GRAM: Generative Retrieval Augmented Matching of Data Schemas in the Context of Data Security

Xuanqing Liu[★], Luyang Kong[★♣], Runhui Wang, Patrick Song,
Austin Nevins, Henrik Johnson, Nimish Amlathe, Davor Golac
{xuanqing,luyankon,runhuiw,patsong,nevinsan,mauritz,amlathe,dgolac}@amazon.com
Amazon Web Services
Seattle, Washington, USA

ABSTRACT

Schema matching constitutes a pivotal phase in the data ingestion process for contemporary database systems. Its objective is to discern pairwise similarities between two sets of attributes, each associated with a distinct data table. This challenge emerges at the initial stages of data analytics, such as when incorporating a third-party table into existing databases to inform business insights. Given its significance in the realm of database systems, schema matching has been under investigation since the 2000s. This study revisits this foundational problem within the context of large language models. Adhering to increasingly stringent data security policies, our focus lies on the zero-shot and few-shot scenarios: the model should analyze only a minimal amount of customer data to execute the matching task, contrasting with the conventional approach of scrutinizing the entire data table. We emphasize that the zero-shot or few-shot assumption is imperative to safeguard the identity and privacy of customer data, even at the potential cost of accuracy. The capability to accurately match attributes under such stringent requirements distinguishes our work from previous literature in this domain.

CCS CONCEPTS

• **Information systems** → **Extraction, transformation and loading**; • **Computing methodologies** → *Information extraction.*

KEYWORDS

Schema matching, Generative modeling, Retrieval augmented generation

ACM Reference Format:
Xuanqing Liu[★], Luyang Kong[★♣], Runhui Wang, Patrick Song,, Austin Nevins, Henrik Johnson, Nimish Amlathe, Davor Golac. 2024. GRAM: Generative Retrieval Augmented Matching of Data Schemas in the Context of Data Security. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/XXXXXX.XXXXXX

[★] First two authors contributed equally; [♣] corresponding.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
KDD '24, August 25–29, 2024, Barcelona, Spain.
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0490-1/24/08
https://doi.org/10.1145/XXXXXX.XXXXXX

1 INTRODUCTION

Today's SaaS providers that supports diverse data suppliers with ingesting, managing, and searching for potentially sensitive records, they face the challenge of dealing with inhomogeneous data schemas that naturally occur among different suppliers. For instance, an insurance company may have a table named BusinessProfile that contains columns like `num_employees`, `mailing_address_city`, `business_phone`, and so forth. However, when we ingest data records for this customer, we discover that the schema is not perfectly aligned with our internal system, which necessitates manually creating mappings such as `#employees` ↦ `num_employees`, `recipient_city` ↦ `mailing_address_city` and `phone#` ↦ `business_phone`. Unfortunately, these mappings are hardly reusable for another customer due to naming conventions and other nuances. A common scenario in data science domain is that experts spend up to several weeks to designing mappings for moderately sized tables between 100 to 200 attributes. Even with tools like Microsoft BizTalk [2] or COMA 3.0 GUI [1], the data internalization process is typically error-prone and requires multiple rounds of trials and errors.

To address the challenge of data ingestion, the research community has proposed the concept of automated schema matching as a solution to streamline the associated processes. This concept is visually represented in Figure 1. In essence, it transforms the attribute-to-attribute mapping task into a hierarchical multi-class classification problem. Given an input table with N columns, the approach involves a non-overlapping partition of the N columns into k subgroups, denoted as $N_1, N_2, \ldots, N_k$ columns, where $\sum_{i=1}^{k} N_i = N$. Each subgroup corresponds to a distinct object type, exemplified in Figure 1 by showcasing the `Profile` and `Order` object types. For each object type, a predefined attribute tree is established, comprising nodes and attributes (with leaf nodes serving as aliases for attributes). By deploying a classifier at each non-leaf node to predict the correct child node containing the relevant attribute, the methodology simplifies the process to traversing an n-nary tree. This traversal follows the direction indicated by classification results at each level. As discussed later, numerous prior works align with and contribute to this overarching framework.

Diverging from prior research efforts, we reexamine the aforementioned issue by harnessing the latest advancements in language understanding, specifically leveraging Large Language Models (LLMs) and their adept in-context learning capabilities. By encapsulating the previously outlined hierarchical classification problem within the framework of in-context learning, we proficiently repurpose LLMs as readily available classifiers through the mechanism of few-shot prompting.

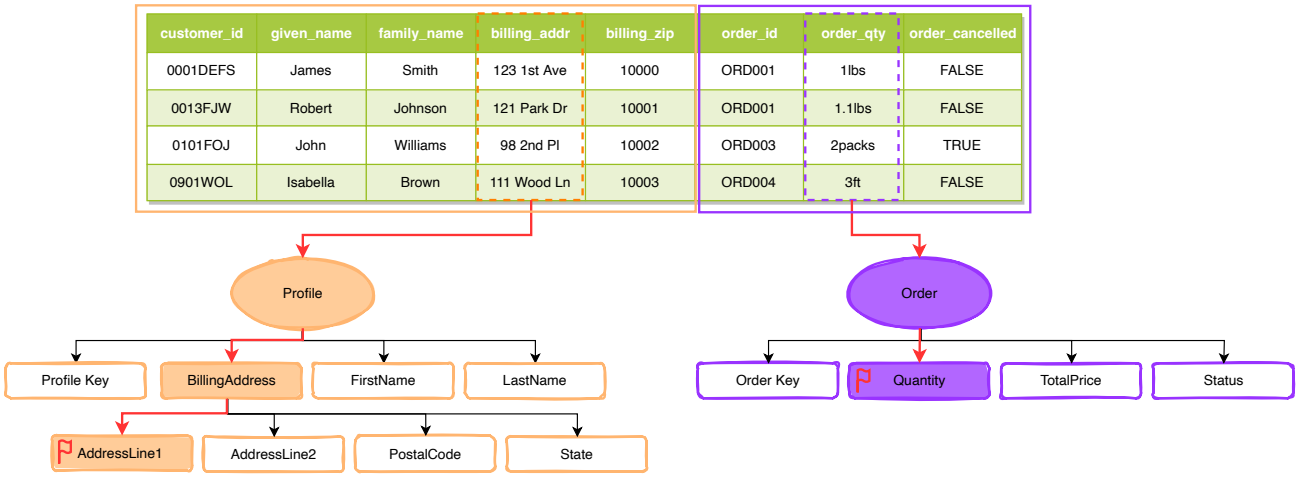

Figure 1: Illustration of the idea of hierarchical prediction in schema mapping. First, columns of input data table are partitioned and grouped into one or more *object types*, here are `Profile` and `Order` (two ellipse shapes in figure). Next, we take a column from partition group, then the column traverses through the n-ary tree based on the classification results at each level, until a root node is found (marked in red arrows). Each root node corresponds to a target attribute defined by target schema. We repeat the same process for each column until all columns are mapped to target attributes.

