Robust Table Integration in Data Lakes: From Integrable Set Discovery to Multi-Tuple Conflict Resolution

Daomin Ji[1], Hui Luo[2], Zhifeng Bao[1], J. Shane Culpepper[3]
[1]RMIT University, [2]University of Wollongong, [3]The University of Queensland
daomin.ji@student.rmit.edu.au,huil@uow.edu.au,zhifeng.bao@rmit.edu.au,s.culpepper@uq.edu.au

ABSTRACT

In this paper, we study two core tasks related to the integration of tables within data lakes: 1) *integrable set discovery* finds all integrable sets, each comprising a set of tuples that can be safely integrated, taking into account occurrences of semantic equivalence and typographical errors. 2) *multi-tuple conflict resolution* resolves conflicts among multiple tuples to be integrated. In a first step, we train a binary classifier to judge whether two tuples should be integrated when faced with semantic equivalence and typographical errors. Given the scarcity of labeled data in data lakes, we propose a self-supervised adversarial contrastive learning algorithm. It uses data augmentation methods and adversarial examples to automatically generate new training data, hence mitigating the challenge of scarce labeled data. Subsequently, upon the integrability of any two tuples as judged by the binary classifier, we proceed to tackle the task of Integrable Set Discovery by casting it to a problem of finding connected components in a graph, and solving it with a depth-first search algorithm. Next, to tackle the task of *multi-tuple conflict resolution*, we propose an in-context learning based method. It leverages the extensive knowledge embedded in pretrained large language models to resolve conflicts that arise during the integration of multiuple tuples, and notably, it also requires minimal labeled data. Since there are no suitable benchmarks for our tasks, we create our own benchmarks using two real-word data pools, Real and Join, which have been made publicly available. Extensive experiments have been conducted to verify the effectiveness of our proposed methods in comparison to state-of-the-art approaches.

PVLDB Reference Format:
Daomin Ji, Hui Luo, Zhifeng Bao, J. Shane Culpepper. Robust Table Integration in Data Lakes: From Integrable Set Discovery to Multi-Tuple Conflict Resolution. 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/jdm199605/Robin.

1 INTRODUCTION

Data lakes [28, 49] are large repositories that can store various kinds of raw data. Recently, investigating table search algorithms [21, 22, 33, 50, 51, 72] has emerged as a prominent research topic, aiming to find unionable, joinable or related tables in data lakes. The integration of such tables into a single and comprehensive table can help yield new knowledge and insights that would be otherwise inaccessible when utilizing the tables individually. Meanwhile, it poses significant challenges since the tables usually come from a multitude of diverse resources. To enhance the process of table integration in data lakes, three core tasks have been identified: 1) *Schema Alignment* aims to align attributes across multiple tables; 2) *Integrable Set Discovery* aims to identify all integrable sets, each comprising a set of tuples that are judged as integrable; 3) *Multi-tuple Conflict Resolution* aims to resolve conflicts that may arise when integrating multiple tuples.

Most recently, a pioneering work ALITE [34] made the first attempt to integrate tables within data lakes. ALITE has demonstrated promising outcomes on the task of schema alignment, so the input tables can be safely combined into an intermediate table T for the usage of the following two tasks. However, while ALITE lays a solid foundation for integrable set discovery and multi-tuple conflict resolution, it is noteworthy that it does not address two critical yet commonly encountered cases in practice – semantic equivalence and typographical errors – which limit ALITE's suitability to the two tasks.

Case 1: Semantic Equivalence and Typographical Errors. The tuples marked in grey in Fig. 1 are instances of semantic equivalence and typographical errors. Consider the two tuples t_1 and t_2, their values for the attribute **Country**, namely "United States" and "U.S.", are semantically equivalent, since they refer to the same country. Ideally, t_1 and t_2 can be integrable. Furthermore, tuple t_3's value for the attribute **Country**, "United Stakes", is a typographical error of that of t_1. Ideally, t_1 and t_3 are also integrable. However, ALITE lacks the capability to address semantic equivalence or typographical errors, and consequently, it fails to integrate these cases effectively.

Case 2: Conflicts are a key issue in multi-tuple integration and must be appropriately resolved during the integration process. As illustrated in Fig. 1, tuple t_4 can be integrated with either t_5 or t_6. However, when attempting to fill the missing value of the attribute **Star** in t_4 using the corresponding attribute values from t_5 or t_6, two different possibilities are encountered: "Joaquin Phoenix" and "Tom Cruise". This constitutes an attribute value conflict. To resolve this conflict, ideally our solution should select the correct value, "Joaquin Phoenix" to fill the attribute **Star** in t_4. Unfortunately, ALITE lacks support for conflict resolution. Hence, it integrates both (t_4, t_5) and (t_5, t_6) into new tuples, resulting in t_9 and t_{10}, respectively. This leads to the inclusion of incorrect information remains in the new table, potentially causing negative effects on downstream tasks.

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

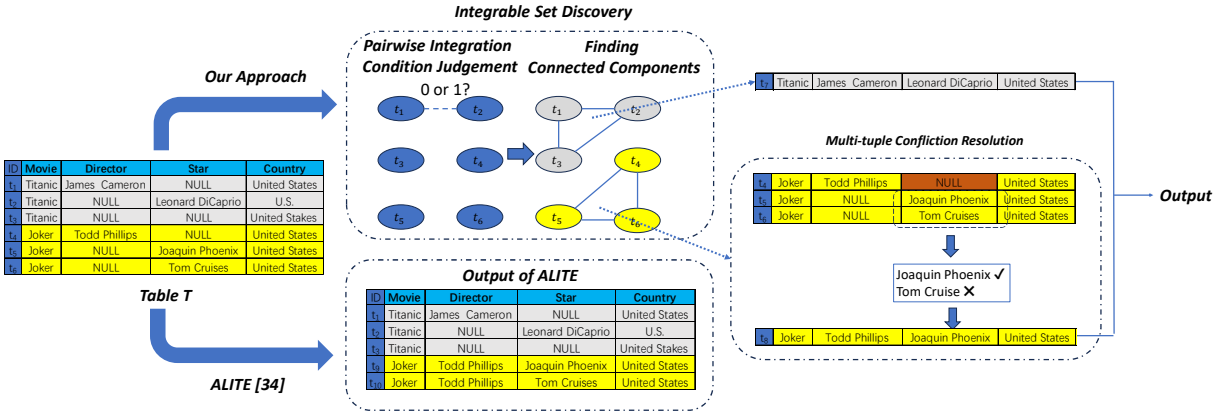


Figure 1: An example illustrating ALITE's limitations and our proposed approach.

Furthermore, it is worth noting that we have identified other related work in the field that encounter similar challenges to ours. Specifically, entity resolution [19, 41, 48, 65] encounters similar challenges to our integrable set discovery task. To address such challenges, a large number of entity resolusion methods [19, 41, 65] are devised based on deep learning models, which can bring significant performance improvement. While achieving promising results, a common constraint is their heavy reliance on the availability of massive labeled data collections for training a powerful and reliable deep learning model. Unfortunately, in many data integration tasks and particularly within data lakes [49], labeled data is often scarce or entirely absent. The manual labeling of new data is both expensive and time consuming, further limiting the practical application of these techniques to our problem. Furthermore, conflict resolution for data fusion [6, 18, 38, 59, 67, 71] encounters similar challenges to our multi-tuple conflict resolution task. Similarly, most existing methods for conflict resolution for data fusion [59, 67, 67] rely on massive labeled data or meta-data to estimate the trustworthiness of each data resource, which also restricts their application to the task of multi-tuple conflict resolution in data lakes.

To overcome the above limitations, we propose a new approach to multi-tuple integration. Our approach primarily focuses on two tasks: *integrable set discovery* and *multi-tuple conflict resolution*. More importantly, we achieve so by using limited number of labeled data, as illustrated in the upper part of Fig. 1.

In particular, to tackle the task of *integrable set discovery*, we devise a two-stage solution. In the first stage, our objective is to accurately predict the integrability of tuple pairs in Table T, even when they contain semantic equivalence and typographical errors. This challenge can be framed as a binary classification problem. In the second stage, based on the integrability judgement made by the classifier for any two tuples, our objective is to find all integrable sets within T. This can be framed as a problem of finding connected components in a graph. While the latter problem can be resolved using a DFS-based algorithm, the former problem presents greater complexity. That is primarly due to the scarcity of labeled data, which can hinder the development of a reliable binary classifier.

To address this issue, for each tuple t, we regard the semantically equivalent or typographically erroneous version as a minor perturbation of t. Then, we employ data augmentation techniques and adversarial examples to simulate these minor perturbations. This strategy allows us to automatically generate a sufficient amount of training data for training a binary classifier, overcoming the challenge of limited labeled data.

Next, to tackle the task of *multi-tuple conflict resolution*, we depart from the conventional conflict resolution approaches for data fusion that typically necessitate extensive labeled data collections. Instead, we propose a novel in-context learning (ICL)-based method. This method leverages the extensive knowledge contained in pre-trained large language models (LLMs), thereby effectively reducing the model's reliance on large labeled data while maintaining comparable performance level. Specifically, our ICL-based method proceeds in three steps – (1) *Instruction Template Construction*: we create an instruction template designated to transform the conflict resolution task into a natural language query. (2) *Demonstration Examples Selection*: a small number of tuples are carefully selected and labeled as demonstration examples. Remarkably, even with just a few labeled examples, it can provide sufficient contextual information to guide the LLM in reliably resolving conflicts. (3) *Conflict Resolution*: armed with the demonstration examples and the natural language query, we utilize the capability of a large language model, such as T5 [57], to score candidate resolutions and select the one with the highest score.

In summary, the above approach exhibits the potential to achieve a robust table integration within data lakes, and specifically, we make the following contributions:

- To solve the task of integrable set discovery, we propose a novel self-supervised adversarial contrastive learning framework, namely SSACL, to train a binary classifier using limited labeled data, enabling it to predict pairwise integrability for any tuple pair (Sec. 3). Leveraging the predictions made by this classifier, we identify all integrable sets by detecting connected components in a graph, a problem that can be efficiently solved using a depth-first search (DFS) algorithm. (Sec. 4)
- To solve the task of multi-tuple conflict resolution, we develop an in-context learning (ICL) [3, 47, 68] method, namely ICLCF,

which demonstrates promising performance even with limited labeled data. (Sec. 5)
- Since no suitable benchmarks exist for evaluating our tasks, we have taken initiatives to create our own benchmarks. Our benchmarks are established upon two existing data pools, Real and Join [34], where we inject different types and percentages of errors and conflicts. We also have publicly released our benchmarks[1]. (Sec. 6)
- We conduct extensive evaluations of our methods and competitors on these benchmarks. When compared against the best-performing competitors, our SSACL exhibits a relative improvement of 3.87% and 3.64% in terms of *F1* and *similarity*, and our ICLCF achieves a relative improvement of 20.8% in accuracy. Both SSACL and ICLCF only experience a decrease in performance of less than 5%, under a scenario of limited labeled data. (Sec. 7)

2 PROBLEM FORMULATION

In this section, we will introduce the problem definition of two tasks, *integrable set discovery* and *multi-tuple conflict resolution*.

