Cost-Effective In-Context Learning for Entity Resolution: A Design Space Exploration

Meihao Fan[†], Xiaoyue Han[†], Ju Fan[†], Chengliang Chai[‡], Nan Tang[§], Guoliang Li[*], Xiaoyong Du[†]
[†]*Renmin University of China*, [‡]*Beijing Institute of Technology*,[§]*HKUST (GZ)*, [*]*Tsinghua University*
{fmh1art, cloverhxy, fanj, duyong}@ruc.edu.cn, ccl@bit.edu.cn, nantang@hkust-gz.edu.cn, liguoliang@tsinghua.edu.cn

2024 IEEE 40th International Conference on Data Engineering (ICDE) | 979-8-3503-1715-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICDE60146.2024.00284

***Abstract*—Entity resolution (ER) is an important data integration task with a wide spectrum of applications. The state-of-the-art solutions on ER rely on pre-trained language models (PLMs), which require fine-tuning on a lot of labeled matching/non-matching entity pairs. Recently, large languages models (LLMs), such as GPT-4, have shown the ability to perform many tasks without tuning model parameters, which is known as *in-context learning* (ICL) that facilitates effective learning from a few labeled input context demonstrations. However, existing ICL approaches to ER typically necessitate providing a task description and a set of demonstrations for each entity pair and thus have limitations on the monetary cost of interfacing LLMs. To address the problem, in this paper, we provide a comprehensive study to investigate how to develop a cost-effective batch prompting approach to ER. We introduce a framework BatchER consisting of demonstration selection and question batching and explore different design choices that support batch prompting for ER. We also devise a covering-based demonstration selection strategy that achieves an effective balance between matching accuracy and monetary cost. We conduct a thorough evaluation to explore the design space and evaluate our proposed strategies. Through extensive experiments, we find that batch prompting is very cost-effective for ER, compared with not only PLM-based methods fine-tuned with extensive labeled data but also LLM-based methods with manually designed prompting. We also provide guidance for selecting appropriate design choices for batch prompting.**

I. Introduction

Entity resolution (ER), which finds entities that refer to the same real-world object, is a crucial task for data cleaning and data integration. Its applications span across various domains, with particular significance in healthcare, finance, customer relationship management, law enforcement, and many others.

The state-of-the-art (SOTA) results in ER are achieved through the application of deep learning methodologies. These methods [1]–[5] involve the utilization of Transformer-based models, which are trained on extensive datasets comprising numerous (*e.g.,* hundreds or thousands) labeled entity pairs.

Standard Prompting and Batch Prompting. Meanwhile, large-scale pre-trained language models (LLMs), such as GPT models [6], have adopted an emerging learning paradigm called *in-context learning (ICL)*, which does not require to update the model parameters of LLMs [7]–[10]. It facilitates effective learning from a restricted set of labeled input context demonstrations, referred to as demonstrations.

Next, we use an example to illustrate the typical way of in-context learning, referred to as **standard prompting**.

Ju Fan and Chengliang Chai are the corresponding authors.

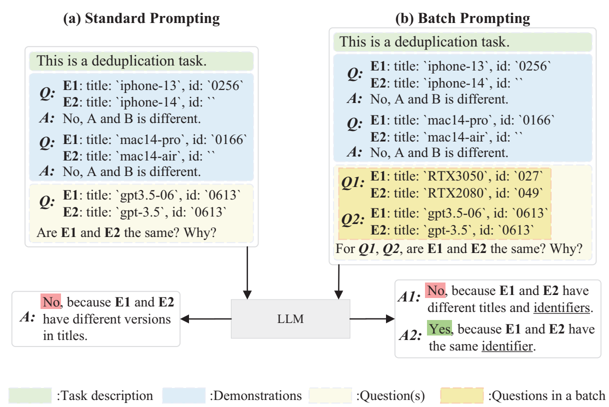


Fig. 1: Standard Prompting and Batch Prompting

Example 1: [Standard Prompting] Figure 1(a) shows standard prompting for ER. The user needs to provide a *task description*, several demonstrations (*i.e.,* the ER pairs with known matching or non-matching labels), and one question (*i.e.,* the ER pair whose label is unknown). An LLM (*e.g.,* GPT-4) can then answer whether the two entities in the question match or not. □

Recent studies have shown that standard prompting for ER is effective on *matching accuracy* [11], [12]. However, a key limitation of this approach is its *monetary cost* of calling APIs of LLMs, as it necessitates providing a task description and a set of demonstrations for each question, as explained in the following example. For instance, consider a real-world ER dataset, *i.e.,* Abt-Buy from Magellan [13]. With a dedicated blocking technique [14], we obtain about 10K pairs to be matched, where each pair has about 100 tokens. Then, querying GPT-4 with standard prompting consisting of 10 demonstrations and 1 question will cost $10,000 \times (100 \times (10 + 1)) \times (0.01/1000) = \$110$, where the pricing of GPT-4 API services is \$0.01 per 1K tokens (https://openai.com/pricing).

To be cost-effective, a natural alternative is to use a set, or a *batch* of questions when prompting the LLMs, which is known as **batch prompting**.

Example 2: [Batch Prompting] As shown in Figure 1(b), the user needs to provide a *task description*, a set of demonstrations, and a set of questions. Subsequently, the underlying LLM can answer whether each question (*i.e.,* entity pair) in this batch matches or not. □

2375-026X/24/$31.00 ©2024 IEEE
DOI 10.1109/ICDE60146.2024.00284

However, despite some very recent attempts of batch prompting for general natural language tasks [15]–[18], as far as we know, exploring the effectiveness of batch prompting for ER under different design choices is not addressed. To bridge the gaps, we provide a comprehensive study to investigate how to develop a cost-effective batch prompting approach to ER. To achieve this, we introduce a batch prompting framework called BATCHER that consists of two main modules, demonstration selection and question batching. Based on the framework, we conduct extensive experiments on well-known ER benchmarks to systemically investigate the following two key questions.

A Design Space Exploration on Both Accuracy and Cost. Due to the importance of ER and the increasing ability of in-context learning, it is highly desired to systemically study batching prompting for ER, under different design choices, on both matching accuracy and monetary cost. To this end, we categorize different choices in question batching and demonstration selection. For question batching, we categorize existing methods as *similarity-based*, *diversity-based* and *random question batching*. For demonstration selection, we classify existing methods as *fixed*, *kNN*-batch and *kNN*-question.

A Covering-based Selection Strategy. While empirically exploring the above design space, we find that existing solutions only consider selecting relevant demonstrations after a batch of questions is determined, without considering whether the selected demonstrations can well cover all questions in a batch. Thus, we further study the problem:"*how to select a batch of questions and how to select a set of demonstrations collectively, such that the demonstrations can well cover all questions which can best guide LLMs to provide answers*"? We model the problem as a set cover problem, which is known as NP-hard. We solve the problem by devising a covering-based selection strategy, which selects demonstrations by considering relevance and coverage. The covering-based strategy aims to generate a labeled demonstration set by selecting the minimum number of demonstrations to cover all questions and then labeling them, and thus can effectively balance the trade-off between accuracy and cost.

A Summary of Experiments. We conduct a thorough evaluation to explore the design space and evaluate our proposed strategies. Our experimental findings reveal insights into accuracy and cost of different batch prompting strategies. (1) Batch prompting can bring 4x-7x cost saving and achieve higher and more stable accuracy than standard prompting. (2) The design choice that combines diversity-based question batching and our proposed covering-based demonstration selection is the most favorable, *i.e.,* achieving the highest accuracy while incurring the lowest cost. (3) Our BATCHER framework is the most cost-effective, compared with not only PLM-based methods [1]–[3] fine-tuned with extensive labeled data, but also LLM-based methods with manually designed prompting [11].

Contributions. We make the following notable contributions.

1) We investigate the design space of batch prompting for ER, by introducing a framework BATCHER and systematically categorizing existing methods for question batching and demonstration selection in Section II.
2) We introduce various question batching strategies (Section III) and demonstration selection methods for ER (Section IV). We devise a novel covering-based selection strategy to connect the process of question batching and demonstration selection in Section V.
3) We empirically evaluate our batch prompting framework BATCHER (Section VI). We make all codes and datasets in our experiments public at Github[1]. Based on the evaluation, we provide insights on the strengths and limitations of various strategies, which guide designing cost-effective ICL approaches to ER.

II. BATCH PROMPTING FOR ENTITY RESOLUTION: A DESIGN SPACE EXPLORATION

A. Entity Resolution

Let T_A and T_B be relational tables with m attributes. Each tuple refers to an entity consisting of m properties, *i.e.,* for a tuple $a \in T_A$, $a = \{\mathtt{attr}_i, \mathtt{val}_i\}_{i=1}^m$ where $\mathtt{attr}_i$ and $\mathtt{val}_i$ denote the i-th attribute name and value respectively. The problem of **entity resolution** (ER) is to identify all the entity pairs $(a, b) \in T_A \times T_B$ that refer to the same object in the real world based on the corresponding attributes.

An end-to-end ER system consists of a blocker and a matcher. The blocker's goal is to identify a subset of $T_A \times T_B$ containing candidate pairs with a high probability of being matched [1], [19], [20] while the matcher's objective is to determine whether each entity pair (a, b) in the above candidate set refers to the same real-world entity (*i.e.,* matching) or not (*i.e.,* non-matching). While the design of an effective blocking strategy is beyond the scope of this paper, we employ a widely accepted blocking method [1], [20], [21] to produce the aforementioned pairwise candidate set.

B. In-Context Learning

In-context learning (ICL). It refers to the capability of LLMs to learn from a few demonstrations in the input context without any parameters updating [6].

