Optimized Batch Prompting for Cost-effective LLMs

Zhaoxuan Ji
Beijing Institute of Technology
jizhaoxuan@bit.edu.cn

Xinlu Wang
Beijing Institute of Technology
xinlu_wang@bit.edu.cn

Zhaojing Luo
Beijing Institute of Technology
zjluo@bit.edu.cn

Zhongle Xie
Zhejiang University
xiezl@zju.edu.cn

Meihui Zhang
Beijing Institute of Technology
meihui_zhang@bit.edu.cn

ABSTRACT

Large Language Models (LLMs) have recently achieved record-breaking performance in various real-world applications with in-context learning (ICL) that needs a task description and several demonstrations. Most LLMs are charged based on the number of input tokens. In real-world LLM platforms (e.g., OpenAI), a batch of relevant questions may arrive simultaneously. Therefore, it is costly to perform inference on them individually as some of them are with the same demonstrations and redundant task descriptions. In this paper, we investigate the idea of batch prompting, which is to group questions for LLMs to perform inference in batch. Nevertheless, simple random grouping may result in significant performance degradation. In order to minimize LLM costs without compromising the performance, we explore factors that may affect the performance and formalize the batch prompting problem for the first time. Then, we study the hardness of this problem and propose a series of efficient exact and approximation algorithms. Finally, we conduct comprehensive experiments to evaluate our method on 14 datasets. Extensive experimental results show that our proposed solution outperforms the state-of-the-art baselines while consuming fewer costs.

PVLDB Reference Format:
Zhaoxuan Ji, Xinlu Wang, Zhaojing Luo, Zhongle Xie, and Meihui Zhang. Optimized Batch Prompting for Cost-effective LLMs. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/jzx-bitdb/BatchPrompt.

1 INTRODUCTION

Large language models (LLMs) have demonstrated considerable effectiveness across various real-world applications, such as question answering, machine translation, context summarization, etc [53]. In particular, recent works [2, 19, 20, 38, 52] delve into applying LLMs to data management tasks to improve the overall performance, such as entity resolution, data cleaning, etc. These works typically exploit in-context learning (ICL) techniques, which include the task description, several demonstrations (i.e., some examples with answers from the same task), and the question. This specific form is shown in the top left of Figure 1, where we present two examples of the entity resolution task.

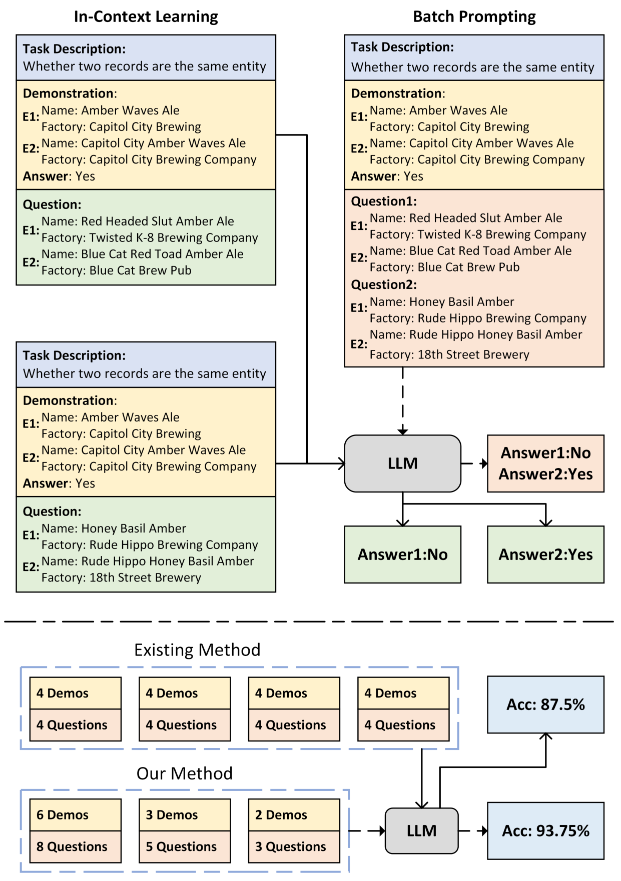

In-Context Learning
Batch Prompting
Task Description:
Whether two records are the same entity
Demonstration:
E1: Name: Amber Waves Ale
Factory: Capitol City Brewing
E2: Name: Capitol City Amber Waves Ale
Factory: Capitol City Brewing Company
Answer: Yes
Question:
E1: Name: Red Headed Slut Amber Ale
Factory: Twisted K-8 Brewing Company
E2: Name: Blue Cat Red Toad Amber Ale
Factory: Blue Cat Brew Pub
Question1:
Question2:
E1: Name: Honey Basil Amber
Factory: Rude Hippo Brewing Company
E2: Name: Rude Hippo Honey Basil Amber
Factory: 18th Street Brewery
LLM
Answer1:No
Answer2:Yes
Existing Method
4 Demos
4 Questions
Acc: 87.5%
Our Method
6 Demos
8 Questions
3 Demos
5 Questions
2 Demos
3 Questions
Acc: 93.75%

Figure 1: The top part of the figure shows the specific form of in-context learning and batch prompting, and the bottom part displays an example about batch prompting.

Most closed-source LLMs are not free, which are charged based on the number of input tokens when calling the provided APIs. Take the gpt-4o model of OpenAI [40] as an example, the price is \$5 per 1M tokens. In real-world LLM platforms (e.g., OpenAI), a batch of questions may arrive simultaneously. They are typically processed sequentially, which is overwhelmingly expensive. For instance, some users leverage LLMs to do entity resolution [26].

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

Given a table with n items, they need $O(n^2)$ questions for LLMs to reason [17]. However, most questions in a batch may belong to the same task and thus be related. Consequently, there is room for improvement to reduce LLM costs by combining several questions into a single prompt.

Batch prompting is an effective way for this scenario. To be specific, we leverage LLMs to process a batch of questions simultaneously for a certain task, and there may be overlapping demonstrations in their respective prompts. To reduce the cost, several questions can be combined into a single prompt, namely a "batch" prompt, for LLMs to perform inference, in order to eliminate redundant demonstrations. The specific form is shown in the top right of Figure 1, where the cost of one demonstration is reduced after grouping the questions. Batch prompting has been preliminarily explored in two recent works [9, 17]. For example, [9] proposes to randomly combine questions into groups and uses fixed demonstrations for all groups. [17] targets for the entity resolution task and proposes a batch prompting strategy. In particular, they cluster all questions and then select one question from each cluster to generate one group. The size of all groups is fixed, which is 8 by default.

While these methods have validated the effectiveness of batch prompting in reducing LLM costs, their studies are too preliminary to form a general guideline for the problem of batch prompting. Specifically, the following challenges are insufficiently studied in effective batch prompting: **(1) Key performance factors**. Regarding batch prompting, there are various factors that interfere with each other and can greatly affect the performance. While some methods, including selecting similar demonstrations and grouping diverse questions, have been examined in recent works [9, 17], more factors need to be defined formally and investigated deeply. For example, varying input lengths can greatly affect the performance of LLMs [25, 30, 34, 36]. Thus, if a large number of questions are input into LLMs all at once, the performance may deteriorate. **(2) Adaptive and effective grouping.** Existing works either assign fixed demonstrations to each group, or target one specific application scenario. However, these methods may not be effective and adaptive for different kinds of tasks. For example in the bottom part of Figure 1, existing methods divide 16 questions into 4 groups, with each group containing 4 questions and 4 examples. However, this does not achieve optimal performance and minimal cost (our method only needs 11 demonstrations while the existing method requires 16 demonstrations). Thus, how to develop a solution that is general and can dynamically adapt to various tasks is another challenge.

In this paper, we first define and study four factors that may affect the performance of LLMs, namely: (1) **Question Group Affinity**: the correspondence between questions, which guides the question grouping; (2) **Question Demonstration Affinity**: the correspondence between the question and the demonstration, which relates to the selection of the demonstrations for each group; (3) **Group Cost**: the token count of each group; (4) **Demonstration Coverage**: the number of demonstrations for each group cannot be significantly small, which may degrade the LLM performance. Thus, this factor limits the number of demonstrations for each group.

Second, we formalize the batch prompting problem for the first time, incorporating the aforementioned key performance factors. Intuitively, the objective is to group a batch of questions and corresponding demonstrations with minimal cost, while satisfying constraints about the factors. Furthermore, we present a rigorous theoretical analysis of the batch prompting problem and show that the overall problem is NP-hard.

Third, we propose a framework called Optimized Batch Prompting (OBP), which includes an exact method based on Mixed Integer Linear Programming (MILP) and an approximation method (*QueClt+CltSplt*). We show that the MILP-based solution is prohibitively expensive while the approximation algorithm is computationally efficient. Specifically, we use correlation clustering to split the whole questions into clusters (*QueClt*). For each cluster, we propose a three-staged method to split the clusters (*CltSplt*): (1) Selecting demonstrations with minimal cost by reduction to the set cover problem; (2) Remove the overlapping questions in the sets (one set indicates a demonstration with its similar questions); (3) Combining these sets into a minimal number of groups. Besides the algorithms, we further propose two additional optimizations to accelerate the computations.

In summary, our main contributions are as follows:

- We investigate the optimization opportunities in batch prompting of LLMs, and propose four key factors that affect the performance and formalize the batch prompting problem.
- We propose a comprehensive theoretical analysis of batch prompting and design an exact solution based on Mixed Integer Linear Programming.
- To make our solution scalable to a larger number of questions, we propose an efficient approximation method, which first clusters the questions, and then splits each cluster to maintain the proposed constraints. Additionally, we propose several optimizations including reducing the number of distance computations and filtering unnecessary demonstrations.
- We conduct extensive experiments on three tasks with 14 datasets to demonstrate the effectiveness and efficiency of our proposed method. Results show that our method reduces the costs by up to 35% compared to the state-of-the-art baselines. Meanwhile, our method outperforms the baselines upon almost all datasets in terms of accuracy.

