Appendix - Matchmaker: Schema Matching with self-improving compositional LLM programs

Table of Contents

A MATCHMAKER ADDITIONAL DETAILS

A.1 MATCHMAKER WITHIN THE CONTEXT OF LLM TABLE REASONING.

There has recently been works on LLMs for table reasoning. We contrast them to Matchmaker along a variety of dimensions below.

Task/Goal: The table reasoning papers tackle a variety of tasks centered around understanding and interacting with tabular data. Some examples include: TabSQLify (Nahid & Rafiei, 2024) and OPENTAB (Kong et al., 2023) focus on table question answering and fact verification, aiming to extract relevant information from tables to answer questions or verify statements. Chain-of-Table (Wang et al., 2023b) and "Large Language Models are Few-Shot Table Reasoners" (Chen, 2023) explore LLMs' capabilities in reasoning over tables for question answering and fact verification tasks. The survey paper "Large Language Model for Table Processing" (Lu et al., 2024) covers a broader range of tasks, including table manipulation, table augmentation, and text-to-SQL conversion, showcasing LLMs' potential in interpreting and manipulating tabular data. In contrast, Matchmaker addresses the task of schema matching, which aims to find correspondences between attributes across different schemas or tables. The goal is to enable data integration by mapping attributes from a source schema to a target schema, considering the structural and semantic differences between them. This task is crucial for creating ML-ready datasets by harmonizing data from diverse sources.

Approach: Table reasoning approaches span prompting LLMs for direct answers (Chen, 2023), program synthesis to generate SQL/code (Nahid & Rafiei, 2024; Kong et al., 2023), iterative table transformation (Wang et al., 2023b), instruction tuning (Lu et al., 2024), and agent-based methods (Lu et al., 2024). Matchmaker proposes a novel self-improving compositional language model program. It leverages LLM reasoning via a pipeline with multiple LLM calls for candidate generation, refinement and confidence scoring. It also self-improves without labeled data via synthetic in-context examples.

Inputs: The table reasoning papers mostly focus on single tables as input along with a question/query. Matchmaker takes as input two tables/schemas (source and target) that need to be matched. It operates solely on schema-level information (attribute names, metadata) without access to raw data in the tables. This is also a key difference compared to the table reasoning papers, which often rely on the actual data values for answering questions or verifying facts.

Outputs: Table reasoning papers aim to output answers to questions, binary fact verification labels, updated tables after manipulation, generated SQL/code, etc. In contrast, Matchmaker outputs a mapping between the source and target schema attributes, or indicates no match is possible for certain attributes. The set of attribute pairs representing the schema matching results, can be used to guide data integration processes.

Use of the LLM: Table reasoning employs LLMs for direct answer generation (Chen, 2023), program synthesis (Nahid & Rafiei, 2024; Kong et al., 2023), iterative prompting (Wang et al., 2023b), or as part of an agent system (Lu et al., 2024). Matchmaker uses LLMs for reasoning within a compositional program, generating candidates, refining them, and scoring confidence.

Optimization/Training: Table reasoning works explore fine-tuning (Nahid & Rafiei, 2024), instruction tuning (Lu et al., 2024), and in-context few-shot learning (Chen, 2023). Matchmaker introduces a novel optimization process to select synthetic in-context examples for self-improvement without labeled data or fine-tuning.

Key differences: In summary, while the table reasoning papers focus on tasks like question answering, fact verification, and table manipulation on single tables, Matchmaker addresses the distinct task of schema matching across table pairs. Its novel approach of a self-improving compositional language model program operating on schema-level information contrasts with general table reasoning which mostly use LLMs for direct table QA or program synthesis.

A.2 MATCHMAKER ALGORITHM

Below we provide a high-level overview algorithm of Matchmakers compositional language model program for schema matching.

Algorithm 2 Matchmaker: Schema Matching with Self-Improving Compositional Language Model Programs

Require: Source schema S_s, Target schema S_t
Ensure: Schema matches M

Stage 1: Multi-Vector Document Creation
for each table $T \in S_t$ **do**
  Create document D_T with attribute names and descriptions
  Append table metadata to D_T
  Encode D_T using ColBERT-v2 to obtain multi-vector representation V_T
  Add V_T to vector database $\mathcal{V}$
end for
Stage 2: Candidate Generation
for each source attribute $q_i \in S_s$ **do**
  Encode q_i using ColBERT-v2 to obtain query embedding E_{q_i}
  Retrieve top-k semantic candidates C_s from $\mathcal{V}$ using E_{q_i}
  Generate reasoning-based candidates C_R using LLM $l_c(q_i, S_t)$
  Refine candidate set $C^* \leftarrow l_r(C_s \cup C_R, q_i)$
end for
Stage 3: Confidence Scoring
for each source attribute $q_i \in S_s$ **do**
  Format candidate set C as multiple-choice question Q_i
  for each candidate $c_j \in C$ **do**
    Compute confidence score $s_j \leftarrow l_s(Q_i, c_j)$
  end for
  $m_i \leftarrow \arg\max_{c_j \in C} s_j$ ▷ Select match with highest confidence
  Add (q_i, m_i) to schema matches M
end for
Self-Improvement Optimization (Over all steps)
Generate evaluation set D_{eval} from unlabeled schemas
for each example $e_i \in D_{eval}$ **do**
  $(\hat{y}_i, \text{trace}_i) \leftarrow \text{Matchmaker}(e_i)$ ▷ Run Matchmaker to get output and traces
  $s_i \leftarrow E_l(e_i, \hat{y}i)$ ▷ Compute evaluation score using LLM E_l
  Add $(e_i, \text{trace}i, \hat{y}i, s_i)$ to D_{demo}
end for
Sort D_{demo} by score s_i
Select top-n examples from D_{demo} as synthetic in-context examples
Update Matchmaker components with selected in-context examples
return Final output: Schema matches M

A.3 SCHEMA MATCHING CHALLENGES.

- **Database Heterogeneity**: The number of tables in each schema may differ, i.e., $|T_s| \neq |T_t|$, making it challenging to establish correspondences between attributes across schemas.
- **Structural Heterogeneity**: Schemas may have different architectures, hierarchies, and representational granularity. If we define a hierarchy function $h(T_i)$ that describes the level of nesting within tables, differences in $h(T_{sj})$ and $h(T_{tk})$ for any j, k can lead to significant challenges in aligning attributes A_{sj} and A_{tk}.
- **Semantic Heterogeneity**: Attributes in different schemas may have the same name but different meanings, or different names but the same meaning. Let $N_i = \{n_{ij} | A_{ij} \in A_i\}$ be the set of attribute names for schema S_i. Semantic heterogeneity occurs when $\exists A_{sj} \in A_s, A_{tk} \in A_t : f(A_{sj}) = A_{tk} \wedge n_{sj} \neq n_{tk}$ or when $\exists A_{sj} \in A_s, A_{tk} \in A_t : f(A_{sj}) \neq A_{tk} \wedge n_{sj} = n_{tk}$.
- **Data Type Heterogeneity**: Attributes in different schemas may have different data types, even if they refer to the same concept. Let d_{ij} be the data type of attribute A_{ij}. Data type heterogeneity occurs when $\exists A_{sj} \in A_s, A_{tk} \in A_t : f(A_{sj}) = A_{tk} \wedge d_{sj} \neq d_{tk}$.
- **Information Mismatch**: Some attributes in one schema may lack a corresponding match in the other schema. This necessitates reasoning about "no possible match" cases, which is as important as reasoning about possible matches.
- **Unsupervised Nature**: Schema matching is unsupervised, where no labeled data pairs (A_{sj}, A_{tk}) are available to train or validate the mappings. This necessitates reliance on the intrinsic structure and semantic information encoded in A_i, making the development of an effective mapping function f challenging without external supervision.

A.4 Complexity of the MIMIC-OMOP task

MIMIC-OMOP is a real-world healthcare schema matching task, which is reflective of complex structures, interlinking and hierarchies that can be expected in real-world schema matching tasks. Hence, Matchmakers ability to empirically outperform baselines on these tasks highlights its ability to handle complex schemas.

To illustrate the complexity of the schemas that Matchmaker can handle, Figure 5 illustrates the complex schema structure and multiple tables.