Column name:	*order_cancelled*
Data type:	*boolean*
Nullable:	*False*
Column meaning:	*A variable indicating if the order has been cancelled by customer*
Values:	[False, False, True, ...]
Length:	112,000,000

Figure 2: An example of how an individual attribute in the schema look like. We highlight the required field (column name) with shades, and all other fields (data type, nullable, column meaning, values, length, etc.) as optional.

Our primary contributions can be summarized as follows:

(1) We address the automated schema matching problem within the context of data privacy, employing a novel perspective that incorporates zero-shot and few-shot predictions.
(2) Our solution seamlessly integrates the recent surge in Large Language Models (LLMs). We conduct a comprehensive benchmarking exercise across various open-source and proprietary LLMs to assess their performance.
(3) Introducing a dynamic prompt selection method based on input characteristics, our approach not only enhances inference speed but also augments the in-context learning accuracy of LLMs.
(4) Beyond the conventional scope of schema matching, our solution incorporates object type detection and unique key detection. These additional components transform the standalone schema matching module into a more feature-complete data-table ingestion service.
(5) We rigorously benchmark the accuracy of our methodology against relevant approaches using both public and production-quality, synthetic datasets. Particularly noteworthy is the utilization of datasets designed to mirror realistic workloads in various industrial applications.

2 A BRIEF HISTORY OF SCHEMA MATCHING RESEARCH

2.1 Pioneering solutions

LSD. [10] stands as one of the pioneering machine learning-based schema matching frameworks. It formulates the matching problem as a multi-class classification challenge. Notably, LSD employs an ensemble of classifiers to enhance accuracy, incorporating a nearest neighbor Whirl learner, a Naive Bayesian learner, a name matcher, and a county-name recognizer. Classifiable under the dichotomies outlined earlier, LSD falls within the category of one-to-one matching based on linguistic features and is trained on both schema and instances.

CUPID. [21] is considered one of the first general-purpose schema matching systems with a focus on feature completeness. It employs linguistic features and predefined rules to match pairs or groups of attributes. CUPID's core idea is to determine the highest weighted similarity ($wsim$) between two attributes using the formula $wsim = w_{\text{struct}} \cdot ssim + (1 - w_{\text{struct}}) \cdot lsim$, where $ssim$ is the structural similarity score, and $lsim$ is the linguistic similarity score. As an early work from the 2000s, CUPID's feature extractors are basic compared to modern language models. However, CUPID falls short in extracting insights from column values, missing opportunities to address ambiguities inherent to schema-data alone.

Similarity Flooding. [23] introduces a method to transform the schema matching problem into computing the fixpoint over graph propagation. Initially, the SQL2Graph operation converts a pair

of table schemas into two graphs for matching. The StringMatch operation assigns initial similarity scores over nodes in the graphs. Subsequently, the SFJoin operation, essentially a label propagation algorithm over a directed graph, is applied to iteratively obtain the fixpoint. Attribute pairs are then pruned based on a specified threshold. Similar to CUPID, the text similarity metric appears basic by contemporary standards, considering only the length of common prefixes and suffixes between two strings. Additionally, it does not incorporate column values, rendering it suboptimal for challenging use cases.

COMA/COMA++/COMA 3.0. [3, 9, 22] constitute a line of research that focuses on combining matching algorithms in a flexible manner, thus presenting an orthogonal approach to the methods discussed earlier. The notable aspect of this software system, along with the underlying algorithms, is its provision of a user-friendly interface for executing multiple matching algorithms iteratively, allowing for human intervention. Additionally, the software extends beyond merely matching two schemas, encompassing a comprehensive workflow that includes storage, match execution, mapping processing, and user connectivity.

S-MATCH. [12] shares similarities with the COMA family as it is an open-source framework that provides multiple built-in matching algorithms. Users can readily adopt the system and make necessary extensions as required.

2.2 Modern solutions based on neural nets

Sahay et al. [27] presented a straightforward hybrid approach incorporating both schema data and column values, applicable to both one-to-one and one-to-many mappings. Employing extensive feature engineering, the authors utilize self-organizing maps or K-means clustering to cluster similar attributes. Consequently, during testing, an attribute is paired with the nearest cluster, and the best attribute within that cluster is selected based on the minimum edit-distance principle.

Hättasch et al. [13] introduced a neural embedding-based method, making a significant contribution with a two-level matching scheme. The first level involves table matching, followed by attribute matching at the second level. The matching process entails computing the similarity, such as cosine similarity, between two embeddings derived from textual information, including column name, data type, comments, etc.

LSM. [37] is a schema matcher that leverages pre-trained language models. Notably relevant due to its recent development and utilization of modern transformer-based language models, LSM employs a finetuned BERT featurizer at its core. This featurizer transforms pairs of schema information into similarity scores, considering two attributes as a match if the similarity score surpasses a specified threshold. The BERT featurizer undergoes finetuning based on human-provided labels. Once the finetuning process is complete, the model is prepared to generalize to new schema pairs.

2.3 Goals of our solution

Given the recent surge in large language models (LLM) and generative AI (GenAI), it is intuitive to explore the application of the "emergent abilities" described by Wei et al. [34] to the realm of schema matching. Our decision to integrate these advancements stems from the belief that LLMs offer language understanding and reasoning capabilities approaching human levels. With this upgrade, we anticipate a transformative impact on how we conceptualize the similarity between two data schemas. In this paper, we aim to elevate the quality and usability of schema matching systems along the following dimensions:

Enhanced Language Understanding with Efficient Inference. When framing the schema mapping as a natural language processing (NLP) problem, one observes that the advancements in solutions reviewed over the past two decades are intricately linked to the evolving landscape of language modeling. Early solutions relied on string similarity and hand-crafted features, often complemented by shadow models such as linear classifiers, naive Bayes classifiers, or k-means clustering methods. In contrast, contemporary solutions leverage deep learning text featurizers like Word2vec, GloVe, FastText, and BERT, extracting text similarity scores within an end-to-end paradigm. This paper benefits from superior language understanding capabilities offered by open-source large language models, specifically the FLAN-T5 family. Additionally, we introduce methods to expedite inference speed while preserving accuracy, a critical consideration for handling large-scale data prevalent in industrial applications.

