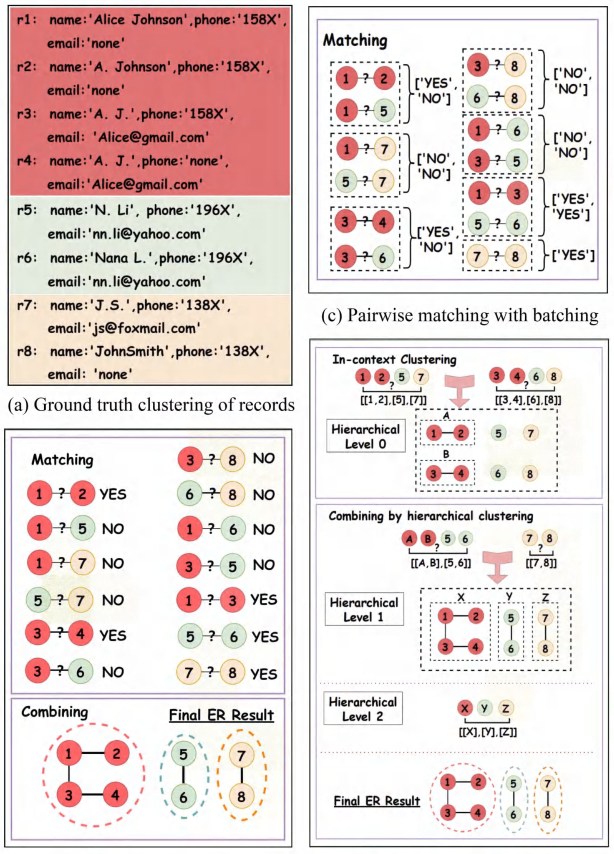
In-context Clustering-based Entity Resolution with Large Language Models: A Design Space Exploration

ABSTRACT

Entity Resolution (ER) is a fundamental data quality improvement task that identifies and links records referring to the same real-world entity. Traditional ER approaches often rely on pairwise comparisons, which can be costly regarding both time and monetary resources, especially when large datasets are involved. Recently, Large Language Models (LLMs) have demonstrated promising results in ER tasks. Still, existing methods typically focus on pairwise matching, missing the potential of LLMs to directly perform clustering in a more cost-effective and scalable manner. In this paper, we propose a novel *in-context clustering* approach for ER, where LLMs are used to cluster records directly, reducing both time complexity and monetary costs. We systematically investigate the design space for in-context clustering, analyzing the impact of factors such as set size, diversity, variation, and ordering of records on clustering performance. Based on these insights, we develop LLM-CER (LLM-powered Clustering-based ER) that obtains high-quality ER results while minimizing LLM API calls. Our approach addresses key challenges, including efficient cluster merging and LLM's hallucination, providing a scalable and effective solution for ER. Extensive experiments on nine real-world datasets demonstrate that our method significantly improves result quality, achieving up to 150% higher accuracy, 10% increase in the FP-measure, and reducing API calls by up to 5×, while maintaining a comparable monetary cost to the most cost-effective baseline.


r1: name:'Alice Johnson',phone:'158X', email:'none'
r2: name:'A. Johnson',phone:'158X', email:'none'
r3: name:'A. J.',phone:'158X', email: 'Alice@gmail.com'
r4: name:'A. J.',phone:'none', email:'Alice@gmail.com'
r5: name:'N. Li', phone:'196X', email:'nn.li@yahoo.com'
r6: name:'Nana L.',phone:'196X', email:'nn.li@yahoo.com'
r7: name:'J.S.',phone:'138X', email:'js@foxmail.com'
r8: name:'JohnSmith',phone:'138X', email: 'none'

(a) Ground truth clustering of records (b) Pairwise matching (c) Pairwise matching with batching (d) In-context clustering

Figure 1: (a) Ground truth ER result. (b) Pairwise matching and (c) Pairwise matching with batching (state-of-the-art) vs. (d) In-context clustering (this work) for ER. The combining phase is omitted for pairwise matching with batching [27], as it remains the same with pairwise matching [55].

1 INTRODUCTION

Entity Resolution (ER) is a fundamental data quality task that identifies and disambiguates real-world entities represented by different identifiers or descriptions across multiple records. In ER, a *record* denotes a specific data instance; an *entity* corresponds to a unique object represented by one or more potentially duplicate records; and *attributes* are the properties of each record, which may be numerical (e.g., price), categorical (e.g., brand), or textual (e.g., descriptions). ER aims to link and cluster these records, ensuring a clean and unified dataset [14, 16, 25, 56]. Traditional ER is computationally expensive, particularly for large datasets, as it often requires evaluating *all possible pairs* of records to detect duplicates. To reduce this cost, ER typically includes a pre-processing step, *blocking* or *filtering* [60], which partitions the dataset into (possibly overlapping) blocks, where records in different blocks are unlikely to refer to the same entity. The subsequent *resolution* phase, which involves matching and combining records, is the more resource-intensive step, ensuring that duplicates are grouped within the same cluster.

Existing ER solutions can be broadly divided into three categories: crowdsourcing-based frameworks [75], machine learning (ML) and deep learning-oriented methods [24], and those leveraging large language models (LLMs) [44], such as GPT or LLAMA. Crowdsourcing frameworks rely on human intelligence to achieve high-quality results but incur significant costs. For example, platforms like Amazon Mechanical Turk (AMT) typically charge around USD 0.02 per Human Intelligence Task (HIT) involving pairwise comparisons [78]. For a dataset containing 1,000 records, naive pairwise ER would require approximately $O(10^6)$ comparisons—amounting to around USD 20,000 costs. To mitigate such expenses, hybrid human-machine frameworks like CrowdER [78] employ automated techniques to eliminate obvious non-matches, reserving human effort for only the most uncertain pairs. Machine-based approaches, such as rule-based [5], ML-based [6], and deep learning-based [24] works, formulate pairwise matching as a classification problem. These models can be trained on labeled data to provide faster inference. Still, they require *task-specific supervision* and *large amounts of labeled data*, both of which are expensive and time-consuming to acquire. For instance, AMT charges USD 0.08 per labeling task [27], making the data annotation process costly and difficult. Recently emerged large language models, e.g., GPT-4, are pre-trained on massive corpora in a self-supervised manner [48] and have achieved remarkable results in a series of natural language processing and data science tasks like text generation [67], machine translation [26], data research and education [3]. General-purpose LLMs circumvent the need for expensive task-specific supervision and labeled data, thanks to simpler prompt-based interactions, making them easier to operate and iterate in downstream applications. Moreover, their lower costs—e.g., GPT-4o-mini charges only USD 0.15 per million tokens [1]—and superior performance on zero-shot tasks [39] make them an attractive alternative for entity resolution tasks.

Recently, several works have pursued LLMs for ER. Narayan et al. [55] pioneered the exploration of GPT-3's potential for ER in a few-shot manner using task demonstrations. Their approach has exhibited notable performance improvements compared to methods based on pre-trained language models (PLMs), e.g., Ditto [47]. ZeroMatch [90] takes a more advanced approach by using parameter-efficient fine-tuning, maintaining many pre-trained LLM variants from different domains, and selecting the most appropriate variant at inference time for zero-shot ER. Further advancements in prompt engineering for ER have been investigated in works such as [37, 54, 64, 81]. Li et al. [44] develop BoostER, a framework that applies LLMs to select the best answer from multiple different partitionings generated by existing ER tools. Their focus is on the next record pair selection problem, reducing the uncertainty of the current ER result. Additionally, batch prompting, which involves packing multiple pairwise questions into a single prompt to reduce the monetary cost of calling LLM APIs, has been studied in [27, 88].

However, all state-of-the-art LLM-based ER methods rely on *comparing record pairs* and do not fully exploit the emergent capability of LLMs to *directly cluster a set of records*. Recent studies have shown that LLMs are surprisingly effective at clustering text data in a zero-shot or few-shot manner, preserving semantic meaning and context [74, 77, 89]. Given that entity resolution is essentially a clustering problem, LLMs' clustering capabilities can be directly leveraged to reduce both the time and monetary costs associated with end-to-end ER. To bridge this gap, we – for the first time – formulate and investigate the problem of in-context clustering-based entity resolution, that is, instructing an LLM to cluster different record sets to achieve the end-to-end ER task at a significantly lower cost and time. Consider the following example.

Example 1. *Figure 1(b) compares existing pairwise matching [55] with our proposed in-context clustering approach for an ER task. Figure 1(a) shows the records with ground-truth entity assignments, where each color represents an entity.*

Pairwise questioning-based ER consists of two phases: matching and combining (excluding blocking for simplicity). Matching aims to determine if a pair of records refers to the same entity. Transitivity can reduce comparisons; for example, if records a and b are the same entity, and b and c are the same (or different), we can infer that a and c are also the same (or different). As shown in Figure 1(b), the matching phase concludes when all record pairs are compared explicitly, or their status is inferred through transitivity or anti-transitivity. In the combining phase, the final ER result is obtained by clustering records belonging to the same entity.

In contrast, our in-context clustering approach directly creates input record sets for clustering by the LLM, as depicted in Figure 1(d) (with a maximum size of 4). Records within each LLM cluster are considered the same entity (i.e., transitivity holds). For example, in the LLM-based clustering of the set $(1, 2, 5, 7)$*, records* 1 *and* 2 *are clustered together, indicating they are the same entity, while* 2 *and* 5 *belong to different entities (i.e., anti-transitivity), as reflected in their assignment to clusters* $A = [1, 2]$ *and* $[5]$*. In cases where transitivity or anti-transitivity relationships are unknown, clusters are packed to form the next level of input sets, such as* $(A, B, 5, 6)$*. This hierarchical process continues, merging clusters until all record pairs from different clusters satisfy anti-transitivity, yielding the final ER output.*

Unlike pairwise matching, which classifies two records at a time and heavily relies on transitivity, in-context clustering *partitions multiple records simultaneously* and *hierarchically merges them.* Each pairwise comparison requires an LLM API call, driving up both cost and time. In the worst case, the ER process requires comparing all pairs of records when they belong to different entities. As illustrated in Figure 1(b), pairwise matching requires 13 comparisons (i.e., 13 LLM API calls) for 8 records, even with the help of transitivity. Our method, however, requires only 5 record sets for clustering, leading to fewer LLM API calls, lower costs, and reduced running time. This efficiency becomes particularly significant for large-scale datasets, where processing costs and ER time are crucial.

Example 2. *An improved baseline, pairwise matching with batching [27], reduces costs by processing multiple pairwise questions in batches. With the same constraint of 4 records per LLM API call for a fair comparison (i.e., 2 pairwise questions in a batch), Figure 1(c) shows that batching reduces the number of API calls to 7, still higher than that of 5 by our in-context clustering approach. This inefficiency arises because the pairwise questioning format inherently introduces redundancy when conveying information to the LLM. For instance, exploring all possible relationships among four records require 6 pairwise questions, which necessitates 3 batches of 2 pairwise questions each. Even with transitivity and anti-transitivity among 4 records, the number of batches can only be reduced to 2. In contrast, our approach can capture all relationships among 4 records in a single API call, thus reducing the number of API calls. A key follow-up question is the highest possible information density (i.e., maximum set size constraint in our clustering solution) that the LLM can effectively process in a single API call*[1]*. This motivates our in-depth investigation in § 4.2.*

Although LLMs have demonstrated strong performance in ER via pairwise matching [27, 55], the application of in-context clustering for LLM-based ER remains largely unexplored. We observe that directly applying in-context clustering to ER, without a carefully designed mechanism, introduces several key challenges: **(1)** *Clustering Performance Variability:* The clustering performance of an LLM can vary significantly based on how the record sets are configured. Determining the optimal design for these sets—specifically in terms of size, diversity, and the ordering of records—is crucial. For instance, if the record set is too large, the LLM's performance may degrade due to long context lengths [45], whereas a smaller set size will result in excessive LLM API calls, thereby increasing the monetary cost. (2) *Hallucination Risks:* LLMs are prone to hallucinations, where the model generates outputs that may not be grounded in the input data [34]. As a result, the outputs of in-context clustering may not always be accurate. Addressing this challenge requires the development of effective result verification strategies, such as

[1]The number of relationships to explore grows quadratically with the set size.

implementing LLM "guardrails" and record set regeneration. **(3)** *Efficient Result Combination:* Combining the outputs of the in-context clustering efficiently presents another challenge. The process involves balancing the trade-off between the quality of the end-to-end ER result and the number of LLM API calls, which directly affects the monetary cost. To address these challenges, we extensively explore the design space and introduce several novel algorithms that optimize the creation of input record sets, the validation of LLM clustering outputs, and the merging of clustering results.

Our empirical evaluations reveal that in-context clustering generally performs best when each record set contains 9 records drawn from 4 distinct entities, with roughly equal representation per entity. For domain-specific datasets, however, the optimal configuration tends to involve fewer records and entities, as factors such as attribute type and noise can significantly impact clustering quality. Performance further improves when records from the same entity appear consecutively within the set. Building on these insights, we propose an effective record set creation strategy (§ 5.2) that balances set size, diversity, and intra-set variation to maximize clustering performance while minimizing monetary cost. Across nine real-world datasets, our method LLM-CER (LLM-powered Clustering-based ER) yields substantial improvements—up to 150% in Accuracy (ACC) and 10% in FP-measure—while reducing API calls by up to 5× and maintaining comparable monetary cost to the most cost-effective baseline (§ 6.2). Additionally, scalability tests confirm that our method remains both effective and efficient as dataset size increases (§ 6.5).

Contribution. Our main contributions are summarized below.

- We are the first to apply a clustering-based approach using LLMs for entity resolution, exploring the design space of clustering-based LLM ER (§ 4).
- We identify four key factors—set size, set diversity, set variation, and record order—and analyze their impact on the performance of in-context clustering (§ 4.2).
- We propose the Next Record Set Creation algorithm, which effectively addresses the four identified factors. To mitigate potential LLM misclassifications, we design a Misclustering Detection Guardrail and a Record Set Regeneration strategy for further refinement of results (§ 5.2). Our experimental findings demonstrate that MDG incurs 10% time overhead, while improving the FP-measure by up to 75% (§ 6.4).
- We propose a Hierarchical Cluster Merging approach that generates record sets hierarchically and performs efficient merging, improving the quality of cluster merging and overall entity resolution performance (§ 5.3).
- We empirically evaluate our clustering-based end-to-end ER algorithm on nine real-world datasets. The results show that our method significantly improves ER quality, achieving up to 150% higher ACC, reduces API calls by up to 5×, while maintaining competitive cost efficiency compared to the most cost-effective baseline and demonstrating strong scalability (§ 6).

2 RELATED WORK

ER problems can be broadly classified into deduplication (dirty ER) and record linkage (clean-clean ER). The former partitions a single record collection into multiple entity clusters. The latter involves matching records from two separate, typically overlapping but duplicate-free, collections (e.g., two tables) and identifying pairs of records that refer to the same entity [40]. Dirty ER is generally more challenging than clean-clean ER due to inherent data quality issues, such as spelling errors, missing values, and noise. In contrast, clean-clean ER typically involves tables that are already cleaned and standardized [15]. This work focuses on the dirty ER problem, while our solution is also applicable to the clean-clean ER problem by considering the union of records across tables as a single collection. We categorize related work as follows.

PLM and LLM-based Entity Resolution. Before the advent of large language models, pre-trained transformer language models (PLMs) were commonly used for ER tasks [42]. DeepBlocker [73] evaluates various deep learning methods, including SBERT (Sentence BERT) [69] for blocking. Ditto [47] exploits PLMs such as BERT [23], DistilBERT [71], or RoBERTa [49] for clean-clean ER. JointBERT develops a dual-objective training method for BERT, combining binary matching and multi-class classification, to predict both match/non-match decisions and entity identifiers in training pairs [63]. RobEM aims to enhance the robustness of PLM-based entity matching models [68].

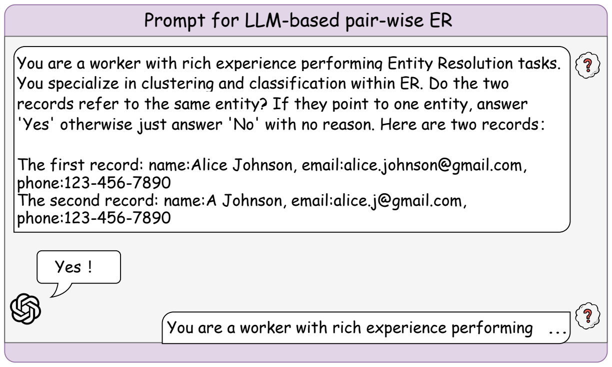
PLM-based ER approaches typically require task-specific labeled data and fine-tuning, which are resource-intensive. More recently, LLMs, a.k.a. foundation models, have demonstrated strong performance in data cleaning and integration tasks, including ER, at a lower cost. This is primarily due to the more straightforward prompt-based interactions without requiring task-specific model retraining or labeled data. State-of-the-art LLM-based ER approaches are discussed in § 1. Among them, [27, 81, 88] are the closest to ours. Both [27, 88] introduce batch prompting, where multiple pairwise questions are packaged into a single batch for the LLM. However, our approach differs by packing multiple records into a set for direct, in-context clustering via the LLM, allowing for more efficient and scalable ER. Furthermore, ComEM [81] employs advanced prompts such as "match", "compare", or "select" to explore interactions across multiple records beyond pairwise questions. However, unlike ours, it does not leverage the direct clustering capacity of LLMs for a record set. As pairwise questioning is a special case of our in-context clustering (with set size = 2), our framework generalizes the state-of-the-art ER approaches, offering improved cost-effectiveness and efficiency.

