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

ABSTRACT

In this paper, we study two core tasks related to the integration of tables in a data lake. The first task, *integrable set discovery*, focuses on finding all sets of integrable items, each of which is comprised of a set of tuples that can be safely integrated, taking into account any occurrences of semantic equivalence or typographical errors. The second task, *multi-tuple conflict resolution*, resolves conflicts between multiple tuples being integrated. We train a binary classifier to determine if two tuples should be integrated, when containing semantic equivalence or typographical errors. Given the scarcity of labeled data in data lakes, we present a self-supervised adversarial contrastive learning algorithm to do classification, which incorporates data augmentation methods and adversarial examples to autonomously generate new training data. Following the classification of integrability of any two tuples using our classifier, we address the Integrable Set Discovery task by framing it as the equivalent problem of finding connected components in a graph. We solve this problem using a depth-first search algorithm. Next, for the task of *multi-tuple conflict resolution*, we propose an in-context learning method, which leverages extensive knowledge embedded in pretrained large language models to resolve any conflicts that arise when integrating multiple tuples using minimal labeled data. Lacking suitable benchmarks for our tasks, we create our own benchmarks utilizing two real-word data pools: Real and Join, both publicly accessible. Extensive experiments are conducted to verify the effectiveness of our proposed methods comparing against state-of-the-art approaches on the two tasks.

ACM Reference Format:
. 2018. Robust Table Integration in Data Lakes: From Integrable Set Discovery to Multi-Tuple Conflict Resolution. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 18 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Data lakes are large repositories that can store various types of raw data [28, 47]. Recently, there has been a growing interest in performing table search algorithms [21, 22, 32, 48, 49, 73], aiming to find unionable, joinable or similar tables in large data lakes. The integration of data lake tables into a unified more comprehensive structure has the potential to create new knowledge and insights that would be otherwise inaccessible from using the tables in isolation. However, this poses significant challenges since the tables usually come from a variety of different resources. To enhance the process of table integration in data lakes, three core tasks must be resolved: 1) *Schema Alignment*, which aligns attributes across multiple tables; 2) *Integrable Set Discovery*, which identifies all integrable sets, each containing a set of integrable tuples; 3) *Multi-tuple Conflict Resolution*, which resolves conflicts that may arise during the integration process.

Most recently, a pioneering work ALITE [33] 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 sequential two tasks. However, while ALITE lays a solid foundation for integrable set discovery and multi-tuple conflict resolution, it does not address two critical yet commonly encountered cases in practice: 1) semantic equivalence and typographical errors, and 2) conflicts, which will be introduced in details as follows.

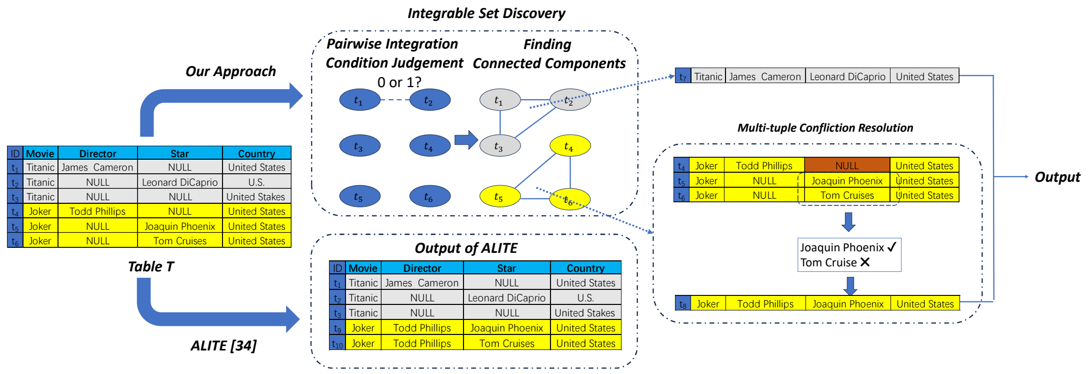
Case 1: Semantic Equivalence and Typographical Errors. The tuples marked in grey in Fig. 1 are examples of semantic equivalence and typographical errors. Consider the two tuples t_1 and t_2 in Table T, whose values for the attribute **Country**, namely "United States" and "U.S.", are semantically equivalent, since they refer to the same entity using different terms. Ideally, t_1 and t_2 can be integrable. Furthermore, the value for the tuple t_3 on the attribute **Country**, "United Stakes", is a typographical error of t_1. Ideally, t_1 and t_3 are also integrable. However, ALITE does not address semantic equivalence or typographical errors, and consequently, it cannot integrate these two cases effectively.

Case 2: Conflicts are also a key issue in multi-tuple integration and must be appropriately resolved during the integration process. As illustrated in Fig. 1, the 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** for t_4 using the corresponding attribute values from t_5 or t_6, there are two different possibilities: "Joaquin Phoenix" and "Tom Cruise", which lead to 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 does not support conflict resolution. Hence, it integrates both (t_4, t_5) and (t_5, t_6) as new tuples, resulting in t_9 and t_{10}, respectively. This leads to incomplete information in the new table, which can potentially have negative effects on downstream tasks.

To overcome these limitations, we propose a new approach for multi-tuple integration. Our approach primarily focuses on two tasks: *integrable set discovery* and *multi-tuple conflict resolution*. More importantly, we achieve it using a limited amount of labeled data, as illustrated at the top of Fig. 1.

In particular, to resolve 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 each tuple pair in table T, even

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX



ID	Movie	Director	Star	Country
t_1	Titanic	James Cameron	NULL	United States
t_2	Titanic	NULL	Leonard DiCaprio	U.S.
t_3	Titanic	NULL	NULL	United Stakes
t_4	Joker	Todd Phillips	NULL	United States
t_5	Joker	NULL	Joaquin Phoenix	United States
t_6	Joker	NULL	Tom Cruises	United States

ID	Movie	Director	Star	Country
t_1	Titanic	James Cameron	NULL	United States
t_2	Titanic	NULL	Leonard DiCaprio	U.S.
t_3	Titanic	NULL	NULL	United Stakes
t_9	Joker	Todd Phillips	Joaquin Phoenix	United States
t_{10}	Joker	Todd Phillips	Tom Cruises	United States

t_7	Titanic	James Cameron	Leonard DiCaprio	United States

t_4	Joker	Todd Phillips	NULL	United States
t_5	Joker	NULL	Joaquin Phoenix	United States
t_6	Joker	NULL	Tom Cruises	United States

t_8	Joker	Todd Phillips	Joaquin Phoenix	United States

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

when semantic equivalence and typographical errors exist. The solution can be framed as a binary classification problem, where a binary classifier is expected to determine if the tuple can be integrated. In the second stage, based on the integrability classification made by the classifier for any two tuples, our objective is to find all integrable sets within T. This is equivalent to the problem of finding connected components in a graph. The latter problem can be resolved using a DFS-based algorithm, and the former problem is more complex. This is primarily due to the scarcity of labeled data, which can hinder the creation of a reliable binary classifier.

To address this issue, for each tuple t, we regard the semantically equivalent or typographically errors 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, alleviating the challenge of limited labeled data.

Next, to solve *multi-tuple conflict resolution*, we depart from the conventional conflict resolution approaches for data fusion that require extensive labeled data collections. Instead, we propose a novel in-context learning (ICL)-based method. This method leverages extensive knowledge contained in pre-trained large language models (LLMs), which effectively reduces the reliance on large labeled data in the model, while maintaining comparable performance level. Specifically, our ICL-based approach proceeds in three steps – (1) *Instruction Template Construction*: we create an instruction template designed to transform the conflict resolution task into a natural language query. (2) *Demonstration Example Selection*: a small number of tuples are carefully selected and labeled as demonstration examples. Remarkably, even a few labeled examples can provide sufficient contextual information to guide the LLM to reliably resolve conflicts. (3) *Conflict Resolution*: armed with the demonstration examples and a natural language query, we use a large language model, such as T5 [54], to score candidate resolution and select the one with the highest score.