The rest of the paper is organized as follows. Section 2 presents the factors and problem setting about the batch prompting. We propose our OBP framework in Section 3. We evaluate the OBP framework in Section 4. Finally, we discuss the related work in Section 5 and conclude the paper in Section 6.

2 PROBLEM STATEMENT

In this section, we first introduce the factors that impact the performance, and then, we formalize the batch prompting problem.

2.1 Key Performance Factors

For a batch of questions $Q = \{q_1, q_2, ..., q_N\}$, we have a pool of demonstrations $\mathcal{D} = \{d_1, d_2, ..., d_M\}$. The goal is to combine these questions into groups, which are denoted as $\mathcal{G}$. Note that $\mathcal{G}$ covers all questions, and each question exactly belongs to one group. We assume that the optimal number of final groups is K, i.e., $\mathcal{G} =$

Table 1: Notations used in this paper.

Notations	Definition
N	The number of questions in one batch.
M	The number of total demonstrations.
K	The number of optimal groups for a batch of N questions, which is unknown in advance.
c_l	The cost of the group $\mathcal{G}_l$.
c_T	The cost of the task description.
c_{q_i}	The cost of the question q_i.
c_{d_j}	The cost of the demonstration d_j.
τ_0	The question group affinity threshold for each group.
τ_1	The question demonstration affinity threshold.
τ_2	The cost threshold of each group.
τ_3	The coverage threshold of each demonstration.
QGA	Question group affinity
QPA	Question pairwise affinity
QDA	Question demonstration affinity
GC	Group cost
DC	Demonstration coverage

$\{\mathcal{G}_1, \mathcal{G}_2, ...\mathcal{G}_l, ..., \mathcal{G}_K\}$. Each group $\mathcal{G}_l$ consists of the task description, a subset of questions from Q and a subset of demonstrations from $\mathcal{D}$. Thus, $\mathcal{G}_l$ can be written as:

$$\mathcal{G}_l = \{\mathcal{T}, \mathcal{D}_i, Q_i\}, where$$
$$\mathcal{T}: Task\ Description,\ \mathcal{D}_i = \{d_1, d_2, d_3...\},\ Q_i = \{q_1, q_2, q_3...\} \quad (1)$$

Since LLMs charge based on the number of input tokens, we thus measure the cost by token counts. Each question and demonstration has its own costs due to their different tokens. The cost of each group includes the costs of the task description and the costs of questions and demonstrations in the group. The corresponding notations are shown in Table 1. Note that for all groups within a particular task, the task description remains the same.

To explore the factors that impact the performance, we select several questions from three tasks, namely entity resolution, data transformation and code assertion generation. They are representative tasks in data management, and the corresponding dataset details are shown in Section 4. We then conduct a series of evaluations which are shown in Figure 2. Note that *GC* and *DC* in the figure refer to the group cost and demonstration coverage respectively, which will be elaborated upon subsequently. Based on the evaluation results, we propose the following four factors.

(1) Question Group Affinity (QGA). How to group questions is significant for batch prompting. Although related work acknowledges the importance of questions' affinity to enable effectiveness [17], there is no attempt to formalize them. As shown in Figure 2 , we find that grouping similar questions into one group outperforms random grouping for all three tasks. Thus, we define the *QGA* to guide question grouping, which is as follows:

$$QGA(\mathcal{G}_l) = \max_{\forall q_i, q_i' \in \mathcal{G}_l} QPA(q_i, q_i') \quad (2)$$

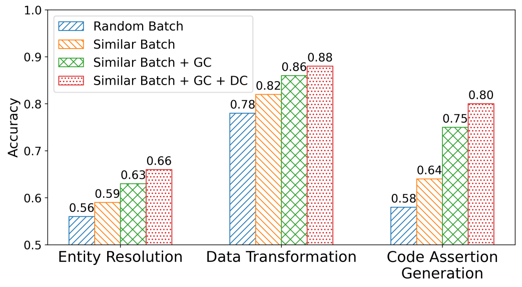


Figure 2: The performance comparison considering different factors on three tasks.

where $QPA(q_i, q_i')$ indicates the affinity (e.g., similarity) between two questions q_i and q_i', and we assume smaller QPA ensures better performance for LLMs.

.

(2) Question Demonstration Affinity (QDA). As recent works [21, 22, 39] show, LLMs produce better performance if selected demonstrations are similar to the question. In the batch prompting problem, we also observe the same trend, i.e., selecting similar demonstrations helps improve the performance. We thus define the affinity (i.e., the similarity) between the question q_i and demonstration d_j as $QDA(q_i, d_j)$, which can guide how to select demonstrations for each group. Note that how to measure the affinity between questions and demonstrations (i.e., computing *QGA* and *QDA*) is orthogonal to our method. Users can employ BERT-based methods [14] or other advanced techniques to covert them into embeddings and then calculate the affinity based on Euclidean distance, Cosine similarity, etc.

(3) Group Cost (GC). The cost of each group cannot be significantly large for two reasons. First, LLMs have the input token limit, e.g., the input token limit for gpt-4o is 128,000 tokens [40]. The second and most important reason is that a large input context for LLMs tends to result in poor performance [25, 30, 34, 36]. As shown in Figure 2, we observe that limiting the input length can improve the performance of batch prompting. Thus, we define the group cost, i.e., *GC*, to limit the token counts of one group (the input length of LLMs).

(4) Demonstration Coverage (DC). For a certain demonstration, it can cover many questions (i.e., the corresponding *QDA* is less than a pre-defined threshold). This may lead to a situation where there are a large number of questions in some groups, with only a few demonstrations. However, such groups would result in bad performance when inputted into LLMs. As shown in Figure 2, when we restrict the maximum number of questions covered by each demonstration (e.g., each demonstration can cover at most 3 questions), the total accuracy increases. For example, the accuracy of the code assertion generation task is from 0.75 ("Similar Batch + GC") to 0.8 ("Similar Batch + GC + DC"). Motivated by this observation, we define the demonstration coverage of a specific d_j, i.e., $DC(d_j)$, as the maximum number of questions it can cover.

2.2 Constraints and Objective

To ensure the performance of LLMs for the batch prompting setting, the following constraints should be considered:

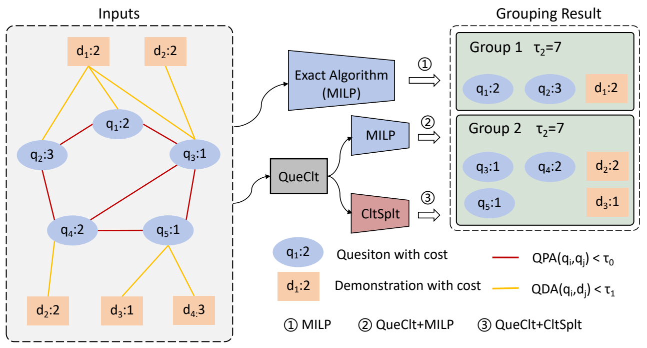


Figure 3: The OBP framework for batch prompting.

- *QGA Constraint*: For each group $\mathcal{G}_l \in \mathcal{G}$, the $QGA(\mathcal{G}_l)$ has to be at least as low as the pre-defined threshold τ_0.
- *QDA Constraint*: For each question, there should be at least one similar demonstration. i.e., $\exists d_j \in \mathcal{G}_l, QDA(q_i, d_j) \leq \tau_1, \forall q_i \in \mathcal{G}_l$.
- *GC Constraint*: For each group $\mathcal{G}_l \in \mathcal{G}$, the total cost of the group (c_l) cannot exceed the pre-defined maximum cost, i.e., $c_l \leq \tau_2$.
- *DC Constraint*: For each demonstration d_j, the number of questions it covers cannot exceed τ_3, i.e., $DC(d_j) \leq \tau_3$.

To sum up, given a batch $\mathcal{Q}$ of questions and a pool $\mathcal{D}$ of demonstrations, the objective is to group all questions in $\mathcal{Q}$ and corresponding demonstrations into K groups $\mathcal{G}$, such that the overall cost of all groups is minimized, with the constraints of LLM performance satisfied, which is formulated as:

$$\begin{aligned}
&\min \sum_{l}^{K} c_l \\
&s.t. QGA(\mathcal{G}_l) \leq \tau_0, \forall l \in [1, K] \\
&\quad QDA(q_i, d_j) \leq \tau_1, \forall q_i \in \mathcal{G}_l, \exists d_j \in \mathcal{G}_l \\
&\quad c_l \leq \tau_2, \forall l \in [1, K] \\
&\quad DC(d_j) \leq \tau_3, \forall d_j \in \mathcal{G}_l
\end{aligned} \tag{3}$$

3 THE OBP FRAMEWORK

In this section, we shall present our framework Optimized Batch Prompting (OBP) as shown in Figure 3, which includes three methods, namely *MILP*, *QueClt+MILP* and *QueClt+CltSplt*. The input of our framework is a batch of questions $\mathcal{Q}$ and a demonstration pool $\mathcal{D}$. (1) First, we transform the optimization problem in Equation 3, so that it can be solved by MILP solver (e.g., GUROBI [29]). We then propose several optimizations to accelerate the computations of the MILP solver. The details are explained in Section 3.2. (2) Due to the expensive computation of the MILP solver, it cannot scale to a larger number of questions. Consequently, we propose to use correlation clustering to split original questions, so that each cluster can be solved by the MILP solver individually, which is faster than the original format (Section 3.3). (3) To further accelerate the computation, we design an approximation algorithm to replace the MILP solver, which will be explained in Section 3.4. In the following, we first give a theoretical analysis of the batch prompting problem.

3.1 Theoretical Analysis

THEOREM 1. *The problem of batch prompting shown in Equation 3 is NP-hard.*

PROOF. Given a batch of questions $\mathcal{Q}$, the decision version of the problem is, whether there is a set of groups $\mathcal{G}$ that can cover all questions with all constraints in Equation 3 satisfied. Specifically, we can assign one question as a group, and then select a demonstration for each group whose *QDA* is less than τ_1, so the membership verification of the decision version of the problem is polynomial.