ICL for ER. Given any entity pair (a_i, b_i), we utilize a serialization function to serialize it into a text by concatenating all attribute names and values within the entity pair,

$$\begin{aligned} \mathcal{S}((a_i, b_i)) &= \mathcal{S}(a_i)\,\mathtt{[SEP]}\,\mathcal{S}(b_i) \\ \mathcal{S}(e) &= \mathtt{attr}_1 : \mathtt{val}_1 ... \mathtt{attr}_m : \mathtt{val}_m \end{aligned} \tag{1}$$

where `[SEP]` is used to separate entities of a pair and $\mathcal{S}(\cdot)$ denotes the serialization function of each data entity e.

Then, we construct a prompt consisting of a task description `Desc`, several serialized pairs with golden labels `Demos` (denoted as demonstrations in this paper) and a serialized pair `Question` to be queried (denoted as question). By feeding them to an LLM G, we generate the target y with the next

[1]https://github.com/fmh1art/BatchER

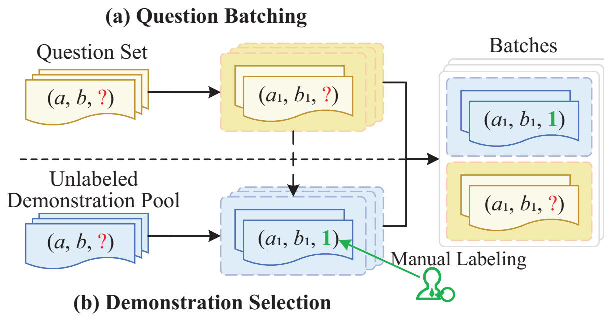


Fig. 2: Our proposed BATCHER framework, which consists of (a) question batching and (b) demonstration selection.

token prediction, which can be regarded as a conditional text generation problem:

$$y = \arg\max_{y \in Y} P_G(y \mid \overbrace{\texttt{Desc} \oplus \texttt{Demos}}^{\text{supervision of ER task}} \oplus \texttt{Question}) \quad (2)$$

where $Y = \{\text{matching, non-matching}\}$ is the label space.

As Eq. 2 shows, G receives the task's supervision only from a pre-defined task description (Desc) and the concatenated demonstrations (Demos). Usually, In-context learning is highly sensitive to the provided demonstrations and different question selection strategies will bring huge fluctuations in performance [22], [23]. Thus, a comprehensive exploration for selecting beneficial demonstrations deserves a detailed design.

C. The BATCHER Framework and Design Space

Despite the good accuracy of ICL [15], [18], [24], the cost of finance may be very expensive, since most LLM companies such as OpenAI charge users based on the token consumption.

To reduce the cost of interfacing LLMs while maintaining high accuracy, batch prompting is proposed, which allows to query a batch of questions with several demonstrations and asks LLM to make multiple predictions in one interface [25].

Example 3: Figure 1 shows the difference between Standard Prompting and Batch Prompting. Although both select two demonstrations for the in-context learning of LLMs, Batch Prompting asks LLMs to answer 2 questions at one interface, which approximately saves tokens of 2 demonstrations and 1 task descriptions. Naturally, the more questions we put in a batch, the more cost of interfacing LLMs will be reduced. □

The BATCHER Framework. We can observe that two critical components in the prompt of Batch Prompting are in-context demonstrations and questions. Thus, to design effective Batch Prompting, we introduce a framework called BATCHER that consists of the modules of in-context demonstration selection and question batching, as shown in Figure 2. The BATCHER framework takes a set of questions, *i.e.,* entity pairs $\{q\}$ as input, and aims to produce a set of *batch prompts*, which are then fed into an LLM. As a prompt needs in-context demonstrations, BATCHER also considers a set of entity pairs without matching/non-matching results as an Unlabeled Demonstration Pool. In this section, we first formally define the above two modules and then systematically explore the design space of Batch Prompting for ER by categorizing each individual module in the BATCHER framework.

TABLE I: **A Design Space Exploration**

Modules	Categorization
Question Batching	(1) Random
	(2) Similarity-based
	(3) Diversity-based
Demonstration Selection	(1) Fixed
	(2) kNN-batch
	(3) kNN-question
	(4) Covering-based (Our proposal)

- **Question Batching.** Considering a Question Set M of questions to be queried, Question Batching aims to iteratively select b questions and group them into one batch $B_i = \{q_j\}_{j=1}^{b}$. To ensure all questions will be queried at least once, the union set of all batches should equal to the original question set, satisfying $\bigcup B_i = M$.
- **Demonstration Selection.** Considering a large pool of unlabeled demonstrations D_u from which we iteratively select several data points $\{d_j\}$ for each batch B_i. We assume manual annotation will be adopted for the selected data to generate labeled demonstrations $D_i = \{(d_j, y)\}$ which will be used to guide LLMs to make predictions for batched questions.

To put the above together, the BATCHER framework takes a Question Set M and an Unlabeled Demonstration Pool D_u as input and outputs a set of question batches $B = \{B_i\}$ along with a set of corresponding demonstrations $D = \{D_i\}$, satisfying $\bigcup B_i = M$ and $\bigcup D_i \subseteq D_u$.

A Design Space Exploration. To utilize in-context learning for ER, several challenges should be addressed.

First, question batching and demonstration selection require a feature extractor to map questions and demonstrations into vectors, which facilitates the measurement of their relevance. However, the ER data typically consists of structured tables with multiple attributes, which makes the process of feature extraction more complicated. Existing semantics-based feature extractors [26]–[28] simply concatenate words from various attribute-value pairs into a sentence, e.g., "title:Rashi, album:Here…, genre:Dance…", and may have a limitation of neglecting the *structure* information of the tuples to be matched. Therefore, this motivates us to investigate *structure-aware feature extractors* in our design space.

Second, in-context learning shows stable and remarkable performance in Standard Prompting with relevant demonstration selection [18], [29], [30]. However, previous methods typically adopt a kNN strategy to select k most "relevant" demonstrations for each question. Naturally, this strategy may result in a large number of demonstrations, especially for large ER datasets with many pairs, which would incur significant data annotation expenses. Thus, effective demonstration selection strategies still lack a comprehensive investigation on the trade-off between accuracy and cost.

Third, the choice of batching strategy is of great significance in downstream performance, which is neglected by existing studies and thus deserves in-depth investigation.

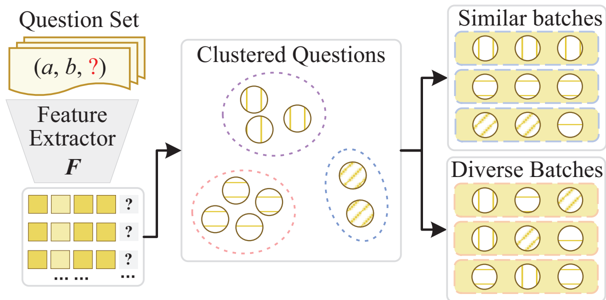


Fig. 3: Question Batching Framework, where circles with different stripes represent questions in different clusters.

To address the challenges, we propose a categorization of design choices for each module in BATCHER, which forms a design space as shown in Table I. We first explore strategies for the question batching module and discuss different feature extractors used for measuring relevance among questions (Section III). Subsequently, we investigate methods for selecting demonstrations for a batch (Section IV). We note that BATCHER is extensible, *i.e.,* it is possible to incorporate new modules, new categories, or new methods or variants of existing methods. Moreover, it is possible to define the search space from a different angle; that is, we contend that our proposal is rational, but may not be unique.

III. Question Batching

This section explores the question batching strategies, as shown in Table I. To this end, we first describe a general framework of question batching, as illustrated in Figure 3. Specifically, given a Question Set M of entity pairs, the framework produces batches of questions in three steps.

- **Feature Extraction.** We first use a Feature Extractor to cast the questions into feature vectors
- **Question Clustering.** We then adopt an unsupervised clustering algorithm such as DBSCAN or K-Means to group the questions into clusters.
- **Question Batching.** We finally group questions into batches based on the clusters using various strategies.

In the remaining of this section, we mainly introduce three representative batching strategies, including similarity-based question batching, diversity-based question batching and random question batching, which have been adopted by previous studies [25], [31] (Section III-A). Next, as feature extraction and distance measurement (for clustering) are involved in the batching process, we then discuss two feature extraction methods in Section III-B. Note that, for question clustering, we adopt DBSCAN [32], as the algorithm achieves the best performance. Due to the space limit, this section does not discuss various clustering algorithms.

A. Batching Strategies

Given clustered questions, BATCHER generates batches based on the following three representative strategies.

Similarity-based Question Batching. The intuition of this strategy is to group *similar* questions within the same clusters into the same batch. To this end, we iteratively select b (*i.e.,* batch size) questions from the same cluster to form a batch, to ensure that questions in the same batch have similar feature vectors to each other. In particular, during the final stage of batch generation, some clusters may contain questions fewer than the required batch size b. In such case, we select the largest remaining cluster, denoted as $C_{\max}$. We then seek to pair it with another cluster whose size exactly matches $b - |C_{\max}|$, to form a complete batch. If no such cluster exists, we opt for the next largest cluster, randomly selecting $b - |C_{\max}|$ elements from them to form a batch in conjunction with $C_{\max}$.

Diversity-based Question Batching. The intuition of this strategy is to group questions that are from diversified clusters into a batch. In this batching strategy, batches are also generated in two stages. Firstly, we ensure batch diversity by selecting one question from each of b different clusters, such that the questions in different batches have obvious differences in feature vectors from each other. Then, when the batching process almost completes, we may encounter scenarios where the number of available clusters is less than b. In such instance, we simply ensure the diversity of batches generated from a limited number of clusters by selecting questions from remaining clusters in a round-robin manner.