In summary, the above approach exhibits the potential to achieve more robust table integration in 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, SSACL, to train a binary classifier using limited labeled data, to predict pairwise integrability for a tuple pair (Sec. 3). Leveraging the predictions made by this classifier, we identify the integrable sets by detecting connected components in a graph, a problem that can be efficiently solved using a DFS algorithm (Sec. 4).
- To solve the task of multi-tuple conflict resolution, we develop an in-context learning-based [3, 45, 68] method, namely In-context Learning for Conflict Resolution (ICFCL), which demonstrates promising performance using limited labeled data (Sec. 5).
- Since no suitable benchmarks exist for evaluating our problem, we have taken the initiative to create our own benchmarks and make it public [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 ICLCF achieves a relative improvement of 20.8% in accuracy. Furthermore, both SSACL and ICLCF only experience a decrease in performance of less than 5%, using limited labeled data (Sec. 7).

2 PROBLEM FORMULATION

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

Building on the state-of-the-art work [33], we assume the schemas of the input tables have been aligned, and they have been integrated into an intermediate table T. Using T as input, *integrable set discovery* aims to find all integrable sets $\mathcal{S}$, each of which encompasses a collection of tuples awaiting integration. For each integrable set $S \in \mathcal{S}$, we aim to integrate all tuples inside into a new tuple t_{new} using *multi-tuple conflict resolution.* Before providing the problem formulation of the first task, a fundamental operator is required 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 semantically equivalent or typographical errors, a function $f(t_i, t_j)$ produces an output, a* 1 *or* 0*, which indicates whether t_i and t_j are integrable or not.*

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

Definition 2. ***Integrable Set Discovery****. Given a table T, an integration condition judgement function f, multiple disjoint integrable* *sets are produced, denoted as $\mathcal{S} = \{S_1, S_2, ..., S_{|\mathcal{S}|}\}$. Each integrable set S ($S \in \mathcal{S}$) is comprised of 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 according to Definition 2, all tuples in S are regarded as integrable into a single tuple, and denoted as t_{new}. This integration process involves filling each attribute of t_{new} with a correct value. However, when a certain attribute t_{new} has multiple distinct values originating from different tuples in S, we call a conflict, and formalize the problem of *multi-tuple conflict resolution* to address it.

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

3 PAIRWISE INTEGRATION JUDGEMENT CONDITION

3.1 Key Idea

From Definition 1, it is clear that the pairwise integration condition judgement function f can be a binary classifier, which outputs either 1 or a 0 to indicate if two tuples can be integrated. Thus, our goal is to derive an integration condition judgement function f using a machine learning model. For simplicity, we use f to denote the integration condition judgement function and the binary classifier interchangeably. However, a significant hurdle exists due to the unavailability of a labeled data in data lake environments, and manually labeling data could be prohibitively expensive, as discussed in Sec. 1.

To overcome this hurdle, we propose a novel approach – *self-supervised adversarial contrastive learning* (SSACL), which is designed to automatically generate training data for both positive and negative instances. For a given tuple t, although the negative case (i.e., tuples that cannot be integrated using t) can be obtained using negative sampling [4, 5], the key challenge is *how to generate positive instances (i.e., tuples that can be integrated with t)*, particularly tuples that are semantically equivalent to tuple t or exhibit typographical errors. To address this challenge, 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, 57] and *adversarial examples* [27, 29, 71], which will be described in Sec. 3.2 and Sec. 3.5, respectively.

Naturally, for a tuple pair (t, t^+), $\{(\boldsymbol{emb}(t), \boldsymbol{emb}(t^+)) = 1$ should hold. Once the model f has undergone sufficient training using positive instances (t, t^+),accurately inferring the integrability of two tuples is possible, even when the tuples are semantically equivalent or contain typographical errors. Furthermore, the binary classifier is trained using the contrastive learning framework, with the objective of producing embeddings where positive tuple pairs are closer together in the embedding space and negative tuple pairs are farther apart.

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

- The *Data Generator* introduces data augmentation and negative sampling 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* aims to match two tuples, which includes embeddings from the *encoder* as input, and outputs 1 or 0 to determine if they are integrable or not (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* (Sec. 3.5).

3.2 Data Generator

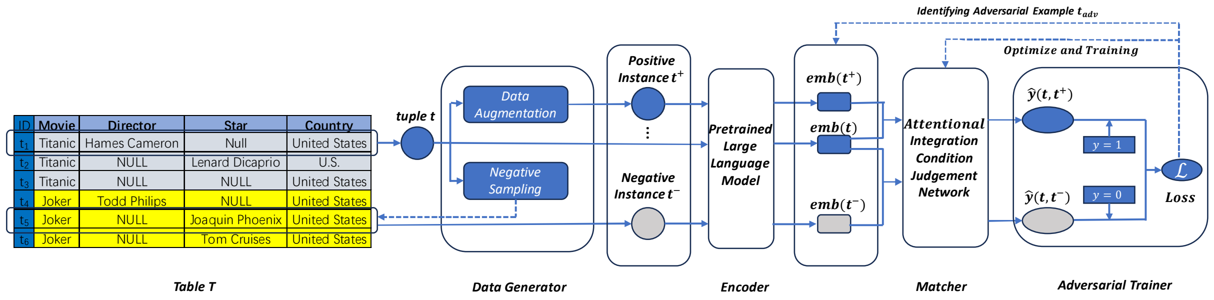
A data generator automatically generates training data for integrable set discovery. Specifically, for each tuple t, data is augmented to produce a collection of perturbed tuples, $\mathcal{P}_t = \{t_i^+ | t_i^+ = p_i(t)\}$, where p_i corresponds to a specific perturbation function. Consequently, every tuple pair (t, t_i^+) is a positive training instance for the matcher. Obviously, deciding how to create the slight perturbations p_i impacts the quality of the training data, which in turn affects the performance of the training model f. So, the selection of perturbation functions should be comprehensive. Perturbation functions are carefully selected to adhere to two key principles [25, 58]: (1) The perturbations should preserve the semantics of the original tuple t or make minimal changes; (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 an attribute value a of a tuple t, and create a new tuple t^+ by removing its attribute value, $t^+[a] == NULL$. In all cases, a is expected to be non-null.
- *Attribute substitution.* This is applied to text attributes only. We randomly select an attribute A of a tuple t, and create a new tuple t^+ by using backtranslation methods [20, 58], which generate a semantically equivalent description for $t[a]$, and use this new description to replace the original value $t[a]$. Backtranslation is a widely used data augmentation method in NLP to generate a sentence with similar semantics to the original sentence. The method first translates the 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 attribute value, we randomly select a term and delete it.
- *Word Substitution.* For a tuple attribute value, we randomly select a term, and use WordNet [44] to find synonyms or hypernyms



ID	Movie	Director	Star	Country
t_1	Titanic	Hames Cameron	Null	United States
t_2	Titanic	NULL	Lenard Dicaprio	U.S.
t_3	Titanic	NULL	NULL	United States
t_4	Joker	Todd Philips	NULL	United States
t_5	Joker	NULL	Joaquin Phoenix	United States
t_6	Joker	NULL	Tom Cruises	United States

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

to form a candidate set. Finally, we randomly select a term from the candidate set to substitute the original term.

- *Word Swapping*. For a tuple attribute value, we randomly select two neighboring terms, and reverse the ordering.

Character-level. For 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 to 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 adopt the widely used strategy of 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 for 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 such cases, 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 masking 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 an embedding using a pretrained language model. There are two different embedding methods, namely word-level embeddings such as word2vec [43] and GloVE [53], and subword-level embeddings such as FastText [11] and BERT [31]. Word2vec encodes each term individually and 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 use a subword-level embedding method. This choice is driven by their capability to handle unknown and uncommon terms, while also exhibiting greater resilience to typographical errors.

For every sequence of tokens, the respective token embeddings are aggregated into a single embedding vector. We adopt a transformer-based architecture [31] for this aggregation process. This choice stems from their proven ability to adeptly capture contextual information embedded in sequences in the 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 representation for two tuples $\boldsymbol{emb}(t_1)$ and $\boldsymbol{emb}(t_2)$, the matcher outputs 1 or 0, indicating whether the two tuples should be integrated. One straightforward way to achieve this is to compute the cosine similarity between $\boldsymbol{emb}(t_1)$ and $\boldsymbol{emb}(t_2)$, or concatenate the two embeddings into a Multi-layer Perceptron (MLP). A common drawback emerges from the uniform treatment of every attribute, which ignores any potential variations in the contributions to the semantic correctness for a given tuple.