To prove NP-hardness, we create an instance of this problem by a reduction from a variant of set cover [6] problem, i.e., weighted set cover [11]. A weighted set cover problem (*WSC*) can be described as follows: Given a universe set $U = \{a_1, a_2, ..., a_n\}$ with n items, and m subsets $S = \{s_1, s_2, ..., s_m\}, \forall s_j \subset U$. Each subset s_j has a weight w_j. The *WSC* is to find a set of S whose total weight is minimal, and the selected set covers all items in U.

Given an arbitrary instance of the *WSC* problem, we construct an instance for *batch prompting* as follows: we first assign τ_0 to ∞, and then we relax the cost limit (τ_2) of each group, i.e., all questions are in one group. So this problem can be translated to selecting demonstrations with minimum costs to cover all questions. For each demonstration d_j, there are several similar questions whose *QDA* is less than τ_1, and the number of these questions is less than τ_3, we denote Q_{d_j} as these similar questions for d_j. We map $\{d_j + Q_{d_j}\}$ as the subset s_j in *WSC*, the cost of demonstration d_j and corresponding similar questions Q_{d_j} as the weight w_j of the s_j, and all questions are the universe U.

After this mapping, we can observe that there is a set cover in the *WSC* instance if and only if the selected demonstrations cover all questions with minimal cost. In this way, we can reduce *WSC* to *batch prompting*. Since *WSC* is known to be NP-hard [27], *batch prompting* is also NP-hard. □

3.2 Exact Method

In this section, we discuss a MILP-based solution. Specifically, we need to transform the abstract format in Equation 3, such that it can be solved by MILP solvers. Let $x(\mathcal{G}_l, i)$ denote a boolean decision variable of whether a question q_i belongs to group $\mathcal{G}_l$ or not, and $y(\mathcal{G}_l, j)$ denotes a boolean decision variable of whether a demonstration d_j belongs to group $\mathcal{G}_j$ or not. The following equation shows the transformation:

$$\min \sum_{l}^{N} (\sum_{i}^{N} c_{q_i} x(\mathcal{G}_l, i) + \sum_{j}^{M} c_{d_j} y(\mathcal{G}_l, j) + u(l) c_T) \tag{4a}$$

$$s.t. \sum_{l}^{N} x(\mathcal{G}_l, i) = 1, \forall i \in [1, N]; \tag{4b}$$

$$QGA(q_i, q_{i'}) x(\mathcal{G}_l, i) x(\mathcal{G}_l, i') \leq \tau_0, \forall l, i, i' \in [1, N]; \tag{4c}$$

$$\sum_{l}^{N} \sum_{j}^{M} (QDA(q_i, d_j) x(\mathcal{G}_l, i) y(\mathcal{G}_l, j)) \geq 1, \forall i \in [1, N]; \tag{4d}$$

$$\sum_{i}^{N} c_{q_i} x(\mathcal{G}_l, i) + \sum_{j}^{M} c_{d_j} y(\mathcal{G}_l, j) + c_T \leq \tau_2, \forall l \in [1, N]; \quad (4e)$$

$$u(l) \leq \sum_{i}^{N} x(\mathcal{G}_l, i) + \sum_{j}^{M} y(\mathcal{G}_l, j), \forall l \in [1, N]; \quad (4f)$$

$$u(l)L \geq \sum_{i}^{N} x(\mathcal{G}_l, i) + \sum_{j}^{M} y(\mathcal{G}_l, j), \forall l \in [1, N]; \quad (4g)$$

Equation 4a is the objective of our optimization problem, which minimizes the total costs of all groups. Equation 4b constrains that each question can only belong to one group. Equations 4c and 4d are the constraints about the *QGA* and *QDA*. To demonstrate the *QDA* constraint, we can intuitively express as: $\sum_l^N \sum_j^M \mathcal{I}(\tau_1 - QDA(q_i, d_j)x(\mathcal{G}_l, i)y(\mathcal{G}_l, j)) \geq 1$. Note that $\mathcal{I}()$ denotes the indicator function, where $\mathcal{I}(x)$ returns true if $x \geq 0$, false otherwise. However, this format is challenging to be solved due to the indicator function. Fortunately, we can remove the indicator function by pre-processing it. Specifically, for each $QDA(q_i, d_j)$, we rewrite it as 1 if it is less than τ_1, 0 otherwise. Thus, the format is shown in Equation 4d. Meanwhile, to maintain the *DC* constraint, for each demonstration d_j, we sort all $QDA(q_i, d_j)$, and assign the first τ_3 questions as 1 (whose *QDA* is less than τ_1), 0 otherwise. While this pre-processing is an approximation method[1], it exhibits excellent performance in practice. Besides, the pre-processing can make the MILP solver more efficient.This is because all coefficients are changed from float numbers to boolean values. Note that we also change $QPA(q_i, q_i^{'})$ into 0 or 1. The *Group Cost* constraint is shown in Equation 4e. Note that a total of K groups $(\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_K)$ would cover all questions, where $1 \leq K \leq N$ (i.e., at least one group, or at most N singleton groups could be formed). We cannot know the optimal group number K in advance, so we introduce an additional variable $u(l)$ with Equations 4f and 4g to identify the number of groups. Note that L in Equation 4g is a pre-defined positive number that can be considered as infinity.

Once the MILP is formalized as shown in Equation 4, we use a general-purpose solver to solve it (e.g., Gurobi). We take the Figure 3 as our example. The inputs consist of five questions and four demonstrations with their costs. For example, the cost of q_1 is 2. The edge colored red indicates that the *QPA* between the two questions is less than τ_0, and the yellow edge indicates that the *QDA* between the question and the demonstration is less than τ_1. Note that we set τ_2 as 7 and τ_3 as 3. After running our MILP-based method, we can get two groups shown in the top right of the figure, which are $\{q_1, q_2, d_1\}$ and $\{q_3, q_4, q_5, d_2, d_3\}$ respectively.

3.3 Question Clustering (QueClt)

The optimal algorithm presented in Section 3.2 may be expensive since the MILP-based solution has an exponential computation time in the worst case. To expedite this procedure, we propose further optimizations.

To allow the MILP-based solution to scale to a larger number of questions, we break down the original optimization problem into a sequence of small problems. To be specific, we divide the

[1]This constraint is enforced during the MILP solving, rather than before the data is input.

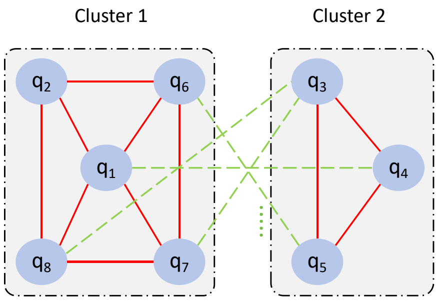


Figure 4: An example of Question Affinity Graph. The red lines denote positive edges while the green lines are negative edges. Note that we only draw partial green edges.

original questions Q into multiple clusters, making each cluster as independent as possible. After the splitting, each cluster can formalize an optimization problem without the *QGA* constraint (i.e., the Equation 4c), so that the MILP solver can be exploited in each cluster independently. As it turns out, it is faster than the original MILP-based solution due to fewer constraints and smaller data volume.

To cluster original questions, we first build the question affinity graph to present the relationships between the questions. The affinity graph is defined as follows:

Definition 1 (Question Affinity Graph (QAG)). *Given a (complete) graph $G_Q = (V_Q, E_Q)$, each node in V_Q refers to a question q_i, and each edge in E_Q refers to $QPA(q_i, q_i^{'})$ between q_i and $q_i^{'}$, whose weight is either 1 (i.e., $QPA(q_i, q_i^{'}) \leq \tau_0$, denoted as the positive edge) or -1 (otherwise, denoted as the negative edge).*

For example, Figure 4 presents a constructed *QAG*. The next step is to decompose the graph into subgraphs, which are as independent as possible. The goal is to find an optimal clustering of nodes, such that the number of unrelated nodes in the same cluster (connecting with negative edges) and the number of related nodes (connecting with positive edges) in different clusters are minimized. This is exactly the goal of correlation clustering [3]. To be specific, we give the formal definition of correlation clustering as follows:

Definition 2 (Correlation Clustering). *Let $G = (V, E)$ be an undirected (complete) graph, and the edge weights are 1 or -1. Let E^+ be the set of positive edges, and E^- be the set of negative edges. Intuitively, edge $e_{uv} \in E^+$ if u and v are similar, and $e_{uv} \in E^-$ if u and v are dissimilar. A clustering of G is a non-overlapping partition of its node set. The goal of correlation clustering problem is to minimize: $|\{e_{uv} \in E^+ | C(u) \neq C(v)\}| + |\{e_{uv} \in E^- | C(u) = C(v)\}|$. Note that $C(u) = C(v)$ means node u and v are in the same cluster, and $C(u) \neq C(v)$ means that they are in different clusters.*

There are various approximation algorithms [5, 23, 44, 51] for correlation clustering, and we can utilize them to resolve the *QueClt*. For example in Figure 4, after running the correlation clustering algorithm, two clusters are produced respectively: $\{q_1, q_2, q_6, q_7, q_8\}$ and $\{q_3, q_4, q_5\}$.

3.4 Cluster Splitting (CltSplt)

While we can use the MILP solver to obtain the optimization solution for each cluster, it is still intractable because there may be large clusters (i.e., substantial questions in the cluster) after *QueClt*. Consequently, we need to design an efficient algorithm to minimize the total costs satisfying the *GC* and *QDA* constraint. To this end, we propose our approximation method Cluster Splitting (*CltSplt*).