Minimal Training Data Dependency. The conventional approach to utilizing finetuned language models, as seen in works such as [37], involves the collection of a substantial amount (ranging from 10^3 to 10^4) of human-labeled data to calibrate the language classifier using certain loss functions. In contrast, we adopt zero-shot and few-shot learning, also known as in-context learning (ICL) in Large Language Models (LLM) literature, reducing the dependency on data quantity. This attribute is particularly significant in addressing contemporary concerns regarding data security and privacy, as it obviates the need for accessing and annotating large volumes of customer data.

Comprehensive Feature Integration. The solution presented in this paper transcends the boundaries of a mere schema matcher, evolving into an end-to-end service fueled by language models. This service harmonizes disparate data sources, rendering them into uniformly searchable data records. Central to this endeavor are two supplementary components around the attribute mapper: the object type detector and column key detector. Both components leverage language models to enhance their functionalities. Specifically, the object type detector identifies the appropriate object type (target schema) for a subset of input columns; the attribute mapper establishes connections between each input attribute and a unique target attribute; and finally, the key detector designates one of the attributes as the unique key, enabling the ingestion of the entire table with duplicates removed.

3 BACKGROUND KNOWLEDGE

3.1 Large language model

Language understanding stands as a fundamental capability to showcase advanced artificial intelligence. Pretrained language models (PLM) [8] have proven to be a powerful and scalable approach for embedding general knowledge into transformer-based neural networks. The conventional application of PLMs involves finetuning them on domain-specific datasets collected from experts, leading to the creation of one model for each task. This practice, however, limits usability in scenarios where high-quality datasets are scarce. With the growing demand for more generalized language models, researchers have identified a promising avenue. By scaling up both the size of the pretraining corpus and the parameter count of the language model, adhering to scaling-up principles [14, 31], and subsequently finetuning the model on diverse tasks using instructional prompts [5, 20, 24], a robust language model emerges. This model exhibits the capability to comprehend natural instructions with strategic prompt engineering.

3.2 Retrieval augmentation

Standalone Large Language Models (LLMs) encode world knowledge within their model weights, placing smaller scale models at a disadvantage when tasked with memorizing intricate language corpora that demand hundreds of billions of parameters. Additionally, Retrieval Augmented Generation (RAG) [17] enhances model capacity by integrating an external knowledge search engine. RAG excels in consolidating domain-specific knowledge that proves challenging to memorize from a web corpus using LLM-based readers. In this context, RAG emerges as particularly well-suited for the schema matching task, given the often vaguely defined connections between two attributes.

4 GRAM: GENERATIVE RETRIEVAL AUGMENTED MATCHING

4.1 Motivating example

To explore how instruction finetuned large language models can be prompted with few-shot examples to effectively match similar attribute pairs against unrelated ones, we have developed a straightforward demonstration using Anthropic Claude via the AWS Bedrock SDK [29], as illustrated in Fig. 3. In this instance, we instruct Claude to match the column name `contact_name` with an example value `Amazon.com Inc.` against other profile-related attributes, such as `FirstName`, `LastName`, `HomePhoneNumber`, `EmailAddress`, and so on. The prompt adheres to the standard in-context learning paradigm: it begins with a formulation of the problem statement and the success goal, followed by a list of choices and subsequently a list of examples with ground-truth labels. Finally, the query example is appended at the end.

It is noteworthy that this particular problem is non-trivial to answer accurately. The column name `contact_name` alone can refer to both `FirstName`, `LastName`, and `BusinessName`. The resolution of this ambiguity is dependent on examining the example value `Amazon.com Inc.`, where the model deduces that `BusinessName` is the sole appropriate match. Generally, the schema matching problem proves to be highly challenging, even for domain experts.

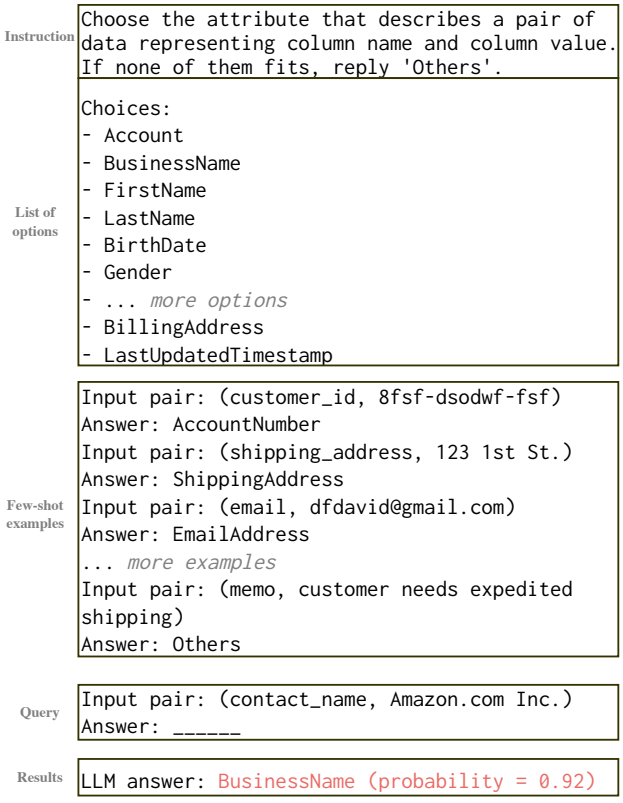

Figure 3: An illustrative example outlining the concept of prompting Large Language Models (LLMs) to match a source attribute (e.g., contact_name for Amazon.com Inc.) to a list of 15 target attributes is provided for clarity.

For instance, the column name `state` may represent a U.S. state name or serve as an equivalent to the word "`status`," without additional information discernible from the value section.

We hypothesize that leveraging large language models equipped with commonsense knowledge represents a promising approach to effectively address the challenge of schema understanding. Concurrent research indicates that gigantic language models boasting 100+ billion parameters demonstrate human-level reading comprehension and near-human-level logical reasoning capabilities [34]. This hypothesis serves as the driving force behind our decision to incorporate an instruction-finetuned large language model as the central component of our schema matching service.

However, translating the concept illustrated in Fig. 3 into live production proves to be non-trivial. The target processing speed of schema matching service is 10 transactions per second (TPS) per host, each equipped with inference-optimized GPU devices, typically Nvidia T4 or Nvidia A10. Benchmark results reveal that, without any optimization, the naive solution achieves less than 6 TPS per host.