To overcome this drawback, we propose an *Attentional Integration Condition Judgment Network* (AICJNet), which assigns different weights to the attributes in the matching process. Specifically, for two embeddings $\boldsymbol{emb}(t_1)$ and $\boldsymbol{emb}(t_2)$, we first concatenate both into a single embedding: $\boldsymbol{emb}(t_1, t_2) = [\boldsymbol{emb}(t_1), \boldsymbol{emb}(\boldsymbol{t_2})]$. Next, we consider the varying importance of each attribute and reformulate the representation of the tuple pair (t_1, t_2) using a self-attention mechanism [64]:

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

Here, $\boldsymbol{emb}^*(t_1, t_2)$ is the final representation of the tuple pair (t_1, t_2), d_k represents the size of each attribute embedding, and $\boldsymbol{Mask}(t_1, t_2)$, is the concatenation of $\boldsymbol{Mask}(t_1)$ and $\boldsymbol{Mask}(t_2)$, is used to mask the impact of a missing attribute value relative 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 [64].

Finally, $\boldsymbol{emb}^*(t_1, t_2)$ is the input, and 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 contrastive training objective optimized using an SGD-based algorithm; (2) Identify adversarial examples to further enrich the training set. Next, we will explain this idea in detail.

Objective Function. In this work, we use a binary noise contrastive estimation (NCE) loss function [34] for the training objective, which is formulated as:

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

Here, N is the total number of tuples in the training table, and t_{ij}^{+} and t_{ik}^{-} denote one positive instance and negative instance for the tuple t_i, respectively. The optimization of the NCE loss function enables the model to make similar tuples that are close in the embedding space while scattering dissimilar tuples.

Adversarial Examples and Training. We propose the use of adversarial examples [29, 71] to further enrich the pool of positive training instances for our model. Technically speaking, an adversarial example is typically viewed as a perturbed version of the original input example, which results in a significant impact in a decision made by a machine learning model. Conversely, perturbations in data augmentation that operate on the original tuples while perturbations in adversarial training directly impact the embedding vectors, expressed as:

$$p(\boldsymbol{emb}(t_i)) = \boldsymbol{emb}(t_i) + \boldsymbol{r}, \tag{5}$$

where $\boldsymbol{r}$ denotes the perturbation vector. Since the adversarial example has the most significant impact on model performance, this can also be expressed as the following objective function:

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

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

Since r is very small, the loss function is approximately equivalent to the following equation derived using a first-order Taylor approximation [50]:

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

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

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

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

4 INTEGRABLE SET DISCOVERY

Given the integrability of any two tuples in Table T, determined using a binary classifier, we introduce how to derive all of the integrable sets contained in T in this section. Most exisitng work, such as ALITE [33], is restricted to integrating two tuples in each step of the algorithm. This limitation precludes including these tuples to resolve conflicts that may arise in a larger set of tuples. In contrast, *our objective is to integrate multiple tuples simultaneously*, which provides an opportunity to detect conflicts and resolve them as they occur. So, the goal is: Given a set of tuples drawn from a table T, identify all of 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 to 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 integrability of any two tuples in T using pairwise integration condition judgement (Definition 1). 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 a 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 equivalent to the integrable set that we hope to identify. To identify all of 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, each 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. After every vertex has been visited, we may obtain multiple vertex sets, each of which is considered to be an integrable set. Later, we use $\mathcal{S}$ to indicate a collection of integrable sets, where $S \in \mathcal{S}$. The DFS algorithm is well-understood, so we refer readers to a technical report [1] for details to find connected components.

5 CONFLICT RESOLUTION FOR 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 of the tuples contained in S into a *single* tuple t_{new}, which provides more comprehensive information than each individual tuple $t \in S$. Thus, the problem now is how to determine the correct attribute value $t_{new}[a]$, which can fall into one of three potential types:

- **Non-missing Attributes.** An attribute a refers to a non-missing attribute if $t[a] \neq NULL$ for any $t \in S$. For 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.
- **Totally-missing Attribute.** An attribute a is referred to as a totally-missing attribute if $t[a] == NULL$ for all $t \in S$. In this case, $t_{new}[a]$ is assigned a $NULL$ value. This choice is motivated by the absence of any reliable information to populate $t_{new}[a]$.
- **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-null values for multiple tuples on the attribute a may conflict with each other, 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 a data fusion problem rely on fact discovery approaches [36, 37, 72], which first estimate the trustworthiness of a data source, and subsequently choose the value from the most reliable source to fill a missing value. However, fact discovery approaches typically rely on a lot of training data or metadata (i.e., paper citation and reviews) in order to estimate the reliability of the source. 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 requires considerable human curation and financial burdens.

Our solution for conflict resolution is inspired by the recent success of in-context learning (ICL) [3, 45, 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 pretrained on an extensive corpora to accumulate a broad spectrum of knowledge, ICL models exhibit outstanding performance even when fine-tuning using limited labeled data. This allows us to extend predictions to downstream machine learning tasks despite any 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 as a classifier. 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 into a natural language query, such that it can be processed using an LLM. For example, consider a sentiment analysis task that uses binary predictions for reviews, the instruction template can be defined as "*Review:__, Sentiment:__*". Here, "Review" and "Sentiment" are the keywords that provide instructions to the LLM. The first blank line is filled with the content from a review and the second blank line is filled with either a "positive" representing the label 1, or a "negative" representing the label 0; these labels will then be predicted using the LLM.

In this work, we focus on the task using a given tuple t_{new}, along with values for its non-null attribute a_{non}. Our objective is to fill the partially-missing attribute a_{par} for the resulting tuple t_{new}, where the candidate values come from other tuples in the integrable set that have non-null values for a_{par}. To accomplish this, we create a natural language instruction template as follows:

For a tuple, if the values of an attribute __ is __ , the attribute __ is __, ..., respectively, then the value of the attribute __ should be __.

Example 1. *If we use a custom instruction template to formulate the conflict resolution task as depicted in Fig. 1, the following natural language query is created: "For a tuple, if the value for the attribute 'movie' is 'Joker', and the attribute 'director' is 'Todd Philips', and the attribute 'Country' is 'United States', respectively, then the 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 for the tuple.*

Step 2: Selecting Demonstration Examples. Demonstration examples are labeled data and tell LLMs the correct answer for a template-stylized query. For example, "Joaquin Phoenix" is the correct answer for the natural language query in Example 1. In other words, demonstration examples guide LLMs how to make prediction for the downstream tasks. The choice of demonstration examples, denoted as D, has a significant impact on the effectiveness of ICL. 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 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 from non-null attributes within the integrable set, which produces a set of multiple t_{new} of size $|\mathcal{S}|$. Furthermore, we encode each t_{new} using Eq. 1 to generate an encoded representation $\boldsymbol{emb}(t_{new})$, and then compute the average embedding $\overline{\boldsymbol{emb}(t_{new})}$ for all $\boldsymbol{emb}(t_{new})$ with size $|\mathcal{S}|$. Next, we perform a k-NN search to identify k tuples in table T that are most similar to $\overline{\boldsymbol{emb}(t_{new})}$ as the demonstration examples.

Step 3: Using LLM for Candidate Selection. 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} into a large language model (T5 [54]) 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 the entry $t_{new}[a_{par}]$.