To better explain the *CltSplt*, we show an example in Figure 5. Specifically, the cluster in Figure 5 has five questions and four demonstrations, which can be presented as a bipartite graph (Figure 5(a)). Note that these questions correspond to one cluster in Figure 4. The blue nodes on the left part indicate the questions and the orange nodes on the right part indicate the demonstrations. The edge between one question and one demonstration means that their *QDA* is less than τ_1. Next, I will introduce the whole procedure of *CltSplt* based on this example.

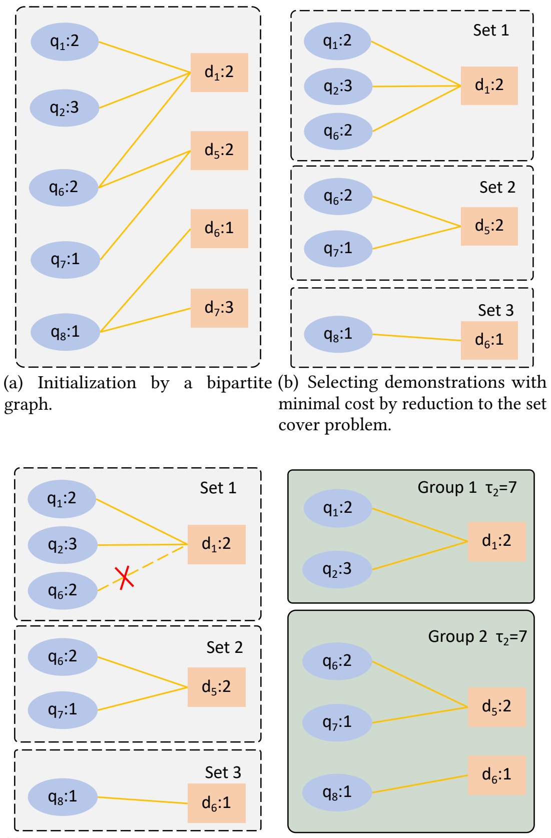
Our solution includes three stages: (1) For each cluster, we select demonstrations with minimal cost to cover all questions. Intuitively, this problem is an instance of weighted set cover problem[2], so we can use polynomial time approximation algorithms [11, 27] to solve it. As the example in Figure 5(b) shows, the selected three sets can cover all questions with minimal cost; (2) After selecting demonstrations, each demonstration may cover several questions. We thus need to combine these demonstrations with corresponding questions in groups. However, different demonstrations may cover the same question, so we should deduplicate them. We propose a metric for deduplication and a deduplication method in Section 3.4.1, aiming to balance the number of questions covered by each demonstration. For example, in Figure 5(c), we select to remove the question q_6 from “Set 1”, making the number of questions covered by these demonstrations more balanced.; (3) Because each group has a task description cost c_T, so in this stage, we need to combine the demonstrations and questions into the minimum number of groups to reduce the c_T, while satisfying the *GC* constraint. We will elaborate on the combining strategy in Section 3.4.2. For example in Figure 5(d), we combine the three sets into “Group 1” and “Group 2”, leading to minimum number of groups.

3.4.1 Deduplication Method. For deduplication, we should first establish a metric to measure the effectiveness of deduplication. As discussed in Section 2.1, we propose *Demonstration Coverage* to limit the number of questions each demonstration can cover. Similarly, after deduplication, some demonstrations may correspond to many questions, while others may correspond to few questions. This may result in some inputs having few demonstrations but many questions. This should be avoided and hence we formally define a metric as follows:

$$BC_j = \max_{d_i} |Cov(d_i)|, \forall d_i \in C_j \tag{5}$$

Note that $Cov(d_i)$ indicates questions covered by demonstration d_i after deduplication, and C_j indicates the selected demonstrations for the j-th cluster. For example in Figure 5(d), $Cov(d_1) = 2$. The goal of deduplication is to minimize BC_j. We propose a dynamic programming (DP) algorithm for deduplication. After running the set cover approximation algorithm in stage (1), we denote the set as the selected demonstration with its covered questions. We define the $DP(i, \mathcal{S})$ as the deduplication result of the first i sets with minimizing BC_j after removing all questions in $\mathcal{S}$. Thus, the final deduplication result is $DP(n, \hat{\mathcal{S}})$, assuming that the total set count is n and all redundant questions are in $\hat{\mathcal{S}}$. Thus, the recursive formula for dynamic programming can be computed as follows:

[2]The proof procedure is similar to Section 3.1



(a) Initialization by a bipartite graph.

(b) Selecting demonstrations with minimal cost by reduction to the set cover problem.

(c) Removing overlapping questions between sets.

(d) Combing sets into a minimal number of groups.

Figure 5: Illustration of the *CltSplt* procedure.

$$DP(i, \mathcal{S}) = \min \begin{cases} DP(i, \mathcal{S}) \\ \max(DP(i-1, \mathcal{S}^{'}), |Cov(d_i)| - j), \forall j \in [1, r] \end{cases} \tag{6}$$

Note that $\mathcal{S}^{'}$ plus the j questions removed from $Cov(d_i)$ are all the removed questions in $\mathcal{S}$. We assume that there are r duplicate questions in the i-th set, so we need to calculate $max(DP(i-1, \mathcal{S}^{'}), |Cov(d_i)| - j)$ for r times. $\mathcal{S}^{'}$ is not unique for $DP(i-1, \mathcal{S}^{'})$, and it can be any permutation of $\mathcal{S}$. The time complexity of this DP algorithm is $O(nr|\mathcal{S}|)$. Note that $|\mathcal{S}|$ denotes the number of $DP(i-1, \mathcal{S}^{'})$. Thus, the computation time may be exponential for the number of overlapping questions, making it challenging to obtain the optimal solution for large overlapping questions. To this end, we propose a heuristic method which is shown in Algorithm 1. We first calculate the average questions in each set after deduplication, i.e., d_{avg} (line 6). Then, for each set, if the number of questions

Algorithm 1 Deduplication Method

```
Input: Sets $\mathcal{D}_{opt}$ after Stage (1).
Output: Deduplicated Sets S.
1: S={}.
2: for $d_i$ in $\mathcal{D}_{opt}$ do
3:     S+={$d_i$, $Cov(d_i)$}.
4: end for
5: $d_{avg} = \frac{|Q_c|}{|\mathcal{D}_{opt}|}$.
6: for $s_i$ in S do
7:     while $|Cov(d_i)| > d_{avg}$ and Overlap($s_i$) do
8:         Remove overlapping questions from $s_i$.
9:     end while
10: end for
11: for $s_i$ in S do
12:     while Overlap($s_i$) do
13:         Remove overlapping questions from $s_i$.
14:     end while
15: end for
16: Return S.
```

is larger than d_{avg}, we remove its overlapping questions (lines 7-11). Finally, we iterate all sets again to avoid missing any overlapping questions (lines 12-16). Finally, the time complexity is reduced to $O(nr)$.

3.4.2 Packing Demonstrations and Questions. After deduplication, for cluster C_j, we denote s_i as one demonstration d_i with corresponding questions. In this stage, we need to assign all s_i in cluster C_j to some groups. Intuitively, we can assign each s_i as one group. However, this method is inappropriate. The reasons are twofold: (1) Each group only has one demonstration, which may cause bad performance for LLMs; (2) The most important reason is that the task description cost c_T cannot be ignored for most of tasks. For example, c_T in entity resolution is 109 tokens, but the cost of one demonstration is only 50 tokens on average. So we need to group these sets, so as to decrease the total task description cost c_T.

This problem can be reduced to the well-known bin packing problem (BPP) [24], in which items of different sizes must be packed into a finite number of bins, each with a fixed given capacity, in a way that minimizes the number of bins used. We map each set s_i as the item, each group as the bin, and τ_2 as the capacity of the bin. Given this mapping, we can reduce BPP to this problem. While BPP is NP-hard, there are some approximation algorithms for it [32, 50].

3.5 Optimization for Efficient Computation

In this section, we propose two pruning strategies to decrease some unnecessary computations.

3.5.1 Filtering affinity computations (FAC). For the MILP-based solution and approximation method, we all initially calculate the *QPA* and *QDA* for questions and demonstrations. The time complexity of them are $O(N^2)$ and $O(MN)$, separately. When the embeddings of them are high-dimensional, these computations are time-consuming. In the following, we propose an optimization method to reduce the number of computations by means of triangle

Algorithm 2 FAC

```
Input: A pivot question $p$, a batch of questions $Q$, all demonstrations $\mathcal{D}$, threshold $\tau_0$ and $\tau_1$.
Output: $QPA$ and $QDA$.
1: Calculate $QPA(p, \_) = QPA(p, q_i), \forall q_i \in Q$ and sort them.
2: for $q_i$ in $Q$ do
3:     s=BinarySearch($QPA(p, \_), QPA(p, q_i) + \tau_0, >$).
4:     e=BinarySearch($QPA(p, \_), QPA(p, q_i) - \tau_0, <$).
5:     $QPA = QPA(Q_s^e, q_i)$.
6: end for
7: Calculate $QDA(p, \_) = QDA(p, d_j), \forall d_j \in \mathcal{D}$ and sort them.
8: for $d_j$ in $\mathcal{D}$ do
9:     s=BinarySearch($QDA(p, \_), QDA(p, d_j) + \tau_1, >$)
10:    e=BinarySearch($QDA(p, \_), QDA(p, d_j) - \tau_1, <$)
11:    $QDA = QDA(\mathcal{D}_s^e, d_j)$
12: end for
13: Return $QPA$ and $QDA$.
```

inequality. First, we give the formal definition of triangle inequality property:

Definition 3 (Triangle Inequality Property). *Let the distance metric between two points a and b as dist(a,b). For any three points a, b, and c, we call the distance metric satisfying the triangle inequality if $dist(a, c) \leq dist(a, b) + dist(b, c)$.*

Note that the common metric Euclidean distance and cosine similarity all satisfy this property. For example through the computation of *QPA*, for each question q_i, we need to obtain questions whose affinity between q_i is below τ_0, denoted by $N_{\tau_0}(q_i)$. We have the following lemma:

Lemma 1. *For any two questions $q_i, q_j \in Q$, and any question q_r: $dist(q_i, q_r) - dist(q_j, q_r) > \tau_0 \Rightarrow q_i \notin N_{\tau_0}(q_j)$ and $q_j \notin N_{\tau_0}(q_i)$.*

Proof. Assume that $dist(q_i, q_r) - dist(q_j, q_r) > \tau_0$. By Definition 3, $dist(q_i, q_j) \geq dist(q_i, q_r) - dist(q_j, q_r)$, so $dist(q_i, q_j) > \tau_0$. Hence, $q_i \notin N_{\tau_0}(q_j)$ and $q_j \notin N_{\tau_0}(q_i)$. □

We can use Lemma 1 to reduce some computations for *QPA*. The designed efficient method is shown in Algorithm 2. First, we select a pivot question p (e.g., the first question), and then calculate the affinity between p and all other questions (line 1). For each question q_i except p, we use the Lemma 1 and binary search to filter questions whose affinity with q_i is larger than τ_0 and then calculate the affinity for remaining questions (lines 2-6). For the computation of *QDA*, the procedure is similar (lines 7-12). While the computation complexity remains quadratic in the worst-case, our method can reduce the computation by 90% in most cases, which are shown in Section 4.6.