Following the state-of-the-art work [34], we also assume the schema of the input tables have been aligned, and they have been integrated into an intermediate table T. Taking T as the input, *integrable set discovery* aims to find all integrable sets $\mathcal{S}$, each of which encompasses a collection of tuples awaiting for integration. For each integrable set $S \in \mathcal{S}$, we aim to integrate all the tuples inside into a new tuple t_{new} by implementing the task of *multi-tuple conflict resolution*. Before providing the problem formulation of the first task, a fundamental operator is presented to judge whether two tuples can be integrated in Definition 1.

Definition 1. ***Pairwise Integration Condition Judgement*** *Given a tuple pair (t_i, t_j) that may contain semantic equivalence and typographical errors, the judgement serves as a function $f(t_i, t_j)$, which outputs* 1 *or* 0 *to indicate whether t_i and t_j are integrable or not.*

Having defined pairwise integration condition judgement, we are now ready to formulate the task of integrable set discovery.

Definition 2. ***Integrable Set Discovery.*** *Given a table T, an integration condition judgement function f, it produces multiple disjoint integrable sets, denoted as $\mathcal{S} = \{S_1, S_2, ..., S_{|\mathcal{S}|}\}$. Each integrable set S ($S \in \mathcal{S}$) comprises a collection of tuples that meet two conditions:*

- *for each tuple $t_i \in S$, there must exist at least one tuple $t_j \in S$ where $i \neq j$, such that $f(t_i, t_j) = 1$;*
- *for each tuple $t_i \in S$, and for any tuple $t_j \in T/S$, $f(t_i, t_j) = 0$.*

For each integrable set $S \in \mathcal{S}$ that has been discovered, by Definition 2 all tuples in S are regarded as integrable into a single tuple, denoted as t_{new}. This integration process involves filling each attribute of t_{new} with a correct value. However, when a certain attribute of t_{new} has multiple distinct values originating from different tuples in S, we call it as a conflict and formalize the task of *multi-tuple conflict resolution* below to address it.

Definition 3. ***Multi-tuple Conflict Resolution.*** *Given an integrable set S, it produces a tuple t_{new} by integrating all tuples in S, such that: for each attribute a in t_{new} where a conflict exists due to multiple different values, a correct value v^* is chosen from a candidate set $C = \{t[a] | t \in S \wedge t[a] \neq NULL\}$ to fill in its attribute value $t_{new}[a]$.*

[1] https://zenodo.org/record/8310401

3 PAIRWISE INTEGRATION CONDITION JUDGEMENT

3.1 Main Idea

From Definition 1, it becomes clear that the pairwise integration condition judgement function f can be interpreted as a binary classifier, and our goal is to learn a model that approximates an optimal classifier f^*. This model will produce an output of 1 or 0 according to whether two tuples are deemed integrable or not.

Specifically, we encode each tuple t using a semantically meaningful and representative embedding vector, denoted as $\boldsymbol{emb}(t)$, and aim to train a model $\mathcal{M}$ to approximate the behavior of the optimal classifier. However, a significant hurdle is the unavailability of a labeled dataset in data lakes, and manually labeling the dataset would be prohibitively costly, as discussed in Sec. 1.

In order to overcome this hurdle, we propose a novel method, namely *self-supervised adversarial contrastive learning* (SSACL), which is able to automatically generate training data that contains both positive and negative instances. For a given tuple t, although the negative case (i.e., tuples that cannot be integrated with t) can be obtained using negative sampling [4, 5], the key hurdle is *how to generate positive instances (i.e., tuples that can be integrated with t)*. So, for each tuple t, we introduce a slight perturbation function p to generate a positive instance t^+, i.e., $t^+ = p(t)$. This slight perturbation function p introduces minimal semantic divergence between t and t^+. Specifically, we employ two strategies to simulate p: *data augmentation* [7, 25, 60] and *adversarial examples* [27, 30, 70], which will be elaborated in Sec. 3.2 and Sec. 3.5, respectively.

Naturally, for a tuple pair (t, t^+), $\mathcal{M}(\boldsymbol{emb}(t), \boldsymbol{emb}(t^+)) = 1$ should hold. Once the model $\mathcal{M}$ has been trained with enough positive training instances (t, t^+), $\mathcal{M}$ should be able to accurately infer whether two tuples are integrable, even when confronted with semantic equivalence or typographical errors.

Fig. 2 presents the architecture of our proposed SSACL, which has the following main components:

- The *Data Generator* introduces data augmentation and negative sampling methods to generate positive instances t^+ and negative instances t^- for each tuple t, to form a training data set $\mathcal{D}_{train}$. (Sec. 3.2)
- The *Encoder* transforms each tuple $t \in \mathcal{D}_{train}$ generated from the *data generator* to a compact and semantically meaningful representation $\boldsymbol{emb}(t)$. (Sec. 3.3)
- The *Matcher* is a machine learning model $\mathcal{M}$ which is trained using $\mathcal{D}_{train}$. This model is used to approximate the optimal classifier f^*; it includes embeddings from the *encoder* as input, and outputs a 1 or a 0 to decide whether two tuples are integrable. (Sec. 3.4)
- The *Adversarial Trainer* is designed to create additional positive training instances using adversarial examples. Additionally, it can be used to optimize the parameters of *matcher* during model training. (Sec. 3.5)

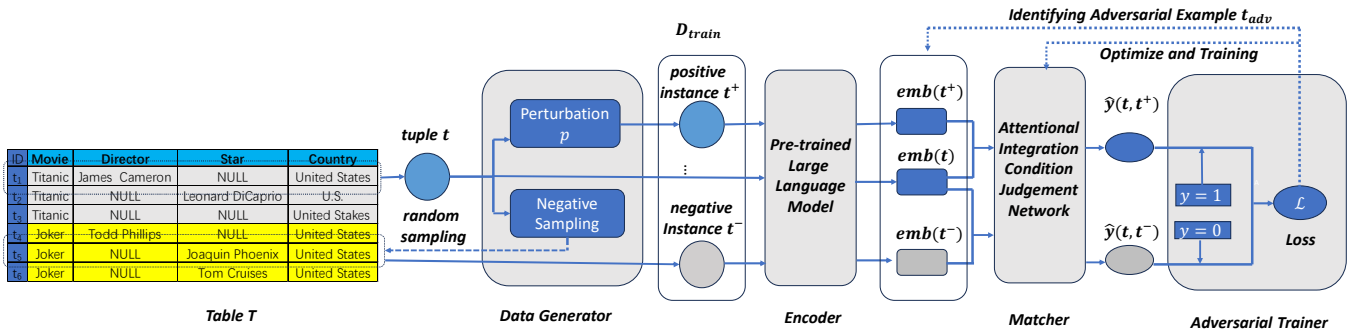


Figure 2: The Architecture of Self-Supervised Adversarial Contrastive learning (SSACL) Method

3.2 Data Generator

A Data generator automatically generates training data for the integrable set discovery task. Specifically, for each tuple t, data is augmented to produce a collection of perturbated tuples, $\mathcal{P}_t = \{t_i^+ | t_i^+ = p_i(t)\}$, where p_i corresponds to a specific perturbation function. Consequently, every tuple pair (t, t_i^+) is a positive training instance for the matcher. Obviously, deciding how to create these slight perturbations p_i impacts the quality of the training data, which in turn affects the performance of the training model $\mathcal{M}$. So, the selection of perturbation functions should be reasonable and comprehensive. Perturbation functions are carefully selected that adhere to two key principles [25, 61]: 1) the perturbations should preserve the semantics of the original tuple t or make only minimal changes to it; 2) the perturbation functions should ideally cover a diverse range of possibilities to effectively represent multiple real-world scenarios. To achieve this, we develop a variety of perturbation functions which are organized into three types: attribute-level, word-level, and character-level.

Attribute-level. There are two kinds of attribute-level perturbations, namely attribute removal and attribute substitution.

- *Attribute removal.* This is applied to both numerical and text attributes. We randomly select a non-missing attribute value a of a tuple t, and create a new tuple t^+ by removing its attribute value, i.e., $t^+[a] == NULL$.
- *Attribute substitution.* This is applied to text attributes only. We randomly select a non-empty attribute A of a tuple t, create a new tuple t^+ by using backtranslation methods [20, 61], which generate a semantically equivalent description for $t[a]$, and use this new description to replace the original value $t[a]$. Notably, backtranslation is a widely used data augmentation method in NLP to generate a sentence with the same or similar semantics to the original sentence. The method first translates the original sentence into a target language and then reverses the translation using a reverse model.

Word-level. There are three types of word-level perturbations – word removal, word substitution, and word swapping – all of them are applied to text attributes only.

- *Word Removal.* For a tuple's attribute value, we randomly select a word and delete it.
- *Word Substitution.* For a tuple's attribute value, we randomly select a word, and use WordNet [46] to find a synonyms or hypernyms to form a candidate set. Finally, we randomly select a word from the candidate set to substitute the original word.
- *Word Swapping.* For a tuple's attribute value, we randomly select two neighboring words, and reverse the ordering.

Character-level. In character-level perturbations, we only simulate common typographical errors to create new tuples, which can be applied to both text attributes and numerical attributes. More details on how we simulate common typographical errors can be found in a technical report [1].

Specifically, for each original tuple t, we randomly select N_{pos} perturbation functions to perturb the tuples t^+. In other words, we create N_{pos} positive training instances for each original tuple t. And for negative training instances, we adpopt a widely used strategy negative sampling [4, 5] to uniformly select N_{neg} tuples (t to be excluded) at random in the training table for each tuple t.

3.3 Encoder

Given a tuple t, an *encoder* learns a compact and semantically meaningful embedding $\boldsymbol{emb}(t)$ to represent t. In this paper, we adopt attribute-level representations for each tuple, enabling us to perform fine-grained comparisons with the attribute values from any two tuples. Specifically, for each tuple t, the *encoder* encodes the following representation:

$$\boldsymbol{emb}(t) = [\boldsymbol{emb}(t[a_1]), \boldsymbol{emb}(t[a_2]), ..., \boldsymbol{emb}(t[a_m])]. \quad (1)$$

Here, m denotes the number of attributes in Table T, $t[a_i]$ represents the value of the attribute A_i in the tuple t, and $\boldsymbol{emb}(t[a_i])$ denotes the corresponding embedding. Note that for a tuple t, there may be missing attributes. To handle this case, we use a special token $[NULL]$ to mark missing values, which will also be mapped to an embedding. Furthermore, for each tuple t, we also create a masking vector $\boldsymbol{Mask}(t)$ that has m dimensions:

$$\boldsymbol{Mask}(t) = [d_1, d_2, ..., d_m], \quad (2)$$

where we set the i-th element of the mask vector d_i to 0 if the corresponding attribute value for tuple t is missing; otherwise, we set it to 1. The masking vectors are discussed further in Sec. 3.4.

