Schema Matching using Pre-Trained Language Models

Yunjia Zhang
University of Wisconsin-Madison
yunjia@cs.wisc.edu

Avrilia Floratou
Microsoft
avflor@microsoft.com

Joyce Cahoon
Microsoft
jcahoon@microsoft.com

Subru Krishnan
Microsoft
subru@microsoft.com

Andreas C. Müller
Microsoft
amueller@microsoft.com

Dalitso Banda*
Sentry
dalitso.banda@sentry.io

Fotis Psallidas
Microsoft
fopsalli@microsoft.com

Jignesh M. Patel
University of Wisconsin-Madison
jignesh@cs.wisc.edu

***Abstract*—Schema matching over relational data has been studied for more than two decades. However, the state-of-the-art methods do not address key modern-day challenges encountered in real customer scenarios, namely: 1) no access to the source (customer) data due to privacy constraints, 2) target schema with a much larger number of entities and attributes compared to the source schema, and 3) different but semantically equivalent entity and attribute names in the source and target schemata. In this paper, we address these shortcomings. Using real-world customer schemata, we demonstrate that existing linguistic matching approaches have low accuracy. Next, we propose the Learned Schema Mapper (LSM), a novel linguistic schema matching system that leverages the natural language understanding capabilities of pre-trained language models to improve the overall accuracy. Combining this with active learning and a smart attribute selection strategy that selects the most informative attributes for users to label, LSM can significantly reduce the overall human labeling cost. Experimental results demonstrate that users can correctly match their full schema while saving as much as 81% of the labeling cost compared to manual labeling.**

I. Introduction

More and more organizations in various industries (retail, healthcare, manufacturing, etc.) are adopting a data-driven approach to discover non-trivial and strategic insights. At the same time, the amount of information collected has never been greater [1], highlighting the need for modern data analytics tools that can keep up with the volume, variety, and velocity of data associated with different industries. To that end, multiple companies are offering industry-specific analytics solutions [2]–[4] tailored to meet customers' domain-specific needs. Azure Industry solutions [2], for example, offers a variety of services that aim to optimize industry-specific processes, provide enhanced collaboration capabilities and reduce the time to actionable insights. Enabling these enhanced experiences often requires a substantial understanding of customer data. To tackle this problem, the service operator usually maps the customer's data to a known industry-specific schema (ISS) whose semantics are clearly defined, and then builds tools on top of it to streamline various processes.

*Work done while at Microsoft.

In this paper, we study the schema matching problem in the context of known ISSs and customer data in the form of a relational schema. While the problem has been studied for over two decades, we find that existing solutions are not adequate in our environment.

First, information about customer data is often limited due to privacy considerations. In particular, we find that customers are reluctant to grant access to individual data records [5], [6]. As a result, the service operator cannot leverage customer data records when performing schema matching. The same considerations do not constrain their access to the customer schemata. Indeed, since the outcome of schema matching reveals what entities and attributes in the known ISS are present in the customer data, further obfuscating the customer schema from the service operator is not valuable [1]. As a result, customers are generally willing to provide schema information to the operator to facilitate schema matching.

Second, ISSs are typically large (in terms of entities and attributes) as they try to capture a wide variety of concepts associated with a given industry. The customer schema, however, does not necessarily encapsulate all these concepts and is often much smaller than the target ISS. This discrepancy can complicate the schema matching process as it results in a large number of available (and potentially irrelevant) candidate matches for each attribute of the customer schema.

Finally, the entity and attribute names of the customer schema are often hard to understand due to abbreviations and/or customer-specific terminology, further complicating the automation of schema matching for the service operator.

The externally-imposed customer constraints along with the challenges associated with real customer schemata significantly limit the accuracy of existing schema matching approaches (see Section III). In this work, we propose the Learned Schema Matcher (LSM), a novel human-in-the-loop linguistic schema matching approach. LSM does not require access to the individual data records (data-free schema matching) but only to the source schema, respecting the customer

[1]There are other scenarios where both the source and target schemata need to be obfuscated from the service operator [7]. Extending our approach to meet these more stringent privacy requirements is left for future work.

requirements. At the core of our approach lies a fine-tuned language model that is able to scale to large ISSs and allows us to better handle noise in the source schema. Further, by combining this model with active learning and a smart interaction strategy to obtain targeted feedback from the user, we can fully map a source schema to an ISS while incurring significantly less human labeling cost than prior work.

Our contributions are the following:

- We study the state-of-the-art linguistic schema matching approaches on real-world schemata from Microsoft customers and demonstrate that they have low accuracy.
- We develop LSM, a novel matching approach that relies on a fine-tuned pre-trained language model to tackle data-free schema matching. To the best of our knowledge, this is the first work demonstrating that leveraging the natural language capabilities of pre-trained language models can improve the accuracy of data-free schema matching. Previous methods that are either not data-free [8]–[10] or do not leverage natural language understanding [11]–[15].
- The externally-imposed customer constraints render near-100% accuracy virtually impossible without humans-in-the-loop. We thus propose to combine our schema matching model with an active learning strategy. Our method employs a novel attribute selection strategy to obtain targeted feedback from the user with the goal of reducing the human labeling cost.
- We evaluate LSM on real customer schemata and publicly available datasets. We show that with LSM, users can correctly match their full schema to ISS with a 81% reduction in labeling cost compared to manual labeling.

The remainder of the paper is organized as follows: Section II contains the problem statement. Section III presents an evaluation of the state-of-the-art linguistic matching approaches on real customer schemata. Section IV describes the design of our proposed LSM method. Section V contains our experimental evaluation, and Section VI discusses our experiences in designing LSM. Finally, Section VII presents the related work, and our conclusions are stated in Section VIII.

II. Problem Statement

We now define the problem of schema matching in our environment. Given a source (customer) schema and a known target (ISS) schema that both follow the E/R model [16], our goal is to map each attribute of the source schema to an attribute at the target schema. Figure 1 shows an example of an ISS and a customer schema and their corresponding match.

More formally, we are given a source schema S_s consisting of a set of entities E_s, a set of attributes A_s, and a set of PK/FK relationships R_s. Similarly, we also have a known target schema S_t that consists of a set of entities E_t, a set of attributes A_t, and a set of PK/FK relationships R_t. We assume that each attribute in A_s (A_t) belongs to a single entity in E_s (E_t).

Each entity e ($e \in E_s \cup E_t$) has a name $e.name$, a primary key $e.pk$, and a set of foreign keys $e.fks$. Each attribute a ($a \in A_s \cup A_t$) has a name $a.name$, a data type $a.dtype$, and optionally a natural language description $a.desc$ associated with it.

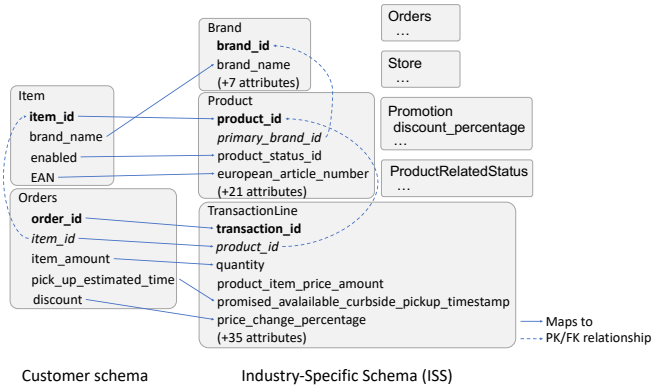


Fig. 1: An example customer schema and corresponding ISS.

We define an attribute correspondence as $r_{ij} = (a_i, a_j)$ where a_i is an attribute in the source schema S_s and a_j is an attribute at the target schema S_t. As discussed in [17], intuitively, an attribute correspondence specifies a relationship between a_i and a_j. The correspondence may specify that the two attributes are equal to each other or that there may be a transformation function involved (e.g., from dollars to euros). In this work, we focus on correspondences that denote equality and leave more complex transformations as part of future work.

Definition 1. *An entity match $\mathcal{M}$ is a triple (e_s, e_t, m), where e_s is an entity in E_s, e_t is an entity in E_t, m is a set of attribute correspondences between e_s and e_t, and each source and target attribute occurs in at most one correspondence in m.*

Based on the above definition, we now define the schema matching problem addressed in this paper.