In the subsequent sections, we delve into strategies for accelerating inference latency, or equivalently, increasing the TPS count. While various techniques exist for optimizing Large Language

Model (LLM) inference, including intelligent decoding methods [16, 28], improved memory access patterns [6, 15], and model compression and quantization [11, 35], among others, this paper introduces an orthogonal approach specifically tailored for schema matching acceleration, known as *prompt compression*. Our approach is inspired by the observation that the inference time T for transformer-based LLMs is quadratic to the input length L_{input}, i.e., $T = O(L_{\text{input}}^2)$. This is because the self-attention output is computed as

$$X_{\text{out}} = Softmax\Big(\frac{Q^T K}{\sqrt{d}}\Big)V, \tag{1}$$

where $Q = W_q^T X_{\text{in}}$, $K = W_k^T X_{\text{in}}$, $V = W_v^T X_{\text{in}} \in \mathbb{R}^{d \times L_{\text{input}}}$ are attention query, key and value matrices respectively; X_{in} and X_{out} are the inputs and outputs of attention block. The bottleneck for computing Eq. (1) is matrix multiplication $Q^T K$ with a complexity of $O\big(L_{\text{input}}^2 d\big)$. As a result, it is most beneficial to minimize the input text length L_{input} to accelerate the inference speed. At the same time, according to the prompt structure in Fig. 3, we can decompose L_{input} to

$$L_{\text{input}} = L_{\text{instruct}} + N \cdot \big(\bar{L}_{\text{option}} + M \cdot \bar{L}_{\text{example}}\big), \tag{2}$$

where L_{instruct} is the length of instruction text, $\bar{L}_{\text{option}}$ is the average length of destination attribute name, $\bar{L}_{\text{example}}$ is the average length of each example; N is the number of options in prompt, and this is equivalent to number of mapping destinations; M denotes number of examples per option (M-shot prompting).

Our empirical observation indicates that listing all possible matching destinations in each Large Language Model (LLM) query is unnecessary. Instead, by employing a combination of techniques detailed in the following sections, we can effectively eliminate a substantial number of irrelevant options and examples associated with the source attribute. This results in a significant reduction in the values of N and M in Eq. (2). Consequently, a smaller value for L_{input} is achieved.

4.2 NER-based destination filter

Named Entity Resolution (NER) serves as a potent method for extracting and recognizing categorical information from free texts. For example, consider the text:

"Jim bought 300 shares of Acme Corp. in 2006."

A successful NER task would label "Jim" as **Person**, "Acme Corp." as **Organization**, and "2006" as **Time**. With NER models, we can move beyond merely matching the column data type, as seen in prior works (e.g., [3]), to introduce a new destination attribute filter denoted as $\mathcal{F}_{\text{NER}}$. This filter retains only those destination attributes that share both the same data type and data category as the source attribute. Mathematically,

$$\begin{aligned}\mathcal{F}_{\text{NER}}\big(S|\langle k, v\rangle\big) = \big\{o|o \in S \wedge \text{DType}(o) = \text{DType}(v) \\ \wedge\ \text{NER}(o) = \text{NER}(v)\big\},\end{aligned} \tag{3}$$

in which S is the set of all destination attributes; $\langle k, v\rangle$ is the input data pair containing attribute name k and attribute value v; DType is the data type extraction operator by reading column metadata; NER denotes a named entity resolution model we trained on schema matching tasks.

To highlight the potential impact of the filter $\mathcal{F}_{\text{NER}}$, let's revisit the prompt depicted in Fig. 3. Post-filtering, the available options are significantly reduced to just two - `Account` and `BusinessName`, down from the original 15 options. This reduction is attributed to the NER model's recognition of the input value `Amazon.com Inc.` as an organization name, while the remaining options fall into distinct data categories such as phone numbers, person names, addresses, etc.

We implemented a Named Entity Recognition (NER) model tailored for schema matching tasks, closely adhering to standard practices outlined in the literature ([18] and references therein), with a few noteworthy modifications. First, we defined a more fine-grained label space. Traditional NER models are typically trained on a coarser label space, where the target category "address" encompasses street addresses, cities, states/provinces, and even countries. However, this standard practice limits usability in schema matching tasks where the goal is to determine if a column storing zip codes matches another column storing cities, even if both are mapped to the "address" category with traditional NER models. The second modification we introduced pertains to the training loss. In traditional NER models, the loss is computed on a per-token basis, treating it as a token-level classification problem. This approach is justified when the input is a sentence containing multiple entities, and the goal is to predict the text span encompassing all entities along with their labels. In contrast, our approach computes the loss at the sequence level, treating it as a sequence-level classification problem. Our approach is valid under the assumption that there is only one entity for each input sequence, a condition that is widely applicable to schema matching tasks.

Implementation-wise, we choose RoBERTa-base [19] as the backbone model to initialize training. An input sequence for training or inference consists of a few samples ranging from 1 to 6 elements sampled from a column, serialized as a list of values

$$\langle s\rangle\{\text{value}_1\}[SEP]\{\text{value}_2\}[SEP]\cdots\{\text{value}_k\}\langle/s\rangle, 1 \le k \le 6, \tag{4}$$

$\langle s\rangle, \langle/s\rangle, [SEP]$ are special tokens in vocabulary, $\{\text{value}_i\}$ is the i-th sample of the column, and To enhance robustness and generalizability, we employ random sampling, selecting $1 \le k \le 6$ examples to construct a training sequence. For additional training details, please refer to the appendices.

4.3 Double-RAG filter

The NER-based filter discussed in the previous section assesses the coherence of two attributes based on column values. Essentially, two attributes can be considered a good match when their corresponding column values are mapped to the same Named Entity Recognition (NER) label. In this section, we adopt a different perspective and gauge the inter-attribute coherency through the semantic similarity of column names. Our approach draws inspiration from the efficacy of the retrieval augmentation (RAG) technique in enhancing accuracy across various Large Language Model (LLM) applications (refer back to Section 3.2 for additional background). What sets our use of the RAG technique apart is that we not only search for the best possible options but also seek the most suitable few-shot examples for a particular option, giving rise to the term "Double-RAG." Consequently, both the options and the few-shot examples in the prompt dynamically change with different input

queries. In this regard, our proposed prompting method can be viewed as another instance of automatic prompt tuning [39], with the goal of minimizing the prompt length while maintaining robust reasoning abilities.