To obtain $\boldsymbol{emb}(t[a_i])$, we first serialize an attribute value $t[a_i]$ into a sequence of words. Then, for each word w in the sequence, we generate its embedding from a pre-trained language model. Technically speaking, there are two different embedding methods, namely word-level embeddings such as word2vec [45] and GloVE [55], and subword-level embeddings such as FastText [11] and BERT [32]. Word2vec encodes each word individually uses an embedding vector to represent it, whereas GloVE tokenizes a word into a sequence of subwords and represents each subword using an embedding. In this work, we adopt subword-level embedding methods. This choice is driven by their capability to handle unknown and uncommon

words, while also exhibiting greater resilience to typographical errors.

For every sequence of tokens, the respective token embeddings are aggregated into a single embedding vector. Thus we adopt a transformer-based architecture [32] for this aggregation process. This choice stems from the proven capacity to adeptly capture contextual information embedded within sequences in a transformer. As a result, we obtain an embedding $\boldsymbol{emb}(t[a_i])$ tuned for the matcher, which we will discuss in Section 3.4.

3.4 Matcher

Given the embedding representations of two tuples $\boldsymbol{emb}(t_1)$ and $\boldsymbol{emb}(t_2)$, the matcher $\mathcal{M}$ outputs a 1 or a 0, indicating whether the two tuples should be integrated or not. One straightforward way to achieve so is to compute the cosine similarity between $\boldsymbol{emb}(t_1)$ and $\boldsymbol{emb}(t_2)$, or feed the concatenation of the two embeddings into a Multi-layer Perceptron (MLP). A common drawback emerges from the uniform treatment of every attribute, which ignores potential variations in their contribution to the semantic correctness of a given tuple.

To overcome this drawback, we propose an *Attentional Integration Condition Judgment Network* (AICJNet). Specifically, for two embeddings $\boldsymbol{emb}(t_1)$ and $\boldsymbol{emb}(t_2)$, we first concatenate them into a single embedding: $\boldsymbol{emb}(t_1, t_2) = [\boldsymbol{emb}(t_1[a_1]), ..., \boldsymbol{emb}(t_2[a_1]), ...]$. Then we consider the varying importance across different attributes and reformulate the representation of a tuple pair (t_1, t_2) using a self-attention mechanism [66]:

$$\boldsymbol{emb}^*(t_1, t_2)_i = \sum_{j=1}^{2m} \text{softmax}\left(\boldsymbol{Mask}(t_1, t_2)_j \frac{\boldsymbol{Q}_i \cdot \boldsymbol{K}_j^T}{\sqrt{d_k}}\right) \cdot \boldsymbol{V}_j. \quad (3)$$

Here, $\boldsymbol{emb}^*(t_1, t_2)$ represents the final representation of the tuple pair (t_1, t_2), d_k represents the size of each attribute embedding, and $\boldsymbol{Mask}(t_1, t_2)$, as the concatenation of $\boldsymbol{Mask}(t_1)$ and $\boldsymbol{Mask}(t_2)$, is used to mask the impact of a missing attribute value to other attributes. $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ are the query matrix, the key matrix, and the value matrix, respectively, which are computed using a linear transformation [66].

Finally, with $\boldsymbol{emb}^*(t_1, t_2)$ as the input, we use an MLP to output a binary decision y, indicating whether t_1 and t_2 can be integrated.

3.5 Adversarial Trainer

The adversial trainer serves dual purposes: 1) establish a training objective to be optimized using an SGD-based algorithm; 2) identify adversarial examples to further enrich the training set. Next, we will explain this idea in details.

Objective Function. In this work, we use a binary noise contrastive estimation (NCE) loss function [35] for the training objective, which can be formulated as:

$$\mathcal{L} = \sum_{i=1}^{N} \left(\sum_{j=1}^{N_{pos}} log\mathcal{M}(t_i, t_{ij}^+) + \sum_{j=1}^{N_{neg}} log\mathcal{M}(t_i, t_{ij}^-) \right). \quad (4)$$

Here, N is the total number of tuples in the training table, and t_{ij}^+ and t_{ik}^- denote one of the positive instances and negative instances of tuple t_i, respectively.

Adversarial Examples and Training. We propose the use of adversarial examples [30, 70] to further enrich the pool of positive training instances in our model. Technically speaking, an adversarial example is typically viewed as a perturbed version of an original input example, which results in a significant impact in a decision made by a machine learning model. In contrast to perturbations in data augmentation that operate on original tuples, perturbations in adversarial training directly impact the embedding vectors, which can be expressed as:

$$p(\boldsymbol{emb}(t)) = \boldsymbol{emb}(t) + \boldsymbol{r}, \quad (5)$$

where $\boldsymbol{r}$ denotes the perturbation vector. Since adversarial examples t^{adv} have the most significant impact on a model's performance, this can also be expressed as the following objective function:

$$max_{\boldsymbol{r}} \mathcal{L}(\mathcal{M}((t_i, t_i^{adv}), y_i), s.t. ||\boldsymbol{r}||_2 < \epsilon. \quad (6)$$

Here, ϵ is a small value that constrains the perturbation, y_i is set to 1 since the pair of original tuples and the corresponding adversarial example (t, t^{adv}) should consistently result in a positive decision.

Since r is very small, the loss function is approximately equivalent to the following equation derived using a first-order Taylor approximation [52]:

$$\mathcal{L}(\mathcal{M}((t_i, t_i^{adv}), y_i) \approx \mathcal{L}(\mathcal{M}((t_i, t_i), y_i) + \nabla_{t_i} \mathcal{L}(\mathcal{M}((t_i, t_i), y_i)^T \boldsymbol{r}. \quad (7)$$

When using a Lagrange Multiplier Method [12] to solve Eq. 6 and Eq. 7, we can get:

$$\boldsymbol{r} = -\epsilon \frac{\nabla_{t_i} \mathcal{L}(\mathcal{M}((t_i, t_i), y_i)}{||\nabla_{t_i} \mathcal{L}(\mathcal{M}((t_i, t_i), y_i)||^2} \quad (8)$$

Consequently, for each tuple t, we can derive an adversarial example t_i^{adv}. This process is repeated in every training epoch, with (t_i, t_i^{adv}) being added the positive training instance pool.

4 INTEGRABLE SET DISCOVERY

Given integrability of any two tuples in Table T judged by the binary classifier, we introduce how to derive all the integrable sets contained in T in this section. Most exisitng work, such as ALITE [34], is restricted to integrating two tuples in each step of the algorithm. This limitation precludes including them to resolve conflicts that may arise in a set of more than two tuples. In contrast, *our objective is to integrate multiple tuples simultaneously*, providing an opportunity to detect conflicts and resolve them when they occur. So, the goal is: given a set of tuples drawn from a table T, identify all the integrable sets within Table T, where each integrable set refers to a set of tuples that can be integrated, as shown in Definition 2. To achieve this goal, we reformulate the problem as finding all connected components in a graph, where each component represents an integrable set. Specifically, this problem can be solved as follows:

- **Graph Construction**. We first construct a graph $\mathcal{G} = \{V, E\}$ from a table T based on the predicted results from pairwise integration condition judgement. Each vertex $v_i \in V$ corresponds to a distinct tuple t_i. When the context is clear, we use v and t interchangeably. For every tuple pair (t_i, t_j) where $\mathcal{M}(t_i, t_j) = 1$, an edge e_{ij} is established between v_i and v_j.

- **Identifying Connected Components**. In the graph $\mathcal{G}$, every connected component is a subgraph $\mathcal{G}'$, such that for any two vertices v_i and v_j in $\mathcal{G}'$, there exists at least one path from v_i to v_j. Since each edge e_{ij} indicates the integratability between t_i and t_j, it follows that a connected component is exactly the integrable set that we hope to identify. To identify all the connected components in a graph $\mathcal{G}$, we use a depth-first search (DFS) algorithm. Specifically, all vertexes are set as unvisited initially. Then, every time we visit an unvisited vertex v, we traverse all vertexes that can be reached from v and add them to a vertex set S_i. After every vertex has been visited, we can obtain multiple S_i, each of which is an integrable set. The DFS algorithm is straightforward and well-known, so we refer readers to our technical report [1] for using it to find connected components.

Example 1. *Taking the table T in Fig. 1 for instance, we construct a graph $\mathcal{G}$ based on the results of integration condition judgements. Using a DFS algorithm, we find two connected components in the graph $\mathcal{G}$, whose vertex sets are $V_1' = \{t_1, t_2, t_3\}$, $V_2' = \{t_4, t_5, t_6\}$, respectively.*

5 CONFLICT RESOLUTION IN MULTI-TUPLE INTEGRATION

Based on integrable set discovery, we obtain a collection of integrable sets $\mathcal{S}$. For each integrable set $S \in \mathcal{S}$, we must integrate all tuples contained in S into a *single* tuple t_{new}, which will provide more comprehensive information than the individual tuples $t \in S$. Thus, the problem now is how to determine each attribute value $t_{new}[a]$, which can fall into one of three potential types:

- **Non-missing Attribute.** An attribute a refers to a non-missing attribute if $t[a] \neq NULL$ for any $t \in S$. In this case, we randomly select a tuple $t \in S$ and use the attribute value $t[a]$ to fill $t_{new}[a]$. This approach is justified by the fact that, given the nature of integrable set discovery, all of the tuple values for a non-missing attribute a possess equivalent semantics within the integrable set.
- **Fully-Missing Attribute.** An attribute a is referred to as a fully-missing attribute if $t[a] == NULL$ for any $t \in S$. In this case, $t_{new}[a]$ is assigned a $NULL$ value. This choice is motivated by the absence of any reliable information to fill $t_{new}[a]$.
- **Partially-missing Attribute.** An attribute a is referred to as a partially-missing attribute, if at least one tuple $t_i \in S$ has $t_i[a] == NULL$ and at least one tuple $t_j \in S$ has $t_j[a] \neq NULL$. Given the heterogeneous nature of the data in the input tables, non-missing values for multiple tuples for the attribute a may conflict with each other, thus resulting in a candidate set $C(a) = \{t[a] | t \in S \wedge t[a] \neq NULL\}$. To resolve this, we must select the correct value $v^* \in C$ to fill $t_{new}[a]$.

The first two types of attributes can be resolved easily, and hence we focus on how to fill partially-missing attributes. This can be formulated as a conflict resolution problem, as articulated in Definition 3. Existing solutions for conflict resolution in the context of data fusion problem rely on truth discovery approaches [38, 39, 71], which first estimate the trustworthiness of a data source, and subsequently choose the value from the most reliable source to fill the missing value. However, truth discovery approaches typically rely on a lot of training data or metadata (e.g., paper citation and reviews) in order to estimate source reliability. This imposes limitations on the applicability of these methods, as practical scenarios often lack access to the type of data needed, and obtaining such data necessitates considerable human curation and financial cost.