Example 4: [Question Batching] Consider the questions in Figure 3. We denote the three clusters as $C_a = \{q_i^a\}_{i=1}^2$, $C_b = \{q_i^b\}_{i=1}^3$, and $C_c = \{q_i^c\}_{i=1}^4$, respectively.

(1) For similarity-based question batching, we sequentially select C_b and C_c, forming batches $B_1 = \{q_1^b, q_2^b, q_3^b\}$ and $B_2 = \{q_1^c, q_2^c, q_3^c\}$. Subsequently, from the remaining clusters $C_a = \{q_1^a, q_2^a\}$ and $C_c = \{q_4^c\}$, we choose the larger cluster C_a and combine it with C_c to create $B_3 = \{q_1^a, q_2^a, q_4^c\}$.

(2) For diversity-based question batching, we can generate diverse batches $B_1 = \{q_1^a, q_1^b, q_1^c\}$ and $B_2 = \{q_2^a, q_2^b, q_2^c\}$ in the initial stages by iteratively selecting one question from C_a, C_b and C_c. Then with remaining clusters $C_b = \{q_3^b\}$ and $C_c = \{q_3^c, q_4^c\}$, we sequentially select questions from C_c, C_b and C_c to generate the final batch $B_3 = \{q_3^c, q_3^b, q_4^c\}$. □

Random Question Batching. We also consider a straightforward random question batching strategy, which is commonly adopted in the existing works [25], [31]. In this approach, each batch is formed by randomly selecting questions from the remaining question set. Due to this randomness, the generated batches may contain a mix of both similar and dissimilar questions. This implies that a random batch, to some extent, represents a middle ground between a similar batch and a diverse batch.

B. Feature Extractor

The process of batching questions in the previous section relies on the utilization of a feature extractor to convert questions into corresponding feature vectors. Subsequently, these feature vectors are used to calculate distances between questions and then serve as the basis for the clustering procedure. Formally, given a set of questions M, we need

title	album	genre		title	album	genre
Rashi	Here Comes the Fuzz	Dance,Music ,Hip-Hop	q_1	Rashi	Here Comes The Fuzz [Explicit]	Music
Act My Age	FOUR	Pop, Music	q_2	Change My Mind	Take Me Home	Pop

Fig. 4: An example instance of Entity Resolution.

to define a feature extractor f and a distance function $\mathtt{dist}$, and thus the distance of any two questions q_i and q_j can be calculated via $\mathtt{dist}(\mathbf{v}_i, \mathbf{v}_j)$ between the two feature vectors, *i.e.,* $\mathbf{v}_i$ and $\mathbf{v}_j$. We notice that the distance function can be further defined by a variety of ways, such as Euclidean distance or cosine similarity (distance). In our experiments, we define the distance function based on the Euclidean distance, which achieves the best performance among others.

Next, we introduce two types of feature extractors, one based on semantics and the other being structure-aware.

Semantics-based Feature Extractor. Semantics-based feature extractor utilizes a pre-trained language model (PLM) to encode each serialized question. For ER task, as all questions are structural pairs, *i.e.,* with multiple attributes, we first use the serialization function defined in Eq.(1) to generate serialized questions and pass it to a PLM, such as SBERT [26] and RoBERTa [27] to generate embedding-based representations. Formally, given a question q, the feature vector $\mathbf{v}$ can be generated as $\mathbf{v} = \mathtt{Encoder}(\mathcal{S}(q))$, where $\mathtt{Encoder}$ denotes the encoding function of a PLM. Although the above feature extractor formulates the relevance as semantic distance, it may have the limitation of ignoring the structural information. This inspires us to introduce another feature extraction method, which can capture structural similarity to model relevance.

Structure-aware Feature Extractor. Structure-aware feature extractor employs a string similarity function to map attribute-matching signals of two entities of a question into a low-dimensional space, which enables the generated feature vectors to capture structural information and task-related knowledge. Formally, given a structural pair (a, b), we derive the feature vector by calculating the similarities of attributes between a and b. Since attribute values typically take a string format, we can compute similarity s_i on attribute $\mathtt{attr}_i$ with string similarity function, e.g., Levenshtein ratio and Jaccard.

Using the Jaccard similarity, we tokenize $\mathtt{val}_i^a$ and $\mathtt{val}_i^b$ into sets and compute the similarity as $s_i = \mathtt{JAC}(\mathtt{val}_i^a, \mathtt{val}_i^b) = \frac{|\mathtt{val}_i^a \cap \mathtt{val}_i^b|}{|\mathtt{val}_i^a \cup \mathtt{val}_i^b|}$, where $\mathtt{val}_i^a$ represents the tokenized set of attribute value $\mathtt{val}_i$ of entity a and $|\mathtt{val}_i^a|$ represents corresponding token-set size.

The Levenshtein ration (LR) derives from the Levenshtein edit distance (LED) [33], representing the minimum number of edits needed to transform one string into another, as $s_i = \mathtt{LR}(\mathtt{val}_i^a, \mathtt{val}_i^b) = 1 - \frac{\mathtt{LED}(\mathtt{val}_i^a, \mathtt{val}_i^b)}{s}$, where $\mathtt{LED}$ is the Levenshtein edit distance function and s represents the sum of string length of $\mathtt{val}_i^a$ and $\mathtt{val}_i^b$.

Thus, given a question q with entity pair (a, b), the feature vector $\mathbf{v}$ can be generated by concatenating the similarities of all attributes make $\mathbf{v} = \{s_i\}_{i=1}^m$.

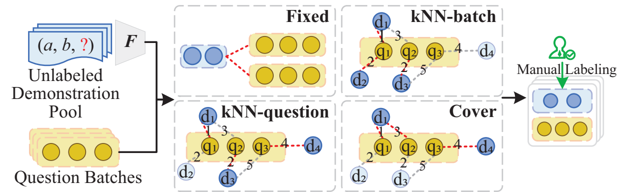


Fig. 5: Demonstration Selection Framework, where blue circles and yellow circles represent demonstrations and questions respectively, and values on edges represent distances.

IV. Demonstration Selection

Figure 5 illustrates the framework of demonstration selection and describes four demonstration selection methods. Given an Unlabeled Demonstration Pool D_u and a set of generated question batches B, demonstration selection aims to select beneficial in-context demonstrations D_i for each batch $B_i \in B$, which will be then manually labeled. To further specify the concept of four demonstration selection methods, we give an illustration for each method. For simplicity, we only consider two closest demonstrations for each question.

A. Fixed Demonstration Selection

A basic idea is to sample fixed K demonstrations and then allocate them to every batch. In Figure 5, we generate two fixed demonstrations by randomly sampling from the unlabeled demonstration pool and allocate these two demonstrations to each batch. This method brings a fixed annotation cost. However, existing studies show that random demonstrations may incur unstable performance of ICL [22], [23].

B. kNN-batch Demonstration Selection

Similar to the strategy in Standard Prompting of recommending top k most relevant demonstrations [34], this strategy selects the k most relevant demonstrations for each batch. Specifically, we first define the relevance between B_i and d based on the distance function $\mathtt{dist}$ defined in Section III-B:

$$\mathtt{dist}^*(B_i, d) = \min_{q_j \in B_i} \mathtt{dist}(q_j, d) \quad (3)$$

which shows that the relevance between B_i and d calculated as the distance between the d and the *closest* question in B_i. Based on this, we can use the $k\mathtt{NN}$ algorithm to generate k in-context demonstrations for B_i.

Take Figure 5 as an example. Considering $k = 3$, kNN-batch selects d_1, d_2 and d_3 as demonstrations. We notice that some questions, *e.g.,* q_3, may not be assigned with relevant demonstrations, if they have larger distances to the demonstrations compared with other questions.

C. kNN-question Demonstration Selection

To address the above issue, we investigate a demonstration selection method kNN-question that select the k most relevant demonstrations for each question in the batch. This is based on the assumption that, since relevant demonstrations are beneficial when querying the individual question, the set of relevant demonstrations will also benefit when querying the whole batch. Formally, considering a batch $B_i = \{q_i\}_{i=1}^b$, the in-context demonstration set D_i can be generated as $D_i = \bigcup_{q_j \in B_i} k\text{NN}(q_j, D_u)$. Figure 5 illustrates the basic idea of the kNN-question method where we set $k = 1$ and select d_1 for p_1, d_3 for p_2 and d_4 for p_3 respectively.

Although this method can improve accuracy of ICL, it may have a limitation of incurring large monetary cost. Also, it may generate long prompts which could lead to long text comprehension issue and input length overrun. Specifically, for large b and k, the input length (*i.e.,* number of tokens) for the LLM is more likely to exceed the length limit, *e.g.,* 4096 tokens for GPT-3.5. For example, given $b = 8$ and $k = 3$, all the batches on our 8 ER datasets exceed the input length limit. Moreover, some existing studies [25] have shown that such long inputs increase the difficulty for LLMs to understand the questions, resulting in a degradation of performance.

V. Covering-based Demonstration Selection

A key limitation of kNN-question and kNN-batch is that they may incur substantial labeling cost, which is caused by labeling the selected demonstrations. To mitigate this, we introduce a new approach based on the idea of using demonstrations to **"cover"** all questions in the batch B_i where "cover" means that the distance between question q and demonstration d is smaller than a threshold t. This is based on the assumption that the beneficial demonstrations are a set of relevant data points and all beneficial to a given question. In Figure 5, we assume that demonstrations with a shorter distance than 5 can be regarded as a beneficial reference when answering the question. Thus, we first select d_1 to cover q_1 and q_2. Then, to cover q_3, the demonstration d_4 is selected.

