LakeHopper: Cross Data Lakes Column Type Annotation through Model Adaptation

Yushi Sun
HKUST
Hong Kong, China
ysunbp@cse.ust.hk

Nan Tang
HKUST (GZ)
Guangzhou, China
nantang@hkust-gz.edu.cn

Lei Chen
HKUST / HKUST(GZ)
Hong Kong / Guangzhou, China
leichen@cse.ust.hk

ABSTRACT

Column type annotation is crucial for various data-related tasks, such as data cleaning, data integration, and data visualization. The latest advancements in column type annotation solutions are resource intensive: use encoder language models fine-tuned on a substantial amount of well-annotated columns within a particular set of tables, i.e., a *source* data lake. In this paper, we study whether we can adapt an existing pre-trained LM-based model to a new (i.e., *target*) data lake, with the main goal of reducing the number of annotated columns required from the new data lake. However, cross-data lake column type annotation is challenging due to the problem of identifying the *source-target knowledge gap*, the difficulty of selecting the most informative target training data in order to fill the source-target knowledge gap, and the hardness of fine-tuning a model without losing the source-target shared knowledge. To address these challenges, we propose LakeHopper (https://github.com/ysunbp/LakeHopper), a novel cross-data lake model adaptation framework. LakeHopper provides a novel language model (LM) interaction strategy to identify and resolve the knowledge gap. We further introduce a cluster-based data selection scheme to discover the most helpful unannotated target columns to be annotated. Finally, we propose an incremental fine-tuning mechanism to adapt the source model to the target data lake. Our experimental results validate the effectiveness of LakeHopper on two different data lake transfers under both low-resource and high-resource settings.

PVLDB Reference Format:
Yushi Sun, Nan Tang, and Lei Chen. LakeHopper: Cross Data Lakes Column Type Annotation through Model Adaptation. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ysunbp/LakeHopper.

1 INTRODUCTION

Column Type Annotation (CTA) refers to the process of labeling the semantic data type (*e.g.,* Film, Scientist) for columns in a dataset or a data lake. It is crucial for various tasks, including data mining [57], data integration [27, 46, 60, 68], data cleaning [31, 47], and data visualization [4, 8]. However, in real-world datasets, column types are frequently absent, particularly in web tables or tables deposited into data lakes. This underscores the growing necessity for semi-automated methods to annotate missing column types. Such annotations are vital for enhancing the accuracy of data analytics across a wide range of applications.

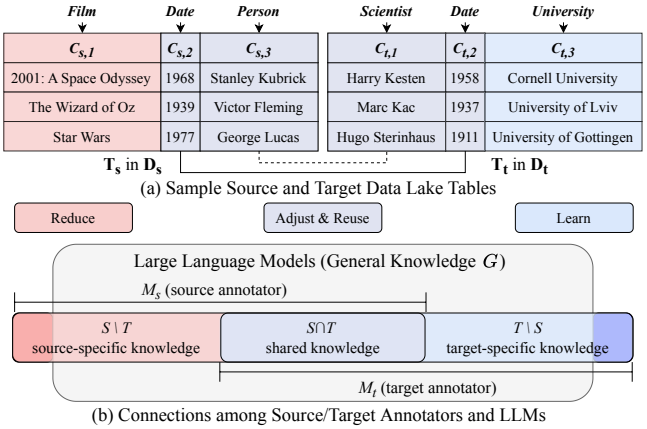


Figure 1: (a) Cross data lakes CTA example. (b) Knowledge of fine-tuned models (S and T for source and target annotators) and generic models (G).

Existing Solutions and Limitations. CTA is commonly performed on a set of tables, *e.g.,* a data lake.

Single data lake. Many open-source libraries and commercial data preparation systems rely on manually defined heuristics or patterns to detect a limited set of semantic types [2–4, 8]. Some approaches [17, 23, 42–44, 51] perform fuzzy lookups to annotate columns with the help of knowledge graphs to reduce human cost. One key problem is that their approaches are prone to noisy table content [33], limiting their adaptability to diverse datasets.

Recently, there has been extensive research into pre-trained-LM-based (PLM-based) annotation models ("**annotator**" for short), particularly those utilizing PLMs, such as [20, 28, 55, 57, 63]. However, these approaches are not cost-effective due to the substantial requirement for well-annotated column types to train these models. For instance, the VizNet dataset [10, 68] used by these approaches contains over 80k table annotations. Obtaining the annotations on such a large scale requires substantial efforts to design heuristics, perform annotations, and verify the labels, incurring significant expenses when we encounter new data lakes [36].

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

Cross data lakes. When annotating column types for new data lakes, it is not ideal to directly apply an annotator trained from one data lake to an unseen one. This is due to the differences between source and target data lake content and the discrepancies between the source and target semantic type sets. [36] shows that, when directly applying a trained Sato [68] annotator to an unseen PublicBI data lake [5, 61], the accuracy drops significantly, from 90% to 35%.

Example 1. *Figure 1(a) shows two tables, $T_s = (C_{s,1}, C_{s,2}, C_{s,3})$ from a* source *film data lake D_s and $T_t = (C_{t,1}, C_{t,2}, C_{t,3})$ from a* target *mathematician data lake.*

*Let M_s be the annotator trained on the source data lake. It is unlikely to correctly annotate $C_{t,3}$ of T_t which is the **university** since the film data lake D_s does not contain this type of data. As for the columns that share similar content ($C_{s,2}, C_{s,3}, C_{t,1}, C_{t,2}$), M_s might also wrongly annotate $C_{t,1}$, because the semantic type set of D_t has a finer-grained requirement (**Scientist** instead of coarse-grained type **Person**). This implies that even for shared knowledge between D_s and D_t in M_s, we need to adjust it properly for reusing in D_t.* □

Example 1 illustrates that the annotation performance might drop when we directly apply the source annotator to the target data lake. The main **limitation** of employing learned annotators is that users are typically required to create a significant volume of ground truth annotations for retraining the annotators for target data lakes. This process can severely restrict the practical application of these methods in real-world contexts. Although STEER [36] studies applying labeling functions (LFs) to reduce the human annotation burden on new data lakes, it heavily depends on human domain knowledge when designing the LFs that hinders its adoption. Even though advancements in large decoder-LMs like GPTs [15] have demonstrated impressive capabilities in generating responses with extensive domain-agnostic knowledge, they often underperform specifically trained PLM-based annotators because of the lack of task-specific fine-tuning, and domain-specific knowledge (*i.e.,* $(T \setminus S) \setminus G$ in Figure 1(b)). Besides, due to the generative nature of their decoder design, without resource-intensive fine-tuning, they often hallucinate by providing out-of-domain annotations, which makes them unsuitable for classification tasks such as CTA (detailed in Section 4.3.1).

Opportunities. Many trained PLM-based annotators exist for some widely used data lakes. Hence, it is natural to study whether we can reuse them for new data lakes. Specifically, given an annotator M_s trained on a source data lake D_s, our goal is to adapt it as M_t for a target data lake D_t, with a minimum of number training data from D_t. The new annotator needs to (see Figure 1): **i) reduce the knowledge** regarding the source-data-lake-specific columns (*e.g.,* $C_{s,1}$); **ii) adjust and reuse the shared knowledge** between D_s and D_t, some of the knowledge cannot be reused directly due to the potential discrepancies between source and target semantic type sets (*e.g.,* $C_{s,3}, C_{t,1}$); and **iii) learn new knowledge** on target-domain-specific columns (*e.g.,* $C_{t,3}$). We have identified three opportunities to realize the target annotator, corresponding to the above three needs (see Figure 1(b)).

(1) **Source-target knowledge gap:** We need to effectively identify the knowledge in the target data lake that the source annotator does not learn well (*i.e.,* $T \setminus S$) and the shared knowledge that needs to be adjusted to fit in the target data lake.
(2) **Target training data selection.** We need to select a minimal target train data to adapt annotators effectively.
(3) **Fine-tuning without forgetting.** We need to design a fine-tuning strategy that adapts annotators from source to target (*i.e.,* remove $S \setminus T$ and learn $T \setminus S$) while adjusting and reusing shared knowledge (*i.e.,* $T \cap S$).

Table 1: Comparing existing methods and LakeHopper.

	Generalizability	Accuracy	Out-of-domain
PLM-based	low	high	zero
LLM-based*	high	low	high
LLM-based+	low	high	low
LakeHopper	**high**	**high**	**zero**

* and + are LLMs without or with domain-specific fine-tuning.

Challenges. Unfortunately, none of the three opportunities is trivial. Next, we will discuss three main challenges, in response to the above three opportunities.

(1) The source-target knowledge gap is hard from two aspects. From the data perspective, as suggested by [35], a model could perform well on certain types in the source data lake while performing significantly worse on the same types in the target data lake due to the difference in table content. From the model perspective, due to the black-box nature of the PLM-based approaches, there is no explicit indication of how well the model will perform on the target data lake.

(2) Target training data selection is hard because it often involves a trial-and-error exercise to figure out which training data samples are effective for a certain task, which is not well explored for the problem of CTA.

(3) The target data lake may contain new semantic types or lack some of the source data lake semantic types [36], therefore, fine-tuning a source annotator for the target data lake without losing useful learned knowledge requires special treatment during the fine-tuning process.

Our Proposal. We propose LakeHopper, a novel framework for cross data lakes CTA model adaptation, with the main objective of minimal annotations required. It first tests on which target columns, the source annotator performs poorly, which is achieved by asking the source annotator to annotate selected target columns and then employing an LLM with general knowledge, such as ChatGPT, to check whether the annotations are correct. Only those annotations that are considered inaccurate by LLMs, will be asked for annotations.