Our solution for conflict resolution is inspired by the recent success of in-context learning (ICL) [3, 47, 68] within the field of natural language processing. ICL-based methods mainly rely on a large language model (LLM) to make predictions in downstream tasks, such as semantic analysis.

Given that an LLM is pre-trained on an extensive corpora to accumulate a broad spectrum of knowledge, ICL models exhibit impressive performance even when fine-tuned using limited labeled data from a target corpora. This allows us to extend predictions to downstream machine learning tasks despite constraints posed by having limited training data. Specifically, ICL-based algorithms have three main steps: 1) design an instruction template tailored to the requirements of the downstream task; 2) select k demonstration examples that conform to the prescribed instruction template; 3) deploy an LLM to make predictions. Next, we will elaborate how we design an ICL-based algorithm to effectively address our problem.

Step 1: Designing an Instruction Template. Technical speaking, an instruction template is used to transform a downstream task to a natural language query, such that it can be understood by an LLM. For example, consider a sentimental analysis task that uses a binary prediction for reviews, the instruction template can be defined as "*Review:__, Sentiment:__*". Here, "Review" and "Sentiment" are the keywords that guide the LLM. The first blank line is filled with the content from a review and the second blank line is filled with either "positive" representing the label 1, or "negative" representing the label 0; these labels will then be predicted by the LLM.

In this work, our focus centers on the task involving a given tuple t_{new}, along with its values of the non-missing attribute a_{non}. Our objective is to fill a partially-missing attribute a_{par} of the resulting tuple t_{new}, by selecting the best value v^* from the set $C(a_{par})$. To accomplish this, we create a natural language instruction template that reads as follows:

For a tuple, if its values on attribute __ is __ , on attribute __ is __, ..., respectively, then its value on attribute __ should be __.

Example 2. *For example, if we use the custom instruction template to formulate a conflict resolution task as depicted in Fig. 1, the following natural language query is created: "For a tuple, if its value on the attribute 'movie' is 'Joker', on the attribute 'director' is 'Todd Philips', on the attribute 'Country' is 'United States', respectively, then its value on the attribute 'Star' should be __." In this context, there are two candidate values, 'Joaquin Phoenix' and 'Tom Cruise', and we want to choose the correct one to populate the attribute in the tuple.*

Step 2: Selecting Demonstration Examples. Actually, demonstration examples are labeled data. With the demonstration examples, demonstration examples can tell LLMs what the correct answer is in this case, thus guiding LLMs how to make prediction on the downstream tasks. The choice of demonstration examples, denoted as D, has a significant impact on the effectiveness of the ICL approach. In this work, we investigate two different strategies to select demonstration examples: (1) The first strategy is random selection, where k tuples are randomly selected from the remaining tuples in table T to form the demonstration examples. (2) The

Algorithm 1: Multi-tuple Conflict Resolution

```
   Data: An integrable set S
   Result: A new tuple t_new
1  Tuple t_new = Dict();
2  foreach attribute a in t_new do
3      if a is a non-missing attribute then
4          randomly select a tuple t ∈ S;
5          t_new[a] = t[a];
6      else if a is a fully-missing attribute then
7          t_new[a] = NULL;
8      else
9          Candidate Set C = ∅();
10         foreach t ∈ S do
11             if t[a] ≠ NULL then
12                 C.add(t[a]);
13         Design a natural language template Temp;
14         Select k tuples from T to be demonstration examples D;
15         Query Q = String();
16         foreach t ∈ D do
17             Q.append(Temp(t));
18         Q.append(Temp(t_new));
19         Select a large language model LLM;
20         t_new[a] = LLM(Q, C)
21 return t_new;
```

second strategy is k-NN selection. For each integrable set $S \in \mathcal{S}$, an initial t_{new} is derived by collating all available values form non-missing attributes within the integrable set, which produces a set of multiple t_{new} of size $|\mathcal{S}|$. Then, we encode each t_{new} using Eq. 1 to generate an encoded representation $\boldsymbol{emb}(t_{new})$, and then compute the average embedding $\overline{\boldsymbol{emb}(t_{new})}$ of all $\boldsymbol{emb}(t_{new})$ with size $|\mathcal{S}|$. Furthermore, we perform a k-NN search to identify k tuples in table T that are the most similar to $\overline{\boldsymbol{emb}(t_{new})}$ as the demonstration examples.

Step 3: Using LLM to Make Predictions. At this moment, we have selected a collection of k demonstration examples, denoted as $\{x_i, y_i\}_{i=1}^{k}$. Here, x_i represents the template-based demonstration example t_i and y_i represents the correct value v^* for $t_i[a_{par}]$. Then, we feed the concatenation $[D; t_{new}]$ of the demonstration examples D and the target tuple t_{new} to a large language model (T5 [57]) to yield a score for each candidate value v in the set $C(a_{par})$. From these candidate values, the one with the highest score is chosen to fill entry $t_{new}[a_{par}]$.

In summary, Algorithm 1 presents the pseudocode of our method to solve multi-tuple conflict resolution. Given an integrable set S, at first, we use dictionary structure to create a new tuple t_{new} with all of the attributes to be filled (Line 1). Then we check each attribute a of t_{new} (Line 2), if a is fully missing or a partially-missing attribute, we can easily fill $t_{new}[a]$ (Line 3-7). If attribute a is a partially-missing attribute, we identify the candidate sets (Line 9-12) and follow the above three steps to fill $t_{new}[a]$ (Line 13-20).

Table 1: The Statistics of Two Data Pools

Data Pool	#Datasets	#Input Tables	#Attributes	#Tuples
Real	11	5-14	9-49	588-76,312
Join	28	2-20	5-26	260-99,883

6 DATA PREPARATION AND EVALUATION

In this section, we introduce how to prepare the benchmarks, followed by the evaluation framework to be used in the experiments.

A desirable benchmark should exhibit two key characteristics: 1) every dataset should contain semantic equivalence issues, typographical errors, and conflicts, all of which we intend to address effectively; 2) a dataset should be accompanied by definitive ground-truth annotations that label integrable sets and candidate values during conflict resolution, in order to facilitate a comprehensive effectiveness evaluation of a proposed solution. The presence of the first characteristics makes it impossible to directly use the benchmark proposed in ALITE [34], the most closely related work to ours, because errors and conflicts are not considered in ALITE's problem setting. Thus, we have created our own benchmarks by injecting semantic equivalence, typographical errors as well as conflicts, and recorded the ground-truth for evaluation.

In Sec. 6.3, we will discuss further why benchmarks from similar tasks, such as entity resolution and conflict resolution, are not suitable for our problem.

6.1 Benchmark Creation

We create our benchmarks using two data pools from ALITE [34], **Real** and **Join** [34]. Both data pools contain multiple datasets, each of which contains a set of input tables to be integrated. Table 1 illustrates the statistics for both data pools. We use $R1 - R11$ and $J1 - J28$ to denote the 11 datasets in *Real* and the 28 datasets in *Join*, respectively. Furthermore, we employ an outer-union operator [10, 34] to integrate tables in each dataset to produce a single intermediate table T, which is used to create our benchmarks.

6.1.1 Noise Injection. First, we inject noises into the clean table T, so as to simulate semantic equivalence and typographical errors for a certain error rate. Since the datasets used in the literature of data cleaning [2, 54] have a noise rate between 5% and 40%, we set the default noise rate to 30% in our experiments. Furthermore, we test three different settings of the ratios between semantic equivalence and typographical errors, 10%/20% (SE-heavy, short for semantic equivalence-heavy), 15%/15% (Balanced), and 20%/10% (TE-heavy, short for typographical error-heavy). Unless specified otherwise, we use the case of balanced noises by default.

- **Semantic Equivalence.** The technique of backtranslation [20, 61] has been widely employed to generate a range of sentences that maintain similar semantic meaning to the original sentence, in the field of natural language processing. Once a cell is chosen for noise injection, we use backtranslation [20, 61] to generate an alternative description with similar meaning to replace the original value. If the backtranslated description is the same as the original one, we randomly select a word in the original description and use WordNet [46] to generate a synonym or hypernym to replace it. We only inject this kind of noise in text attributes.
- **Typographical Errors.** We simulate typographical errors in a comprehensive way, such as character swapping and missing

characters. More details about this setting can be seen in the technical report [1]. We inject typographical errors for both text and numerical attributes.

6.1.2 Conflict Generation. To generate conflicts, for each integrable set in T (we will introduce how to obtain the ground-truth integrable set later), we choose one partially-missing text attribute a (introduced in Sec. 5), and a tuple t that has a non-empty value for attribute a. We use tuple t as a template to generate the conflict tuples. Specifically, we assume that the conflict tuples come from different resources, and different resources have a reliability score r_i between 0 and 1. Then, we randomly replace $t[a]$ with another value from attribute a in T with probability $1 - r_i$. Using this strategy, we create 3-5 conflict tuples in each integrable set. We set r_i to a value between 30%-80%.

6.1.3 Ground-truth. We also need to create the ground-truth for our benchmarks. For multi-tuple integration, we use ALITE [34] to integrate tuples in an error and conflict free manner. During the integration, we track tuple pairs that are integrated, which are considered as part of the ground-truth. Then based on the results from pairwise integration condition judgements using ALITE, we run the proposed DFS algorithm to find integrable sets in T. Thus, in our benchmarks, the ground-truth can be generated by obtaining which tuples form the integrable sets. For conflict resolution, the template tuple value $t[a]$ for attribute a is the ground-truth since we create the conflict tuples based on tuple t.

6.2 Evaluation Metrics

Recall that the multi-tuple integration is achieved via implementing two core tasks, *integrable set discovery* and *multi-tuple conflict resolution*. As mentioned in Sec. 2, pairwise integration condition judgement plays a predominant role in the first task, so we also evaluate the performance of this subtask. Hence, we introduce the evaluation metrics tailored for each of them.

Metrics for The Task of Pairwise Integration Condition Judgement. Pairwise Integration Condition Judgement serves a similar purpose to the well-known entity resolution task [19, 41, 48], so we employ *Recall*, *Precision*, and *F1* to evaluate the results. Here, $Recall = \frac{T_P}{T_P+F_N}$, $Precision = \frac{T_P}{T_P+F_P}$, and $F1$ is expressed as the harmonic sum of *Recall* and *Precision*, $F_1 = 2 \times \frac{Recall \times Precision}{Recall+Precision}$. Here, T_P denotes the number of pairs of integrable tuples that are correctly judged as integrable, F_N represents the number of pairs of integrable tuples that are incorrectly judged as not integrable, and F_P indicates the number of tuple pairs that are not integrable but judged as integrable.