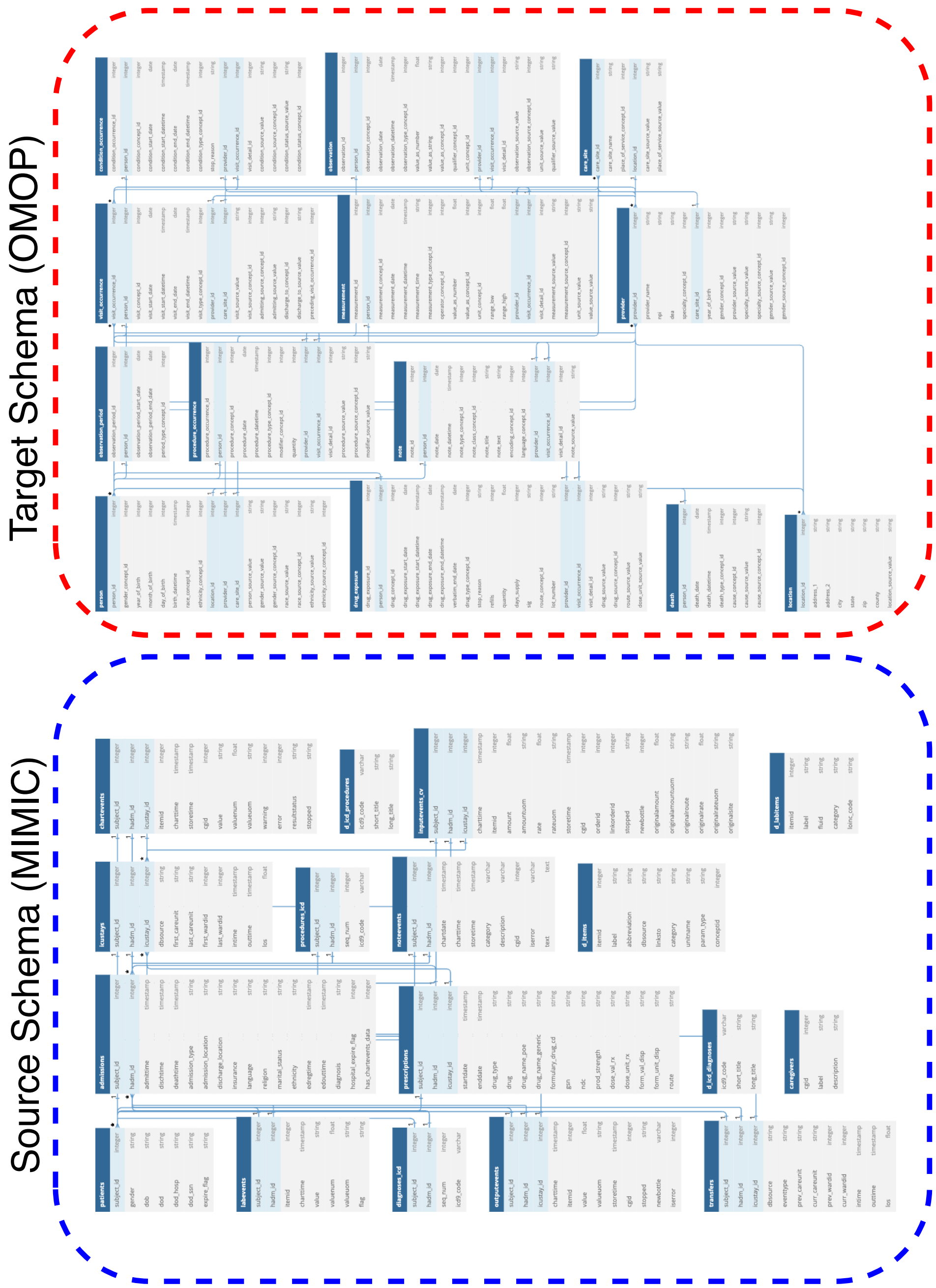

Figure 5: Illustration of the MIMIC-OMOP schema matching task showing the complexity and schema hierarchies.

A.5 Further details on schema matching formalism

In this appendix, we provide further details on the formulation of schema matching. We look at properties that a schema matching algorithm or function should possess, as well as, detailing how Matchmaker satisfies these properties.

Properties necessary. In practice, correctness in schema matching is evaluated against expert-validated ground truth mappings between the datasets (e.g. MIMIC to OMOP and Synthea to OMOP). However, this begs the question what properties would be useful ti improve emprical performance.

These lie along the following dimensions:

- Semantic Equivalence/Consistency: $f(A_S) = A_t$ implies A and A_t represent the same real-world concept (i.e. the mapped attributes serve equivalent purposes)
- Type Compatibility: Mapped attributes must have compatible data types
- Structural Consistency: Mappings must respect schema hierarchies
- Coverage: f identifies all valid matches while avoiding incorrect mappings through abstention. i.e. coverage is maximized by improved accuracy@k

We can then practically assess if a function f (such as Matchmaker) satisfies these criteria based on its performance against expert-validated ground truth mappings in real-world benchmark datasets as has been done in the paper.

How does Matchmaker satisfy these properties?

While we have empirically shown Matchmaker satisfies the properties needed of a schema matching function f, based on its strong performance on real-world schema matching tasks where it significantly outperforms existing approaches on standard benchmarks. In particular, the strong empirical performance outperforming the baselines implies that Matchmaker better satisfies the properties compared to the baseline schema matching algorithms.

However, let us analyze how Matchmaker also has specific design aspects within its compositional LLM structure that promotes addressing the properties.

- Semantic equivalence/consistency: Matchmaker employs multiple mechanisms: multi-vector document representation captures semantic nuances beyond simple name matching, while dual candidate generation combines both semantic retrieval and LLM reasoning to identify conceptually equivalent attributes.
- Type compatibility: enforced through inclusion of data type information in our multi-vector documents (Section 4.1) and LLM reasoning during candidate generation and refinement (Section 4.2), with examples in Appendix C showing explicit consideration of type compatibility (e.g., string->varchar, integer->bigint).
- Structural consistency is maintained by incorporating table metadata and hierarchical information in document creation (Section 4.1), using reasoning-based candidate generation that considers schema structure (Section 4.2), and including table context in confidence scoring.
- Coverage is optimized through our MCQ format with a "None of the above" option enabling abstention when no good match exists, while confidence scoring helps identify and rank high-quality matches. Our empirical results validate that these properties translate to superior performance in practice.

A.6 Detailed explanation of self-improvement

The self-improvement mechanism of Matchmaker is a pivotal component. We provide the Algorithm below.

Algorithm 3 Optimize LM program $\mathcal{L}$

Input: Set of evaluation queries $\mathcal{D}_{eval} = e_1, e_2, \ldots, e_n$
Output: Set of top n demonstrations D_{demo}
for each input $e_i \in \mathcal{D}_{eval}$ **do**
  $\hat{y}_i, trace_i \leftarrow \mathcal{L}(e_i)$ ▷ Teacher $\mathcal{L}$ predicts, storing outputs and intermediate traces
  $s_i \leftarrow \mathcal{E}(e_i, \hat{y}_i)$ ▷ Evaluation score
  $D_{demo} \leftarrow D_{demo} \cup (e_i, trace_i, \hat{y}_i, s_i)$
end for
Sort D_{demo} by score
return $D_{demo}[0:n]$ ▷ Select top n

In particular, we clarify that the self-improvement approach aims to address the issue of in-context learning for multi-stage LLM programs like Matchmaker. However, in doing so we need to address two fundamental challenges in our setting (C1 and C2):

(C1) Lack of labeled demonstrations: We do not have access to labeled input-output demonstrations from which to select in-context examples.

(C2) Lack of an evaluator for selection: To assess Matchmaker's capabilities and guide selection of examples, we need an evaluator.

We address each as follows:

- Addressing (C1): The process begins by creating an evaluation dataset D_eval from unlabeled schemas with two properties: "easy queries" where top-n semantic matches have similarity scores > 0.95, and "challenging queries" with the lowest semantic match scores. This ensures diverse coverage of different matching scenarios. The complete Matchmaker compositional program L is then run on each evaluation example $e_i \in D_{eval}$. We capture full execution traces including intermediate reasoning steps, candidate generation and refinement decisions, and final confidence scores and matches. The synthetic in-context examples refer to the intermediate input-output pairs generated by the LLM for the intermediate steps of the compositional LLM program. This deals with the challenge of a lack of labeled examples (i.e. zero-shot).
- Addressing (C2): To handle the lack of an evaluator (validation metric), we use an evaluator LLM E (i.e. an LLM-as-a-judge) to assess match quality through chain-of-thought reasoning, producing scores from 0-5 based on match relevance. Finally, the top-n traces are selected based on these evaluation scores. This systematic approach, detailed in Algorithm 1, enables principled selection of in-context examples based on traces that lead to good performance. We then use these as in-context examples for the different parts of the LLM program (as they led to good performance) — in order to guide the reasoning. As shown in the main paper our novel approach to self-improve outperforms random selection of in-context examples and self-reflection confirming that our systematic selection of in-context samples is the key driver of performance gains, rather than the mere inclusion of any in-context examples.