Besides, our work LLM-CER differs from existing LLM-based ER methods in its comprehensive end-to-end pipeline, which includes blocking/filtering (except for [27]), and addresses critical issues like LLM hallucinations, which are not explicitly handled before.

Crowdsourcing-based Entity Resolution. A key concern in crowdsourced entity resolution (ER) is minimizing the number of questions posed to workers, which directly impacts the overall cost. Transitivity is commonly used to reduce the number of questions in [76, 79]. Closer to our approach, CrowdER [78] develops a clustering-based method where crowd workers are directly tasked with clustering a set of records. Various models for selecting high-quality questions have been developed in [10, 83]. ZenCrowd combines algorithmic and crowdsourcing-based matching techniques within a probabilistic framework [22], while Roomba uses a decision-theoretic approach to select matches to confirm based on their utility [38]. More recent works [33, 75, 80, 86] consider crowd errors in ER tasks. Corleone crowdsources the entire ER workflow, including blocking rules learning [32], and Falcon scales Corleone using RDBMS-style query execution over a Hadoop cluster [20].

Although CrowdER [78] provides record sets to human taskers for direct clustering, there are fundamental differences between their approach and our in-context clustering using LLMs. (1) CrowdER aims to minimize the number of record sets required (equivalently the number of HITs), given a predefined set size, to cover and resolve all uncertain record pairs in a block. In contrast, we adopt a hierarchical approach for record set creation. Unlike CrowdER, which generates all record sets *at once* and covers all uncertain record pairs, our method incrementally generates record sets across multiple layers. In the initial layers, we focus on generating non-overlapping sets that cover all records, *without immediately resolving all uncertain pairs*. As we progress to higher layers, we leverage transitivity and anti-transitivity to *identify and reduce unnecessary record sets*. This approach minimizes the total number of sets, thus reducing the number of LLM API calls and ultimately lowering monetary costs. According to our experimental results over real-world datasets and considering the same set size as well as the same blocking approach (e.g., LSH [24]), the number of record sets (equivalently the number of HITs) needed for CrowdER can be 2-5× higher than the number of record sets (equivalently the number of LLM API calls) required by ours. (2) Our combining approach differs from CrowdER's in several key ways. The initial record sets generated by our algorithm are non-overlapping. Based on clustering results via the LLM, we conduct hierarchical record sets generation and further in-context clustering to merge those clusters. Our hierarchical record sets generation process maintains the optimal record set configuration, it also leverages transitivity and anti-transitivity to reduce the number of LLM API calls. In summary, direct and robust LLM-based clustering is employed both in our matching and combining phases. In contrast, CrowdER allows overlaps of records in different HITs, and their cluster merging process relies on these overlaps to indirectly facilitate merging through transitivity. This could suffer due to human errors as two clusters could be incorrectly merged. (3) Finally, our approach includes *a guardrail mechanism to assess the quality of the LLM's clustering results*. This allows us to identify and discard incorrect clustering results and regenerate better record sets. In contrast, CrowdER assumes that crowdsourcing results are always accurate, it lacks a built-in process to validate or modify the clustering outcomes. These differences ensure that our method not only minimizes the number of record sets (equivalently the number of LLM API calls) but also maintains higher quality of the final ER results.

Filtering and Blocking. Blocking is a critical preprocessing step in ER, aimed at reducing computational overhead by partitioning records into groups and ensuring that only potentially matching records are compared. Blocking methods could be classified as rule-based, sorting-based, and hash-based. Rule-based methods organize tuples by applying fixed keys or decision rules created by experts or derived from heuristics [66]. Sorting-based methods cluster records by rapidly sorting them according to their textual similarities, which are determined using different similarity functions [41]. Hash-based approaches utilize hashing techniques, such as Min-Hashing and Locality-Sensitive Hashing (LSH), to place records into different buckets [24]. Filtering complements blocking by eliminating record pairs that are guaranteed not to match, thus focusing comparisons only on the remaining pairs to improve computational efficiency. Prominent filtering techniques are categorized into prefix-based, partition-based, and tree-based. We refer to [60] for details.


Prompt for LLM-based pair-wise ER

You are a worker with rich experience performing Entity Resolution tasks. You specialize in clustering and classification within ER. Do the two records refer to the same entity? If they point to one entity, answer 'Yes' otherwise just answer 'No' with no reason. Here are two records:

The first record: name:Alice Johnson, email:alice.johnson@gmail.com, phone:123-456-7890
The second record: name:A Johnson, email:alice.j@gmail.com, phone:123-456-7890

Yes !

You are a worker with rich experience performing ...

Figure 2: Example zero-shot prompt for pairwise ER

3 PRELIMINARIES

We discuss the background on the entity resolution (ER) problem (§3.1), followed by how pre-trained LLMs can be adapted for ER in a zero-shot prompting paradigm (§3.2).

3.1 Entity Resolution

An ER algorithm receives an input set of records $\mathcal{R}$ and returns a partition of them: $\mathbb{C} = \{C_1, C_2, \dots, C_n\}$, such that $C_i \cap C_j = \emptyset$ for all i, j, and $\cup_i C_i = \mathcal{R}$. Each C_i is called a *cluster* or a *group* of $\mathcal{R}$ and represents a distinct real-world entity. If two records r_i, r_j refer to the same (different) entity, they are denoted as $r_i = r_j$ ($r_i \neq r_j$).

There are three key steps in entity resolution, *blocking/ filtering*, *matching*, and *combining* [16, 30, 59]. Blocking and filtering aim to separate almost dissimilar records and keep similar records in the same block to reduce the number of comparisons in the subsequent matching step and make the algorithm more efficient [61]. Matching determines whether records in the same block refer to the same entity. Finally, combining creates the final clusters of entities by inferring indirect matching relations following the matching step.

In pairwise ER, a pairwise similarity function is used over each pair of records to find the matching pairs. In this case, each pair is referred to as a *question* posed to the similarity function. An ER algorithm generally obeys *transitivity* and *anti-transitivity* rules, which can be employed in the matching and combining phases to further reduce the number of questions [5].

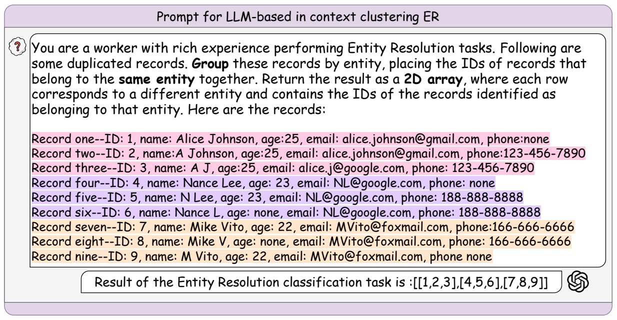
Transitivity. Given three records r_1, r_2, and r_3, if $r_1 = r_2$ and $r_2 = r_3$, then we have $r_1 = r_3$.

Anti-transitivity. Given three records r_1, r_2, and r_3, if $r_1 = r_2$ and $r_2 \neq r_3$, then we have $r_1 \neq r_3$.

A clustering $\mathbb{C}$ of the input set of records $\mathcal{R}$ is transitively closed. For instance, if a pairwise similarity function decides $r_1 = r_2$ and $r_2 = r_3$, then we can infer $r_1 = r_3$ without employing the similarity function on the pair (r_1, r_3). This reduces the number of questions.

3.2 LLM: Adaptation Techniques

Deep neural network–based models have long been effective as similarity functions for record matching, typically taking a pair of records as input and predicting whether they refer to the same real-world entity [36, 43, 57]. However, these methods demand large quantities of high-quality, task-specific labeled data, which can be



Prompt for LLM-based in context clustering ER

You are a worker with rich experience performing Entity Resolution tasks. Following are some duplicated records. **Group** these records by entity, placing the IDs of records that belong to the **same entity** together. Return the result as a **2D array**, where each row corresponds to a different entity and contains the IDs of the records identified as belonging to that entity. Here are the records:

Record one--ID: 1, name: Alice Johnson, age:25, email: alice.johnson@gmail.com, phone:none
Record two--ID: 2, name:A Johnson, age:25, email: alice.johnson@gmail.com, phone:123-456-7890
Record three--ID: 3, name: A J, age:25, email: alice.j@google.com, phone: 123-456-7890
Record four--ID: 4, name: Nance Lee, age: 23, email: NL@google.com, phone: none
Record five--ID: 5, name: N Lee, age: 23, email: NL@google.com, phone: 188-888-8888
Record six--ID: 6, name: Nance L, age: none, email: NL@google.com, phone: 188-888-8888
Record seven--ID: 7, name: Mike Vito, age: 22, email: MVito@foxmail.com, phone:166-666-6666
Record eight--ID: 8, name: Mike V, age: none, email: MVito@foxmail.com, phone: 166-666-6666
Record nine--ID: 9, name: M Vito, age: 22, email: MVito@foxmail.com, phone none

Result of the Entity Resolution classification task is :[[1,2,3],[4,5,6],[7,8,9]]

Figure 3: Example zero-shot prompt for in-context clustering

costly to obtain. In contrast, large language models (LLMs)—pre-trained on massive text corpora via self-supervision—have recently demonstrated strong zero-shot performance on ER matching tasks [27, 37, 44, 55, 65, 81]. Their extensive prior knowledge and superior classification capabilities [9, 13, 72] make them well suited to entity disambiguation. Figure 2 illustrates a typical LLM prompt for pairwise ER. Our approach adopts a purely *zero-shot* strategy: rather than fine-tuning or providing in-context examples, it relies entirely on the LLM's internal representations to assess record similarity. Consequently, the underlying similarity function remains latent, rather than implemented as an explicit, hand-crafted metric.

4 PROBLEM: IN-CONTEXT CLUSTERING

We first introduce our novel record set for in-context clustering and its key factors (§4.1), followed by experiments on how these factors impact clustering on individual record sets (§4.2). Based on these results, we formally define the end-to-end ER problem (§4.3).

4.1 Record Set and Key Factors

Different from pairwise input schemes used by recent LLM-based ER solutions [27, 44], we combine the advantages of LLMs and innovatively propose to input a record set. In particular, we pack a number of records in a set as the input prompt and ask the LLM to output a clustering of the records in this set. Such an in-context clustering scheme can make good use of the LLM's ability to process batch data [88], as well as cluster text data preserving semantic similarity [89]. Direct clustering of larger record sets reduces the number of questions to be prepared, subsequently minimizing the number of LLM API calls, the number of LLM tokens, and thereby obtaining high-quality answers at a lower cost and time.

Given a set of records as input to the LLM, the goal is to partition the records into clusters. We refer to this paradigm as "*in-context clustering*". Three key factors may influence the performance of the LLM within this clustering-based scheme: set size, set diversity, and set variation. The definitions are given below.

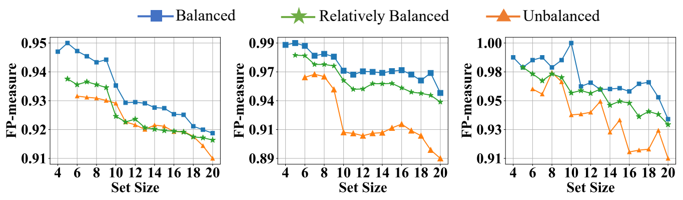
i) Set size: the number of records in an input set.

ii) Set diversity: the number of distinct entities (i.e., clusters) within an input set.

iii) Set variation: the variability in cluster sizes within an input set, quantified by the **coefficient of variation**:

$$variation(S) = \frac{\sigma(S)}{\mu(S)} \tag{1}$$

where $\sigma(S)$ and $\mu(S)$ denote the standard deviation and mean of the cluster sizes, respectively, in the input set S. This metric measures the degree of variability in cluster sizes relative to the mean.



(a) Music 20K (b) Cora (c) Alaska

Figure 4: Clustering performance vs. set size and variation

Additionally, our follow-up experimental results demonstrate the importance of *ordering similar records sequentially inside a record set*, this enhances an LLM's in-context clustering performance by facilitating better context differentiation [9]. Similar findings were also demonstrated by prior RAG work [50], which shows that ordering semantically similar sentences together improves LLM output.

Figure 3 presents an example of an LLM prompt for our novel in-context clustering-based entity resolution. Instead of pairwise ER, the prompt explicitly asks to cluster the records in an input set.

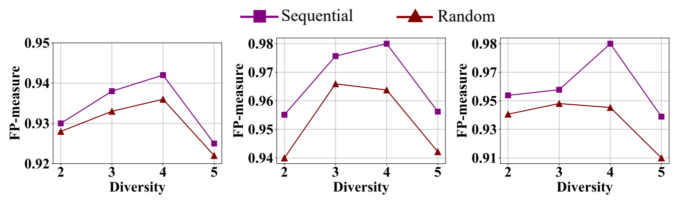
Example 3. *As shown in Figure 3, we use an LLM to group a record set containing 9 records, thus the* ***set size*** *is 9. The* ***set diversity*** *of this record set is 3, since it contains three distinct entities (clusters), which are represented by three different colors. The mean of cluster sizes in this record set is* $(3+3+3)/3 = 3$, *while the standard deviation of cluster sizes in the record set is* $\sqrt{\frac{(3-3)^2+(3-3)^2+(3-3)^2}{3}} = 0$, *indicating a* ***set variation*** *of 0, i.e., no variation in relation to three cluster sizes.*

4.2 Empirical Evaluations for the Key Factors

We examine the impact of varying **set size** (S_s), **set diversity** (S_d), and **set variation** (S_v) on the LLM's performance for in-context clustering of individual record sets. We further explore how the order of records in a set affects the LLM's performance by implementing: **sequential record order**, where records belonging to the same entity are placed consecutively in the record set; and **random record order**, where the records are randomly placed. These analyses help us identify the *optimal information density and structure* that LLMs can handle within a single API call, hence *guiding our design* for creating effective record sets and fully utilizing the knowledge provided by blocking techniques (§ 5).

We define three levels of set variations (S_v) to represent different degrees of imbalance within the record sets: **balanced** ($S_v < 0.3$), **relatively balanced** ($0.3 \leq S_v \leq 0.7$), and **unbalanced** ($S_v > 0.7$). For each fixed S_s, S_d, S_v, and record ordering, we uniformly at random select 200 questions (i.e., record sets) to query the LLM, provided that the total number of questions available under the given parameter settings exceeds 200. The ground truth for the questions corresponds to the correct clustering of the records within each record set. More detailed experimental settings, evaluation metrics, and dataset descriptions can be found in § 6.

Figure 4 illustrates the in-context clustering performance of the LLM on individual record sets under varying S_s and S_v. The results indicate that LLM performance *consistently improves as set variation decreases*, suggesting that clustering is more accurate when records are evenly distributed among entities within a set. This trend holds across all datasets and evaluation metrics, regardless of the set size. Moreover, the LLM *maintains relatively stable and robust performance* across balanced, relatively balanced, and unbalanced settings



(a) Music 20K (b) Cora (c) Alaska

Figure 5: Clustering performance vs. set diversity, ordering

as the set size increases from 4 to 9. However, a marked decline in clustering accuracy is observed once the set size *exceeds 9 or 10*. This degradation can be attributed to the increased cognitive load placed on the model: larger sets require the LLM to process and compare more records simultaneously, which dilutes semantic focus and heightens the risk of clustering errors. Additionally, managing coherent contextual relationships across more records becomes increasingly challenging, further reducing accuracy. Across different set variation settings, the trends in FP-measure *remain largely consistent*, as this metric reflects the alignment between predicted clusters and ground truth labels. Taken together, these results suggest that *the optimal configuration for in-context clustering occurs at a set size of 9*, and that *minimizing variation within each set* is key to maximizing LLM performance.

Fixing the optimal values of S_s and S_v based on previous experiments, we further investigate the impact of S_d and the ordering of records within record sets on the performance of LLM-based in-context clustering (Figure 5). We notice that when S_d is set to 4, the in-context clustering performance of the LLM is better across all datasets in terms of FP-measure compared to other diversity settings. Thus, we consider $S_d = 4$ *as our optimal diversity setting*. With 4 entities, the clustering task provides enough granularity to differentiate entities while avoiding excessive sparsity or overlap in record distribution. In contrast, fewer entities (e.g., 2 or 3) might result in overly homogeneous clusters that fail to capture subtle differences, while a larger number of entities (e.g., 5) might introduce higher complexity and noise, negatively impacting the LLM's ability to identify distinct clusters accurately. Meanwhile, it is observed that the LLM achieves better in-context clustering performance when the *sequential record order* is used. This improvement can be attributed to the sequential arrangement of similar records, which likely enhances the coherence of contextual information presented to the LLM. By maintaining entity-related records in close proximity, this ordering reduces potential context-switching overhead and provides a more structured input sequence [50], thereby enabling the model to better discern subtle patterns and relationships essential for accurate clustering.

In conclusion, for the LLM's in-context clustering performance, our optimal configuration includes a set size of 9, a balanced distribution (S_v close to 0), a set diversity of 4, and the sequential record order. Larger set sizes should be avoided, as the LLM's performance under extremely unbalanced ratios deteriorates rapidly beyond this point. Conversely, smaller set sizes should also be avoided, as maximizing the set size is crucial for reducing API call frequency, improving efficiency and minimizing token usage and cost. More analyses about the choice of key factors can be found in § 6.3.

4.3 Problem Statements for End-to-end ER

We are now ready to define our main problems.