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

6 DATA PREPARATION AND EVALUATION

This section describes how to create the benchmarks, followed by the evaluation framework employed in our experiments. An ideal benchmark must exhibit two critical characteristics: (1) Every dataset should contain semantic equivalence mistakes, typographical errors, and conflicts, all of which we address; (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 thorough effectiveness evaluation of our proposed solution. Given the presence of the first characteristics, directly using the benchmark proposed in ALITE [33], the most closely related work to ours, is impossible since errors and conflicts are not considered by ALITE. Thus, we have created our own benchmarks by injecting semantic equivalence, typographical

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 $t \in S$;
5          $t_{new}[a] = t[a]$;
6      else if $a$ is a totally-missing attribute then
7          $t_{new}[a] = NULL$;
8      else
9          Candidate Set $C = \emptyset$;
10         foreach $t \in S$ do
11             if $t[a] \neq NULL$ 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 $t \in D$ 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 $t_{new}$;
```

Table 1: The Statistics of the 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

errors, and conflicts, while recording the ground-truth annotations for evaluation. In Sec. 6.3, we also discuss why benchmarks from related 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 [33], **Real** and **Join** [33]. Both data pools contain multiple datasets, each of which contains a set of input tables to be integrated. Table 1 shows various statistics for the 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 perform an outer-union operator [10, 33] 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 noise into the clean table T, so as to simulate semantic equivalence and typographical errors using a certain error rate. Since the datasets used in the literature for data cleaning [2, 52] 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 for the ratio 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 noise by default.

- **Semantic Equivalence.** Backtranslation [20, 58] has been widely employed to generate 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 back-translation [20, 58] 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 [44] to generate a synonym or hypernym to replace it. We only inject this type of noise in text attributes.
- **Typographical Errors.** We simulate typographical errors in a comprehensive way, such as character swapping and character deletion. 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 below), we choose one partially-missing text attribute a (introduced in Sec. 5), and a tuple t that has a non-null value for the 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, each of which has a reliability score r between 0 and 1. Then, we randomly replace $t[a]$ with another value from the other values in the attribute a with a probability $1 - r$. Using this strategy, we create 3-5 conflict tuples for each integrable set. We set r_i to a random 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 [33] to integrate tuples in an error free and conflict free manner. During the integration, we track tuple pairs that are integrated, which are considered to be part of the ground-truth. Then based on the results from the pairwise integration condition decisions from ALITE, we run our DFS algorithm to find integrable sets in T. Thus, in our benchmarks, the ground-truth can be generated by obtaining the tuples that form the integrable sets. For conflict resolution, the template tuple value $t[a]$ for the attribute a is the ground-truth since we create the conflict tuples based on the tuple t.

6.2 Evaluation Metrics

Recall that multituple integration is achieved by implementing two core tasks, *integrable set discovery* and *multi-tuple conflict resolution.* As mentioned in Sec. 2, pairwise integration condition judgement has 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 one.

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, 39, 46], so we use *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 mean of $Recall$ and $Precision$, namely $F_1 = 2 \times \frac{Recall \times Precision}{Recall+Precision}$. T_P denotes the number of integrable tuple pairs that are judged as integrable, F_N represents the number of integrable tuple pairs 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. *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 any 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 $J = \frac{|R_i \cap S_j|}{|R_i \cup S_j|}$.

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, 38, 46] on entity resolution and conflict resolution provide benchmarks, they cannot be used in our experiments for the following reasons: (1) In most entity resolution benchmarks [19, 39], 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 using an outer-union operator [10, 33] by using several tables of different schemas, it usually produces many missing values in 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 [39, 63], one of the most widely used benchmarks in entity resolution, has only five attributes, while the average number of attributes in the datatsets in *Real* and *Join* data pools are 19.6 and 10.6, respectively.