We maintain two databases to store the options and examples for each option. Let $D_{\text{opt}} = \{o_1, o_2, \ldots, o_N\}$ be the database containing options (destination attribute names) and

$$D_{\text{ex}} = \begin{Bmatrix} e_{11} & e_{12} & \cdots & e_{1M} \\ e_{21} & e_{22} & \cdots & e_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ e_{N1} & e_{N2} & \cdots & e_{NM} \end{Bmatrix}$$

be the database containing all available examples, in which e_{ij} is the j-th example of i-th option. We provide M examples for each of the N options, totaling NM examples. To understand how members in each database look like, we can pick a few examples from both. Suppose o_1 ="PhoneNumber", then we can store e_{11} ="phone", e_{12} ="tel", e_{13} ="phone_number" and so on. In principle, we should collect a diverse number of examples that give LLM enough idea of how the concept of option o_i is like.

Building upon the databases D_{opt} and D_{ex}, we further incorporate a similarity measure $\text{sim}(x, y) \in [0, 1]$, supported by either machine learning models or traditional string similarity algorithms. The trade-off in this context revolves around whether the critical factor is the semantic understanding ability from machine learning models and the computational budget available in practice.

For instance, we anticipate the similarity value of sim(phone, tel) to be closer to 1.0, but none of the string similarity algorithms yields expected results in such cases without the aid of external thesaurus dictionaries. This is because the words "phone" and "tel" share only one common character, "e", resulting in a bi-gram Jaccard similarity of 0.0. In contrast, even the simplest GloVe [1] embedding indicates a significant cosine similarity of 0.50, not to mention more sophisticated BERT-based embeddings. The experiments will revisit the choice of similarity measures with further details.

Equipped with two databases D_{opt} and D_{ex}, and a similarity measure $\text{sim}(x, y) \in [0, 1]$, we are ready to formulate the way we short-listing the options together with their exemplars:

$$\begin{aligned} \widehat{D}_{\text{opt}} &= \left\{ o_i | o_i \in D_{\text{opt}} \wedge i \in \text{Top}_{k_1}\left(\text{sim}(o_i, q)\right) \right\}, \\ \widehat{D}_{\text{ex}} &= \left\{ e_{ij} | e_{ij} \in D_{\text{ex}} \wedge o_i \in \widehat{D}_{\text{opt}} \wedge j \in \text{Top}_{k_2}\left(\text{sim}(e_{ij}, q)\right) \right\}. \end{aligned} \tag{5}$$

Above we defined $q = \langle k, v \rangle$ as the key-value query pair; $\widehat{D}_{\text{opt}}$ and $\widehat{D}_{\text{ex}}$ as two compressed databases by filtering out the dissimilar choices and exemplars to $k_1 \ll N$ and $k_1 \cdot k_2 \ll NM$ elements, respectively.

4.4 Other components

For the sake of comprehensive service functionality, we have designed two additional components that work in conjunction with the core Large Language Model (LLM)-based attribute mapper to execute the data integration task. These components are the object type detector and key detector. While the primary focus of this paper revolves around attribute mapping, as an integral part of the overall system, we briefly introduce their functionalities as follows.

[1]URL: https://nlp.stanford.edu/projects/glove/, we used glove.42B.300d.zip version.

Object Type Detector. This component serves as a preprocessor for the attribute mapper. Its role is to partition the columns of the input table into multiple subgroups, each representing a uniform topic (also referred to as an object type, as illustrated in Fig. 3), such as personal profile, customer order, issue ticket, and so forth. It is important to note that real-life input tables can be a combination of multiple topics, and the two databases $D_{\text{opt}}/D_{\text{ex}}$ used in the LLM attribute mapper are determined by the topic. Hence, the system needs to cluster the columns and identify the topic of each cluster before proceeding to the attribute mapping stage. Our implementation of the object type detector adheres to standard practices: we first convert the input table into CSV format, then serialize its header into a text string. Next, the entire string is tokenized to train a BERT-based multi-class classifier with per-token level cross-entropy loss. During the inference stage, we group columns with the same predicted labels together into a subgroup, effectively partitioning the entire table.

Key Detector. This component functions as a postprocessor for the attribute mapper. Its role is to enhance the mapping results with a few keys for searching or de-duplication. The underlying concept aligns with the LLM-based attribute mapper introduced earlier; in fact, we reuse the same LLM model with a different prompting method, thereby improving hardware utilization rates. Initially, we allow users to customize heuristic rules to filter out columns unlikely to serve as potent keys. A simple illustrative rule could be any column name with the pattern "*_id". Users have the flexibility to chain multiple rules together to strike a balance between recall and precision. Ideally, the aim is to retain all valid keys while minimizing the list of candidates to query LLM.

4.5 Workflow

Bringing all the components together, we illustrate the entire workflow in Fig. 4. At a high level, the custom data slated for ingestion first undergoes the object type detector, where columns are partitioned and labeled with one of the pre-defined object types. In the second stage, each individual column, along with its associated object type, is formatted as a query to the attribute mapper. The outcome of stage 2 is the predicted destination attribute generated from the instruction-finetuned Large Language Model (LLM). Finally, in stage 3, the key detector assigns one or more keys to the mapped attributes, ensuring that the ingested table is accompanied by keys for searching and data de-duplication.

5 EXPERIMENTS

We have designed a series of experiments to assess the effectiveness of the Large Language Model (LLM)-based attribute mapper. Specifically, we aim to address the following questions:

(1) How does this retrieval-augmented LLM solution compare with traditional solutions in terms of accuracy?
(2) What benefits are observed in throughput when incorporating the prompt compression techniques discussed in Sec. 4.2 and 4.3?
(3) What is the most practical choice among LLM backbones of different sizes?
(4) How does the number of k-shot examples influence end-to-end accuracy?

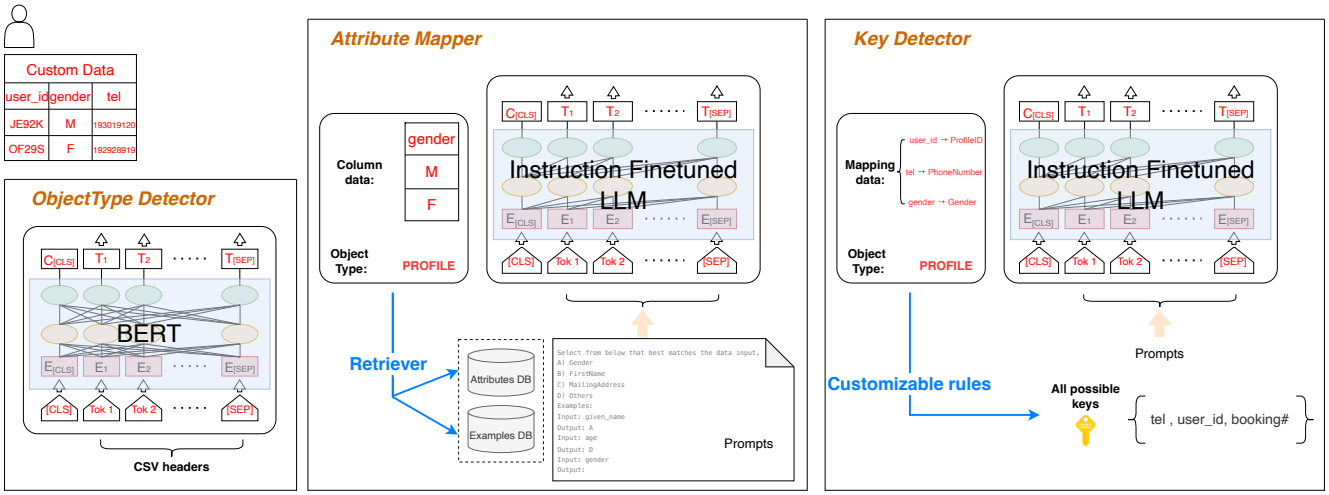

Figure 4: Architecture and workflow of GRAM.