Definition 2. *The result $\overline{\mathcal{M}}$ of the schema matching process is a set of entity matches between entities in E_s and in E_s, where each attribute in S_s and S_t appears in at most one entity match.*

As we discuss in Section IV, our approach generates $\overline{\mathcal{M}}$ by utilizing all the available information in the source and target schemata (i.e., entities, attributes, and PK/FK relationships).

III. Motivation

Given that in our setting, we do not have access to the customer data, we evaluated the applicability of the state-of-the-art schema matching approaches that rely solely on schema-level information. In this section, we demonstrate that prior approaches have poor accuracy on our real customer schemata. We then identify a set of challenges that motivate the design of the Learned Schema Matcher.

State-of-the-art Methods. We choose four representative schema-based matching methods: CUPID [11], COMA [18],

	# Entities	# Attr.	# Unique Attr. Names	# PK/FK	Desc.
Customer A	3	29	28	2	Y
Customer B	8	53	45	7	N
Customer C	3	84	82	2	N
Customer D	7	136	125	7	N
Customer E	25	530	475	24	Y

TABLE I: Statistics on the customers' (source) schemata.

		# Entities	# Attributes	# PK/FK
RDB-star	Source	13	65	12
	Target	5	34	4
IPFQR	Source	1	51	0
	Target	1	67	0
MovieLens-IMDB	Source	6	19	5
	Target	7	39	6

TABLE II: Statistics on publicly available schemata.

S-MATCH [12], and Similarity Flooding [15], which are all heuristic-based. To evaluate the effectiveness of machine learning-based approaches, we also experimented with the LSD [13] and MLM [14] methods. Note that these methods require access to both the schema and the data records, which is not possible in our environment. We thus had to adapt them to leverage schema-level information only. We implemented all the methods except COMA [18] from scratch according to the descriptions in [11]–[14] respectively. We use the latest community edition of COMA 3.0 [2]. Note that COMA and CUPID can optionally accept user feedback during the matching process. In this section, we present results without user feedback, but we do use the interactive mode in our experiments in Section V where we present a comparison with our approach. A summary of the methods is presented below:

- ***CUPID:*** CUPID tackles the schema matching problem for both XML and relational schemata by combining both linguistic (through a synonym dictionary) and structural information and computing the weighted sum of the associated similarities. In our implementation, we use the pre-trained word embedding from FastText [19] as the synonym dictionary and generate the similarity score using cosine similarity. For each customer schema, we search the best-performing weights for the weighted sum and report only the best results.
- ***COMA:*** COMA uses a set of matchers to compute the linguistic similarity between two attributes. The matchers cover a broad spectrum of similarity metrics such as affix, n-gram, Soundex, edit distance, etc. To combine the similarities, COMA can choose from various aggregation functions such as min, max, average, etc.
- ***S-MATCH:*** S-MATCH (SM) solves the semantic matching problem on two trees. It uses WordNet [20] to understand the meaning of the nodes in the trees and identify synonyms. It covers multiple semantic relations such as equivalence, mismatch, overlapping, etc. We use S-MATCH to tackle the schema matching problem as defined in Definition 2 and thus consider only attribute equivalence.

[2]https://sourceforge.net/projects/coma-ce/

	CUPID	COMA	SM	SF	LSD	MLM
RDB-Star	0.96	**1.00**	**1.00**	0.70	0.26	**1.00**
IPFQR	**1.00**	0.98	0.82	**1.00**	0.08	0.98
MovieLens-IMDB	0.64	0.54	**0.72**	0.71	0.00	0.64
Customer A	0.18	**0.21**	0.07	0.11	0.00	0.11
Customer B	**0.14**	0.02	0.06	0.02	0.00	0.08
Customer C	0.08	0.22	0.11	**0.23**	0.00	0.14
Customer D	0.27	**0.31**	0.13	0.14	0.00	0.14
Customer E	0.27	0.17	**0.28**	0.12	0.00	0.14

TABLE III: The top-3 accuracy of six state-of-the-art approaches on our schemata.

- ***Similarity Flooding:*** Similarity flooding (SF) [15] builds a pairwise connectivity graph based on the structure of the source and target schemata. The connectivity graph contains element pairs as nodes. Two nodes (x_1, y_1) and (x_2, y_2) are connected if x_1, x_2 and y_1, y_2 are neighbors in the original schema graphs respectively. SF initializes the similarity scores of the element pairs and propagates the scores for each element pair following the edges in the connectivity graph. In our implementation, we use embedding similarities as the initial scores.
- ***LSD:*** The LSD method is one of the first to leverage machine learning for schema matching. LSD learns to match between the source and target schemata based on a set of provided examples. LSD has four individual learners: 1) the `whirl` learner, which classifies the attributes based on nearest neighbors of TF-IDF encodings, 2) a naive Bayesian learner that classifies the descriptions, 3) a name matcher based on name similarities, and 4) a county-name recognizer that matches the names if the attribute is a county name. We randomly select 50% of the ground truth matches as the training set and measure the matching quality for the rest 50% of the attributes.
- ***MLM:*** MLM is another machine learning-based method. Different from LSD, MLM featurizes the candidate matches using both the schema specifications and the data records. Following the featurization, MLM generates matches with a clustering model (K-means or self-organizing map [14]). To apply MLM in our problem setting, we only use the features from the schema-level specification. Since MLM adopts unsupervised learning, all the attributes in the target (ISS) schema are treated as the training set.

Datasets. We use five real-world schemata from Microsoft customers in retail. The characteristics of these schemata are shown in Table I. As shown in the table, the five schemata are of different sizes: Customer A has the smallest schema with 29 attributes spread across three entities, while Customer E has the largest schema with 25 entities and 530 attributes. Among the five customer schemata, two of them contain attributes with natural language descriptions. The corresponding target ISS is also defined for the retail industry, which encapsulates information about customers, goods, stores, promotions, sales, and other assets that are typically tracked. The retail ISS we consider consists of 92 entities, 1218 attributes, and 184 PK/FK relationships. The ground truth matches are obtained from domain experts.

In addition, we also evaluate the performance of the state-of-the-art solutions on three publicly available schemata shown in Table II. RDB-Star is a synthetic dataset used in [11]; IPFQR is the Inpatient Psychiatric Facility Quality data set [3]; MovieLens-IMDB contains the mapping between the MovieLens [21] schema and the IMDB schema [4]. For these data sets, we manually created the ground truth matches.

Methodology. All the methods that we study generate a matching score for each pair of attributes at the source and target schema. The score shows the probability that the pair is a good match. For each attribute in the source schema, we examine whether the correct target attribute is in the top-3 candidate target attributes list and report the top-3 accuracy.

Results. Table III shows our results. As shown in the table, all the baseline methods, except LSD, have near-perfect accuracy on the RDB-Star and IPFQR datasets. Since LSD is based on the TF-IDF encodings, it has difficulty with a small training set and concise (not verbose) input attribute names. For MovieLens-IMDB, the best baseline achieves a 0.72 accuracy since the matches are not as straightforward as in RDB-Star and IPFQR datasets. Moreover, the accuracy of all methods significantly drops (below 0.3) on the real-world customer schemata. In addition, no single approach outperforms the others on all schemata.

Taking a closer look at the ground truth matches in the public schemata, we observe that the names between the source and target attributes are similar for most of the cases. For example, in RDB-Star, for the source attribute *Sales.Discount*, the corresponding correct target attribute is *OrderDetails.Discount*. Note that not only they both have similar names (aka *Discount*), but there is also no other attribute in the target schema with this name, which makes the traditional matching task relatively straightforward. The real-world schemata, however, do not follow the same naming conventions as the ISS. Since we cannot directly showcase the customer schemata due to privacy reasons, we explain the challenges through the example in Figure 1. As shown in the figure, the *Orders.discount* in the customer schema maps to the *TransactionLine.price_change_percentage* attribute in the ISS. The two attributes have very different names, yet they are a match. We observe that more than 30% of the matches in the customer schemata follow such a pattern. The RDB-Star dataset, on the other hand, does not contain any such matches.