Problem 1. (Next Record Set Selection). *This problem is relevant for an LLM's in-context clustering during the entity matching phase. Given a set of records $\mathcal{R}$, the current clustering result $\mathbb{C}$, and predefined constraints on the set size (S_s), diversity (S_d), and variation (S_v), the Next Record Set Selection problem identifies a subset of records $\mathcal{R}^* \subseteq \mathcal{R}$ satisfying the constraints $\langle S_s, S_d, S_v \rangle$ with the objective of minimizing the number of LLM API calls by leveraging the identified transitivity and anti-transitivity relationships.*

Problem 2. (Cluster Combination). *Given the current clustering result $\mathbb{C}$ and the LLM's in-context clustering output $\mathbb{C}^*$ derived from a record set $\mathcal{R}^*$, the Cluster Combination problem updates $\mathbb{C}$ by combining the clusters in $\mathbb{C}$ based on the merge decisions of the LLM over $\mathbb{C}^*$, leveraging the identified transitivity and anti-transitivity relationships. This problem is relevant in combining phase to gradually refine the LLM's clustering outputs and obtain the final ER result.*

The number of record sets generated by our algorithm determines the number of LLM API calls. Since each LLM API call roughly consists of the same number of tokens (e.g., see Figure 3), and an LLM's monetary cost and inference time are determined by the total number of input and output tokens, minimizing the number of LLM API calls also reduces the number of LLM tokens, monetary cost, as well as the end-to-end ER time. We discuss our solutions in the following section. In particular, the next record set selection is considered in §5.2, while cluster combination is detailed in §5.3. Notice that the existing pairwise ER scheme is a *special case* of our in-context clustering-based ER paradigm when the record set size $S_s = 2$. Therefore, our problems and solutions generalize the state-of-the-art ER approaches.

5 SOLUTION: END-TO-END ER

We present an overview of our solution, LLM-powered Clustering-based ER (LLM-CER), before delving into details. Our key technical contributions are: (1) an effective Next Record Set Creation (NRS) algorithm for constructing optimal record sets for in-context clustering from initial blocks generated by standard blocking techniques.; (2) a guardrail technique, Misclustering Detection Guardrail (MDG) for assessing and subsequently improving the accuracy of the LLM's in-context clustering outputs; and (3) an efficient heuristic algorithm, Cluster Merge (CMR) for merging the in-context clustering outputs. Putting them together, we systematically explore the design space of clustering-based entity resolution powered by LLMs.

As illustrated in Figure 6, LLM-CER begins with blocking to generate initial candidate blocks (Section 5.1), which are then processed by the NRS algorithm (Algorithm 1) to construct record sets for in-context clustering by an LLM (Section 5.2). The clusterings are assessed and refined by the MDG algorithm (Algorithm 2), which identifies and corrects likely misclusterings. Improved clusters are then hierarchically merged using the CMR algorithm (Algorithm 3), which repeatedly invokes LLM-based clustering over newly formed record sets while leveraging transitivity and anti-transitivity (Section 5.3). This process continues until a termination condition is met, producing the final entity resolution result (Section 5.4).

5.1 Filtering and Blocking

We begin by introducing the filtering and blocking strategies used as a pre-processing step in the ER pipeline. Given a collection of records $\mathcal{R}$, a naïve approach compares all pairs, incurring a computational complexity of $O(|\mathcal{R}|^2)$. To mitigate this, filtering and

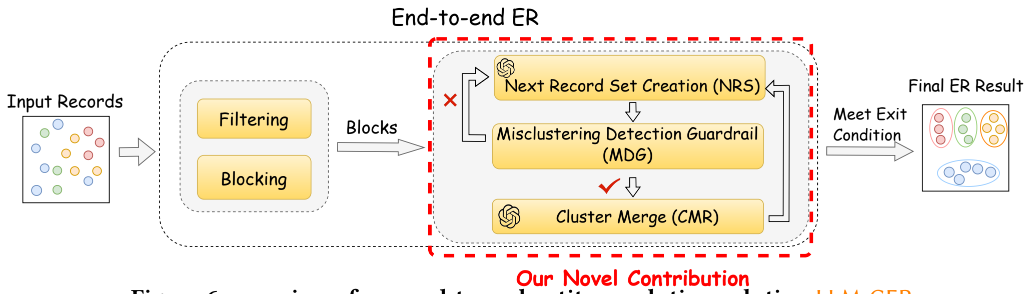


Figure 6: overview of our end-to-end entity resolution solution LLM-CER

blocking techniques are employed [35, 52, 62]. Blocking groups similar records into blocks, while filtering typically applies similarity joins to identify candidate matches for each record. Although not the focus of our technical contributions, this work is the first to explore their trade-offs in the context of (1) LLM-based ER, which leverages large language models for ER [27, 44], and (2) clustering-based ER, where resolution is achieved by iteratively clustering subsets of records rather than pairwise comparisons. We consider the following filtering and blocking strategies.

Filtering-based Block Creation. Similarity joins identify records with similarity scores above a threshold using metrics like Jaccard similarity [7, 8]. While effective for detecting near-duplicates, pairwise comparisons incur $O(|\mathcal{R}|^2)$ complexity, making filtering computationally expensive. To address this, positional filtering [85] prunes candidate records by leveraging token ordering, reducing complexity to $O(|\mathcal{R}| \cdot k)$ when k candidates are allowed per record. The similarity threshold b_t is set empirically by maximizing the F1-score of clustering on a validation dataset [29], iterating thresholds from 0.05 to 0.95 in 0.05 increments. When ground truth is unavailable, we exploit LLM-based clustering results over a certain number of record subsets as ground truth.

Locality Sensitive Hashing-based Blocking. Distributed representations effectively capture semantic similarities (e.g., "p53" and "cancer") in ER tasks [31]. While cosine similarity can measure embedding proximity, its $O(|\mathcal{R}|^2)$ complexity becomes impractical for large datasets. Locality-sensitive hashing (LSH) [24] hashes similar embeddings into identical buckets with high probability, reducing complexity to $O(|\mathcal{R}| \cdot k)$, where k is the number of buckets. However, the stochastic hash functions may co-locate dissimilar records, introducing false positives. To mitigate this issue, we retain only pairs with similarity exceeding a threshold b_t to purify the initial blocks, where b_t is determined via the aforementioned approach.

Canopy Blocking [51]. Two thresholds b_s and m_s, with $b_s \geq m_s$, are utilized to balance efficiency and accuracy. A computationally inexpensive metric (e.g., inverted index-based edit distance on a single attribute) assigns record pairs with similarity $> b_s$ to the same block. Pairs with similarity $> m_s$ are grouped into overlapping canopies. Within these canopies, a refined metric (e.g., Jaccard similarity across all attributes) computes pairwise similarities. Matching pairs trigger transitive block merging: If records from two blocks match, the blocks are combined iteratively until convergence. While initial canopy formation retains quadratic complexity, its reliance on lightweight metrics makes it orders of magnitude faster than refined similarity computations [51]. We tune b_s and m_s empirically on validation datasets, or when ground truth is unavailable, derive labels via LLM-based clustering (as described earlier).

Remark. We employ LSH as the default blocking method based on empirical results (§6.4). We adopt the above filtering and blocking techniques for their simplicity and compatibility with our framework. As our primary technical contributions lie in the in-context clustering for matching (§5.2) and the merging strategy for combining clusters (§5.3), we leave the integration of advanced blocking methods—such as progressive blocking [28]—to future studies.

5.2 In-context Clustering of a Record Set

We develop novel in-context clustering for the matching phase.

Next Record Set Creation Algorithm. Consider the records $\mathcal{B}_{remain}$ in a block that are not used in forming a record set yet. We design an algorithm NRS (Algorithm 1) to create the next record set with $\mathcal{B}_{remain}$. If $|\mathcal{B}_{remain}|$ is smaller than a predefined set size (S_s), the next record set is created by grouping similar records sequentially, which improves an LLM's in-context clustering performance by facilitating better context differentiation, as shown in our experiment (§4.2) (Lines 2-6). Otherwise, the next record set generation is optimized based on the empirical findings from §4.2. Our results show that an LLM's in-context clustering accuracy depends on the record set design. In particular, we select an optimal record set size S_s, diversity S_d, and variation S_v based on experimental results. We then apply the elbow method and k-means [19] to perform a preliminary clustering of records in $\mathcal{B}_{remain}$, while also assessing its diversity k. Next, we construct the next record set while ensuring the set size constraint S_s. For the record set, we minimize S_v as much as possible and aim to meet the S_d requirement (Lines 8-17).

Experimental results indicate that the optimal values are $S_s = 9$ and $S_d = 4$, while LLM-based in-context clustering achieves better performance when S_v is minimized, ideally approaching zero.

Mitigating In-context Clustering Errors of LLMs. In practice, LLMs are prone to hallucination [34], leading to seemingly plausible, yet factually incorrect contents. As a result, LLM-based in-context clustering outcomes for record sets may not be entirely reliable. To ensure high accuracy of the output, we further devise a misclustering detection guardrail (MDG) algorithm to verify the clustering results of the LLM, as detailed in Algorithm 2. We first define some key terms.

Definition 1. (Inter- and Intra-cluster Similarity) *Given a similarity function $F(\cdot)$ to measure the similarity between record pairs and a known record r, the intra-cluster similarity of r is defined as the minimum similarity between r and other records within the same cluster as r. The inter-cluster similarity of r is defined as the maximum similarity between r and records from other clusters.*

Specifically, for any cluster from the in-context clustering output, if the intra-cluster similarity of a record inside it is lower than its inter-cluster similarity, Algorithm 2 indicates the record to be

Algorithm 1 Next record set creation (NRS)

```
Input: remaining records in a block $\mathcal{B}_{remain}$, $S_s$, $S_d$, $S_v$
Output: record set $\mathcal{R}_{set}$
1: $\mathcal{R}_{set} \leftarrow \emptyset$
2: if $|\mathcal{B}_{remain}| \leq S_s$ then
3:     $r_{pre} \leftarrow$ first element in $\mathcal{B}_{remain}$
4:     while $|\mathcal{B}_{remain}| > 0$ do
5:         add $r_{pre}$ to $R_{set}$ and remove it from $\mathcal{B}_{remain}$
6:         $r_{pre} \leftarrow$ update $r_{pre}$ w/ its most similar record from $\mathcal{B}_{remain}$
7: else
8:     $k \leftarrow$ compute diversity of $\mathcal{B}_{remain}$ using the elbow method
9:     $\mathbb{B}_{remain,k} \leftarrow$ perform $k$-means on $\mathcal{B}_{remain}$
10:    $target_{size} \leftarrow \lfloor S_s/S_d \rfloor$
11:    for $\mathcal{B}_{remain,i}$ in $\mathbb{B}_{remain,k}$ do
12:        if $|\mathcal{R}_{set}| < S_s$ and $|\mathcal{B}_{remain,i}| \geq target_{size}$ then
13:            select $target_{size}$ records from $\mathcal{B}_{remain,i}$ and add to $\mathcal{R}_{set}$
14:            delete those records from $\mathcal{B}_{remain}$
15:    while $|\mathcal{R}_{set}| < S_s$ do
16:        $r_{set} \leftarrow$ find record in $\mathcal{B}_{remain}$ to least increase in $S_v$ (via Eq. 1)
17:        add $r_{set}$ to $\mathcal{R}_{set}$ and delete from $\mathcal{B}_{remain}$
18:    order similar records together in $\mathcal{R}_{set}$ (similar to Lines 3-6)
19: return $\mathcal{R}_{set}$
```

Algorithm 2 Misclustering Detection Guardrail (MDG)

```
Input: In-context clustering result $\mathbb{C}_{set}$ of a record set
Output: whether in-context clustering result is acceptable ($True/False$)
1: for cluster $C_i$ in $\mathbb{C}_{set}$ do
2:     for record $r_j$ in $C_i$ do
3:         if intra-cluster sim. of $r_j$ < inter-cluster sim. of $r_j$ then
4:             return $False$
5: return $True$
```

misclustered; and hence, the result is not acceptable. When computing similarity between records or clusters, our choice of similarity function is closely linked to the block creation method. For example, when using Filtering-based Block Creation, we employ Jaccard similarity to measure record similarity. In contrast, with LSH-based Block Creation, we assess similarity through cosine similarity applied to record embeddings. By default, we use cosine similarity for record embeddings unless otherwise specified. The overall time complexity of the MDG algorithm is $O(S_s^2)$, where S_s is typically 9. Given that S_s is small and the computation is highly parallelizable, the actual runtime overhead remains low in practice.

Record Set Regeneration. For each misclustered record r, our end-to-end algorithm LLM-CER identifies the cluster with the highest inter-cluster similarity to r, and relocates it immediately after that cluster within the record set. This targeted adjustment leaves all other records unchanged and aims to bring semantically similar records closer together inside the record set, resulting in an overall time complexity of $O(S_s)$. As shown in §4.2, LLM's clustering accuracy improves when records from the same entity appear consecutively. Next, we conduct in-context clustering with this more sequentially-ordered record set, enhancing the LLM's performance significantly, which is evident in our empirical results (§ 6.4).

5.3 Hierarchical Cluster Merging

Leveraging the optimal set size $S_s = 9$ as stated earlier, each block is divided into several record sets (e.g., a block of size 27 is split into 3 record sets). Each record set is then in-context clustered

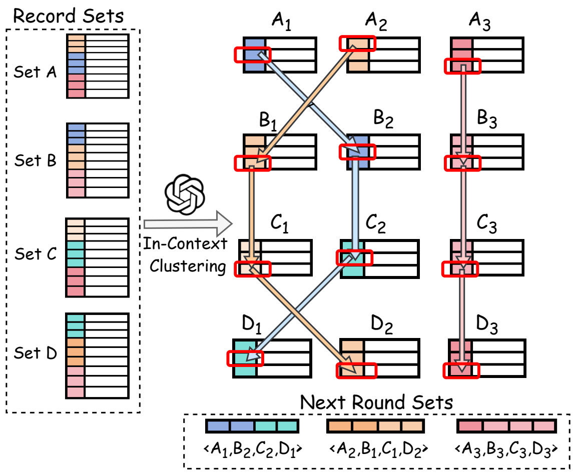


Figure 7: An example of the next round of record sets creation. For simplicity, consider that each of the record sets A, B, C, D generates three in-context clusters: $A = \{A_1, A_2, A_3\}$, $B = \{B_1, B_2, B_3\}$, etc. Assume that $S_s \geq 4$ and $S_d = 1$ for the sake of brevity. Then, the next round of record sets should pack the most similar clusters from A, B, C, and D, forming three new sets for the next round as follows: $\{A_1, B_2, C_2, D_1\}$, $\{A_2, B_1, C_1, D_2\}$, and $\{A_3, B_3, C_3, D_3\}$. Each cluster appears exactly once in the next round of record sets to minimize the overall LLM API calls. Clusters from the same record set (e.g., both A_1 and A_2) are not packed together in the same record set of the next round owing to anti-transitivity.

independently by the LLM. For example, if record set A is clustered into subsets $\{A_1, A_2, A_3\}$, record set B into $\{B_1, B_2\}$, and record set C into $\{C_1, C_2, C_3, C_4, C_5\}$, it becomes necessary to merge subsets across record sets to obtain the final partition.

To achieve this, we note that clusters within the same record set (e.g., A_1 and A_2) are already disjoint (i.e., belong to different entity groups) due to the LLM's in-context clustering and hence exhibit non-transitive relationships. However, clusters from different record sets (e.g., A_1 and B_1) may belong to the same entity group. Merging clusters across different record sets can be formulated as a K-dimensional maximum matching problem (analogous to a 3-dimensional maximum matching problem, which is proven to be **NP**-Hard [17]), where K is the number of record sets.

Problem 3 (Cluster Merging). *Given K record sets and their in-context clustering outputs, the Cluster Merging problem treats each output cluster as a "new" individual record for the next round and packs them optimally to construct record sets for the LLM's next round of in-context clustering. The problem is subject to the following conditions: (1) Each cluster (i.e., the corresponding "new" individual record) is selected exactly once in the next round of record sets to minimize the total number of LLM API calls. (2) Clusters from the same record set are never packed together in the same record set of the next round owing to anti-transitivity. (3) The next round of record sets should also ensure the set size constraint S_s, set diversity S_d, and minimize S_v as much as possible.*

The complexity of evaluating all combinations in the cluster merging problem is $O(K \cdot n!)$, where n is the average number of clusters per record set. This results in an exponential time cost, making it impractical for real-world applications with large blocks and multiple record sets.

To address this issue, we propose a heuristic Cluster Merge (CMR) algorithm (Algorithm 3) that avoids exhaustive enumeration. Specifically, for each cluster of a given record set, we identify its most similar cluster from the next record set based on a defined similarity measure. For instance, consider two record sets A and B with their in-context clustering results $A = \{A_1, A_2, A_3\}$, $B = \{B_1, B_2\}$. If A_1 is most similar to B_2, these two clusters are packed together. This process is repeated iteratively across all record sets (e.g., finding a cluster of the next record set C that is most similar to the group A_1B_2), gradually forming a new record set for the next round of in-context clustering, while satisfying S_s, S_d, and S_v requirements. Figure 7 shows an example of our cluster merging procedure that constructs record sets for the next round of clustering.