More formally, the covering-based method aims to address two main problems, namely Demonstration Set Generation and Batch Covering. First, Demonstration Set Generation aims to reduce the *labeling cost*. To this end, we need to select a minimal subset of demonstrations from an unlabeled demonstration pool to cover all the questions of all batches. Second, given the selection results, each batch may be assigned with multiple sets of demonstrations, which have varying numbers of tokens. Thus, we devise Batch Covering to further select the demonstration set with the minimum token number, so as to reduce the *API cost*. We also empirically evaluate the effect of Batch Covering, and find that this procedure can achieve $11.11\% - 21.58\%$ for API cost reduction on our datasets in the experiments.

A. Demonstration Set Generation

Definition. Given a Question Set M containing all questions to be queried, an unlabeled demonstration pool D_u and a non-negative distance threshold t, we need to select a subset of demonstrations $D_s \subset D_u$, satisfying $\forall q \in M$, exists at least one $d \in D_s, \texttt{dist}(q, d) < t$. The goal is to minimize the size of selected Demonstration Set $|D_s|$. We can prove the Demonstration Set Generation Problem to be NP-hard by a reduction from the Set Cover Problem, which is proven to be NP-hard [35]. We omit the proof due to the space limit.

Algorithm 1 Demonstration Set Generation/Batch Covering

Input: Set of questions Q, set of demonstrations D, nondecreasing value function f, weight function w.
Output: set of selected demonstrations D_s.

1: $D_s \leftarrow \varnothing$
2: **while** $f_Q(D_s) \neq f_Q(D)$ **do**
3: $\quad d \leftarrow \arg\max_{d \in D} \frac{f_Q(D_s \cup \{d\}) - f_Q(D_s)}{w(d)}$
4: $\quad D_s \leftarrow D_s \cup \{d\}$
5: **end while**

Greedy Algorithm. To efficiently address the Demonstration Set Generation Problem, we propose a greedy-based algorithm. To start with, we define a non-decreasing value function $f_Q(D_s) = \sum_{i=1}^{|M|} z_i$ to measure the value of intermediate demonstration set D_s, where for $q_i \in Q$, $z_i = 1$ if $\min_{d_j \in D_s} \texttt{dist}(q_i, d_j) < t$, otherwise, $z_i = 0$. Generally, the value function calculates the number of covered questions by D_s. Then, taking the value function f, set of questions M, and an unlabeled demonstration set D_u as input, we iteratively select the most efficient demonstration. Efficiency is defined by the ratio of the incremental value a demonstration contributes to the intermediate Demonstration Set D_s relative to its weight. For the Demonstration Set Generation Problem, we set the weights of all demonstrations to be 1, since selecting any demonstration brings us equivalent cost. The pseudo-code is shown in Algorithm 1.

We first initialize the demonstration set D_s to an empty set (line 1). Then we determine whether the value of intermediate set D_s meets the value of full unlabeled demonstration pool D_u (line 2) which is probably equaled to $|M|$ with a large enough pool size. If not, we will iteratively select the most efficient demonstration and add it to the intermediate demonstration set (lines 3$\sim$4). Otherwise, the algorithm ends and outputs the selected demonstration set D_s (line 5).

Assuming that the optimal sum of Demonstration Set Generation Problem is OPT and the final sum of our greedy algorithm is ans^*, we have $ans^* \leq H_k \cdot OPT$, where $H_k = \sum_{i=1}^{k} \frac{1}{i}, k = \max_{d_i \in D_s} f_Q(\{d_i\})$. A complete proof can be found in [36].

For Demonstration Set Generation problem, by setting a target function and designing a greedy algorithm to optimize it, we can generate an effective solution, that is, selecting a small number of demonstrations to cover all the questions to be queried, thereby greatly reducing the labeling cost.

B. Batch Covering

Next, based on the generated Demonstration Set, we will allocate relevant demonstrations to each batch, so as to covering all the questions in the batch. At this stage, we ask a question: Is there further optimization space when allocating demonstrations? To answer this question, we consider an example of a Question Set $M = \{q_1, q_2, q_3, q_4\}$ and a labeled Demonstration Set $\{d_1, d_2\}$. We have d_1 covers q_1, q_2, q_3 and d_2 covers q_2, q_3, q_4. Given a batch $B_i = \{q_2, q_3\}$, we need to allocate demonstrations to cover all questions in B_i. It can be seen that, at this time, whether allocating d_1 or d_2

TABLE II: **Statistics of Datasets.**

Dataset	Domain	# Attr.	# Pairs	# Matches
Walmart-Amazon (WA)	Electronics	5	$10,242$	962
Abt-Buy (AB)	Product	3	9575	1028
Amazon-Google (AG)	Software	3	$11,460$	$1,167$
DBLP-Scholar (DS)	Citation	4	$28,707$	$5,347$
DBLP-ACM (DA)	Citation	4	$12,363$	$2,220$
Fodors-Zagats (FZ)	Restaurant	6	946	110
iTunes-Amazon (IA)	Music	8	532	132
Beer	Beer	4	450	68

can cover all questions in the batch. Therefore, although we only consider covering each question once when generating the Demonstration Set, there is still room for choice when allocating demonstrations for each batch.

Definition. Given a batch B_i of questions $B_i = \{q\}$, a generated demonstration set D_s and a non-negative distance threshold t, we need to select a set of demonstrations $D_i \subset D$, satisfying $\forall q \in B_i$, exists at least one $d \in D_i$ such that $\mathtt{dist}(q, d) < t$. The goal is to minimize the weight of selected demonstrations $\sum_{d \in D_i} w(d)$. We define the weights of demonstrations as token numbers, and the goal of our problem is to find a demonstration set to cover the batch with minimum token assumption. Also, we can prove the batch covering problem as an NP-hard problem.

Greedy Algorithm. We again use Algorithm 1 to address the Batch Covering Problem. We use the same value function defined in section V-A and define the weights of demonstrations as token numbers. Taking the value function f, batch B_i of questions, the generated Demonstration Set D_s, and weight function w as input, the algorithm will output the allocated demonstration set D_i for batch B_i. This greedy algorithm yields an approximation ratio of $\ln|B_i| - \ln\ln|B_i| + \Omega(1)$. A complete proof can be found in [36].

For Batch Covering Problem, by defining the weights of demonstrations as token numbers and formulating it as Weighted Set Cover Problem, we can generate an effective solution with the minimum sum of tokens of batch prompts, thereby reducing the interfacing API cost.

VI. Experiments

This section evaluates our batch prompting framework BatchER investigated in this paper.

A. Experimental Setup

Datasets. We evaluate our proposed batch prompting framework BatchER using well-adopted benchmarking datasets from Magellan benchmark [13], which range from a variety of domains, such as product, software, and citation. Table II provides detailed statistics of the datasets. Specifically, each dataset contains entities from two relational tables with multiple attributes, and a set of labeled matching/non-matching entity pairs. Take the Amazon-Google (AG) dataset as an example: it contains software products from Amazon and Google with three attributes (`title`, `manufacturer`, `price`), and has $11,460$ entity pairs where $1,167$ pairs are matches. For fair comparison, the set of labeled entity pairs is split into train, validation and test sets with a ratio of 3:1:1, which is consistent with existing ER studies [1], [5], [14].

Evaluation Metrics. In this paper, we evaluate the performance of ER approaches on both *Accuracy* and *Cost*.

(1) **Matching Accuracy.** Following existing ER studies [1]–[3], [14], we use F1 score to measure the matching accuracy of an ER approach. Specifically, let $\mathtt{TP}$, $\mathtt{FP}$, $\mathtt{FN}$ denote the number of true positives (*i.e.,* matching pairs correctly identified), false positives (non-matching pairs incorrectly identified) and false negatives (matching-pairs incorrectly omitted) respectively. Then, we can respectively compute Precision and Recall as $\mathtt{P} = \mathtt{TP}/(\mathtt{TP}+\mathtt{FP})$ and $\mathtt{R} = \mathtt{TP}/(\mathtt{TP}+\mathtt{FN})$, and derive F1 score as harmonic mean of Precision and Recall, *i.e.,* $\mathtt{F1} = 2\cdot\mathtt{P}\cdot\mathtt{R}/(\mathtt{P}+\mathtt{R})$.

(2) **Monetary Cost.** We evaluate an approach by considering its incurred monetary cost, which consists of two parts.

- **API Cost** measures how much an approach pays for calling the API of a proprietary LLMs (*e.g.,* GPT-3.5 and GPT-4). In particular, the API is priced per token. For example, according to the pricing of GPT API services[2], GPT-4 incurs \$0.01 / 1K tokens for input texts.
- **Labeling Cost** measures how much an approach pays for labeling entity pairs to prepare demonstrations. To calculate the cost, we refer to the latest rates on the crowdsourcing platform, Amazon Mechanical Turk (AMT) [3] for text data labeling, which is \$0.08 per labeling task. Following the existing crowdsourcing approach to ER [37], we group ten entity pairs into one labeling task and ask the crowd to label them in batch. Based on this, we estimate the cost of labeling one entity pair as \$0.008.

Baselines. We consider two types of baselines. The first type is the SOTA PLM-based approaches to ER, including Ditto [1], JointBert [2] and RobEM [3]. The other type is the LLM-based approaches [11] to ER via in-context learning, equipped with manually designed prompts. We briefly describe the methods.

(1) **Ditto** [1] is a well-recognized PLM-based approach to ER, which utilizes pre-trained language model RoBerta [27] and employs labeled entity pairs for fine-tuning. We use the code and default setting of Ditto in its original paper [1].

(2) **JointBert** [2] is a dual-objective training method for BERT that combines binary matching and multi-class classification for entity matching. We use the code provided from [38]. We select the uncased base versions of BERT for JointBert and set all the hyper-parameters as default as in the original paper.

(3) **RobEM** [3] is a recent work that investigates the robustness of PLM-based ER methods with varying data distributions and identifies data imbalance as a critical issue. To solve this, it proposes simple yet effective modifications to enhance PLMs and achieves superior performance on ER. We run its original code from [39] and keep all the setting as default.

