SchemaTune: A Hybrid Framework Combining Transformer Fine-Tuning and Large Language Models for Schema Matching

Sayandeep Mitra
TCS Research, Tata Consultancy Services
India
mitra.sayandeep@tcs.com

Manasi Patwardhan
TCS Research, Tata Consultancy Services
India
manasi.patwardhan@tcs.com

Raveendra Kumar Medicherla
TCS Research, Tata Consultancy Services
India
raveendra.kumar@tcs.com

Abstract

Schema matching is a critical task in major enterprise tasks like data migration and integration, enabling interoperability across heterogeneous data sources. The current practice in the industry is mostly manual, which is time consuming, costly and error-prone. Automated approaches over the years often rely on heuristic or embedding-based similarity measures, which struggle with nuanced semantic relationships. In this work, we present SchemaTune, a novel framework that combines fine-tuned sentence transformer models using scarce data with various large language models (LLMs) for enhanced schema matching. SchemaTune first fine-tunes a sentence transformer on a small number of domain-specific schema pairs to capture contextual and structural similarities effectively. The candidate matches generated by the transformer are then re-ranked using LLMs using a novel dynamic prompt, leveraging their deep semantic reasoning capabilities to refine alignment decisions. Our experiments on real life datasets demonstrate that SchemaTune significantly outperforms state-of-the-art methods This hybrid approach illustrates the synergy between specialized embedding models and general-purpose LLMs, even locally executable small LLMs, paving the way for more accurate and scalable schema matching solutions.

Keywords

LLM, GPT, Schema Matching, Artificial Intelligence, Data Migration, Data Mapping, Llama3, embedding, fine tuning

1 Introduction

Schema Matching relates a set of fields of a *source* schema S to a set of fields of a *target* schema T without specifying the nature of the relationship [5]. It is an important first step in *semantic matching* of schemas, a key requirement in data integration, data migration, and data mediation. Schema matching is a knowledge intensive, challenging task due to variations in surface forms of the field names, table names, ambiguous semantics of the fields, data types between source and target schemas [5, 13]. To illustrate this challenge, consider databases two policy administration systems(PAS), Legacy PAS and New PAS shown in Figure 1. As part of software upgrade, the data of Legacy PAS required to be migrated to database of New PAS. For the field `INDEM_AMT (Indemnity Paid Amount)` of table `CLAIM_SETTLEMENT` of new PAS, a data analyst identified two fields `LOSS_AMT (Loss Amount)` and `PAID_AMT (Paid Amount)` of table `POL_DTL` of Legacy PAS, as *matching* fields. Automatically discovering such a semantic match is non-trivial. Schema matching problem is a widely researched topic [2, 13]. However, due to inherent complexity of the problem, only a few automatic tools are developed and used in practice. Most schema matching tools available in market support only manual matching. Recently, Large Language Models (LLM) have gained popularity as a tool to solve the schema matching task [1, 17, 19].

We have developed two variants of semantic matching tools to *assist* our analysts in software and data service business to support data migration and data integration projects. The first tool, Auto-MI [11, 13], learns with probabilities of *concepts* and relationships of database fields used in historical schema with the help of a domain specific knowledge graph and discover matching new concepts and relationships using the learned model. The second tool, Mapgen [1] used sentence transformer [15] to find the match by sentence similarity, rank the matches based on similarity score, and use LLMs to improve the ranking. MapGen approach is cost effective and has better performance in interactive tool setting when compared to pure play LLM based matching [17, 19] due to faster sentence transformers and fewer number of calls to LLMs. Both these approaches leveraged a novel approach called *semantic type* introduced in [13], a marker assigned to every field by capturing the semantic class of the field in the form of data type and making it independent of physical type. Semantic types aids in filtering out improbable candidate matches, e.g., a field assigned semantic type `MONEY` is unlikely to have a match with a field having semantic type `DATE`.

Each of these tools suffered with different issues when they are deployed in production environment. While Auto-MI is lightweight and cost-effective, its dependency on right knowledge graph, scarcity of historical matching, low Recall@k, and inconsistent performance across schemas are the key issues. Recall@k signifies the tool's performance in identifying the *correct* match for a given target field within k suggested source fields. Also, learned model in one organization can not be redeployed into another organization due to data confidentiality agreements. This means, an AutoMI model needs to learn from scratch for each customer deployment leading to inferior model performance forever. On the other hand, the Mapgen's performance is better than Auto-MI, and it has no dependency on knowledge graph and historical matching data. But, it also suffers from unsatisfactory Recall@k for deployment in enterprises due to lack of organization specific data. A low Recall@k impacts the cost-benefit of the tool negatively as it requires significant effort from data analyst to manually identify matching fields. The other key requirement from our users is that the tool should be deployable in environments with varied data privacy policies, hence the access to external LLMs always cannot be assumed. The tool should be self-contained in the sense, the required sentence transformer model and LLM need to be potentially bundled into the tool. This means, a potential tool can only use LLMs with smaller number of parameters (SLM) that is locally executable and requires less computing resources (< 16 GB GPU) [18, 20].

EDBT '26, Tampere (Finland)

Above factors lead us to *fine-tune* the models, as the only viable solution to improve the performance. While technically it is possible to fine-tune the SLM, but, due to cost effectiveness and data availability we decided to fine-tune the smaller sentence embedding model. The other influential factor of this choice is that the fine-tuning job may run in customer environment, which might have a limit on computing resources required for fine-tuning. In this paper we present *SchemaTune*, fine-tuning a small sentence embedding model using a small set of focused training data which is used to generate a ranked list of matches. This ranked list is then passed on to an LLM agent that re-ranks it using additional context. We introduce a dynamic contextualized prompt, which makes use of database key relationships (Primary Key, Foreign Keys), table information and the current matching results dynamically to allow for improved results, even with SLMs. SchemaTune is designed to have minimal user involvement and computing resources.