A.7 EXTENDED RELATED WORK

Classical Schema Matching approaches. Classical approaches to schema matching, as thoroughly reviewed by Rahm & Bernstein (2001), use a range of strategies, including heuristic-driven linguistic matching, constraint-based methods, and structural analysis. These methods have historically focused on simple relational schemas, matching elements between individual tables or flat structures. In particular, the primary focus is matching between individual tables or simple schemas (such as purchase orders).

Key Weaknesses of Classical Approaches and How Matchmaker Addresses Them:

- Single-Table and Flat Structure Focus: Classical methods typically perform schema matching at the element level, treating tables as isolated entities and matching attributes based on direct comparisons of names, data types, or simple structural cues. In particular, often a focus was simple relational schemas, where the goal was to map elements between single tables. However, this approach fails to handle the complexity of modern data systems, where schemas are often multi-table, hierarchical, or require cross-table reasoning. **Contrast:** Matchmaker, in contrast, uses LLM-based reasoning to connect attributes across multi-table and hierarchical schemas, understanding how data relationships span multiple tables. This makes our approach significantly more capable of handling complex and interrelated schema structures.

- Dependency on Heuristics and Limited Semantic Understanding: Classical methods rely on heuristic-driven matching based on linguistic similarities (e.g., name matching using synonyms, hypernyms, or edit distance) and structural constraints like key relationships. While these heuristics work in well-defined contexts, they are insufficient for domains where semantic meaning is implicit, such as in healthcare and as we show in Fig 1 — only semantic matching is in fact insufficient. **Contrast:** Matchmaker employs chain-of-thought prompting and advanced LLMs to perform reasoning, allowing it to capture relationships that are not explicitly defined in the schema structure or names. This enables Matchmaker to handle complex mappings that classical methods cannot infer.
- Manual Effort and Lack of Adaptability: Classical techniques require significant manual effort for tuning and adaptation, making them less suitable for rapidly evolving or heterogeneous environments. Constraint-based approaches, in particular, are difficult to scale across different domains without manual intervention. Alternatively, they might also rely on labeled data for effective matching. This makes these classical approaches impractical in real-world environments. **Contrast:** Matchmaker's zero-shot and self-optimization capabilities mean it can adapt autonomously to new schemas using synthetic in-context examples, significantly reducing the need for manual tuning and making it more practical for dynamic, real-world data integration tasks.

Key Weaknesses of SMAT and how Matchmaker improves: We also compared Matchmaker to state-of-the-art (SOTA) methods like SMAT Zhang et al. (2021), which applies attention mechanisms for schema matching. While SMAT represents an important advancement over classical methods, it has several limitations that Matchmaker overcomes:

- High Dependency on Labeled Data: SMAT requires extensive labeled data (over 50% labeled matches) for training, which is often impractical in real-world schema matching. **Contrast:** Matchmaker's zero-shot matching capability allows it to perform well without any labeled training data, using LLMs to generate and refine matches autonomously.
- Binary formulation: SMAT formulates the problem as binary classification task over the full Cartesian product of source and target schema attributes. e.g. for each pair of source-target attributes. This leads to a large amount of comparisons. **Contrast:** Matchmakers formulation as information retrieval reduces the number of comparisons and leads to greater efficiency — in addition to the better performance.

A.8 METRICS: ACCURACY, PRECISION, RECALL, F1-SCORE

In our m:1 schema matching setup, accuracy@1, precision, recall, and F1-score are equivalent due to the specific constraints of the task and the prediction mechanism employed. Below, we provide a detailed explanation of this equivalence:

2. **Task Constraints:** The schema matching task is constrained such that each source attribute can match to at most one target attribute (m:1 constraint). This ensures that the number of predictions equals the number of source attributes.

Equivalence of Metrics Given the above setup, the following equivalences hold:

Precision:

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

In our setup, every prediction corresponds to exactly one target attribute, and there are no extraneous or unassigned predictions. Therefore:

$$\text{Precision} = \frac{\text{Correct Matches}}{\text{Total Predictions}} = \text{Accuracy@1}.$$

Recall:

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

Since every source attribute must be matched to a target attribute, there are no unassigned predictions in our setup. However, incorrect matches can occur, leading to both false positives (FP) and false negatives (FN). In our m:1 schema matching setup, a prediction is either correct (a true positive, TP) or incorrect. An incorrect match to the wrong target attribute results in a false positive (FP) for the predicted target and a corresponding false negative (FN) for the true target. Consequently, the number of FP and FN are always equal, as they reflect the same prediction errors. In this setup, precision, recall, and accuracy@1 are equivalent because they all measure the proportion of correct matches (TPs) relative to the total predictions, with incorrect matches impacting all metrics identically. This equivalence holds when correctness is measured against the ground truth annotations from the benchmark datasets. Thus:

$$\text{Recall} = \frac{\text{Correct Matches}}{\text{Total Predictions}} = \text{Accuracy@1}.$$

F1-Score:

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

As both precision and recall are equal to accuracy@1 in this setup, the F1-score simplifies to:

$$\text{F1-Score} = \text{Accuracy@1}.$$

In summary, due to the constraints of our m:1 schema matching task and the `argmax` prediction mechanism, accuracy@1, precision, recall, and F1-score are mathematically equivalent. We report accuracy@1 in the main results, but the corresponding precision, recall, and F1-scores are identical and can be directly interpreted from the accuracy@1 values. We note this equivalence does not hold for one-to-many mappings

B Experimental details: Benchmarks & datasets

All experiments are run on a single Nvidia A4000 GPU with 20 GB of vram. We invoke GPT-4 via the Azure OpenAI API.

B.1 Benchmarks

B.1.1 Matchmaker

Matchmaker is a compositional language model program for schema matching made up of multiple component modules — formulated in the context of information retrieval.

GPT-4 Hyper-parameters. The model version used as the LLM was GPT-4-1106, with the following settings: {'temperature': 0.5, 'max_tokens': 1024, 'top_p': 1, 'frequency_penalty': 0, 'presence_penalty': 0, 'n': 1, }

Embedding model and documents. We use Colbert-V2 (Santhanam et al., 2022) as the embedding model and follow the document creation process as outlined in Sec. 4.1. We use the implementation of Colbert-v2 from RAGatouille (https://github.com/bclavie/RAGatouille/).

Candidates. For both semantic and reasoning-based candidates, we set k=5.

Optimization. As described in the main paper, we generate synthetic in-context samples to address the unique challenges of a lack of labeled data and no demonstrations. As described, to achieve this we follow a boostrapping process like in DSPy (Khattab et al., 2023). For our experiments we select at maximum 4 synthetic in-context examples

Prompts: We show examples with the prompts for each component of Matchmaker in Appendix C.

B.1.2 ReMatch

In the main text we report the numbers directly from the ReMatch paper, as there is no open-source implementation.

How we selected the numbers to report: The ReMatch paper does an exploration of the number of documents retrieved. Hence, we use the following two criteria.
(i) At least 1 document must be retrieved. i.e. the retrieval step cannot be skipped.

(ii) We then select the result that satisfies (i), with the highest accuracy@5.

Our implementation of ReMatch follows the original paper (Sheetrit et al., 2024). We use OpenAI Ada embeddings for the embedding model and GPT-4 as the LLM.

We following the document creation procedure and use the prompt template as provided.

GPT-4 Hyper-parameters. The model version used for generation was GPT-4-1106, with the following settings from the ReMatch paper: {seed=42, temperature=0.5, max_tokens=4096, top_p=0.9, frequency_penalty=0, presence_penalty=0}

B.1.3 Jellyfish

Jellyfish (Zhang et al., 2023b) is a fine-tuned language model tailored for data preprocessing tasks including schema matching. The 7B and 13B models are fine tuned upon the OpenOrca-Platypus2 model.

Implementation (7b): https://huggingface.co/NECOUDBFM/Jellyfish-7B

Implementation (13b): https://huggingface.co/NECOUDBFM/Jellyfish-13B

B.1.4 LLM-DP

LLM-DP (Narayan et al., 2022; Zhang et al., 2023a) refer to works which have used pre-trained LLMs like GPT-3.5 or GPT-4 for data processing tasks like schema matching via prompting. Since the papers in the few-shot case use labeled examples we do not use those — given its unrealistic in practice. Hence, for these baselines they operate in a zero shot manner.