7 EXPERIMENTS

In this section, we conduct experiments to verify the effectiveness of our proposed method for two tasks, integrable set discovery and multi-tuple conflict resolution. Considering that integrable set discovery is similar to entity resolution, in Sec. 7.3, we conduct an additional experiment to compare our method and representative entity resolution methods on five widely used ER datasets.

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 using supervised training, which is described in Sec. 7.3.1.
- **Unicorn [63]** – This method unifies six data matching tasks and achieves state-of-the-art performance for the task of entity resolution, which is close to our own approach.

For all of the methods except SSACL-SSL, we use data augmentation methods (described in Sec. 3.2) to generate training data for all the three methods. 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:

- **ICLCF** – Our ICL-based conflict resolution method introduced in Sec. 5. We use the k-NN demonstration example selection strategy by default.
- **Random** – A simple 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** [56] – A state-of-the-art truth discovery model for conflict resolution for single fact scenarios.

Table 2: Key Parameters used in our Methods

Methods	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].

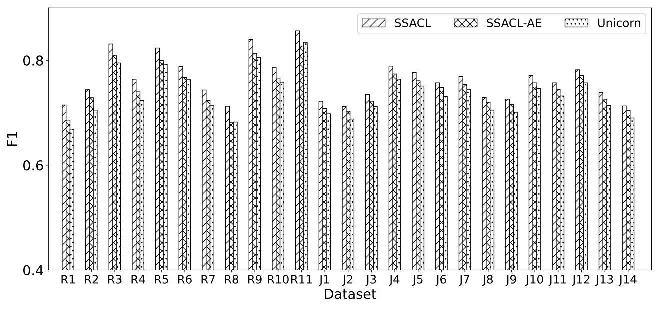
7.1.3 Model Settings. We use BERT to obtain pretrained word embeddings and the embedding size is set to 768 by default. When training SSACL, the encoder parameters are fixed, and we only optimize the parameters for matcher during model training. Specifically, we use Adam [35] to optimize the model with 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 the T5 [54] LLM 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

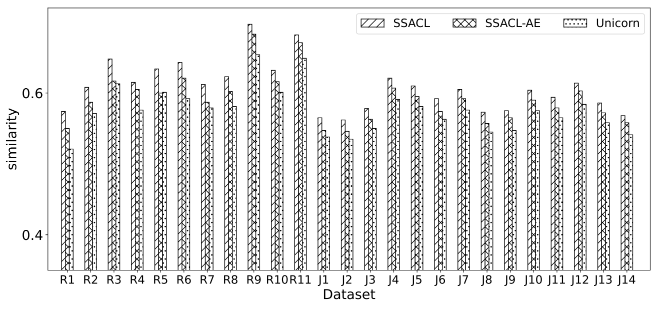
7.2.1 Effectiveness Study. We compare the effectiveness of three methods: SSACL, SSACL-AE, and Unicorn. As introduced in Sec. 6.2, we use *similarity* to evaluate the quality of integrable sets found by each method.

Fig. 3, 4 and 5 present the effectiveness of SSACL and the baselines in different cases of balanced noise, SE-heavy noise and typographical error-heavy noise. Due to space limitations, 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.

Based on the above figures, we can observe the following: First, for all cases of noise injection, SSACL achieves the highest *similarity* for the majority of datasets, and outperforms Unicorn by a large margin. Specifically, on average, the relative improvement of SSACL

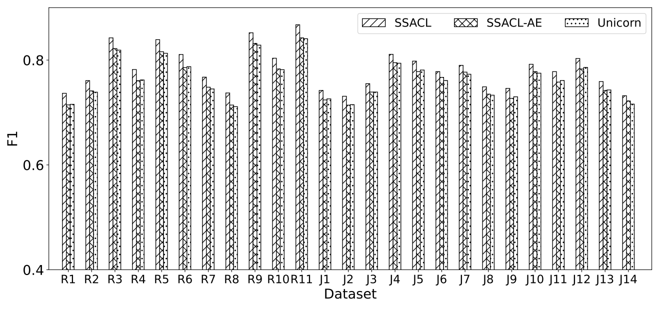


(a) F1

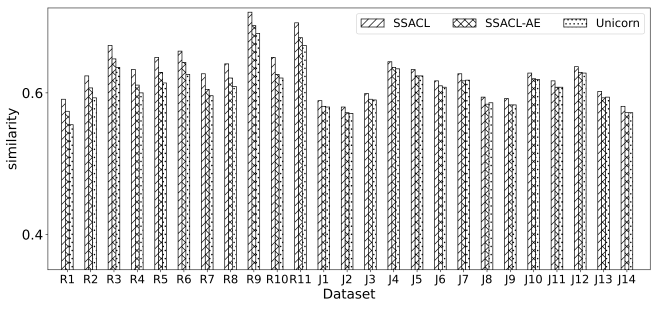


(b) similarity

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

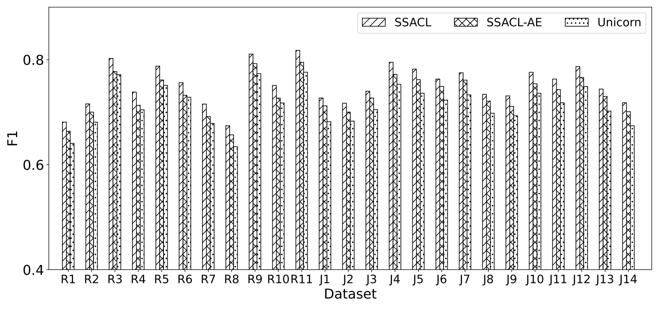


(a) F1

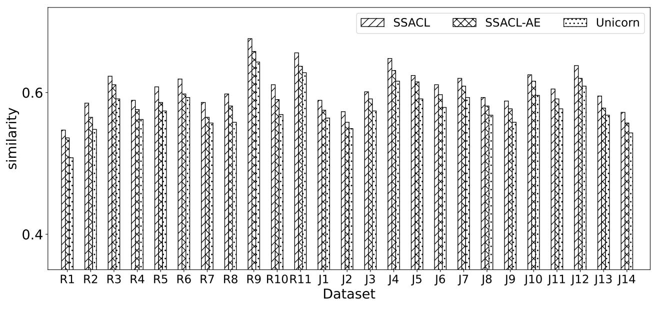


(b) similarity

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

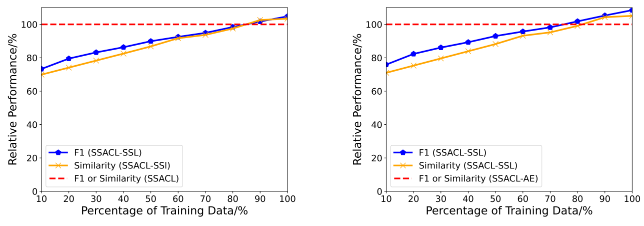
Figure 5: *F1* and *similarity* in the case of TE-heavey noise (R1-R11 are datasets from *Real* and J1-J14 are datasets from *Join*).

over Unicorn in the cases of balanced, SE-heavy, and typographical error-heavy noise are 3.11%, 2.54%, and 5.27%, respectively. 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 in Sec. 3, which is able to distinguish the important attributes when matching two tuples.

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 *similarity* achieved in the case of balanced noise. 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 to the average *simialrity* achieved in the case of balanced noise, which demonstrates that, compared to semantic equivalence, typographical errors are much more difficult 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 pairwise integration condition judgement by *F1*, as illustrated in Fig. 3, 4 and

5. The results in terms of *Recall* and *Precision* show similar trends as those 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 judgements.



(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

7.3 Evaluation on Entity Resolution

As discussed in Sec. 8, despite different challenges encountered such as representative noise and missing values, entity resolution and pairwise integration condition judgement share similar goals. Hence, we conduct an additional experiment to compare our proposed SSACL with state-of-the-art ER methods on five widely used ER datasets: DBLP-Scholar (DS), DBLP-ACM (DA), Walmart-Amazon (WA), Amazon-Goolge (AG), and Itunes-Amazon (IA). Since the original ER datasets are clean ones, we create two versions of dirty datasets for each original ER dataset: (1) We employ the noise injection method introduced in Sec. 6.1 to create the dirty version of the ER dataset; (2) Following [46], the "dirty" datasets are generated by randomly emptying attributes and appending their values to another randomly selected attribute. We compare our proposed SSACL with both Unicorn (a state-of-the-art supervised method) and sudowoodo [65] (a state-of-the-art unsupervised method).