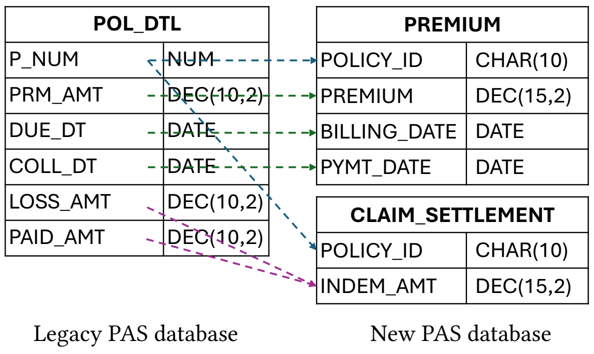

Figure 1: Schema matching of databases of two Policy Administration Systems(PAS)

The main contributions of this work are:

- A task specific and domain adapted embedding model, fine-tuned with scarce training data available, which leads to improvement in the performance at part with larger LLMs at a reduced cost.
- A new LLM re-ranking agent yields further improvement, due to the introduction of a new prompt which dynamically provides relevant schema and table information along with the current matching information to the LLM/ SLM which provides additional context.
- State of the art results, which shows how using SLMs in SchemaTune give comparative results with proprietary LLMs.
- Key lessons learned while selecting the sentence transformer models, selecting various fine-tuning parameters based on different trade-offs.

The rest of the sections are arranged as follows. In Section 2, we discuss the related works in this space, followed by Section 3 where we present our problem statement and show the approach we have taken to empirically arrive at the underlying framework of SchemaTune. Section 4 discusses the evaluation results of our solution and how they perform against the SOTA approaches, followed by our conclusion and possible future work in Section 5.

2 Related Work

Schema matching has traditionally been a manual endeavour which is a time consuming and error prone process [6] and still remains the dominant practice in industry. Lack of high-quality datasets [9] has been a major challenge, which has limited the applicability of ML-based solutions [3, 4] to toy datasets. SMAT [21] and LSM [22] have taken advancements in natural language processing and facilitated semantic matching between source and target schemas. However, they require significant manual intervention for data tagging making them unsuitable for enterprise practice.

More recent works by [14, 16, 17] have leveraged the power of LLMs to make significant strides in schema matching. [17] and [14] evaluates on openly available health datasets, which modern LLMs are well-versed and potentially trained. Furthermore, these studies have not incorporated table and field enrichment while identifying matches, which can lead to a loss of context. Additionally, [14] do not address the challenge of context size for large-scale industrial schemas. Rematch [17] uses commercial LLMs to achieve their results, which differs from our view of creating a localised solution for both cost and data privacy reasons. It also leverages a static prompt for re-ranking which we want to improve upon by our dynamic prompt. MatchMaker [16] provides a dynamic prompt with a chain of LLM calls to provide superior results. Their prompt, however does not consider the schema relationships, which experts agree is critical to solving industry level schema matching. Multiple proprietary LLM calls also means a heightened cost factor and data privacy issues, which like ReMatch, is different to our objective.

Magneto [10] is the closest solution to our work where they fine-tune a sentence embedding model, generate a set of matching results and then uses either local or commercial LLMs to improve them. However, there exists some key differences with our work. Magneto work under the assumption that matching data is not provided and instead rely upon schema data to perform the schema matching. In enterprise, access to schema data is extremely difficult due to confidentiality reasons. Magneto also, similar to ReMatch, uses a static prompt which does not consider schema relationships.

3 Our Approach

This section describes the matching problem formally. Then we describe different steps followed to prepare the data, selection of suitable sentence transformer model, different parameters for fine-tuning, and factors considered while creating the prompt for LLM based re-ranking.

3.1 Prerequisites

We have a source database schema $\mathcal{S} = \{s_1, \ldots, s_m\}$, and a target database schema $\mathcal{T} = \{t_1, \ldots, t_n\}$, where s_i is the i^{th} field in the source schema and t_j is the j^{th} field in the target schema.

We say a matching function μ is

$$\mu : \mathcal{T} \to \mathcal{P}(\mathcal{S}),$$

where $\mathcal{P}(.)$ is the power set.

Thus for each target field $t_j \in \mathcal{T}$,

$$\mu(t_j) \subseteq S$$

represents the set of source fields matching to t_j.

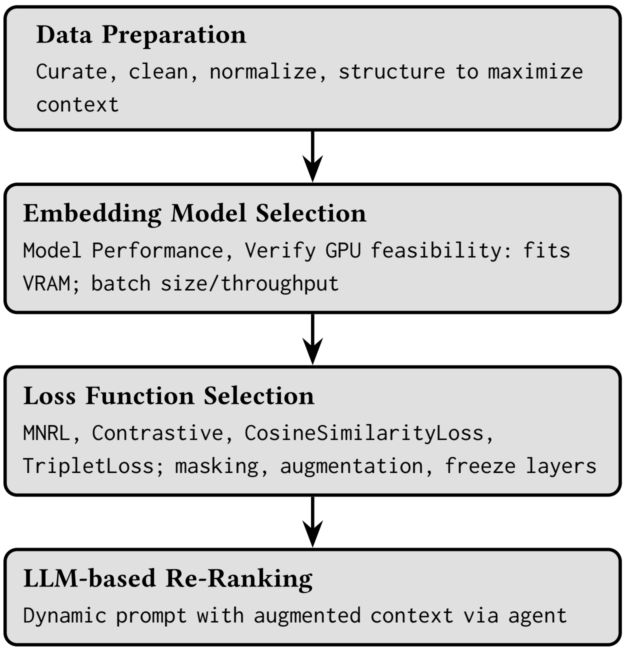
Data Preparation
Curate, clean, normalize, structure to maximize context
Embedding Model Selection
Model Performance, Verify GPU feasibility: fits VRAM; batch size/throughput
Loss Function Selection
MNRL, Contrastive, CosineSimilarityLoss, TripletLoss; masking, augmentation, freeze layers
LLM-based Re-Ranking
Dynamic prompt with augmented context via agent

Figure 2: Detailed workflow of our evaluations

3.2 Problem Statement

In real life enterprise, discovering μ automatically is a hard problem. Given source and target schemas $\mathcal{S}, \mathcal{T}$, and a sample *ground truth* matches M^* (given by users) which is a *small* sample drawn from μ, our objective is to learn a matching function μ' that is close to μ, with respect to *coverage*. We define μ' as

$$\mu' : \mathcal{T} \to \mathcal{L}(\mathcal{P}(\mathcal{S})), \qquad \mu'(t_j) = \big[(X_i, \rho_i)\big]_{i=1}^{k},$$
$$X_i \subseteq \mathcal{S}, \quad \text{and } \rho_i \in \mathbb{R} \text{ is a } \textit{score} \text{ such that } \rho_1 \geq \rho_2 \geq \cdots \geq \rho_k,$$
$$\mathcal{L}(\mathcal{P}(\mathcal{S})) \text{ denotes a } \textbf{ranked list} \text{ over subsets of } \mathcal{S}$$

Additionally, μ' should have the following properties:

(1) **High coverage:**
$$\text{Recall@}k = \frac{|M'|}{|M|} \approx 1,$$
where M' and M are the set of matchings induced by μ' and μ respectively.