As shown in Table 1, in comparison with existing state-of-the-art solutions, LakeHopper has the advantage of achieving high cross data lake generalizability, and high domain-specific annotation accuracy, with no out-of-domain hallucination, while requiring minimal domain adaptation cost. Its counterparts, which can be classified into two different types have their distinct drawbacks in completing cross data lake CTA: 1) PLM-based approaches [20, 28, 55, 57, 63] adopt lightweight encoder-LMs (*e.g.,* BERT) with domain-specific fine-tuning to generate column embeddings for CTA. These methods typically suffer from low cross data

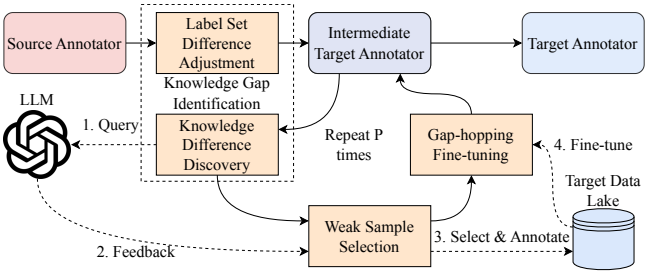


Figure 2: The System Architecture of LakeHopper.

lake generalizability and high demand for domain adaptation training data; 2) LLM-based approaches [21, 34, 37, 70] transform CTA into QA and leverage the QA capabilities of LLMs (*e.g.,* ChatGPT) to perform annotations. These approaches *without domain-specific fine-tuning* are highly generalizable thanks to the world knowledge contained in the core LLMs and can perform zero-shot and few-shot domain-agnostic annotation. However, the annotation quality is generally poor since the task itself is non-trivial and they are likely to provide out-of-domain answers. *LLMs with domain-specific fine-tuning* require high training time and GPU resources thus resulting in low generalizability. LakeHopper combines the merits of PLMs and LLMs approaches: 1) obtain high generalizability through the general knowledge of the LLMs (*e.g.,* ChatGPT); 2) achieve high accuracy via the domain-specific fine-tuning strategy of the PLMs approaches; and 3) avoid introducing out-of-domain annotations.

Figure 2 overviews LakeHopper. It first uses the source annotator to annotate columns from the target data lake and then query LLMs (step 1), in order to discover the target columns that the source annotator is not good at, *i.e.,* the source-target knowledge gap $(T \setminus S) \cap G$ and the shared knowledge that needs to be adjusted due to type set discrepancy (**Challenge 1**). To reduce annotation cost, we introduce a cluster-based approach to identify the most informative training samples from the target data lake that the source annotator fails (step 2) and forward them for annotations (step 3) (**Challenge 2**). We introduce an iterative fine-tuning strategy (step 4) to gradually adapt the source annotator for the target data lake (remove $S \setminus T$ and learn $(T \setminus S) \cap G$), without forgetting the shared knowledge between the source and target data lake $(T \cap S)$, while obtaining the target domain-specific knowledge $((T \setminus S) \setminus G)$ (**Challenge 3**).

We have conducted extensive experiments to validate the superiority of LakeHopper over the pure PLM/LLM-based CTA approaches. Specifically, we obtain average performance gains under low-resource settings for the three state-of-the-art PLM approaches with 11.7% and 41.0% for the two F1 scores. Moreover, our comprehensive experiments show the superiority of our approach over LLM-based approaches when jointly considering the accuracy and efficiency: our approach achieves comparable performance with fine-tuned LLM-based approaches, with 27 to 131 times faster in training speed.

Contributions. We have made the following contributions.

(1) We identify the strengths and weaknesses of the source PLM-based annotator *w.r.t.* the target data lake (Challenge 1).

Table 2: Notations.

Notations	Descriptions
D_s, D_t	the source and target data lakes
S_s, S_t	the source and target label type sets
M_s, M_t	the source and target annotators
$\bar{M}_{t,l-1}$	the intermediate target annotator in round i
L_s, L_t	the output layers of source and target annotators
n_s, n_t	the sizes of the source and target type sets
C	the notation for column
y	the ground truth semantic type of C
N_t	the number of target training samples allowed
N_f	the number of fine-tuning epochs
N_e	the number of early stop threshold
P	the number of iterations allowed
$D_{f,l}$	the weak columns selected for fine-tuning in the l-th interation
$\Phi(v)$	the confidence level of v
δ	the confidence threshold
A_l, $\tilde{A}_l$	the query column set and the difficult column set at the l-th iteration

(2) We design a data clustering strategy, which accurately tackles the weakness of the source annotator (Challenge 2).

(3) We design an incremental fine-tuning without forgetting mechanism to gradually adapt the source annotator for the target data lake, which greatly improves the annotation performance of existing CTA works (Challenge 3).

(4) We conduct extensive experiments on two data lake transfer pairs to show that LakeHopper outperforms all the baselines under multiple application settings.

2 PROBLEM AND SYSTEM ARCHITECTURE

2.1 Problem Statement

Let T be a table, $\{C_1, C_2 \ldots, C_m\}$ be a set of columns of T to be annotated, and S be the pre-defined semantic type set, where the semantic types are disjoint types without hierarchy selected from ontology for the practical application needs. Hence, each column C_i is mapped to only one type in S.

Table Column Type Annotation (CTA). The problem is to design a function $f()$ that maps a column C_i to a semantic type as $\bar{y}_i = f(C_i) \in S$, such that each cell in column C_i is an instance of $\bar{y}_i$ [32].

As discussed earlier, in order to implement the function $f()$, traditional methods typically rely on manually defined rules [2–4, 8] or propose fuzzy lookup rules to annotate column types with the help of knowledge graphs [17, 23, 42–44, 51].

Single Data Lake Column Type Annotation. Given a data lake D that consists of a collection of tables. The problem is to annotate the column semantic types in each table in the data lake ($T \in D$).

Because of the large number of tables in a data lake, manually designing a function $f()$ for each table is not feasible and cannot effectively capture the deep semantics of all the tables. Hence, the state-of-the-art solutions [28, 55, 57, 63] use PLMs that train one language model $M()$ as a learned function, such that this single model $M()$ can annotate all tables in the data lake.

Cross Data Lakes Column Type Annotation. Given a model M_s fine-tuned on a source data lake D_s, a target data lake D_t, and a fixed budget N_t of training samples on the target data lake, the problem is to select at most N_t samples (each sample is a (C_i, y_i) pair) from the target data lake, and then use these training samples to obtain a transformed model M_t for the target data lake, such that M_t achieves the best column type annotation accuracy on the target data lake.

Note that, as a beginning step of exploring cross data lake CTA, we consider only a given single source annotator M_s in this work. We leave the selection and collaboration of source models as a future work (see Section 7). The key notations are summarized in Table 2.

2.2 The System Architecture of LakeHopper

As shown in Figure 2, the overall structure of LakeHopper contains the following components: knowledge gap identification, weak sample selection, and gap-hopping fine-tuning.

2.2.1 Knowledge Gap Identification. In general, the knowledge gap between the source and target annotators includes two aspects: **label set difference** and **knowledge difference**.

The **label set difference** arises from the discrepancy in the label type sets S_s and S_t. For instance, the VizNet and Semtab2019 datasets contain 78 and 275 types respectively. Adapting the annotator from the VizNet dataset to the Semtab2019 dataset requires adjustment to the output layer.

The **knowledge difference** is shown in Figure 1, where the source annotator trained on the film data lake is unlikely to correctly annotate the university names in the target data lake since the source annotator is unlikely to know the university names. However, identifying such inherent knowledge differences is difficult due to the black-box nature of the deep-learning-based approaches. Therefore, we need to design a low-cost probing approach that can effectively analyze the knowledge that the current annotator lacks.

The problems to be studied in this step are defined below.

Label Set Difference Adjustment. Given a source annotator M_s and the source and target label sets S_s and S_t, we aim at reusing some of the weights of M_s in the intermediate target annotator $\bar{M}_{t,0}$, such that $\bar{M}_{t,0}$ inherits part of the annotation ability of M_s and can perform CTA for the exactly matched types on target data lake.

Knowledge Difference Discovery. Given $\bar{M}_{t,l-1}$ at the l-th iteration and the target data lake D_t, we identify target data samples $\tilde{A}_l$ that are difficult for $\bar{M}_{t,l-1}$ to correctly annotate.

2.2.2 Weak Sample Selection. After identifying the knowledge gap, how to effectively select the target training data to fine-tune the annotator remains a challenge. As mentioned in Section 1, the training data selection often involves a trial-and-error exercise to find which training samples are effective for fine-tuning, which is not feasible if we need to check all the data samples to get a holistic view of the target data lake. As a result, we need to define an effective weak sample selection scheme, with a comprehensive view of the target data lake, which identifies the informative training data that can best improve the current annotator.

Weak Sample Selection. Given a set of samples $\tilde{A}_l$ where $\bar{M}_{t,l-1}$ cannot confidently annotate and the target data lake D_t, we want to identify other samples that $\bar{M}_{t,l-1}$ is unlikely to correctly annotate from D_t.

2.2.3 Gap-hopping Fine-tuning. As shown in Figure 2, the model adaptation process proceeds in iterations. During the process, the knowledge of the intermediate target annotator is expected to be refined continuously until it converges to the target annotator. During the process, the change in the intermediate target annotator's knowledge is dynamic. Therefore, we need to design a fine-tuning process that facilitates the incremental update of the annotator. However, there is a key challenge called catastrophic forgetting [22, 40, 41, 50, 64]: During the incremental learning process, the annotator is likely to forget useful information about previous data when learning new data. In our case, if each iteration, we naively only fine-tune the intermediate target annotator with the weak columns selected in that iteration, the annotator is likely to overfit the weak samples identified in the current iteration, and thus cannot hop toward the desired target annotator. Therefore, properly designing a fine-tuning mechanism for the intermediate target annotator to gradually adapt to the target annotator without forgetting useful knowledge is required.

Gap-hopping Fine-tuning. Given $\bar{M}_{t,l-1}$ and training sample sets $D_{f,0}, \ldots, D_{f,l}$, we need to design fine-tuning strategy such that $\bar{M}_{t,l}$ learns the knowledge from the current samples $D_{f,l}$, while preserving the knowledge in previous samples $D_{f,0}, \ldots, D_{f,l-1}$.

3 LAKEHOPPER

Next, we describe our proposed algorithms in LakeHopper.

3.1 Knowledge Gap Identification

3.1.1 Label Set Difference Adjustment. Before the source annotator can be applied to annotate columns on the target data lake, the first step is to adjust its out layer to fit in the label type set S_t. The output layer L_s of the source annotator is a matrix with shape $m \times n_s$, where m is the output dimension of the PLM core and $n_s = |S_s|$, by default, $m = 768$ for the most commonly used BERT base models [18]. To inherit the annotation ability of the source annotator to the target annotator as much as possible [67], we first transfer the PLM weights to the target annotator and then adjust the target annotator output layer L_t as shown in Figure 3: for all the shared types such as the *company*, *year*, and *team* in our example, we map the corresponding weights to the corresponding entries of the target annotator output layer and randomly initialize the rest of the weights in L_t. We denote the intermediate target annotator as $\bar{M}_{t,0}$. In this way, we finish the adjustment of the label set difference.