Table 3, 4, and 5 show the F1 Scores for all methods using the three different versions of ER datasets, respectively. We have several observations: (1) Unicorn undoubtedly achieves the best performance, since it is the only method that is trained using high-quality labeled data. However, the performance drops sharply on "dirty" datasets when compared with SSACL. Specifically, Unicorn exhibits an average performance decrease of 10.0% and 6.6% when shifting from clean datasets to dirty datasets with typographical errors and semantic equivalence, and attribute replacement, respectively, while the average performance decrease of SSACL is 3.7% and 1.1%, respectively. (2) On dirty datasets with typographical errors and semantic equivalence, our method SSACL outperforms Sudowoodo, with an average improvement of 2.03%. (3) On dirty datasets with the attribute replacement, SSACL achieves a comparable performance with that of Sudowoodo. (3) Even on the clean datasets, the performance of SSACL is similar to Sudowoodo.

In summary, our SSACL exhibits robustness to noisy data, while maintaining a promising level of performance on clean data.

Table 3: Relative Performance on Five ER Datasets with Typographical Errors and Semantic Equivalence by F1 Score.

	Sudowoodo	**Unicorn**	**SSACL**
DBLP-Scholar	76.3	83.4	78.2
Walmart-Amazon	64.1	71.8	65.6
DBLP-ACM	86.4	89.0	87.4
Amazon Google	59.7	64.0	61.2
Itunes-Amazon	82.7	90.3	85.4

Table 4: Relative Performance on Five ER Datasets with Attribute Misplace by F1 Score.

	Sudowoodo	**Unicorn**	**Ours**
DBLP-Scholar	81.8	87.7	80.3
Walmart-Amazon	68.1	75.4	67.2
DBLP-ACM	90.3	92.3	91.7
Amazon Google	63.7	67.2	64.1
Itunes-Amazon	83.2	90.3	84.4

Table 5: Relative Performance on Five Clean ER Datasets by F1 Score.

	Sudowoodo	**Unicorn**	**Ours**
DBLP-Scholar	84.7	93.1	82.5
Walmart-Amazon	71.6	82.9	68.4
DBLP-ACM	93.8	98.5	92.9
Amazon Google	65.4	70.4	63.2
Itunes-Amazon	87.3	97.6	85.6

7.3.1 Effectiveness with Limited Labeled Data. Next, we investigate how SSACL performs using limited labeled data by comparing against three baselines: SSACL, SSACL-AE and SSACL-SSL. SSACL is trained using labeled data. Here, we use the largest dataset $J11$ in *Real* to perform the 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 from 10% to 100% at an interval of 10% and train SSACL-SSL. To provide a fair comparison, we evaluate all 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.

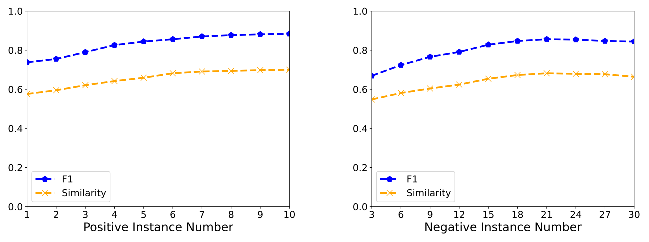
Fig. 6 shows the experimental results. Note that the performance of SSACL-SSL increases as the training data increases. When the percentage of training data is between 10% and 70%, the performance of SSACL-SSL is worse than SSACL and SSACL-AE. When the training data percentage is 80% or above, we can see that SSACL marginally outperforms SSACL-SSL, and SSACL-AE achieves similar performance to SSACL-SSL. Even with all training data available, SSACL-SSL can only marginally outperform SSACL and SSACL-AE, with a relative improvement of 3.4% and 5.1% on the *similarity* metric, respectively. Note that our model performance may be further improved using additional tables during the pre-training stage. This demonstrates the effectiveness of our proposed method when automatically generating labeled data, which enables our model to achieve comparable performance to the 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 the value of introducing adversarial examples to enhance model training. Overall, the experimental results verify the effectiveness of SSACL using limited labeled data.

7.3.2 Impact of The Number of Positive and Negative Instances. Finally, we investigate the impact of two key hyperparameters – the number of positive instances N_{pos} and the number of negative instances N_{neg}, on the model performance of SSACL. We use the $J11$ dataset again to evaluate the 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, enabling the model trained to more consistently identify equivalent semantic matches, 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 from 3 to 21. Within this range, a 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.

7.3.3 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 using a blocking technique may bring slight improvements 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*

7.4 Evaluation for Multi-tuple Conflict Resolution

7.4.1 Effectiveness Study. To ensure a fair comparison, we maintain parity for the amount of labeled data available for training in both ICLCF and SlimFast [56]. 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 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 the 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.

7.4.2 Effectiveness using 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. 7.4.1) 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 achieves 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 observed, 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. 7.4.1). This demonstrates the effectiveness of ICLCF in a scenario where labeled data is limited.

7.4.3 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.

7.4.4 Efficiency Study. On average, it takes ICLCF and SlimFast 35ms and 157ms to make the prediction for each integrable set in the 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, the additional time cost is an acceptable tradeoff.

8 RELATED WORK

We first describe the two operators complementation and subsumption which are commonly used in data integration, and then present related work on entity resolution and data fusion.

Complementation and Subsumption. Complementation and subsumption [15, 16, 30, 51, 55] are two operators used to integrate

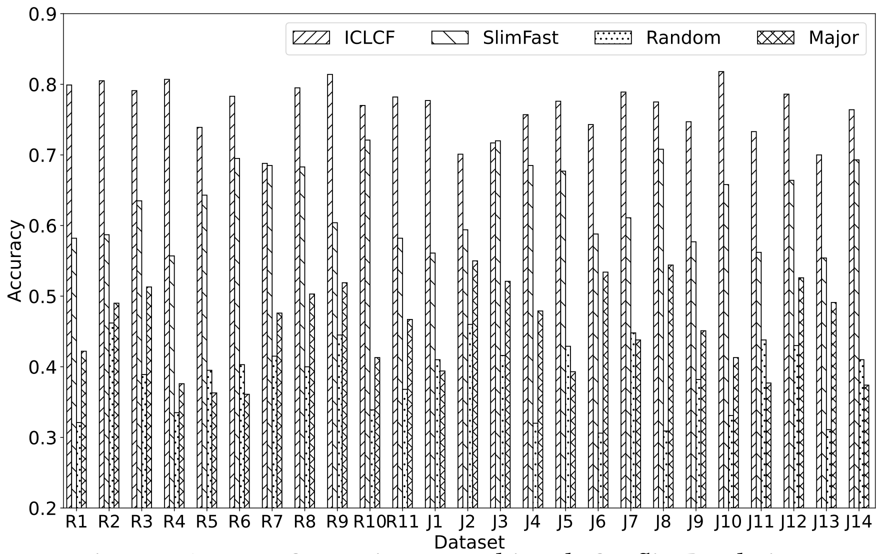