3.5.2 Filtering inefficient demonstrations (FID). The demonstration pool may comprise a large number of demonstrations, causing expensive computations. Intuitively, we can prune the demonstrations whose *QDA* exceeds τ_1. After that, the remaining demonstrations are denoted as *valid* demonstrations.

Meanwhile, we can rule out those *valid* demonstrations which are *dominated*, as given in the Lemma 2 below.

Table 2: Dataset statistics.

Task	Dataset Name	# Question	# Token	Max/Min Token
ER	Fodors-Zagats(FZ)	189	8956	36/61
	Beer	91	3668	31/58
	iTunes-Amazon(iA)	109	2688	6/49
	DBLP-ACM(DA)	2378	178605	24/251
	Walmart-Amazon(WA)	2025	96860	16/123
	Abt-Buy(AB)	1650	55893	17/82
	Amazon-Google(AG)	1773	51147	12/78
	DBLP-Scholar(DS)	2707	150517	16/137
DT	Web Tables and Spreadsheet(WTS)	1259	18943	3/37
	Knowledge Base Web Tables(KBWT)	180	1008	3/10
CAG	AssertTrue(ATr)	179	44971	40/936
	AssertNotNull(ANN)	329	87635	22/1370
	AssertEquals(AE)	301	74108	54/887
	AssertThat(ATh)	75	18696	62/856

We say that a demonstration d_a is dominated by a demonstration d_b (denoted as $d_a \prec d_b$), if $DC(d_a) \subset DC(d_b)$ and $c_{d_a} \geq c_{d_b}$. We propose the following lemma:

LEMMA 2. *Pruning Dominated Demonstrations. Given two valid demonstrations d_a and d_b, if it holds that $d_a \prec d_b$, then we can safely prune d_a.*

PROOF. Suppose that $DC(d_a) \subset DC(d_b)$ and $c_{d_a} \geq c_{d_b}$, and we select d_a for some groups. In this scenario, we can replace d_a with d_b without violating the constraints in Equation 3. □

4 EVALUATION

In this section, we shall present the experimental evaluation of OBP and analyze its performance against the state-of-the-art baselines.

4.1 Experimental Setup

We implement the proposed framework in Python 3 and use gpt-3.5-turbo, gpt-4o of OpenAI [40] and Claude 3 Opus of Anthropic [10] as our backend LLMs. Meanwhile, our MILP solver is GUROBI [29].

Dataset. We evaluate our framework OBP using three tasks, i.e., entity resolution, data transformation and code assertion generation in the data management domain. In the following, we illustrate the corresponding datasets and Table 2 presents the detailed statistics of the datasets, e.g., dataset name, the number of questions of a batch (batch size), total tokens of the questions, and minimal and maximal tokens of the questions.

(1) Entity Resolution (ER). We use the well-adopted datasets from Magellan [15], which are from a variety of domains, such as products, software, etc. They include eight datasets, and we split each dataset into train, validation, and test sets by the existing ER studies [37, 43]. Each sample has two items with attributes and corresponding values, and LLMs need to determine whether these two items refer to the same entity.

(2) Data Transformation (DT). It is a critical task in the data management domain that involves converting data from its original format into another format suitable for further processing, such as data cleaning and data integration [13, 31]. We use the datasets introduced by [1, 54], which are collected from real-world tables. Each sample contains the source format and the target format, e.g., "Norm Adams" and "n.adams". LLMs need to generate the target format given the source format.

(3) Code Assertion Generation (CAG). We use the dataset from ATLAS [46] to evaluate our method, which is a CAG benchmark for JAVA code. Each sample contains the context of a unit test followed by the instruction, and LLMs need to generate a single assertion for the given test method. In this paper, we select samples and generate four datasets from ATLAS by the category of assertions, i.e., AssertTrue, AssertNotNull, AssertEquals, and AssertThat. For example, we put all the samples whose assertion method is AssertTrue into one dataset.

Baseline. We compare OBP with LLM-based baselines, which includes batching baselines (i.e., *Batcher* and *1demo*) and the non-batching baseline (i.e., *Single*). Furthermore, we compare with non-LLM SOTAs for each task. The brief introduction of these baselines is as follows:

(1) Batcher [17]. They introduce a batch prompting framework that consists of two modules, question batching and demonstration selection. Specifically, they first cluster the questions, and then select one question from each cluster separately, to generate one group (In their experiments, all the group sizes are 8). After generating groups, they leverage set cover to select demonstrations for each group.

(2) Single [17, 39]. This method asks LLMs to resolve one question at a time, with one demonstration that is most similar to the question.

(3) 1demo [17]. In this baseline, we use the same method in *Batcher* to generate groups. Then, we select the most similar demonstrations for each question in the group. For one group, the number of questions and demonstrations are the same [3].

(4) Non-LLM SOTAs. We compare against the non-LLM SOTA methods for each task. For entity resolution, we compare against Ditto [37], the current SOTA DL-based approach which finetunes BERT [14]. For data transformation, we compare against DTT [13], a SOTA example-driven solution. For code assertion generation, we compare against ATLAS [46], a Neural Machine Translation based approach.

Metric. We evaluate our method on the following metrics.

(1) Cost. This metric measures how much an approach pays for calling the API of a certain LLM, which is the main metric we consider. It is worth noting that the LLM cost is charged by input tokens. Hence, we define the *Cost* metric as the number of input tokens in this paper.

(2) Accuracy. Based on the nature of the task, we adopt different accuracy metrics for ER, DT and CAG. For ER, we use $F1$ score to measure the performance of an approach, which is a common metric for existing ER works [17, 37, 43]. Specifically, we denote the true positives, false positives, and false negatives as TP, FP, FN respectively. Then, the precision $P = \frac{TP}{TP+FP}$, and the recall $R = \frac{TP}{TP+FN}$. Finally, the score $F1 = \frac{2PR}{P+R}$. For DT and CAG, we use exact match accuracy, which denotes the percentage of numbers

[3]If the most similar demonstration of a certain question has already been selected, we choose the next most similar demonstration.

Table 3: Efficacy evaluation for the approximation method on 14 datasets. The results are under gpt-3.5-turbo. The unit of Acc is %, and the Cost refers to token counts (k). Note that the best results are bolded and the second best results are underlined.

	OBP		Batcher		Single		1demo	
	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost
FZ	**95.2**	**23.7**	**95.2**	30.2	90	53.2	90.5	38.2
Beer	**92.3**	**9.5**	88.9	11.9	88	20.3	85.7	14.2
iA	**94.1**	**7.9**	92.3	10.4	87.7	19.4	92.3	11.8
DA	93	**400.3**	**93.5**	522.7	91.3	766.9	93	612.5
WA	**83.7**	**223.7**	82.8	280.3	75.8	475.2	83	341.4
AB	**82**	**127.2**	79.5	171.2	80.7	323.1	79.4	208
AG	57.8	**175.5**	57.4	193.7	49.4	346.5	**59.4**	231.9
DS	**78.5**	**471.8**	**78.5**	501.6	78.3	778.7	77.2	588.5
WTS	**87.7**	**22.7**	87.6	32.7	82.1	136.5	83.7	51.9
KBWT	**97.2**	**2.2**	92.8	3.4	91.1	17.5	96.6	5.1
ATr	**88.8**	**87.4**	78.3	91.3	82.7	112.6	76.7	107.8
ANN	**92.1**	**174.6**	78.7	190.6	82.6	225.3	91.5	206.7
AE	**87.7**	**117.6**	79.7	133.8	79.4	176.8	87.4	175.2
ATh	**81.3**	**25**	61.3	35.6	78.7	46.4	74.7	44.7

Table 4: Performance comparison on different LLMs. The unit of Acc is %. Note that the best results are bolded and the second best results are underlined.

Acc	non-LLM	gpt-3.5		gpt-4o		claude	
	SOTA	OBP	Batcher	OBP	Batcher	OBP	Batcher
FZ	91.67	95.2	95.2	**97.7**	**97.7**	**97.7**	**97.7**
Beer	90.4	92.3	88.9	92.3	92.3	**96.5**	93.3
iA	82.1	94.1	92.3	**94.3**	89.3	91.2	89.3
DA	87.5	93	93.5	**97.2**	97	93.6	93.5
WA	74.8	83.7	82.8	**84.1**	81.5	82.8	78.1
AB	73.1	82	79.5	**89.3**	89	88	83
AG	57.8	57.8	57.4	**67.5**	67.3	61.3	57.6
DS	85.7	78.5	78.5	**93.3**	90.1	92.3	89.1
WTS	61.3	87.7	87.6	93	92.3	**94**	91
KBWT	33.3	**97.2**	92.8	96.7	92.8	94.4	91.2
ATr	72.7	**88.8**	87.6	86.6	82.7	80.5	76
ANN	90.8	92.1	78.7	**96.1**	80	94.6	77.5
AE	73.2	87.7	79.7	**91.7**	84	83.1	78.5
ATh	63.9	81.3	61.3	**82.7**	66.1	82	64.7

where the predicted output matches lexically with the expected output.