The adjusted intermediate target annotator has not been trained on the target data lake yet. To equip the annotator with a basic annotation ability to avoid cold-start issues, we randomly sample a small subset of training samples $D_{f,0}$ from the target data lake, annotate, and train the annotator with the sampled data. The aim is to guide the partially randomly initialized output layer of $\bar{M}_{t,0}$ to get familiar ('warm-up') with the target domain CTA task.

3.1.2 Knowledge Difference Discovery. After the 'warm-up' stage, we now identify the knowledge difference between $\bar{M}_{t,0}$ and M_t. As

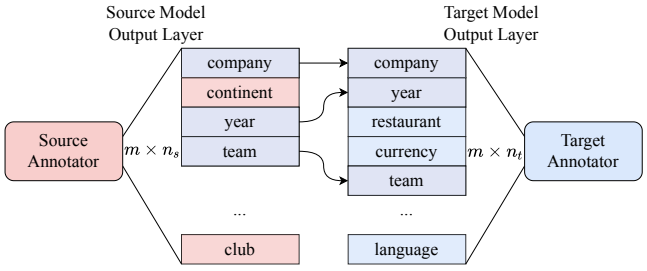


Figure 3: An illustration of label set difference adjustment.

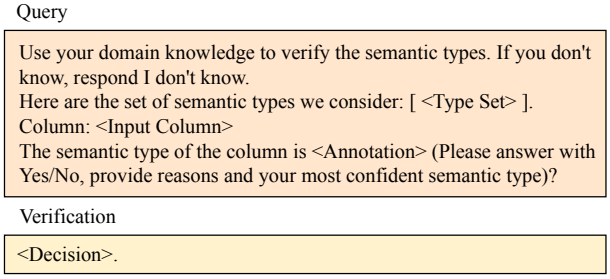


Figure 4: The LLM query verification template.

shown in Figure 2, the Knowledge Difference Discovery step is the first step in an iteration, for ease of demonstration, we now consider the l-th iteration. We first get the output embedding $v \in \mathbb{R}^{n_t}$ from the output layer of $\bar{M}_{t,l-1}$ for each column C in the target data lake D_t. We randomly sample a subset D_l of columns that have not been sampled from D_t yet and denote the corresponding embeddings of the columns in D_l as v's. Based on the embedding, we learn about the confidence level $\Phi(v)$ of the $\bar{M}_{t,l-1}$ regarding its own annotation by computing the infinity norm of softmax scores:

$$\Phi(v) = |\text{Softmax}(v)|_\infty = \max_i |\frac{e^{v_i}}{\sum_{k=i}^{n_t} e^{v_i}}|, \quad (1)$$

Based on the confidence score $\Phi(v)$ of each embedding in D_l, we can now select the annotations made by $\bar{M}_{t,l-1}$ to query LLM to know about the knowledge gap of the current intermediate target annotator with the help of the general knowledge of LLM. We denote the confident threshold as δ. If $\Phi(v) \geq \delta$, we do not include the corresponding column C in the query set A_l for the LLM. Otherwise, we include it in the query set A_l for the LLM. The motivation is that when the PLM $\bar{M}_{t,l-1}$ is 'confident' enough for its annotation, we do not rely on the verification provided by the LLM. We consider this from two different aspects: **1) Efficiency aspect**: Only query the LLM with the columns where the current annotator $\bar{M}_{t,l-1}$ is not confident can reduce the overall query trials made and thus reduce the overall monetary and time costs induced by calling LLM API keys;

2) Effectiveness aspect: The knowledge contained by the LLM tends to be general. On specific tasks like CTA, if the annotator is confident enough about its decision, it is more likely that the annotator is correct. Since the LLM tends to be domain-agnostic and thus may not have enough in-domain knowledge on the target data lake, the annotator is equipped with in-domain knowledge during the incremental fine-tuning.

With the selected query set A_l in hand, we can now proceed to query the LLM so as to utilize the general knowledge of the LLM to discover the knowledge gap in $\bar{M}_{t,l-1}$. Based on the LLM query template developed by [34], we construct our LLM query template as shown in Figure 4. Specifically, considering the current column C_x in A_l, we first list the task description that asks the LLM to verify the annotation given by $\bar{M}_{t,l-1}$. Then we provide the semantic type set S_t at <Type Set>. After that, we concatenate the cells in column C_x into a string and provide it at <Input Column>. Then we provide the annotation given by $\bar{M}_{t,l-1}$ regarding the input column at <Annotation>. The LLM receives our query and provides the following verification: <Decision>. Note that our LLM query template is different from the previous work that employs ChatGPT to perform CTA [34] in the sense that our template provides the annotations given by the annotator and only asks the LLM to verify the annotations (True/False question), while the template in [34] asks the LLM to select the most appropriate semantic type from a type set (Multiple Choice question). The difficulty level of our template is much lower than theirs since the size of the type set is normally very large. Selecting the most appropriate type from a large type set is challenging for domain experts as discussed in [63].

We record the <Decision> (Yes/No/I don't know) made by LLM and denote it as d_x. Based on the decision d_x obtained, we further classify the columns in A_l into two types: If d_x = No or I don't know, we consider the column C_x as difficult, since either the annotator $\bar{M}_{t,l-1}$ is likely to be wrong on this column or the LLM does not have sufficient knowledge to annotate this column. Both cases should be identified by the current annotator since it does not have enough confidence regarding its own annotation and either 1) it makes the wrong annotation ($(T \setminus S) \cap G$) and part of $T \cap S$ that needs adjustment or 2) the column content itself is out of the domain of the general knowledge of LLM and thus should be learned through the domain-specific fine-tuning ($(T \setminus S) \setminus G$). If d_x = Yes, we consider it to be less difficult, since although $\bar{M}_{t,l-1}$ is not confident enough regarding its own annotation, the annotation is correct in the realm of the general knowledge of the LLM. We denote the set of columns that are classified as difficult as $\tilde{A}_l$. These columns are regarded as representative samples of the knowledge difference between $\bar{M}_{t,l-1}$ and M_t. By interacting with the LLM, we are able to probe the knowledge inside a black-box annotator with a relatively low cost, which resolves the challenge in Section 2.2.1.

3.2 Weak Sample Selection

Although querying the LLM can help us identify the difficult columns of the current annotator $\bar{M}_{t,l-1}$, it is not feasible to query the whole set D_t due to the monetary and time costs induced by the LLM API. Therefore we select a query set A_l to perform the query operations in Section 3.1.2. However, we want to maximize the usage of the difficult column set $\tilde{A}_l$ and get a holistic view of the ability of the current annotator on the whole dataset D_t as discussed in Section 2.2.2.

Given difficult columns $\tilde{A}_l$ identified in the l-th iteration, we perform K-means clustering [38, 39] to actively learn [54] other difficult samples from the data lake D_t. Specifically, we cluster all columns in D_t with K-means clustering, where we set $K = n_t$. For each cluster Q_k, if it contains any difficult column in $\tilde{A}_l$, we mark it as a difficult column cluster. We provide an exemplar demonstration of the process in Figure 5. All difficult clusters constitute what we call weak samples in the target data lake.

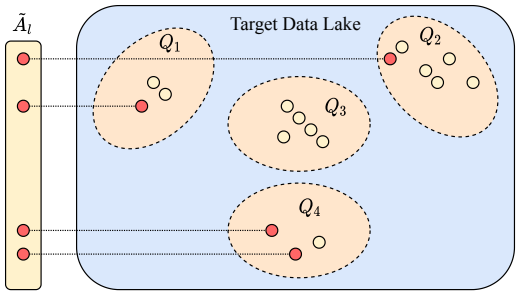


Figure 5: The K-means clustering example for weak sample selection. Q_1, Q_2, and Q_4 are considered as the difficult column clusters in the l-th iteration since they contain columns identified in $\tilde{A}_l$

The motivation for using the K-means clustering to identify the weak samples is as follows: 1) The current intermediate target annotator is not confident regarding its annotations for these ambiguous columns in $\tilde{A}_l$. 2) The current annotator is likely wrong or these columns are very domain-specific, thus they should be used to fine-tune the current annotator. 3) The other columns that share similar output vectors as the difficult columns identified in $\tilde{A}_l$ in the embedding space are also likely to be ambiguous/difficult/domain-specific. Therefore, by clustering the columns in the target data lake and identifying similar columns based on the difficult columns, we expect to identify weak samples in the target data lake that are difficult for the current intermediate target annotator $\bar{M}_{t,l-1}$. The reason why we set $K = n_t$ is that ideally the target data lake should contain n_t clusters corresponding to the types in S_t. The value n_t becomes the natural and intuitive choice of K especially when we have scarce information regarding the distribution of the column labels in the target data lake.

3.3 Gap-hopping Fine-tuning

With the weak samples selected from the target data lake, we now introduce how to use these weak samples to perform gap-hopping fine-tuning. We randomly sample a subset $D_{f,l}$ of columns from the weak sample set and send them to get annotated through manual/heuristic/KB-based annotations[1] [26, 36]. We then conduct fine-tuning in the current iteration.

In view of the catastrophic forgetting challenge discussed in Section 2.2.3, we design the gap-hopping fine-tuning process based on the rehearsal incremental training practice mentioned by [24, 49, 52, 53]. Specifically, as shown in Figure 6, we denote the batch of training samples in the l-th iteration l as $D_{f,l}$ and the initial warm-up training samples as $D_{f,0}$. Then at the l-th iteration, we fine-tune the annotator $\bar{M}_{t,l-1}$ with the collection of samples: $\{D_{f,0}, D_{f,1}, ..., D_{f,l}\}$ with N_f epochs such that the annotator converges on the training collection in the current iteration. As a result, the intermediate target annotator can preserve the useful knowledge obtained from previous iterations and obtain new knowledge with the weak samples identified at the current iteration.

We further notice that the improvement of the intermediate target annotator is more significant at the early stage of incremental

[1]In our experiment, we directly use the ground truth labels provided by each dataset. In reality, the annotations can be semi-automated with the help of heuristics and KB supports.

