (probabilities) can lead to numerical instability due to underflow. To stabilize this, we use the log-sum approach.

Instead of multiplying probabilities, we sum the log-probabilities of each token:

$$\text{log-sum} = \sum_{i=1}^{n} \log P(t_i)$$

Then, we exponentiate the result to obtain the final probability:

$$P(\text{candidate}) = e^{\text{log-sum}}$$

Finally, we normalize the confidence scores across all candidates. Let $P(c)$ be the unnormalized confidence score for candidate c. The normalized confidence score for candidate c is:

$$P_{\text{normalized}}(c) = \frac{P(c)}{\sum_j P(c_j)}$$

where the denominator is the sum of the unnormalized scores for all candidates.

This approach gives us the LLM's confidence in each candidate's match with the input query.

D Many-to-Many Stable Matching Algorithm

Stable matching is a fundamental problem in computer science. It involves two sets of entities, A and B, where each element in A ranks the elements in B based on preference [7] , and vice versa. The goal is to pair elements from A and B such that no unmatched pair would prefer each other over their current matches [**?**].

The Stable Marriage Problem addresses one-to-one matching, where each participant from one set is matched to a single participant from the other set [**?**]. The Hospitals-Residents Problem extends this to one-to-many matching, allowing entities in one set (e.g., hospitals) to match with multiple entities from the other set (e.g., residents) up to a capacity [**?**]. A many-to-many matching example is allocating multiple workers to multiple tasks, where each worker has preferences for various tasks, and each task can be assigned to multiple workers [2]. For the schema matching problem, where attributes in two schemas can have overlapping relationships, we adopt a many-to-many stable matching approach. This ensures that the matches are consistent in both directions while maintaining stability.

[7] If attribute b prefers a over a', it means a is a better match for b than a'.

The steps of the many-to-many stable matching algorithm, presented in Algorithm 1, are described below:

- **Step 1 - Lines 1 to 15:** The input consists of two sets of attributes, one from Schema A and one from Schema B, along with ranked preferences for each attribute in A over certain attributes in B, and vice versa. Initially, the matching set M is empty, and $acceptable\_match$ is initialized to store the acceptable matches for each attribute. We set $k = \infty$ to obtain all possible stable matches.
- **Step 2 - Lines 16 to 31:** In the Initial Matching Phase, each attribute in schema A proposes [8] to its most preferred match in schema B. Some attributes may not be proposed to at all, while others may receive multiple proposals. Attributes that do not receive any proposals will remain unmatched. If an attribute receives a single proposal, it will be paired with the proposing attribute, as long as the proposer is in its $acceptable\_match$. If an attribute proposes to a match already paired with another attribute, the match will evaluate whether it prefers the new proposal over its current pair. If the new proposal is preferred, the current match is replaced, and the previous pair is left unmatched. This process continues until all attributes in A have proposed to the matches they prefer, either resulting in a stable match or leaving certain attributes unpaired.
- **Step 3 - Lines 32 to 37:** To extend the algorithm for many-to-many matching, the selection process is performed over several rounds. Step 2 outlines the procedure for a single round. In subsequent rounds, the preference lists and acceptable matches are updated, and paired attributes are removed from each other's set of acceptable matches.
- **Termination Condition:** The process continues until no free attributes can make a proposal, or a maximum number of rounds is reached. If the algorithm terminates at $r = k$, M includes up to k stable matches for each attribute. The final set M contains the stable pairs of attributes matched between A and B, ensuring a stable many-to-many schema match.