Metric for The task of Integrable Set Discovery. Given a set of ground-truth integrable sets $\mathcal{R} = \{R_1, R_2, ..., R_n\}$ and a set of integrable sets $\mathcal{S} = \{S_1, S_2, ..., S_m\}$, we use the *similarity* between $\mathcal{R}$ and $\mathcal{S}$ as the evaluation metric. Then *similarity* is defined as the maximum weighted matching score in a bipartite graph, which has two disjoint sets of vertices $\mathcal{R}$ and $\mathcal{S}$, and a set of edges (R_i, S_j) between two vertices from $\mathcal{R}$ ($R_i \in \mathcal{R}$) and $\mathcal{S}$ ($S_j \in \mathcal{S}$), respectively. The weight of each edge (R_i, S_j) is the jaccard similarity $J = \frac{|R_i \cap S_j|}{|R_i \cup S_j|}$. We use the well-known Hungarian algorithm [37] to calculate the similarity scores.

Metric for The Task of Multi-tuple Conflict Resolution. We evaluate several methods in terms of *Accuracy*, which is the ratio computed by dividing the number of correctly filled attributes by the total number of conflict attributes available.

6.3 Further Discussion on Other Benchmarks

Although previous work [19, 40, 48] on entity resolution and conflict resolution provide their associated benchmarks, they cannot be used in our experiments for the following reasons: 1) In most entity resolution benchmarks [19, 41], a tuple can be matched with another tuple, which means each integrable set contains only two tuples and there is no need for conflict resolution. 2) Since T is obtained from an outer-union operator [10, 34] by using several tables of different schemas, it usually produces many missing values in most real-world scenarios, while in benchmarks for entity resolution, there are usually a small number of missing values. This makes it difficult to compare directly against these methods. 3) Most entity resolution studies have only a few attributes involved in their benchmarks. For example, DBLP-scholar [41, 65], one of the most widely used benchmarks in entity resolution, has only 5 attributes, while the average number of attributes in the datatsets within *Real* and *Join* data pools are 19.6 and 10.6, respectively.

7 EXPERIMENT

7.1 Experimental Setup

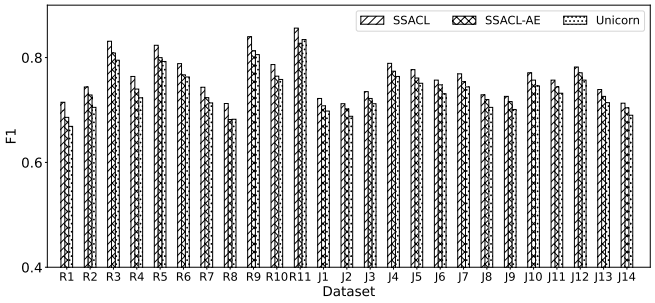
7.1.1 Methods to Compare. For the task of integrable set discovery, we compare the following methods:

- **SSACL** – our proposed method introduced in Sec. 3, which uses both data augmentation and adversarial examples to generate training examples.
- **SSACL-AE** – a variant of SSACL, which only uses data augmentation to generate training examples.
- **SSACL-SSL** – a variant of SSACL, where neither data augmentation nor adversarial examples is used to generate training examples. Instead, it uses supervised training, which is described in Sec. 7.2.3.
- **Unicorn [65]** – this method unifies six data matching tasks and achieves state-of-the-art performance for the task of entity resolution, a task close to our own.

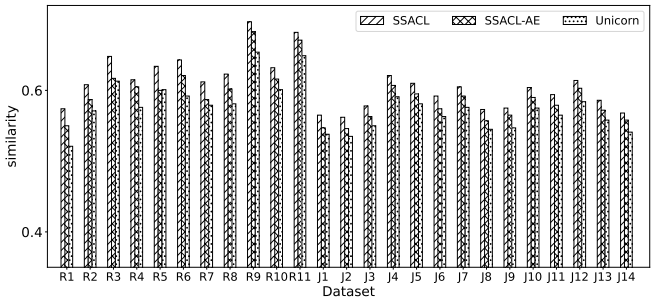
For all of the methods except SSACL-SSL, we use data augmentation methods (described in Sec. 3.2) to generate training data. Specifically, we pretrain models using a subset[2] of the GitTables corpus [29] that contains 1,101 tables. Note that we do not use adversarial examples to generate positive instances for the other methods, since adversarial examples are heavily model-dependent. To improve the efficiency, when judging the integrability of any two tuples in Table T, we employ the blocking techniques described in [19] for every method, which partitions tuples into different groups such that the method only needs to evaluate the integrability for two tuples from the same group.

For the task of multi-tuple conflict resolution, we compare the following methods:

[2] https://zenodo.org/record/5706316

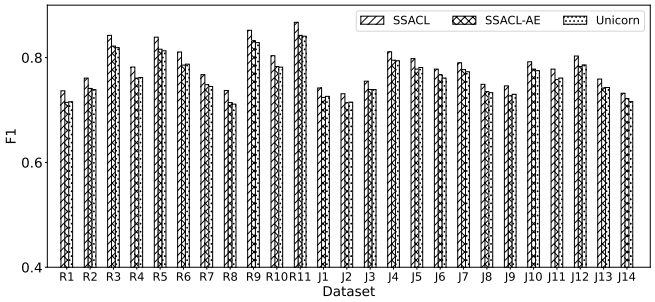


(a) F1

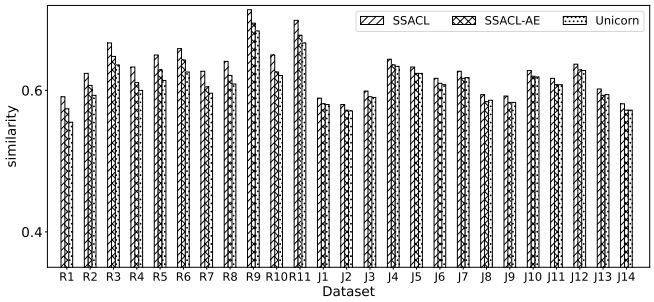


(b) similarity

Figure 3: *F1* and *similarity* in the case of balanced noise (R1-R11 are datasets from *Real* and J1-J14 are datasets from *Join*).

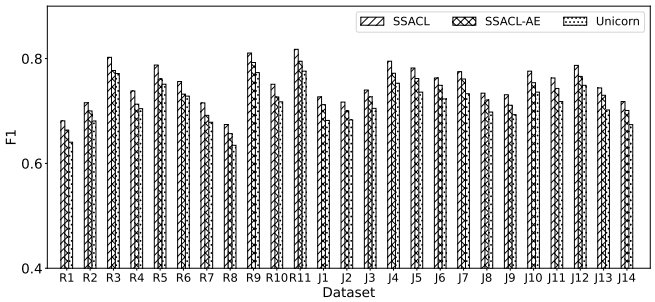


(a) F1

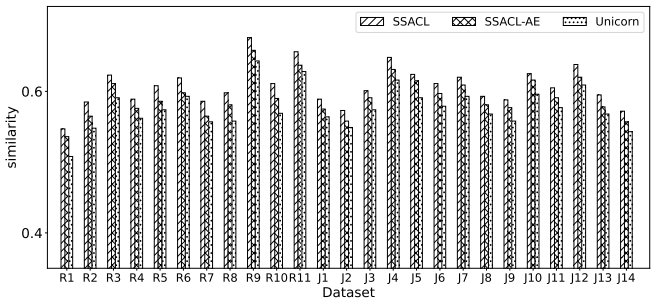


(b) similarity

Figure 4: *F1* and *similarity* in the case of SE-heavy noise (R1-R11 are datasets from *Real* and J1-J14 are datasets from *Join*).



(a) F1



(b) similarity

Figure 5: *F1* and *similarity* in the case of TE-heavey noise (R1-R11 are datasets from *Real* and J1-J14 are datasets from *Join*).

- **ICLCF** – our ICL-based conflict resolution method introduced in Sec. 5. We use the k-NN demonstration example selection strategy by default.
- **Random** – a straightforward baseline that resolves conflicts by uniformly selecting a value from the candidate set at random.
- **Major** – a baseline that resolves conflicts by selecting the most frequent value from the candidate set.
- **SlimFast** [59] – a state-of-the-art truth discovery model for conflict resolution for single truth scenarios.

Table 2: Key Parameters in Our Methods

Method	**SAACL**		**ICLCF**
Parameters	N_{pos}	N_{neg}	k
Range	1-10	3-30	0-50
Default	6	20	35

7.1.2 Environment. We implement all algorithms in Python 3.9 and run the experiments on an Ubuntu sever with an Intel(R) Xeon(R) Platinum 8375C CPU and RTX 4090 GPU. The source code is available at [1].

7.1.3 Model Settings. We use BERT to obtain the pretrained word embeddings and the embedding size is set to 768 by default. When training SSACL, the encoder parameters are fixed, and we only optimize the parameters for matcher during model training. Specifically, we use Adam [36] to optimize the model with a learning rate of 10^{-6} for 30 training epochs. By default, the number of positive instances N_{pos} is set to 6 while the number of negative instances N_{neg} is set to 20. The training time of SSACL is around 68 hours using the default settings. For ICLCF, we employ a LLM called

T5 [57], to make predictions. Furthermore, the number of demonstration examples is set to 35 by default. This method does not require additional training time.

7.2 Evaluation for the Task of Integrable Set Discovery

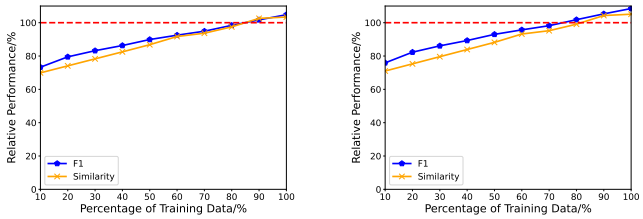
7.2.1 Effectiveness Study. We compare the effectiveness of three methods: SSACL, SSACL-AE, and Unicorn. As introduced in Sec. 6.2, we use *similarity* to evaluate the quality of integrable sets found by each method.

Fig. 3, 4 and 5 present the effectiveness of SSACL and the baselines for cases of balanced noise, SE-heavy noise and typographical error-heavy noise. Due to space limit, we only report the results for the *Real* data pool, and specifically, we report the results of the largest 14 datasets (denoted as $J1 - J14$). For more results, please refer to the technical report [1] since they show similar trends.

From the above figures, we observe the following: First, for all the cases of noise injection, SSACL achieves the highest *similarity* for most datasets, and outperforms Unicorn by a large margin. Specifically, on average, the relative improvement of SSACL over Unicorn in the cases of balanced, SE-heavy, and typographical error-heavy noise are 3.11%, 2.54%, and 5.27%. Second, SSACL-AE also exhibits promising results. In the cases of balanced and TE-heavy noise, it outperforms Unicorn for most datasets, with an average relative improvement of 2.7%, while in the case of TE-heavy noise, it achieves very competitive performance to Unicorn. The above results confirm the effectiveness of our proposed SSACL, and we attribute the improvements to the design of AICJNet, which determines differences in various attributes and uses them and mitigates the impact of missing values.