Figure 8: *Accuracy* Comparison on Multi-tuple Conflict Resolution

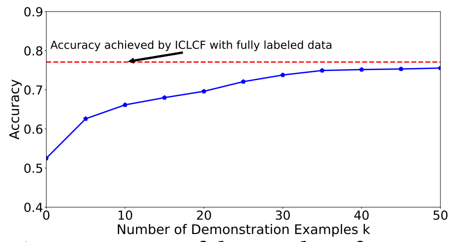


Figure 9: Impact of the Number of Demonstration Examples on *Accuracy of ICLCF*

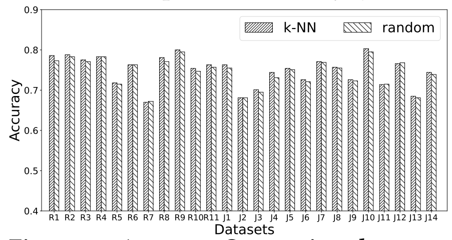


Figure 10: Accuracy Comparison between the Two Selection Strategies

tuples in a given table. Subsumption [26] was introduced to implement full disjunction (FD), an associative version of an outer join, and complementation [10] was introduced as an operator for data fusion. Both operators involve iterative pairwise comparisons between two tuples. Previous studies mainly focus on enhancing efficiency. For example, the most recent work ALITE [33] explored how to integrate data lake tables using complementation and subsumption, providing notable efficiency gains by using blocking techniques. However, as complementation and subsumption have restrictive requirements for integration condition judgements, and only integrate two tuples in a single step without handling typographical errors or conflicts.

Entity Resolution. The most relevant work for pairwise integration condition judgements is *entity resolution*, which determines if two entities refer to the same thing. Existing methods can be divided into four categories: rule-based methods [24, 59], crowdsourcing methods [13, 14, 17, 23, 69], conventional machine learning (ML)-based methods [8, 41, 60], and deep learning (DL)-based methods [19, 39, 40, 46, 61–63, 65, 66, 70]. Currently, DL-based methods often exhibit the best overall performance and they can be further divided into two categories, supervised ER methods [19, 39, 40, 46, 62, 63] and unsupervised methods [65, 66]. While supervised ER methods mainly focus on how to employ massive high-quality labeled data to train the models to achieve high effectiveness, unsupervised ER methods mainly investigates how to train an effective machine learning model using limited labeled data.

While entity resolution is similar to integration condition judgments, there are two important distinctions: (1) Integration condition judgments match tuples containing *representative noise*, which encompasses issues such as typographical errors or semantic equivalence. (2) Entities in entity resolution are typically derived using comprehensive information, whereas the tuples in our task often contain missing values – a common occurrence in large data lakes, especially when produced using an outer union operator. This disparity demands a more different type of comparison at the attribute-level for two tuples, as opposed to a tuple-level assessment based on previous knowledge.

Data Fusion. Data fusion [9, 42] determines how to merge multiple records, which map to the same entity but originate from different resources, using a unified representation. At the core of data fusion lies the challenge of addressing conflicts when multiple values exist for a particular attribute. Although a few simple, yet intuitive methods, such as using the mean or median values for numerical values or using majority voting for categorical data can be employed, they often reduce the quality of the overall data. Existing solutions [6, 18, 36, 56, 67, 72] for data fusion predominantly rely on the concept of fact discovery, which involves assessing the reliability of each data source and selecting the value using the most reliable source. However, estimating source reliability for fact discovery methods relies heavily 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 fact discovery is similar to our own, the methodologies are difficult to apply to our problem since the data necessary is rarely available.

9 CONCLUSION

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

REFERENCES

[1] [n. d.]. Technical Report and Source Code. https://anonymous.4open.science/r/Robin-8311/.

[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] 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).

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

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

[36] 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.

[37] 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).

[38] 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).

[39] 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.

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

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

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

[43] 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).

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

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

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

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

[52] 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.

[53] 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.

[54] 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.

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

[56] 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.

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

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

[59] 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.

[60] 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.