(4) **ManualPrompt** [11] is a pioneering initiative that uses LLMs (GPT-3) for ER as well as other data wrangling tasks.

[2]https://openai.com/pricing
[3]https://www.mturk.com/

Similar to our work, it also employs in-context learning to answer ER questions. However, the key difference is that ManualPrompt utilizes standard prompting (*i.e.,* asking questions one by one) and manually designed demonstrations. We reproduce the results of ManualPrompt by using its original code and following its instruction at Github[4]. We notice there exist performance discrepancies between the reproduced results and the results reported in its original paper. This may be attributed to the different versions of the underlying LLMs. While the original paper's results were obtained using "text-davinci-002", the current version of its code has changed the default LLM to "text-davinci-003". Moreover, different from BATCHER, for each dataset, Manual Prompting directly provides a set of demonstrations designed by experts, instead of utilizing data annotation. Thus, we can not provide data annotation expenses for Manual Prompting.

Implementation Details. We briefly present the implementation details of our proposed framework as follows.

(1) **Batch Prompting**. We implement the design choices in Table I for question batching and demonstration selection, and compare them on matching accuracy and monetary cost. For question batching, we set the batch size to 8, which ensures that none of the design choices exceeds the maximum token limit of LLMs' text input, and employ DBSCAN [32] for question clustering. For fair comparison of demonstration selection strategies (*i.e.,* fixed, kNN-batch and kNN-question), we choose 8 demonstrations for each batch. For our covering-based strategy, we calculate the threshold t by first computing the distances between all questions and then taking the 8-th percentile as t since it can achieve great balance between cost and accuracy: with smaller t, the labeling cost will become larger while larger t will degrade the matching accuracy.

(2) **Large Language Models**. In our experiments, we use GPT-3.5-turbo-0301, or GPT-3.5-03 for short, as the default LLM, where 0301 means that the model version was finalized on March 1st. In particular, according to the guideline of OpenAI[5], we set the temperature parameter of GPT-3.5-03 as 0.01. Moreover, we also investigate other proprietary LLMs, GPT-3.5-turbo-0613 (or GPT-3.5-06 for short) and GPT-4-1106-preview (or GPT-4 for short), as well as a very recent open-source LLM, LLama2-chat-70B [40].

B. Comparing Batch Prompting with Standard Prompting

Exp-1: How does *Batch Prompting* compare with *Standard Prompting*? We conduct experiments to compare batch prompting with standard prompting on matching accuracy and monetary cost. For fair comparison, we use the same 8 fixed demonstrations, which are selected randomly, for both approaches. In this case, we only need to consider the API cost, as labeling costs of both approaches are the same. Moreover, we run the experiments for three times, and compute mean and standard variance of the obtained F1 scores.

[4]https://github.com/HazyResearch/fm_data_tasks
[5]https://platform.openai.com/docs/api-reference/completions

TABLE III: **Comparing Batching Promting with Standard Prompting on Matching Accuracy and API Cost (The best results are bolded).**

Dataset	Metric	Standard Prompting	Batch Prompting
WA	F1	$67.54_{\pm 8.08}$	$\mathbf{78.92}_{\pm 0.32}$
	API ($)	1.43	**0.33**
AB	F1	$65.70_{\pm 10.81}$	$\mathbf{85.79}_{\pm 1.01}$
	API ($)	1.10	**0.24**
AG	F1	$53.72_{\pm 3.88}$	$\mathbf{61.07}_{\pm 0.83}$
	API ($)	1.29	**0.29**
DS	F1	$75.08_{\pm 6.03}$	$\mathbf{80.79}_{\pm 1.72}$
	API ($)	5.31	**1.22**
DA	F1	$85.96_{\pm 4.45}$	$\mathbf{92.10}_{\pm 0.88}$
	API ($)	2.93	**0.63**
FZ	F1	$89.95_{\pm 3.67}$	$\mathbf{94.13}_{\pm 1.11}$
	API ($)	0.19	**0.04**
IA	F1	$90.59_{\pm 0.94}$	$\mathbf{91.75}_{\pm 0.84}$
	API ($)	0.06	**0.01**
Beer	F1	$\mathbf{91.11}_{\pm 2.22}$	$88.31_{\pm 2.60}$
	API ($)	0.07	**0.01**

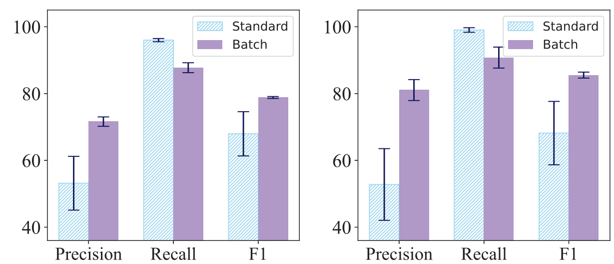


Fig. 6: Comparing Batch Prompting (Batch) and Standard Prompting (Standard) on Recall, Precision and F1.

The experimental results are reported in Table III. We can see that, batch prompting significantly outperforms standard prompting on both accuracy and cost. First, batch prompting improves F1 score by **1.3%-30.6%** on all datasets except Beer. The reason that batch prompting performs worse than standard prompting on the Beer dataset is that the dataset is very small (with only 91 pairs for testing), and the two methods actually output very similar matching results. Moreover, we can also observe that batch prompting is more stable than standard prompting, *i.e.,* achieving much smaller standard variance. Second, compared with standard prompting, batch prompting can achieve **4x-7x** cost saving on API callings.

While it is intuitive that batch prompting can save cost, it is somewhat surprising that it can also significantly improve the accuracy. Thus, we conduct a detailed analysis to report Precision and Recall on WA and AB datasets, as shown in Figure 6. We can see batch prompting achieves much higher Precision than standard prompting, while their Recall scores are comparable. This is mainly attributed to the batching mechanism, where the LLM can refer to not only the provided demonstrations, but also the answers generated for previous questions within the same batch. This may help the LLM to identify some key characteristics that are useful to differentiate the entities. For example, on the WA dataset, batch prompting can help the LLM to focus on a critical attribute "`modelno`", and enable the LLM to understand entities with different "`modelno`" tend to be non-matching pairs.

Finding 1: Batch prompting can not only bring 4x-7x cost saving, but also achieve higher and more stable matching accuracy than standard prompting.

C. Exploring Design Space of Batch Prompting for ER

Exp-2: What are effective strategies in our design space of question batching and demonstration selection? We explore the design space shown in Table I by comparing the 12 combinations of three question batching methods and four demonstration selection methods. From the experimental results reported in Table IV, we have the following observations.

Evaluation on question batching. As reported in Table IV, the diversity-based question batching achieves the highest overall F1 scores. Moreover, it is interesting to see that the similarity-based question batching performs the worst on matching accuracy, even achieving lower F1 scores than the random question batching. This is because the questions within a batch is very similar, thus making the LLM difficult to differentiate entities by comparing different questions. Consequently, the LLM tends to produce identical answers for various questions, leading to degradation of matching accuracy. On the other hand, we can see that different question batching strategies have similar results on API cost and labeling cost, given varying demonstration selection methods. The reason is straightforward since prompts of different question batching strategies have similar amounts of tokens.

Evaluation on demonstration selection. Observing Table IV again, we can see that kNN-question and our covering-based strategy (denoted as Cover) outperform other strategies on accuracy, while the F1 scores of these two strategies are comparable. For example, under diversity-based batching, kNN-question yields the highest F1 score on 2 datasets, while Cover is the best on the remaining 6 datasets. This is because both kNN-question and Cover aim to select relevant demonstrations for each individual question within a batch, which is helpful for the LLM to understand varying cases of ER. However, kNN-question method suffers from the heavy labeling cost since the number of demonstrations selected by the kNN algorithm will increase proportionally as the number of predicted entity pairs grows.

By contrast, Cover is much more cost-effective than kNN-question on demonstration labeling, *e.g.,* brings 10x-100x labeling cost savings on the former five large datasets and 5x savings on the latter three small datasets. The results validate the effectiveness of our *covering-based* mechanism: by selecting a minimal set of demonstrations that cover all questions in a batch, we can significantly reduce the number of required demonstrations, and thus save the labeling cost.

Finding 2: The design choice that combines Diversity-based Question Batching and our Covering-based Demonstration Selection is the most favorable, *i.e.,* achieving the highest accuracy while incurring the lowest cost.

D. Comparing with PLM-based Approaches to ER

Exp-3: How does our BATCHER framework compare with PLM-based approaches to ER? We compare our framework with the PLM-based approaches mentioned in Section VI-A, by varying the size of training set for these approaches. Note that we use the best design choices shown in Table IV, *i.e.,* Diversity-based Question Batching and Covering-based Demonstration Selection, as the default setting.

Figure 7 shows the experimental results on the eight datasets, where the results of our framework are represented as red solid lines. Not surprisingly, our framework is *much more cost-effective* than Ditto [1], JointBert [2] and RobEM [3]. For example, on the WA, AB and AG datasets, the three PLM-based methods require at least 2000 training samples to achieve a similar F1 score of our framework. In contrast, our framework requires no more than 50 labeled samples on all the datasets. According to our cost calculation method in Section VI-A, the monetary cost incurred by these PLM-based approaches is about **300x-400x** larger than our overall cost (*i.e.,* API cost plus labeling cost). Furthermore, we also observe that once models like RobEM catching up with the F1 score of our framework, additional training samples do not substantially increase the performance; on some datasets (*e.g.,* FA, IA and Beer), even the entire training set is insufficient for the baselines to reach the F1 score of our framework.

Finding 3: With much less labeled data, our batch prompting framework achieves competitive performance with PLM-based method trained with hundreds of or even thousands of labeled matching/non-matching entity pairs.