(3) Execution Time. We evaluate the efficiency of our method by the execution time, which includes the time of processing (i.e., group generation) and calling LLMs.

4.2 Overall Efficacy Comparison

In this section, we compare our framework OBP with four baselines and different LLMs on 14 real-world datasets.

4.2.1 Comparison for the Approximation Method. For the approximation method (i.e., *QueClt+CltSplt*), the comparison results are shown in Table 3. In our experiments, for each dataset, we first sort their *QPA* and *QDA* values, taking the top 25% quantile as τ_0, and the top 10% quantile as τ_1 at most. Meanwhile, τ_2 is set between 200 and 600, and τ_3 is set between 3 and 8 for all three tasks. For *Batcher*, we use the default group size 8 in ER and DT, while the group size in CAG is 2, which performs best. Furthermore, when computing *QGA*, we use the reciprocal of the Euclidean distance as the similarity metric for the ER task, which is consistent with Batcher [17]. Meanwhile, we use the Euclidean distance as the similarity metric for DT and CAG tasks, as it yields the best performance. .

Accuracy. From Table 3, we can observe that OBP performs best for almost all datasets. and only performs second in 2 datasets out of 14 datasets under gpt-3.5-turbo, which indicates that our method does not compromise the LLM performance. For all the three baselines, their performance varies across different tasks. For example, *Batcher* performs best except OBP in only 6 datasets.

Batcher uses the fixed group size (8 by default), which may lead to some questions being in the same group inappropriately. Our approach guides the grouping strategies through τ_0 and does not require a fixed number of questions per group. For *Single* and *1demo*, the performance across different datasets behaves inconsistently. For example, *Single* outperforms *1demo* on the AB dataset, but it performs worse than 1demo on most of the other datasets, which indicates that batch prompting is superior to single prompting. This is because grouping some suitable questions facilitates the reasoning of LLMs. However, *1demo* performs worse than OBP, which indicates the question grouping in *1demo* is not optimal.

Cost. Experimental results in Table 3 show that our method incurs the least cost. Compared with *Batcher*, OBP can reduce the costs by up to 35% (i.e., KBWT). Except for OBP, *Batcher* outperforms the other two baselines on cost. This is because for each group, *Batcher* exploits the set cover to select demonstrations with minimal cost, and *1demo* selects a fixed number of demonstrations, which is the same as the number of questions in the group. Thus, Batcher uses fewer examples than 1demo. Meanwhile, *1demo* produces less cost than *Single*. While the demonstration number of *Single* and *1demo* are the same (each question corresponds to a demonstration), *Single* treats each question as a group to query LLMs, resulting in higher task description cost c_T (c_T is proportional to the number of groups, and *Single* has more groups than *1demo*). However, the cost of *Batcher* is the local optimum, and our method can reach the global optimum approximately. Our method can dynamically adjust the number of groups and the number of questions per group based on the data, which is adaptive.

In summary, OBP achieves least cost without compromising performance. Additionally, the performance of other methods varies across different datasets while OBP is robust.

4.2.2 Comparison for Non-LLM SOTAs. In this section, we compare OBP with non-LLM SOTA methods of each task, which are shown in Table 4. Our results show that LLMs outperform non-LLM SOTA methods on almost all datasets. For example, gpt-4o achieves up to 97.2% in terms of accuracy on the DA dataset, whereas the non-LLM SOTA (i.e., Ditto [37]) only reaches 87.5%. Additionally, the LLM-based method is few-shot, requiring little training data (i.e., demonstrations). We find that the number of demonstrations needed for LLMs is between 10% and 20% of that needed by the non-LLM baselines. These results demonstrate that LLMs generalize and achieve SOTA performance on these datasets.

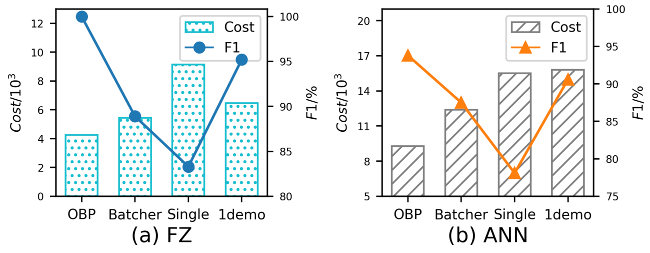


Figure 6: Efficacy evaluation for the MILP-based method.

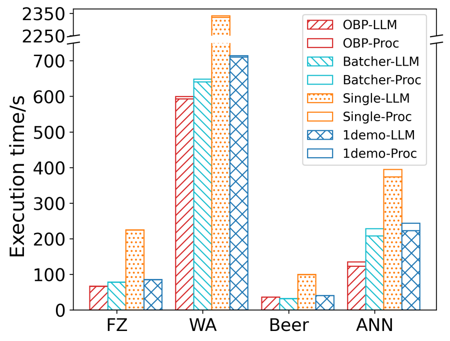


Figure 7: Efficiency evaluation on four datasets.

4.2.3 Comparison for Different LLMs. In this section, we compare the performance of OBP with *Batcher* on different LLMs. The corresponding results are shown in Table 4. We can observe that OBP is superior to *Batcher* for almost all tasks and LLMs, which demonstrates that our method is robust in terms of accuracy. As for cost, our method yields the same cost across different LLMs for the same dataset. Therefore, the cost results are the same as those in Table 3. Among these LLMs, gpt-4o performs well on most datasets, it performs best in 10 datasets out of all 14 datasets, which demonstrates its powerful performance. Note that the other experimental results in Section 4 are based on gpt-3.5-turbo. It can be observed that using a more powerful LLM (e.g., gpt-4o) would yield even better results.

4.2.4 Comparison for the MILP-based Solution. For our MILP-based solution, the comparison results are shown in Figure 12. The MILP-based solution is computationally expensive, even 100 questions require at least half an hour. So in our experiments, we only use the first 32 questions as a batch and display the results of two datasets due to the limitation of space. We can observe a similar trend as Table 3. First, our method achieves the least cost. Take the FZ dataset as an example, we can save costs up to 20% compared to *Batcher*. Then, for accuracy, our method outperforms others on most datasets. For example, OBP achieves 100% on the FZ dataset. Since the optimization objective of the MILP-based solution is to minimize costs without compromising the LLM performance, the results validate that our formulation in Equation 4 is reasonable.

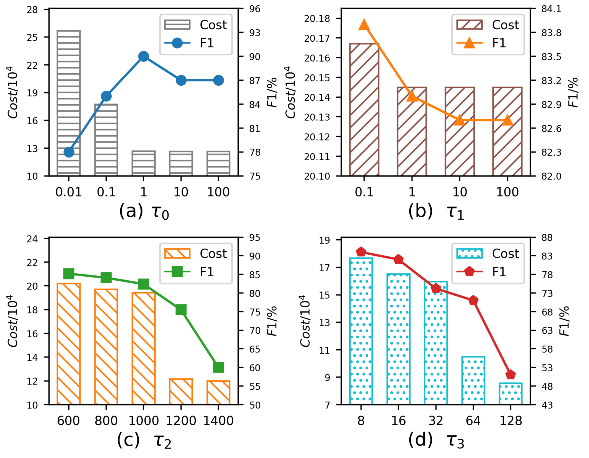


Figure 8: Sensitivity analysis of hyperparameters on the WA dataset.

4.3 Evaluation on End-to-end Execution Time

In this section, we compare the execution time between OBP and the baselines. In Figure 7, we present the processing time and the time of calling LLMs on four datasets. We can find that the processing time can be ignored compared to the time of calling LLMs. Note that the processing time for the dataset AssertNotNull is relatively long, primarily due to the embedding computations in this dataset being time-consuming. Meanwhile, we find that our method requires less time when calling LLMs. The reasons are twofold. First, OBP generates fewer groups compared to baselines, thereby reducing the number of LLM callings. Second, the generated groups of OBP can provide LLMs with more useful information, thereby saving the inference time. However, we spend more time calling LLMs compared to *Batcher* on the Beer dataset. This is because our method generates more groups than the *Batcher* (17 vs 12).

In summary, OBP achieves the least execution time on most datasets, whereas *Single* has the longest execution time due to its large number of groups.

4.4 Sensitivity Analysis of Hyperparameters

Our framework has four hyperparameters: $\tau_0, \tau_1, \tau_2, \tau_3$. In this section, we evaluate the impact of each parameter on the performance, which are shown in Figures 8 and 9.

Hyperparameters τ_0. Figure 8(a) and 9(a) illustrate the trend of accuracy and cost as τ_0 varies. First, we find that the cost decreases as τ_0 increases. This is because a smaller τ_0 implies fewer questions per group, resulting in a larger number of groups and thus increasing the task description cost c_T. Additionally, some questions that could share demonstrations cannot be grouped together, leading to more demonstrations. Furthermore, the trend of accuracy is to first increase and then decrease. For small τ_0 such as 0.01 in the WA dataset, they become *Single*, i.e., one question is a group. From Table 3, we can observe that *Single* performs poorly in most cases, so the accuracy is relatively low when τ_0 is very small. When τ_0 becomes large, the questions within each group tend to be randomly

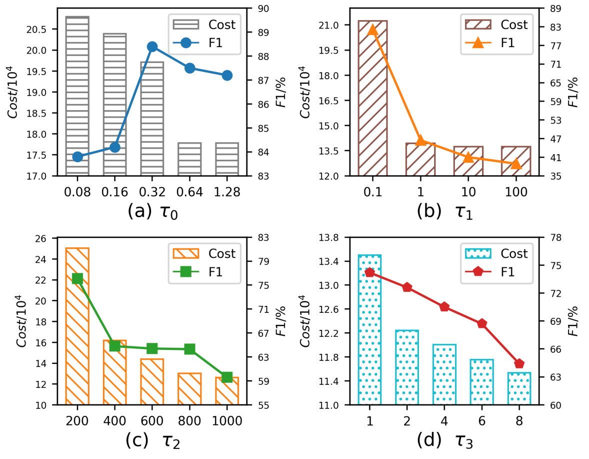


Figure 9: Sensitivity analysis of hyperparameters on the ANN dataset.

selected. Even when τ_0 is relatively large, all questions are in one group (when τ_2 is also large), which leads to poor performance.