We initiate the process with dataset preparation, which is undeniably one of the most challenging steps given the multi-decade history of schema mapping research since the 2000s. Numerous datasets referenced in early works are either lost or unpublished. Despite these challenges, we have managed to reconstruct a substantial collection of evaluation sets from diverse domains, as listed below.

- **Personal Contacts:** This domain revolves around personal and business profiles, which are commonly found in customer databases, employee databases, or social media records. In total, there are 1400 columns.
- **Sales:** This domain encompasses sales and transaction records for a merchant, such as airline bookings and shopping checkouts. In total, there are 400 columns.
- **Products:** This domain comprises databases storing products or services available in the market, including airlines, hotel rooms, groceries, etc. In total, we have collected 200 columns.
- **Issue Tickets:** This domain includes issue tickets, totaling 330 columns.

PII Disclosure: None of the datasets mentioned above contain any real identity information. This includes metadata such as column names and/or data types (`first_name(str)`, `dob(str)`, `zip(int32)`, `address_line1(str)`, `sales_amount(float32)`). The column values are all synthetic or randomly generated.

We have implemented and deployed our Large Language Model (LLM)-based schema matching system using PyTorch [25], based on the FLAN-T5 model. For very early methods, such as LSD [10] and CUPID [21], for which no first-party implementation is available, we implemented their methods following the ideas presented in the original papers. For other similar works, such as Similarity Flooding [23], we were unable to replicate the algorithm due to the lack of critical details; hence, we exclude them from our experiments. When benchmarking throughput, we executed all programs on hardware comprising 4× Nvidia A10 GPUs (each with 24GB of memory), 24 physical CPU cores, and 192GB of memory.

5.1 Comparing LLM-based schema matching with baselines

Algorithms	Mean accuracy (%) in domain				
	Person	Sales	Products	Tickets	Avg.
LSD [10]	73.0	63.6	61.8	74.7	68.3
CUPID [21]	52.2	50.6	39.8	62.7	51.3
COMA 3.0 [1]	56.6	48.7	69.0	50.6	56.2
LSM [37]	81.0	78.5	70.2	71.4	75.3
GRAM (ours)	**91.9**	**80.3**	**92.3**	**90.3**	**88.7**

Table 1: Comparing the accuracy numbers across different domains among traditional algorithms, deep neural nets based algorithms, and our LLM based algorithm.

In this study, we conduct a comprehensive comparison of various schema matching algorithms. The primary objective is to assess the comparative advantages of machine learning (ML)-based and large language model (LLM)-based algorithms in comparison to conventional rule-based methods. The outcomes are presented in Figure 1. Based on the experimental findings, several observations can be made: 1) a noteworthy improvement in accuracy is evident when employing an instruction-finetuned LLM, surpassing even contemporary pretrained language model (LM) approaches, such as LSM [37]; 2) it is generally observed that embedding-based cosine similarity complements lexical similarity metrics effectively. Our internally developed implementation of the LSD method exhibits noteworthy performance, particularly when utilizing an ensemble of word embeddings and the Sorensen-Dice [30] string similarity algorithm.

5.2 Effect of NER-based filter and Double-RAG filter

Settings	Mean accuracy (%) in domain				
	Person	Sales	Products	Tickets	Avg.
No filter	83.0	78.7	81.1	90.5	83.3
+NER	**92.7**	**86.6**	88.1	85.4	88.2
+Double-RAG	89.4	74.9	89.6	**91.6**	86.4
+Both filters	91.9	80.3	**92.3**	90.3	**88.7**

Table 2: Comparing the testing accuracy with and without filters. Filters do not change model accuracy in a consistent direction.

We explore the impact of the Named Entity Recognition (NER) filter and the Double-RAG filter on both inference speed and accuracy. In principle, activating either of these filters introduces false negatives, as they possess the capability to exclude positive selections and crucial instances intended to assist reasoning during test time. The discernible effect of these filters on accuracy is detailed in Table 2. Remarkably, a consistent decline in accuracy

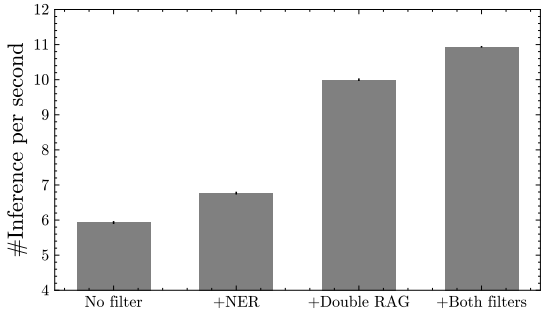

Figure 5: Inference speedup due to NER and double-RAG filters. With double-RAG filter, we keep $k_{\text{opt}} = 4$ options and $k_{\text{ex}} = 1$ examples for each of the 4 options. Error bars are provided but barely visible.

is not readily observable upon activating the filters. We posit that the incorporation of high-quality filters aids the Large Language Model (LLM) in decision-making by eliminating evidently incorrect option items and unrelated few-shot examples. Despite introducing false negatives through occasional removal of correct options and valuable few-shot examples, the overall impact appears to enhance the LLM's decision-making process. Meanwhile, the filters demonstrate a noticeable acceleration in inference speed, as illustrated in Figure 5 across typical workloads simulated with synthetic datasets.

5.3 Does larger language model perform better in schema-matching?

In this section, we investigate the correlation between larger Large Language Models (LLMs) and enhanced accuracy in schema matching tasks, as observed in related works across various domains (e.g., [4, 5, 33]). To explore this relationship, we conduct an experiment comparing downstream accuracy using FLAN-T5 [5] as the backend with varying LLM sizes (Small-80M, Base-250M, Large-780M, XL-3B, XXL-11B). Evaluation settings, including filter hyperparameters, remain consistent across all assessments. The average accuracy across four domains plotted against model sizes is depicted in Fig. 6.