Another difference between the real-world scenario and the public dataset is that the ISS contains a large number of entities and attributes compared to the customer schema. As we discussed in Section I, this is typically the case as the ISS tries to capture almost every concept within an industry, which might not be the case for an individual customer. As a result, the search space becomes much larger, resulting in a higher number of candidate target attributes for each source attribute. As an example, in Figure 1, a simple edit distance matcher like the one used in COMA will map the source attribute *Orders.item_amount* to the *TransactionLine.product_item_price_amount* attribute instead of the correct *TransactionLine.quantity*. The existence of a large number of attributes in the target schema along with multi-word attribute names increase the chances that such candidates will occur. In the public dataset, both schemata are relatively small and thus we do not observe this problem. The scale of ISS, however, creates plenty of opportunities for erroneous candidates, which the existing approaches cannot tackle.

Insights. Based on our evaluation of the state-of-the-art approaches and the challenges we encountered, we now discuss opportunities for improvement in the space.

- ***Schema-only access.*** The customer-imposed constraints discussed in Section I, along with the challenges of real-world customer schemata, render near-100% accuracy virtually impossible. We thus believe that a combination of a reasonably good model with human-in-the-loop intervention in a targeted fashion is likely the best way to improve the schema matching process. As we discuss in the following sections, we opt for an active learning approach that attempts to minimize the human labeling cost while still allowing users to map their full schema correctly to the ISS.
- ***Entity and attribute naming.*** As shown in this section, the similarity metrics used by existing approaches fail to capture matches where the entity and attribute names between the two schemata are different, yet semantically equivalent. The existence of multi-world names further complicates the issue. We believe there is an opportunity to address some of these issues by leveraging the natural language understanding capabilities of pre-trained language models. Our approach relies on BERT, an encoder-only model, as discussed below.
- ***Large ISS.*** The ISS is typically large in terms of the number of entities and attributes and is known in advance. Thus, there is an opportunity to leverage pre-training techniques to create a model that better understands the domain we are operating on.

To address the above challenges, we design a novel linguistic schema matching approach based on semi-supervised and active learning that is presented in Section IV.

IV. The Learned Schema Matcher

To address the shortcomings of prior work, we propose the Learned Schema Matcher (LSM), a linguistic schema matching approach based on semi-supervised and active learning. In this section, we first present an overview of the matching pipeline of the Learned Schema Matcher, followed by a detailed description of each step.

A. Matching Pipeline Overview

Figure 2 depicts the matching pipeline of the Learned Schema Matcher. LSM takes as input the source schema S_s

[3]https://www.cms.gov/Medicare/Quality-Initiatives-Patient-Assessment-Instruments/HospitalQualityInits/IPFQR. We use the state data schema as source schema and national data schema as target schema.

[4]https://www.imdb.com/interfaces/

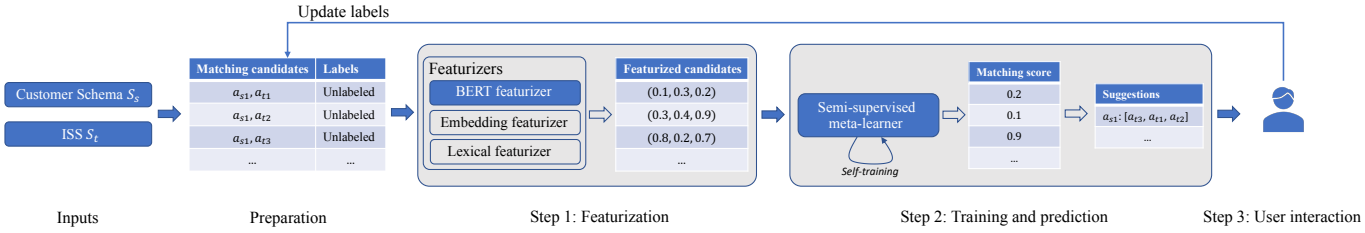


Fig. 2: The Matching Pipeline of the Learned Schema Matcher

and the target ISS S_t and attempts to perform the full matching in an iterative and interactive fashion.

For a new source schema S_s, we first generate the set of all candidate pairs by calculating the Cartesian product between the two attribute sets A_s and A_t (preparation phase). The resulting candidate pairs are unlabeled to begin with. We iteratively build our model using active learning, where at each iteration, a new candidate pair is labeled by the user. The interaction with the user starts in an iterative fashion. Each iteration consists of three phases: 1) featurizing candidate pairs, 2) model training and output of matching suggestions, and 3) incorporating user feedback.

During the featurization phase (Step 1 in Figure 2), a modular featurizer converts the candidate pairs into numerical vectors. Our model consists of several pre-trained featurizers, including BERT and FastText [19] as well as lexical features. A key innovation in our work is the design of a fine-tuned BERT featurizer built using a pre-trained language model, that contributes to the high accuracy of our approach. We create a similarity score based on BERT, using a classifier trained just using the ISS schema, which can be reused across all customers. During each iteration, we improve the similarity score provided by this featurizer using human feedback, as described below.

During the training and prediction phase (Step 2 in Figure 2), a ML model is trained on the partially labeled data via self-training, a method from semi-supervised learning, using the output of the featurizers as input and the user-provided labels (available only after the first iteration) as the target. After the training is completed, the ML model predicts the probability that an unlabeled candidate pair represents a correct match and then generates a set of top-k matching suggestions for each unmapped source attribute in S_s.

In the next phase, the user receives the suggestions and they can either choose to continue interacting with LSM or terminate the process (Step 3 in Figure 2). If they continue, the next step is to review the top-k suggestions for each unmapped attribute and either mark one of them as correct or indicate that none of them is correct. When the reviewing process is completed, LSM selects a subset of N source attributes for which none of the suggestions was correct and asks the user to provide the correct matching to the ISS (N is typically 1). To reduce the human labeling cost, we employ a smart attribute selection strategy to identify the set of N most "informative" attributes for users to label (see Section IV-E). The feedback provided by the users is subsequently used to update the labels of the candidate pairs before the next iteration starts.

We now discuss in more detail the various phases in the matching pipeline of LSM.

B. Preparation

During this phase, we generate a set of candidate pairs by calculating the Cartesian product between the two sets of attributes A_s and A_t. Specifically, we generate a set of candidate pairs P by calculating $P = A_s \times A_t = \{(a_s, a_t) \,|a_s \in A_s, a_t \in A_t\}$. Each candidate pair $p \in P$, has an associated label l_p to eventually be assigned by the user, indicating whether the candidate pair represents a correct match ($l_p = 1$), an incorrect match ($l_p = 0$) or is currently unlabeled ($l_p = -1$). The labels for all the candidate pairs are initially set to -1 after this phase is completed.

C. Featurization

During the featurization step, we convert the candidate pairs $(a_s, a_t) \in P$ into numerical vectors. We apply a set of different featurizers that measure the similarity between the attributes a_s and a_t using a variety of metrics. Each individual featurizer takes the two attributes in the candidate pair, and outputs a similarity score. We have built a modular featurization pipeline with currently three featurizers plugged in, but our design allows for easy incorporation of more featurizers in the future. We use a word embedding featurizer and a lexical featurizer, and we also design a novel BERT featurizer whose details are presented next. We note that our architecture is generic enough to accept any encoder-only model such as BERT, Electra [22], RoBERTa [23], etc.

1) BERT Featurizer:
In Section III, we showed that existing linguistic schema matching approaches have low accuracy on real customer datasets. One of the reasons behind their sub-optimal performance is that they are not robust to noise in the entity and attribute names in the source schema. This noise might be in the form of customer-specific terminology or abbreviations (see *Item.EAN* in Figure 1). Additionally, the ISS contains a large number of entities and attributes with multi-word names, such as *TransactionLine.product_item_price_amount* that further complicates the matching process.

Recently, language models have emerged as strong generic feature extraction solutions in the context of natural language processing (NLP) for tasks such as text generation [24],

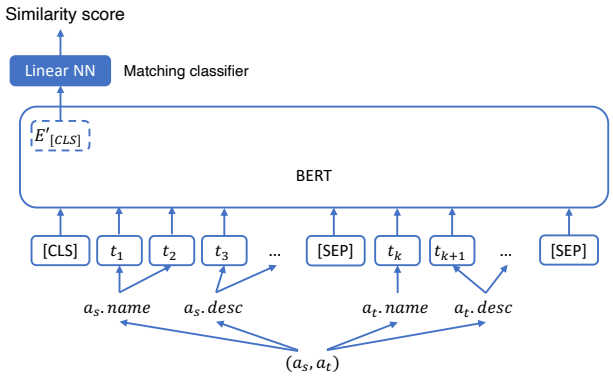


Fig. 3: Architecture of the BERT featurizer.

text classification [25], [26], summary extraction [27], [28], entity matching [10], etc. Therefore, we decided to leverage them to better capture the similarity between the attributes in each candidate pair using their corresponding names and descriptions. Specifically, we regard this problem as a binary text classification problem: given a sentence that describes the two attributes, we predict whether these two attributes represent a correct match.