(2) **High precision:**
$$\forall t \in \mathcal{T},\ \mu'(t) \approx \mu(t),$$
ensuring that the predicted set of source fields for each target field is close to the true set of source, thereby reducing false positives.

(3) **High ranking quality:** Correct matches should appear with higher ρ_i values, so that Recall@k is maximized when considering the top-k ranked candidates.

Our Workflow

Figure 2 presents steps of SchemaTune as a workflow. We begin by cleaning and analyzing the provided data and arranging it in suitable format for our evaluations, which captures maximum context. Post that, we choose our embedding model and loss function for training to proceed with based on both empirically and our requirements. Finally, we implement re-ranking using an LLM agent leveraging a newly designed dynamic prompt aided by additional context and information.

3.3 Data Preparation

Our target schema has 5658 fields in 371 unique tables, while we have three different source schemas, S1, S2 and S3 having 150, 100, 850 tables and 2800, 1700, 10000 fields respectively. The data in the source schemas are encrypted, a standard practice in enterprises. All these schemas are part of real life business schema matching projects, where the target schema is of enterprise standard while the three source schemas are legacy schemas which are to be upgraded to the target schema. Due to confidentiality reasons, the datasets cannot be made public. Descriptions and the format of data is provided for the fields where applicable. We have 1328 verified schema matches, including m:1 matches, i.e., M^* from the three source schemas together to the fields of the target schema

The table and field names are provided in abbreviated cryptic forms, a standard representation in enterprises, e.g., PRM_AMT, INDEM_AMT, etc. in Figure 1. We term these as *physical* representations. We represent each field from both source and target in M^* in the following *physical* form for our evaluations and term the corresponding matching dataset as M_p^*.

Table_Physical_Name.Column_Physical_Name

Intuitively, it feels using the *physical* representations to calculate embeddings and querying LLMs raises the risk of ambiguity, especially when we are dealing with enterprise specific terms. To mitigate this, we use techniques explained in [11] and [1], where the *physical* names are *enriched* into meaningful human understandable forms which we term as the *logical* names. As an example, `PRM_AMT` is expanded to *Premium Amount*. We represent each field from both source and target in M^* in the following enriched format, to capture the maximum semantics of the field. This representation ensures that we have a clear distinction between the table and field names, and we term the dataset as M_e^*.

Column_Physical_Name (*Column_Logical_Name*) of *Table_Physical_Name* (*Table_Logical_Name*)

We experiment with both M_p^* and M_e^*, and empirically determine which format allows for better fine-tuning of the embedding model and LLM re-ranking. In future, when we use the term M^*, we mean both M_p^* and M_e^*, unless stated otherwise. We build our `test set` from M^*, by identifying 136 matches which are for critical target fields. From the three sources we get 39, 56 and 41 matches, respectively. We split the remaining 1192 matches from M^* in a $70 : 30$ ratio in a random manner for our `training set` and our `validation set`. We ensure that there is no overlap between the training, validation and test set.

3.4 Model Selection

In our earlier work [1], we used the *all-mpnet-base-v2* [7] as our sentence embedding model, due to it's compact size and wide usage. To be certain that we are using the most suitable model for our fine-tuning task, we compared the performance of multiple models which featured in the MTEB [12] leaderboard, on a subset of our validation set. The models were selected based on

- their ability to run efficiently on GPU-less systems.
- their storage size (due to the nature of our task, models which are smaller in size are preferred).
- their performance in Semantic Textual Similarity (STS) tasks. We look at STS scores because schema matching is essentially a semantic similarity task, and models with

strong STS scores tend to provide tighter clusters of conceptually related attributes, and handle synonyms and abbreviations more robustly.

Based on these criteria, we shortlisted the following models,

(1) sentence-transformers/all-mpnet-base-v2[1]
(2) mixedbread-ai/mxbai-embed-large-v1[2]
(3) NovaSearch/stella_en_1.5B_v5[3]
(4) jinaai/jina-embeddings-v3[4]

Out of these four models, the *all-mpnet-base-v2* is the smallest model whereas the *stella_en_1.5B_v5* is the largest model.

3.5 Identifying Loss Functions

As discussed earlier, for the task of fine-tuning the model for schema matching, we have positive pairs of matching only, provided in M^* and no explicit negatives.

3.5.1 MultipleNegativesRankingLoss. In MultipleNegativesRankingLoss (MNRL) [8], given a batch of N target field-matching source field pairs (t_i, s_i) where $i = 1, \dots, N$, $s \in \mathcal{S}$ the loss for an anchor t_i considers its corresponding positive s_i as the one true match, and all other positives s_j (where $j \neq i$) and all other negatives (if provided) in the batch as negative examples. The goal is to maximize the similarity between s_i and t_i while minimizing the similarity to all other samples in the batch.

$$L = -\frac{1}{N} \sum_{i=1}^{N} \log \frac{e^{\text{sim}(s_i, t_i) \cdot \text{scale}}}{\sum_{j=1}^{N} e^{\text{sim}(s_i, t_j) \cdot \text{scale}}} \tag{1}$$

where:

- N is the batch size.
- t_i is the i-th anchor embedding.
- s_i is the i-th positive embedding (the only true positive for t_i in the batch).
- $\text{sim}(u, v)$ is the similarity function (commonly cosine similarity or dot product).
- scale is a temperature scaling factor, which is multiplied by the similarity scores.

MNRL works with only positive samples, a straight fit to our dataset which contains only positive matches. However, due to the reduced size for the number of training sample, there is a possibility of overfitting of the model. To minimize this risk, we perform the following variations while using MNRL.

Freezing all but the final 1–2 layers: A common approach to prevent *overfitting*, which is a possible concern due to the relatively small size of our training data. We train our model by updating only the final and the final two layers of the model using MNRL.