For simplicity, assume $K \leq S_s$, where K denotes the number of record sets in the current round, and S_s is the optimal record set size. For ease of understanding, we also assume that each of these record sets generates n in-context clusters by the LLM. Algorithm 3 outlines the process of constructing record sets for the next round of in-context clustering. Given K record sets, the algorithm first generates clusters for each record set using the LLM, then replaces each cluster with a representative "new" record. This replacement is possible because the records within a cluster represent the same entity and satisfy transitivity. The representative record can either be uniformly chosen at random or selected as the one with the smallest distance from the average embedding of the records in that cluster (Lines 1-3). Then, it partitions the K record sets into groups of size $\lceil K/S_d \rceil$, iterating over these groups to construct a record set $\mathcal{R}_{next}$ for the next round (Lines 5-6). For each group, it selects an unselected cluster C from the first record set of the group and uses its representative record as the initial record (Lines 7-8). Subsequently, for each remaining record set in the group, it identifies the most similar unselected cluster C' to the previously selected cluster C and adds its representative record to $\mathcal{R}next$ (Lines 9-11). Finally, the resulting record set is returned for the next round of in-context clustering (Line 12). This process also ensures that (1) similar elements of $\mathcal{R}next$ are sequentially ordered; and (2) we meet the S_d requirement, while minimizing S_v as much as possible.

Considering an average of n clusters from a record set, the complexity of comparing clusters between two record sets is $O(n^2)$. Since there are K record sets in the current round, the total complexity of CMR algorithm is $O(K \cdot n^2)$ in the current round. This represents a significant reduction from the original exponential complexity $O(K \cdot n!)$. The maximum value of n can be up to the record set size, which is typically 9 in our experiments. This is significantly smaller than the total number of records $|\mathcal{R}|$, leading to a reduced complexity in real-world usage. Moreover, the total hierarchy layers remain limited as observed in our experiments and summarized in Table 3 (§6.2).

5.4 End-to-End ER Solution

Algorithm 4 presents our end-to-end entity resolution framework outlining the key steps, given the input set of records $\mathcal{R}$. The algorithm begins by applying blocking functions to $\mathcal{R}$, generating initial set of blocks $\mathbb{B}$ (§5.1), after which we build a disconnected graph based on it to keep track of similar records. For each block in $\mathbb{B}$, record sets are generated using the NRS algorithm (Algorithm 1), they are in-context clustered leveraging an LLM, with acceptable outputs verified by the MDG algorithm (Algorithm 2) given in §5.2. Subsequently, the clustering outputs of these record sets are merged hierarchically using the CMR algorithm (Algorithm 3) and next rounds of in-context clustering (§5.3).

Algorithm 3 Cluster Merge (CMR): Depicting the creation of one record set for the next round

```
Input: $K$ ($\leq S_s$) record sets $\{\mathcal{R}_1, \dots, \mathcal{R}_K\}$ each with $n$ clusters; $S_s, S_d, S_v$
Output: one record set $\mathcal{R}_{next}$ for the next round with size $K$
1:  for $i \leftarrow 1$ to $K$ do
2:      $\mathbb{C}_i \leftarrow n$ output clusters of record set $\mathcal{R}_i$
3:      replace each cluster $C \in \mathbb{C}_i$ by a representative record $r_C \in C$
4:  $\mathcal{R}_{next} \leftarrow \phi$
5:  for $j \leftarrow 1$ to $S_d$ do
6:      $w \leftarrow (j-1) \cdot \lceil K/S_d \rceil + 1$
7:      select a previously unselected cluster $C \in \mathbb{C}_w$
8:      insert $C$ (i.e., its representative record $r_C$) into $\mathcal{R}_{next}$
9:      for $i \leftarrow (j-1) \cdot \lceil K/S_d \rceil + 2$ to $\min\{j \cdot \lceil K/S_d \rceil, K\}$ do
10:         find cluster $C' \in \mathbb{C}_i$, previously unselected & most similar to $C$
11:         Insert $C'$ (i.e., its representative record $r_{C'}$) into $\mathcal{R}_{next}$
12: return $\mathcal{R}_{next}$
```

Algorithm 4 End-to-end ER

```
Input: A set of records $\mathcal{R}$
Output: final partition $\mathbb{C}$ of $\mathcal{R}$
1:  $\mathbb{B} \leftarrow$ apply blocking functions on $\mathcal{R}$ and get initial blocks
2:  generate record sets for each block $\mathcal{B} \in \mathbb{B}$ using NRS algorithm
3:  in-context clustering of record sets with LLM
4:  check correctness of in-context clustering by MDG algorithm
5:  record sets regeneration to improve in-context clustering if needed
6:  while True do
7:      generate record sets for next round using Cluster Merge algorithm
8:      in-context clustering of record sets with LLM
9:      check correctness of in-context clustering by MDG algorithm
10:     record sets regeneration to improve in-context clustering
11:     if Exit condition then
12:         Break
13: Use each singleton cluster to reconstruct the final partition $\mathbb{C}$
14: return $\mathbb{C}$
```

Exit Condition. We keep generating hierarchical record sets for in-context clustering and subsequent cluster merging. The process continues until no more clusters can be merged owing to anti-transitivity. Here, anti-transitivity is identified by in-context clustering of the record sets. In particular, consider a current round when the in-context clustering outputs only singleton clusters, i.e., each cluster having only one element from the current round. The exit condition is thus satisfied. A naive method for the "final check" is to conduct pairwise comparisons of all singleton clusters to ensure that, indeed, no more merging is feasible, which costs quadratic time in the number of singleton clusters. During this final check, we, however, adopt a more efficient method by packaging multiple singleton clusters in a record set to reduce the number of LLM API calls, which is discussed in the following theoretical analysis.

Finally, we use each singleton cluster to reconstruct the full partition $\mathbb{C} = \{C_1, \dots, C_n\}$ and complete the ER process.

Theoretical Analysis. We analyze the number of LLM API calls required by our LLM-CER method, focusing on a single block of m records. To highlight the efficiency benefits, we consider two extreme cases that capture the boundaries of practical scenarios. **(1)** *All records belong to same entity.* For each hierarchy

Table 1: Dataset statistics: Rec. stands for records, ent. represents entities, attr. denotes attributes, E_d indicates entity dispersion[11] (=#Rec./#Ent.). The 'T' in Attr. Types denotes 'Textual', 'N' represents 'Numeric', 'C' indicates 'Categorical', the number after the attribute category indicates the count of attributes of the corresponding type.

Datasets	Domain	#Rec.	#Ent.	E_d	#Attr. per Rec.	Attr. Types
Alaska	Product	12k	1.48k	≈ 8	9	T(9)
AS	Geo	2.26k	0.33k	≈ 7	1	T(1)
Song	Music	4.85k	1.19k	≈ 3	7	T(4), N(3)
Music-20K	Music	19.3k	10k	≈ 2	6	T(4), N(1), C(1)
DBLP-Google	Citation	7.63k	2.35k	≈ 3	4	T(3), N(1)
Cora	Citation	1.29k	0.11k	≈ 12	12	T(12)
Cite-seer	Citation	9.13k	2.49k	≈ 4	6	T(4), N(1), C(1)
Amazon-Google	Software	2.16k	0.99k	≈ 2	3	T(2), N(1)
Walmart-Amazon	Electronics	1.81k	0.85k	≈ 2	5	T(3), N(1), C(1)

level, the number of records is reduced by a factor of the record set size (S_s). Hence, the total number of LLM API calls equals $(\lceil m/S_s \rceil + \lceil \lceil m/S_s \rceil / S_s \rceil + \ldots + 1) \approx \lceil m/(S_s - 1) \rceil$. For larger S_s, the series converges quickly, and the total number of LLM API calls reduces significantly, justifying the superiority of our in-context clustering approach over pairwise comparisons (whose $S_s = 2$) in the end-to-end ER. Specifically in this case, the API calls reduce from $O(m)$ in pairwise with transitivity to only $O(\frac{m}{S_s})$ in LLM-CER.
(2) *All records belong to different entities.* For hierarchy level 0, there are $p = \lceil m/S_s \rceil$ record sets, resulting in p LLM API calls. For simplicity, assume $m = p \cdot S_s$, i.e., each record set has exactly S_s records. Since all records are unique, each record forms a singleton cluster, satisfying the clustering exit condition immediately. However, a final disambiguation step is still needed to verify cross-set matches. This involves comparing each record to all others outside its record set, resulting in $\frac{m(m-S_s)}{2}$ comparisons. Using our in-context framework, we batch these comparisons into new record sets. Each set of size S_s enables $\frac{S_s(S_s-1)}{2}$ pairwise comparisons, meaning we require $\frac{m(m-S_s)}{S_s(S_s-1)}$ LLM API calls for the final check. Including level 0, the total number of calls is $\frac{m}{S_s} + \frac{m(m-S_s)}{S_s(S_s-1)} = \frac{m(m-1)}{S_s(S_s-1)}$, which again is much smaller than the $\frac{m(m-1)}{2}$ pairwise calls needed in conventional methods. This yields a non-trivial reduction by a factor of $O(S_s^2)$. Since all practical scenarios are between these two extreme cases, our theoretical analysis demonstrates that our design significantly reduces the LLM API calls compared to pairwise ER.

6 EXPERIMENTAL RESULTS

In this section, we empirically evaluate our proposed algorithms as well as the competing approaches. All methods are implemented in Python and executed on a Linux server with 2.20 GHz CPU and 128GB of RAM. Our code and datasets are available at [4].

6.1 Experimental Setup

Datasets. We utilize nine real-world entity resolution (ER) datasets spanning various domains, including e-commerce products, academic citations, geography, and music. Details of these datasets are provided in Table 1. Specifically, CORA is a text-based dataset comprising duplicate references to scientific publications. For this study, we use the structured CSV version provided by [58], with the raw data accessible online[2]. *Alaska* [18] is an e-commerce product dataset collected from multiple websites and previously featured in the SIGMOD 2020 programming contest [21]. *DBLP-Google [53]* and *Citeseer-DBLP* belong to the citation domain, containing bibliographic duplicates sourced from DBLP, Google Scholar, and Citeseer. The *Song* dataset represents the music domain and includes music information from various platforms, with variations in attributes such as duration and release year. Both *Song* and *Citeseer-DBLP* can be found here[3]. The *AS* and *Music 20K* datasets are widely recognized for Dirty ER benchmarks [70]. Lastly, we consider *Amazon-Google* [53] and *Walmart-Amazon* [53] datasets, originating from the Software and Electronics domains, respectively. These domains are more specialized and require deeper domain knowledge, making the corresponding ER tasks more complex than others.

Evaluation Metrics. We select several widely-used metrics to evaluate the in-context clustering results, as well as the final ER clustering results. These metrics are FP-measure (a variant of the F-measure), Accuracy (ACC), Normalized Mutual Information (NMI), and Adjusted Rand Score (ARI) [2, 12, 82, 87]. All experiments are repeated three times to ensure robustness, and we report the average values over three runs. Due to space limitations, we present results for ACC and FP-measure in the main paper. The complete experimental results are available in the full version [4].

The definition of ACC is as follows: Let $\mathbb{Y} = \{\mathcal{Y}_1, \mathcal{Y}_2, \ldots, \mathcal{Y}_m\}$ represent the ground truth clusters and $\mathbb{X} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n\}$ denote the predicted clusters. The total number of records in both sets is $|\mathcal{R}| = \bigcup_i |\mathcal{X}_i| = \bigcup_j |\mathcal{Y}_j|$. The accuracy (ACC) is defined as:

$$\text{ACC} = \frac{\text{CorrectCount}}{|\mathcal{R}|} \tag{2}$$

$$\text{CorrectCount} = \sum_{\mathcal{X}_j \in \mathbb{X}} \sum_{x \in \mathcal{X}_j} \mathbb{I}(\exists y \in \mathcal{Y}'_j, y = x) \tag{3}$$

where $\mathbb{Y}' = \{\mathcal{Y}'_1, \ldots, \mathcal{Y}'_m\}$ is obtained by reordering $\mathbb{Y}$ based on their intersection sizes with the predicted clusters, and $\mathbb{I}(\cdot)$ is the indicator function that equals to 1 if the condition satisfies.

FP-measure evaluates clustering results from the perspective of homogeneity and stability. It is the harmonic mean of purity and inverse-purity.

$$purity\,(\mathbb{X},\mathbb{Y}) = \sum_{\mathcal{X}_i \in \mathbb{X}} \frac{|\mathcal{X}_i|}{|\mathcal{R}|} \max_{\mathcal{Y}_i \in \mathbb{Y}} Overlap(\mathcal{X}_i, \mathcal{Y}_i) \tag{4}$$

$$inverse\text{-}purity\,(\mathbb{X},\mathbb{Y}) = \sum_{\mathcal{Y}_i \in \mathbb{Y}} \frac{|\mathcal{Y}_i|}{|\mathcal{R}|} \max_{\mathcal{X}_i \in \mathbb{X}} Overlap(\mathcal{Y}_i, \mathcal{X}_i) \tag{5}$$

$$Overlap(\mathcal{X}_i, \mathcal{Y}_i) := \frac{|\mathcal{X}_i \cap \mathcal{Y}_i|}{|\mathcal{X}_i|} \tag{6}$$

[2]https://www.gabormelli.com/RKB/CORA_Citation_Benchmark_Task.
[3]https://pages.cs.wisc.edu/~anhai/data.

Table 2: Comparison with pairwise matching-based ER

Metrics	*Cora*		*Alaska*		*AS*	
	$S_s = 2$ (*pairwise*)	$S_s = 9$ (*clustering*)	$S_s = 2$ (*pairwise*)	$S_s = 9$ (*clustering*)	$S_s = 2$ (*pairwise*)	$S_s = 9$ (*clustering*)
ACC	0.88	**0.90**	0.81	**0.82**	**0.70**	**0.70**
FP-measure	0.67	**0.71**	0.78	**0.79**	0.60	**0.63**
Cost (USD)	0.67	**0.03**	0.43	**0.15**	0.08	**0.02**
Tokens (M)	3.45	**0.12**	2.29	**0.73**	0.35	**0.07**
Time (min)	297.27	**5.42**	241.31	**39.57**	77.2	**8.01**
# API Calls (K)	30.23	**0.28**	24.54	**2.04**	7.85	**0.41**

Table 3: Number of records set in each hierarchy level

Dataset	Level 0	Level 1	Level 2	Level 3	Level 4	Level 5
Cora	183	76	15	5	/	/
Alaska	1312	604	101	20	5	1
AS	251	107	48	6	1	/

Finally, we derive the FP-measure:

$$\textit{FP-measure}(\mathbb{X}, \mathbb{Y}) = \frac{2}{1/\textit{purity}(\mathbb{X}, \mathbb{Y}) + 1/\textit{inverse-purity}(\mathbb{X}, \mathbb{Y})} \quad (7)$$

6.2 Performance on End-to-End ER

We use `gpt-4o-mini` for end-to-end tests and set the temperature parameter to zero. For embedding generation in blocking, we adopt widely-used `all-MiniLM-L6-v2` model[4].

6.2.1 In-context clustering vs. pairwise questioning. We compare the performance of our in-context clustering-based ER ($S_s = 9$) with the pairwise matching method ($S_s = 2$) to demonstrate the effectiveness and efficiency of in-context clustering that can process multiple records simultaneously. For fair comparison, we also adopt our guardrail strategy to mitigate LLM-induced errors for pairwise matching as stated in § 5.2. Table 2 shows that when S_s is set to 9, the final ER results achieved by our algorithm are nearly identical or even better than those of the pairwise matching approach. On the other hand, our in-context clustering significantly reduces the number of LLM API calls by 12-108×, token consumption by 3-28×, monetary cost by 3-22×, and end-to-end ER time by 6-55×, compared to pairwise matching.

Table 3 illustrates the distribution of the number of record sets across hierarchy levels, providing insights into our hierarchical clustering approach. For all datasets, the number of record sets decreases significantly from Level 0 to higher levels, and are nearly completed before Level 5, demonstrating the effectiveness of the hierarchical structure in progressively merging similar entities. Larger datasets generally require more hierarchy levels for processing, as evidenced by *Alaska* (the largest dataset) extending to Level 5, while *Cora* (the smallest dataset) completes by Level 3.

6.2.2 Comparison with state-of-the-art. We compare our LLM-CER framework with several recent LLM-based ER methods to highlight its effectiveness and efficiency.

Booster [44]: Booster employs traditional blocking to generate multiple candidate partitions and iteratively queries the LLM using a *next-question selection*: identifying record pairs that are most informative in distinguishing between different partitioning. The final output is one of the partitionings with the highest probability. Unlike Booster, which is limited to selecting from predefined partitions, our method integrates blocking with hierarchical clustering and misclustering correction to produce refined clusters beyond the initial partitions—resulting in higher flexibility and accuracy.

BQ [27]: BQ reduces LLM usage by batching multiple pairwise match questions into a single prompt, allowing contextual information to improve prediction quality across questions. It also uses an enhanced demonstration selection strategy for better few-shot generalization. Each batch of k records forms $\frac{k}{2}$ pairwise comparisons. In contrast, our method with set size k performs clustering over the entire set, avoiding up to $\frac{k(k-1)}{2}$ comparisons, yielding significantly fewer API calls and lower monetary cost.