Implementation: https://github.com/HazyResearch/fm_data_tasks

B.1.5 SMAT

SMAT is a supervised learning approach which performs schema matching via an attention mechanism. Of course, the model needs labeled data to train on. In our experiments, we assess two variants given that labeled training data for schema matching is hard to access: (i) 20-80: 20% train and 80% test and (ii) 50-50: 50% train and 50% test.

We use the default hyper-parameters: {Learning Rate: 0.8, Batch Size: 64, Epochs: 30}

Implementation: https://github.com/JZCS2018/SMAT

B.2 Datasets

We outline the two real-world schema matching benchmarks used in this paper — MIMIC and Synthea. These datasets mapping different clinical/healthcare schemas were chosen as they are the standard datasets used in schema matching literature and consequently, used by prior works providing fair assessment. They are also considered the most reflective of real-world schema matching complexity and challenges. We note that the scarcity of complex and challenging real-world datasets, underscores the challenges in collecting and annotating real-world schema matching data. For instance, as noted in Sec 1, annotating MIMIC-OMOP alone required 500 hours from two medical experts.

Table 5 provides a summary of the table properties.

Note there is no specific train-test sets used as in supervised learning. As we perform the schema matching task in a zero-shot manner.

Table 5: Summary of the table properties of our two schema matching datasets.

Dataset	Source Tables	Target Tables
MIMIC-OMOP	26	14
SYNTHEA-OMOP	12	21

MIMIC Dataset: The dataset contains a schema mapping between the MIMIC-III electronic health record (Source schema) (Johnson et al., 2016) and The Observational Medical Outcomes Partnership Common Data Model (OMOP schema) (Target schema).

This dataset is currently the largest publicly available schema matching dataset (Sheetrit et al., 2024) and is the cloest to a real-world schema matching use case, wherein a proprietary database created for a specific purpose (a source schema) is mapped to a given industry standard (a target schema) for further uses. In this case the proprietary database schema is MIMIC and the industry standard is the OMOP common data model.

Open-source data: https://github.com/meniData1/MIMIC_2_OMOP

Synthea Dataset: The Synthea dataset is part of the OMAP benchmark (Zhang et al., 2021) and is a partial mapping of the Synthea (Walonoski et al., 2018) (Source Schema) which is a synthetic healthcare dataset of a Massachusetts health records and attempts to map it to a subset of the OMOP CDM (Target Schema). The dataset has widely been used in previous schema matching papers (Sheetrit et al., 2024; Narayan et al., 2022; Zhang et al., 2021) as a realistic and challenging real-world schema matching benchmark.

Open-source data: https://github.com/JZCS2018/SMAT/tree/main/datasets/omap/

C EXAMPLES USING MATCHMAKER (WITH PROMPTS)

C.1 MATCHMAKER PROMPT EXAMPLES

We show two end-to-end schema matching examples with Matchmaker, where other methods fail. (1) Example 1: case with No possible target schema match for the source schema query, (2) Example 2: challenging reasoning case, where there is a match possible between source and target schema.
► In each component, we can show the "Optimized" In-context examples.

C.1.1 EXAMPLE 1.

Source schema query: admissions-marital_status(string): Table admissions details-the admissions table gives information regarding a patient's admission to the hospital., Attribute marital_status details -describe patient demographics.

Target scheme match: None possible.

Matchmaker: None of the above.

Figure 6: EXAMPLE 1: Candidate generation.

Candidate generation

You are an OMOP Schema expert. Your goal is to take the OMOP schema and based on the input, refine the schema to include only 5 most likely matches to the input query.

—

Follow the following format.
Input Schema: Input OMOP schema values Input Query: input query Refined Schema: Five most likely matches to input query. Include most likely matches to the input query. Respond with a single JSON object. JSON Schema: {"defs": {"Extractor": {"properties": {"related": {"description": "related matches", "title": "Related", "type": "string"}}, "required": ["related"], "title": "Extractor", "type": "object"}}, "properties": {"value": {"items": {"ref": "/defs/Extractor"}, "title": "Value", "type": "array"}}, "required": ["value"], "title": "Output", "type": "object"}

—

Input Schema: ['visit_occurrence-person_id(bigint)', 'visit_occurrence-visit_occurrence_id(bigint)', ... 'procedure_occurrence-provider_id(bigint)', 'visit_detail-visit_detail_source_value(varchar(50))']
Input Query: procedureevents_mv-itemid
Refined Schema: "value": ["related": "procedure_occurrence-person_id(bigint)", "related": "procedure_occurrence-visit_occurrence_id(bigint)", "related": "procedure_occurrence-procedure_date(date)", "related": "procedure_occurrence-procedure_source_value(varchar(50))", "related": "procedure_occurrence-procedure_concept_id(integer)"]

—

Input Schema: ['visit_occurrence-person_id(bigint)', 'visit_occurrence-visit_occurrence_id(bigint)', ... 'procedure_occurrence-provider_id(bigint)', 'visit_detail-visit_detail_source_value(varchar(50))']
Input Query: noteevents-text
Refined Schema: "value": ["related": "note-note_text(varchar(max))", "related": "note-note_title(varchar(250))", "related": "note-note_source_value(varchar(50))", "related": "note-note_date(date)", "related": "note-note_datetime(datetime)"]

—

Input Schema: ['visit_occurrence-person_id(bigint)', 'visit_occurrence-visit_occurrence_id(bigint)', ... 'procedure_occurrence-provider_id(bigint)', 'visit_detail-visit_detail_source_value(varchar(50))']
Input Query: admissions-marital_status(string): Table admissions details-the admissions table gives information regarding a patient's admission to the hospital., Attribute marital_status details -describe patient demographics.
Refined Schema:

LLM Response

{"value": [{"related": "person-person_id(bigint)"}, {"related": "person-gender_source_value(varchar(50))"}, {"related": "person-birth_datetime(datetime)"}, {"related": "person-death_datetime(datetime)"}, {"related": "visit_occurrence-admitted_from_source_value(varchar(50))"}]}

Figure 7: EXAMPLE 1: Candidate refinement.

Candidate refiner

You are an expert OMOP matching ranker. Your task is to take the OMOP candidates and based on the input, refine the candidates to select the 5 most likely matches to the input query. Return ONLY the keys.

—

Follow the following format.

Input Schema: List of key: value pairs

Input Query: input query

Reasoning: Let's think step by step in order to {produce the refined_string_list}. We ...

Refined String List: Five most likely matches to input query. Include maximum of the 5 most likely matches to the input query. Return ONLY the keys

—

Input Schema: ['person-person_id(bigint): Table person details-this table serves as the central identity management for all persons in the database. it contains records that uniquely identify each person or patient, and some demographic information., Attribute person_id details -it is assumed that every person with a different unique identifier is in fact a different person and should be treated independently.', ... 'measurement-measurement_concept_id(integer): the measurement_concept_id field is recommended for primary use in analyses, and must be used for network studies']

Input Query: procedureevents_mv-itemid

Reasoning: Let's think step by step in order to produce the refined string list. We are looking for a match to 'procedureevents_mv-itemid', which seems to relate to a procedure event and its item identifier. The 'procedureevents_mv' part suggests we're looking at a procedure occurrence, and the 'itemid' part suggests we're looking for an identifier or id field. Given that, we should look for keys in the 'procedure_occurrence' table that represent identifiers.

Refined String List: 'procedure_occurrence-person_id(bigint)', 'procedure_occurrence-visit_occurrence_id(bigint)', 'procedure_occurrence-procedure_concept_id(integer)', 'procedure_occurrence-visit_detail_id(bigint)', 'procedure_occurrence-provider_id (bigint)'

—

Input Schema: ['person-person_id(bigint): Table person details-this table serves as the central identity management for all persons in the database. it contains records that uniquely identify each person or patient, and some demographic information., Attribute person_id details -it is assumed that every person with a different unique identifier is in fact a different person and should be treated independently.', ... 'measurement-measurement_concept_id(integer): the measurement_concept_id field is recommended for primary use in analyses, and must be used for network studies']

Input Query: noteevents-text

Reasoning: Let's think step by step in order to produce the refined string list. We are looking for matches to "noteevents-text" in the input schema. The term "noteevents" matches with "note" in the input schema, and "text" matches with "note_text". So, we should consider those keys that contain both "note" and "text".