E. Comparing with Manual Prompting for ER

Exp-4: How does our BATCHER framework compare with LLM-based approaches to ER? We compare our framework with the existing LLM-based approach [11], equipped with manually designed prompts, including hand-picked demonstrations. The results are reported in Table V. The reason for the absence of a comparison for the Abt-Buy dataset in the Table V is that ManualPrompt approach [11] is not tested on this dataset. We can see that, with only 20% of the API cost, our BATCHER framework can achieve comparable F1 score, compared with the ManualPrompt approach. In particular, on four datasets (DS, DA, FZ, Beer), our framework even outperforms ManualPrompt. The results implies that BATCHER, despite requiring cost of labeling selected demonstrations, may still be more practical than ManualPrompt, which requires domain experts for prompt designing.

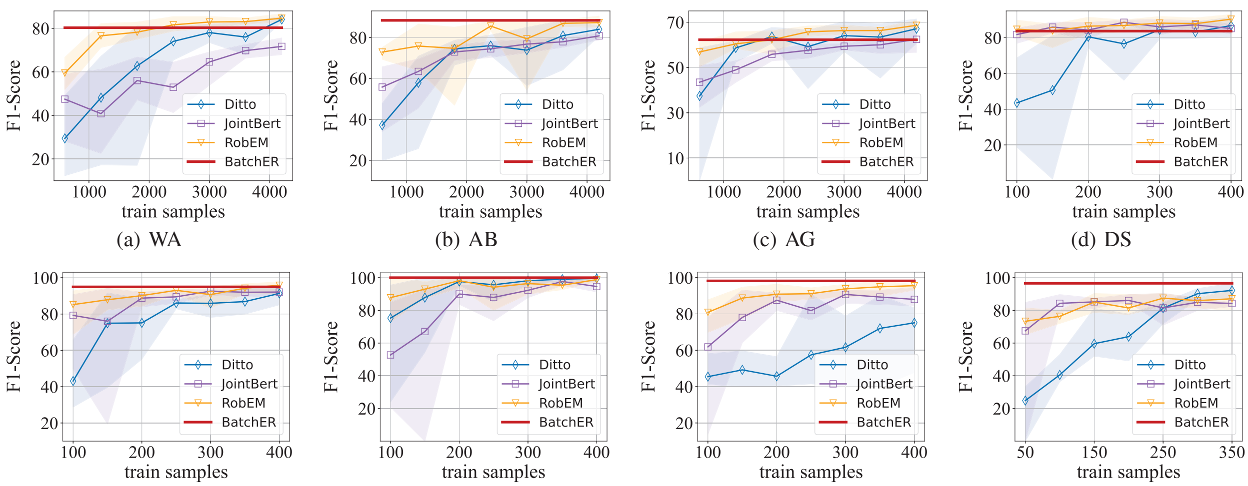
Finding 4: Our automatic batch prompting framework achieves comparable or even better F1 scores with manual prompting methods for LLMs, with much less API cost.

F. Evaluation on Different Underlying LLMs

Exp-5: What is performance of our approaches given various underlying LLMs? We evaluate the performance

TABLE IV: **Exploring the Design Space of Three Question Batching Methods and Four Demonstration Selection Methods (The best results are bolded and the second best results are underlined).**

Dataset	Metric	Random Question Batching				Similarity-based Question Batching				Diversity-based Question Batching			
		Fix	kNN-batch	kNN-question	Cover	Fix	kNN-batch	kNN-question	Cover	Fix	kNN-batch	kNN-question	Cover
WA	**F1**	78.92	79.15	79.06	78.64	73.50	77.43	78.30	76.43	79.24	78.87	80.18	**80.66**
	API ($)	0.33	0.34	0.35	0.30	0.34	0.34	0.35	**0.24**	0.35	0.34	0.34	0.28
	Label ($)	**0.06**	11.53	12.63	0.34	**0.06**	14.15	12.63	0.34	**0.06**	13.30	12.63	0.34
AB	**F1**	85.79	86.24	86.79	85.71	85.19	85.65	87.02	87.16	85.03	86.38	87.91	**88.38**
	API ($)	0.24	0.23	0.24	0.21	0.24	0.23	0.24	**0.20**	0.24	0.23	0.24	0.20
	Label ($)	**0.06**	10.86	6.07	0.28	**0.06**	10.86	6.07	0.28	**0.06**	11.21	6.07	0.28
AG	**F1**	61.07	61.82	61.90	60.69	58.90	60.74	60.96	60.62	60.24	57.85	**64.57**	62.16
	API ($)	0.29	0.30	0.30	0.25	0.30	0.30	0.30	**0.25**	0.29	0.30	0.30	0.25
	Label ($)	**0.06**	14.20	9.70	0.23	**0.06**	14.09	9.70	0.23	**0.06**	13.84	9.69	0.23
DS	**F1**	80.79	82.49	83.55	82.36	76.44	73.78	77.09	75.59	79.07	79.80	83.46	**83.70**
	API ($)	1.22	1.27	1.28	1.13	1.31	1.27	1.29	**1.04**	1.27	1.15	1.28	1.12
	Label ($)	**0.06**	35.38	27.94	0.31	**0.06**	35.92	28.24	0.31	**0.06**	35.96	28.24	0.31
DA	**F1**	92.10	93.00	93.62	92.32	91.59	92.42	92.44	92.06	92.27	94.21	94.28	**94.96**
	API ($)	0.63	0.62	0.63	0.54	0.62	0.62	0.63	**0.50**	0.62	0.62	0.63	0.53
	Label ($)	**0.06**	15.50	14.61	0.32	**0.06**	15.50	14.61	0.32	**0.06**	15.09	14.61	0.32
FZ	**F1**	94.13	93.33	95.24	93.33	95.24	90.48	93.02	92.68	93.02	88.37	95.24	**100.00**
	API ($)	0.04	0.04	0.03	0.03	0.04	0.04	0.04	**0.03**	0.04	0.04	0.04	0.03
	Label ($)	**0.06**	1.18	1.27	0.30	**0.06**	1.25	1.32	0.30	**0.06**	1.18	1.27	0.30
IA	**F1**	91.75	94.74	94.55	92.59	92.59	94.34	96.30	92.86	88.00	94.55	**98.17**	96.43
	API ($)	0.01	0.01	0.01	0.01	0.01	0.01	0.01	**0.01**	0.01	0.01	0.01	0.01
	Label ($)	**0.06**	0.60	0.56	0.16	**0.06**	0.69	0.56	0.16	**0.06**	0.42	0.56	0.16
Beer	**F1**	88.31	76.92	81.48	89.66	85.71	84.62	81.48	88.89	92.86	89.66	89.66	**96.55**
	API ($)	0.01	0.01	0.01	0.01	0.01	0.01	0.01	**0.01**	0.01	0.01	0.01	0.01
	Label ($)	**0.06**	0.65	0.66	0.14	**0.06**	0.68	0.66	0.14	**0.06**	0.64	0.62	0.14



(a) WA (b) AB (c) AG (d) DS

(e) DA (f) FZ (g) IA (h) Beer

Fig. 7: Comparing our Batch Prompting framework BATCHER with existing PLM-based approaches to ER.

of BATCHER on various underlying LLMs, including two versions of GPT-3.5 and GPT-4, which are mentioned in Section VI-A. Note that we also evaluate the well-known open-source LLM, LLAMA2 [41]. However, we find that LLAMA2 is not suitable for batch prompting: When prompted to answer multiple questions, LLAMA2 fails to produce any output in most cases. Thus, we omit the results of LLAMA2.

The experimental results are shown in Table VI. First, considering matching accuracy, GPT-4 achieves the best results on five datasets, demonstrating its superior capability on text comprehension and task solving. Moreover, we also find GPT-3.5-03 is comparable to GPT-4. Specifically, GPT-3.5-03 achieves the second highest F1 overall and the largest F1 difference from GPT-4 is less than 6.4%. Second, as per the latest pricing, the token pricing of GPT-4 is **10x** higher than GPT-3.5, leads to considerably high API costs. To summarize, the results show that GPT-3.5-03 achieves the best trade-off between matching accuracy and monetary cost, making it a more favorable choice for practical applications.

Finding 5: As the underlying LLM of BATCHER, GPT3.5-03 achieves the best trade-off between matching accuracy and monetary cost.

TABLE V: **Comparing BATCHER with Manual Prompting.**

Dataset	Metric	Manual Prompting	BATCHER
WA	F1	**82.63**	80.66
	API ($)	1.40	**0.28**
AG	F1	**65.40**	62.16
	API ($)	1.65	**0.25**
DS	F1	70.44	**83.70**
	API ($)	5.87	**1.12**
DA	F1	94.90	**94.96**
	API ($)	2.65	**0.53**
FZ	F1	97.67	**100**
	API ($)	0.14	**0.03**
IA	F1	**98.11**	96.43
	API ($)	0.05	**0.01**
Beer	F1	92.23	**96.55**
	API ($)	0.05	**0.01**

TABLE VI: **Evaluating Different Underlying LLMs on Matching Accuracy and API Cost.**

Dataset	Metric	GPT-3.5-03	GPT-3.5-06	GPT-4
WA	F1	80.66	80.32	**81.22**
	API ($)	**0.28**	**0.28**	2.81
AB	F1	**88.38**	69.08	85.22
	API ($)	**0.20**	**0.20**	2.02
AG	F1	62.16	52.40	**64.06**
	API ($)	**0.25**	**0.25**	2.52
DS	F1	83.70	65.94	**89.48**
	API ($)	**1.12**	**1.12**	11.24
DA	F1	94.96	91.29	**96.04**
	API ($)	**0.53**	**0.53**	5.27
FZ	F1	**100.00**	92.68	**100.00**
	API ($)	**0.03**	**0.03**	0.32
IA	F1	**96.43**	92.31	94.34
	API ($)	**0.01**	**0.01**	0.09
Beer	F1	**96.55**	92.31	96.30
	API ($)	**0.01**	**0.01**	0.11