Table 4: End-to-end comparison of our in-context clustering-based ER with state-of-the-art LLM-based ER

Dataset	Metrics	LLM-CER	Booster	BQ	CrowdER +LLM
Alaska	ACC	**0.82**	0.71	0.33	0.68
	FP	**0.79**	0.55	0.49	0.62
	Cost ($)	0.15	**0.02**	1.55	0.42
	Tokens (M)	0.73	**0.19**	5.59	2.04
	Time (s)	**2374.2**	2450.1	8798.9	6547.2
	# API Calls	**2043**	2606	8035	5845
AS	ACC	**0.70**	0.62	0.54	0.52
	FP	**0.63**	0.62	0.51	0.50
	Cost ($)	0.02	**0.01**	0.29	0.11
	Tokens (M)	0.07	**0.03**	0.34	0.37
	Time (s)	**480.6**	622.9	925.5	2356.2
	# API Calls	**413**	723	842	2084
Song	ACC	**0.72**	0.52	0.59	0.52
	FP	**0.78**	0.68	0.67	0.64
	Cost ($)	0.06	**0.02**	0.77	0.12
	Tokens (M)	0.22	**0.11**	1.98	0.43
	Time (s)	**933.2**	903.3	2581.5	1856.3
	# API Calls	**668**	921	2338	1247
Music	ACC	**0.71**	0.59	0.60	0.62
	FP	**0.61**	0.60	0.54	0.55
	Cost ($)	0.19	**0.02**	2.18	0.39
	Tokens (M)	0.90	**0.15**	8.96	1.82
	Time (s)	**2388.4**	2585.1	17515.8	4562.3
	# API Calls	**3859**	3915	17365	7782
DG	ACC	**0.81**	0.56	0.62	0.72
	FP	**0.70**	0.68	0.63	0.65
	Cost ($)	0.07	**0.02**	1.12	0.34
	Tokens (M)	0.37	**0.18**	3.92	1.79
	Time (s)	**1552.4**	2552.2	6052.2	7456.3
	# API Calls	**1285**	3085	6456	6504
Cora	ACC	**0.90**	0.75	0.62	0.51
	FP	**0.71**	0.60	0.56	0.61
	Cost ($)	0.03	**0.01**	1.45	0.07
	Tokens (M)	0.12	**0.06**	4.23	0.29
	Time (s)	**325.5**	605.4	4085.3	598.5
	# API Calls	**279**	698	4882	483
Citeseer	ACC	**0.88**	0.72	0.64	0.60
	FP	**0.95**	0.78	0.79	0.69
	Cost ($)	0.03	**0.01**	0.63	0.08
	Tokens (M)	0.13	**0.05**	1.64	0.37
	Time (s)	**1360.8**	1585.2	6228.9	3895.6
	# API Calls	**1302**	2169	6420	3858
Amazon-Google	ACC	**0.71**	0.58	0.53	0.50
	FP	**0.64**	0.55	0.50	0.48
	Cost ($)	0.02	**0.01**	0.62	0.09
	Tokens (M)	0.07	**0.03**	0.86	0.42
	Time (s)	**465.6**	785.2	1658.2	1985.2
	# API Calls	**452**	998	1895	2025
Walmart-Amazon	ACC	**0.61**	0.50	0.42	0.51
	FP	**0.56**	0.48	0.41	0.50
	Cost ($)	0.02	**0.01**	0.59	0.08
	Tokens (M)	0.06	**0.03**	0.68	0.39
	Time (s)	**375.8**	475.2	1498.5	3895.6
	# API Calls	**398**	825	1585	1958

[4]https://www.sbert.net/

CrowdER+LLM [78]: CrowdER+LLM follows a clustering-based design that filters irrelevant pairs and generates record sets to resolve ambiguous cases, merging clusters via transitive closure. Our approach improves on this by incrementally constructing record sets and leveraging both transitivity and anti-transitivity to prune unnecessary sets. For a fair comparison, we adopt their record grouping strategy but replace crowdsourcing with LLM-based clustering, using the same blocking and record set size.

Implementation. For BQ, we follow [27] by including 8 demonstrations per prompt and applying the same blocking strategy as ours. Each prompt contains 5 pairwise questions (i.e., 10 records), approximately matching our 9-record clustering prompts for a fair workload comparison. As BQ requires labeled pairs for demonstrations, we include their annotation cost following the same accounting method. For Booster, we apply our blocking method with tuned parameters. We evaluate all methods using ACC and FP-measure.

End-to-end Performance. As shown in Table 4, the following observations can be made: (1) Our LLM-CER method consistently outperforms all baselines across datasets in both clustering quality (ACC and FP-measure) and operational efficiency (API calls and runtime), while maintaining competitive monetary cost. (2) Booster selects from multiple blocking-based partitionings using an LLM-driven scoring mechanism but does not refine these partitions. While it may choose a relatively strong candidate, its inability to correct or merge partitions limits its clustering quality: its ACC and FP scores remain suboptimal. In terms of monetary cost, however, Booster benefits from minimal token consumption, as it only selects the best partitioning without further modifying it. (3) BQ performs exhaustive pairwise matching within blocks using few-shot prompts, incurring substantial API and token costs—5–35× higher than ours—even after applying transitivity and anti-transitivity. Although demonstrations slightly improve pairwise accuracy, the lack of result verification often leads to incorrect merges, resulting in the lowest ACC and FP scores. Furthermore, BQ requires labeled examples for few-shot prompting, adding additional annotation cost, whereas our method operates entirely without supervision. (4) CrowdER+LLM uses clustering and achieves the second-best accuracy. However, its HIT design permits overlapping records and lacks alignment with LLM-optimized record set formats (§ 2), resulting in greater API calls and monetary cost. Moreover, like BQ, it lacks verification for LLM outputs, limiting its clustering quality compared to our LLM-CER approach.

Remark. Some of the competing methods are also *complementary to ours.* For example, BQ's batch processing can be used in our case to batch clustering of multiple record sets simultaneously. Similarly, Booster's best partitioning selection can serve as an effective pre-processing step, providing high-quality initial blocks that our algorithm can refine for enhanced accuracy. These complementarities underscore the potential for integrating strengths from different methods to advance ER performance.

6.3 Impact of Dataset Characteristics

In this section, we analyze how dataset characteristics—namely the number of attributes, attribute types, and overall dataset difficulty—affect the optimal configuration of key parameters and the performance of our end-to-end ER framework. To ensure fair evaluation, we retain critical attributes (e.g., title) across all settings.

Table 5: Optimal values vs. attribute count (A_n) and attribute types. The 'T' denotes 'Textual', 'N' represents 'Numeric', 'C' indicates 'Categorical'. Inc. denotes 'Included'

Dataset	A_n	S_s	S_d
Cora	4	9	3
	8	9	4
	12	9	4
Alaska	3	9	4
	6	9	4
	9	9	4

Dataset	Inc. type	S_s	S_d
WA	T, N, C	7	3
	N, C	12	4
	T, C	8	3
	T, N	8	4
Cite-seer	T, N, C	9	4
	N, C	8	4
	T, C	9	4
	T, N	9	4

Table 6: End-to-end ER performance vs. attribute count

Dataset	Metrics	$A_n = 4$	$A_n = 8$	$A_n = 12$
Cora	ACC	0.82	0.85	**0.90**
	FP	0.66	0.67	**0.71**
	Cost ($)	**0.02**	0.03	0.03
	Tokens (M)	**0.05**	0.09	0.12
	Time (min)	**5.04**	5.21	5.43
	API Calls	288	283	**279**
Alaska	ACC	0.74	0.77	**0.82**
	FP	0.74	0.75	**0.79**
	Cost ($)	**0.06**	0.11	0.15
	Tokens (M)	**0.26**	0.51	0.73
	Time (min)	**37.54**	38.24	39.57
	API Calls	2064	2055	**2043**

Impact of Attribute Count. To isolate the effect of increasing attribute count, we use the *Cora* and *Alaska* datasets, which contain many single-type (textual) attributes. As shown in Table 5, the optimal values for S_s and S_d remain largely stable. For *Cora*, S_s stays at 9 across all attribute counts, while S_d increases slightly from 3 to 4. For *Alaska*, both values remain fixed at 9 and 4, respectively. This suggests that, for single-type textual datasets, the parameter configuration is robust to changes in attribute count.

Building on these configurations, we conduct end-to-end ER experiments (Table 6). Results show that increasing attribute count consistently improves clustering performance: for *Cora*, ACC improves from 0.84 to 0.90 and FP-measure from 0.66 to 0.71 when moving from 4 to 12 attributes. Similar trends are observed for *Alaska*. These improvements persist with MDG, indicating that richer attribute sets enhance record distinguishability. While token usage increases moderately (0.05M to 0.12M for *Cora*), the accuracy gains justify retaining all attributes in single-type datasets.

Impact of Attribute Types and Dataset Difficulty. To assess the role of attribute types, we use *Walmart-Amazon* and *Citeseer*, both containing textual, numerical, and categorical fields. Controlled ablations omit one attribute type at a time to evaluate its contribution.

As shown in Table 5, the simpler *Citeseer* dataset exhibits stable parameter values, with only a minor drop when textual attributes are removed—likely due to increased reliance on less informative modalities. In contrast, the more complex *Walmart-Amazon* dataset shows notable deviations. When all attributes are retained, optimal values are $S_s = 7$ and $S_d = 3$, lower than the typical 9 and 4, indicating higher clustering difficulty. Removing textual attributes increases S_s and S_d to 12 and 4, suggesting that pruning noisy fields

Table 7: End-to-end ER performance vs. attribute types

Dataset	Metrics	Original	w/o Textual	w/o Numeric	w/o Categorical
Walmart-Amazon	ACC	0.61	**0.72**	0.66	0.60
	FP	0.56	**0.66**	0.58	0.54
	Cost ($)	0.02	**0.01**	0.02	0.02
	Tokens (M)	0.06	**0.04**	0.05	0.06
	Time (min)	6.25	**5.89**	6.02	6.36
	API Calls	398	**374**	393	409
Cite-seer	ACC	**0.88**	0.82	0.86	0.86
	FP	**0.95**	0.90	0.92	0.93
	Cost ($)	0.03	**0.02**	0.03	0.03
	Tokens (M)	0.13	**0.11**	0.12	0.12
	Time (min)	22.68	**20.98**	21.88	22.03
	API Calls	**1302**	1331	1312	1314

Table 8: Effect of MDG algorithm and record set regeneration on end-to-end performance

Metrics	*Cora*		*Alaska*		*AS*	
	w/o MDG	*w/ MDG*	*w/o MDG*	*w/ MDG*	*w/o MDG*	*w/ MDG*
ACC	0.60	**0.90**	0.35	**0.82**	0.52	**0.70**
FP-measure	0.58	**0.71**	0.47	**0.79**	0.52	**0.63**
Cost (USD)	**0.03**	**0.03**	**0.14**	0.15	**0.02**	**0.02**
Tokens (M)	**0.11**	0.12	**0.66**	0.73	**0.06**	0.07
Time (min)	**5.04**	5.42	**36.88**	39.57	**7.35**	8.01
# API Calls (K)	**0.26**	0.28	**1.85**	2.04	**0.37**	0.41

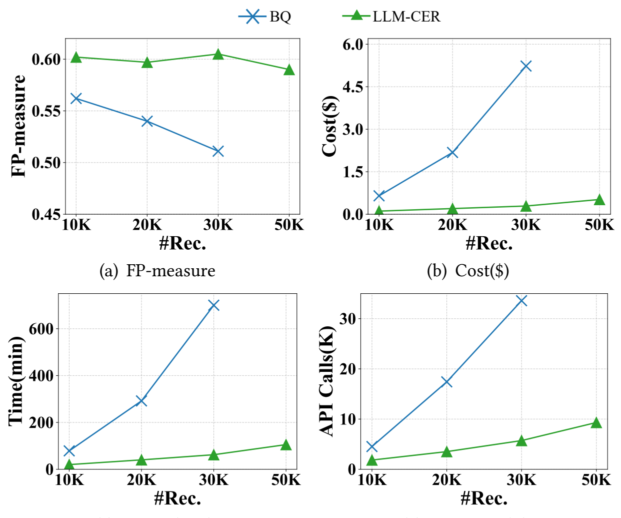
simplifies the task and enables larger input sets. Excluding numerical or categorical attributes also shifts S_s to 8, with corresponding adjustments in S_d. These results highlight that attribute type plays a critical role in complex domains, where noisy or redundant fields may obscure true entity boundaries.

End-to-end results (Table 6) further confirm this. For *Walmart-Amazon*, removing noisy textual attributes—while preserving key fields like title—boosts ACC from 0.61 to 0.72 and FP-measure from 0.56 to 0.66, with token usage dropping from 0.06M to 0.04M. This is due to common extraction errors (e.g., 'brand' values in 'name', or 'leather red' misclassified as 'color') that introduce significant noise. Eliminating these sources of noise allows the model to focus on cleaner signals, enhancing clustering quality. Similar observations are reported in [53]. Excluding numerical attributes yields a modest ACC gain (to 0.66), while removing categorical attributes slightly degrades performance, suggesting categorical fields offer informative structure. In contrast, *Citeseer* suffers across all ablations, reflecting the complementary value of all attribute types in well-structured datasets. These findings underscore the importance of selective attribute pruning for noisy, domain-specific datasets, and comprehensive attribute inclusion for simpler, structured ones.

6.4 Impact of LLM Guardrails

The results in Table 8 show that the incorporation of the Misclustering Detection Guardrail (MDG) algorithm leads to improvements in ACC and FP-measure. For instance, ACC increases by up to 50% and FP-measure by 22% on *Cora*. This supports the claim that MDG reduces the occurrence of misclassifications in the ER process.

Moreover, these performance gains are achieved with minimal additional computational cost. The total cost, token usage, processing time, and number of API calls remain almost unchanged or only slightly increase with the addition of MDG. This suggests that MDG enhances ER performance without incurring a significant resource overhead, making it an efficient approach for improving entity resolution in practical applications.



(a) FP-measure (b) Cost($)

(c) Times (min) (d) API Calls (K)

Figure 8: End-to-end performance vs. #Records

6.5 Scalability Test

To assess the scalability of our end-to-end ER algorithm, we conduct experiments on progressively larger subsets of the publicly available *Music* dataset [70], containing 10K, 20K, 30K, and 50K records. These subsets preserve consistent entity dispersion, enabling a controlled evaluation of how performance and resource usage evolve with scale. We also compare our method with BQ [27] (introduced in §6.2.2), a state-of-the-art baseline designed for efficient LLM-based ER. To ensure fairness, all experiments are subject to a maximum execution time limit of 24 hours.

Figure 8 shows that our LLM-CER consistently maintains high FP-measure as the dataset size increases, demonstrating strong robustness to scale. In contrast, BQ experiences a notable accuracy loss. Regarding efficiency, our method exhibits a predictable and modest increase in both cost and the number of API calls as data size grows, whereas BQ incurs substantially higher overheads. Execution time for our method scales linearly from approximately 20 minutes on *Music 10K* to just over 100 minutes on *Music 50K*. BQ becomes prohibitively slow on larger datasets, particularly failing to complete on the *Music 50K* dataset within the time limit.

7 CONCLUSION

In this work, we proposed an end-to-end in-context clustering-based entity resolution solution utilizing the Large Language Models (LLMs). We thoroughly explored the design space for in-context clustering, investigating how key factors – including set size, set diversity, set variation, and record order –influence clustering performance. Building on these insights, we introduced a robust end-to-end framework that incorporates an innovative record set creation strategy, a hierarchical clustering merge algorithm, and a tailored misclustering detection mechanism. Comprehensive evaluations on nine real-world datasets demonstrated that our approach, LLM-CER not only achieves substantial improvements in result quality but also significantly optimizes resource efficiency, including token usage, running time, and API call volume. In the future, ideas from other LLM-based ER methods could complement our approach, such as batch clustering of multiple record sets simultaneously and advanced prompt engineering for in-context clustering.

REFERENCES

[1] 2024. API Pricing of Open-AI. Retrieved December 16, 2024 from https://openai.com/api/pricing/

[2] Faraz Ahmed and Muhammad Abulaish. 2012. An MCL-based approach for spam profile detection in online social networks. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*. 602–608.

[3] Sihem Amer-Yahia, Angela Bonifati, Lei Chen, Guoliang Li, Kyuseok Shim, Jianliang Xu, and Xiaochun Yang. 2023. From large language models to databases and back: A discussion on research and education. *SIGMOD Rec.* 52, 3 (2023), 49–56.

[4] Anonymous Authors. 2025. Full version, source code and datasets. Retrieved January 14, 2025 from https://anonymous.4open.science/r/LLM4ER-CB0F/

[5] Omar Benjelloun, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, and Jennifer Widom. 2009. Swoosh: A generic approach to entity resolution. *VLDB Journal* 18, 1 (2009), 255–276.

[6] 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.

[7] Christian Böhm and Hans-Peter Kriegel. 2001. A cost model and index architecture for the similarity join. In *Proceedings 17th International Conference on Data Engineering*, Dimitrios Georgakopoulos and Alexander Buchmann (Eds.). 411–420.

[8] A. Broder. 1997. On the resemblance and containment of documents. In *Compression and Complexity of (SEQUENCES)*. 21–29.

[9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*. 1877–1901.

[10] 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.

[11] Zhaoqi Chen, Dmitri V Kalashnikov, and Sharad Mehrotra. 2005. Exploiting relationships for object consolidation. In *Proceedings of the 2nd international workshop on Information quality in information systems*. 47–58.

[12] Zhaoqi Chen, Dmitri V Kalashnikov, and Sharad Mehrotra. 2007. Adaptive graphical approach to entity resolution. In *ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL)*. 204–213.

[13] Zhikai Chen, Haitao Mao, Hongzhi Wen, Haoyu Han, Wei Jin, Haiyang Zhang, Hui Liu, and Jiliang Tang. [n.d.]. Label-free node classification on graphs with large language models (llms). In *The Twelfth International Conference on Learning Representations year=2024*.