Refined String List: ['note-note_text(varchar(max))', 'note-note_title(varchar(250))', 'note-note_source_value(varchar(50))', 'note-note_text(varchar(max))']

—

Input Schema: ['person-person_id(bigint): Table person details-this table serves as the central identity management for all persons in the database. it contains records that uniquely identify each person or patient, and some demographic information., Attribute person_id details -it is assumed that every person with a different unique identifier is in fact a different person and should be treated independently.', ... 'measurement-measurement_concept_id(integer): the measurement_concept_id field is recommended for primary use in analyses, and must be used for network studies']

Input Query: admissions-marital_status(string): Table admissions details-the admissions table gives information regarding a patient's admission to the hospital., Attribute marital_status details -describe patient demographics.

Reasoning: Let's think step by step in order to

LLM Response

"produce the refined string list. We are looking for a match to 'admissions-marital_status', which seems to relate to a hospital admission and the marital status of the patient. The 'admissions' part suggests we're looking at a visit occurrence or visit detail, and the 'marital_status' part suggests we're looking for demographic information, which is usually found in the 'person' table. Given that, we should look for keys in the 'visit_occurrence', 'visit_detail', and 'person' tables.String List: 'visit_occurrence-person_id(bigint)', 'visit_occurrence-visit_occurrence_id(bigint)', 'visit_detail-person_id(bigint)', 'visit_detail-visit_occurrence_id(bigint",

Figure 8: EXAMPLE 1: MCQ Formatter.

MCQ Formatter

You are an expert MCQ formatter. Your task is to take a list of schema values and convert them into a multiple choice question format with (letter)Schema value, where the schema values should be key(description).

—

Follow the following format.

Input: input list of schema values Mcq: MCQ format of schema values e.g (A)Schema value, (B)Schema value. Do not include additional options, only the schema values as options. where the schema values should be key(description). Add a No Match option.

—

Input: ["'procedure_occurrence-person_id(bigint)', 'procedure_occurrence-visit_occurrence_id(bigint)', 'procedure_occurrence-procedure_concept_id(integer)', 'procedure_occurrence-visit_detail_id(bigint)', 'procedure_occurrence-provider_id (bigint)'"] Mcq: (A)'procedure_occurrence-person_id(bigint)', (B)'procedure_occurrence-visit_occurrence_id(bigint)', (C)'procedure_occurrence-procedure_concept_id(integer)', (D)'procedure_occurrence-visit_detail_id(bigint)', (E)'procedure_occurrence-provider_id (bigint)', (F)No Match

—

Input: ["['note-note_text(varchar(max))', 'note-note_title(varchar(250))', 'note-note_source_value(varchar(50))', 'note-note_text(varchar(max))']"] Mcq: (A)note-note_text(varchar(max)), (B)note-note_title(varchar(250)), (C)note-note_source_value(varchar(50)), (D)note-note_text(varchar(max)), (E)No Match

—

Input: ["'visit_occurrence-person_id(bigint)', 'visit_occurrence-visit_occurrence_id(bigint)', 'visit_detail-person_id(bigint)', 'visit_detail-visit_occurrence_id(bigint"] Mcq:

————————————————————————

LLM Response

————————————————————————-

"(A)'visit_occurrence-person_id(bigint)', (B)'visit_occurrence-visit_occurrence_id(bigint)', (C)'visit_detail-person_id(bigint)', (D)'visit_detail-visit_occurrence_id(bigint)', (E)No Match",

Figure 9: EXAMPLE 1: Confidence scoring.

Confidence scoring

You are a schema matching expert. Your task is given the input and the MCQ format of the schema, predict the likelihood or relation score from 0-100 of the input query being related to each option. Your scores will be calibrated. If there is no good match score No Match as 100

—

Follow the following format.
Input Mcq: Input MCQ format of schema values
Input Query: input query
Relation: Relation score of input query being related to the option as value. Assess each independently including No Match, returning a score from 0-100 for each. Return with key as MCQ letter e.g (A) and score=value as JSON

—

Input Mcq: (A)'procedure_occurrence-person_id(bigint)', (B)'procedure_occurrence-visit_occurrence_id(bigint)', (C)'procedure_occurrence-procedure_concept_id(integer)', (D)'procedure_occurrence-visit_detail_id(bigint)', (E)'procedure_occurrence-provider_id (bigint)', (F)No Match
Input Query: procedureevents_mv-itemid
Relation: {"(A)": 0, "(B)": 0, "(C)": 0, "(D)": 0, "(E)": 0, "(F)": 100}

—

Input Mcq: (A)note-note_text(varchar(max)), (B)note-note_title(varchar(250)), (C)note-note_source_value(varchar(50)), (D)note-note_text(varchar(max)), (E)No Match
Input Query: noteevents-text
Relation: {"A": 90, "B": 0, "C": 0, "D": 90, "E": 10}

—

Input Mcq: (A)'visit_occurrence-person_id(bigint)', (B)'visit_occurrence-visit_occurrence_id(bigint)', (C)'visit_detail-person_id(bigint)', (D)'visit_detail-visit_occurrence_id(bigint', (E)No Match Input Query: admissions-marital_status(string): Table admissions details-the admissions table gives information regarding a patient's admission to the hospital., Attribute marital_status details -describe patient demographics.
Relation:

————————————————————————

LLM Response

————————————————————————-

"{(̈A)̈: 0, (̈B)̈: 0, (̈C)̈: 0, (̈D)̈: 0, (̈E)̈: 100}"

C.1.2 EXAMPLE 2

Source schema query: admissions-marital_status(string): Table admissions details-the admissions table gives information regarding a patient's admission to the hospital., Attribute marital_status details -describe patient demographics.

Target scheme match: 'procedure_occurrence- quantity

Matchmaker: 'procedure_occurrence- quantity

Figure 10: Candidate generation.

EXAMPLE 2: Candidate generation.

You are an OMOP Schema expert. Your goal is to take the OMOP schema and based on the input, refine the schema to include only 5 most likely matches to the input query.
—
Follow the following format.
Input Schema: Input OMOP schema values Input Query: input query Refined Schema: Five most likely matches to input query. Include maximum of the 10 most likely matches to the input query. Respond with a single JSON object. JSON Schema: {"defs": {"Extractor": {"properties": {"related": {"description": "related matches", "title": "Related", "type": "string"}}, "required": ["related"], "title": "Extractor", "type": "object"}}, "properties": {"value": {"items": {"ref": "/defs/Extractor"}, "title": "Value", "type": "array"}}, "required": ["value"], "title": "Output", "type": "object"}
—
Input Schema: ['visit_occurrence-person_id(bigint)', 'visit_occurrence-visit_occurrence_id(bigint)', ... 'visit_detail-visit_detail_source_value(varchar(50))'] Input Query: procedureevents_mv-itemid Refined Schema: {"value": [{"related": "procedure_occurrence-person_id(bigint)"}, {"related": "procedure_occurrence-visit_occurrence_id(bigint)"}, {"related": "procedure_occurrence-procedure_date(date)"}, {"related": "procedure_occurrence-procedure_source_value(varchar(50))"}, {"related": "procedure_occurrence-procedure_concept_id(integer)"}]}
—
Input Schema: ['visit_occurrence-person_id(bigint)', 'visit_occurrence-visit_occurrence_id(bigint)', ... 'visit_detail-visit_detail_source_value(varchar(50))'] Input Query: noteevents-text Refined Schema: {"value": [{"related": "note-note_text(varchar(max))"}, {"related": "note-note_title(varchar(250))"}, {"related": "note-note_source_value(varchar(50))"}, {"related": "note-note_date(date)"}, {"related": "note-note_datetime(datetime)"}]}
—
Input Schema: ['visit_occurrence-person_id(bigint)', 'visit_occurrence-visit_occurrence_id(bigint)', ... 'visit_detail-visit_detail_source_value(varchar(50))'] Input Query: procedures_icd-seq_num(integer): Table procedures_icd details-contains icd procedures for patients, most notably icd-9 procedures, Attribute seq_num details -provides the order in which the procedures were performed Refined Schema:
"{v̈alue: [{r̈elated: p̈rocedure_occurrence-person_id(bigint)}̈, {r̈elated: p̈rocedure_occurrence-visit_occurrence_id(bigint)}̈, {r̈elated: p̈rocedure_occurrence-procedure_date(date)}̈, {r̈elated: p̈rocedure_occurrence-procedure_concept_id(integer)}̈, {r̈elated: p̈rocedure_occurrence-quantity(integer)}̈]}"

Figure 11: EXAMPLE 2: Candidate Refinement.