[61] Saravanan Thirumuruganathan, Han Li, Nan Tang, Mourad Ouzzani, Yash Govind, Derek Paulsen, Glenn Fung, and AnHai Doan. 2021. Deep learning for blocking in entity matching: a design space exploration. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2459–2472.

[62] 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.

[63] 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.

[64] 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).

[65] Runhui Wang, Yuliang Li, and Jin Wang. 2023. Sudowoodo: Contrastive self-supervised learning for multi-purpose data integration and preparation. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 1502–1515.

[66] Renzhi Wu, Sanya Chaba, Saurabh Sawlani, Xu Chu, and Saravanan Thirumuruganathan. 2020. Zeroer: Entity resolution using zero labeled examples. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1149–1164.

[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] Dezhong Yao, Yuhong Gu, Gao Cong, Hai Jin, and Xinqiao Lv. 2022. Entity resolution with hierarchical graph attention networks. In *Proceedings of the 2022 International Conference on Management of Data*. 429–442.

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

[72] 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.

[73] 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 6: Typographical Errors for Numerical Attributes

Error type	Description
char swap	Swap two random consecutive word characters in the text.
missing char	Skip a random word character in the text.
extra char	Add an extra, keyboard-neighbor, letter next to a random word character.
nearby char	Replace a random word character with keyboard-neighbor letter.
similar char	Replace a random word character with another visually similar characte
skipped space	Skip a random space from the text.
random space	Add a random space in the text.
repeated char	Repeat a random word character.
unichar	Replace a random consecutive repeated letter with a single letter.

Table 7: Typographical Errors for Numerical Attributes

Error type	Description
Method	Description
digit swap	Swap two random consecutive digits in the number.
missing digit	Skip a random digit in the number.
extra digit	Add an extra, keyboard-neighbor, digit next to a random digit in the number.
nearby digit	Replace a random digit in the number with a keyboard-neighbor digit.
similar digit	Replace a random digit with another visually similar digit.
repeated digit	Repeat a random digit in the number.
unidigit	Replace a random consecutive repeated digit with a single digit.
unichar	Replace 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

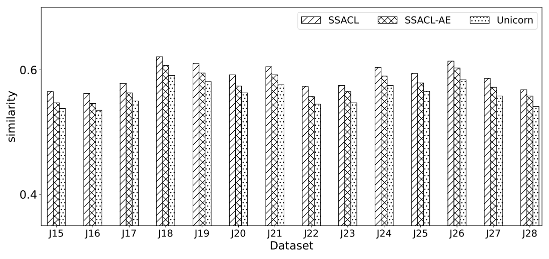
```
Data: Graph $G$
Result: List of connected components $\mathcal{S} = S_1, S_2, S_3, \ldots$
Function DFS(vertex $v$, list $S$):
    Mark $v$ as visited;
    Add $v$ to $S$;
    foreach neighbor $u$ of $v$ do
        if $u$ is not visited then
            DFS($u, S$);
        end
    end
end
Function FindConnectedComponents(Graph $G$):
    Initialize an empty list $\mathcal{S}$;
    foreach vertex $v$ in $G$ do
        if $v$ is not visited then
            Create an empty list $S$;
            DFS($v, S$);
            $\mathcal{S}.append(S)$
        end
    end
    return $\mathcal{S}$;
end
```

C EXPERIMENT RESULTS

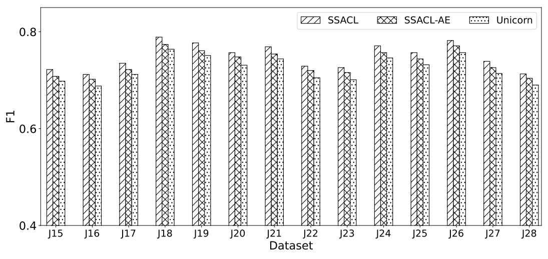
C.0.1 Effectivenss Comparison on the task of Integrable Set Discovery. Fig. 11, 12, and 13 illustrates the *similarity* scores achieved by the compared methods on the datasets of $J15 - J28$. The experimental results show the similar trends. SSACL also achieved the best performance on all the cases of 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%.

We also use *Recall* and *Precision* to evaluate the compared methods, whose results are illustrated in Fig. 14-19. Those results also demonstrate similar trends that SSACL achieve the best *Recall* and *Precision* in almost all the datasets for all the cases of noises injection. Specifically, in the case of balanced noises, the *Recall* and *Precision* of SSACL are 0.768 and 0.729, respectively, outperforming those of Unicorn by 4.73% and 2.65%, respectively; in the cases of SE-heavy noises, the *Recall* and *Precision* of SSACL are 0.781 and 0.754, respectively, outperforming those of Unicorn by 2.86% and 2.02%, respectively; in the cases of TE-heavy noises, the *Recall* and *Precision* of SSACL are 0.761 and 0.724, respectively, outperforming those of Unicorn by 6.07% and 5.01%, respectively.

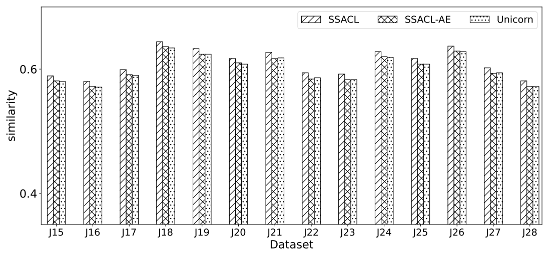


(a) F1

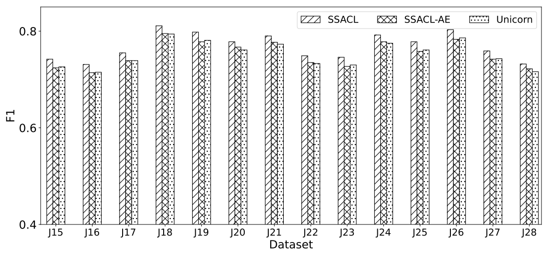


(b) similarity

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

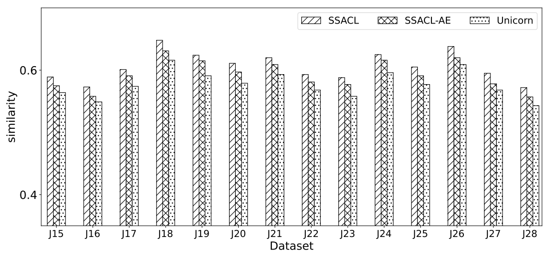


(a) F1

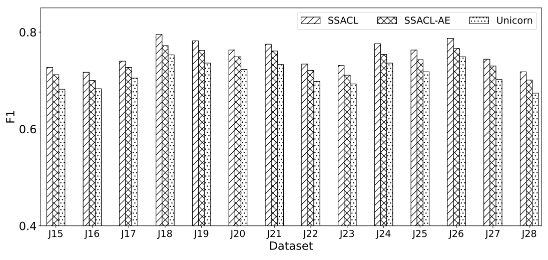


(b) similarity

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 $J15 - J28$.

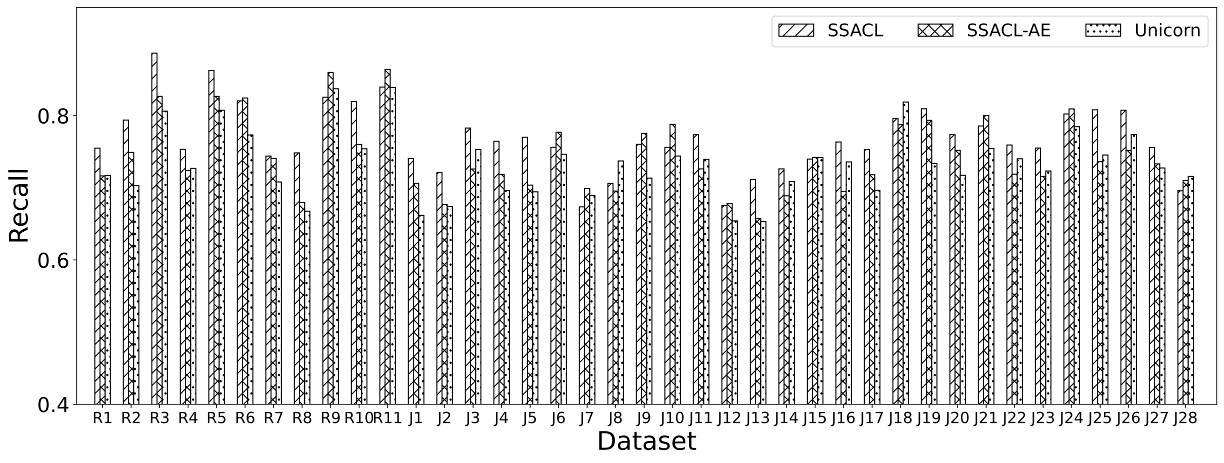


Figure 14: *Recall* Comparison in the case of Balanced Noises

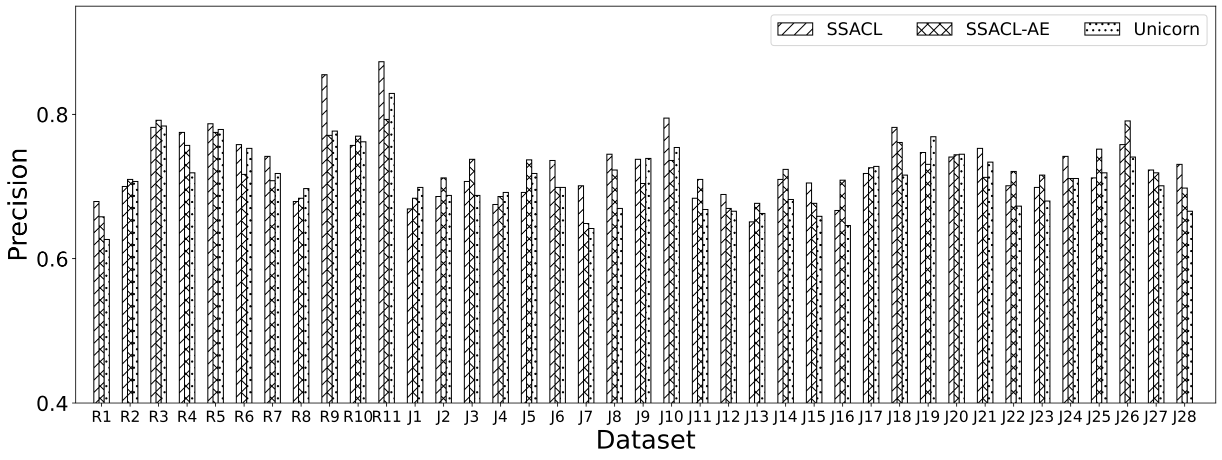


Figure 15: *Precision* Comparison in the case of Balanced Noises

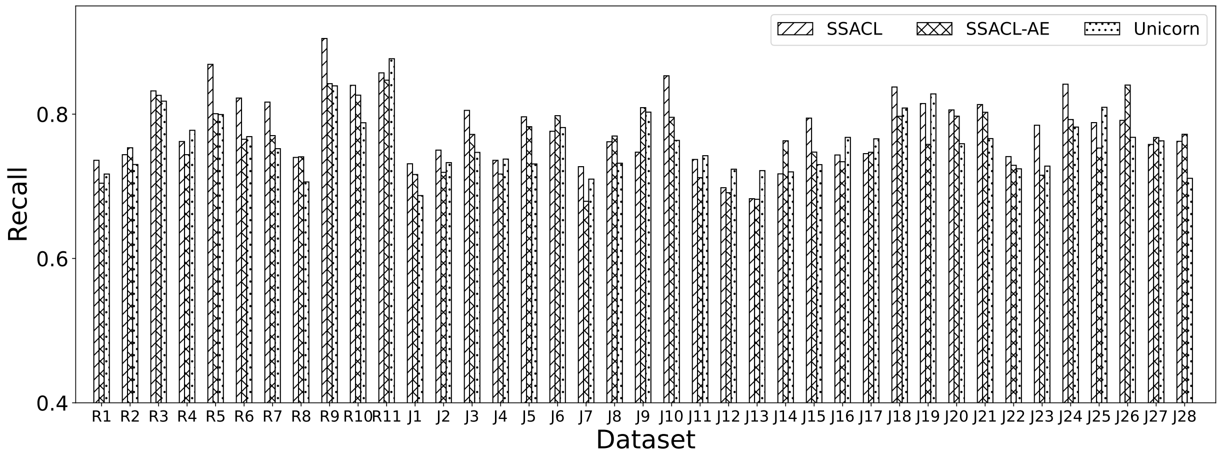


Figure 16: *Recall* Comparison in the case of SE-heavy Noises

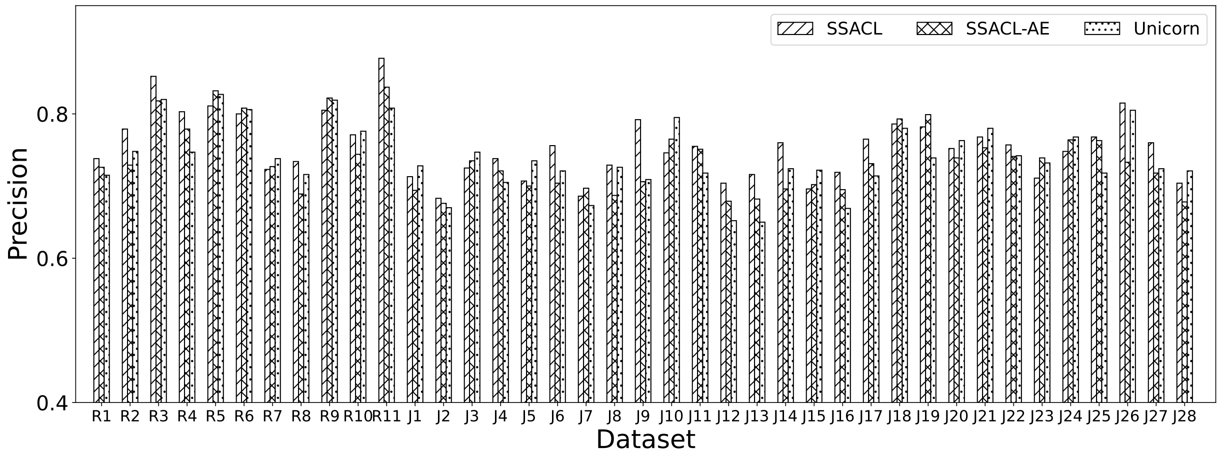


Figure 17: *Precision* Comparison in the case of SE-heavy Noises

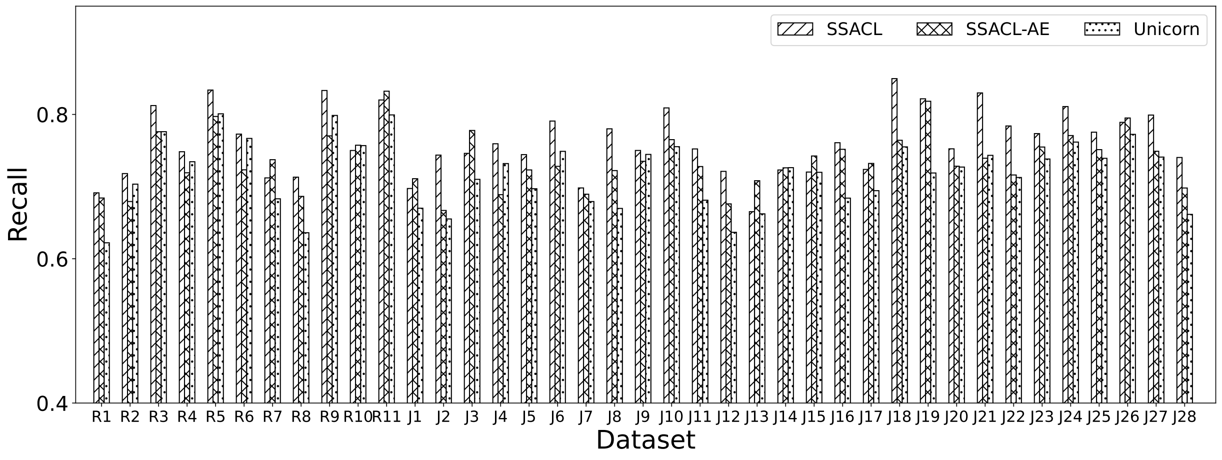


Figure 18: *Recall* Comparison in the case of TE-heavy Noises

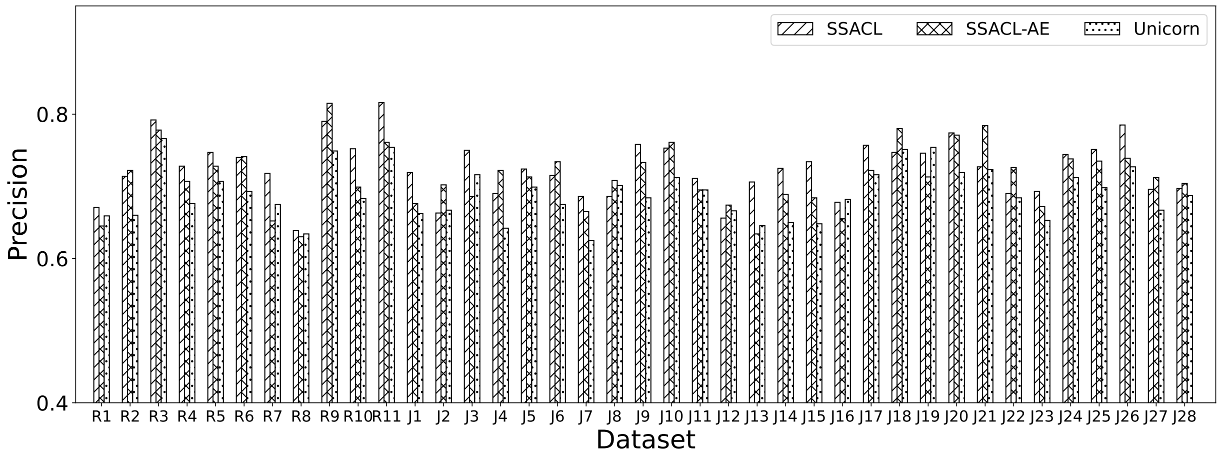


Figure 19: *Precision* Comparison in the case of TE-heavy Noises