Model Architecture. Among the various language models, we choose BERT [29] as our base language model. The model architecture of our fine-tuned BERT is similar to the one presented in [10] and is shown in Figure 3. We use the pre-trained BERT on the Toronto Book and Wikipedia corpora as described in [30]. On top of the BERT hidden state $E'_{[CLS]}$, we add a binary classifier consisting of a single hidden layer neural network with a sigmoid activation function. We refer to this classifier as the `matching classifier` as it is responsible for fine-tuning BERT for schema matching.

For the inputs of our fine-tuned BERT model, we convert the two input attributes in a candidate pair into a sentence by concatenating the names and descriptions of the two attributes. Specifically, for each candidate pair (a_s, a_t), the input sentence to the model is generated as "*[CLS]* $a_s.name$ $a_s.desc$ *[SEP]* $a_t.name$ $a_t.desc$ *[SEP]*". *[CLS]* and *[SEP]* are two special tokens in BERT that mark the start and separation of the input sequence, respectively. The output of the fine-tuned BERT featurizer is a similarity score for the candidate pair.

Pre-training the Matching Classifier. To optimize the performance of the BERT featurizer, we pre-train the `matching classifier` by leveraging the content of ISS. Intuitively, the pre-training phase informs the `matching classifier` on the output of the language model (BERT) when the input attributes are linguistically similar. Pre-training aims to quickly impart the basic knowledge of "similar" attributes to the `matching classifier` without any matching labels.

Pre-training happens only once per ISS, in other words, per vertical, and the resulting classifier can be used without any additional training for feature extraction on any source schema. We generate the labeled input sequences as follows:

- ***Positive Samples.*** We generate three types of sentences with positive labels: 1) self-repeating, 2) self-explaining, and 3) PK/FK linking. **Self-repeating**: For each attribute $a_t \in S_t$, we generate a sentence "*[CLS]* $a_t.name$ $a_t.desc$ *[SEP]* $a_t.name$ $a_t.desc$ *[SEP]*" with a positive label. **Self-explaining**: For each attribute $a_t \in S_t$, we generate a sentence "*[CLS]* $a_t.name$ *[SEP]* $a_t.desc$ *[SEP]*" with a positive label. **PK/FK linking**: For every two attributes $a_t, a_k \in S_t$ with a PK/FK relationship, we generate a sentence "*[CLS]* $a_t.name$ $a_t.desc$ *[SEP]* $a_k.name$ $a_k.desc$ *[SEP]*" with a positive label.
- ***Negative Samples.*** We randomly corrupt one side of the positively labeled sentences to generate negative samples. Particularly, we randomly choose a $a'_t \in S_t, a'_t \neq a_t$, and replace a_t with a'_t for all the three types of positive samples.

Updating the Matching Classifier based on Human Labels. Besides pre-training the `matching classifier`, we can further improve our results by updating the classifier based on the human labels provided by the users in each iteration. In particular, for all attribute pairs $p = (a_s, a_t), p \in P$ with labels l_p, we add the sentences "*[CLS]* $a_s.name$ $a_s.desc$ *[SEP]* $a_t.name$ $a_t.desc$ *[SEP]*" and the corresponding labels l_p to the training set and assign them a larger sample weight than the samples generated using just the ISS schema.

By pre-training the `matching classifier` , the BERT featurizer is able to identify the basic linguistic similarities even when no/limited matching labels from human experts are initially provided. Updating the `matching classifier` based on user labels allows the BERT featurizer to adapt to the characteristics of each individual source schema.

2) Word Embedding and Lexical Featurizers:
In addition to the BERT featurizer, we employ two other common featurizers, namely, the word embedding featurizer and the lexical featurizer, which capture the embedding similarity and lexical similarity between a pair of attributes.

- ***Word Embedding Featurizer.*** The word embedding featurizer calculates the cosine similarity between the embedding representations of the attribute names. We use the pre-trained FastText [19] embeddings in our experiments.
- ***Lexical Featurizer.*** The lexical featurizer measures whether two attributes have high lexical similarity. For the attribute pair (a_s, a_t), the similarity score is calculated as $\frac{lsc(a_s.name, a_t.name)}{min(len(a_s.name), len(a_t.name))}$ where function lsc computes the length of the longest common subsequence. The lexical featurizer is capable of handling abbreviations.

The combination of various featurizers improves the accuracy of the model, especially when the source schema uses different naming conventions than the ISS and a limited number of labels are provided.

D. Meta-learning Model Training and Prediction

Training. With the currently provided labels, we train the meta-learner and predict the matching labels for each of the

unlabeled attributes. We use self-training [31], a standard semi-supervised training framework. In self-training, we first train the meta-learning model on the labeled subset of the training data. This model is then used to generate (pseudo-)labels for the unlabeled data points.

The base classifier for the semi-supervised framework is a simple linear classifier using logistic loss. The inputs of the classifier are the similarity scores given by each of the three featurizers, and the output labels are provided using the self-training procedure.

Prediction. After the model is trained, we use it to make a prediction on each candidate pair (a_s, a_t) in P and obtain a list of matching scores. We further tune the matching scores based on other schema-level information as follows:

- ***Handling data type mismatches.*** In our application scenario, we observe that in nearly all correct matches, the source and target attributes have compatible data types. Therefore, we set the score of a pair consisting of attributes with incompatible data types to be 0, i.e. $score(a_s, a_t) \leftarrow 0$ if $a_s.dtype \neq a_t.dtype$. If the source schema is not well designed, this process can be skipped.
- ***Penalizing introduction of new entities.*** Users would typically prefer to map their schema to a concise subset of ISS if possible. Thus, we introduce a heuristic that penalizes matches that result in source attributes being mapped across multiple target entities in ISS. Specifically, we apply a penalization term $z \in [0, 1]$ to penalize the matching score, i.e. $score(a_s, a_t) \leftarrow z \times score(a_s, a_t)$, if the entity that contains a_t is not part of the current matches so far. Intuitively, the closer the newly added entity is to the current entities in the ISS graph, the lower the cost of adding it to the set of matches. This is because the user would need to perform fewer join operations to merge the data. Thus, the penalization term is set as $z = \frac{1}{1+\log\left(1+sp(a_t, \overline{\mathcal{M}})\right)}$, where $sp(a_t, \overline{\mathcal{M}})$ denotes the shortest path (on the join graph of ISS) between the entity containing a_t and the entities in ISS that are already part of the set of matches $\overline{\mathcal{M}}$.

For each attribute a_s belonging to a candidate pair $(a_s, a_t) \in P$, we provide a list of matching suggestions k_s by selecting the target attributes a_t with the top-k predicted matching scores. We define the prediction confidence c_s of the matching suggestions k_s for a_s as the max score of all the candidate pairs (a_s, a_t), i.e. $c_s = \max_{a_t \in k_s} score(a_s, a_t)$.

E. User Interaction

After providing the matching suggestions for the unlabeled source attributes a_s, the user can decide whether to continue the interaction loop. If they choose to continue, they perform the following two tasks: 1) review the matching suggestions and mark the ones that are correct, and 2) label a set of incorrectly matched attributes selected by the Learned Schema Matcher to improve the predictions.

1) Reviewing Matching Suggestions:
During the reviewing process, for each unlabeled attribute a_s, the user is given a list of k attributes (k_s) from A_t, indicating the top-k matching suggestions. For each suggestion, the user either 1) selects the correct attribute that a_s maps to, or 2) indicates that there are no correct matches in the k matching suggestions. If the correct matching attribute a_t is selected, the label for the pair (a_s, a_t) is positively set. In addition, negative labels are generated for the pairs (a_s, a'_t) where $a'_t \neq a_t$. If the user indicates that there are no correct matching attributes in the top-k suggestions, negative labels are generated for all the pairs (a_s, a_t) where $a_t \in k_s$. This process can be skipped if the user decides to improve the quality of the suggestions by providing more labels before reviewing.