Generating aliases: We augment our dataset using GPT-4o, prompting it to create aliases of table names and field names, using field metadata when available, with the aim of improving the model's understanding of semantics and context, rather than focusing on exact matches of words.

Masking: We mask certain table names at random in our training data to act as a regularization technique to prevent overfitting. This forces the model to learn more generalizable features, by focusing less on the repetitive words in the table names, and instead focus on the field relationships.

3.5.2 Explicit Negatives. To improve the existing results, we add to our training dataset explicit negatives by considering all source fields which are not guaranteed matches, i.e., $\{\mathcal{S} - s\}$ for a particular target field t, $\forall \langle t, s \rangle \in M^*$. This creates a large number of negatives which makes our dataset extremely skewed towards one side. We randomly sample negatives for each target field and maintain the ratio of the number of positives and negatives in our training set as $1 : 1$ and $1 : 2$, and term these datasets as M^*_{p,neg_1}, M^*_{p,neg_2}, M^*_{e,neg_1} and M^*_{e,neg_2}. Post this we additionally consider the following three loss functions,

- *ContrastiveLoss:* Given a batch of N target–source pairs (t_i, s_i) with binary labels $y_i \in \{0, 1\}$ indicating whether the pair is a true match ($y_i = 1$) or a negative ($y_i = 0$), the objective is to minimize the distance for positives and enforce a margin for negatives:

$$L = \frac{1}{N} \sum_{i=1}^{N} \left[y_i \cdot d(t_i, s_i)^2 + (1 - y_i) \cdot \max\left(0,\ m - d(t_i, s_i)\right)^2 \right] \tag{2}$$

  where:
  - N is the batch size.
  - t_i is the i-th anchor (target) embedding.
  - s_i is the i-th source embedding.
  - y_i is 1 for positive pairs and 0 for negative pairs.
  - $d(u, v)$ is a distance function (e.g., Euclidean; for cosine one may use $d(u, v) = 1 - \text{sim}(u, v)$).
  - $m > 0$ is the margin hyperparameter.
- *CosineSimilarityLoss:* This encourages high cosine similarity for positives and pushes negatives below a margin m:

$$L = \frac{1}{N} \sum_{i=1}^{N} \left[y_i \cdot \left(1 - \text{sim}(t_i, s_i)\right) + (1 - y_i) \cdot \max\left(0,\ \text{sim}(t_i, s_i) - m\right) \right] \tag{3}$$

  where:
  - N is the batch size.
  - t_i is the i-th anchor (target) embedding.
  - s_i is the i-th source embedding.
  - y_i is 1 for positive pairs and 0 for negative pairs.
  - $\text{sim}(u, v)$ is cosine similarity (often computed on normalized embeddings).
  - $m \in [-1, 1]$ is the similarity margin for negatives.
- *TripletLoss:* This loss function operates on triplets $\left(t_i, s_i^+, s_i^-\right)$, where s_i^+ is the positive (true match to t_i) and s_i^- is a negative. It enforces the positive to be closer than the negative by at least margin m:

$$L = \frac{1}{N} \sum_{i=1}^{N} \max\left(0,\ d(t_i, s_i^+) - d(t_i, s_i^-) + m\right) \tag{4}$$

  where:
  - N is the number of triplets in the batch.
  - t_i is the i-th anchor (target) embedding.
  - s_i^+ is the i-th positive source embedding.
  - s_i^- is the i-th negative source embedding.
  - $d(u, v)$ is a distance function (e.g., Euclidean or $1 - \text{sim}(u, v)$ with cosine).
  - $m > 0$ is the margin hyperparameter.

As a variation, instead of random sampling of our negatives, we further refine our fine-tuning procedure by computing the *hard positives* and *hard negatives* after each epoch using the

[1]https://tinyurl.com/2datnscr
[2]https://tinyurl.com/2xee6v9t
[3]https://tinyurl.com/4h3ae2zd
[4]https://tinyurl.com/mwuer8nt

updated checkpoint. We assign a higher weight to these samples and make use of our custom sampling strategy by using `WeightedRandomSampler`, which prioritizes the hard samples while also selecting normal samples for training. This ensures that the model learns from the greater challenging scenarios more often. We term these datasets with added negatives $M^*_{p,\ hard}$ and $M^*_{e,\ hard}$, for the physical and logical representations respectively.

3.6 LLM Re-ranking

We introduce a new prompt for our re-ranking, where along with the ranked list, we add the additional context based on schema relationships and the way legacy schemas are usually defined. In our earlier work in [1], we provided just the ranked list along with the field descriptions to be re-ranked by LLMs. This lacked any information about how the target field and the source fields are present in their respective tables and schema, i.e., schema relationships, a critical aspect during schema matching was missing for the LLM to analyze.

To address this shortcoming, we `dynamically` include in the re-ranking prompt, for a particular target field how the concerned source and target tables are linked by Primary keys (PK) and Foreign keys (FK), the overall context of the tables concerned and make use of the non-normalized form of the source tables, by providing related confirmed matchings to improve the re-ranking performance. This `dynamic` update ensures that for a particular target field, the LLM is aware of its' relevant schema relationships. We discuss them in more detail.

Key Relationships: We put in the Primary keys of the tables of both the target field and the source fields which are to be re-ranked. Additionally, we put in any foreign keys that are present and to which other tables they are linked to on both the source and the target side. This provides LLM with knowledge about the complete data flow, context and dependency between the tables, and in turn the role of the fields concerned. Consider the diagram in Figure 3.
Attachments.MessageID and *Attachments.EntityID* both can be possible matches with *EmailAttachments.EmailID*. The LLM can easily match *Attachments.EntityID* to the target field, as they are both ID fields and both reference an entity related to *email*. Only by means of the FK relationships, can we deduce that the correct match is with *Attachments.MessageID* since *EmailAttachments.EmailID* has a FK relationship with *Emails.EmailID* and that EntityID holds the ID of the entity that sends the email.
Additionally, these key relationships are integral to deducing $m : 1$ conditional matchings as well by the LLM. For example, consider the matching *Attachments.EntityID* and *Attachments.EntityType* with *Client.ClientID*. Normally, the likely match is *Attachments.EntityID* with *Client.ClientID*. However, we now have the key relationships which establish *EmailAttachments→Emails→ Policies→Clients/Agents/Company*, i.e., Policy is linked to three different ID columns. LLM with this knowledge suggests that there is a possible $m : 1$ scenario which requires the conditional compliance of *Entity Type* to copy the value of *Entity ID* to *Client ID*.