Candidate refinement

You are an expert OMOP matching ranker. Your task is to take the OMOP candidates and based on the input, refine the candidates to select the 5 most likely matches to the input query. Return ONLY the keys.

—

Follow the following format.
Input Schema: List of key: value pairs
Input Query: input query
Reasoning: Let's think step by step in order to {produce the refined_string_list}. We ...
Refined String List: Five most likely matches to input query. Include maximum of the 5 most likely matches to the input query. Return ONLY the keys

—

Input Schema: ['procedure_occurrence-person_id(bigint): Table procedure_occurrence details-this table contains records of activities or processes ordered by, or carried out by, a healthcare provider on the patient with a diagnostic or therapeutic purpose., Attribute person_id details -the person_id of the person for whom the procedure is recorded. this may be a system generated code.' .... 'visit_detail-care_site_id(bigint): this field provides information about the care site where the visit detail took place']
Input Query: procedureevents_mv-itemid
Reasoning: Let's think step by step in order to produce the refined string list. We are looking for a match to 'procedureevents_mv-itemid', which seems to relate to a procedure event and its item identifier. The 'procedureevents_mv' part suggests we're looking at a procedure occurrence, and the 'itemid' part suggests we're looking for an identifier or id field. Given that, we should look for keys in the 'procedure_occurrence' table that represent identifiers.
Refined String List: 'procedure_occurrence-person_id(bigint)', 'procedure_occurrence-visit_occurrence_id(bigint)', 'procedure_occurrence-procedure_concept_id(integer)', 'procedure_occurrence-visit_detail_id(bigint)', 'procedure_occurrence-provider_id (bigint)'

—

Input Schema: ['procedure_occurrence-person_id(bigint): Table procedure_occurrence details-this table contains records of activities or processes ordered by, or carried out by, a healthcare provider on the patient with a diagnostic or therapeutic purpose., Attribute person_id details -the person_id of the person for whom the procedure is recorded. this may be a system generated code.' .... 'visit_detail-care_site_id(bigint): this field provides information about the care site where the visit detail took place']
Input Query: noteevents-text
Reasoning: Let's think step by step in order to produce the refined string list. We are looking for matches to "noteevents-text" in the input schema. The term "noteevents" matches with "note" in the input schema, and "text" matches with "note_text". So, we should consider those keys that contain both "note" and "text".
Refined String List: ['note-note_text(varchar(max))', 'note-note_title(varchar(250))', 'note-note_source_value(varchar(50))', 'note-note_text(varchar(max))']

—

Input Schema: ['procedure_occurrence-person_id(bigint): Table procedure_occurrence details-this table contains records of activities or processes ordered by, or carried out by, a healthcare provider on the patient with a diagnostic or therapeutic purpose., Attribute person_id details -the person_id of the person for whom the procedure is recorded. this may be a system generated code.' .... 'visit_detail-care_site_id(bigint): this field provides information about the care site where the visit detail took place']
Input Query: procedures_icd-seq_num(integer): Table procedures_icd details-contains icd procedures for patients, most notably icd-9 procedures, Attribute seq_num details -provides the order in which the procedures were performed
Reasoning: Let's think step by step in order to
"produce the refined string list. We are looking for a match to 'procedures_icd-seq_num', which seems to relate to a procedure occurrence and its sequence number. The 'procedures_icd' part suggests we're looking at a procedure occurrence, and the 'seq_num' part suggests we're looking for a sequential number or order field. Given that, we should look for keys in the 'procedure_occurrence' table that represent order or sequence.
Refined String List: 'procedure_occurrence-person_id(bigint)', 'procedure_occurrence-visit_occurrence_id(bigint)', 'procedure_occurrence-procedure_date(date)', 'procedure_occurrence-procedure_concept_id(integer)', 'procedure_occurrence-quantity(integer)'"

Figure 12: EXAMPLE 2: MCQ Formatter.

MCQ Formatter

You are an expert MCQ formatter. Your task is to take a list of schema values and convert them into a multiple choice question format with (letter)Schema value, where the schema values should be key(description).

—

Follow the following format.

Input: input list of schema values Mcq: MCQ format of schema values e.g (A)Schema value, (B)Schema value. Do not include additional options, only the schema values as options. where the schema values should be key(description). Add a No Match option.

—

Input: ["'procedure_occurrence-person_id(bigint)', 'procedure_occurrence-visit_occurrence_id(bigint)', 'procedure_occurrence-procedure_concept_id(integer)', 'procedure_occurrence-visit_detail_id(bigint)', 'procedure_occurrence-provider_id (bigint)'"] Mcq: (A)'procedure_occurrence-person_id(bigint)', (B)'procedure_occurrence-visit_occurrence_id(bigint)', (C)'procedure_occurrence-procedure_concept_id(integer)', (D)'procedure_occurrence-visit_detail_id(bigint)', (E)'procedure_occurrence-provider_id (bigint)', (F)No Match

—

Input: ["['note-note_text(varchar(max))', 'note-note_title(varchar(250))', 'note-note_source_value(varchar(50))', 'note-note_text(varchar(max))']"] Mcq: (A)note-note_text(varchar(max)), (B)note-note_title(varchar(250)), (C)note-note_source_value(varchar(50)), (D)note-note_text(varchar(max)), (E)No Match

—

Input: ["'procedure_occurrence-person_id(bigint)', 'procedure_occurrence-visit_occurrence_id(bigint)', 'procedure_occurrence-procedure_date(date)', 'procedure_occurrence-procedure_concept_id(integer)', 'procedure_occurrence-quantity(integer)'"] Mcq:
"(A)'procedure_occurrence-person_id(bigint)', (B)'procedure_occurrence-visit_occurrence_id(bigint)', (C)'procedure_occurrence-procedure_date(date)', (D)'procedure_occurrence-procedure_concept_id(integer)', (E)'procedure_occurrence-quantity(integer)', (F)No Match",

Figure 13: EXAMPLE 2: Confidence scoring.

Confidence scoring

You are a schema matching expert. Your task is given the input and the MCQ format of the schema, predict the likelihood or relation score from 0-100 of the input query being related to each option. Your scores will be calibrated. If there is no good match score No Match as 100

—

Follow the following format.

Input Mcq: Input MCQ format of schema values Input Query: input query Relation: Relation score of input query being related to the option as value. Assess each independently including No Match, returning a score from 0-100 for each. Return with key as MCQ letter e.g (A) and score=value as JSON

—

Input Mcq: (A)'procedure_occurrence-person_id(bigint)', (B)'procedure_occurrence-visit_occurrence_id(bigint)', (C)'procedure_occurrence-procedure_concept_id(integer)', (D)'procedure_occurrence-visit_detail_id(bigint)', (E)'procedure_occurrence-provider_id (bigint)', (F)No Match Input Query: procedureevents_mv-itemid Relation: {"(A)": 0, "(B)": 0, "(C)": 0, "(D)": 0, "(E)": 0, "(F)": 100}

—

Input Mcq: (A)note-note_text(varchar(max)), (B)note-note_title(varchar(250)), (C)note-note_source_value(varchar(50)), (D)note-note_text(varchar(max)), (E)No Match Input Query: noteevents-text Relation: {"A": 90, "B": 0, "C": 0, "D": 90, "E": 10}

—

Input Mcq: (A)'procedure_occurrence-person_id(bigint)', (B)'procedure_occurrence-visit_occurrence_id(bigint)', (C)'procedure_occurrence-procedure_date(date)', (D)'procedure_occurrence-procedure_concept_id(integer)', (E)'procedure_occurrence-quantity(integer)', (F)No Match Input Query: procedures_icd-seq_num(integer): Table procedures_icd details-contains icd procedures for patients, most notably icd-9 procedures, Attribute seq_num details -provides the order in which the procedures were performed Relation: "{(̈A)̈: 0, (̈B)̈: 0, (̈C)̈: 0, (̈D)̈: 0, (̈E)̈: 90, (̈F)̈: 10}",

C.2 LLM EVALUATOR

We provide examples of the LLM evaluator, showing demonstrations achieving high and low scores.

LLM Evaluator Example: High Score

You are a schema matching expert, your task is to rate if any of the suggested matches are potential good matches for the query. Be lenient and rate a match as good (4 or 5) if it is relevant to the query. Rate the matches from 1-5. If none of the matches are good, rate 0.