2) Selecting Attributes to be Labeled:
As the next step, to further improve the model's performance, LSM selects N attributes (N is typically set to 1) and asks users to provide the correct mapping. Since providing this information requires the users to have some knowledge of the ISS, we design a smart attribute selection strategy, namely, the *least confident anchor* strategy, to reduce the number of labels needed. In Section V, we compare this sophisticated strategy with a purely random strategy and demonstrate its impact on the model's performance.

Least Confident Anchor. In the least confident anchor strategy, we keep a set of *anchor* attributes, containing the most "informative" attributes of the schema. LSM can either adopt a user-provided anchor set or create a default anchor set based on the PK/FK relationships in the source schema S_s. These relationships carry a lot of information as they indicate how various entities and attributes are connected with each other. Specifically, the anchor set of the source schema consists of the attributes in $\{e.pk, e.fks | \forall e \in E_s\}$. For example, the default anchor set corresponding to the example source schema in Figure 1 is { *Item.item_id, Orders.order_id, Orders.item_id* }.

To select N attributes from the anchor set, we use the *least confidence strategy*, which is common in active learning scenarios [32]–[35]. More specifically, among the set of anchor attributes, LSM chooses N unlabeled anchor attributes a_s with the least prediction confidence c_s. The prediction confidence c is calculated by Softmax function on the matching scores for a. At the first iteration, LSM selects the first N attributes from the anchor set for users to label. If all the attributes in the anchor set are labeled, LSM applies the least confidence selection strategy to all other non-anchor attributes.

After the N attributes are selected, users are asked to provide the corresponding matches to the ISS. As before, for each correct match (a_s, a_t), we mark the label for (a_s, a_t) to be 1, and all other (a_s, a'_t) to be -1.

V. Experimental Evaluation

We now experimentally evaluate the quality of LSM on both customer and publicly available schemata and perform an in-depth analysis of the results.

A. Experimental Setup

Datasets. We follow the setup presented in Section III and use the five real-customer schemata and three publicly available datasets.

Baseline Approaches. We compare LSM with the state-of-the-art matching approaches (COMA, CUPID, SM, SF, LSD, and MLM). We tune the baselines by performing a grid search of their hyper-parameters as discussed in Section III. To ensure a fair comparison with LSM, we run the baselines in interactive mode as described in [11], [18]. We additionally implement a similar interaction strategy for S-MATCH (SM) and Similarity Flooding (SF). Note that as opposed to our work, the works in [11], [18] do not propose a concrete strategy for selecting attributes to be mapped by the user. Thus, instead of employing a random strategy, we further optimize the baseline approaches by applying our smart attribute selection strategy.

Evaluation Goals. Our experiments aim to answer the following questions:

- What is the prediction quality of our model? (Section V-B)
- What is the human labeling cost to map the full source schema to the ISS? (Section V-C)
- What is the contribution of the BERT featurizer and the attribute descriptions to the overall performance? (Section V-D)
- How is the performance of LSM affected in the presence of noise? (Section V-F)
- How much time is spent re-training the model after the user provides new labels? (Section V-G).

Evaluation Metrics. To answer the above questions, we use the following metrics to evaluate the performance of our model:

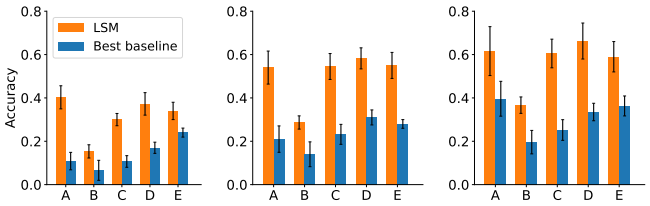
- ***Human Labeling Cost.*** This metric captures the number of human labels needed to map the full source schema to the ISS.
- ***Prediction Accuracy.*** Since the ISS captures a wide variety of concepts for an industry, each of the source attributes $a_i \in A_s$ in the customer schema has a matching attribute in the target A_t. For each source attribute, we provide a list of matching suggestions and measure the top-k accuracy of these suggestions on the unlabeled part of the source schema [36]. This metric also implicitly reflects the expected reviewing cost (see Section IV-E1): the higher the accuracy, the fewer attributes would need to be reviewed in the next iteration. We use $k = 3$ for our experiments unless specified otherwise.
- ***Response Time.*** Given the interactive nature of our approach, the users' experience will be affected by how fast we can retrain our model using the provided human labels at each iteration. Therefore, we measure the response time in seconds at each iteration and report the average.

B. Evaluating Model Performance

In this experiment, we test our model in a non-interactive manner in order to decouple the evaluation of the active learning strategy from the evaluation of the model itself: given a set of training matching labels, we train our model and evaluate how accurate it is on the test set. We follow the same methodology as in Section III but now report top-k accuracy for k=1,3,5. We use both the publicly available and the real customer schemata. For each schema, we only report the results from the best baseline according to Table III.

	Best Baseline			LSM		
	top-1	top-3	top-5	top-1	top-3	top-5
RDB-Star	0.95	**1.00**	**1.00**	**0.98**	0.98	**1.00**
IPFQR	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
MovieLens-IMDB	0.53	0.72	0.75	**0.65**	**0.83**	**0.87**

TABLE IV: Top-k accuracy of LSM on the public schemata.



(a) Top-1 (b) Top-3 (c) Top-5

Fig. 4: Top-k accuracy of LSM vs the best baseline on customer schemata A-E.

Public Schemata. The results on public schemata are shown in Table IV. We report the median top-1, 3, 5 accuracy of five independent trials. As shown in the table, both LSM and the best baseline approach can achieve near-perfect matching predictions on the relatively easy RDB-Star and IPFQR datasets, while LSM out-performs the best linguistic matching baseline by over 10% on the MovieLens-IMDB dataset. This shows LSM can perform well on small and relatively straightforward matching tasks.

Customer Schemata. Figure 4 presents the accuracy on customer schemata. The height of the bars show the average accuracy of five independent trials and the error bars correspond to the standard errors. The best baseline refers to the baseline that leads to the highest accuracy on the given customer's schema. Note that the best baseline is not the same for all customer schemata (Table III). To directly showcase the ranking quality of the matching results, we compare the top-1, 3, 5 accuracy between LSM and the best baseline. As Figure 4 shows, LSM constantly outperforms the best baseline for all values of k in all schemata. The accuracy gap between LSM and the best baseline is up to 0.3 for $k = 1$, 0.38 for $k = 3$, and 0.36 for $k = 5$.

Takeaways. The above results show that LSM outperforms the best baselines in predicting the matching results on both relatively straightforward public schemata and complex real customer schemata.

C. End-to-end Evaluation

In this experiment, we evaluate the full human-in-the-loop pipeline that encompasses the language model in an active learning setting as well as the attribute selection policy. The goal is to quantify the amount of human effort needed to map the entire source schema to the ISS using the human labeling cost as a proxy. We simulate the users' matching workflow and measure the number of human labels required.

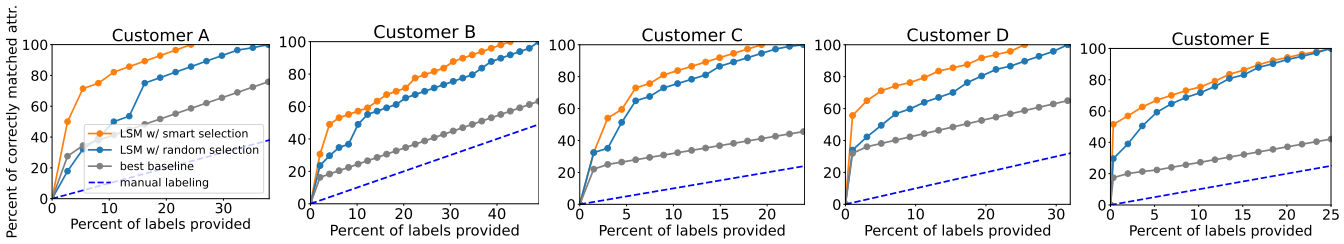


Fig. 5: Percentage of the attributes correctly matched vs. percentage of human labels provided.