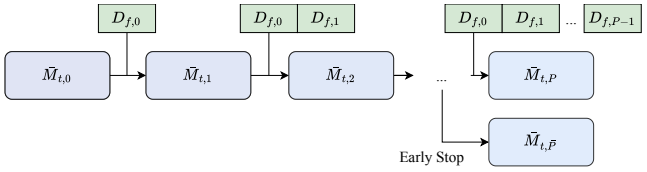


Figure 6: The incremental gap-hopping fine-tuning with or without early stop.

gap-hopping fine-tuning. Intuitively, this coincides with the observation of the poor performance of the LLMs under long-tail samples and domains [56]. Since the annotator adapts to the domain-specific target data lake gradually, the guidance and improvement that the intermediate target annotator can receive from the general knowledge of the LLM are expected to gradually decrease. In other words, the intermediate target annotator is likely to gradually possess all the general knowledge of LLM over the target data lake as the interactions repeat. Given this, we design an early stop mechanism in the iterations as shown in Figure 6. Specifically, when the validation loss of the annotator does not decrease for over N_e iterations, we stop the iteration process, obtain the current annotator $\bar{M}_{t,\bar{P}}$, and reserve the remaining training budget (if any) to randomly sample un-used training columns from the target data lake to complete the fine-tuning process. The early stop mechanism can also be activated if the user observes that the intermediate annotator achieves satisfactory performance on the target data lake or the training data budget available is used up.

3.4 Analysis

We present the pseudocode for LakeHopper in Algorithm 1. Compared with the original PLM, the running time overhead lies in the label difference adjustment (lines 1-5), the query and response time with the LLMs (lines 11-18), the K-means clustering step (line 20), and the additional fine-tuning cost brought by the incremental fine-tuning (lines 10, 23, 31). We will leave the query and response time analysis with the LLMs in Section 5.1. For the adjustment of the label difference, we build a source to target type mapping which induces a running time cost of $O(n_s * n_t)$, with the dictionary mapping, the adjustment of the annotator output layer is $O(n_t)$, where we query the dictionary and make adjustment for each row of L_t. Overall the running time complexity is $O(n_s * n_t)$. The K-means clustering step takes $O(|D_t| * n_t * m * I * P)$ [38, 39], where m is the output embedding dimension (768 for BERT), I is the iteration round of K-means, which is constant. We denote the fine-tuning time cost of a single column as F_1, and the additional fine-tuning cost induced in training is $O((P*|D_{f,0}|+P*(P-1)*|D_{f,l}|/2+(N_t-(P-1)*|D_{f,l}|-|D_{f,0}|))*F_1*N_f) = O((P^2*N_D+N_t)*F_1*N_f)$, where $N_D = \max\{|D_{f,0}|, |D_{f,l}|\}$, $l > 0$. If we constrain the value of P and N_D, such that $P*N_D \leq N_t$, we have the overall time complexity of $O(N_f * P * N_t * F_1)$, which is P times the original fine-tuning time cost of $O(N_f * N_t * F_1)$ for the annotator. Another additional running time cost comes from the validation step performed at each epoch. We denote the validation time cost of a single column as F_2, the size of the validation set as N_v, and the additional running time cost induced in validation is $O(P * N_f * F_2 * N_v)$, which is also P times of the original validation time cost $O(N_f * F_2 * N_v)$. We

Algorithm 1 LakeHopper

Input:
Number of iterations P
Target data lake D_t
Number of fine-tuning epochs in each iteration N_f
Number of early stop iteration threshold N_e
Number of training samples allowed N_t
The confidence threshold δ
The input source annotator M_s
The source label sets S_s

Output:
The adapted target annotator M_t

Copy the weights of M_s except the output layer L_s to the intermediate target annotator $\bar{M}_{t,0}$.
for $j = 1, 2, ..., n_t$ **do**
  If $s_{t,j} \in S_s$ and $s_{t,j} = s_{s,i}$ **then**, assign $L_{s,i}$ to $L_{t,j}$.
  Otherwise, randomly initialize $L_{t,j}$.
end for
Randomly sample a subset $D_{f,0}$ samples from D_t to warm-up $\bar{M}_{t,0}$, update the training budget $N_t = N_t - |D_{f,0}|$.
Initialize the current best validation loss $\bar{v} = \infty$, no improving iterations $\alpha = 0$.
for $l = 1, 2, ..., P$ **do**
  Initialize $A_l = \emptyset$ and $\tilde{A}_l = \emptyset$
  Randomly sample a subset D_l of columns that have not yet been sampled from D_t.
  for each column C in D_l **do**
    Obtain the output embedding v with $\bar{M}_{t,l-1}(C)$.
    If $\Phi(v) < \delta$ **then**, append C to A_l.
  end for
  for each column C in A_l **do**
    Query the LLM with C, obtain the decision d_x.
    If d_x = 'No' or 'I don't know' **then**, add C to $\tilde{A}_l$.
  end for
  Initialize the weak sample set $D_w = \emptyset$.
  Perform K-means clustering, obtain clusters Q_k's.
  Include Q_k to D_w if it contains any column in $\tilde{A}_l$.
  Randomly sample a subset $D_{f,l}$ of columns from D_w, update training budget $N_t = N_t - |D_{f,l}|$.
  Fine-tune $\bar{M}_{t,l-1}$ with $\{D_{f,0}, D_{f,1}, ..., D_{f,l}\}$ for N_f epochs to obtain $\bar{M}_{t,l}$.
  Compute the validation loss $\bar{v}_l$ of $\bar{M}_{t,l}$.
  If $\bar{v}_l < \bar{v}$ **then**, assign $\bar{v} = \bar{v}_l$, $\alpha = 0$, $M_t = \bar{M}_{t,l}$
  Otherwise, assign $\alpha = \alpha + 1$
  if $\alpha \geq N_e$ **then**
    break
  end if
end for
Fine-tune M_t by randomly sample N_t columns from D_t.
return M_t.

believe the running time complexity of LakeHopper is acceptable in real-world applications.

4 EXPERIMENTS

4.1 Experimental Designs

4.1.1 Metrics. We conducted evaluation using F1 scores as evaluation metrics ($F1 = 2 \times \frac{precision \times recall}{precision + recall}$). To address the imbalanced distribution of semantic types, as suggested by [57, 68], we employed two distinct F1 scores: Support-weighted F1 (SW F1) and macro average F1 (MA F1). The Support-weighted F1 score is a weighted average of per-type F1 scores, with weights based on each type's support. Meanwhile, the macro average F1 score computes the mean of all per-type F1 scores, with a focus on long-tail types.

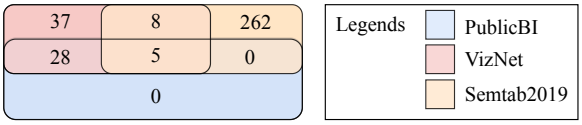


Figure 7: Type set overlapping among three datasets.

4.1.2 Datasets. To evaluate the performance of LakeHopper, we considered three different real-world datasets as data lakes: PublicBI [5, 61], VizNet [10, 68], and Semtab2019 [6, 30] datasets, which were used by previous works [36, 57, 68]. Specifically, we selected the multi-table-only subset of the WebTables corpus from the VizNet dataset following the settings of [57, 68]. As for the Semtab2019 dataset, we selected the same subset in alignment with RECA [57]. We present the detailed statistics of the three datasets in Table 3. The PublicBI dataset contains the fewest number of tables and the tables in PublicBI are much wider than those in the other two datasets. The VizNet dataset is the largest, while the tables in the VizNet dataset tend to be narrow. All of the three datasets are annotated with the DBpedia ontology yet covering different aspects and granularity levels. We believe the selection of these three datasets can represent the nature of tables in real-world applications: different data lakes contain tables with different sizes, content aspects, and annotations with different granularity levels.

Based on the three datasets, we designed two sets of cross data lake model adaptation experiments: PublicBI to VizNet and VizNet to Semtab2019. As shown in Figure 7, the type set of the PublicBI dataset is a subset of that of the VizNet dataset. The overlapping between the type sets of VizNet and Semtab2019 is 13, which accounts for 16.7% of the types of VizNet and 4.7% that of Semtab2019. The PublicBI to VizNet transfer represents an easier model adaptation case, where all the types from the source data lake are preserved, and the annotator only needs to extend its knowledge with the additional types that occur in the target data lake. The VizNet to Semtab2019 transfer represents a more significant domain knowledge shift, where the annotator needs to forget some types from the source data lake while learning the new types from the target data lake. We directly used the ground truth labels provided by the datasets as a replacement for the annotation step stated in our pipeline as shown in Figure 2. Since the focus of the paper is on developing low-resource domain adaptation methods for PLM-based CTA models, instead of innovating on the training data labeling process. As a reference of human annotation speed, we sampled 100 columns from VizNet and Semtab2019 datasets and recruited 10 volunteers to perform manual annotation. The averaged manual annotation times are 21.2 and 27.1 seconds per column. We believe that by conducting the above-mentioned cross data lake model adaptation experiments, we can evaluate the effectiveness of LakeHopper under different application scenarios with different difficulty levels.

Table 3: Stats of PublicBI, VizNet, and Semtab2019 datasets

Datasets	PublicBI	VizNet	Semtab2019
# types	33	78	275
# tables	160	32262	3045
# annotated columns	1424	74141	7603
Avg. # columns	64.6	2.3	4.5

4.1.3 Baselines. We selected the following baselines to compare and evaluate the performance of LakeHopper:

- Sherlock [27]: Sherlock leverages a fusion strategy that combines features from various levels of granularity, encompassing characters, words, paragraphs, and global context, in order to generate table representations.
- TABBIE [28]: TABBIE employs a dual-transformer architecture for encoding both columns and rows. Subsequently, the resulting embeddings of the target column are utilized for annotating the column's semantic types.
- DODUO [55]: DODUO utilizes a transformer-based framework for the joint encoding of all table columns, enabling seamless integration of intra-table context.
- Sudowoodo [63]: Sudowoodo uses contrastive learning to capture the inter-table context information to improve the model performance under low-resource settings.
- RECA [57]: RECA introduces a novel named entity schema to discover tables that are related in structures and jointly encode the inter-table context and the original table to enhance the annotation.

Among these approaches, DODUO, Sudowoodo, and RECA are the state-of-the-art approaches. DODUO utilizes the inner-table context, while Sudowoodo and RECA aim to improve annotation performance by introducing contrastive learning and capturing the inter-table context respectively. Both DODUO and RECA claim to be learning efficient (i.e., require a small amount of training data to achieve good performance) [55, 57]. Sudowoodo is tailored for low-resource settings due to its nature of contrastive learning. We believe the three methods are representative of the model adaptation ability of existing PLM-based CTA approaches.

4.1.4 Our Approaches. In order to evaluate the performance of LakeHopper, we performed model adaptation on the three state-of-the-art CTA approaches: DODUO, Sudowoodo, and RECA to see if LakeHopper can effectively improve their annotation performance on the new, unseen data lakes. We denote the LakeHopper based on DODUO, Sudowoodo, and RECA as LakeHopper(D), LakeHopper(S), and LakeHopper(R) respectively. We record the average relative performance gains of the three LakeHoppers over DODUO, Sudowoodo, and RECA under low-resource settings and mark them as Avg. Gain in Tables 4 and 5. We further included the ablation variants of the three types of LakeHoppers in our experiments, marked as -LLM. The design of the ablation variants is that we replace the weak samples discovered through the LLM interactions with the same amount of randomly selected samples from the training set. By comparing the performance of LakeHoppers with their -LLM variants, we can evaluate the effect brought by LLM interactions.