—

Follow the following format.
Query: The query.
Answers: possible matches
Reasoning: Let's think step by step in order to {produce the rating}. We ...
Rating: Rate if any of the suggested matches are good for the query from 1-5. Only output the rating and nothing else.

—

Query: noteevents-chartdate
Answers: ["'note-note_date(date)', 'note-note_datetime(datetime)', 'note-visit_occurrence_id(bigint)', 'note-provider_id(bigint)', 'note-person_id(bigint)'"]
Reasoning: Let's think step by step in order to

————————————————————————

LLM Response

————————————————————————-

produce the rating. We can see that the query is looking for a match related to the date of a note event in a chart. The first two suggested matches, 'note-note_date(date)' and 'note-note_datetime(datetime)', are both related to dates and notes, making them potentially good matches for the query. The other three suggestions are not directly related to dates or note events, so they are not as good matches.
Rating: 4

Figure 14: LLM evaluator example, rated with a high score.

LLM Evaluator Example: Low Score

You are a schema matching expert, your task is to rate if any of the suggested matches are potential good matches for the query. Be lenient and rate a match as good (4 or 5) if it is relevant to the query. Rate the matches from 1-5. If none of the matches are good, rate 0.

—

Follow the following format.

Query: The query.

Answers: possible matches

Reasoning: Let's think step by step in order to {produce the rating}. We ...

Rating: Rate if any of the suggested matches are good for the query from 1-5. Only output the rating and nothing else.

—

Query: cptevents-subsectionheader

Answers: ['visit_occurrence-visit_type_concept_id(int)', 'visit_detail-visit_detail_concept_id(int)', 'procedure_occurrence-procedure_concept_id(int)', 'procedure_occurrence-visit_detail_id(int)', 'condition_occurrence-visit_occurrence_id(int)']

Reasoning: Let's think step by step in order to

———————————————————————

LLM Response

———————————————————————-

"produce the rating. We can see that the query is looking for a match related to 'cptevents-subsectionheader'. However, none of the suggested matches seem to directly relate to this query. They all refer to different concepts such as 'visit_occurrence', 'visit_detail', 'procedure_occurrence', and 'condition_occurrence'. Therefore, none of the matches seem to be a good fit for the query.

Rating: 0

Figure 15: LLM evaluator example, rated with a low score.

D Additional experiments

D.1 Number of LLM calls

Goal. To compare the number of LLM calls required by Matchmaker and other baseline methods for schema matching on the MIMIC-OMOP and SYNTHEA-OMOP datasets.

Experiment. We count the number of LLM calls made by each method during the schema matching process on both the MIMIC-OMOP and SYNTHEA-OMOP datasets. For methods that do not rely on LLMs (e.g., SMAT), we consider the number of forward passes through the neural network as equivalent to an LLM call for comparison purposes.

Results. Table 6 presents the number of LLM calls required by each method on the two datasets.

Table 6: Number of LLM calls

Method	MIMIC-OMOP	SYNTHEA-OMOP
Matchmaker	1340	890
ReMatch	268	178
Jellyfish-13b	24771	29637
Jellyfish-7b	24771	29637
LLM-DP	24771	29637
SMAT	24771	29637

Discussion. The results in Table 6 highlight the efficiency of Matchmaker and ReMatch in terms of the number of LLM calls required for schema matching.

Both Matchmaker and ReMatch formulate schema matching as an information retrieval problem, which significantly reduces the search space compared to the binary classification formulation used by Jellyfish-13b, Jellyfish-7b, LLM-DP, and SMAT.

The high number of LLM calls required by Jellyfish-13b, Jellyfish-7b, LLM-DP, and SMAT can be attributed to their formulation of schema matching as a binary classification problem over the Cartesian product of source and target attributes. In this formulation, the LLM is prompted to provide a label of Yes/No for each pair of source-target attributes, resulting in a large number of LLM calls that scales ($O(n^2)$). Consequently, these methods are computationally expensive and less scalable compared to Matchmaker and ReMatch, which employ a more efficient approach.

The fewer number of LLM calls used by Matchmaker and ReMatch has practical implications in terms of computational cost and runtime efficiency. By reducing the number of LLM calls, these methods can perform schema matching more quickly and with lower computational overhead compared to methods that rely on a large number of calls. This is particularly important when dealing with large-scale schemas or when schema matching needs to be performed frequently in real-world applications.

D.2 Matchmaker with other LLMs

Goal. To understand the performance of Matchmaker when using a less powerful LLM backbone compared to GPT-4, and contrast it with the ReMatch baseline using GPT-4.

Experiment. We evaluate the performance of Matchmaker using GPT-3.5 as the backbone LLM for all components, instead of GPT-4 which was used in the main experiments. We compare this to the performance of Matchmaker with GPT-4 and ReMatch with GPT-4. All other aspects of the setup remain the same as in the main text.

Results. Table 7 shows the schema matching accuracy@k for the different methods. We observe that Matchmaker with GPT-3.5 performs worse than Matchmaker with GPT-4, which is expected given GPT-3.5 is a less powerful LLM. Interestingly, Matchmaker with GPT-3.5 achieves comparable performance to ReMatch with GPT-4, despite GPT-3.5 being a much weaker LLM than GPT-4. On MIMIC, Matchmaker with GPT-3.5 slightly outperforms ReMatch with GPT-4 for accuracy@1 and is competitive at higher k. On Synthea, performance is similar for accuracy@1 but Matchmaker with GPT-3.5 outperforms ReMatch with GPT-4 for accuracy@3 and accuracy@5.

Table 7: Comparison of schema matching performance of different baselines.

		Matchmaker (GPT-4)	Matchmaker (GPT-3.5)	ReMatch (GPT-4)
MIMIC	acc@1	**62.20 ± 2.40** ↑	48.30± 2.80 ↑	42.50
	acc@3	**68.80 ± 2.00**	62.00 ± 4.20	63.80
	acc@5	**71.10 ± 2.00**	70.00 ± 4.20	**72.90**
Synthea	acc@1	**70.20 ± 1.70**	47.80 ± 3.20	50.50
	acc@3	**78.60 ± 2.50**	63.30 ± 4.30 ↑	58.10
	acc@5	**80.90 ± 1.10**	77.10 ± 0.70 ↑	74.30

Discussion. These results demonstrate that the Matchmaker approach of using a compositional LLM program is quite robust and can provide good schema matching performance even with weaker LLM backbones. The fact that Matchmaker with GPT-3.5 is competitive with ReMatch using GPT-4 highlights the strength of the multi-stage Matchmaker approach over ReMatch's single-stage LLM usage. However, using a more powerful LLM like GPT-4 still provides significant gains, underlining the importance of using an LLM with powerful reasoning capabilities for this complex task.

D.3 FURTHER PERFORMANCE RESULTS: REMATCH REIMPLEMENTATION

Goal. To compare the performance of Matchmaker against the ReMatch baseline, using both the original reported results from the ReMatch paper and the re-implementation of ReMatch.

Experiment. In the main paper, we report the performance of the ReMatch baseline using the results directly from the paper, as code is not available for ReMatch. However, for completeness, we also re-implement the ReMatch approach based on the details provided in the ReMatch paper.

Our re-implementation uses the OpenAI Ada-002 text embeddings for the retrieval step, following the same procedure as ReMatch for chunking and creating documents. We then use the same prompts as described in the ReMatch paper for the schema matching task. We compare the performance of our re-implemented ReMatch with the original reported results and Matchmaker.

Results. Table 8 presents the schema matching accuracy@k for Matchmaker, the original ReMatch results, and our re-implemented ReMatch. We observe that Matchmaker consistently outperforms both the original ReMatch results and our re-implementation across all metrics and datasets. We also find the re-implemented ReMatch achieves lower performance compared to the original reported results.

Table 8: Comparison of schema matching performance of different baselines.

		Matchmaker	ReMatch (Original)	ReMatch (Reimplemented)
MIMIC	acc@1	**62.20 ± 2.40**	42.50	41.99 ± 0.61
	acc@3	**68.80 ± 2.00**	63.80	46.63 ± 1.99
	acc@5	**71.10 ± 2.00**	**72.90**	46.63 ± 1.99
Synthea	acc@1	**70.20 ± 1.70**	50.50	29.10 ± 0.80
	acc@3	**78.60 ± 2.50**	58.10	32.71 ± 0.35
	acc@5	**80.90 ± 1.10**	74.30	33.46 ± 0.63

Discussion. These results further confirm the superiority of Matchmaker over the ReMatch baseline, even when considering our re-implementation of the method. The lower performance of the re-implemented ReMatch compared to the original reported results could be due to differences in implementation details, such as the choice of text embeddings or variations not accounted for. However, it is important to note that even with these differences, Matchmaker consistently outperforms ReMatch (original) by a significant margin. The fact that Matchmaker achieves strong performance gains over both the original ReMatch and our re-implementation underscores the value of the novel techniques introduced in Matchmaker, such as the multi-stage language model program, the use of diverse candidate generators and the self-improvement mechanism through synthetic in-context examples.