At each iteration, the user first reviews the matching results suggested by LSM and marks the correct ones. Then, they select one more attribute from the unmarked portion of the customer schema and map it to the ISS (i.e., provide a new matching label). The model is then retrained and the next iteration begins. During the reviewing phase, a match is marked as correct only if for a given attribute in the customer schema, the correct corresponding ISS attribute is in the top-3 suggestions produced by LSM. For LSM, we test and compare the two different attribute selection strategies: *random selection* and *smart selection* (Section IV-E2). As for the baseline approaches, we use the same smart interaction strategy to optimize their performance. We report the percentage of customer schema attributes matched correctly as the user provides more matching labels.

Results. Figure 5 shows the schema matching progress with LSM and the best baseline as more labels are provided by the user. Overall, we observe that LSM ends up requiring as few as 19% of total customer schema attributes to be manually labeled by the user in order to map the full customer schema to the ISS, as opposed to 75% with the best baseline. In addition, with less than 5% of human labels, our model can correctly match around 70% of the customer schema attributes. Using the smart selection strategy, our model outperforms not only the best baseline but also the random selection strategy by reducing the total labels required by up to 11%. Moreover, compared with random selection strategy, the smart selection strategy boosts the performance of LSM especially when a limited number of labels are provided. As for the baseline approaches, their performance becomes similar to that of manual labeling after 10% of labels are provided, i.e., providing more labels does not help them generalize to a larger number of attributes.

Figure 5 also demonstrates the attribute reviewing cost. Since the curves in the figure present the percentage of attributes matched, the distance between each data point on the curve and 100% represents the percentage of the attributes not matched yet and thus needed to be reviewed in the next iteration. Thus, the area above the curve, denotes the total number of attributes that need to be reviewed by the user. We observe that LSM with smart selection strategy can largely reduce the reviewing cost compared to the baseline approaches.

Takeaways. As opposed to manual labeling and existing baselines, the Learned Schema Matcher with smart selection strategy requires only a small number of labels from the user to match the entire source schema. Thus, LSM provides a better overall user experience than existing approaches.

D. Impact of the BERT Featurizer

As described in Section IV-C, a key distinction of our approach is that LSM leverages the natural language capabilities of a fine-tuned pre-trained language model. The model takes as input pairs of attributes from the source and ISS schema and returns a natural language similarity score. The score is based on the input attribute names and the corresponding natural language descriptions (if available). We perform an ablation study to quantify the contribution of the BERT featurizer to the overall performance. We adopt the same setting as Section V-C.

Results. Figure 6 shows the percentage of correctly matched attributes as more labels are provided. If we disable the BERT featurizer in our model (denoted as LSM w/o BERT), the user may need to provide up to 17% more labels (Customer B) to map their full schema to ISS. The gap between the two approaches is more significant when limited number of labels are provided for all the customer schemata. This large gap demonstrates that the BERT featurizer is a critical component of our model.

Takeaways. The pre-trained language model significantly contributes to the overall performance of our method on a variety of customer schemata. It can quickly lead to higher accuracy with a limited number of labels.

E. Impact of Attribute Descriptions

The LSM can optionally take attribute descriptions contained in the schema as input. Although the ISS schema is typically well-documented, such descriptions might not be available in the customer-provided source schema. Among the five customer schemata, only two of them contain attribute descriptions (customers A and E). In this experiment, we study the impact of natural language descriptions to the overall performance.

We perform an ablation study by removing the natural language descriptions from LSM's input. We use the two customer schemata that already contain natural language descriptions (Customer A and E). Our experiment settings are the same as in Section V-C.

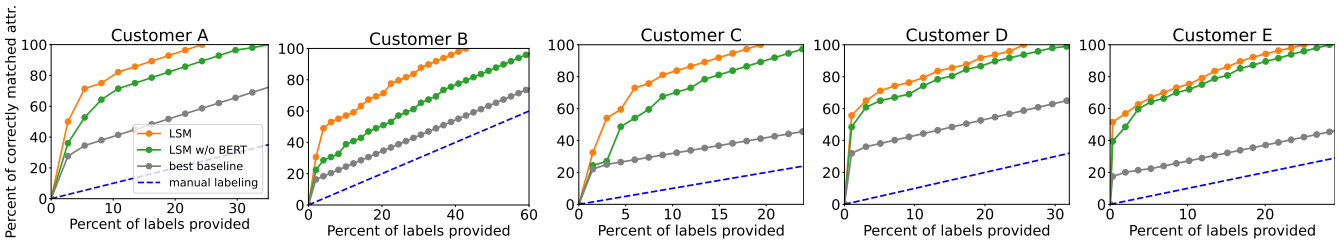


Fig. 6: Ablation study on the BERT Featurizer using various customer schemata.

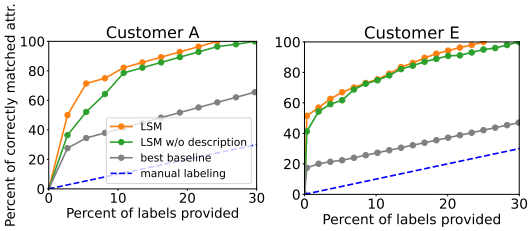


Fig. 7: Performance of LSM with and without attribute descriptions.

Results. The performance of LSM with and without natural language descriptions is shown in Figure 7. As shown in the figure, LSM still outperforms the best baseline, even when descriptions are not available. This is because by using the BERT featurizer, we are able to match attributes that are semantically equivalent but lexically different, which is not possible with the baselines. Two example matches that fall under this category are: (*OrderLine.TotalOrderLineAmount*, *Orders.items_subtotal*) and (*ProductPriceList.SuggestedRetailPrice*, *Sales_Item.full_price*).

As shown in the figure, descriptions reduce the overall human labeling cost: for both Customer A and E, removing the descriptions results in a 4% increase in the labeling cost. The effect is more pronounced when a limited number of labels are provided. For example, for Customer A, LSM can match 70% of the attributes with 5% of labels when descriptions are provided, but can only match 52% of the attributes if the descriptions are not available. Such an effect is mitigated as more labels are provided.

Takeaways. Attribute descriptions can help improve the performance of LSM, especially when a limited number of labels are provided. Without the descriptions, LSM still outperforms the best baselines on the real customer schemata.

F. Performance in the Presence of Noise

Noisy labels can be a significant challenge in modern machine learning tasks. Likewise for LSM, it is also possible that the user provides erroneous labels when matching the source attributes to the ISS. Therefore, in this experiment, we evaluate how the presence of noisy labels affects our model's overall performance.

Noise Generation. To generate noisy labels, we corrupt the ground truth matching pair (a_s, a_t) to (a_s, a'_t) by selecting a corruption attribute a'_t from ISS ($a'_t \neq a_t$) with a probability (noise rate) $n < 1$. During the human labeling phase, noise can often be generated when the user selects an attribute from ISS that is semantically close to the source attribute but is actually not the correct matching target. To simulate this noise generation process, we select the corruption attribute a'_t to be the attribute in ISS with the maximum word embedding similarity with a_s (where $a'_t \neq a_t$). We follow the same experimental setup as in Section V-C and report the human labeling cost. We use noise rates $n = 0.1, 0.2, 0.3$ to show the performance of LSM under different level of noise. As for the baseline approaches, we still report the best baseline performance with correct labels.

Results. Figure 8 shows the human labeling effort of LSM with noise rates $n = 0$ (original LSM), 0.1, 0.2, and 0.3. As expected, we observe that as we increase the noise rate, the total number of correctly matched attributes drops. In particular, the output contains 90%, 80% and 70% correctly matched attributes for $n = 0.1$, 0.2, and 0.3 respectively (shown as dashed horizontal line in Figure 8). However, even when the LSM is trained with 30% of noisy labels, it still outperforms the best baseline without noisy labels on all customer schemata.

Takeaways. The Learned Schema Matcher can still outperform the best baseline even in the presence of noisy labels.

G. Evaluating Model Response Time

Given the interactive nature of our approach, it is important to quantify how fast we can retrain our model using the provided human labels.