[14] Peter Christen. 2012. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering* 24, 9 (2012), 1537–1555.

[15] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. 2020. An overview of end-to-end entity resolution for big data. *ACM Comput. Surv.* 53, 6 (2020), 1–42.

[16] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. 2021. An overview of end-to-end entity resolution for big data. *ACM Comput. Surv.* 53, 6 (2021), 127:1–127:42.

[17] Pierluigi Crescenzi, Viggo Kann, Magnus Halldorsson, Marek Karpinski, and Gerhard Woeginger. 2000. Maximum 3-dimensional matching. *A Compendium of NP Optimization Problems* (2000).

[18] Valter Crescenzi, Andrea De Angelis, Donatella Firmani, Maurizio Mazzei, Paolo Merialdo, Federico Piai, and Divesh Srivastava. 2021. Alaska: A flexible benchmark for data integration tasks. *arXiv Preprint arXiv:2101.11259* (2021).

[19] Mengyao Cui. 2020. Introduction to the k-means clustering algorithm based on the elbow method. *Accounting, Auditing and Finance* 1, 1 (2020), 5–8.

[20] Sanjib Das, Paul Suganthan G. C., AnHai Doan, Jeffrey F. Naughton, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, Vijay Raghavendra, and Youngchoon Park. 2017. Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1431–1446.

[21] Database Research Group of the Roma Tre University. 2020. SIGMOD 2020 programming contest official website. Retrieved September 22, 2024 from http://www.inf.uniroma3.it/db/sigmod2020contest

[22] Gianluca Demartini, Djellel Eddine Difallah, and Philippe Cudré-Mauroux. 2012. ZenCrowd: Leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *Proceedings of the 21st International Conference on World Wide Web*. 469–478.

[23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, (NAACL-HLT)*, 4171–4186.

[24] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed representations of tuples for entity resolution. In *Proc. VLDB Endow.*, Vol. 11. 1454—-1467.

[25] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. 2007. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering* 19, 1 (2007), 1–16.

[26] Maxim Enis and Mark Hopkins. 2024. From LLM to NMT: Advancing Low-Resource Machine Translation with Claude. In *Proceedings of the 41st International Conference on Machine Learning*. 55204–55224.

[27] Meihao Fan, Xiaoyue Han, Ju Fan, Chengliang Chai, Nan Tang, Guoliang Li, and Xiaoyong Du. 2024. Cost-effective in-context learning for entity resolution: A design space exploration. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 3696–3709.

[28] Sainyam Galhotra, Donatella Firmani, Barna Saha, and Divesh Srivastava. 2021. Efficient and effective ER with progressive blocking. *The VLDB Journal* 30, 4 (2021), 537–557.

[29] Papadakis George, Mandilaras George, Gagliardelli Luca, Simonini Giovanni, Thanos Emmanouil, Giannakopoulos George, Bergamaschi Sonia, Palpanas Themis, and Koubarakis Manolis. 2020. Three-dimensional entity resolution with JedAI. *Information Systems* 93 (2020), 101565.

[30] Lise Getoor and Ashwin Machanavajjhala. 2012. Entity resolution: theory, practice & open challenges. *Proc. VLDB Endow.* 5, 12 (2012), 2018—-2019.

[31] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, Vol. 99. 518–529.

[32] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F. Naughton, Narasimhan Rampalli, Jude Shavlik, and Xiaojin Zhu. 2014. Corleone: Hands-off crowdsourcing for entity matching. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. 601–612.

[33] Anja Gruenheid, Donald Kossmann, and Besmira Nushi. 2013. When is A=B? *Bull. EATCS* 111 (2013).

[34] Nuno M. Guerreiro, Duarte M. Alves, Jonas Waldendorf, Barry Haddow, Alexandra Birch, Pierre Colombo, and André F. T. Martins. 2023. Hallucinations in large multilingual translation models. *Transactions of the Association for Computational Linguistics* 11 (2023), 1500–1517.

[35] Oktie Hassanzadeh, Fei Chiang, Hyun Chul Lee, and Renée J Miller. 2009. Framework for evaluating clustering algorithms in duplicate detection. In *Proc. VLDB Endow.*, Vol. 2. 1282–1293.

[36] Boyi Hou, Qun Chen, Jiquan Shen, Xin Liu, Ping Zhong, Yanyan Wang, Zhaoqiang Chen, and Zhanhuai Li. 2019. Gradual machine learning for entity resolution. In *The World Wide Web Conference*. 3526–3530.

[37] Arvanitis Kasinikos Ioannis. 2024. Entity resolution with small-scale LLMs: A study on prompting strategies and hardware limitations.

[38] Shawn R. Jeffery, Michael J. Franklin, and Alon Y. Halevy. 2008. Pay-as-you-go user feedback for dataspace systems. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. 847–860.

[39] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Proceedings of the 36th International Conference on Neural Information Processing Systems* 35 (2022), 22199–22213.

[40] Pradap Konda, Sanjib Das, Paul Suganthan G. C., AnHai Doan, Adel Ardalan, Jeffrey R. Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeffrey F. Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Raghavendra. 2016. Magellan: Toward Building Entity Matching Management Systems. *Proc. VLDB Endow.* 9, 12 (2016), 1197–1208.

[41] Michael Levandowsky and David Winter. 1971. Distance between sets. *Nature* 234, 5323 (1971), 34–35.

[42] Bing Li, Yukai Miao, Yaoshu Wang, Yifang Sun, and Wei Wang. 2021. Improving the efficiency and effectiveness for BERT-based entity resolution. In *AAAI Conference on Artificial Intelligence*. 13226–13233.

[43] Bing Li, Yukai Miao, Yaoshu Wang, Yifang Sun, and Wei Wang. 2021. Improving the efficiency and effectiveness for bert-based entity resolution. In *AAAI Conference on Artificial Intelligence*, Vol. 35. 13226–13233.

[44] Huahang Li, Shuangyin Li, Fei Hao, Chen Jason Zhang, Yuanfeng Song, and Lei Chen. 2024. BoostER: Leveraging large language models for enhancing entity resolution. In *Companion Proceedings of the ACM Web Conference 2024*. 1043–1046.

[45] Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. 2024. Long-context llms struggle with long in-context learning. *CoRR* (2024).

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

[47] Yuliang Li, Jinfeng Li, Yoshi Suhara, AnHai Doan, and Wang-Chiew Tan. 2023. Effective entity matching with transformers. *The VLDB Journal* 32, 6 (2023),

1215–1235.

[48] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. 2021. Self-supervised learning: Generative or contrastive. *IEEE Transactions on Knowledge and Data Engineering* 35, 1 (2021), 857–876.

[49] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *CoRR* abs/1907.11692 (2019).

[50] Alessio M., Faggioli G., Ferro N., Nardini F. M., and Perego R. 2024. Improving RAG systems via sentence clustering and reordering. In *CEUR WORKSHOP PROCEEDINGS, vol. 3784, pp. 34-43. Washington DC, USA, 07/07/2024*. CEUR-WS, 34–43.

[51] Andrew McCallum, Kamal Nigam, and Lyle H Ungar. 2000. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 169–178.

[52] Venkata Vamsikrishna Meduri, Lucian Popa, Prithviraj Sen, and Mohamed Sarwat. 2020. A comprehensive benchmark framework for active learning methods in entity matching. In *ACM SIGMOD International Conference on Management of Data*. 1133–1147.

[53] 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.

[54] Navapat Nananukul, Khanin Sisaengsuwanchai, and Mayank Kejriwal. 2024. Cost-efficient prompt engineering for unsupervised entity resolution in the product matching domain. *Discover Artificial Intelligence* 4, 1 (2024), 56.

[55] Avanika Narayan, Ines Chami, Laurel Orr, Simran Arora, and Christopher Ré. 2022. Can foundation models wrangle your data?. In *Proc. VLDB Endow.*, Vol. 16. 738—-746.

[56] Felix Naumann and Melanie Herschel. 2010. *An introduction to duplicate detection*. Morgan & Claypool Publishers.

[57] Hao Nie, Xianpei Han, Ben He, Le Sun, Bo Chen, Wei Zhang, Suhui Wu, and Hao Kong. 2019. Deep sequence-to-sequence entity matching for heterogeneous entity resolution. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 629–638.

[58] Konstantinos Nikoletos, George Papadakis, and Manolis Koubarakis. 2022. pyJedAI: a Lightsaber for Link Discovery. In *ISWC (Posters/Demos/Industry)*.

[59] George Papadakis, Ekaterini Ioannou, Emanouil Thanos, and Themis Palpanas. 2021. *The four generations of entity resolution*. Morgan & Claypool Publishers.

[60] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. 2020. Blocking and filtering techniques for entity resolution: A survey. *ACM Comput. Surv.* 53, 2 (2020), 1–42.

[61] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. 2020. Blocking and filtering techniques for entity resolution: A survey. *ACM Comput. Surv.* 53, 2, Article 31 (2020), 42 pages.

[62] George Papadakis, Jonathan Svirsky, Avigdor Gal, and Themis Palpanas. 2016. Comparative analysis of approximate blocking techniques for entity resolution. In *Proc. VLDB Endow.*, Vol. 9. 684–695.

[63] Ralph Peeters and Christian Bizer. 2021. Dual-objective fine-tuning of BERT for entity matching. *Proc. VLDB Endow.* 14, 10 (2021), 1913–1921.

[64] Ralph Peeters and Christian Bizer. 2023. Using ChatGPT for entity matching. In *New Trends in Database and Information Systems*. 221–230.

[65] Ralph Peeters, Aaron Steiner, and Christian Bizer. 2025. Entity matching using large language models. In *International Conference on Extending Database Technology (EDBT)*. 529–541.

[66] Kun Qian, Lucian Popa, and Prithviraj Sen. 2017. Active learning for large-scale entity resolution. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 1379–1388.

[67] Leigang Qu, Shengqiong Wu, Hao Fei, Liqiang Nie, and Tat-Seng Chua. 2023. Layoutllm-t2i: Eliciting layout guidance from llm for text-to-image generation. In *Proceedings of the 31st ACM International Conference on Multimedia*. 643–654.

[68] Mehdi Akbarian Rastaghi, Ehsan Kamalloo, and Davood Rafiei. 2022. Probing the robustness of pre-trained language models for entity matching. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 3786–3790.

[69] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 3982–3992.

[70] Alieh Saeedi, Eric Peukert, and Erhard Rahm. 2017. Comparative evaluation of distributed clustering schemes for multi-source entity resolution. In *Advances in Databases and Information Systems (ADBIS)*. 278–293.

[71] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In *Workshop on Energy Efficient Machine Learning and Cognitive Computing@NeurIPS*.

[72] Xiaofei Sun, Xiaoya Li, Jiwei Li, Fei Wu, Shangwei Guo, Tianwei Zhang, and Guoyin Wang. 2023. Text classification via large language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

[73] 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. *Proc. VLDB Endow.* 14, 11 (2021), 2459–2472.

[74] Sindhu Tipirneni, Ravinarayana Adkathimar, Nurendra Choudhary, Gaurush Hiranandani, Rana Ali Amjad, Vassilis N. Ioannidis, Changhe Yuan, and Chandan K. Reddy. 2024. Context-Aware Clustering using Large Language Models. arXiv:2405.00988 [cs.CL] https://arxiv.org/abs/2405.00988

[75] Vasilis Verroios and Hector Garcia-Molina. 2015. Entity resolution with crowd errors. In *2015 IEEE 31st International Conference on Data Engineering*. 219–230.

[76] Norases Vesdapunt, Kedar Bellare, and Nilesh N. Dalvi. 2014. Crowdsourcing algorithms for entity resolution. *Proc. VLDB Endow.* 7, 12 (2014), 1071–1082.

[77] Vijay Viswanathan, Kiril Gashteovski, Carolin Lawrence, Tongshuang Wu, and Graham Neubig. 2024. Large language models enable few-shot clustering. *Transactions of the Association for Computational Linguisticss* 12 (2024), 321–333.

[78] Jiannan Wang, Tim Kraska, Michael J Franklin, and Jianhua Feng. 2012. Crowder: Crowdsourcing entity resolution. In *Proc. VLDB Endow.*, Vol. 5. 1483—-1494.

[79] Jiannan Wang, Guoliang Li, Tim Kraska, Michael J. Franklin, and Jianhua Feng. 2013. Leveraging transitive relations for crowdsourced joins. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 229–240.

[80] Sibo Wang, Xiaokui Xiao, and Chun-Hee Lee. 2015. Crowd-based deduplication: An adaptive approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 1263–1277.

[81] Tianshu Wang, Hongyu Lin, Xiaoyang Chen, Xianpei Han, Hao Wang, Zhenyu Zeng, and Le Sun. 2025. Match, compare, or select? An investigation of large language models for entity matching. In *Proceedings of the 31st International Conference on Computational Linguistics*. 96–109.

[82] Yu Wang, Xinjie Yao, Pengfei Zhu, Weihao Li, Meng Cao, and Qinghua Hu. 2024. Integrated heterogeneous graph attention network for incomplete multi-modal clustering. *International Journal of Computer Vision* 132, 9 (2024), 3847–3866.

[83] Steven Euijong Whang, Peter Lofgren, and Hector Garcia-Molina. 2013. Question selection for crowd entity resolution. *Proc. VLDB Endow.* 6, 6 (2013), 349–360.

[84] Jianlong Wu, Keyu Long, Fei Wang, Chen Qian, Cheng Li, Zhouchen Lin, and Hongbin Zha. 2019. Deep comprehensive correlation mining for image clustering. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 8150–8159.

[85] Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu, and Guoren Wang. 2011. Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst.* 36, 3 (2011), 1–41.

[86] Vijaya Krishna Yalavarthi, Xiangyu Ke, and Arijit Khan. 2017. Select your questions wisely: For entity resolution with crowd errors. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 317–326.

[87] Ka Yee Yeung and Walter L Ruzzo. 2001. Details of the adjusted rand index and clustering algorithms, supplement to the paper an empirical study on principal component analysis for clustering gene expression data. *Bioinformatics* 17, 9 (2001), 763–774.

[88] Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. 2023. Large language models as data preprocessors. *Workshops at the International Conference on Very Large Data Bases*.

[89] Yuwei Zhang, Zihan Wang, and Jingbo Shang. 2023. ClusterLLM: Large Language Models as a Guide for Text Clustering. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 13903–13920.

[90] Zeyu Zhang, Paul Groth, Iacer Calixto, and Sebastian Schelter. 2024. Directions towards efficient and automated data wrangling with large language models. In *2024 IEEE 40th International Conference on Data Engineering Workshops (ICDEW)*. 301–304.

8 APPENDIX

This section presents the additional experimental results omitted from § 6 due to space constraints.

More Evaluation Metrics. We select several widely-used metrics to evaluate the in-context clustering results, as well as the final ER clustering results, including FP-measure (a variant of the F-measure), Accuracy (ACC), Normalized Mutual Information (NMI), and Adjusted Rand Score (ARI) [2, 12, 82, 87]. All experiments are reported as the average of 3 repeats. Due to space limitations, we present results only for ACC and FP-measure in the main paper. We define the other two measures below.

Let $\mathbb{Y} = \{\mathcal{Y}_1, \mathcal{Y}_2, \ldots, \mathcal{Y}_m\}$ represents the ground truth clusters and $\mathbb{X} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n\}$ denotes the predicted clusters. The total number of records in both sets is $|\mathcal{R}| = \bigcup_i |\mathcal{X}_i| = \bigcup_j |\mathcal{Y}_j|$.

NMI measures the similarity between clustering results and ground truth, from the perspective of clustering entropy.

$$NMI(\mathbb{X}, \mathbb{Y}) = \frac{2I(\mathbb{X}, \mathbb{Y})}{H(\mathbb{X}) + H(\mathbb{Y})} \tag{8}$$

where the function $I(\cdot)$ measures the degree of information sharing between the two clusters and the function $H(\cdot)$ is used to measure the uncertainty of elements in clusters.

$$I(\mathbb{X}, \mathbb{Y}) = \sum_{i=1}^{|\mathbb{X}|} \sum_{j=1}^{|\mathbb{Y}|} p(i,j) \log \left(\frac{p(i,j)}{p(i)p(j)} \right) \tag{9}$$

$$H(\mathbb{X}) = -\sum_{i=1}^{|\mathbb{X}|} p(i) \log p(i) \tag{10}$$

where $p(i)$ is the probability of element i being assigned to cluster $\mathcal{X}_i$, i.e., $p(i) = \frac{|\mathcal{X}_i|}{|\mathcal{R}|}$, and $p(i,j)$ is the joint probability of element i correctly being assigned to both the predicted cluster $\mathcal{X}_i$ and the ground truth cluster $\mathcal{Y}_j$, i.e., $p(i,j) = \frac{|\mathcal{X}_i \cap \mathcal{Y}_j|}{|\mathcal{R}|}$.