Additional fields of tables: For every field t which belongs to a target table T, that we are trying to generate matches for and a source field s belonging to a table S, which is a potential match we put in $\{T - t\}_k$ and $\{S - s\}_k$, i.e., a set of k fields from these tables which are dissimilar to one another in the prompt. This is designed to provide unique additional clues about the tables, which allows LLM to have a more clearer understanding of the table's context and it's overall role in the schema.

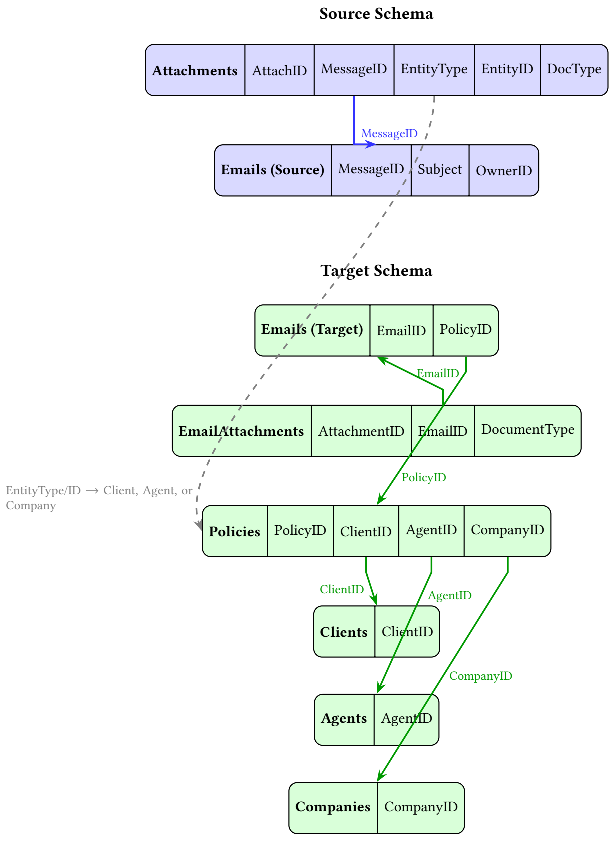

Figure 3: Tables showing key relationships on both source and target schema

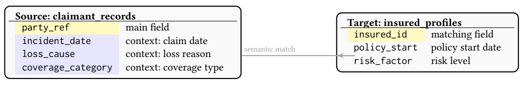

Figure 4: Schema matching between `party_ref` (source) and `insured_id` (target) with extra context fields aiding semantic equivalence.

Consider the scenario shown in Figure 4. `party_ref` is not a straightforward match with `insured_id`, as party reference can refer to people other than the insured person. The other fields `incident_date`, `loss_cause` and `coverage_category`, together conveys that this table focuses on the insured-exposure space, as we are taking note of when the incident happened, the cause of the loss and the coverage applied. This additional information removes any ambiguity for the LLM agent.
Consider a separate scenario, where the target field *payout_amount* is present with *incident_date* and *lost_cause*, implying that it is an outcome of a claim, i.e., the money paid. Source field *limit_amount* appears with *policy_start* and *risk_factor* in the source table, suggesting it is a policy

design parameter - the coverage gap. Without the additional fields, any model might focus on the knowledge that these two fields *limit_amount* from source and *payout_amount* from target are money related fields and can be related. However, the presence of the neighbouring fields from the same table allows the LLM to understand that even though they are both money related fields but likely not a match, hence ensuring during re-ranking it is ranked at a lower position.

Matchings in current context: If we are trying to match a field t in table T and there exists another field t' in T which has already been matched to a source table S, then we provide information of S in the prompt. We also provide information of all tables S' that are related to S via PK-FK relationships. Finally, we add information about T', set of tables which are related to T via PK-FK relationships and each of their corresponding S'. This dynamic update to our prompt is based on input from past schema matching projects that fields of a particular target table are more likely to be matched to a small set of related source tables, as in most practical cases of schema matching in enterprises, the legacy source tables are non-normalized, which leads to related information being limited to a small set of source tables.

4 Evaluation Results

We present the results of the evaluations that we have performed in Section 3, the lessons learnt from them and how we use them to make our choices for the most optimal performing SchemaTune. We also compare it with the state of the art schema matching tool [10].

4.1 Model Selection

We compute how the four different models that we selected in Section 3.4.

We ran each of these models on a subset 40 target fields of our validation set from both datasets M_p^* and M_e^*, against the corresponding source schema, to see which model performed best. Table 1 shows the summary of the results. We present Recall@k scores, which shows for how many target fields, the correct matches were present in the top k suggested matches.

Table 1: Performance of different models on subset of our dataset

	all-mpnet-base-v2	mxbai-embed-large-v1	stella_en_1.5B_v5	jina-embeddings-v3
M_p^*	Recall@1: 0.08 Recall@5: 0.18 Recall@20: 0.53	Recall@1: 0.08 Recall@5: 0.3 Recall@20: 0.55	Recall@1: 0.05 Recall@5: 0.13 Recall@20: 0.2	Recall@1: 0.08 Recall@5: 0.23 Recall@20: 0.45
M_e^*	Recall@1: 0.13 Recall@5: 0.28 Recall@20: 0.6	Recall@1: 0.15 Recall@5: 0.38 Recall@20: 0.65	Recall@1: 0.08 Recall@5: 0.18 Recall@20: 0.4	Recall@1: 0.08 Recall@5: 0.25 Recall@20: 0.45

For both the datasets, *mxbai-embed-large-v1* and *all-mpnet-base-v2* outperformed the other two models, with mxbai outperforming mpnet marginally. Considering the sizes of these two models and the computing limitations (16GB GPU) that was discussed in Section 1, we only consider *all-mpnet-base-v2* for further evaluations.