Response Time. Figure 9 shows the response time of LSM as the number of matching labels provided varies. Our experiments are run on an Azure cloud server with an 8-core CPU, 112 GiB of memory and a Tesla P100 GPU. We report the average time over 10 runs. The response time includes the time spent: 1) training the BERT featurizer, 2) featurizing the matching candidates, 3) training the semi-supervised model, and 4) making predictions. We observe that for small schemata (Customer A-D), the response time can be as low as a few seconds, and for larger schemata with hundreds of attributes (Customer E), the response time is in the order of one minute. This is because LSM is actually more sensitive to the number of attributes in the source schema than the number of labels provided. The reason behind this behavior is that the training

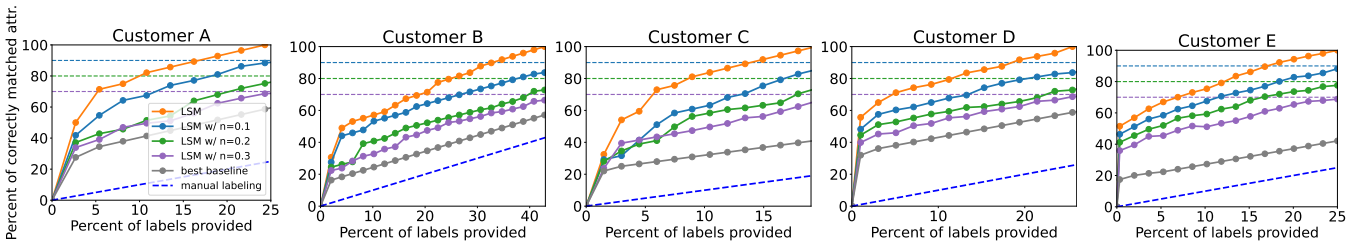


Fig. 8: Performance of LSM in the presence of noise with varying noise rates n.

time is affected more by the number of candidate pairs (which in turn depends on the number of attributes in the source schema) than the number of labels, as in a semi-supervised setting both labeled and unlabeled examples will anyway be used for training.

Takeaways. The Learned Schema Matcher has a good response time, especially for smaller source schemata. It is worth noting that the overall response time is minimal compared to the time needed to manually label the full schema.

VI. Discussion

In this section, we summarize the lessons we learned during our exploration and provide a discussion on the various design choices in the space and the limitations of our approach.

A. Tuning Hyper-parameters

When studying the existing approaches on our customer schemata and ISS, we observed that: 1) tuning hyper-parameters is challenging, and 2) the optimal hyper-parameter values are typically schema-specific. Tuning the hyper-parameters requires a metric to optimize, i.e. the accuracy on a set of labeled attributes. In our case, optimizing the baselines was an easy task because we were given the ground truth matches and had access to a large number of labels. However, when on boarding a new customer, labeled data are often not available and thus automatic hyper-parameter tuning is almost impossible. In that case, we need to rely on a human expert that fully understands the naming conventions in both the source and target schema to configure the behavior of the matching tool. We thus believe that designing matching tools with a limited number of components to tune is important for usability and performance. For this reason, we opt for an active learning approach that relies on a limited number of matching examples provided by the user.

B. ML in Schema Matching

Given that we do not maintain any historical matching records when matching customer schemata to ISS due to privacy constraints, the only resource we have for training is the limited user-provided matching labels. However, training with a small amount of data entails the risk of overfitting. We run the risk of creating a model that is overly focused on the matching patterns provided by the user and that has poor ability to generalize to other unlabeled parts of the schema.

To avoid this problem, we use an active learning approach that relies on a small number of human labels. To prevent overfitting, we leverage: 1) a light-weight model (logistic classification) with a semi-supervised framework (self-training) and 2) a pre-trained BERT featurizer on the ISS. Firstly, by making use of a semi-supervised framework, the model not only considers the labeled attributes, but also the unlabeled part of the schema to obtain a "global view" of the matches. Secondly, using a light-weight model reduces overfitting [37] as a model with fewer parameters is less likely to "memorize" the training examples and thus maintains the ability to generalize. Finally, by pre-training the BERT featurizer on ISS, we create a model that better understands the domain we operate on.

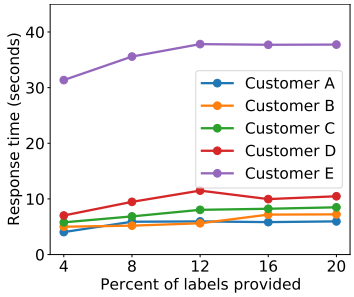


Fig. 9: Response time of the LSM as the number of labels is varied.

C. Response Time

As pointed out in [38], achieving a good response time is a challenge when both the source and target schemata are large. Previous approaches tried to address the problem of large-scale matching mainly by 1) partitioning the schemata and 2) parallelizing the matching algorithms [39].

The complexity of the Learned Schema Matcher in terms of the number of attributes is $O\left(|A_s| \times |A_t|\right)$, where $|A_s|$ and $|A_t|$ are the number of attributes in the source and target schemata, respectively. Although we propose methods to efficiently rank the matching pairs, the response time of our model still grows as the number of the input attributes increases (from several seconds for tens of attributes to the order of minutes for hundreds of attributes). We believe that this cost is still acceptable given the manual matching alternative. As part of future work, we plan to investigate potential performance optimizations when matching large customer schemata to ISS.

D. Design Space

As discussed in previous sections, prior work that either does not use ML or trains models from scratch (see Section III) has limitations in our context. We thus decided to explore whether pre-trained language models applied on a relational schema can be leveraged instead to provide more accurate results for data-free schema matching.

The potential designs would be:

- Use large language models like Codex [40] or GPT-3 [41] along with few-shot prompting.
- Fine-tune a smaller model such as BERT.
- Perform domain specific pre-training for a model like BERT (e.g, BERT for healthcare, retail, etc.)

We opted for the second option as we do have enough data to fine tune a smaller model like BERT. The other two approaches are also valid and worth exploring in future work.

E. Limitations of the LSM

Due to externally imposed constraints, LSM heavily relies on schema-only information to perform the matching. However, there is a class of problems where the user does not control or understand their schema. This often happens when the schema is generated by a third-party application with encoded names. Our methods here do not address this class of problems and likely data access would be required to solve those.

VII. Related Work

Schema matching approaches typically fall into two categories: schema-based and instance-based. Hybrid/composite matchers combine a set of individual matchers (schema and instance-based) to improve the overall accuracy [42]. We now discuss the most representative works in these two categories and compare them with the LSM. In addition, we also discuss the approaches that incorporate user feedback.

Schema-level matching Approaches. Schema information used by previous approaches includes entity and attribute names, relationships, integrity constraints, etc. Several metrics have been proposed to evaluate the similarity of the entity and attribute names between the source and target schemata. DIKE [43] uses a dictionary created by human experts to store the similarity of commonly used names. CUPID [11] uses a thesaurus to evaluate word similarity. COMA [18] utilizes a combination of individual name marchers to evaluate the similarity of two attribute names such affix, n-gram, edit distance, Soundex, etc. COMA then combines the matching results using various aggregation functions (min, max, average, etc). However, selecting a well-performing strategy is a non-trivial task and the selection often ends-up being schema-specific. (see Section VI). S-MATCH [12] uses WordNet [20] to infer synonyms for a set of words. More recently, SemProp [44] uses a semantic matcher which leverages word embeddings to find objects that are semantically related. As opposed to prior works, LSM utilizes a robust pre-trained language model to better understand the semantics of the various entity and attribute names.

Beyond linguistic information, various schema-based approaches leverage structural information as well. For example, COMA [18] computes the structural similarity between two trees. Along the same lines, CUPID [11] computes the structure similarity between two hierarchical schemata. The structural information in a relational schema is not as rich as in an XML schema or an ontology, so the Learned Schema Matcher does not currently leverage such information. It does, however, use the PK/FK relationships to optimize the interaction with the user during the labeling phase.

Instance-level matching Approaches. In addition to schema-based approaches, a variety of instance-based approaches have been proposed. SEMINT [45] associates attributes in the schema with signatures derived from instance values; LSD [13] takes a multi-strategy learning approach to exploit various information such as names, data distributions, word frequencies, etc. GLUE [46] is an extension of LSD that tackles the problem of matching a pair of ontologies. More recently, EmbDI [9] generates local embeddings specific to relational data and uses them for data integration tasks. Similarly, REMA [47] leverages graph-embeddings for relational schema matching. In a parallel context, [48] uses tabular models for table understanding tasks, and [8] leverages pre-trained language models for column annotation. As we previously discussed, these approaches are not applicable to our setting due to lack of access to the customer's data.