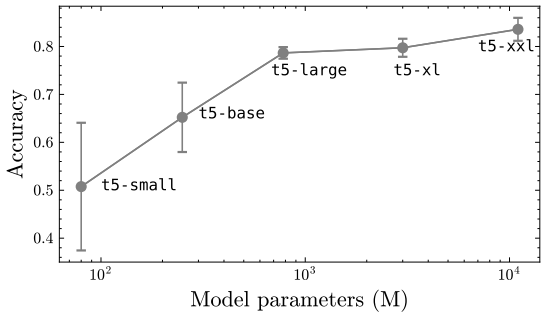

Figure 6: Comparing the matching accuracy among different sizes of instruction finetuned models, accuracy is averaged across domains and datasets therein.

5.4 Effect of number of shots to matching accuracy

In this experiment, we explore the effect of adding more ground-truth examples in the prompt for LLM to conduct in-context learning. It is widely believed that more diverse few-shot examples typically converts to higher accuracy. However, we noticed that the return of adding more samples diminishs very quickly beyond 1-shot setting. In Fig. 7 we showed an significant accuracy boost moving zero-shot (73.05%) to 1-shot (83.58%), whereas the accuracy improvement beyond 1-shot is not significant given the error band. This finding led us to configure our model to consume just one example per class label.

5.5 Launch strategy, user experience, and learning

Due to the service availability requirement, we reserved three nodes on each region with instance type `ml.g5.2xlarge` as well as several

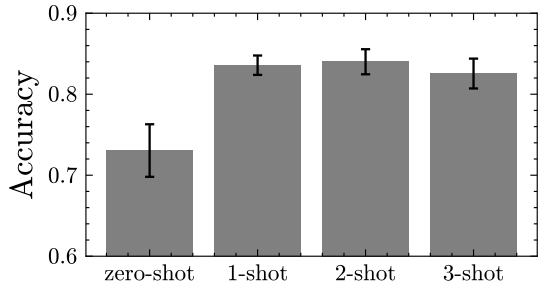

Figure 7: Comparing matching accuracy under varying k-shot examples in the prompt.

back-up instance types (`ml.g4dn.4xlarge` etc.) so that in case one instance type isn't available at a certain region we will still be able to serve the request at similar throughput. Another challenge is the volatility of payloads: some payload consists of small schema (≤ 30 attributes) while in extreme cases this number can be as large as 120 attributes. To prevent the requests from queuing up, we distribute the attribute mapping requests originating from the same table to at least 3 hosts with an automatic scaling-up policy.

We present the schema matching service to customers by enabling a human-in-the-loop process: schema matcher never generates the final mapping result in one shot, instead, customers have the chance to examine the predicted mapping table as well as other machine generated metadata (such as searchable keys) and correct any incorrect predictions on the fly. While we are not permitted to record the user activities (e.g. number of modifications they made when composing the schema mapping) due to data privacy, internal studies show that with our LLM aided schema mapping, the amount of human efforts measured by editing operations reduced by 90%.

Lastly, we also learned from some negative feedback, mostly about the instability of prediction results. Although the model performs reliably on canonical input schema, it predicts wrongly when we slightly change the column name. For instance, by adding a meaningless prefix "XYZ_" to all column names, the mapping accuracy drops under certain inputs (although not very common). We attribute this as adversarial examples and we plan to focus on this problem as the next research topic.

6 DISCUSSION

The schema matching task has been under investigation for over a decade. We posit that the fundamental challenge stems from comprehending attributes in highly heterogeneous environments. The rapid evolution of large language models has elevated language understanding capabilities to unprecedented levels. In light of this advancement, we address the longstanding and intricate problem using this innovative tool, yielding encouraging results. Looking ahead, our future direction involves contemplating the optimal approach for task adaptation to the backbone model, with the aim of further enhancing matching accuracy.

REFERENCES

[1] [n. d.]. Software: COMA 3.0 | Database Group Leipzig. https://dbs.uni-leipzig.de/research/projects/coma.

[2] [n. d.]. Using BizTalk Mapper. https://learn.microsoft.com/en-us/biztalk/core/using-biztalk-mapper. Last updated: 02/01/2021.

[3] David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. 2005. Schema and ontology matching with COMA++. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. 906–908.

[4] A Chowdhery, S Narang, J Devlin, M Bosma, G Mishra, A Roberts, P Barham, HW Chung, C Sutton, S Gehrmann, et al. 2022. PaLM: Scaling Language Modeling with Pathways (No. arXiv: 2204.02311). arXiv.

[5] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416* (2022).

[6] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems* 35 (2022), 16344–16359.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018). arXiv:1810.04805 http://arxiv.org/abs/1810.04805

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[9] 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.

[10] AnHai Doan, Pedro M Domingos, and Alon Y Levy. 2000. Learning Source Description for Data Integration.. In *WebDB (informal proceedings)*. 81–86.

[11] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323* (2022).

[12] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. 2004. S-Match: an algorithm and an implementation of semantic matching. In *The Semantic Web: Research and Applications: First European Semantic Web Symposium, ESWS 2004 Heraklion, Crete, Greece, May 10-12, 2004. Proceedings 1*. Springer, 61–75.

[13] Benjamin Hättasch, Michael Truong-Ngoc, Andreas Schmidt, and Carsten Binnig. 2022. It's AI Match: A Two-Step Approach for Schema Matching Using Embeddings. *arXiv preprint arXiv:2203.04366* (2022).

[14] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).

[15] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 611–626.

[16] Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*. PMLR, 19274–19286.

[17] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[18] Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. 2020. A survey on deep learning for named entity recognition. *IEEE Transactions on Knowledge and Data Engineering* 34, 1 (2020), 50–70.

[19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[20] Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. 2023. The flan collection: Designing data and methods for effective instruction tuning. *arXiv preprint arXiv:2301.13688* (2023).

[21] Jayant Madhavan, Philip A Bernstein, and Erhard Rahm. 2001. Generic schema matching with cupid. In *vldb*, Vol. 1. 49–58.

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

[23] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. 2002. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings 18th international conference on data engineering*. IEEE, 117–128.

[24] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems* 35 (2022), 27730–27744.

[25] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[26] Erhard Rahm and Philip A Bernstein. 2001. On matching schemas automatically. *VLDB journal* 10, 4 (2001), 334–350.