4.2 Selecting Loss Functions

We first fine-tune the model with MultipleNegativesRankingLoss as the loss function and using both M_p^* and M_e^* datasets. Table 2 shows that our fine-tuning improves the performance of the model on the test set significantly. It is to be noted that using M_p^* dataset, we get poor results even post fine-tuning. For the M_e^*, even with the overall improvement, on closer inspection, we see that for three target fields, the fine-tuned model pushes them from within Recall@5 to Recall@20, i.e., the performance drops for these three fields. On closer analysis of these three target fields, we see the reason being that their table names contain some words which have a high occurrence in the training data in particular combinations, which causes the model to overfit for them. As an example, consider that *marketing* is jointly present with *party* multiple times in the training set which causes them to be pulled closer in the embedding space post fine-tuning. Now when the fine-tuned model generates a ranked list, it will tend to rank source fields containing *party* and its associated words higher than source fields containing words actually related to *marketing*. This is a common problem that is present when the training dataset is not adequate and varied.

Table 2: fine-tuned all-mpnet-base-v2 using MNRL vs pre-trained all-mpnet-base-v2

		all-mpnet-base-v2	all-mpnet-base-v2 (ft-mnrl)
S1	M_p^*	Recall@1: 0.1 Recall@5: 0.31 Recall@20: 0.38	Recall@1: 0.23 Recall@5: 0.36 Recall@20: 0.46
	M_e^*	Recall@1: 0.15 Recall@5: 0.33 Recall@20: 0.49	Recall@1: 0.36 Recall@5: 0.59 Recall@20: 0.69
S2	M_p^*	Recall@1: 0.25 Recall@5: 0.41 Recall@20: 0.61	Recall@1: 0.32 Recall@5: 0.55 Recall@20: 0.71
	M_e^*	Recall@1: 0.39 Recall@5: 0.63 Recall@20: 0.84	Recall@1: 0.46 Recall@5: 0.77 Recall@20: 0.95
S3	M_p^*	Recall@1: 0.15 Recall@5: 0.34 Recall@20: 0.61	Recall@1: 0.27 Recall@5: 0.46 Recall@20: 0.76
	M_e^*	Recall@1: 0.17 Recall@5: 0.41 Recall@20: 0.63	Recall@1: 0.51 Recall@5: 0.78 Recall@20: 0.9

The results of the different variations of MNRL that we tested to mitigate the above challenges are as follows:

- **Freezing all but the final 1–2 layers:** While the drop for these three fields is reduced, this comes at a cost of rank drops for the rest of the target fields.
- **Generating aliases:** We however, fail to see much improvement in the results. We put this lack of improvement down to the fact that the table names and field names are tightly related in an enterprise domain, where the focus is on a particular domain, causing aliases to be limited as well.
- **Masking:** We apply three different ratios to mask the table names, but for all of them the performance degrades showing that for our small training set, masking is not an efficient approach.

Explicit Negatives: The results of the three different loss functions when using explicit negatives are as follows. When choosing the negative samples randomly, for both M_{e,neg_1}^* and M_{e,neg_2}^*, the results of all these three loss functions showed an average drop of 23% and 29% respectively for all three sources as compared to the results shown in Table 2, while M_{p,neg_1}^* and M_{p,neg_2}^* showed average drops of 34% and 37%. For the second variation, when we computed the *hard positives* and *hard negatives* after each epoch,the fine-tuned models using this approach showed an average drop of 11% and 18% for $M_{e,hard}^*$ and $M_{p,hard}^*$ respectively.

Even MNRL with the explicit negatives did not show an improvement in performance.

Based on these results, we choose MNRL as our loss function using only M^* with no additional negatives. In all our evaluations we see that the expanded representation, i.e., M_e^* performs better than the physical representation M_p^*. We consider only the enriched logical representation in our solution.

4.3 Performance of new prompt for LLM Agent

Table 3: Performance of 3 datasets with all-mpnet-base-v2 model (original and fine-tuned version) with both prompts using GPT-4

	mpnet + GPT-4 (Prompt_1)	mpnet + GPT-4 (Prompt_2)	mpnet (ft) + GPT-4 (Prompt_1)	mpnet (ft) + GPT-4 (Prompt_2)
S1 (39)	Recall@1: 0.33 Recall@5: 0.44 Recall@20: 0.79	Recall@1: 0.41 Recall@5: 0.51 Recall@20: 0.82	Recall@1: 0.49 Recall@5: 0.69 Recall@20: 0.85	**Recall@1: 0.54** **Recall@5: 0.72** **Recall@20: 0.9**
S2 (56)	Recall@1: 0.59 Recall@5: 0.73 Recall@20: 0.91	Recall@1: 0.66 Recall@5: 0.86 Recall@20: 0.91	Recall@1: 0.64 Recall@5: 0.84 Recall@20: 0.95	**Recall@1: 0.71** **Recall@5: 0.91** **Recall@20: 0.96**
S3 (41)	Recall@1: 0.34 Recall@5: 0.66 Recall@20: 0.95	Recall@1: 0.51 Recall@5: 0.78 Recall@20: 0.93	Recall@1: 0.61 Recall@5: 0.85 Recall@20: 0.95	**Recall@1: 0.73** **Recall@5: 0.90** **Recall@20: 0.95**

Table 4: Performance of 3 datasets with all-mpnet-base-v2 model (original and fine-tuned version) with both prompts using GPT-4o

	mpnet + GPT-4o (Prompt_1)	mpnet + GPT-4o (Prompt_2)	mpnet (ft) + GPT-4o (Prompt_1)	mpnet (ft) + GPT-4o (Prompt_2)
S1 (39)	Recall@1: 0.33 Recall@5: 0.46 Recall@20: 0.79	Recall@1: 0.41 Recall@5: 0.51 Recall@20: 0.79	Recall@1: 0.46 Recall@5: 0.72 Recall@20: 0.85	**Recall@1: 0.54** **Recall@5: 0.74** **Recall@20: 0.90**
S2 (56)	Recall@1: 0.59 Recall@5: 0.73 Recall@20: 0.91	Recall@1: 0.66 Recall@5: 0.86 Recall@20: 0.91	Recall@1: 0.71 Recall@5: 0.84 Recall@20: 0.96	**Recall@1: 0.71** **Recall@5: 0.89** **Recall@20: 0.96**
S3 (41)	Recall@1: 0.34 Recall@5: 0.66 Recall@20: 0.95	Recall@1: 0.54 Recall@5: 0.78 Recall@20: 0.93	Recall@1: 0.61 Recall@5: 0.85 Recall@20: 0.95	**Recall@1: 0.73** **Recall@5: 0.9** **Recall@20: 0.95**