4.1.5 Plans. To comprehensively evaluate the effectiveness of LakeHopper in improving existing state-of-the-art CTA models under different application scenarios, we divided our evaluation into low-resource settings, high-resource settings, and ablation study. As discussed by [25], the definition of low-resource is task-specific and there lack of a universal hard threshold for all the tasks. In our paper, for ease of demonstration and understanding, we refer to the experiments with less than 10% of training data as low-resource, while referring to those with more than 10% of training data as high-resource. Specifically, we experimented with 25%, 50%, and 100% as high-resource settings on both data lake transfers. For experiments conducted under low-resource settings, we designed them based on the number of iterations performed using LakeHopper. We experimented with the same amount of training data as the labeled samples used by LakeHopper for running 5, 10, 20, and 30 iterations, which accounts for 1.6%, 2.5%, 4.2%, and 5.9% for the PublicBI to VizNet transfer and 2.4%, 3.8%, 6.5%, and 9.3% for the VizNet to Semtab2019 transfer. Indeed, the low-resource experiments are more representative of real-world applications, where typically the annotation budget for adapting an annotator to a new data lake is limited. To understand the effect of each component of LakeHopper, we conducted an ablation study.

4.1.6 Settings. We randomly selected 10% of the total labeled data to form the test set of the PublicBI dataset and the rest of the data was used as training data. For the VizNet dataset, we selected three out of five folds from the WebTables corpus and used one fold for training, one for validation, and the other one for testing. For the Semtab2019 dataset, we followed the practice of RECA [57], where we randomly sampled 10% of the annotated columns to form the test set, and split the rest data with a ratio of 1:4 to form the validation and training set. All the models were trained to converge on the source data lake and then re-trained on the target data lake. We used Adam as the optimizer and selected the learning rates from the set $\{0.00001, 0.00002, 0.00005\}$. Since the problem of CTA is in a multi-type classification manner, we adopted the cross-entropy loss as the loss function. We set the number of iterations $P = 50$, with an early stop threshold of $N_e = 5$ and the number of fine-tuning epochs in each iteration $N_f = 5$. The size of the warm-up sets for the PublicBI to VizNet and VizNet to Semtab2019 data lake transfers were set to 50 and 25 tables respectively in consideration of the size of the target data lakes. The size of the fine-tuning subsets $D_{f,l}$, $l > 0$ were set to 25 and 15 columns for the two data lake transfers. We set the batch size as 8. The number of early stop rounds for LLM transfer iterations was set to 5. The confidence level threshold δ was set to 0.9. We set the maximum BERT sequence length as 128. We followed official implementations provided by the baseline approaches and preserved their experimental settings as much as we could [7, 9, 11–13]. For Sherlock [27], the low-resource evaluation with 2.4% and 3.8% training data on the VizNet to Semtab2019 data lake transfer cannot be completed due to the reason that the official implementation of Sherlock [7] requires each semantic type to be present in the training set for at least once in order to compile the model. When the training ratios are 2.4% and 3.8%, the numbers of training samples are less than the size of the semantic type set, as a result, the Sherlock model cannot be compiled successfully under these two settings. We accessed the gpt-3.5-turbo-4k model through

Table 4: Low-resource experimental results on the PublicBI to VizNet data lake transfer.

	low1 1.6% (239 col)		low2 2.5% (364 col)		low3 4.2% (614 col)		low4 5.9% (864 col)		Avg. Gain	
	SW F1	MA F1	SW F1	MA F1	SW F1	MA F1	SW F1	MA F1	SW F1	MA F1
Sherlock [27]	0.344	0.130	0.470	0.238	0.558	0.303	0.591	0.345	-	-
TABBIE [28]	0.505	0.204	0.565	0.268	0.637	0.278	0.709	0.315	-	-
DODUO [55]	0.499	0.190	0.569	0.254	0.644	0.280	0.742	0.416	-	-
Sudowoodo [63]	0.561	0.213	0.601	0.277	0.705	0.374	0.724	0.427	-	-
RECA [57]	0.587	0.206	0.610	0.216	0.716	0.303	0.749	0.312	-	-
LakeHopper(D)	0.612	0.323	0.664	0.343	0.746	0.425	0.783	0.486	15.2% ↑	43.4% ↑
LakeHopper(S)	0.609	0.317	0.679	0.384	**0.776**	0.446	**0.814**	**0.558**	11.0% ↑	34.3% ↑
LakeHopper(R)	**0.621**	**0.331**	**0.705**	**0.412**	0.749	**0.506**	0.793	0.522	8.0% ↑	71.4% ↑

Table 5: Low-resource experimental results on the VizNet to Semtab2019 data lake transfer.

	low1 2.4% (131 col)		low2 3.8% (206 col)		low3 6.5% (356 col)		low4 9.3% (506 col)		Avg. Gain	
	SW F1	MA F1	SW F1	MA F1	SW F1	MA F1	SW F1	MA F1	SW F1	MA F1
Sherlock [27]	-	-	-	-	0.225	0.096	0.313	0.154	-	-
TABBIE [28]	0.322	0.097	0.375	0.127	0.494	0.210	0.580	0.287	-	-
DODUO [55]	0.302	0.110	0.393	0.164	0.521	0.245	0.594	0.303	-	-
Sudowoodo [63]	0.343	0.134	0.452	0.219	0.535	0.278	0.576	0.299	-	-
RECA [57]	0.340	0.100	0.468	0.169	0.578	0.243	0.624	0.304	-	-
LakeHopper(D)	0.365	0.144	0.475	0.215	0.573	0.296	0.620	0.370	14.0% ↑	26.2% ↑
LakeHopper(S)	0.377	**0.160**	**0.514**	**0.259**	0.579	0.320	0.619	0.357	9.8% ↑	18.0% ↑
LakeHopper(R)	**0.424**	**0.160**	0.503	0.248	**0.633**	**0.394**	**0.672**	**0.429**	12.3% ↑	52.5% ↑

Azure OpenAI APIs version 2023-05-15 and OpenAI official APIs. All the experiments were conducted on Intel(R) Xeon(R) Gold 6240 CPU @ 2.60GHz CPUs and four NVIDIA A800 80GB PCIe GPUs.

4.2 Main Experimental Results

4.2.1 Low-resource Settings. We present the low-resource experiments results on the PublicBI to VizNet and VizNet to Semtab2019 data lake transfers in Tables 4 and 5 respectively. We first notice that LakeHopper can significantly improve the performance of the state-of-the-art CTA models when adapting to new data lakes. Specifically, on the low-resource PublicBI to VizNet data lake transfer (Table 4), LakeHopper on average boosts the performance of directly re-training the three state-of-the-art models by relatively 15.2%, 11.0%, and 8.0% for the SW F1s, and 43.4%, 34.3%, and 71.4% for the MA F1s. Similarly, on the low-resource VizNet to Semtab2019 data lake transfer (Table 5), we observe average relative performance gains of 14.0%, 9.8%, 12.3%, 26.2%, 18.0%, and 52.5% of the three models on two metrics. We attribute these phenomena to the fact that LakeHopper identifies the knowledge gap between the source and the target annotators and selects the weak samples that can significantly improve the model adaptation performance to the target data lake. We further observe that the performance uplifts of MA F1s are much larger than those of SW F1s, which implies that LakeHopper greatly improves the annotation accuracy of the state-of-the-art CTA models over long-tail types. This phenomenon can be explained by the mechanism of the knowledge gap identification and the weak sample selection steps. Initially, the CTA models are likely to perform extremely poorly on the long-tail types. The knowledge gap identification step is more likely to identify these long-tail types in comparison with randomly selecting the training samples. The weak samples selection step then identifies more long-tail weak samples that can greatly improve the annotation performance of the models over long-tail types. As a result, the CTA annotators achieve great performance uplift on the MA F1 scores. Besides, we notice that across two different data lake transfers and multiple low-resource settings, LakeHopper(R) and LakeHopper(S) achieve the best performance, which because of that their core models RECA and Sudowoodo utilize inter-table context information, containing more information of the holistic data lake, which is beneficial when the training samples are scarce. LakeHoppers on the two models further boost their performance under the low-resource settings, which means LakeHoppers requires less labeled training data to adapt CTA annotators to new data lakes compared with the baselines.

4.2.2 High-resource Settings. We present the high-resource experiments on the two data lake transfers in Table 6. We first observe that all three LakeHoppers still achieve performance uplift over their core models on the two data lake transfers provided different ratios of training data, even though LakeHopper is tailored for low-resource model adaptation. This implies that the performance uplifts brought by LakeHopper at the beginning stage of fine-tuning can be retained to the later stage after the iterations terminate. When comparing the performance of the three LakeHoppers, we observe that LakeHopper(D) achieves the highest F1 scores for the most number of high-resource experiments on the PublicBI to VizNet data lake transfer, while LakeHopper(D) and LakeHopper(R) perform better than LakeHopper(S) on the VizNet

Table 6: High-resource experimental results on the PublicBI to VizNet and VizNet to Semtab2019 data lake transfers.

	PublicBI to VizNet						VizNet to Semtab2019					
	25% (3745 col)		50% (7490 col)		100% (14980 col)		25% (1363 col)		50% (2725 col)		100% (5450 col)	
	SW F1	MA F1	SW F1	MA F1	SW F1	MA F1	SW F1	MA F1	SW F1	MA F1	SW F1	MA F1
Sherlock [27]	0.714	0.465	0.759	0.482	0.791	0.592	0.494	0.292	0.567	0.341	0.637	0.424
TABBIE [28]	0.825	0.470	0.849	0.501	0.862	0.562	0.692	0.446	0.712	0.455	0.765	0.568
DODUO [55]	0.863	0.649	0.872	0.708	0.911	0.734	0.736	0.491	0.778	0.575	0.808	0.608
Sudowoodo [63]	0.805	0.559	0.842	0.603	0.862	0.682	0.701	0.426	0.729	0.500	0.763	0.544
RECA [57]	0.859	0.627	0.860	0.671	0.878	0.679	0.744	0.472	0.797	0.576	0.818	0.613
LakeHopper(D)	0.874	0.661	**0.902**	**0.749**	**0.927**	**0.791**	**0.753**	0.538	**0.798**	0.612	**0.819**	0.622
LakeHopper(S)	**0.875**	**0.700**	0.884	0.707	0.923	0.789	0.713	0.475	0.733	0.502	0.768	0.580
LakeHopper(R)	0.860	0.658	0.871	0.700	0.880	0.705	0.748	**0.545**	0.797	**0.618**	**0.819**	**0.643**

to Semtab2019 data lake transfer. We attribute this phenomenon to the performance of their core models. On the PublicBI to VizNet data lake transfer, DODUO is the best core in comparison with Sudowoodo and RECA, while on the VizNet to Semtab2019 data lake transfer, RECA and DODUO outperform Sudowoodo significantly. The performance of the core models influences their corresponding LakeHopper results.