D.4 IMPROVING PERFORMANCE: USE OF EXISTING MAPPINGS TO REMEDY ERRORS

Goal. To investigate the potential performance improvement in Matchmaker when leveraging readily available mappings to rectify errors between directly mapped attributes.

Experiment. In schema matching, certain attributes like source_value and concept_id have a direct mapping (e.g. in OMOP). If Matchmaker incorrectly maps the source attribute to the wrong target attribute (e.g., mapping to source_value instead of concept_id or vice versa), these errors can be easily rectified by leveraging the existing relationship.

To simulate this error correction, we implement a post-processing step where we adjust Matchmaker's predictions if the predicted target attribute has a direct mapping to the true target attribute. We apply this correction for all values of k and measure the resulting performance improvement.

Results. Figure 16 shows the accuracy gains across different values of k when applying the mapping correction. We observe consistent performance improvements across all k values. These results indicate that leveraging knowledge can indeed help rectify some of the errors made by Matchmaker.

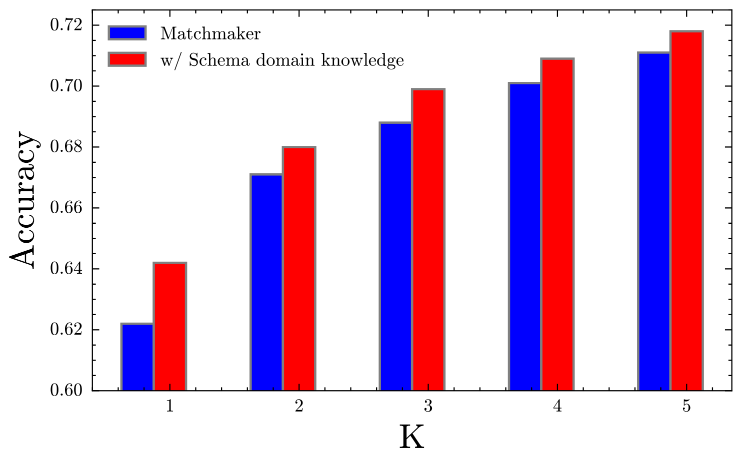

Figure 16: Performance improvement in Matchmaker when leveraging readily available mappings to correct errors between directly mapped attributes like source_value and concept_id.

Discussion. While the results demonstrate the potential benefit of using existing mappings for error correction, it is important to note that the performance gains are relatively modest compared to other strategies like human-in-the-loop deferral based on Matchmaker's confidence scores (as shown in the main text).

Moreover, the mapping correction relies on the availability of direct mappings between attributes, which may not always exist in practice. Therefore, while this approach can serve as a useful post-processing step, it should be seen as a complementary technique to be used alongside other strategies like human-in-the-loop for improving schema matching performance.

D.5 Comparison of Matchmaker on ontology matching tasks

While Schema matching and ontology matching are seemingly related, in reality they are completely different tasks. Specifically, schema and ontology matching fundamentally differ in their task and available information. Ontology matching leverages richer contextual info, including properties, axioms, rules, concept hierarchies and additional annotations. In contrast, schemas are sparser, with only attribute names, data types, descriptions and links.

Despite the difference for completeness we evaluate recent LLM ontology match methods using GPT-4 backbones to mirror Matchmaker namely: OLaLa (Hertling & Paulheim, 2023) and LLMs4OM (Giglou et al., 2024).

As shown in Table 9, Matchmaker outperforms these methods on both datasets.

Table 9: Accuracy@1: Matchmaker vs two LLM-based Ontology matching methods.

Method	**MIMIC**	**Synthea**
Olala	33.58 ± 0.47	31.53 ± 3.37
LLMs4OM	44.78 ± 0.41	64.50 ± 2.02
Matchmaker (Ours)	$\mathbf{62.20 \pm 2.40}$	$\mathbf{70.20 \pm 1.70}$

D.6 Detailed error analysis

Goal. We wish to understand different dimensions of Matchmaker's errors.

Discussion. We analyze the errors made by Matchmaker and find two categories of errors.

- 17% of Matchmaker's errors occur when attempting to find matches for source attributes that have no corresponding target attribute.
- The remaining 83% involve selecting incorrect but semantically related attributes. For these incorrect matches, we find a mean semantic similarity of 0.862 between the erroneous predicted attribute and the true target attribute. This confirms that Matchmaker typically selects attributes semantically close to the correct match rather than completely unrelated attributes.

These results further provide an understanding of Matchmaker's errors, as well as, showing how they can be addressed both via uncertainty deferral and remediation being easy to identify and correct.

D.7 Ranking ablation

Goal. Assess the importance of ranking to Matchmakers performance.

Results. Below we ablate the ranking step. The results shown highlight the importance of the re-ranking step towards achieving better accuracy@1.

Table 10: Comparison of Matchmaker models with and without ranking on MIMIC and Synthea datasets.

		Matchmaker (with ranking)	**Matchmaker (No ranking)**
MIMIC	Acc@1	62.20	57.00
	Acc@3	68.80	66.90
	Acc@5	71.10	71.10
Synthea	Acc@1	70.20	62.40
	Acc@3	78.60	77.20
	Acc@5	80.90	80.90

E Broader Impact

Schema matching is a critical step in data integration, enabling the creation of large, harmonized datasets that can be used to train machine learning models. The proposed Matchmaker approach, with its self-improving compositional language model program, has the potential to significantly accelerate and automate the schema matching process, thus facilitating the development of more accurate and robust ML models across various domains.

The importance and value of schema matching cannot be overstated, as integrating data from various sources such as different regions, organizations or applications is vital in many fields, including healthcare, finance, and e-commerce. By enabling the integration of data from disparate sources, schema matching plays a critical role in creating comprehensive, harmonized datasets that can provide a more complete picture of the domain under study. For example, in healthcare, integrating data from multiple hospitals can lead to more representative and diverse datasets, allowing researchers to identify patterns and insights that may not be apparent when analyzing data from a single institution.

Moreover, schema matching is not only valuable for specific domains but also for the machine learning community as a whole. By increasing the pool of available data (larger and more diverse) for training and validation, schema matching can contribute to the development of more accurate, robust, and generalizable ML models. Furthermore, having access to a larger pool of data can enable more rigorous validation and testing of ML models, allowing researchers to assess their performance across different subpopulations, time periods, and data sources. This, in turn, can lead to more reliable and trustworthy ML models that can be confidently applied in real-world settings.

Below we describe some positive implications that could be unlocked as schema matching approaches such as Matchmaker are used in practice. We also show some drawbacks with mitigation strategies.

Positive Impacts:

- Improved data integration: Matchmaker can help overcome the challenges of integrating data from heterogeneous sources, leading to the creation of larger, more comprehensive datasets. This can enable the development of more powerful and generalizable ML models.
- Accelerated research and discovery: By reducing the time and effort required for data integration, Matchmaker can accelerate research and discovery in fields, where data often resides in disparate databases with diverse schemas.
- Enhanced decision-making: The ability to train ML models on larger, more diverse datasets enabled by Matchmaker can lead to more accurate and reliable predictions, supporting better decision-making in various applications.

Potential Drawbacks and Mitigation Strategies:

- Overreliance on automated schema matching: While Matchmaker can significantly automate the schema matching process, it is not perfect and may make errors. Overreliance on automated methods without human oversight could lead to incorrect data integration. Mitigation: Matchmaker should be used as a tool to assist human experts in the schema matching process, rather than as a complete replacement. The paper demonstrates how Matchmaker can be effectively used with a human-in-the-loop approach, leveraging the strengths of both human expertise and automated methods.
- Propagation of errors: If Matchmaker introduces errors during the schema matching process, these errors can propagate downstream and affect the quality of the resulting integrated datasets and ML models. Mitigation: It is essential to implement rigorous validation and quality control measures to detect and correct errors introduced by Matchmaker. This can include manual spot-checks, automated consistency checks, and the use of domain-specific validation rules. Establishing a feedback loop to continuously monitor and improve Matchmaker's performance based on real-world usage can also help mitigate the propagation of errors.