Hyperparameters τ_1. We use τ_1 to limit the similarity (i.e., the *QDA*) between questions and demonstrations. Figures 8(b) and 9(b) present the trend of accuracy and cost as τ_1 varies. We record the variations of τ_1 from 0.1 to 100. From these figures, we find that the cost decreases as τ_1 increases. This is because when τ_1 is relatively small, each demonstration may cover fewer questions, so more demonstrations are needed to cover all the questions within a group. Meanwhile, as τ_1 increases, the performance deteriorates. This is because there are fewer demonstrations within each group. At the same time, the selected demonstrations may differ significantly from the questions with a large τ_1, which do not effectively assist in the reasoning of LLMs.

Hyperparameters τ_2. τ_2 measures the cost upper bound of each group. The corresponding evaluations are shown in Figures 8(c) and 9(c). When τ_2 becomes large, the cost decreases. This is because a group can accommodate more questions as τ_2 is large, thereby reducing the number of groups (i.e., reducing the task description cost c_T). Additionally, more similar questions can be grouped together, allowing for fewer demonstrations to cover all questions. For accuracy, they consistently decrease as τ_2 increases. When τ_2 becomes large, the LLM performance tends to degrade due to the larger number of questions within each group, leading to large input context.

Hyperparameters τ_3. τ_3 restricts the maximum number of questions that a demonstration can cover, ensuring that the number of demonstrations within a group does not become significantly small. When τ_3 is large, one demonstration may cover more questions, causing fewer costs. As for accuracy, when τ_3 is relatively large, the LLM performance becomes worse. This is because one group may have fewer demonstrations, resulting in less domain knowledge LLMs obtained. LLMs consequently reason correctly for fewer questions.

Table 5: Comparison between deduplication Methods. The unit of Acc is %, and Time is second. Note that the best accuracy are bolded.

	Random		DP		Greedy	
	Acc	Time	Acc	Time	Acc	Time
iA	89.36	0.00008	**95.83**	0.44	93.62	0.00027
ANN	90.25	0.0001	**92.71**	0.1	**92.71**	0.00081

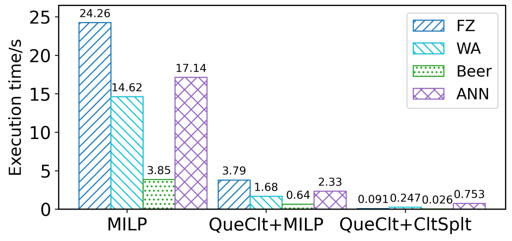


Figure 10: Comparison between *MILP*, *QueClt+MILP* and *QueClt+CltSplt*. The y-axis indicates the execution time.

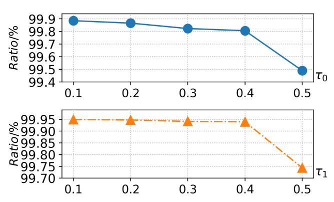
4.5 Comparison of Different Design Choices

Next, we evaluate the effectiveness of different design choices. First, we compare different deduplication methods described in Section 3.4. Then, we compare the three methods in OBP(Section 3), i.e., *MILP*, *QueClt+MILP* and *QueClt+CltSplt*.

Comparison between deduplication Methods. In Section 3.4, we propose two methods to deduplicate questions from different sets, namely a *DP*-based method and a *Greedy* method. We compare these two methods in this section. Furthermore, we add a *Random* method as a baseline, i.e., deleting redundant questions in the current set sequentially. The corresponding results are shown in Table 5. We can observe that the *DP*-based method achieves the optimal performance, which confirms that the proposed metric in Equation 5 is reasonable, i.e., balancing the number of questions covered by each demonstration can improve the performance. However, it is inefficient, which is exponential for the number of overlapping questions. We can find that the *DP*-based method is at least 1000 times slower than the other two methods. Meanwhile, the *Greedy* method performs similarly to the *DB*-based method in terms of accuracy, while its computational complexity is significantly low. In summary, the *Greedy* method is both effective and efficient.

Comparison between three methods in OBP. In Section 3, we first proposed a *MILP*-based method. Then, we propose to leverage correlation clustering to split the original questions into several clusters (*QueClt*), and then use MILP solver to resolve each cluster individually, calling it *QueClt+MILP*. We further propose an approximation method to replace the MILP solver (*CltSplt*) for better efficiency, and call it *QueClt+CltSplt*. Next, we evaluate them.

Figure 10 shows the comparison results on four datasets. Note that the number of questions for these datasets is 80, and the question count of each cluster is around 30 after correlation clustering. After correlation clustering, the MILP solver and *CltSplt* for each cluster can be executed concurrently, so the time for *QueClt+MILP* and *QueClt+CltSplt* refer to the slowest time among all clusters. We can find that compared to *MILP*, the execution time of *QueClt+MILP*


(a) WA

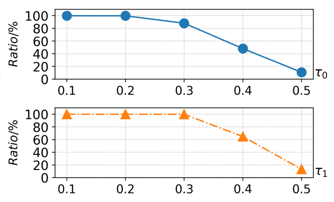

(b) ANN

Figure 11: Evaluation on *FAC*. The x-axis presents different τ_0 or τ_1 values, and the y-axis indicates the ratio of filtering computations.

can be reduced by around 85%. Take the FZ dataset as an example, it takes 24.26s in *MILP*, but only 3.79s in *QueClt+MILP*. If we have n questions, this method can eliminate n boolean variables and $O(n^2)$ constraints in Equation 4, so as to accelerate the MILP solver. Furthermore, *QueClt+CltSplt*, which replaces the computationally expensive MILP solver, achieves the shortest execution time. For example, the runtime of *QueClt+CltSplt* is only 0.2% of *QueClt+MILP* on the FZ dataset, which demonstrates the high efficiency of *QueClt+CltSplt*.

4.6 Evaluation on Efficient Computation

In Section 3.5, we propose two pruning strategies to decrease unnecessary computations, namely *FAC* and *FID*. Next, we will validate these two optimizations.

4.6.1 FAC Evaluation. We need to first generate embeddings for each question and demonstration, and then calculate the *QPA* and *QDA*. For high-dimensional embeddings, these computations are significantly slow [45], so we propose a pruning strategy in Section 3.5.1 to reduce the number of computations.

The evaluation results are shown in Figure 11. We validate the efficiency with different values of hyperparameters τ_0 and τ_1. We find that when τ_0 or τ_1 are small, this method can filter out the vast majority of computations, i.e., over 99.9%. This is because when these thresholds are small, only a small fraction of values in *QPA* and *QDA* are below the thresholds, so most of the values satisfy the Lemma 1. However, when the threshold of the WA dataset changes from 0.4 to 0.5 and the threshold of the ANN dataset changes from 0.3 to 0.4, the ratio drops sharply. This indicates that the number of values in *QPA* and *QDA* below the thresholds is increasing. Meanwhile, it also demonstrates that the distributions of *QPA* and *QDA* are not uniform, and the distribution patterns vary between different datasets. In conclusion, our optimization method indeed reduces the computations of *GPA* and *QDA*, and the extent of reduction depends on the hyperparameters (i.e., τ_0 and τ_1) and data distribution.

4.6.2 FID Evaluation. The demonstration pool comprises large numbers of demonstrations, causing expensive computations. In section 3.5.2, We propose *FID* to reduce the number of demonstrations. In Figure 12, we validate this optimization on four datasets under the MILP-based solution. We can observe that *FID* can reduce computations by up to 42.03% (i.e., the WA dataset). This is because

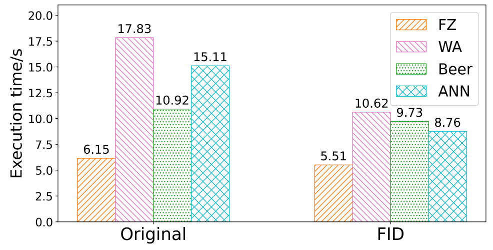


Figure 12: Evaluation on *FID* on four datasets.

FID can filter dominated demonstrations, causing the Equation 4 to have fewer variables.

5 RELATED WORK

This work is related to two broad lines of research: LLMs prompting and LLMs for data management.

LLMs prompting. The main trend of using LLMs is through prompting [4, 47]. This approach has changed the research paradigm in artificial intelligence. LLMs only need to be given a suite of appropriate prompts. The major advantage of LLMs is that they do not need model training or fine-tuning, which is efficient for deployment and generally applicable to the majority of downstream tasks. Commonly, prompt can be divided into hard prompt [48] and soft prompt [49] respectively. Hard prompt means the same context for all questions, such as the task description and fixed demonstrations. Soft prompt typically refers to selecting or generating prompts automatically. For example, recent works try to retrieve demonstrations similar to the question for improving LLM performance [17, 21, 39]. In this paper, we focus on batching questions under the soft prompt setting.

LLMs for data management. LLMs have recently achieved record-breaking results in various real-world applications. Also, recent works [19, 38] explore the LLM performance on the data management tasks. For example, entity resolution [38], data transformation [33], code assertion generation [39], and so on. While LLMs demonstrate promising performance on these tasks, there are still some challenges. For example in some data management tasks, they tend to process a large number of questions, which can be very costly for LLMs, such as entity resolution. So we need to develop a cost-effective method. In this paper, we propose optimizations to reduce costs for LLMs through batching questions and demonstrations. We next elaborate more on three applications: entity resolution, data transformation and code assertion generation.