Algorithm 1 Many-to-Many Stable Matching Algorithm for Schema Matching

1: **Input:** Sets of attributes from schema A and schema B, with preference lists A_pref and B_pref.
2: **Output:** M, the set of stable matches between attributes in A and B.
3: Initialize $M = \emptyset$, $acceptable\_match = \{\}$, $k = \infty$
4: **for** each attribute $a \in A$ **do**
5: Set $acceptable\_match(a) = [\,]$
6: **for** each b in $A\_pref(a)$ **do**
7: Add b to $acceptable\_match(a)$
8: **end for**
9: **end for**
10: **for** each attribute $b \in B$ **do**
11: Set $acceptable\_match(b) = [\,]$
12: **for** each a in $B\_pref(b)$ **do**
13: Add a to $acceptable\_match(b)$
14: **end for**
15: **end for**
16: $r = 1$
17: **while** $r \leq k$ **do**
18: Initialize all $a \in A$ and $b \in B$ as free
19: **Set** *match_made* = **false**
20: **while** $\exists a \in A$ free with a remaining preference in $A\_pref(a)$ **do**
21: a proposes to b, its most preferred attribute
22: **if** $a \in acceptable\_match(b)$ **then**
23: **if** b is free **then**
24: Add (a, b) to M
25: *match_made* = **true**
26: **else if** b prefers a over its current match **then**
27: Replace b's current match in M with (a, b)
28: *match_made* = **true**
29: **end if**
30: **end if**
31: **end while**
32: Update preference lists and acceptable matches for all $a \in A$ and $b \in B$
33: **if not** *match_made* **then**
34: **break**
35: **end if**
36: $r = r + 1$
37: **end while**
38: **return** M

E Evaluation

For our experiment, we configure the model with a temperature of 0.4, a top-p value of 0.95, and an output limit of 800 tokens. The temperature is set to 0.4 to encourage deterministic behavior while still maintaining some diversity in the output. The top-p value of 0.95 ensures that the model considers only the most probable tokens until their cumulative probability reaches 95%, which helps avoid the inclusion of less likely or irrelevant words, making the generated responses more reliable.

E.1 Dataset

MIMIC. The MIMIC dataset contains alignments between two real-world electronic healthcare record databases: MIMIC-III and OMOP. The ground truth alignments, natural language descriptions, and schemas were taken from the ReMatch paper [30]. MIMIC-III[9] is a publicly accessible[10] database containing deidentified health data from critical care patients, while OMOP[11] is a Common Data Model (CDM) meant to provide a standard structure for MIMIC data. Since it is a CDM, OMOP functions like a global schema meant to house data from other sources and, as such, does not contain its own data.

To populate OMOP with data values, we used sample data from MIMIC-IV *formatted under the OMOP model*[12]. The MIMIC-IV sample data is different enough from that of MIMIC-III that it avoids overlap in records. At the same time, it is similar enough that it ensures conceptual overlap between the two databases. Though somewhat rare, we were able to repair most discrepancies between schemas due to using different versions of MIMIC sample data.

[8] If attribute b proposes to attribute a, it means that a is on b's preference list and that b would like to match with a.
[9] https://mimic.mit.edu
[10] Though technically "freely accessible", MIMIC-III requires authorization to obtain its full dataset. At the time of evaluation, we did not have access, so we used a more "freely accessible" sample that required no authorization.
[11] https://ohdsi.github.io/CommonDataModel/
[12] https://physionet.org/content/mimic-iv-demo-omop/0.9/

Even when alignments are defined in the ground truth, corresponding columns may lack sufficient sample data values due to privacy restrictions or omissions in the MIMIC datasets. For instance, the NOTE_EVENTS table, which contains sensitive information, is excluded from these samples. Additionally, while nearly all alignments are 1:1 at the table-to-table level, not all columns have accompanying descriptions, which introduces further complexities in schema alignment tasks.

Synthea [?] Synthea is an open-source dataset that generates realistic synthetic healthcare records. It has been used in the OMOP benchmark, where parts of its schema are mapped to the OMOP CDM. Synthea has been widely used in past schema matching studies [29, 30, 36] as a realistic and challenging benchmark.

Dataset	#Pairs	#Source attr	#Target attr	#Matches
MIMIC	26	268	203	155
Synthea	12	101	134	105

Table 8: Dataset statistics for MIMIC, Synthea and BIRD. #Pairs is the number of table pairs for matching, #Source attr and #Target attr are the number of attributes in the source and target schemas, respectively, and #Matches is the number of matching attributes.

E.2 Baseline

To evaluate the effectiveness of our schema-matching methods, we compare them against two baselines that, like our approach, require no training data.