TABLE VII: **Evaluating Feature Extractors on Accuracy.**

Dataset	Structure-aware		Semantics-based
	BATCHER-LR	BATCHER-JAC	BATCHER-SEM
WA	**80.66**	78.05	78.66
AB	**88.38**	84.23	87.06
AG	**62.16**	59.90	59.20
DS	**83.70**	81.27	80.91
DA	**94.96**	92.70	90.36
FZ	**100.00**	93.62	95.24
IA	**96.43**	90.57	90.91
Beer	**96.55**	89.66	91.67

G. *Evaluation on Different Feature Extractors*

Exp-6: What is performance of our approaches given different feature extractors? We examine the performance of BATCHER using different Feature Extractors described in Section III-B, namely BATCHER-LR, BATCHER-JAC, and BATCHER-SEM. The former two feature extractor use Structure-aware Feature Extractor based on Levenshtein Ratio (LR) and Jaccard Similarity (JAC). The latter uses Semantics-based Feature Extractor based on SBERT [26] embedding. We choose SBERT as the semantics-based feature extractor over BERT and RoBERTa because it more effectively generates embeddings that capture the semantic relevance among sentences in an unsupervised manner [26]. Since their monetary cost is close, we only compare these three variants on F1 scores.

As shown in Table VII, BATCHER-LR achieves the best performance on all the datasets while BATCHER-JAC and BATCHER-SEM achieve comparative results. This results validates that stucture-aware feature extractor can better capture the relevance between entity pairs in the ER scenario. Moreover, compared with BATCHER-JAC, BATCHER-LR is more sensitivity to string order and its superior precision in quantifying the similarity between two strings. For instance, considering two strings "listen" and "silent", the similarity score calculated using LR is 0.5, whereas with JAC, it is 0.89. This clearly demonstrates the former is better effectiveness in quantifying the similarity between the two strings, thus is more effective to generate feature vectors for entity pairs.

Finding 6: The structure-aware feature extractor is preferred for measuring distances among entity pairs in ER.

TABLE VIII: **Evaluating Different Demonstration Labelers on Matching Accuracy and Overall Cost.**

Dataset	Metric	GPT-3.5 Labeler	GPT-4 Labeler	Human Labeler
WA	F1	75.04	75.47	**80.66**
	Cost ($)	**0.29**	0.34	0.62
AB	F1	85.34	85.43	**88.38**
	Cost ($)	**0.21**	0.24	0.48
AG	F1	58.32	58.81	**62.16**
	Cost ($)	**0.26**	0.28	0.48
DS	F1	80.83	81.17	**83.70**
	Cost ($)	**1.13**	1.19	1.43
DA	F1	93.90	93.96	**94.96**
	Cost ($)	**0.53**	0.60	0.85
FZ	F1	**100**	**100**	**100**
	Cost ($)	**0.04**	0.08	0.33
IA	F1	90.21	92.59	**96.43**
	Cost ($)	**0.01**	0.03	0.17
Beer	F1	88.34	**96.55**	**96.55**
	Cost ($)	**0.01**	0.03	0.15

H. *Evaluation on Different Demonstration Labelers*

Exp-7: Is it feasible to substitute the use of crowd workers with API requests to LLMs as a cost-effective alternative? We evaluate BATCHER's performance utilizing different labeling approaches. Specifically, instead of soliciting crowd workers for labeling the selected demonstrations, we respectively utilize GPT-3.5 and GPT-4 as demonstration labelers. After that, we use the labeled demonstrations to guide our default LLM (GPT-3.5) to make predictions for batched questions. The experimental results are reported in Table VIII.

First, using LLMs as demonstration labelers can reduce the overall cost. Take GPT-4 Labeler as an example: compared with Human Labeler, GPT-4 Labeler achieves 52.44% cost reductions on average on our eight datasets. Second, the matching accuracy of using LLMs as demonstration labelers is still worse than that of using human labelers. Considering GPT-4 Labeler again, we can find that the F1 score obtained by GPT-4 Labeler is 2.90% lower than that of Human Labler on average. This is mainly because that GPT-based demonstration Labelers may generate demonstrations with incorrect labels, which mislead the LLMs in answering questions in corresponding batches. Thus, it is worthwhile to further explore this aspect to enhance the overall cost-effectiveness.

Finding 7: Employing LLMs as demonstration labelers may potentially improve cost-effectiveness of BATCHER.

VII. Related Work

PLM-based Methods for Entity Resolution. Entity resolution is a popular data integration task that has been widely studied for decades. With the rise of deep learning, some approaches [42] leverage pre-trained word embeddings to improve the ER performance. However, these methods mainly use the non-contextual embeddings without considering the downstream tasks. Therefore, recent studies [1], [2], [4], [5] have focused on using Transformer-based PLMs to produce contextualized embeddings based on fine-tuning over downstream tasks. To be specific, Ditto [1] regards ER as a sequence-pair classification problem via Transformer, where domain knowledge is injected to further improve the performance. DADER [5] focuses on leveraging the domain adaptation technique: given a labeled source dataset, it trains an ER model for another target dataset by aligning features of both datasets based on PLMs. Based on PLMs, Unicorn [4] focuses on building a unified framework for data matching tasks, including ER. Unicorn uses a unified encoder for any pair of data to be predicted, and a mixture-of-experts module to align the semantics of multiple tasks. Although the above PLMs-based approaches can achieve a relatively good performance, they need plenty of labeled pairs for supervision, which are often expensive to acquire.

LLM-based Methods for Entity Resolution. With the size of pre-training data and model parameters scales, large-scale language models (LLMs) have gained an emergent capability called In-Context Learning (ICL) to learn from a few demonstrations without explicit model update [6], [43]. Recent studies [11], [12], [31] have focused on utilizing the LLMs to tackle ER with less labeled pairs for supervision. Narayan et al. [11] are among the first to explore the capability of GPT-3 [6] for ER with manually designed demonstrations, which achieves remarkable performance compared with PLM-based methods. Since manual demonstrations require professional prompting engineering knowledge, Peeters et al. [12] propose to select relevant demonstrations based on kNN retrieval algorithm, where Jaccard similarity is utilized to measure the relevance. Moreover, Zhang et al. [31] consider batch prompting for ER, which employs a straightforward random batching strategy with manually designed demonstrations. Although question batching and demonstration selection have been considered in existing studies, these studies mainly rely on domain experts or develop heuristics for these two problem, and have not explored the combination of different demonstration selection and batching strategies. Compared to them, we utilize the power of ICL and propose a comprehensive framework BATCHER. We explore a design space to evaluate the performance of different design choices, and propose a covering-based demonstration selection strategy that effectively balances the trade-off between accuracy and cost.

In-Context Learning for Data Management. LLMs are capable to capture rich linguistic patterns and generate coherent text [6], [40], [44], which have shown great success in a wide range of NLP tasks [17], [18], [24]. ICL is an emergent capability of LLMs that enables the model to learn from few demonstrations without explicit gradient update [43]. Recently, researchers have studied to leverage ICL to solve data management tasks, such as data discovery [45], data cleaning and integration [11], and data labeling [46], and also study how to batch questions and select demonstrations. BatchPrompt [25] proposes to group multiple questions into one batch and query LLMs to answer one batch in an interface. In addition, both relevance-based [34], [47] and diversity-based [48], [49] strategies are proposed for demonstration selection. Compared with these studies, as far as we know, we are the first to develop the batch prompting technique tailored to the ER task, and design new methods, such as covering-based demonstration selection and structure-aware feature extraction, which are shown to be effective for ER.

VIII. Conclusion and Future Works

In this paper we have introduced a cost-effective batch prompting framework BATCHER for entity resolution, and explored the effectiveness of BATCHER under different design choices. We also devised a covering-based demonstration selection strategy that achieves effective balance between accuracy and cost. We have conducted extensive experiments to evaluate different combinations of design choices with insightful empirical findings, as summarized using the six findings in Section VI. These findings imply that BATCHER is very cost-effective for ER, compared with not only PLM-based methods fine-tuned with extensive labeled data, but also LLM-based methods with manually designed prompting.

For future directions, it is desirable to study how to mitigate generating erroneous labels from LLMs by using sophisticated mechanisms, *e.g.,* multi-round voting [50] and self-correction [51], [52]. Moreover, integrating the capabilities of both LLMs and crowd workers offers a promising avenue for improving overall accuracy while controlling the cost.

Acknowledgment

This paper was partly supported by the NSF of China (62122090, 62072461, 61925205, 62232009 and 62102215), National Key Research and Development Program of China (2023YFB4503600), the Beijing Natural Science Foundation (L222006), the Research Funds of Renmin University of China, CCF-Huawei Populus Grove Fund (CCF-HuaweiDB202306), and Alibaba Group through Alibaba Innovative Research Program.

REFERENCES

[1] Y. Li, J. Li, Y. Suhara, A. Doan, and W. Tan, "Deep entity matching with pre-trained language models," *Proc. VLDB Endow.*, vol. 14, no. 1, pp. 50–60, 2020.

[2] R. Peeters and C. Bizer, "Dual-objective fine-tuning of bert for entity matching," *Proc. VLDB Endow.*, vol. 14, no. 10, p. 1913–1921, 2021.

[3] M. Akbarian Rastaghi, E. Kamalloo, and D. Rafiei, "Probing the robustness of pre-trained language models for entity matching," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022, p. 3786–3790.