Table 5: Performance of 3 datasets with all-mpnet-base-v2 model (original and fine-tuned version) with both prompts using GPT-4o mini

	mpnet + GPT-4o mini (Prompt_1)	mpnet + GPT-4o mini (Prompt_2)	mpnet (ft) + GPT-4o mini (Prompt_1)	mpnet (ft) + GPT-4o mini (Prompt_2)
S1 (39)	Recall@1: 0.33 Recall@5: 0.44 Recall@20: 0.77	Recall@1: 0.38 Recall@5: 0.46 Recall@20: 0.82	Recall@1: 0.46 Recall@5: 0.72 Recall@20: 0.82	**Recall@1: 0.51** **Recall@5: 0.72** **Recall@20: 0.9**
S2 (56)	Recall@1: 0.59 Recall@5: 0.71 Recall@20: 0.91	Recall@1: 0.63 Recall@5: 0.84 Recall@20: 0.91	Recall@1: 0.64 Recall@5: 0.84 Recall@20: 0.95	**Recall@1: 0.68** **Recall@5: 0.88** **Recall@20: 0.96**
S3 (41)	Recall@1: 0.32 Recall@5: 0.66 Recall@20: 0.95	Recall@1: 0.51 Recall@5: 0.76 Recall@20: 0.93	Recall@1: 0.56 Recall@5: 0.83 Recall@20: 0.95	**Recall@1: 0.71** **Recall@5: 0.93** **Recall@20: 0.95**

Table 3,4,5,6,7 shows the performance of our tool with the following four combinations using the following five LLMs - GPT-4, GPT-4o, GPT-4o-mini, Llama 3.1-8B and Llama 3.1-70B.

(1) pre-trained sentence embedding model and previous prompt [11].
(2) pre-trained sentence embedding model with our proposed new prompt.

Table 6: Performance of 3 datasets with all-mpnet-base-v2 model (original and fine-tuned version) with both prompts using Llama 3.1-8B

	mpnet + Llama 3.1-8B (Prompt_1)	mpnet + Llama 3.1-8B (Prompt_2)	mpnet (ft) + Llama 3.1-8B (Prompt_1)	mpnet (ft) + Llama 3.1-8B (Prompt_2)
S1 (39)	Recall@1: 0.26 Recall@5: 0.38 Recall@20: 0.74	Recall@1: 0.29 Recall@5: 0.39 Recall@20: 0.73	Recall@1: 0.44 Recall@5: 0.66 Recall@20: 0.73	**Recall@1: 0.46** **Recall@5: 0.66** **Recall@20: 0.8**
S2 (56)	Recall@1: 0.57 Recall@5: 0.71 Recall@20: 0.89	Recall@1: 0.59 Recall@5: 0.75 Recall@20: 0.89	Recall@1: 0.64 Recall@5: 0.82 Recall@20: 0.93	**Recall@1: 0.68** **Recall@5: 0.88** **Recall@20: 0.95**
S3 (41)	Recall@1: 0.32 Recall@5: 0.61 Recall@20: 0.95	Recall@1: 0.46 Recall@5: 0.73 Recall@20: 0.9	Recall@1: 0.56 Recall@5: 0.8 Recall@20: 0.95	**Recall@1: 0.68** **Recall@5: 0.9** **Recall@20: 0.95**

Table 7: Performance of 3 datasets with all-mpnet-base-v2 model (original and fine-tuned version) with both prompts using Llama 3.1-70B

	mpnet + Llama3.1-70B (Prompt_1)	mpnet + Llama3.1-70B (Prompt_2)	mpnet (ft) + Llama3.1-70B (Prompt_1)	mpnet (ft) + Llama3.1-70B (Prompt_2)
S1 (39)	Recall@1: 0.33 Recall@5: 0.41 Recall@20: 0.79	Recall@1: 0.41 Recall@5: 0.51 Recall@20: 0.79	Recall@1: 0.49 Recall@5: 0.69 Recall@20: 0.82	**Recall@1: 0.54** **Recall@5: 0.69** **Recall@20: 0.90**
S2 (56)	Recall@1: 0.59 Recall@5: 0.73 Recall@20: 0.91	Recall@1: 0.64 Recall@5: 0.86 Recall@20: 0.91	Recall@1: 0.64 Recall@5: 0.82 Recall@20: 0.95	**Recall@1: 0.71** **Recall@5: 0.89** **Recall@20: 0.96**
S3 (41)	Recall@1: 0.34 Recall@5: 0.66 Recall@20: 0.76	Recall@1: 0.51 Recall@5: 0.80 Recall@20: 0.93	Recall@1: 0.61 Recall@5: 0.85 Recall@20: 0.95	**Recall@1: 0.73** **Recall@5: 0.93** **Recall@20: 0.95**

(3) fine-tuned sentence embedding model and previous prompt.
(4) fine-tuned sentence embedding model and proposed new prompt.

In these tables, Recall@k shows the ratio of target fields for which the correct matches were found within the top k proposed matches. We consider Llama 3.1-8B to be our SLM.

When compared with Table 2, we see that our fine-tuned all-mpnet-base-v2 model performs competitively with our previous solution of using the pre-trained all-mpnet-base-v2 version and LLM re-ranking, especially at Recall@5 scores. For all the different LLMs, the results show that the fine-tuned model outperforms the pre-trained `all-mpnet-base-v2`, with the previous prompt. Our newly introduced prompt also shows a marked improvement in re-ranking when compared to the previous prompt, for both the fine-tuned and the pre-trained all-mpnet-base-v2 model, justifying the addition of the key relationships, additional fields of the tables and current matching information. For S2 and S3, we see that the Recall@1 and Recall@5 gets improved, i.e., more correct matches are ranked higher. For S1, Recall@20 scores shows an improvement, i.e., source fields which were ranked way lower are now being considered as possible matches, highlighting that our new prompt allows LLM to infer relationships which it previously couldn't. When combined together, the fine-tuned model and new prompt provides the best schema matching result. For the three fields, whose rank degraded slightly (slipped out of the Top 5 suggested matches but remained inside Top 20) post fine-tuning, the LLM agent suggested the correct matches as the top matches.

While Llama 3.1-8B, the smallest model shows slightly weaker performance amongst all the other models, it is imperative to note that this model can be run locally, thus *solving one of our biggest practical challenges* of using this solution in an enterprise scenario, i.e., data privacy. This supports our idea that a fine-tuned sentence embedding model and a SLM used with schema and context aware prompt can generate competitive results when compared to the larger proprietary LLMs.