Different from the above, ARI considers both the similarity and dissimilarity of all pairs of data records. The definition of ARI in our work is the same as [84]. Consider a contingency table $[t_{ij}]$ of overlaps between $\mathcal{X}_i$ and $\mathcal{Y}_j$. Each element in $[t_{ij}]$ shows the number of overlapping objects between $\mathcal{X}_i$ and $\mathcal{Y}_j$. We denote by a_i the number of elements in $\mathcal{X}_i$, i.e., $a_i = |\mathcal{X}_i|$, where b_j also represents the cardinality of $\mathcal{Y}_j$, i.e., $b_j = |\mathcal{Y}_j|$.

$$ARI = \frac{\sum_{i,j} \binom{t_{i,j}}{2} - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2} \right] / \binom{n}{2}}{\frac{1}{2} \left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2} \right] - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2} \right] / \binom{n}{2}} \tag{11}$$

Table 9: Optimal key factor values with different LLMs

Factors	GPT 4o-mini-8B	Llama 3.2-1B
S_s	9	6
S_d	4	3
S_v	~ 0	~ 0
Orders	Sequential	Sequential

Table 10: Comparisons of LLM-CER using GPT and Llama

Dataset	Metrics	GPT	Llama
Alaska	ACC	**0.82**	0.64
	FP	**0.79**	0.46
	NMI	**0.79**	0.48
	ARI	**0.65**	0.41
	# API Calls	**2043**	3215
AS	ACC	**0.70**	0.42
	FP	**0.63**	0.52
	NMI	**0.73**	0.46
	ARI	**0.62**	0.49
	# API Calls	**413**	685
Song	ACC	**0.72**	0.45
	FP	**0.78**	0.52
	NMI	**0.74**	0.53
	ARI	**0.66**	0.48
	# API Calls	**668**	1025
Music	ACC	**0.71**	0.52
	FP	**0.61**	0.57
	NMI	**0.74**	0.53
	ARI	**0.62**	0.45
	# API Calls	**3859**	5745
DG	ACC	**0.81**	0.49
	FP	**0.70**	0.57
	NMI	**0.84**	0.51
	ARI	**0.68**	0.49
	# API Calls	**1285**	1865
Cora	ACC	**0.90**	0.63
	FP	**0.71**	0.48
	NMI	**0.82**	0.52
	ARI	**0.69**	0.43
	# API Calls	**279**	412
Cite-seer	ACC	**0.88**	0.61
	FP	**0.95**	0.58
	NMI	**0.85**	0.59
	ARI	**0.74**	0.61
	# API Calls	**1302**	2005

A.1 Impact of Different LLMs on Optimal Key Factor Values and End-to-end ER Performance

Optimal Key Factor Values with Different LLMs. Table 9 shows optimal key factor values for GPT-4o mini-8B and Llama 3.2-1B, reflecting their distinct characteristics. For S_s (record set size), GPT-4o mini's optimal value is 9, indicating its ability to handle larger contexts due to its 8B parameters and better long-range dependency processing. Llama 3.2-1B, with 1B parameters, has an optimal S_s of 6, performing better with smaller record sets due to its limited capacity. For S_d (entities per record set), GPT-4o mini's optimal value is 4, managing more entities effectively, while Llama's is 3, reflecting its reduced ability to handle complex prompts. Both models prefer smaller S_v ($\sim$0) and sequential record ordering, benefiting from coherence in ER tasks regardless of model size.

End-to-end ER Performance Comparison. Table 10 compares in-context clustering-based ER performance of GPT and Llama across seven datasets, focusing on ACC, FP-measure, NMI, ARI, and API calls. Token usage, API costs, and time-based metrics are excluded, as Llama is open-source (no costs) and lacks network latency. GPT consistently outperforms Llama in all quality metrics across all datasets, likely due to its larger model size and stronger

Table 11: Optimal key factor values with entity dispersion

Factors	$E_d = 4$	$E_d = 8$	$E_d = 12$
S_s	8	9	9
S_d	4	4	4
S_v	~ 0	~ 0	~ 0
Orders	Sequential	Sequential	Sequential

Table 12: End-to-end ER performance with varying E_d (with #Entity fixed to 100)

Dataset	Metrics	$E_d = 4$	$E_d = 8$	$E_d = 12$
Cora	ACC	0.87	0.84	0.89
	FP	0.71	0.68	0.72
	Cost ($)	0.02	0.03	0.03
	Tokens (M)	0.06	0.10	0.12
	Time (s)	209.4	285.3	321.5
	API Calls	192	251	277

Table 13: End-to-end ER performance with varying E_d (with #Record fixed to 600)

Dataset	Metrics	$E_d = 12$	$E_d = 16$	$E_d = 20$
Cora	ACC	0.87	0.85	0.88
	FP	0.73	0.73	0.72
	Cost ($)	0.02	0.02	0.01
	Tokens (M)	0.06	0.05	0.04
	Time (s)	163.2	134.5	114.6
	API Calls	139	112	98

context understanding. Llama requires more API calls, indicating greater computational resource use despite its cost-free nature. Larger models like GPT enhance clustering performance, but Llama remains a viable, cost-effective option when accuracy is less critical than cost.

A.2 Impact of Entity Dispersion on Optimal Key Factor Values and End-to-end ER Performance

In this section, we investigate the impact of entity dispersion (E_d), defined as the average number of records per entity, on optimal in-context clustering parameters (S_s, S_d, S_v, and sequential/random ordering) and the performance of our end-to-end ER approach using the semantically diverse *Cora* dataset. In the first experiment, we fix 100 entities and vary duplicate records per entity to 4, 8, or 12, creating datasets with $E_d = 4, 8, 12$ (sizes 400, 800, 1200 records, respectively). This tests the effect of increasing record density per entity with constant entity count. In the second experiment, we fix total records at ≈600 and adjust entity counts to achieve $E_d = 12$ (50 entities, 12 records each), $E_d = 16$ (38 entities, 16 records each), and $E_d = 20$ (30 entities, 20 records each), examining how redistributing records across varying entity counts affects clustering.

Impact on Optimal Values of Key Factors. Table 11 shows that entity dispersion (E_d) has little effect on optimal record set parameters across *Cora* dataset with $E_d = 4, 8, 12$. Optimal S_d, S_v, and "Sequential" ordering remain consistent, with S_s slightly dropping from 9 to 8 at $E_d = 4$. This stability reflects the LLM's strong semantic understanding, effectively distinguishing entities despite varying record counts. The minor S_s adjustment at lower E_d suggests that fewer records suffice for entity differentiation. These results underscore the method's adaptability to diverse datasets without extensive parameter tuning for LLM like GPT-4o-mini.

Impact on End-to-end ER Performance. Table 12 shows that higher entity dispersion (E_d) improves clustering performance with modestly increased resource usage. ACC slightly decreases from 0.87 at $E_d = 4$ to 0.84 at $E_d = 8$ but rises to 0.89 at $E_d = 12$, with FP-measure showing a similar trend. Lower E_d may lack sufficient duplicates for robust clustering, while higher E_d leverages redundancy to resolve ambiguities and enhance stability. Despite larger datasets at higher E_d (e.g., $E_d = 12$ has 3× records of $E_d = 4$), resource use (costs, tokens, processing times, API calls) grows less than threefold, indicating efficiency from consolidating duplicates, as analyzed in §5.4.

In the second scenario (Table 13), increasing E_d from 12 to 20 minimally affects ACC and FP but significantly reduces resource consumption. API calls drop from 139 to 98, token usage from 0.06M to 0.04M, and costs from \$0.02 to \$0.01. Higher E_d maintains clustering quality with lower computational overhead, improving efficiency, consistent with §5.4.

Table 14: Impact of blocking and filtering

Dataset	Metrics	w/o blocking	Filter	Canopy	LSH
Cora	ACC	0.62	0.81	0.67	**0.90**
	FP	0.58	0.78	0.60	**0.71**
	Cost ($)	0.33	0.03	0.06	**0.03**
	Tokens (M)	1.33	0.13	0.22	**0.12**
	Time (s)	3708.8	326.1	529.6	**325.5**
	API Calls	1996	301	440	**279**
AS	ACC	0.61	0.68	0.66	**0.70**
	FP	0.58	**0.64**	0.60	0.63
	Cost ($)	0.11	**0.02**	0.03	**0.02**
	Tokens (M)	0.34	**0.07**	0.09	**0.07**
	Time (s)	1542.3	**305.2**	406.5	325.5
	API Calls	2156	**402**	526	413
Alaska	ACC	0.70	0.77	0.74	**0.82**
	FP	0.69	0.74	0.72	**0.79**
	Cost ($)	0.82	0.17	0.18	**0.15**
	Tokens (M)	3.85	0.80	0.84	**0.73**
	Time (s)	12452.5	2612.5	2745.2	**2374.2**
	API Calls	11542	2252	2354	**2043**

A.3 Impact of Blocking and Filtering Techniques

In this section, we evaluate the impact of different blocking methods on both the performance (e.g., ACC, FP-measure) and resource consumption (e.g., tokens, cost, time, and number of API calls) of our algorithm on three representative datasets: *Cora*, *AS*, and *Alaska*. All three blocking methods are unsupervised and are applied in an alternating manner throughout our experiments.

Table 14 shows that blocking significantly reduces resource consumption compared to the no-blocking baseline, with varying performance across methods. For *Cora*, no blocking yields low ACC and FP-measure (≈ 0.6) with high resource use (>1M tokens, long runtime). LSH performs best, achieving ACC near 0.9 and FP-measure around 0.7 with minimal resources (<0.2M tokens, hundreds of seconds). Filtering follows with ACC near 0.8 and FP-measure close to 0.8, also resource-efficient. Canopy slightly improves over no blocking but uses more resources than LSH and Filtering. These trends hold for *AS* and *Alaska*.

Table 15: Effect of MDG algorithm and record set regeneration on end-to-end performance: NMI and ARI

Metrics	*Cora*		*Alaska*		*AS*	
	w/o MDG	*w/ MDG*	*w/o MDG*	*w/ MDG*	*w/o MDG*	*w/ MDG*
NMI	0.61	**0.82**	0.52	**0.79**	0.54	**0.73**
ARI	0.48	**0.69**	0.33	**0.65**	0.48	**0.62**

The superior performance of LSH stems from forming high-quality clusters by capturing semantic similarities, while Canopy's coarser clusters reduce accuracy. No blocking is costly and less effective due to noise and unstructured input. These findings highlight blocking's role in enhancing efficiency and the need to choose suitable methods to optimize both performance and resource use in LLM-based ER pipelines.

A.4 LLM Guardrails on Misclustering Detection and Record Set Regeneration: MRI and ARI

The results in Table 15 show that the incorporation of the MDG algorithm also leads to improvements in NMI and ARI. In addition to ACC and FP-measure, NMI and ARI also show substantial improvements, indicating that MDG not only enhances classification accuracy but also produces more consistent and accurate clustering results. These improvements can be attributed to the ability of MDG to more effectively prevent misclustering, ensuring that entities which should remain separate are not incorrectly merged. This is reflected in higher values of the clustering metrics (NMI and ARI), indicating better alignment between predictions and truths.

A.5 Comparisons with PLM-based ER Methods

Baselines. We compare our algorithm with two widely-used PLM (pre-trained language model)-based ER methods, Ditto [46] and DeepMatcher [53], both of which follow a pairwise matching paradigm. We use transitivity and anti-transitivity to obtain their final clustering results for comparison.

- **Ditto [46]**: Ditto [46] is a PLM-based entity resolution method that fine-tunes a pre-trained language model using labeled entity pairs. By leveraging task-specific supervision, it enhances entity matching performance. To ensure a fair comparison, we use the original implementation and default hyperparameters from the paper.
- **DeepMatcher [53]**: DeepMatcher [53] is a deep learning framework for entity and text matching, offering pre-built neural networks and utilities for training advanced models. It excels in resolving complex entity alignment and text correspondence tasks with high precision. Given that the authors have released it as a Python library, we adopt it directly for evaluation.

Implementation Details and Cost Evaluation. Ditto [46] and DeepMatcher [53] are PLM-based ER methods designed for general applicability across diverse datasets. To systematically evaluate their performance, we consider two alternative modes: (1) *Without fine-tuning*: The pre-trained models are used directly for ER without additional training, we report the ACC and FP-measure obtained from applying these models to the entire dataset (100% treated as test data). Additionally, we report the monetary cost (discussed below) required for inference over the whole dataset; and (2) *With fine-tuning*: The models are fine-tuned using 20%/80% of the ground truth pairs for each dataset to adapt to domain-specific variations, the remaining 80%/20% of the data is used as test data for obtain ACC and FP-measure. We choose to use 80% of the data for training,

Table 16: End-to-end comparison of our in-context clustering-based ER with PLM-based ER, "FT" denotes fine-tuning

Dataset	Metrics	Ours	**Ditto** [46]			**DeepMatcher** [53]		
			w/ FT (20%)	*w/ FT (80%)*	*w/o FT*	*w/ FT (20%)*	*w/ FT (80%)*	*w/o FT*
Alaska	ACC	**0.82**	0.70	0.81	0.64	0.65	0.74	0.58
	FP	**0.79**	0.62	0.77	0.55	0.51	0.70	0.43
	Cost ($)	0.15	65.67	260.21	**0.12**	65.81	260.93	0.14
AS	ACC	0.70	0.60	**0.72**	0.58	0.57	0.65	0.54
	FP	0.63	0.53	**0.66**	0.51	0.49	0.61	0.47
	Cost ($)	**0.02**	7.06	28.85	0.08	7.18	29.29	0.07
Song	ACC	**0.72**	0.61	0.70	0.54	0.59	0.66	0.53
	FP	**0.78**	0.63	0.76	0.58	0.52	0.72	0.51
	Cost ($)	**0.06**	13.74	41.34	0.09	12.82	38.65	0.15
Music	ACC	0.71	0.63	**0.74**	0.57	0.59	0.66	0.55
	FP	0.61	0.53	**0.65**	0.44	0.52	0.56	0.47
	Cost ($)	0.19	79.30	239.22	0.22	79.08	238.54	**0.14**
DG	ACC	**0.81**	0.68	0.80	0.60	0.67	0.77	0.60
	FP	**0.70**	0.60	0.69	0.52	0.53	0.66	0.50
	Cost ($)	**0.07**	12.93	42.04	0.13	13.01	42.22	0.16
Cora	ACC	**0.90**	0.76	**0.90**	0.67	0.71	0.88	0.66
	FP	0.71	0.56	**0.72**	0.48	0.54	0.70	0.50
	Cost ($)	**0.03**	10.71	42.92	0.07	10.74	43.08	0.06
Cite-seer	ACC	0.88	0.76	**0.89**	0.62	0.60	0.73	0.59
	FP	**0.95**	0.70	0.93	0.65	0.67	0.91	0.59
	Cost ($)	**0.03**	9.62	38.32	0.08	9.73	39.04	0.09
AG	ACC	0.71	0.61	**0.73**	0.56	0.58	0.69	0.57
	FP	0.64	0.57	**0.68**	0.50	0.54	0.64	0.49
	Cost ($)	**0.02**	7.13	28.84	0.07	7.16	28.04	0.05
WA	ACC	0.61	0.56	**0.65**	0.48	0.50	0.60	0.44
	FP	0.56	0.45	**0.60**	0.40	0.44	0.57	0.39
	Cost ($)	**0.02**	6.98	28.03	0.06	6.89	27.96	0.06

as performance improvements for PLM-based methods plateaued at this proportion across nearly all datasets. This ensures a consistent evaluation framework across all approaches. Following [81], we estimate the monetary cost of PLM methods by incorporating both inference and additional fine-tuning costs, based on the required training time and the hourly price of cloud NVIDIA A40 GPUs[5].

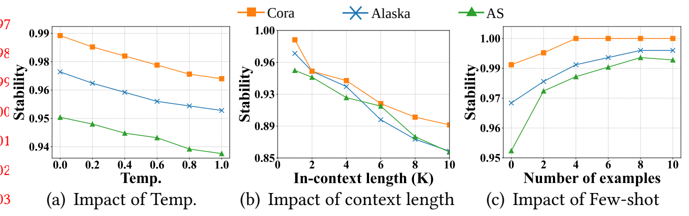
For our proposed method, no fine-tuning is required and we report the results over the entire dataset. Due to the network latency associated with LLM API calls, a direct comparison of our algorithm with PLM-based methods in terms of running time would be unfair, thus we omit time-based comparisons in this analysis.

Experiment Results. Table 16 shows that LLM-CER outperforms PLM-based approaches like Ditto and DeepMatcher across datasets without fine-tuning (w/o FT) or with 20% FT. Even with 80% FT, our method achieves comparable performance at significantly lower costs. On the *Alaska* dataset, our approach yields an ACC of 0.82 and FP-measure of 0.79, surpassing Ditto with 20% FT (ACC: 0.70, FP: 0.62) and DeepMatcher with 20% FT (ACC: 0.65, FP: 0.51). With 80% FT, Ditto (ACC: 0.81, FP: 0.77) and DeepMatcher (ACC: 0.74, FP: 0.70) are close but still trail our method. Non-fine-tuned PLMs perform poorly, with Ditto at 0.64 ACC and 0.55 FP, and DeepMatcher at 0.58 ACC and 0.43 FP. This trend holds across datasets, likely due to our LLM's contextual generalization, unlike PLMs' reliance on fine-tuning. Monetarily, our method is far more efficient. On *Alaska*, our cost is \$0.15, compared to \$260.21 for Ditto (80% FT) and \$260.93 for DeepMatcher (80% FT)—roughly 0.06% of theirs. With 20% FT, costs are \$65.67 for Ditto and \$65.81 for DeepMatcher, still significantly

[5]https://www.runpod.io/pricing

higher. Non-fine-tuned PLMs have closer costs (e.g., \$0.12 for Ditto w/o FT) to ours, but poor performance.

Overall, LLM-CER balances high clustering quality and low cost, outperforming PLMs with 20% FT or w/o FT, and matching 80% FT PLMs at a fraction of the expense. While 80% FT PLMs may slightly excel on some datasets, their marginal gains are cost-prohibitive, making our approach ideal for precision and cost-efficiency.