By checking how these methods behave in the three different types of datasets, we find that as the ratio of typographical errors increases, the *similarity* scores for all of the methods decrease. For example, the average *similarity* of SSACL in the case of balanced noise is 0.594, and SSACL achieves an average *similarity* of 0.613 in the case of SE-heavy noise, meaning there is a relative increase of 3.19%, compared with the average *simialrity* achieved in the case of balanced noises. In contrast, the average *similarity* achieved by SSACL in the case of TE-heavy noise is 0.564, indicating that there is a relative decrease of 5.15% compared with the average *simialrity* achieved in the case of balanced noises, which demonstrates that, compared to semantic equivalence, typographical errors are much more difficult for our methods to resolve. One possible reason is that both Unicorn and our methods use LLMs to produce tuple representations, which capture the semantic equivalence but not necessarily typographical errors.

7.2.2 Effectiveness on pairwise integration condition judgement. We also evaluate the compared methods using the subtask from pairwise integration condition judgement by *F1*, as illustrated in Fig. 3, 4 and 5. The results in terms of *Recall* and *Precision* show similar trend as that of *F1*, and we refer readers to our technical report [1] for additional details. From the experimental results, we observe that SSACL achieves the highest *F1* for most of the datasets, regardless of the type of noise. Specifically, on average, the relative improvement for SSACL over Unicorn in the cases of balanced, SE-heavy, and TE-heavy noise are 3.67%, 2.40%, and 5.54%, respectively. This is reasonable since high-quality integrable sets usually come from accurate pairwise integration condition judgement.



(a) SSACL-SSL vs. SSACL **(b) SSACL-SSL vs. SSACL-AE**

Figure 6: Relative Performance of SSACL-SSL vs. SSACL and SSACL-SSL vs. SSACL-AE

7.2.3 Effectiveness with Limited Labeled Data. Next, we investigate how SSACL performs with limited labeled data by comparing three methods: SSACL, SSACL-AE and SSACL-SSL. SSACL is trained using labeled data. Here, we use the largest dataset $J11$ in *Real* to perform our experiment. To create labeled data, we randomly split the dataset into training data and test data with a ratio 7:3. For the training data, we also use the method proposed in Sec. 6.1 to create the ground-truth. Then, we change the ratio of training data between 10% to 100% and train SSACL-SSL. To ensure that the comparison is fair, we evaluate all of the three methods using the test data. We compare SSACL-SSL with SSACL and SSACL-AE separately to evaluate the relative performance of SSACL-SSL based on the amount of training data available.

Fig. 6 shows the experimental results. Note that the performance of SSACL-SSL increases as the training data increases. When the percentage of training data is between 10%-70%, the performance of SSACL-SSL is worse than SSACL and SSACL-AE. When the training data percentage is 80% or above, we can see that SSACL marginally outperforms SSACL-SSL, and SSACL-AE achieves similar performance to SSACL-SSL. Even with all training data available, SSACL-SSL barely outperforms SSACL and SSACL-AE, with a relative improvement of 3.4% and 5.1% *similarity*, respectively. Note that our model performance may be further improved using additional tables during the pre-training stages. This demonstrates the effectiveness of our proposed method when automatically generating labeled data, which enables our models to achieve comparable performance to a model trained using labeled data. Furthermore, when comparing SSACL with SSACL-AE, note that the former outperforms the latter by 3.5% and 1.6% in terms of *F1* and *similarity*, respectively. This confirms value of introducing dversarial examples to enhancing model training. Overall, the experimental results verify the effectiveness of SSACL using limited labeled data.

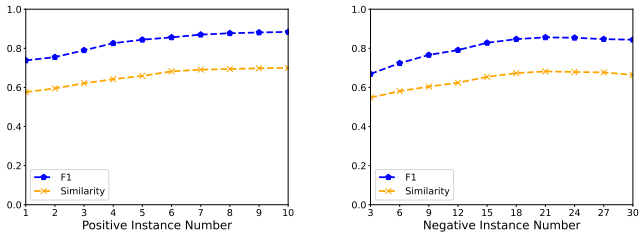
7.2.4 Impact of The Number of Positive and Negative Instances. Finally, we investigate the impact of two key hyperparameters – number of positive instances N_{pos} and number of negative instances N_{neg}, on the model performance of SSACL. We use the $J11$ dataset again to evaluate model quality under varying hyperparameter settings. The experimental results are shown in Fig. 7. Observe that:

As N_{pos} increases, the performance of SSACL consistently improves. This is because the proposed data augmentation methods generate more diverse types of positive instances as more become

available, enabling the model trained to more consistently identify equivalent semantic matehces, and typographical errors. However, when N_{pos} exceeds a certain threshold, namely 6, the model performance improvements are marginal. Since increasing N_{pos} also requires additional model training time, we recommend setting N_{pos} to 6.

In contrast to N_{pos}, whose improvements are easy to see, increasing N_{neg} yields better models only within a range of 3 to 21. Within this range, greater N_{neg} improves the model. However, as N_{neg} exceeds 21, the model performance suddenly begins to deteriorate. This phenomenon can be attributed to the likelihood of including positive instances as negative training data when N_{neg} becomes exceptionally large, thereby reducing model performance. Therefore, we recommend to set N_{neg} to 20.

7.2.5 Efficiency Study. Overall, it takes SSACL and Unicorn 9.3 and 9.8 hours to find the integrable sets on $J11$, respectively, most of which is spent on pairwise integration condition judgement. Since the used blocking technique may bring slightly difference to the total number of pairwise comparison of the two methods, we report their average time of integrability evaluation for each tuple pair, which is 103ms for SSACL and 126ms for Unicorn, respectively. The average inference time of the two methods are similar, since both of them rely on a pre-trained LLM to make predictions.



(a) Positive Instance Number **(b) Negative Instance Number**

Figure 7: The Impact of Positive and Negative Instance Number on *F1* and *similarity*

7.3 Evaluation for Multi-tuple Conflict Resolution

7.3.1 Effectiveness Study. To ensure a fair comparison, we maintain parity for the amount of labeled data available for training in both ICLCF and SlimFast [59]. Specifically, for each dataset, we partition training and test data at the ratio of 7:3. Here, the number of labeled data instances used for training SlimFast is denoted as as N_{train}, and we set the number of demonstration examples of ICLCF $k = N_{train}$. The experimental results are shown in Fig. 8.

Note that our method ICLCF achieves the highest *Accuracy* in most cases when compared against SlimFast. Specifically, the average *Accuracy* of ICLCF across the 25 datasets is 0.766, surpassing SlimFast, at 0.634, by a margin of 20.8%. Furthermore, both Random and Major perform poorly using these datasets, as they rely on heuristics for candidate value selection. This highlights the effectiveness of our ICLCF method, which can be attributed to the fact that ICLCF fully leverages the knowledge embedded within LLMs when combined with in-context learning.

7.3.2 Effectiveness with Limited Labeled Data. In this experiment, we study if ICLCF can maintain competitive performance even when trained using a constrained number of demonstration examples. To test this, we use a version of ICLCF (in Sec. 7.3.1) as a baseline and test the effects of the parameter k – the number of demonstration examples –from 0 to 50. We also evaluate using the dataset J_{11}, and the results are presented in Fig. 9.

When $k = 0$, SSACL does not rely on any demonstration examples to make predictions. However, it still manages an accuracy of 0.525. This outcome is encouraging, considering that it is only 10.2% lower than the accuracy of SlimFast, and requires no labeled training data. As k is increased, a consistent improvement in model performance can be observerd, as the increased number of demonstration examples guides SSACL decisions, enhancing the prediction accuracy. When $k = 35$, SSACL has achieves a 97.2% accuracy when compared to SSACL with fully labeled data (Sec. 7.3.1). This demonstrates the effectiveness of ICLCF in a scenario where labeled data is limited.

7.3.3 Impact of Demonstration Example Selection Strategies. We also compare two different demonstration example selection strategies, k-NN and random, in ICLCF to investigate the impact on model performance. As shown in Fig. 10, ICLCF with a k-NN selection strategy marginally outperforms ICLCF which uses a random selection strategy, for most cases, with an average relative improvement of 1.55%. The main reason is that k-NN selection strategy can identify the most similar demonstration examples for a query, which guides ICLCF to make better prediction for conflict attribute values.

7.3.4 Efficiency Study. On average, it takes ICLCF and SlimFast 35ms and 157ms to make the prediction for each integrable set in dataset $J11$. Our method is more costly for two reasons: 1) ICLCF relies on an LLM to make predictions, which has higher model complexity than SlimFast; 2) our model is based on in-context learning, which requires the model to input each text test case as well as any demonstration examples. Thus a larger input size introduces higher time costs. However, considering that the relative performance improvement for ICLCF over SlimFast is high, as illustrated in Sec. 7.3.1, the additional time cost is acceptable.

8 RELATED WORK

We first describe two operators complementation and subsumption used in data integration, and then we present related literatures on entity resolution and data fusion.

Complementation and Subsumption. Complementation and subsumption [15, 16, 31, 53, 58] are two operators used to integrate tuples for a given table. Subsumption was first introduced by [26] when implementing full disjunction (FD), an associative version of an outer join, and complementation was first introduced by [10] as an operator in data fusion. Both operators involve iterative pairwise comparisons between two tuples. Prior studies mainly focus on enhancing efficiency. For instance, the most recent work ALITE Khatiwada et al. [34] explored how to integrate data lake tables using complementation and subsumption, with notable efficiency gains using blocking techniques. However, as complementation and subsumption have very restrictive requirements for integration condition judgement and they only integrate two tuples

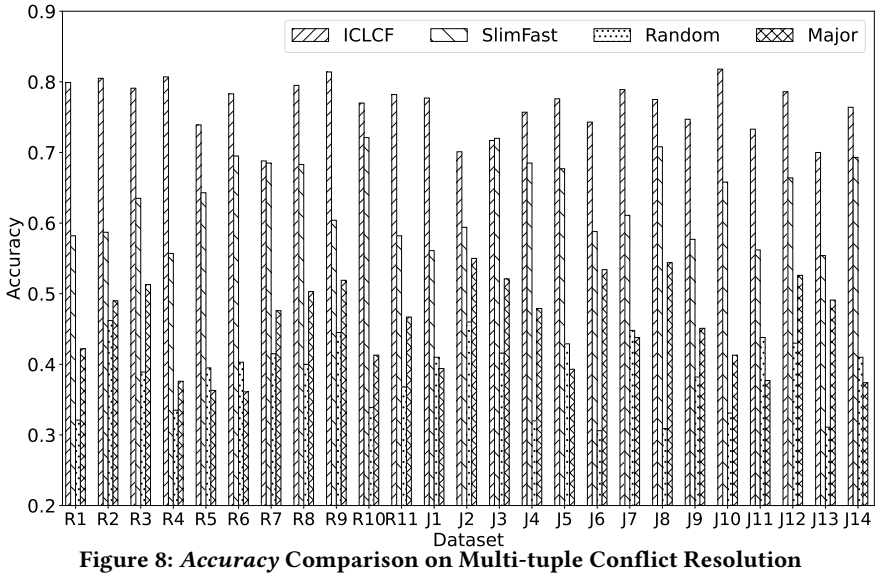


Figure 8: *Accuracy* Comparison on Multi-tuple Conflict Resolution

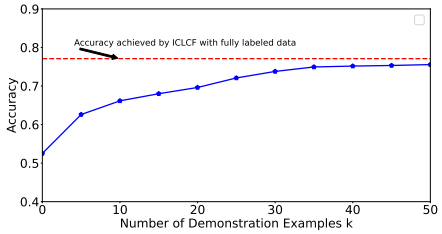


Figure 9: Impact of Number of Demonstration Examples on *Accuracy of ICLCF*

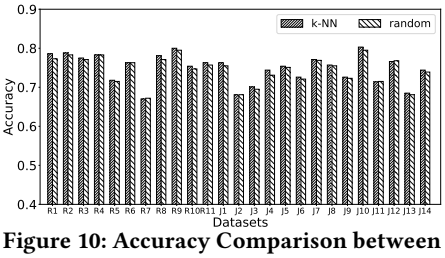


Figure 10: Accuracy Comparison between Two Selection Strategies

at one step, they can not handle typographical errors as well as conflicts.