We also compare our tool's performance against that of Magneto [10]. We rewrite our enriched dataset to that of Magneto's format and use Llama 3.1-8B as the LLM for re-ranking for both the solutions. Table 8 shows the comparison between our solution and that of Magneto's for our use case. SchemaTune *outperforms* Magneto especially at the Recall@1 and Recall@5 scores.

Table 8: Comparison between magneto and SchemaTune

	Magneto	**SchemaTune**
S1 (39)	Recall@1: 0.36	Recall@1: 0.49
	Recall@5: 0.67	Recall@5: 0.69
	Recall@20: 0.77	Recall@20: 0.85
S2 (56)	Recall@1: 0.48	Recall@1: 0.68
	Recall@5: 0.68	Recall@5: 0.88
	Recall@20: 0.82	Recall@20: 0.95
S3 (41)	Recall@1: 0.46	Recall@1: 0.68
	Recall@5: 0.8	Recall@5: 0.9
	Recall@20: 0.93	Recall@20: 0.95

5 Future Work and Conclusion

In this paper, we have presented SchemaTune, an empirically guided solution which fine-tunes a small sentence embedding model using scarce data and then uses that model to generate a ranked list of matches. The ranked list is passed on to an LLM agent which makes use of a newly introduced prompt to re-rank it. Our experimental results show our solution betters the other SOTA approaches present and allows SLMs to match proprietary LLMs in performance for the task of schema matching. We have designed SchemaTune so that it is applicable for use in the enterprises by minimizing cost of computing resources and LLM usage along with the threat of data privacy. With the help of SchemaTune, schema matching, a critical task in IT enterprises today will be faster and have less errors. For our future work, we look to improve upon this work by incorporating business and schema documents into the schema matching process. Presently, business experts make use of these documents which contain a lot of information that is useful in identifying complex data and schema relationships.

References

[1] Arihant Bedagkar, Sayandeep Mitra, Raveendra Medicherla, Ravindra Naik, and Samiran Pal. 2025. LLM Driven Smart Assistant for Data Mapping. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP).* 181–191. doi:10.1109/ICSE-SEIP66354.2025.00022

[2] Philip A Bernstein, Jayant Madhavan, and Erhard Rahm. 2011. Generic schema matching, ten years later. *Proceedings of the VLDB Endowment* 4, 11 (2011), 695–701.

[3] Philip A Bernstein, Jayant Madhavan, and Erhard Rahm. 2011. Generic schema matching, ten years later. *Proceedings of the VLDB Endowment* 4, 11 (2011), 695–701.

[4] Hong-Hai Do and Erhard Rahm. 2002. COMA—a system for flexible combination of schema matching approaches. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases.* Elsevier, 610–621.

[5] AnHai Doan, Alon Halevy, and Zachary Ives. 2012. *Principles of data integration.* Elsevier.

[6] Zlatan Dragisic, Valentina Ivanova, Patrick Lambrix, Daniel Faria, Ernesto Jiménez-Ruiz, and Catia Pesquita. 2016. User validation in ontology alignment. In *International Semantic Web Conference.* Springer, 200–217.

[7] Hugging Face. 2021. MPNet Model. https://huggingface.co/sentence-transformers/all-mpnet-base-v2/tree/main

[8] Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun-Hsuan Sung, László Lukács, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. 2017. Efficient natural language response suggestion for smart reply. *arXiv preprint arXiv:1705.00652* (2017).

[9] 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).* IEEE, 468–479.

[10] Yurong Liu, Eduardo Pena, Aecio Santos, Eden Wu, and Juliana Freire. 2025. Magneto: Combining Small and Large Language Models for Schema Matching. arXiv:2412.08194 [cs.DB] https://arxiv.org/abs/2412.08194

[11] Sayandeep Mitra, Debayan Mukherjee, Atreya Bandyopadhyay, Rajdip Chowdhury, Raveendra Kumar Medicherla, Indrajit Bhattacharya, and Ravindra Naik. 2021. Learning-based Assistant for Data Migration of Enterprise Information Systems. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE).* 1121–1125. doi:10.1109/ASE51524.2021.9678533

[12] mteb. 2025. MTEB Leaderboard. https://huggingface.co/spaces/mteb/leaderboard.

[13] Debayan Mukherjee, Atreya Bandyopadhyay, Rajdip Chowdhury, and Indrajit Bhattacharya. 2021. Learning knowledge graph for target-driven schema matching. In *Proceedings of the 3rd ACM India Joint International Conference on Data Science & Management of Data (8th ACM IKDD CODS & 26th COMAD).* 65–73.

[14] Marcel Parciak, Brecht Vandevoort, Frank Neven, Liesbet M Peeters, and Stijn Vansummeren. 2024. Schema Matching with Large Language Models: an Experimental Study. *arXiv preprint arXiv:2407.11852* (2024).

[15] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[16] Nabeel Seedat and Mihaela van der Schaar. 2024. Matchmaker: Self-improving large language model programs for schema matching. *arXiv preprint arXiv:2410.24105* (2024).

[17] Eitam Sheetrit, Menachem Brief, Moshik Mishaeli, and Oren Elisha. 2024. ReMatch: Retrieval Enhanced Schema Matching with LLMs. *arXiv preprint arXiv:2403.01567* (2024).

[18] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL] https://arxiv.org/abs/2302.13971

[19] Sha Wang, Yuchen Li, Hanhua Xiao, Bing Tian Dai, Roy Ka-Wei Lee, Yanfei Dong, and Lambert Deng. 2025. LLMATCH: A Unified Schema Matching Framework with Large Language Models. *arXiv preprint arXiv:2507.10897* (2025).

[20] An Yang et al. 2025. Qwen3 Technical Report. arXiv:2505.09388 [cs.CL] https://arxiv.org/abs/2505.09388

[21] 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.

[22] Yunjia Zhang, Avrilia Floratou, Joyce Cahoon, Subru Krishnan, Andreas C Müller, Dalitso Banda, Fotis Psallidas, and Jignesh M Patel. 2023. Schema matching using pre-trained language models. In *2023 IEEE 39th International Conference on Data Engineering (ICDE).* IEEE, 1558–1571.