Classic Schema Matching Baseline. For our classic schema matching baseline, we use COMA [10], which is recognized as one of the most efficient schema-matching methods according to the Valentine paper [17]. COMA has been refined through multiple iterations, including COMA++ [5] and COMA 3.0 [23]. COMA is notable for its flexibility in combining multiple matching algorithms, offering both schema-based and instance-based matching capabilities. The schema-based version considers only schematic information when determining matches, whereas the instance-based version considers both schematic information and data values. We evaluate both versions of COMA where *COMA (Sch.)* is the schema-based version and *COMA (Inst.)* is the instance-based version. We use Valentine's implementation of COMA, available as a Python wrapper around COMA 3.0 Community Edition.[13]

Language Model Baselines. We adopt the N-1 prompting method proposed by Parciak et al. [27] due to its simplicity and effectiveness, as it does not require synthetic data or additional model training. We also adopt MatchMaker [29], which includes synthetic in-context examples in the prompt. For a fair comparison, we skip the prefiltering step from MatchMaker's pipeline since our approach does not include such a step. We implement both methods following the detailed instructions and prompts provided in their papers.

E.3 Hint Comparison Tables

The availability of specific hints depends on the source and target tables. Following our assumption that both S and T are relations, we are guaranteed to have table names, as well as attribute names and data types. For all other hints, we evaluate both their inclusion and exclusion within the prompt, comparing them against our classic schema matching baseline. The complete results are shown in Tables 11 and 12, and we discuss the impact of each hint in the following paragraphs.

[13] https://github.com/delftdata/valentine

Dataset	Experiment	Precision	Recall	F1
EHR	LLM( )	0.28 ± 0.012	0.81 ± 0.01	0.41 ± 0.01
	LLM(A)	0.35 ± 0.01	0.78 ± 0.03	0.47 ± 0.01
	LLM(T)	0.31 ± 0.01	0.77 ± 0.01	0.43 ± 0.01
	LLM(A, T)	**0.37 ± 0.00**	**0.78 ± 0.01**	**0.49 ± 0.00**
	COMA Sch.	0.14	0.11	0.10
	COMA Inst.	0.21	0.14	0.16
Synthea	LLM( )	0.47 ± 0.02	0.91 ± 0.03	0.60 ± 0.01
	LLM(A)	0.58 ± 0.02	0.92 ± 0.01	0.69 ± 0.01
	LLM(T)	0.49 ± 0.02	0.87 ± 0.02	0.62 ± 0.02
	LLM(A, T)	**0.62 ± 0.02**	**0.91 ± 0.01**	**0.72 ± 0.01**
	COMA Sch.	0.3	0.15	0.19
	COMA Ins.	0.3	0.16	0.2

Table 9: Evaluation of N-1 prompts with varying natural language descriptions. Each experiment includes at least table names, attribute names, and data types. "A" and "T" represent descriptions of attributes and tables, respectively.

Names and Attribute Type. Even with such basic schema information, the LLM-based method outperforms the Instance-based COMA on MIMIC and Synthea. This implies that the LLM has some semantic understanding of the concepts used within the domain. On BIRD, the LLM-based method achieves a much higher recall than *Schema-based COMA* with a relatively modest drop in precision. Compared with MIMIC and Synthea, BIRD contains more overlap in attribute names. The relative gaps in performance between the LLM-based method and COMA indicates that the former is much more apt at comparing attributes at a semantic level, but cannot quite compete with string matching functions when inputs are syntactically similar.

Natural Language Descriptions. For the MIMIC and Synthea datasets, table descriptions slightly enhance precision, potentially influencing precision more than recall. Across all datasets, including attribute descriptions consistently improves performance. This suggests that even in common domains with less specific attribute names, providing descriptions can be very helpful.

Data Values. for MIMIC and Synthea, the inclusion of data values can cause a minor drop in performance. However, for both datasets, the sampling method and the number of data values seem to have little impact on performance, meaning that even seeing just a few values, regardless of how they are sampled, is enough to affect the model. This effect may have to do with the overlap in data values in the dataset.