[4] J. Tu, J. Fan, N. Tang, P. Wang, G. Li, X. Du, X. Jia, and S. Gao, "Unicorn: A unified multi-tasking model for supporting matching tasks in data integration," *Proceedings of the ACM on Management of Data*, vol. 1, no. 1, pp. 1–26, 2023.

[5] J. Tu, J. Fan, N. Tang, P. Wang, C. Chai, G. Li, R. Fan, and X. Du, "Domain adaptation for deep entity resolution," in *Proceedings of the 2022 International Conference on Management of Data*, 2022, pp. 443–457.

[6] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *NeurIPS 2020*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.

[7] S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer, "Rethinking the role of demonstrations: What makes in-context learning work?" *arXiv preprint arXiv:2202.12837*, 2022.

[8] J. Chen, L. Chen, and T. Zhou, "It takes one to tango but more make trouble? in-context training with different number of demonstrations," *arXiv preprint arXiv:2303.08119*, 2023.

[9] L. Gao, A. Chaudhary, K. Srinivasan, K. Hashimoto, K. Raman, and M. Bendersky, "Ambiguity-aware in-context learning with large language models," *arXiv preprint arXiv:2309.07900*, 2023.

[10] X. Wang, Y. Wang, C. Xu, X. Geng, B. Zhang, C. Tao, F. Rudzicz, R. E. Mercer, and D. Jiang, "Investigating the learning behaviour of in-context learning: A comparison with supervised learning," *arXiv preprint arXiv:2307.15411*, 2023.

[11] A. Narayan, I. Chami, L. J. Orr, and C. Ré, "Can foundation models wrangle your data?" *Proc. VLDB Endow.*, vol. 16, no. 4, pp. 738–746, 2022. [Online]. Available: https://www.vldb.org/pvldb/vol16/p738-narayan.pdf

[12] R. Peeters and C. Bizer, "Entity matching using large language models," *CoRR*, vol. abs/2310.11244, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2310.11244

[13] A. Doan, P. Konda, P. S. G. C., Y. Govind, D. Paulsen, K. Chandrasekhar, P. Martinkus, and M. Christie, "Magellan: toward building ecosystems of entity matching solutions," *Commun. ACM*, vol. 63, no. 8, pp. 83–91, 2020. [Online]. Available: https://doi.org/10.1145/3405476

[14] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra, "Deep learning for entity matching: A design space exploration," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 19–34.

[15] O. Rubin, J. Herzig, and J. Berant, "Learning to retrieve prompts for in-context learning," in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*. Association for Computational Linguistics, 2022, pp. 2655–2671. [Online]. Available: https://doi.org/10.18653/v1/2022.naacl-main.191

[16] Y. Zhang, K. Zhou, and Z. Liu, "What makes good examples for visual in-context learning?" *arXiv preprint arXiv:2301.13670*, 2023.

[17] X. Li, K. Lv, H. Yan, T. Lin, W. Zhu, Y. Ni, G. Xie, X. Wang, and X. Qiu, "Unified demonstration retriever for in-context learning," *arXiv preprint arXiv:2305.04320*, 2023.

[18] S. Agrawal, C. Zhou, M. Lewis, L. Zettlemoyer, and M. Ghazvininejad, "In-context examples selection for machine translation," in *ACL*, A. Rogers, J. L. Boyd-Graber, and N. Okazaki, Eds. Association for Computational Linguistics, 2023, pp. 8857–8873. [Online]. Available: https://doi.org/10.18653/v1/2023.findings-acl.564

[19] G. Papadakis, D. Skoutas, E. Thanos, and T. Palpanas, "Blocking and filtering techniques for entity resolution: A survey," *ACM Computing Surveys (CSUR)*, vol. 53, no. 2, pp. 1–42, 2020.

[20] S. Thirumuruganathan, H. Li, N. Tang, M. Ouzzani, Y. Govind, D. Paulsen, G. Fung, and A. Doan, "Deep learning for blocking in entity matching: a design space exploration," *Proceedings of the VLDB Endowment*, vol. 14, no. 11, pp. 2459–2472, 2021.

[21] C. Ge, P. Wang, L. Chen, X. Liu, B. Zheng, and Y. Gao, "Collaborem: a self-supervised entity matching framework using multi-features collaboration," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[22] Y. Lu, M. Bartolo, A. Moore, S. Riedel, and P. Stenetorp, "Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity," *arXiv preprint arXiv:2104.08786*, 2021.

[23] Y. Chen, C. Zhao, Z. Yu, K. McKeown, and H. He, "On the relation between sensitivity and accuracy in in-context learning," *arXiv preprint arXiv:2209.07661*, 2022.

[24] Z. Wan, F. Cheng, Z. Mao, Q. Liu, H. Song, J. Li, and S. Kurohashi, "Gpt-re: In-context learning for relation extraction using large language models," *arXiv preprint arXiv:2305.02105*, 2023.

[25] Z. Cheng, J. Kasai, and T. Yu, "Batch prompting: Efficient inference with large language model apis," *arXiv preprint arXiv:2301.08721*, 2023.

[26] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[27] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[28] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[29] M. Luo, X. Xu, Z. Dai, P. Pasupat, M. Kazemi, C. Baral, V. Imbrasaite, and V. Y. Zhao, "Dr. icl: Demonstration-retrieved in-context learning," *arXiv preprint arXiv:2305.14128*, 2023.

[30] K. Margatina, T. Schick, N. Aletras, and J. Dwivedi-Yu, "Active learning principles for in-context learning with large language models," *arXiv preprint arXiv:2305.14264*, 2023.

[31] H. Zhang, Y. Dong, C. Xiao, and M. Oyamada, "Large language models as data preprocessors," *arXiv preprint arXiv:2308.16361*, 2023.

[32] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *KDD*, E. Simoudis, J. Han, and U. M. Fayyad, Eds., 1996, pp. 226–231.

[33] V. I. Levenshtein *et al.*, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8. Soviet Union, 1966, pp. 707–710.

[34] J. Liu, D. Shen, Y. Zhang, B. Dolan, L. Carin, and W. Chen, "What makes good in-context examples for gpt-3?" in *Proceedings of Deep Learning Inside Out: The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures, DeeLIO@ACL 2022, Dublin, Ireland and Online, May 27, 2022*. Association for Computational Linguistics, 2022, pp. 100–114. [Online]. Available: https://doi.org/10.18653/v1/2022.deelio-1.10

[35] K. Bernhard and J. Vygen, "Combinatorial optimization: Theory and algorithms," *Springer, Third Edition, 2005.*, 2008.

[36] P. Slavík, "A tight analysis of the greedy algorithm for set cover," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 435–441.

[37] J. Wang, T. Kraska, M. J. Franklin, and J. Feng, "Crowder: Crowdsourcing entity resolution," *Proc. VLDB Endow.*, vol. 5, no. 11, pp. 1483–1494, 2012. [Online]. Available: http://vldb.org/pvldb/vol5/p1483_jiannanwang_vldb2012.pdf

[38] (2021) Code of jointbert. [Online]. Available: https://github.com/wbsg-uni-mannheim/jointbert

[39] (2022) Code of robem. [Online]. Available: https://github.com/makbn/robem

[40] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[41] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. Canton-Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M. Lachaux, T. Lavril, J. Lee,

D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," *CoRR*, vol. abs/2307.09288, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2307.09288

[42] M. Ebraheem, S. Thirumuruganathan, S. R. Joty, M. Ouzzani, and N. Tang, "Distributed representations of tuples for entity resolution," *Proc. VLDB Endow.*, vol. 11, no. 11, pp. 1454–1467, 2018. [Online]. Available: http://www.vldb.org/pvldb/vol11/p1454-ebraheem.pdf

[43] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022.

[44] M. Agrawal, S. Hegselmann, H. Lang, Y. Kim, and D. Sontag, "Large language models are few-shot clinical information extractors," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022, pp. 1998–2022.

[45] M. Kayali, A. Lykov, I. Fountalis, N. Vasiloglou, D. Olteanu, and D. Suciu, "CHORUS: foundation models for unified data discovery and exploration," *CoRR*, vol. abs/2306.09610, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2306.09610

[46] N. Guan, K. Chen, and N. Koudas, "Can large language models design accurate label functions?" *CoRR*, vol. abs/2311.00739, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2311.00739

[47] Y. Lee, C. Lim, and H. Choi, "Does GPT-3 generate empathetic dialogues? A novel in-context example selection method and automatic evaluation metric for empathetic dialogue generation," in *Proceedings of the 29th International Conference on Computational Linguistics, COLING 2022, Gyeongju, Republic of Korea, October 12-17, 2022*. International Committee on Computational Linguistics, 2022, pp. 669–683. [Online]. Available: https://aclanthology.org/2022.coling-1.56

[48] I. Levy, B. Bogin, and J. Berant, "Diverse demonstrations improve in-context compositional generalization," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*. Association for Computational Linguistics, 2023, pp. 1401–1422. [Online]. Available: https://doi.org/10.18653/v1/2023.acl-long.78

[49] H. Su, J. Kasai, C. H. Wu, W. Shi, T. Wang, J. Xin, R. Zhang, M. Ostendorf, L. Zettlemoyer, N. A. Smith, and T. Yu, "Selective annotation makes language models better few-shot learners," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: https://openreview.net/pdf?id=qY1hlv7gwg

[50] J. Lin, M. Diesendruck, L. Du, and R. Abraham, "Batchprompt: Accomplish more with less," *arXiv preprint arXiv:2309.00384*, 2023.

[51] M. Pourreza and D. Rafiei, "Din-sql: Decomposed in-context learning of text-to-sql with self-correction," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[52] L. Pan, M. Saxon, W. Xu, D. Nathani, X. Wang, and W. Y. Wang, "Automatically correcting large language models: Surveying the landscape of diverse self-correction strategies," *arXiv preprint arXiv:2308.03188*, 2023.