Incorporating User Feedback. Some schema matching approaches incorporate human feedback. As indicated in [49], these human interventions include tuning the parameters, answering questions, and reusing previous resources. In our scenario, human feedback entails providing the correct target attribute for a given source attribute. Approaches adopting this type of feedback include [11], [18], [50], [36], etc. As opposed to CUPID and COMA, we present a concrete user interaction strategy (i.e., least confident anchor) that explicitly aims to reduce the number of labels needed.

VIII. Conclusions

In this paper, we studied the schema matching problem in a setting where the records in the source data cannot be accessed due to privacy concerns. We present the Learned Schema Matcher, a schema matching algorithm that relies on active learning and a fine-tuned pre-trained language model. Our evaluation shows that our proposed approach has better performance than prior work. As part of future work, we plan to extend LSM to handle more complex mapping transformations.

References

[1] "Amount of data created, consumed, and stored 2010-2025," https://www.statista.com/statistics/871513/worldwide-data-created/, 2021.

[2] "Azure Industry Solutions," https://azure.microsoft.com/en-us/industries/, 2022.

[3] "Lakehouse for Retail," https://databricks.com/solutions/industries/retail-industry-solutions, 2022.

[4] "SAP Industries," https://www.sap.com/industries.html, 2022.

[5] "Regulation (EU) 2016/679 of the European Parliament," https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=celex%3A32016R0679/, 2016.

[6] "California Consumer Privacy Act," https://oag.ca.gov/privacy/ccpa, 2018.

[7] M. Scannapieco, I. Figotin, E. Bertino, and A. K. Elmagarmid, "Privacy preserving schema and data matching," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 653–664. [Online]. Available: https://doi.org/10.1145/1247480.1247553

[8] Y. Suhara, J. Li, Y. Li, D. Zhang, c. Demiralp, C. Chen, and W.-C. Tan, "Annotating columns with pre-trained language models," in *Proceedings of the 2022 International Conference on Management of Data*, ser. SIGMOD '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1493–1503. [Online]. Available: https://doi.org/10.1145/3514221.3517906

[9] R. Cappuzzo, P. Papotti, and S. Thirumuruganathan, "Creating embeddings of heterogeneous relational datasets for data integration tasks," ser. SIGMOD '20. New York, NY, USA: Association for Computing Machinery, 2020.

[10] Y. Li, J. Li, Y. Suhara, A. Doan, and W.-C. Tan, "Deep entity matching with pre-trained language models," *Proceedings of the VLDB Endowment*, 2020.

[11] J. Madhavan, P. A. Bernstein, and E. Rahm, "Generic schema matching with cupid," in *vldb*, vol. 1. Citeseer, 2001, pp. 49–58.

[12] F. Giunchiglia, P. Shvaiko, and M. Yatskevich, "S-match: an algorithm and an implementation of semantic matching," in *ESWS*, 2004.

[13] A. Doan, P. Domingos, and A. Levy, "Learning source description for data integration." 01 2000, pp. 81–86.

[14] T. Sahay, A. Mehta, and S. Jadon, "Schema matching using machine learning," *CoRR*, vol. abs/1911.11543, 2019. [Online]. Available: http://arxiv.org/abs/1911.11543

[15] S. Melnik, H. Garcia-Molina, and E. Rahm, "Similarity flooding: A versatile graph matching algorithm and its application to schema matching," in *Proceedings 18th international conference on data engineering*. IEEE, 2002, pp. 117–128.

[16] P. P.-S. Chen, "The entity-relationship model—toward a unified view of data," *ACM transactions on database systems (TODS)*, vol. 1, no. 1, pp. 9–36, 1976.

[17] A. Doan, A. Y. Halevy, and Z. G. Ives, *Principles of Data Integration.* Morgan Kaufmann, 2012.

[18] H.-H. Do and E. Rahm, "Coma—a system for flexible combination of schema matching approaches," in *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 2002, pp. 610–621.

[19] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext.zip: Compressing text classification models," *arXiv preprint arXiv:1612.03651*, 2016.

[20] "WordNet," https://wordnet.princeton.edu, 2022.

[21] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *Acm transactions on interactive intelligent systems (tiis)*, vol. 5, no. 4, pp. 1–19, 2015.

[22] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," 2020. [Online]. Available: https://arxiv.org/abs/2003.10555

[23] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019. [Online]. Available: https://arxiv.org/abs/1907.11692

[24] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *arXiv preprint arXiv:2107.13586*, 2021.

[25] T. Schick and H. Schütze, "Exploiting cloze questions for few shot text classification and natural language inference," *arXiv preprint arXiv:2001.07676*, 2020.

[26] P. Röttger and J. B. Pierrehumbert, "Temporal adaptation of bert and performance on downstream document classification: Insights from social media," *arXiv preprint arXiv:2104.08116*, 2021.

[27] S. Min, M. Lewis, L. Zettlemoyer, and H. Hajishirzi, "Metaicl: Learning to learn in context," *arXiv preprint arXiv:2110.15943*, 2021.

[28] T. Bansal, K. Gunasekaran, T. Wang, T. Munkhdalai, and A. McCallum, "Diverse distributions of self-supervised tasks for meta-learning in nlp," *arXiv preprint arXiv:2111.01322*, 2021.

[29] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[30] ——, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[31] J. Du, E. Grave, B. Gunel, V. Chaudhary, O. Celebi, M. Auli, V. Stoyanov, and A. Conneau, "Self-training improves pre-training for natural language understanding," *arXiv preprint arXiv:2010.02194*, 2020.

[32] B. Settles, "Active learning literature survey," 2009.

[33] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.

[34] A. Culotta and A. McCallum, "Reducing labeling effort for structured prediction tasks," in *AAAI*, vol. 5, 2005, pp. 746–751.

[35] B. Settles and M. Craven, "An analysis of active learning strategies for sequence labeling tasks," in *proceedings of the 2008 conference on empirical methods in natural language processing*, 2008, pp. 1070–1079.

[36] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos, "imap: Discovering complex semantic matches between database schemas," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, 2004, pp. 383–394.

[37] C. M. Bishop *et al.*, *Neural networks for pattern recognition*. Oxford university press, 1995.

[38] P. A. Bernstein, S. Melnik, M. Petropoulos, and C. Quix, "Industrial-strength schema matching," *ACM Sigmod Record*, vol. 33, no. 4, pp. 38–43, 2004.

[39] E. Rahm, "Towards large-scale schema and ontology matching," in *Schema matching and mapping*. Springer, 2011, pp. 3–27.

[40] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating large language models trained on code," 2021. [Online]. Available: https://arxiv.org/abs/2107.03374

[41] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[42] E. Rahm and P. Bernstein, "A survey of approaches to automatic schema matching." *VLDB J.*, vol. 10, pp. 334–350, 12 2001.

[43] L. Palopoli, G. Terracina, and D. Ursino, "The system dike: Towards the semi-automatic synthesis of cooperative information systems and data warehouses," in *ADBIS-DASFAA Symposium*, 2000.

[44] R. C. Fernandez, E. Mansour, A. A. Qahtan, A. Elmagarmid, I. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang, "Seeping semantics: Linking datasets using word embeddings for data discovery," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 2018, pp. 989–1000.

[45] W.-S. Li and C. Clifton, "SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks," *Data Knowl. Eng.*, vol. 33, pp. 49–84, 2000.

[46] A. Doan, J. Madhavan, P. Domingos, and A. Halevy, "Learning to map between ontologies on the semantic web," in *Proceedings of the 11th International Conference on World Wide Web*, ser. WWW '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 662–673. [Online]. Available: https://doi.org/10.1145/511446.511532

[47] C. Koutras, M. Fragkoulis, A. Katsifodimos, and C. Lofi, "Rema: Graph embeddings-based relational schema matching," in *EDBT/ICDT Workshops*, 2020.

[48] X. Deng, H. Sun, A. Lees, Y. Wu, and C. Yu, "Turl: Table understanding through representation learning," 2020. [Online]. Available: https://arxiv.org/abs/2006.14806

[49] A. Halevy, A. Rajaraman, and J. Ordille, "Data integration: the teenage years," in *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 2006, pp. 9–16.

[50] R. McCann, A. Doan, V. Varadarajan, and A. Kramnik, "Building data integration systems via mass collaboration," in *Intl. Workshop on the Web and Databases, USA*, 2003.