Given these experiments, there is good evidence that natural language descriptions of attributes are the most helpful of the hints considered. For both data and table descriptions, the effect is a little less clear.

Dataset	Experiment	Precision	Recall	F1
EHR	**LLM( )**	**0.37 ± 0.00**	**0.78 ± 0.01**	**0.49 ± 0.00**
	LLM(10 freq.)	0.37 ± 0.01	0.71 ± 0.01	0.47 ± 0.01
	LLM(10 rand.)	**0.39 ± 0.01**	**0.75 ± 0.04**	**0.49 ± 0.03**
	COMA Sch.	0.14	0.11	0.10
	COMA Inst.	0.21	0.14	0.16
Synthea	**LLM( )**	**0.62 ± 0.02**	**0.91 ± 0.01**	**0.72 ± 0.01**
	LLM(10 freq.)	0.61 ± 0.04	0.89 ± 0.02	0.71 ± 0.03
	LLM(10 rand.)	**0.64 ± 0.05**	**0.89 ± 0.01**	**0.73 ± 0.04**
	COMA Sch.	0.3	0.15	0.19
	COMA Ins.	0.3	0.16	0.2

Table 10: Evaluation of N-1 prompts with varying data values. Each experiment includes table names, attribute types, and natural language descriptions. The sampling method "freq." is most frequent; "rand." is random without duplicates.

In Tables 11 and 12 we evaluate both the inclusion and exclusion of various hints within the prompt. At a minimum, each experiment includes table names along with attribute names and types. "N" indicates the number of data values sampled, and "Smp" indicates the sampling method ("freq." is most frequent; "rand." is random w/o duplicates). "Dsc" indicates natural language descriptions of attributes (A) and tables (T).

Experiment			Metrics		
N	**Smp**	**Dsc**	**Precision**	**Recall**	**F1**
–	–	–	0.28 ± 0.012	0.81 ± 0.01	0.41 ± 0.01
–	–	A	0.35 ± 0.01	0.78 ± 0.03	0.47 ± 0.01
–	–	A+T	**0.37 ± 0.00**	**0.78 ± 0.01**	**0.49 ± 0.00**
–	–	T	0.31 ± 0.01	0.77 ± 0.01	0.43 ± 0.01
5	freq.	–	0.29 ± 0.03	0.69 ± 0.06	0.40 ± 0.03
5	freq.	A	0.36 ± 0.03	0.71 ± 0.02	0.46 ± 0.03
5	freq.	A+T	0.38 ± 0.03	0.72 ± 0.03	0.48 ± 0.03
5	freq.	T	0.35 ± 0.02	0.68 ± 0.04	0.45 ± 0.02
5	rand.	–	0.31 ± 0.02	0.74 ± 0.04	0.42 ± 0.03
5	rand.	A	0.37 ± 0.02	0.72 ± 0.01	0.47 ± 0.01
5	rand.	A+T	0.39 ± 0.01	0.75 ± 0.02	0.50 ± 0.01
5	rand.	T	0.33 ± 0.02	0.68 ± 0.02	0.43 ± 0.02
10	freq.	–	0.28 ± 0.03	0.64 ± 0.03	0.37 ± 0.03
10	freq.	A	0.36 ± 0.01	0.73 ± 0.01	0.47 ± 0.01
10	freq.	A+T	0.37 ± 0.01	0.71 ± 0.01	0.47 ± 0.01
10	freq.	T	0.33 ± 0.02	0.68 ± 0.05	0.42 ± 0.03
10	rand.	–	0.31 ± 0.02	0.73 ± 0.01	0.42 ± 0.02
10	rand.	A	0.37 ± 0.03	0.62 ± 0.52	0.51 ± 0.38
10	rand.	A+T	**0.39 ± 0.01**	**0.75 ± 0.04**	**0.49 ± 0.03**
10	rand.	T	0.33 ± 0.01	0.73 ± 0.01	0.42 ± 0.02
15	freq.	–	0.30 ± 0.01	0.70 ± 0.03	0.41 ± 0.01
15	freq.	A	0.37 ± 0.01	0.71 ± 0.02	0.47 ± 0.02
15	freq.	A+T	0.37 ± 0.01	0.71 ± 0.03	0.48 ± 0.01
15	freq.	T	0.34 ± 0.01	0.70 ± 0.01	0.44 ± 0.01
15	rand.	–	0.30 ± 0.02	0.71 ± 0.04	0.41 ± 0.03
15	rand.	A	0.37 ± 0.01	0.76 ± 0.02	0.49 ± 0.02
15	rand.	A+T	0.37 ± 0.02	0.74 ± 0.01	0.48 ± 0.02
15	rand.	T	0.31 ± 0.01	0.66 ± 0.05	0.41 ± 0.02
COMA Sch.			0.14	0.11	0.10
COMA (Inst.)			0.21	0.14	0.16