Entity Resolution. The most relevant work to pairwise integration condition judgement is entity resolution, which determines whether two entities with different descriptions refer to the same entity. Existing methods can generally be divided into four categories: rule-based methods [24, 62], crowdsourcing methods [13, 14, 17, 23, 69], conventional machine learning (ML)-based methods [8, 43, 63], and deep learning (DL)-based methods [19, 41, 42, 48, 64, 65]. Unsurprisingly, DL-based methods often exhibit the best overall performance. In particular, DeepER [19] uses Long Short-Term Memory (LSTM), a variant of recurrent neural networks, to learn tuple representations using pre-trained word embeddings such as GloVe [55]. Then, a neural network is used to evaluate the similarity between two tuple representations. DeepMatcher [48] categorizes ER problems into three types –structured ER, textual ER, and dirty ER – and provides a comprehensive and systematic design space for ER solutions. Ditto [41] uses pre-trained large language models, such as BERT [32] and GPT-2 [56], to create an ER model. Additional techniques are also explored, such as leveraging domain knowledge and data augmentation, to enhance the performance of Ditto. DADER [64] systematically investigates the problem of domain adaption for entity solutions, addressing scenarios where an ER model trained from a source dataset is applied to a target dataset using minimal or no labeled data. Unicorn [65] introduces a unified framework for entity resolution and five other data matching tasks, and also propose a multi-learning framework that enables tasks to benefit from each other.

While entity resolution is similar to integration condition judgments, two key distinctions set them apart: (1) Entities involved in entity resolution typically come from comprehensive information, while the tuples in our task often contain missing values, which is common in large data lakes combined with the outer union operator. This disparity demands a more delicate comparison between the attribute-level representations of two tuples, rather than a simple tuple-level assessment. (2) The majority of entity resolution methods are constructed using supervised learning; in contrast, in the data lake scenarios, labeled data for our task is rarely available.

Data Fusion. Data fusion [9, 44] studies how to merge multiple records, which map to the same entity but originate from different resources, into a unified representation. At the core of data fusion lies the challenge of addressing conflicts when different values are present for a particular attribute. Although some simple yet intuitive methods, such as using the mean or median values for numerical values or using majority voting for categorical values can be directly employed, they often reduce overall accuracy. Existing solutions [6, 18, 38, 59, 67, 71] for data fusion predominantly rely on the concept of truth discovery, which involves assessing the reliability of each data source and selecting the value from the most reliable source. However, the estimation of source reliability in truth discovery methods heavily relies on labeled data or metadata (such as citation of papers). Unfortunately, such data is usually absent in data lake environments. Thus, although the goal of truth discovery is similar to our own, the methodologies are not applicable to solve our problem since the necessary data is not available.

9 CONCLUSION

In this paper, we aim to solve two core tasks in the data integration of data lake tables, integrable set discovery and multi-tuple conflict resolution. To solve the task of integrable set discovery, we aim to develop a binary classifier, which is able to judge whether tuples should be integrated even when they contain semantic equivalence and typographical errors. Then we cast the problem of integrable set discovery as finding connected components in a graph, and employ a DFS algorithm to address it. For the task of multi-tuple conflict resolution, we propose a novel in-context learning (ICL)-based methods, which enable us to leverage extensive knowledge embedded in the pre-trained large language model to make prediction. Notably, our methods can achieve very promising performance

even with limited labeled data. In future we would like to study how to further boost the efficiency of our methods.

REFERENCES

[1] [n. d.]. Technical Report and Source Code. https://github.com/jdm199605/Robin.

[2] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting data errors: Where are we and what needs to be done? *Proceedings of the VLDB Endowment* 9, 12 (2016), 993–1004.

[3] Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. 2022. What learning algorithm is in-context learning? Investigations with linear models. In *The Eleventh International Conference on Learning Representations*.

[4] Mohammadreza Armandpour, Patrick Ding, Jianhua Huang, and Xia Hu. 2019. Robust negative sampling for network embedding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 3191–3198.

[5] Pranjal Awasthi, Nishanth Dikkala, and Pritish Kamath. 2022. Do more negative samples necessarily hurt in contrastive learning?. In *International Conference on Machine Learning*. PMLR, 1101–1116.

[6] Fabio Azzalini, Davide Piantella, Emanuele Rabosio, and Letizia Tanca. 2023. Enhancing domain-aware multi-truth data fusion using copy-based source authority and value similarity. *The VLDB Journal* 32, 3 (2023), 475–500.

[7] Markus Bayer, Marc-André Kaufhold, and Christian Reuter. 2022. A survey on data augmentation for text classification. *Comput. Surveys* 55, 7 (2022), 1–39.

[8] Mikhail Bilenko and Raymond J Mooney. 2003. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. 39–48.

[9] Jens Bleiholder and Felix Naumann. 2009. Data fusion. *ACM computing surveys (CSUR)* 41, 1 (2009), 1–41.

[10] Jens Bleiholder, Sascha Szott, Melanie Herschel, Frank Kaufer, and Felix Naumann. 2010. Subsumption and complementation as data fusion operators. In *Proceedings of the 13th International Conference on Extending Database Technology*. 513–524.

[11] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the association for computational linguistics* 5 (2017), 135–146.

[12] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press, Cambridge, UK.

[13] Chengliang Chai, Guoliang Li, Jian Li, Dong Deng, and Jianhua Feng. 2016. Cost-effective crowdsourced entity resolution: A partial-order approach. In *Proceedings of the 2016 International Conference on Management of Data*. 969–984.

[14] Chengliang Chai, Guoliang Li, Jian Li, Dong Deng, and Jianhua Feng. 2018. A partial-order-based framework for cost-effective crowdsourced entity resolution. *The VLDB Journal* 27 (2018), 745–770.

[15] Sara Cohen, Itzhak Fadida, Yaron Kanza, Benny Kimelfeld, and Yehoshua Sagiv. 2006. Full disjunctions: Polynomial-delay iterators in action. In *Proceedings of the 32nd international conference on Very large data bases*. Citeseer, 739–750.

[16] Sara Cohen and Yehoshua Sagiv. 2005. An incremental algorithm for computing ranked full disjunctions. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 98–107.

[17] Lizhen Cui, Jing Chen, Wei He, Hui Li, Wei Guo, and Zhiyuan Su. 2021. Achieving approximate global optimization of truth inference for crowdsourcing microtasks. *Data Science and Engineering* 6, 3 (2021), 294–309.

[18] Xin Luna Dong and Felix Naumann. 2009. Data fusion: resolving data conflicts for integration. *Proceedings of the VLDB Endowment* 2, 2 (2009), 1654–1655.

[19] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed representations of tuples for entity resolution. *Proceedings of the VLDB Endowment* 11, 11 (2018), 1454–1467.

[20] Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding Back-Translation at Scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 489–500.

[21] Grace Fan, Jin Wang, Yuliang Li, and Renée J Miller. 2023. Table Discovery in Data Lakes: State-of-the-art and Future Directions. In *Companion of the 2023 International Conference on Management of Data*. 69–75.

[22] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée Miller. 2022. Semantics-aware Dataset Discovery from Data Lakes with Contextualized Column-based Representation Learning. *arXiv preprint arXiv:2210.01922* (2022).

[23] Ju Fan, Guoliang Li, Beng Chin Ooi, Kian-lee Tan, and Jianhua Feng. 2015. icrowd: An adaptive crowdsourcing framework. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 1015–1030.

[24] Wenfei Fan, Xibei Jia, Jianzhong Li, and Shuai Ma. 2009. Reasoning about record matching rules. *Proceedings of the VLDB Endowment* 2, 1 (2009), 407–418.

[25] Steven Y Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. 2021. A Survey of Data Augmentation Approaches for NLP. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. 968–988.

[26] César A Galindo-Legaria. 1994. Outerjoins as disjunctions. In *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*. 348–358.

[27] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[28] Rihan Hai, Sandra Geisler, and Christoph Quix. 2016. Constance: An intelligent data lake system. In *Proceedings of the 2016 international conference on management of data*. 2097–2100.

[29] Madelon Hulsebos, Çagatay Demiralp, and Paul Groth. 2023. Gittables: A large-scale corpus of relational tables. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–17.

[30] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. 2019. Adversarial examples are not bugs, they are features. *Advances in neural information processing systems* 32 (2019).

[31] Yaron Kanza and Yehoshua Sagiv. 2003. Computing full disjunctions. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 78–89.

[32] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT*. 4171–4186.

[33] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J Miller, and Mirek Riedewald. 2023. SANTOS: Relationship-based Semantic Table Union Search. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–25.

[34] Aamod Khatiwada, Roee Shraga, Wolfgang Gatterbauer, and Renée J Miller. 2022. Integrating Data Lake Tables. *Proceedings of the VLDB Endowment* 16, 4 (2022), 932–945.

[35] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised contrastive learning. *Advances in neural information processing systems* 33 (2020), 18661–18673.

[36] DP Kingma. 2014. Adam: a method for stochastic optimization. In *Int Conf Learn Represent*.

[37] Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.

[38] Qi Li, Yaliang Li, Jing Gao, Lu Su, Bo Zhao, Murat Demirbas, Wei Fan, and Jiawei Han. 2014. A confidence-aware approach for truth discovery on long-tail data. *Proceedings of the VLDB Endowment* 8, 4 (2014), 425–436.

[39] Xian Li, Xin Luna Dong, Kenneth Lyons, Weiyi Meng, and Divesh Srivastava. 2012. Truth Finding on the Deep Web: Is the Problem Solved? *Proceedings of the VLDB Endowment* 6, 2 (2012).

[40] Xian Li, Xin Luna Dong, Kenneth Lyons, Weiyi Meng, and Divesh Srivastava. 2012. Truth Finding on the Deep Web: Is the Problem Solved? *Proceedings of the VLDB Endowment* 6, 2 (2012).

[41] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep entity matching with pre-trained language models. *Proceedings of the VLDB Endowment* 14, 1 (2020), 50–60.