4.2.3 Ablation Study. We present the ablation study results in Figure 8. Specifically, we considered replacing the LLM core as GPT-4o (LakeHopper+GPT-4o) and removing the LLM interactions (LakeHopper-LLM). We reported the averaged F1 scores of LakeHopper (D), (S), and (R) on all experimental settings. We observe that using GPT-4o instead of GPT-3.5 as the LLM core boosts the performance of LakeHoppers. While removing the LLM interactions results in performance drops (comparing LakeHopper-LLM against LakeHopper+GPT-3.5). We notice that the performance changes in high-resource settings are significantly lower than those in low-resource settings, which implies that the LLM interactions are especially useful for selecting helpful training samples for improving annotation performance under low-resource settings. The results demonstrate the benefit of introducing LLM interactions in selecting weak samples for better gap-hopping fine-tuning of cross data lake model adaptation.

4.3 Comparison with LLMs

To provide a comprehensive evaluation of the performance of LakeHopper, we present the experimental results of LLMs in this section. Specifically, we evaluate the out-of-box non-fine-tuned performance of ChatGPT [34], GPT-4o [15] and a pre-trained state-of-the-art table generalist LLM: TableLlama [70] in Section 4.3.1 and the domain-specific fine-tuned performance of TableLlama in Section 4.3.2.

4.3.1 Non-fine-tuned Performance. To experiment with ChatGPT on CTA, we followed the template designs discussed in a recent work [34]. Besides, we followed the template provided by TableLlama [70] to evaluate the non-fine-tuned performance of the pre-trained TableLlama model. As shown in the VizNet* and Semtab2019* parts of Table 7, the non-fine-tuned LLM-based approaches perform poorly with less than 50% F1 scores. Besides,

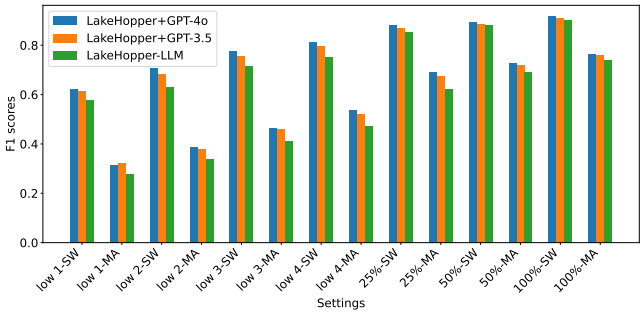
Table 7: Experimental results of LLMs.

Datasets	Methods	SW F1	MA F1	OOD rate
VizNet*	ChatGPT [34]	0.451	0.324	0.264
	GPT-4o [15]	0.584	0.390	0.280
	TableLlama [70]	0.146	0.086	0.177
VizNet+	GPT-4o+Wiki	0.607	0.386	0.057
VizNet++	TableLlama-low1	0.620	0.349	0.027
	TableLlama-low2	0.694	0.395	0.029
	TableLlama-low3	0.749	0.482	0.025
	TableLlama-low4	0.793	0.507	0.027
	TableLlama-25%	0.882	0.591	0.019
	TableLlama-50%	0.903	0.608	0.018
	TableLlama-100%	0.925	0.642	0.017
Semtab2019*	ChatGPT [34]	0.318	0.269	0.078
	GPT-4o [15]	0.499	0.338	0.041
	TableLlama [70]	0.113	0.078	0.476
Semtab2019+	GPT-4o+Wiki	0.549	0.335	0.073
Semtab2019++	TableLlama-low1	0.552	0.386	0.021
	TableLlama-low2	0.541	0.412	0.018
	TableLlama-low3	0.590	0.440	0.014
	TableLlama-low4	0.645	0.467	0.021
	TableLlama-25%	0.755	0.570	0.020
	TableLlama-50%	0.802	0.647	0.009
	TableLlama-100%	0.846	0.702	0.009

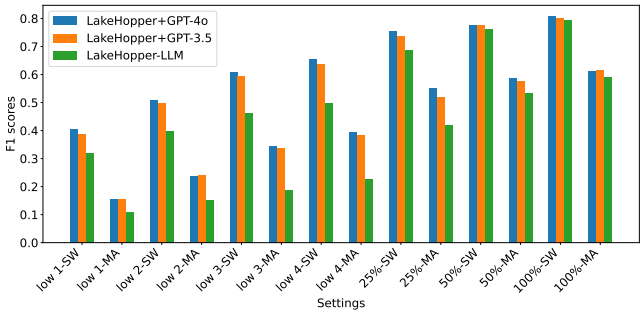
* and ++ are without and with fine-tuning.
+ is the RAG-based solution.

when considering the out-of-domain hallucination of these approaches (the models provide annotations out of the semantic type set S required), they have a high out-of-domain rate: 26.4% and 17.7% for ChatGPT and TableLlama on VizNet dataset; 7.8% and 47.6% for the two models on Semtab2019 dataset. The OOD problem needs to be addressed before we can confidently apply ChatGPT to real-world applications, otherwise, users are likely to receive annotations that seem to be plausible yet do not fit their application needs. In summary, we believe that the non-fine-tuned LLM approaches are not a perfect choice for cross data lakes CTA.

4.3.2 Fine-tuned Performance. Following the official settings (LoRA and flash attention are used) of TableLlama [14, 70], we



(a) PublicBI to VizNet



(b) VizNet to Semtab2019

Figure 8: The averaged F1 scores of LakeHoppers on the PublicBI to VizNet and VizNet to Semtab2019 data lake transfers.

fine-tuned the model with the same amount of training data used in our main low-resource and high-resource experiments to understand the performance of LLMs. Specifically, in VizNet+ and Semtab2019+ parts of Table 7, we show TableLlama's fine-tuned performance under four low-resource and three high-resource settings. We observe that fine-tuned TableLlama achieves comparable F1 scores as the LakeHoppers (shown in Tables 4, 5, and 6). We further compare the fine-tuning speed of TableLlama, LakeHopper(R), Lakehopper(S), and LakeHopper(D), which are 1.43, 39.87, 85.03, and 188.15 columns per second. The fine-tuning speed of TableLlama is significantly slower than the three LakeHoppers. Specifically, the three LakeHoppers are 27.9×, 59.5×, and 131.6× faster than TableLlama during fine-tuning. In summary, although the annotation quality of LakeHopper and fine-tuned TableLlama is similar, LakeHopper provides superior performance over fine-tuned TableLlama in terms of domain adaptation efficiency.

4.3.3 RAG-based Performance. To further explore the limit of LLM-based methods, we develop a solution of applying Retrieval Augmented Generation (RAG) techniques in CTA. We followed the designs of [16, 69], which utilize Wikipedia as the external knowledge source. Specifically, for each entity in the column, we link the entity with a Wikipedia entity by using the *search()* function of the Wikipedia API [1]. Then we retrieve each entity's corresponding Wikipedia page content, send the question with the filtered content to GPT-4o, and ask for its annotations. The experimental results are presented in Table 7. We observe that despite applying RAG on GPT-4o can improve the performance of the model, the overall performance is comparable to TableLlama-low1, which is still relatively poor in comparison with fine-tuning a PLM on the target data lake. Therefore, we believe that applying RAG on LLMs can still not perform well compared to fine-tuning a PLM/LLM on the target data lake.

5 DISCUSSION

5.1 Time and Monetary Overhead

We discuss the time and monetary overhead induced by LakeHopper. In Section 3.4 we analyzed that the training and validation times overhead of the incremental fine-tuning process are constant times that of the original LM cores. In this section, we focus on the time and monetary overhead induced by other components of LakeHopper. In Table 8, we present the total number of queries

Table 8: The number of queries and iterations

Data lake transfers	Methods	# queries	# iter.
PublicBI to VizNet	LakeHopper(D)	1348	38
	LakeHopper(S)	1405	45
	LakeHopper(R)	1550	50
VizNet to Semtab2019	LakeHopper(D)	1506	50
	LakeHopper(S)	1720	50
	LakeHopper(R)	1185	38

to the ChatGPT model and the number of training iterations used (due to the early stop, some methods terminate earlier). We further present the average query-response time and monetary costs for the VizNet and Semtab2019 datasets in Table 9. We observe that the monetary cost of the query-response pair in the Semtab2019 dataset is three times larger than that in the VizNet dataset. We attribute this phenomenon to the fact that the label type set of the Semtab2019 dataset is much larger than that of the VizNet dataset. Listing all the types requires more tokens in Semtab2019, thus inducing a higher monetary cost. For the PublicBI to VizNet data lake transfer, LakeHopper(D), LakeHopper(S), and Lakehopper(R) have time overhead of 1704.3s, 1776.3s, and 1960.0s with monetary overhead of $0.64, $0.66, and $0.73. Similarly, the time and money overheads of the three approaches on the VizNet to Semtab2019 data lake transfer are 1511.6s, $2.26; 1726.4s, $2.58 and 1189.4s, $1.78. As for the K-means clustering time overhead, we record the average time costs of performing K-means clustering on the PublicBI to VizNet and VizNet to Semtab2019 data lake transfers as 1.02s and 1.38s per iteration respectively. Thus the time overheads induced by K-means clustering are less than 70s for all the approaches on both data lake transfers. We conclude that the additional time overhead and monetary costs induced by LakeHoppers are low and acceptable for real-world applications. We understand that in some use cases, the scale of the fine-tuning data could be very large, which leads to high time and monetary costs. In that case, we suggest deploying open-sourced LLMs such as Llama-3 models [19] and Mixtral-8*22B [29] and utilizing techniques such as quantization and knowledge distillation [71] to improve the inference efficiency of the LLMs.

Table 9: Monetary and time costs per query-response.