Table 11: Evaluation over EHR with varying amount of hints.

Experiment			Metrics		
N	**Smp**	**Dsc**	**Precision**	**Recall**	**F1**
–	–	–	0.47 ± 0.02	0.91 ± 0.03	0.60 ± 0.01
–	–	A	0.58 ± 0.02	0.92 ± 0.01	0.69 ± 0.01
–	–	A+T	**0.62 ± 0.02**	**0.91 ± 0.01**	**0.72 ± 0.01**
–	–	T	0.49 ± 0.02	0.87 ± 0.02	0.62 ± 0.02
5	freq.	–	0.51 ± 0.02	0.87 ± 0.02	0.63 ± 0.02
5	freq.	A	0.60 ± 0.03	0.89 ± 0.02	0.70 ± 0.02
5	freq.	A+T	0.65 ± 0.03	0.87 ± 0.03	0.73 ± 0.03
5	freq.	T	0.58 ± 0.04	0.89 ± 0.02	0.69 ± 0.03
5	rand.	–	0.52 ± 0.01	0.88 ± 0.00	0.64 ± 0.01
5	rand.	A	0.62 ± 0.02	0.90 ± 0.01	0.71 ± 0.02
5	rand.	A+T	0.31 ± 0.02	0.71 ± 0.02	0.42 ± 0.02
5	rand.	T	0.57 ± 0.03	0.89 ± 0.02	0.68 ± 0.03
10	freq.	–	0.53 ± 0.01	0.90 ± 0.03	0.65 ± 0.02
10	freq.	A	0.61 ± 0.00	0.90 ± 0.02	0.70 ± 0.01
10	freq.	A+T	0.61 ± 0.04	0.89 ± 0.02	0.71 ± 0.03
10	freq.	T	0.55 ± 0.03	0.85 ± 0.05	0.65 ± 0.03
10	rand.	–	0.51 ± 0.01	0.89 ± 0.04	0.64 ± 0.02
10	rand.	A	0.51 ± 0.19	0.62 ± 0.52	0.51 ± 0.38
10	rand.	A+T	**0.64 ± 0.05**	**0.89 ± 0.01**	**0.73 ± 0.04**
10	rand.	T	0.58 ± 0.02	0.86 ± 0.02	0.68 ± 0.02
15	freq.	–	0.53 ± 0.03	0.88 ± 0.01	0.64 ± 0.02
15	freq.	A	0.61 ± 0.04	0.90 ± 0.05	0.71 ± 0.04
15	freq.	A+T	0.62 ± 0.04	0.90 ± 0.02	0.72 ± 0.02
15	freq.	T	0.57 ± 0.04	0.85 ± 0.04	0.66 ± 0.02
15	rand.	–	0.54 ± 0.01	0.87 ± 0.01	0.65 ± 0.01
15	rand.	A	0.60 ± 0.01	0.89 ± 0.01	0.70 ± 0.01
15	rand.	A+T	0.62 ± 0.02	0.89 ± 0.02	0.71 ± 0.02
15	rand.	T	0.59 ± 0.01	0.88 ± 0.01	0.69 ± 0.01
COMA (Sch.)			0.14	0.11	0.10
COMA (Inst.)			0.21	0.14	0.16

Table 12: Evaluation over Synthea with varying amount of hints.