[42] Yuliang Li, Jinfeng Li, Yoshi Suhara, AnHai Doan, and Wang-Chiew Tan. 2023. Effective entity matching with transformers. *The VLDB Journal* (2023), 1–21.

[43] Andrew McCallum and Ben Wellner. 2004. Conditional models of identity uncertainty with application to noun coreference. *Advances in neural information processing systems* 17 (2004).

[44] Tong Meng, Xuyang Jing, Zheng Yan, and Witold Pedrycz. 2020. A survey on machine learning for data fusion. *Information Fusion* 57 (2020), 115–129.

[45] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* 26 (2013).

[46] George A Miller. 1995. WordNet: a lexical database for English. *Commun. ACM* 38, 11 (1995), 39–41.

[47] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. 11048–11064.

[48] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data*. 19–34.

[49] Fatemeh Nargesian, Erkang Zhu, Renée J Miller, Ken Q Pu, and Patricia C Arocena. 2019. Data lake management: challenges and opportunities. *Proceedings of the VLDB Endowment* 12, 12 (2019), 1986–1989.

[50] Fatemeh Nargesian, Erkang Zhu, Ken Q Pu, and Renée J Miller. 2018. Table union search on open data. *Proceedings of the VLDB Endowment* 11, 7 (2018), 813–825.

[51] Fatemeh Nargesian, Erkang Zhu, Ken Q Pu, and Renée J Miller. 2018. Table union search on open data. *Proceedings of the VLDB Endowment* 11, 7 (2018), 813–825.

[52] Jorge Nocedal and Stephen J. Wright. 2006. *Numerical Optimization* (2e ed.). Springer, New York, NY, USA.

[53] Matteo Paganelli, Domenico Beneventano, Francesco Guerra, and Paolo Sottovia. 2019. Parallelizing computations of full disjunctions. *Big Data Research* 17 (2019), 18–31.

[54] Jinfeng Peng, Derong Shen, Nan Tang, Tieying Liu, Yue Kou, Tiezheng Nie, Hang Cui, and Ge Yu. 2022. Self-supervised and Interpretable Data Cleaning with

Sequence Generative Adversarial Networks. *Proceedings of the VLDB Endowment* 16, 3 (2022), 433–446.

[55] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[56] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[57] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* 21, 1 (2020), 5485–5551.

[58] Anand Rajaraman and Jeffrey D Ullman. 1996. Integrating information by outerjoins and full disjunctions. In *Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. 238–248.

[59] Theodoros Rekatsinas, Manas Joglekar, Hector Garcia-Molina, Aditya Parameswaran, and Christopher Ré. 2017. Slimfast: Guaranteed results for data fusion and source reliability. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1399–1414.

[60] Connor Shorten, Taghi M Khoshgoftaar, and Borko Furht. 2021. Text data augmentation for deep learning. *Journal of big Data* 8 (2021), 1–34.

[61] Connor Shorten, Taghi M Khoshgoftaar, and Borko Furht. 2021. Text data augmentation for deep learning. *Journal of big Data* 8 (2021), 1–34.

[62] Rohit Singh, Vamsi Meduri, Ahmed Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, and Nan Tang. 2017. Generating concise entity matching rules. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1635–1638.

[63] Sheila Tejada, Craig A Knoblock, and Steven Minton. 2002. Learning domain-independent string transformation weights for high accuracy object identification. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. 350–359.

[64] Jianhong Tu, Ju Fan, Nan Tang, Peng Wang, Chengliang Chai, Guoliang Li, Ruixue Fan, and Xiaoyong Du. 2022. Domain adaptation for deep entity resolution. In *Proceedings of the 2022 International Conference on Management of Data*. 443–457.

[65] Jianhong Tu, Ju Fan, Nan Tang, Peng Wang, Guoliang Li, Xiaoyong Du, Xiaofeng Jia, and Song Gao. 2023. Unicorn: A unified multi-tasking model for supporting matching tasks in data integration. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.

[66] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[67] Houping Xiao and Shiyu Wang. 2022. A Joint Maximum Likelihood Estimation Framework for Truth Discovery: A Unified Perspective. *IEEE Transactions on Knowledge and Data Engineering* (2022).

[68] Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. 2021. An Explanation of In-context Learning as Implicit Bayesian Inference. In *International Conference on Learning Representations*.

[69] Jingru Yang, Ju Fan, Zhewei Wei, Guoliang Li, Tongyu Liu, and Xiaoyong Du. 2018. Cost-effective data annotation using game-based crowdsourcing. *Proceedings of the VLDB Endowment* 12, 1 (2018), 57–70.

[70] Jiliang Zhang and Chen Li. 2019. Adversarial examples: Opportunities and challenges. *IEEE transactions on neural networks and learning systems* 31, 7 (2019), 2578–2593.

[71] Shi Zhi, Bo Zhao, Wenzhu Tong, Jing Gao, Dian Yu, Heng Ji, and Jiawei Han. 2015. Modeling truth existence in truth discovery. In *Proceedings of the 21th acm sigkdd international conference on knowledge discovery and data mining*. 1543–1552.

[72] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J Miller. 2019. Josie: Overlap set similarity search for finding joinable tables in data lakes. In *Proceedings of the 2019 International Conference on Management of Data*. 847–864.

A TYPOGRAPHICAL ERRORS

Table illustrates a series of ways to simulating the typographical errors in real world. In this paper, we use it from two aspects: 1) In the data generator of SAACL, we use it as a kind of character-level perturbations; 2) In the benchmark creation, we use it to inject typographical errors into the original data.

Table 3: Typographical Errors for Numerical Attributes

Error type	Description
char swap	Swaps two random consecutive word characters in the text.
missing char	Skips a random word character in the text.
extra char	Adds an extra, keyboard-neighbor, letter next to a random word character.
nearby char	Replaces a random word character with keyboard-neighbor letter.
similar char	Replaces a random word character with another visually similar characte
skipped space	Skips a random space from the text.
random space	Adds a random space in the text.
repeated char	Repeats a random word character.
unichar	Replaces a random consecutive repeated letter with a single letter.

Table 4: Typographical Errors for Numerical Attributes

Error type	Description
Method	Description
digit swap	Swaps two random consecutive digits in the number.
missing digit	Skips a random digit in the number.
extra digit	Adds an extra, keyboard-neighbor, digit next to a random digit in the number.
nearby digit	Replaces a random digit in the number with a keyboard-neighbor digit.
similar digit	Replaces a random digit with another visually similar digit.
repeated digit	Repeats a random digit in the number.
unidigit	Replaces a random consecutive repeated digit with a single digit.
unichar	Replaces a random consecutive repeated letter with a single letter.

B DFS ALGORITHM FOR FINDING CONNECTED COMPONENTS IN A GRAPH

Algorithm 2 the DFS algorithm to solve the problem of finding connected components in a graph. As a result, each connected component corresponds to an integrable sets.

Algorithm 2: DFS for Finding Connected Components

Data: Graph G
Result: List of connected components

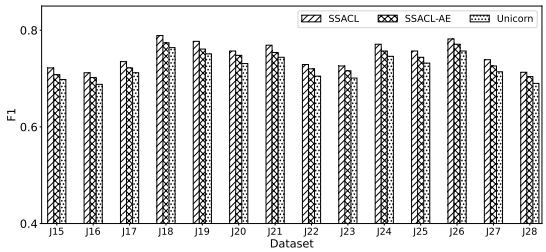
```
Function DFS(vertex v, list component):
    Mark v as visited;
    Add v to component;
    foreach neighbor u of v do
        if u is not visited then
            DFS(u, component);
        end
    end
end
Function FindConnectedComponents(Graph G):
    Initialize an empty list connectedComponentsList;
    foreach vertex v in G do
        if v is not visited then
            Create an empty list component;
            DFS(v, component);
            Add component to connectedComponentsList;
        end
    end
    return connectedComponentsList;
end
```

C EXPERIMENT RESULTS

C.0.1 Effectivenss Comparison on the task of Integrable Set Discovery. Fig. 11, 12, and 13 illustrates the *similarity* scores achieved by the compared methods on the datasets of $J15 - J28$. The experimental results show the similar trends. SSACL also achieved the best performance on all the cases of noise injection, outperforming Unicorn by a largin margin. On average, in the all the cases of noise injection, the relative improvement of SSACL over Unicorn is 3.83% on the 14 datasets.

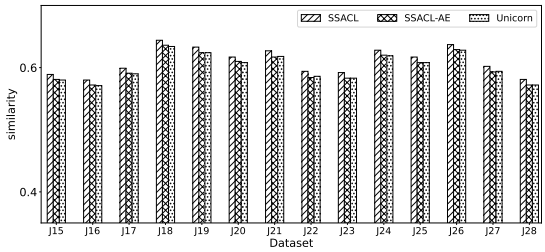
C.0.2 Effectiveness Comparison on the task of pairwise integration condition judgement. Fig. 11, 12, and 13 illustrates the *F1* scores achieved by the compared methods on the datasets of $J15 - J28$. On average, in all the cases of noise injection, the relative improvement of SSACL over Unicorn in terms of *F1* is 4.62%.

(a) F1

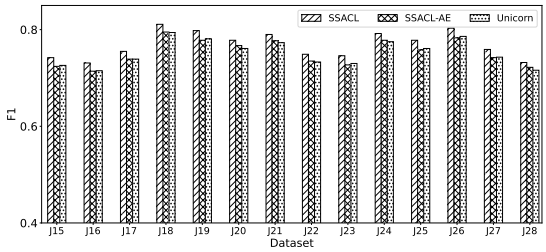


(b) similarity

Figure 11: *F1* and *similarity* in the case of SE-heavy noise on the datasets of $J15 - J28$.

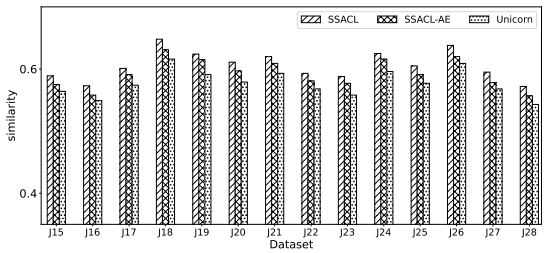


(a) F1

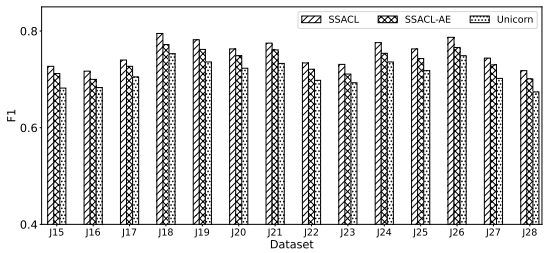


(b) similarity

Figure 12: *F1* and *similarity* in the case of balanced noise on the datasets of $J15 - J28$.



(a) F1



(b) similarity

Figure 13: *F1* and *similarity* in the case of TE-heavy noise on the datasets of $J15 - J28$.