Datasets	monetary cost [$]	time cost [s]
VizNet	0.00047	1.2643
Semtab2019	0.00150	1.0037

Table 10: Comparison between ChatGPT direct answer and verification.

	ChatGPT direct answer	ChatGPT verification
VizNet	0.424	0.807
Semtab2019	0.339	0.870

5.2 Reliability of LLM Verification

In section 3.1.2, we claimed that the LLM query verification template reduces the difficulty level compared to the template that asks the LLM to directly select the most appropriate type from a type set. We include an additional experiment to verify the claim. As shown in Table 10, if we directly prompt the ChatGPT to answer the CTA question, the answer accuracies are 0.424 and 0.339 on the two datasets, while if we ask them to verify whether an annotation is correct/wrong, the answer accuracies become 0.807 and 0.870 on the same set of questions of the two datasets. We therefore conclude that our query verification template is useful for reducing the difficulty level for LLM to perform cross data lake guidance for the PLM-based annotators.

5.3 Effect of Label Set Difference Adjustment

We discuss the initial model knowledge by analyzing the effect of applying label set different adjustments or not. As shown in Table 11, we use P to V to represent the PublicBI to VizNet data lake transfer and V to S to represent the VizNet to Semtab2019 data lake transfer. The **no-label** setting refers to the case where we do not apply label set difference adjustment and randomly initialize the weights at the output layer. The **label** is the setting where we follow the steps to adjust the label set difference as discussed in Section 3.1.1. We notice that the label set difference adjustment step indeed boosts the performance of all three LakeHoppers on two different data lake transfers to a large extent, which validates our intuition in Section 3.1.1 adjusting the weights of the output layer of the target annotator based on the overlapping types with the source annotator can partially inherit the annotation ability from the source to the target annotator.

5.4 Parameter Sensitivity

We experimented on the hyperparameter K's sensitivity in the K-means clustering and reported the results in Figure 9. Specifically, we record the averaged F1 scores of LakeHopper (D), (S), and (R) on all low-resource settings. The LLM core used in the LakeHoppers is GPT-4o. We set the value of K as 1, $0.5n_t$, n_t, $1.5n_t$, $2n_t$. We observe that the performance of LakeHoppers reaches the peak when $K = n_t$, which is reasonable, since intuitively, there exists n_t different types of columns in the target data lake, setting the K as n_t can help the pipeline identify the weak samples from n_t distinct clusters effectively.

Table 11: The effect of Label Set Difference Adjustment

		no-label		label	
	Methods	SW F1	MA F1	SW F1	MA F1
P to V	LakeHopper(D)	0.028	0.004	0.241	0.075
	LakeHopper(S)	0.000	0.000	0.231	0.078
	LakeHopper(R)	0.001	0.000	0.010	0.007
V to S	LakeHopper(D)	0.001	0.000	0.091	0.016
	LakeHopper(S)	0.001	0.000	0.094	0.014
	LakeHopper(R)	0.000	0.000	0.090	0.013

5.5 Influencing Factors of Domain Adaptation

Two important factors influence the amount of necessary training data required: the language model used and the difficulty of the data domain transfer. For example, as shown in Table 5, to obtain an annotator with over 0.6 support weighted F1 scores, we need to provide 506 columns as training data for LakeHopper(D) and LakeHopper(S), while only 356 columns are needed for LakeHopper(R). We believe the faster adaptation of the LakeHopper(R) in comparison with LakeHopper(D) and LakeHopper(S) can be attributed to the fact that the different core PLM used. Moreover, we believe the difficulty of the data domain transfer also influences the amount of required training data. When comparing the results in Tables 4 and 5, we observe that the LakeHoppers adapt faster on the PublicBI to VizNet data lake transfer. We believe the reason lies in the fact that the VizNet to Semtab2019 transfer represents a more significant domain knowledge shift as discussed in Section 4.1.2 and thus is more difficult for the annotator to adapt.

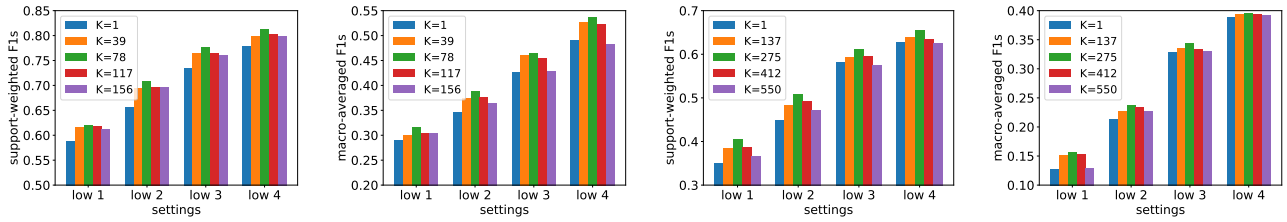
6 RELATED WORK

We classify the previous CTA approaches into three types.

Feature engineering: The feature engineering approaches aim at extracting manually crafted, statistical, and semantic features from tables. In the early stages, considerable emphasis was placed on extracting data semantics through manual feature engineering. Notably, the SemanticTyper framework [48] was introduced to categorize tabular data into textual and numeric types. Afterward, Sherlock [27] employs feature extraction techniques at multiple levels of granularity. Building upon Sherlock [27], Sato [68] incorporates table topic and inner-table features, which enhance the performance.

These approaches have a limitation in failing to capture the intricate semantics of tables and generate expressive representations. These methods struggle to effectively capture nuanced table semantics due to limitations imposed by their shallow network structures [57, 62].

PLM-based: PLMs such as BERT [18] have been introduced to generate table representations with high expressiveness. Specifically, TaBERT [66], a table pre-training method, utilizes the key table rows and the user queries to learn the expressive table representations. TABBIE [28] improves based on TaBERT by using two independent



(a) P to V, SW F1 (b) P to V, MA F1 (c) V to S, SW F1 (d) V to S, MA F1

Figure 9: The parameter K's sensitivity of LakeHopper on two data lake transfers.

transformers to jointly encode table rows and columns, providing a more comprehensive understanding of table structures. On a similar note, DODUO [55] leverages a transformer architecture to encode all columns within tables simultaneously, harnessing the benefits of capturing inner-table semantics. Sudowoodo [63] builds upon the design of DODUO by introducing an additional contrastive learning phase to inject robustness and diversity into the table representations. RECA [57] adds an additional inter-table relationship discovery stage to mine the inter-table context for each table. By leveraging the insights derived from the relationships between multiple related tables, RECA demonstrates enhanced annotation performance.

Despite the state-of-the-art performance achieved by these approaches through domain-specific fine-tuning, most of them achieve significantly poor performance when adapted to new data lakes where the labeled training data is scarce (i.e., low resource settings). Although Sudowoodo and RECA utilize contrastive learning and inter-table relationships respectively to generate table representations that can slightly mitigate the poor performance problem under low resource settings, their investigation angle is limited to the data perspective: their approaches discover the inherent data content similarity while paying less attention to the model perspective, the improvements of the model fine-tuning process itself. Therefore, we propose LakeHopper to fill the research gap from the model perspective. The design of LakeHopper is so flexible that existing state-of-the-art PLM-based approaches can fit into it to get enhanced especially under low-resource domain adaptation scenarios.

LLM-based: Recent advancements in LLMs such as GPTs [15] and Llamas [59] introduce new research opportunities for employing LLMs to perform CTA in a QA manner. Korini et al. [34] introduce well-designed prompt templates to query ChatGPT [45] to get annotations for the columns. An obvious advantage is that this approach leverages the massive amount of world knowledge contained in the LLMs to perform zero-shot annotation, which results in high generalizability. However, as demonstrated in our experiment, the overall annotation quality of ChatGPT [34] on CTA is poor, and the hallucination rate of the zero-shot performance is relatively high. The relatively poor performance of ChatGPT, especially on specific long-tail domains and tasks, can also be observed in previous studies [56, 58, 65]. To overcome the poor performance of the LLM-based approaches on CTA, several recent works introduce fine-tuning schemes [21, 37, 70]. Although the performance of these methods can approach the state-of-the-art PLM-based methods after applying domain-specific fine-tuning, the additional time and memory cost of fine-tuning these large LMs significantly limits the practical use for the domain adaptation scenario of CTA.

Instead of fine-tuning the large LMs directly, LakeHopper leverages the LLMs as the domain-agnostic guide that helps the lightweight PLM annotators generalize across data lakes. By performing the domain-specific fine-tuning on the lightweight PLM-based annotator, while leveraging the world knowledge of the LLMs, we achieve a CTA pipeline that is both generalizable and accurate.

7 CONCLUSION AND FUTURE WORK

In this paper, we proposed LakeHopper, a novel model adaptation framework for CTA. LakeHopper opens up the opportunity for the effective reuse of PLMs and rapid transformation of the CTA annotator between data lakes, which have not been discussed by previous studies. Extensive experiments on two different data lake transfers demonstrate the effectiveness of LakeHopper in transforming CTA models from the source to target data lakes. Specifically, the average improvements of LakeHopper over the state-of-the-art approaches under low-resource settings are 11.7% and 41.0% for the support-weighted and macro average F1 scores. In the future, we plan to broaden the horizon of LakeHopper by exploring opportunities to incorporate the collaboration of multiple source models.

REFERENCES

[1] 2014. wikipedia-API. Retrieved Oct 3, 2024 from https://pypi.org/project/wikipedia/#description

[2] 2019. Datalib: JavaScript Data Utilities. Retrieved June 14, 2023 from https://vega.github.io/datalib/

[3] 2019. messytables. Retrieved June 14, 2023 from https://pypi.org/project/messytables/

[4] 2019. PowerBI. Retrieved June 14, 2023 from https://powerbi.microsoft.com/en-us/

[5] 2019. PublicBI dataset. Retrieved Oct 14, 2023 from https://github.com/cwida/public_bi_benchmark/tree/master

[6] 2019. Semtab2019 dataset. Retrieved June 14, 2023 from https://zenodo.org/records/3518539

[7] 2019. Sherlock. Retrieved Feb 1, 2023 from https://github.com/mitmedialab/sherlock-project

[8] 2019. Trifacta. Retrieved June 14, 2023 from https://www.alteryx.com/about-us/trifacta-is-now-alteryx-designer-cloud

[9] 2021. TABBIE. Retrieved Feb 1, 2023 from https://github.com/SFIG611/tabbie

[10] 2021. VizNet dataset. Retrieved June 14, 2023 from https://github.com/megagonlabs/sato/tree/master/table_data

[11] 2022. DODUO. Retrieved Feb 1, 2023 from https://github.com/megagonlabs/doduo

[12] 2022. Sudowoodo. Retrieved May 1, 2023 from https://github.com/megagonlabs/sudowoodo

[13] 2023. RECA. Retrieved June 14, 2023 from https://github.com/ysunbp/RECA-paper

[14] 2024. TableLlama. Retrieved Sep 25, 2024 from https://github.com/OSU-NLP-Group/TableLlama

[15] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774* (2023).