Entity resolution is a significant data integration task that has been studied for decades. There are a lot methods, such as rule-based methods [18, 42], crowd-based methods [7, 8, 16, 35] and deep learning-based methods [37]. Recently, some researchers have proposed to exploit LLMs to resolve the entity resolution [37, 38]. The second application is data transformation. There are numerous studies in this area [1, 28, 31, 41, 54]. For example, DTT [13] exploits Language Model (LM) to transform source values into the desired target representation. Last, testing software is a vital component of the development process, requiring considerable effort and time investment. Recent works [12, 39] have leveraged LLMs for the generation of code assertions, with an emphasis on the strategic selection and combination of demonstrations. In this paper, we focus on batch prompting these applications, so as to reduce LLM costs while achieving better performance.

6 CONCLUSION

In this paper, we study the problem of batch prompting used in LLMs. To address it, we develop a framework OBP aiming to find the optimal grouping of questions and demonstrations with performance guarantees and minimal cost. We first identify the factors that are significant for this problem, i.e., Question Group Affinity (*QGA*), Question Demonstration Affinity (*QDA*), Group Cost (*GC*) and Demonstration Coverage (*DC*). Then, we propose the formulation of this problem and show that our overall problem is NP-hard. Finally, we provide a MILP-based solution and an efficient approximation algorithm. Extensive experiments on 14 real-world datasets show the superiority of our OBP compared to state-of-the-art baselines.

REFERENCES

[1] Ziawasch Abedjan, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, and Michael Stonebraker. 2016. Dataxformer: A robust transformation discovery system. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 1134–1145.

[2] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. *Proceedings of the VLDB Endowment* 17, 2 (2023), 92–105.

[3] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. 2004. Correlation clustering. *Machine learning* 56 (2004), 89–113.

[4] Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. 2023. Prompting is programming: A query language for large language models. *Proceedings of the ACM on Programming Languages* 7, PLDI (2023), 1946–1969.

[5] Marco Bressan, Nicolò Cesa-Bianchi, Andrea Paudice, and Fabio Vitale. 2019. Correlation clustering with adaptive similarity queries. *Advances in Neural Information Processing Systems* 32 (2019).

[6] Alberto Caprara, Paolo Toth, and Matteo Fischetti. 2000. Algorithms for the set covering problem. *Annals of Operations Research* 98, 1 (2000), 353–371.

[7] 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.

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

[9] Zhoujun Cheng, Jungo Kasai, and Tao Yu. 2023. Batch prompting: Efficient inference with large language model apis. *arXiv preprint arXiv:2301.08721* (2023).

[10] Claude. [n.d.]. Claude API. Retrieved 2024 from https://www.anthropic.com/

[11] Marek Cygan, Łukasz Kowalik, and Mateusz Wykurz. 2009. Exponential-time approximation of weighted set cover. *Inform. Process. Lett.* 109, 16 (2009), 957–961.

[12] Arghavan Moradi Dakhel, Amin Nikanjam, Vahid Majdinasab, Foutse Khomh, and Michel C Desmarais. 2024. Effective test generation using pre-trained large language models and mutation testing. *Information and Software Technology* (2024), 107468.

[13] Arash Dargahi Nobari and Davood Rafiei. 2024. DTT: An Example-Driven Tabular Transformer for Joinability by Leveraging Large Language Models. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–24.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[15] AnHai Doan, Pradap Konda, Paul Suganthan GC, Yash Govind, Derek Paulsen, Kaushik Chandrasekhar, Philip Martinkus, and Matthew Christie. 2020. Magellan: toward building ecosystems of entity matching solutions. *Commun. ACM* 63, 8 (2020), 83–91.

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

[17] Meihao Fan, Xiaoyue Han, Ju Fan, Chengliang Chai, Nan Tang, Guoliang Li, and Xiaoyong Du. 2023. Cost-Effective In-Context Learning for Entity Resolution: A Design Space Exploration. *arXiv preprint arXiv:2312.03987* (2023).

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

[19] Raul Castro Fernandez, Aaron J Elmore, Michael J Franklin, Sanjay Krishnan, and Chenhao Tan. 2023. How large language models will disrupt data management. *Proceedings of the VLDB Endowment* 16, 11 (2023), 3302–3309.

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

[21] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363* (2023).

[22] Shuzheng Gao, Xin-Cheng Wen, Cuiyun Gao, Wenxuan Wang, Hongyu Zhang, and Michael R Lyu. 2023. What makes good in-context demonstrations for code intelligence tasks with llms?. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 761–773.

[23] David García-Soriano, Konstantin Kutzkov, Francesco Bonchi, and Charalampos Tsourakakis. 2020. Query-efficient correlation clustering. In *Proceedings of The Web Conference 2020*. 1468–1478.

[24] Michael R Garey and David S Johnson. 1981. Approximation algorithms for bin packing problems: A survey. In *Analysis and design of algorithms in combinatorial optimization*. 147–172.

[25] Shawn Gavin, Tuney Zheng, Jiaheng Liu, Quehry Que, Noah Wang, Jian Yang, Chenchen Zhang, Wenhao Huang, Wenhu Chen, and Ge Zhang. 2024. LongIns: A Challenging Long-context Instruction-based Exam for LLMs. *arXiv preprint arXiv:2406.17588* (2024).

[26] Lise Getoor and Ashwin Machanavajjhala. 2012. Entity resolution: theory, practice & open challenges. *Proceedings of the VLDB Endowment* 5, 12 (2012), 2018–2019.

[27] Lukasz Golab, Flip Korn, Feng Li, Barna Saha, and Divesh Srivastava. 2015. Size-constrained weighted set cover. In *2015 IEEE 31st International Conference on Data Engineering*. 879–890.

[28] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 317–330.

[29] LLC Gurobi Optimization. [n.d.]. Gurobi Optimizer Reference Manual. Retrieved 2022 from https://www.pgurobi.com

[30] Hongye Jin, Xiaotian Han, Jingfeng Yang, Zhimeng Jiang, Zirui Liu, Chia-Yuan Chang, Huiyuan Chen, and Xia Hu. 2024. Llm maybe longlm: Self-extend llm context window without tuning. *arXiv preprint arXiv:2401.01325* (2024).

[31] Zhongjun Jin, Yeye He, and Surajit Chauduri. 2020. Auto-transform: learning-to-transform by patterns. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2368–2381.

[32] David S Johnson. 1974. Fast algorithms for bin packing. *J. Comput. System Sci.* 8, 3 (1974), 272–314.

[33] Moe Kayali, Anton Lykov, Ilias Fountalis, Nikolaos Vasiloglou, Dan Olteanu, and Dan Suciu. 2023. CHORUS: foundation models for unified data discovery and exploration. *arXiv preprint arXiv:2306.09610* (2023).

[34] Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. 2024. BABILong: Testing the Limits of LLMs with Long Context Reasoning-in-a-Haystack. *arXiv preprint arXiv:2406.10149* (2024).

[35] Guoliang Li, Chengliang Chai, Ju Fan, Xueping Weng, Jian Li, Yudian Zheng, Yuanbing Li, Xiang Yu, Xiaohang Zhang, and Haitao Yuan. 2017. CDB: optimizing queries with crowd-based selections and joins. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1463–1478.

[36] Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. 2024. Long-context llms struggle with long in-context learning. *arXiv preprint arXiv:2404.02060* (2024).

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

[38] Avanika Narayan, Ines Chami, Laurel Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *Proceedings of the VLDB Endowment* 16, 4 (2022), 738–746.

[39] Noor Nashid, Mifta Sintaha, and Ali Mesbah. 2023. Retrieval-based prompt selection for code-related few-shot learning. In *Proceedings of the 45th International Conference on Software Engineering*.

[40] OpenAI. [n.d.]. OpenAI API. Retrieved 2024 from https://platform.openai.com/

[41] Rishabh Singh. 2016. Blinkfill: Semi-supervised programming by example for syntactic string transformations. *Proceedings of the VLDB Endowment* 9, 10 (2016), 816–827.

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

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

[44] Norases Vesdapunt, Kedar Bellare, and Nilesh Dalvi. 2014. Crowdsourcing algorithms for entity resolution. *Proceedings of the VLDB Endowment* 7, 12 (2014), 1071–1082.

[45] Yongxin Wang, Zhen-Duo Chen, Xin Luo, and Xin-Shun Xu. 2021. High-dimensional sparse cross-modal hashing with fine-grained similarity embedding. In *Proceedings of the Web Conference 2021*. 2900–2909.

[46] Cody Watson, Michele Tufano, Kevin Moran, Gabriele Bavota, and Denys Poshyvanyk. 2020. On learning meaningful assert statements for unit test cases. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 1398–1409.

[47] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[48] Yuxin Wen, Neel Jain, John Kirchenbauer, Micah Goldblum, Jonas Geiping, and Tom Goldstein. 2024. Hard prompts made easy: Gradient-based discrete optimization for prompt tuning and discovery. *Advances in Neural Information Processing Systems* 36 (2024).

[49] Hui Wu and Xiaodong Shi. 2022. Adversarial soft prompt tuning for cross-domain sentiment analysis. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2438–2447.

[50] Andrew Chi-Chih Yao. 1980. New algorithms for bin packing. *Journal of the ACM (JACM)* 27, 2 (1980), 207–227.

[51] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, Cecilia M Procopiuc, and Divesh Srivastava. 2011. Automatic discovery of attributes in relational databases. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 109–120.

[52] Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M Patel. 2024. ReAcTable: Enhancing ReAct for Table Question Answering. *Proceedings of the VLDB Endowment* 17, 8 (2024), 1981–1994.

[53] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).

[54] Erkang Zhu, Yeye He, and Surajit Chaudhuri. 2017. Auto-join: Joining tables by leveraging transformations. *Proceedings of the VLDB Endowment* 10, 10 (2017), 1034–1045.