E.4 Aggregation Method Evaluation

We evaluate our aggregation method with and without data values. To identify the optimal number of data values, we tested configurations with 5, 10, and 15 data values for each attribute. As shown in Tables 11 and 12, the configuration with 10 randomly sampled data values achieved the best results. Therefore, we focus on this configuration in our reported findings.

In our experiments using the MIMIC and Synthea dataset, the prompts for both of our methods and the approach by Parciak et al. include the attribute name, type, and description, along with the relation name and description. For the BIRD dataset, however, the relation description was unavailable and therefore is excluded from the prompts in both our method and that of Parciak et al.

As shown in Tables 14 and 13, our aggregation method outperforms all baselines. Among the different aggregation strategies, Intersection achieves the highest precision and F1 scores. Union leads to high recall but substantially lower precision, whereas the majority vote offers a better balance between precision and recall. In addition, we observe including data values in the prompt improves precision across all datasets.

Method	Experiment	Precision	Recall	F1
Aggregation	W/O data values	0.37 ± 0.02	0.78 ± 0.03	0.49 ± 0.02
	— Intersection	0.49 ± 0.02	0.71 ± 0.01	0.56 ± 0.02
	— Union	0.30 ± 0.00	0.84 ± 0.02	0.43 ± 0.00
	— Majority	**0.34 ± 0.01**	**0.81 ± 0.02**	**0.47 ± 0.01**
	W/ data values	0.39 ± 0.01	0.75 ± 0.03	0.50 ± 0.02
	— Intersection	0.52 ± 0.03	0.66 ± 0.01	0.56 ± 0.02
	— Union	0.31 ± 0.01	0.83 ± 0.02	0.44 ± 0.01
	— Majority	**0.35 ± 0.01**	**0.79 ± 0.02**	**0.47 ± 0.01**
Baseline	COMA (Inst.)	0.144	0.114	0.103
	COMA (Sch.)	0.205	0.135	0.159
	Par. et al. [27]	**0.333**	**0.195**	**0.216**

Table 13: Evaluation of the Aggregation Method Compared to the Baseline on the MIMIC dataset.

Method	Experiment	Precision	Recall	F1
Aggregation	W/O data values	0.61 ± 0.04	0.90 ± 0.02	0.71 ± 0.03
	— Intersection	0.74 ± 0.02	0.83 ± 0.01	0.77 ± 0.01
	— Union	0.31 ± 0.01	0.83 ± 0.02	0.44 ± 0.01
	— Majority	**0.55 ± 0.01**	**0.95 ± 0.01**	**0.68 ± 0.01**
	W/ data values	0.64 ± 0.03	0.89 ± 0.02	0.73 ± 0.03
	— Intersection	0.75 ± 0.00	0.80 ± 0.03	0.76 ± 0.02
	— Union	0.30 ± 0.00	0.84 ± 0.02	0.43 ± 0.00
	— Majority	**0.57 ± 0.01**	**0.95 ± 0.01**	**0.70 ± 0.01**
Baseline	COMA (Inst.)	0.30	0.15	0.19
	COMA (Sch.)	0.30	0.16	0.20
	Par. et al. [27]	0.53 ± 0.01	0.11 ± 0.03	0.17 ± 0.03

Table 14: Evaluation of the Aggregation Method Compared to the Baseline on the Synthea dataset.

The difference between our aggregation method with majority vote and that of Parciak et al. lies in the output format and schema serialization in the prompt. Most of the shortcomings in their results can be attributed to the model not generating a final decision. Instead, it often stopped with incomplete responses, such as: *The first attribute to consider is* {$source\_attribute$}. *Does* {$source\_attribute$} *semantically match to* {$target\_attribute$}*?*

Even our N-1 prompt with no data values and aggregation outperforms the baseline. It is important to note that Parciak et al. used the GPT-4 model in their paper, while our model for these experiments is significantly smaller. This means our prompt is a better choice for smaller models. We attribute this improvement to the clear and straightforward output format, as well as our JSON schema serialization.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009