[16] D Chen. 2017. Reading Wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051* (2017).

[17] Marco Cremaschi, Roberto Avogadro, and David Chieregato. 2019. MantisTable: an Automatic Approach for the Semantic Table Interpretation. *SemTab@ ISWC* 2019 (2019), 15–24.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. ACL, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[19] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[20] Ju Fan, Jianhong Tu, Guoliang Li, Peng Wang, Xiaoyong Du, Xiaofeng Jia, Song Gao, and Nan Tang. 2024. Unicorn: A Unified Multi-Tasking Matching Model. *ACM SIGMOD Record* 53, 1 (2024), 44–53.

[21] Benjamin Feuer, Yurong Liu, Chinmay Hegde, and Juliana Freire. 2023. ArcheType: A Novel Framework for Open-Source Column Type Annotation using Large Language Models. *arXiv preprint arXiv:2310.18208* (2023).

[22] Robert M French. 1999. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences* 3, 4 (1999), 128–135.

[23] Sainyam Galhotra, Udayan Khurana, Oktie Hassanzadeh, Kavitha Srinivas, Horst Samulowitz, and Miao Qi. 2019. Automated feature enhancement for predictive modeling using external knowledge. In *2019 International Conference on Data Mining Workshops (ICDMW)*. IEEE, 1094–1097.

[24] Alexander Gepperth and Cem Karaoguz. 2016. A bio-inspired incremental learning architecture for applied perceptual problems. *Cognitive Computation* 8, 5 (2016), 924–934.

[25] Michael A Hedderich, Lukas Lange, Heike Adel, Jannik Strötgen, and Dietrich Klakow. 2020. A survey on recent approaches for natural language processing in low-resource scenarios. *arXiv preprint arXiv:2010.12309* (2020).

[26] Madelon Hulsebos, Sneha Gathani, James Gale, Isil Dillig, Paul Groth, and Çağatay Demiralp. 2021. Making table understanding work in practice. *arXiv preprint arXiv:2109.05173* (2021).

[27] Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César Hidalgo. 2019. Sherlock: A deep learning approach to semantic data type detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, New York, NY, USA, 1500–1508. https://doi.org/10.1145/3292500.3330993

[28] Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. TABBIE: Pretrained Representations of Tabular Data. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. ACL, 3446–3456. https://doi.org/10.18653/v1/2021.naacl-main.270

[29] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088* (2024).

[30] Ernesto Jiménez-Ruiz, Oktie Hassanzadeh, Vasilis Efthymiou, Jiaoyan Chen, and Kavitha Srinivas. 2020. Semtab 2019: Resources to benchmark tabular data to knowledge graph matching systems. In *The Semantic Web: 17th International Conference, ESWC 2020, Heraklion, Crete, Greece, May 31–June 4, 2020, Proceedings 17*. Springer, 514–530.

[31] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, New York, NY, USA, 3363–3372. https://doi.org/10.1145/1978942.1979444

[32] Moe Kayali, Anton Lykov, Ilias Fountalis, Nikolaos Vasiloglou, Dan Olteanu, and Dan Suciu. 2024. Chorus: Foundation Models for Unified Data Discovery and Exploration. *Proceedings of the VLDB Endowment* 17, 8 (2024), 2104–2114.

[33] Udayan Khurana and Sainyam Galhotra. 2021. Semantic Concept Annotation for Tabular Data. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management* (Virtual Event, Queensland, Australia) *(CIKM '21)*. Association for Computing Machinery, New York, NY, USA, 844–853. https://doi.org/10.1145/3459637.3482295

[34] Keti Korini and Christian Bizer. 2023. Column Type Annotation using ChatGPT. *Joint Workshops at 49th International Conference on Very Large Data Bases (VLDBW'23) — TaDA'23: Tabular Data Analysis Workshop* 46, 130,471 (2023), 91.

[35] Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. 2021. Towards learned metadata extraction for data lakes. *BTW 2021* (2021).

[36] Sven Langenecker, Christoph Sturm, Christian Schalles Schalles, and Carsten Binnig. 2023. Steered Training Data Generation for Learned Semantic Type Detection. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–25.

[37] Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2024. Table-GPT: Table Fine-tuned GPT for Diverse Table Tasks. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–28.

[38] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[39] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA, 281–297.

[40] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. 1995. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review* 102, 3 (1995), 419.

[41] Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*. Vol. 24. Elsevier, 109–165.

[42] Hiroaki Morikawa. 2019. Semantic Table Interpretation using LOD4ALL. *SemTab@ ISWC* 2019 (2019), 49–56.

[43] Phuc Nguyen, Natthawut Kertkeidkachorn, Ryutaro Ichise, and Hideaki Takeda. 2019. Mtab: Matching tabular data to knowledge graph using probability models. *arXiv preprint arXiv:1910.00246* (2019).

[44] Daniela Oliveira and Mathieu d'Aquin. 2019. ADOG-Annotating Data with Ontologies and Graphs. *SemTab@ ISWC* 2019 (2019), 1–6.

[45] 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.

[46] Erhard Rahm and Philip A Bernstein. 2001. A survey of approaches to automatic schema matching. *the VLDB Journal* 10, 4 (2001), 334–350.

[47] Vijayshankar Raman and Joseph M Hellerstein. 2001. Potter's wheel: An interactive data cleaning system. In *VLDB*, Vol. 1. Morgan Kaufmann, San Francisco, CA, USA, 381–390.

[48] S Krishnamurthy Ramnandan, Amol Mittal, Craig A Knoblock, and Pedro Szekely. 2015. Assigning semantic labels to data sources. In *European Semantic Web Conference*. Springer, Cham, Switzerland, 403–417. https://doi.org/10.1007/978-3-319-18818-8_25

[49] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. 2017. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2001–2010.

[50] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. 2018. Learning to learn without forgetting by maximizing transfer and minimizing interference. *arXiv preprint arXiv:1810.11910* (2018).

[51] Dominique Ritze, Oliver Lehmberg, and Christian Bizer. 2015. Matching html tables to dbpedia. In *Proceedings of the 5th international conference on web intelligence, mining and semantics*. 1–6.

[52] Anthony Robins. 1993. Catastrophic forgetting in neural networks: the role of rehearsal mechanisms. In *Proceedings 1993 The First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*. IEEE, 65–68.

[53] Anthony Robins. 1995. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science* 7, 2 (1995), 123–146.

[54] Burr Settles. 2009. Active learning literature survey. (2009).

[55] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2022. Annotating columns with pre-trained language models. In *Proceedings of the 2022 International Conference on Management of Data*. ACM, New York, NY, USA, 1493–1503. https://doi.org/10.1145/3514221.3517906

[56] Kai Sun, Yifan Xu, Hanwen Zha, Yue Liu, and Xin Luna Dong. 2024. Head-to-Tail: How Knowledgeable are Large Language Models (LLMs)? AKA Will LLMs Replace Knowledge Graphs?. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 311–325.

[57] Yushi Sun, Hao Xin, and Lei Chen. 2023. RECA: Related Tables Enhanced Column Semantic Type Annotation Framework. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1319–1331.

[58] Yushi Sun, Hao Xin, Kai Sun, Yifan Ethan Xu, Xiao Yang, Xin Luna Dong, Nan Tang, and Lei Chen. 2024. Are Large Language Models a Good Replacement of Taxonomies? *arXiv preprint arXiv:2406.11131* (2024).

[59] 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).

[60] Petros Venetis, Alon Y Halevy, Jayant Madhavan, Marius Pasca, Warren Shen, Fei Wu, and Gengxin Miao. 2011. Recovering semantics of tables on the web. *Proceedings of the VLDB Endowment* 4, 9 (2011), 528–538.

[61] Adrian Vogelsgesang, Michael Haubenschild, Jan Finis, Alfons Kemper, Viktor Leis, Tobias Mühlbauer, Thomas Neumann, and Manuel Then. 2018. Get real: How benchmarks fail to represent the real world. In *Proceedings of the Workshop on Testing Database Systems*. 1–6.

[62] Daheng Wang, Prashant Shiralkar, Colin Lockard, Binxuan Huang, Xin Luna Dong, and Meng Jiang. 2021. TCN: Table Convolutional Network for Web Table Interpretation. In *Proceedings of the Web Conference 2021*. ACM, New York, NY, USA, 4020–4032. https://doi.org/10.1145/3442381.3450090

[63] Runhui Wang, Yuliang Li, and Jin Wang. 2022. Sudowoodo: Contrastive Self-supervised Learning for Multi-purpose Data Integration and Preparation. *arXiv preprint* (2022). arXiv:2207.04122

[64] Georg Wiese, Dirk Weissenborn, and Mariana Neves. 2017. Neural Domain Adaptation for Biomedical Question Answering. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*. 281–289.

[65] Xiao Yang, Kai Sun, Hao Xin, Yushi Sun, Nikita Bhalla, Xiangsen Chen, Sajal Choudhary, Rongze Daniel Gui, Ziran Will Jiang, Ziyu Jiang, et al. 2024. CRAG–Comprehensive RAG Benchmark. *arXiv preprint arXiv:2406.04744* (2024).

[66] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. ACL, 8413–8426. https://doi.org/10.18653/v1/2020.acl-main.745

[67] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? *Advances in neural information processing systems* 27 (2014).

[68] Dan Zhang, Madelon Hulsebos, Yoshihiko Suhara, Çağatay Demiralp, Jinfeng Li, and Wang-Chiew Tan. 2020. Sato: contextual semantic type detection in tables. *Proceedings of the VLDB Endowment* 13, 12 (2020), 1835–1848.

[69] Heidi C Zhang, Sina J Semnani, Farhad Ghassemi, Jialiang Xu, Shicheng Liu, and Monica S Lam. 2024. SPAGHETTI: Open-Domain Question Answering from Heterogeneous Data Sources with Retrieval and Semantic Parsing. *arXiv preprint arXiv:2406.00562* (2024).

[70] Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. 2024. TableLlama: Towards Open Large Generalist Models for Tables. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 6024–6044.

[71] Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, et al. 2024. A survey on efficient inference for large language models. *arXiv preprint arXiv:2404.14294* (2024).