(a) Impact of Temp. (b) Impact of context length (c) Impact of Few-shot

Figure 9: Stability of LLMs in in-context clustering under different settings

A.6 Evaluating LLM In-context Clustering Stability in ER

To assess the stability of LLMs in ER tasks, we evaluate classification consistency across 50 repeated trials for a 40-query set with ground truth. We query the LLM using identical prompts and record sets, measuring stability via the Stability Score (identical responses divided by total queries, averaged across questions). We vary context length (record set size), LLM temperature, and few-shot example counts, with defaults of temperature 0, zero few-shot examples, and optimal record set size S_s (9, $\approx 1k$ tokens).

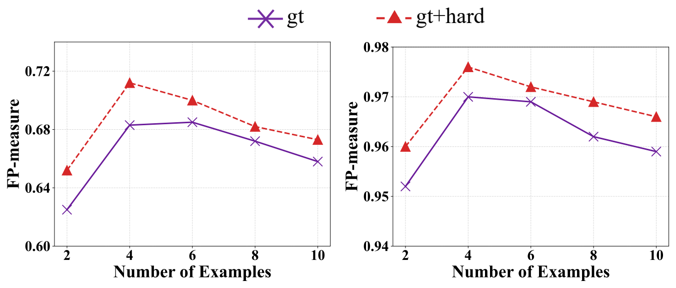
Figure 9 shows stability trends across *Alaska*, *Cora*, and *AS* datasets. The first subplot indicates that Stability Scores slightly decrease as temperature rises from 0 to 1: *Alaska* from 0.97 to 0.95, *Cora* from 0.99 to 0.97, and *AS* from 0.95 to 0.93, reflecting increased randomness. The second subplot shows Stability Scores declining with context length (1k to 10k tokens): *Alaska* from 0.97 to 0.86, *Cora* from 0.99 to 0.89, and *AS* from 0.95 to 0.86, likely due to ambiguity from larger record sets, consistent with §4.2. The third subplot reveals that more few-shot examples improve Stability Scores, enhancing generalization and decision consistency.

In summary, lower temperatures and shorter context lengths enhance LLM stability in ER clustering, while few-shot examples boost consistency. Tuning prompt parameters is crucial for balancing randomness, cognitive load, and task guidance to optimize LLM reliability in ER tasks.

A.7 Few-shot Learning: Out-of-Domain Situation Exploration

In this section, we assess the role of few-shot learning in addressing out-of-domain generalization in end-to-end ER, using the challenging, heterogeneous *Walmart-Amazon* dataset and the simpler, structured *Citeseer* dataset.

To adapt the LLM to domain-specific contexts, we create few-shot prompts with labeled examples from ground truth and hard examples (where the LLM often errs) to guide similarity judgments. We first examine how the number of few-shot examples affects ER performance under two sampling strategies: random sampling from ground truth (gt) and a balanced mix of ground truth and hard examples (gt + hard). Using the optimal example count, we analyze the contributions of few-shot learning and MDG to performance, identifying conditions where each excels in end-to-end ER.



(a) Walmart-Amazon (b) Citeseer

Figure 10: Impact of few-shot example counts

Prompt Template

You are an expert in entity resolution, tasked with clustering records that refer to the same entity based on their attributes. Your goal is to perform in-context clustering of the provided records using the labeled examples as guidance. Each record contains attributes such as title, description, or other identifying features. You must group records that represent the same entity into clusters and return the result as a two-dimensional array, where each sub-array contains the indices of records belonging to the same cluster.

Instructions:

1. Use the provided labeled examples to understand the similarity patterns and attribute importance for clustering records.
2. Output the clustering result strictly as a two-dimensional array (e.g., [[0, 1], [2], [3, 4]]), where each sub-array lists the indices of records that belong to the same cluster.
3. Ensure the output contains only the indices of the records provided in the "Records to Cluster" section, with no duplicates or missing indices.
4. Do not include any explanations or additional text in the output; return only the two-dimensional array.
5. If uncertain, prioritize consistency with the patterns observed in the labeled examples.

Labeled Examples:

{examples}

Records to Cluster:

Record 1: {record 1} Record 2: {record 2} $\cdots$

Output Format:

Return the clustering result as a two-dimensional array, where each sub-array contains the indices of records that refer to the same entity. For example, if Record 1, 2 belong to one entity, and Record 3 belongs to another, the output should be [[0, 1], [2]].

Constraints:

1. The output must be a valid JSON array of arrays (e.g., [[0, 1], [2]]).
2. Every record index from the "Records to Cluster" section must appear exactly once in the output.
3. The indices must be zero-based and correspond to the order of records (Record 1 = index 0, Record 2 = index 1, etc.).
4. The output must not include any text, explanations, or labels outside the two-dimensional array.

Our objective is to evaluate how few-shot learning improves LLM-CER, focusing on its ability to address domain shift in complex, out-of-domain scenarios. We quantify its role in bridging the

Table 17: Quantifying the impact of few-shot learning

Dataset	Metrics	Zero-Shot	Few-Shot w/o MDG	Few-Shot w/ MDG
Walmart-Amazon	ACC	0.61	0.58	**0.77**
	FP	0.56	0.52	**0.71**
	Cost ($)	**0.02**	0.08	0.09
	Tokens (M)	**0.06**	0.24	0.27
	Time (min)	6.25	**5.62**	6.32
	API Calls	398	**332**	365
Cite-seer	ACC	0.88	0.74	**0.90**
	FP	0.95	0.84	**0.97**
	Cost ($)	**0.03**	0.13	0.14
	Tokens (M)	**0.13**	0.58	0.60
	Time (min)	22.68	**21.87**	24.02
	API Calls	1302	**1174**	1214

domain knowledge gap, compare its impact with MDG, and identify conditions where few-shot learning is most effective. Additionally, we determine the optimal number of few-shot examples to maximize clustering performance while maintaining token efficiency and overall effectiveness.

Impact of the Number of Examples in Few-Shot Setting. Figure 10 shows that on the *Citeseer* dataset, both few-shot modes slightly improve FP-measure from 0.95 to a peak of 0.97 with 4–6 examples, with little gain beyond. On the challenging *Walmart-Amazon* dataset, FP-measure starts at 0.63–0.65, rising to 0.70–0.71 with 4–6 examples. The mixed mode (ground truth + hard examples) consistently outperforms the ground truth-only mode on *Walmart-Amazon*, emphasizing hard examples' value in domain-specific settings. However, performance slightly drops with 8–10 examples on both datasets, indicating that overly long contexts may impair the LLM's use of few-shot prompts, reducing ER effectiveness.

Impact of Few-shot Setting. In this experiment, we evaluate the impact of few-shot learning, with and without MDG, compared to zero-shot learning, using a mixed example strategy (ground truth and hard examples) with four few-shot examples for optimal FP-measure and low token usage.

As shown in Table 17, on the domain-specific *Walmart-Amazon* dataset, few-shot learning with MDG significantly boosts ER performance, increasing ACC from 0.61 (zero-shot) to 0.77 and FP-measure from 0.56 to 0.71, though token usage rises from 0.06M to 0.27M. Without MDG, performance declines (ACC: 0.58, FP: 0.52), underscoring MDG's critical role in effective clustering.

On the simpler *Citeseer* dataset, few-shot learning with MDG yields modest gains, with ACC improving from 0.88 to 0.90 and FP-measure from 0.95 to 0.97, while token usage increases from 0.13M to 0.60M. MDG consistently enhances performance across both datasets, providing substantial benefits with minimal additional cost, particularly for *Citeseer*. These findings highlight few-shot learning's value in complex domains and MDG's role as an efficient, effective enhancement.

A.8 Advantages of Merging Similar Clusters

In this section, we conduct an ablation study to assess our similarity-based hierarchical cluster merge strategy in the end-to-end ER pipeline, comparing it to random merging with and without MDG. Random merging experiments are repeated five times, reporting average performance and standard deviations to evaluate stability.

Table 18: Comparison with random clusters merging

Metrics	*Cora*			*Alaska*		
	Sim.	Ran.	Ran. w/o MDG	Sim.	Ran.	Ran. w/o MDG
ACC	**0.90**	0.87 (±0.03)	0.61 (±0.08)	**0.82**	0.79 (±0.02)	0.39 (±0.11)
FP-measure	**0.71**	0.69 (±0.02)	0.57 (±0.07)	**0.79**	0.77 (±0.01)	0.48 (±0.08)
Cost (USD)	**0.03**	0.04 (±0.01)	0.03 (±0.01)	**0.15**	0.16 (±0.02)	0.15 (±0.02)
Tokens (M)	**0.12**	0.15 (±0.04)	0.11 (±0.02)	**0.73**	0.82 (±0.08)	0.72 (±0.07)
Time (min)	**4.95**	5.88 (±0.38)	4.98 (±0.63)	**39.57**	43.52 (±4.08)	39.05 (± 3.87)
# API Calls	**279**	334 (±61)	254 (±32)	**2043**	2308 (±225)	1985 (±188)

Table 18 shows that the similarity-based strategy outperforms random merging (Ran.) and random merging without MDG (Ran. w/o MDG) in efficiency and effectiveness on *Cora* and *Alaska* datasets. On *Cora*, it reduces costs (0.04 to 0.03 USD) and tokens (0.15M to 0.12M) compared to Ran. On *Alaska*, it lowers costs (0.16 to 0.15 USD), tokens (0.82M to 0.73M), time (43.52 to 39.57 minutes), and API calls (2308 to 2043). Ran. w/o MDG matches Ran.'s average efficiency (e.g., 0.03 USD, 254 API calls on *Cora*) but shows high variability (e.g., ±0.01 USD, ±32 API calls), indicating instability. The similarity-based approach's efficiency stems from aligning with S_d and S_v constraints (§4.2), creating cohesive record sets that reduce LLM prediction errors and MDG regenerations, minimizing API usage, tokens, and costs with low variance. Random merging produces less coherent record sets, increasing errors and MDG reliance, with higher variability (e.g., ±61 API calls on *Cora*). Ran. w/o MDG avoids regeneration but is unstable (e.g., ±188 API calls on *Alaska*) due to unstructured merging.

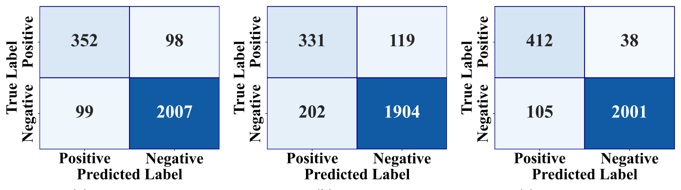
For clustering quality, the similarity-based approach achieves the highest ACC (0.90 on *Cora*, 0.82 on *Alaska*) and FP-measure (0.71 on *Cora*, 0.79 on *Alaska*) with minimal variability, leveraging cohesive record sets. Random merging maintains similar quality (e.g., ACC 0.87 on *Cora*, ±0.03) due to MDG corrections, but Ran. w/o MDG shows lower quality and higher variability without error correction. This underscores MDG's role in robust clustering, while the similarity-based strategy excels in both efficiency and quality.

A.9 Qualitative Analysis of our In-context Clustering-based Method with Other Baselines

We present a qualitative analysis of our in-context clustering-based method compared to two baseline algorithms, BQ [27] and Booster [44], using the *Cora* dataset. To evaluate performance, we employed confusion matrices, measuring True Positives (TP), False Positives (FP), False Negatives (FN), and True Negatives (TN) to highlight differences in clustering accuracy. Specifically, we first apply LSH for preliminary blocking and select a block of 72 records, comprising 6 entities (4 entities with 15 duplicate records each and 2 entities with 6 duplicate records each). We then conduct experiments using our method, BQ, and Booster on this block and evaluate the ER results. Notably, with 72 records, there are total $72 * 71/2 = 2556$ record pairs, which is the sum of the four numbers indicating TP, TN, FP, FN obtained for each method shown in Figure 11.

Experiment Result Analysis. As shown in Figure 11, our algorithm achieves a TP count of 412, surpassing Booster (352) and BQ (331), indicating superior clustering accuracy in correctly grouping records of the same entity. Additionally, our method records an FN count of 38 and an FP count of 105, demonstrating a balanced performance with fewer errors compared to the baselines.

BQ, lacking MDG detection, struggles with transitive errors, often merging records from different entities into the same cluster



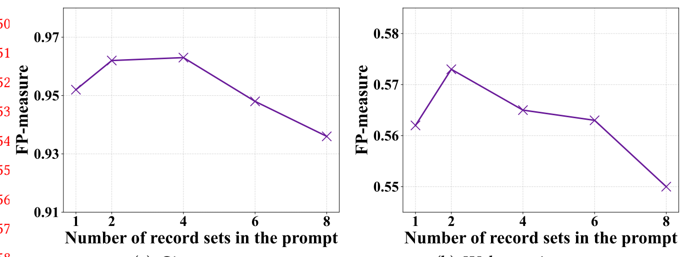
(a) Booster (b) BQ (c) LLM-CER

Figure 11: Qualitative analysis of our experimental results

and splitting records of the same entity across clusters. This results in a high FP (202) and FN (119), significantly degrading its ER quality. In contrast, our MDG-equipped approach effectively identifies and mitigates such errors, enhancing clustering precision. Booster, while adept at distinguishing records from different entities (achieving the lowest FP of 99), tends to over-segment, leading to a higher FN (98) due to excessive differentiation. This over-separation reduces its FP-measure and overall effectiveness. Our algorithm, with an FP (105) close to Booster's and a substantially lower FN, strikes a better balance, yielding a superior overall ER performance.

A.10 Integrating Batch Processing within Our Framework

In this section, we integrate batch processing from the baseline BQ [27] into our framework to enhance end-to-end ER efficiency. Specifically, we incorporate multiple record sets within a single prompt, tasking the LLM with sequentially classifying them. We conduct two experiments: First, we investigate the impact of varying the number of record sets in the prompt on clustering performance. Based on these findings, we evaluate the end-to-end ER results under this setup and quantify the extent of improvement it brings.



(a) Citeseer (b) Walmart-Amazon

Figure 12: FP-measure vs. batch count

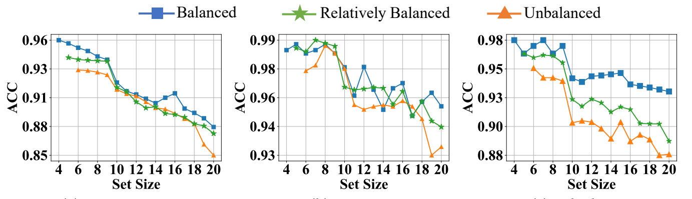
Impact of Batch Size on In-context Clustering Performance. As shown in Figure 12, our experiments evaluate the impact of batch size (number of record sets in a prompt) on in-context clustering performance within our end-to-end ER framework integrated with batch processing. The results reveal that FP-measure initially increases with larger batch sizes, consistent with findings in [27], as the LLM leverages prior classifications to improve subsequent ones. However, as the number of record sets grows further, excessive context length degrades LLM performance, leading to a decline in FP-measure.

End-to-end ER Performance. Based on our prior experiments, we set the batch size to 4, as it optimizes in-context clustering performance while significantly accelerating the algorithm. We evaluate our end-to-end ER framework with batch processing on the *Citeseer* and *Walmart-Amazon* datasets, comparing performance with and without batching. As shown in Table 19, batching improves ACC from 0.88 to 0.90 on *Citeseer*, with FP-measure increasing

Table 19: Impact of batch processing on end-to-end ER

Metrics	*Citeseer*		*Walmart-Amazon*	
	w/ batching	w/o batching	w/ batching	w/o batching
ACC	**0.90**	0.88	**0.64**	0.61
FP-measure	**0.96**	0.95	**0.57**	0.56
Cost (USD)	**0.03**	0.03	**0.02**	0.02
Tokens (M)	**0.12**	0.13	**0.05**	0.06
Time (min)	**6.87**	22.68	**1.72**	6.25
# API Calls	**318**	1302	**92**	398

from 0.95 to 0.96. Additionally, batching reduces computational overhead, lowering processing time more than 4× on both *Citeseer* and *Walmart-Amazon*, while decreasing API calls from 1302 to 318 and 398 to 92, respectively. These improvements stem from the LLM's ability to learn from prior classifications within a batch, enhancing clustering accuracy, and the reduced context overhead from fewer API calls.

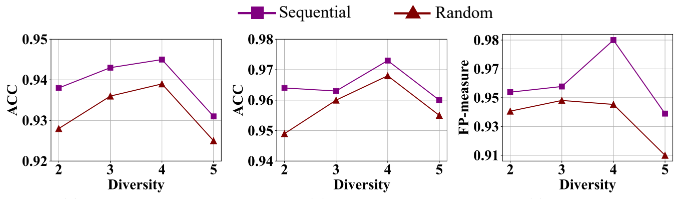


(a) Music 20K (b) Cora (c) Alaska

Figure 13: Clustering performance vs. S_s and S_v (ACC)

A.11 Key Factors Evaluations under ACC Metric

In this section, we present the variation of key factors with respect to S_s, S_d, S_v, and ordering under the ACC metric, as illustrated in the figures below. As shown in Figures 13 and 14, the results are highly consistent with those under the FP-measure. For the *Music 20K*, *Alaska*, and *Cora* datasets, the optimal S_s and S_d values remain 9 and 4, respectively, with S_v performing best when close to 0. Additionally, the records' sequential ordering enhances LLM's in-context clustering performance. These findings indicate that the optimal parameter settings are nearly identical across both clustering metrics.



(a) Music 20K (b) Cora (c) Alaska

Figure 14: Clustering performance vs. S_d, ordering (ACC)