[27] Tanvi Sahay, Ankita Mehta, and Shruti Jadon. 2020. Schema matching using machine learning. In *2020 7th International Conference on Signal Processing and Integrated Networks (SPIN)*. IEEE, 359–366.

[28] Andrea Santilli, Silvio Severino, Emilian Postolache, Valentino Maiorca, Michele Mancusi, Riccardo Marin, and Emanuele Rodolà. 2023. Accelerating Transformer Inference for Translation via Parallel Decoding. *arXiv preprint arXiv:2305.10427* (2023).

[29] Amazon Web Services. 2023. AWS Bedrock. https://aws.amazon.com/bedrock/. [Online;].

[30] Thorvald Sorensen. 1948. A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons. *Biologiske skrifter* 5 (1948), 1–34.

[31] Yi Tay, Mostafa Dehghani, Jinfeng Rao, William Fedus, Samira Abnar, Hyung Won Chung, Sharan Narang, Dani Yogatama, Ashish Vaswani, and Donald Metzler. 2021. Scale efficiently: Insights from pre-training and fine-tuning transformers. *arXiv preprint arXiv:2109.10686* (2021).

[32] Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*. 142–147. https://www.aclweb.org/anthology/W03-0419

[33] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).
[34] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682* (2022).
[35] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*. PMLR, 38087–38099.
[36] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. 2017. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412* (2017).
[37] 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.
[38] Wenxuan Zhou, Sheng Zhang, Yu Gu, Muhao Chen, and Hoifung Poon. 2023. Universalner: Targeted distillation from large language models for open named entity recognition. *arXiv preprint arXiv:2308.03279* (2023).
[39] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910* (2022).

A DICHOTOMIES OF SCHEMA-MATCHING

Before delving into the historical overview of schema matching research, it is pertinent to highlight the dichotomies that characterize existing ideas, as elucidated in the review paper by Rahm and Bernstein [26]:

- *Schema-only* or *schema+instances*: A matching system is categorized as schema-only when it relies solely on schema data without considering column values. In contrast, schema + instances matching incorporates both schema and column values. In the context of modern machine learning, the former is often referred to as *zero-shot*.
- *Element-wise* or *structural* matching: Element-wise matching entails pairing individual attributes, while structural matching involves matching groups of attributes together.
- *Linguistic-based* or *rule-based*: Linguistic-based matching encompasses ideas that employ machine learning or non-machine learning-based text similarity metrics to determine attribute equivalence. Conversely, rule-based matching relies more on schema constraints, such as data types, value ranges, uniqueness, etc.
- *One-to-one* or *many-to-many*: A one-to-one matcher consistently connects one attribute to another, whereas a many-to-many matcher has the capability to associate more than one attribute as the source or destination.
- *Self-contained* or *auxiliary information*: A self-contained matcher operates autonomously, while a matcher incorporating auxiliary information can leverage external knowledge, such as dictionaries, global schemas, previous matching decisions, and user input.

Having elucidated the aforementioned dichotomies, our focus now shifts to a comprehensive review of both seminal contributions and the current state-of-the-art in the field of schema matching.

B IMPLEMENTATION DETAIL OF NER FILTER

We follow the identical modelling steps as standard BERT-based NER [7, 32]. The model architecture (as well as input structure) is illustrated in Fig. 8. We highlight that the input sequence to the NER model is not a single attribute value, but a list of example values in variable length k with noises (such as empty values, invalid values, etc). Adding external noise helps robustifying the model inference, as it simulates the outliers often encountered in real applications.

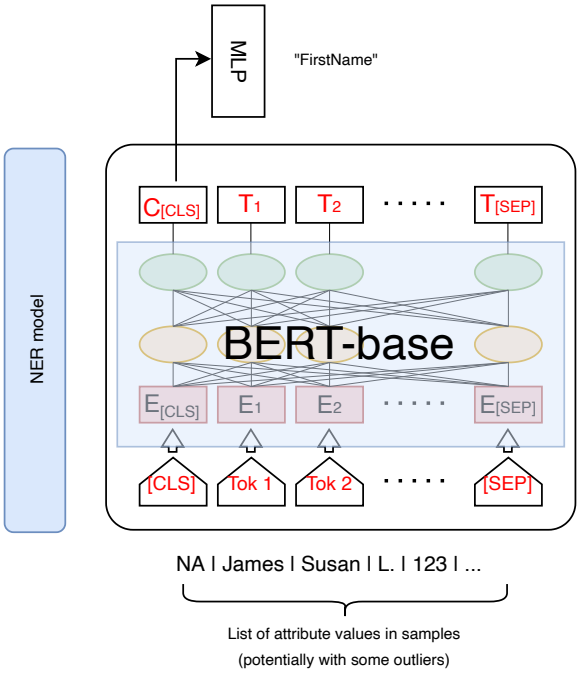

Figure 8: NER model design and input structure.

Different from previous design of NER models, here we consider a much more broader and finegrained labels, specifically, we consider following categories:

- FirstName: Indicates people's first name.
- MiddleName: Indicates people's middle name.
- LastName: Indicates people's last name.
- FullName: Indicates people's first name + middle name (optional) + last name.
- BusinessName: Indicates a company name. Such as *Amazon.com inc.*
- ProductName: Indicates the name (title) of a product. Such as *Apple Iphone 13 pro 128GB.*
- Dates: Indicates a date string in any format compliant to ISO8601. Such as 1989-02-27.
- Gender: Indicates people's gender identities.
- Email: Indicates a valid email address, such as *xyz@gmail.com.*
- URL: Indicates a valid URL, such as *https://www.google.com*
- CreditCardNumber: Indicates a credit card number string.
- Timestamps: Indicates a full datetime in at least seconds. Such as `2001-03-14T19:43:01.342998`.
- AddressLine: Indicates address line 1.
- City: Indicates a city name.
- Province/State: Indicates a province or state name.
- Counties: Indicates a country name.
- Zip/PostalCode: Indicates a zip code.

- PhoneNumber: Indicates a phone/mobile number with optional area code.
- Prices: Indicates product prices, such as 12.29$.
- Currencies: Indicates currency symbol, such as $, JPY, CAD, etc.
- Weights/units: Indicates the weight or unit of products, such as 2lbs, 15ct.
- FreeText: The fall-back category not captured by any of the above labels.

A majority of data categories can be synthesized by random generation. Part of the data are collected from the internet / open-source datasets; while we also collected some useful examples with LLM prompting, similar to the idea of UniversalNER [38]. In total we have 10,000 data entries. During training, we leverage the idea of mixup [36] to further augment the training dataset, in case there are multiple different categories in the input, we also create soft-labels when computing the training loss.

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