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 demonstrated exceptional performance in various real-world data management tasks through in-context learning (ICL), which involves structuring prompts with task descriptions and several demonstrations. However, most LLMs are not free and charge based on the number of input tokens. Specifically, for data management tasks, there may be massive related questions, leading to high inference cost due to redundant prompt content (i.e., overlapping demonstrations and repeated task descriptions). In this paper, we investigate the idea of batch prompting in leveraging LLMs for data management, which leads to cost-effective LLMs by grouping questions and demonstrations to perform inferences in batches. Current studies on batch prompting are preliminary and mostly based on heuristics, making it difficult to generalize to various types of tasks and adapt to different grouping strategies. To address these challenges, in this work we first formalize the batch prompting problem in general setting. Then, we study the hardness of this problem and propose efficient algorithms for adaptive grouping. Finally, we conduct comprehensive experiments on 14 datasets. Extensive experimental results demonstrate that our solution consistently outperforms the state-of-the-art baselines while consuming lower cost.

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 a wide range of real-world applications, such as question answering, machine translation, context summarization, etc [61]. In particular, recent works [7, 24, 26, 27, 45, 60] delve into applying LLMs to data management tasks for improving accuracy, such as entity resolution [24] and data transformation [45]. These works typically employ in-context learning (ICL) [21], where several demonstrations (i.e., examples with corresponding answers from the same/similar task) are provided together with the task description and the target question in the prompt, see Single Prompting in Figure 1 for the illustration.

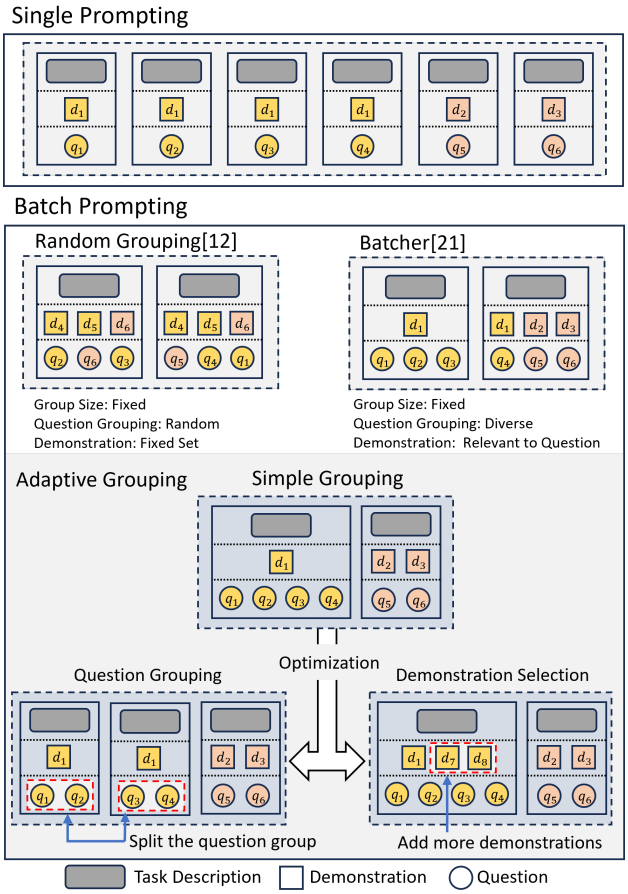


Figure 1: Comparison of different prompting strategies. Questions and demonstrations with the same color indicate they are affinitive (i.e., similar or diverse) to each other.

However, most closed-source LLMs are not free and charge based on the number of input tokens when calling the provided APIs. For data management tasks, there are usually a large number of questions, leading to prohibitively high monetary cost for using

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

LLMs. Consider the data transformation [18] task as an example. Real-world datasets may contain a massive number of data, e.g., there are 165,236 data tables in the open dataset published by the U.S. government in March 2017 [46], each with tens of thousands of data rows on average. Transforming each row using LLMs, with around 100 tokens per row, would cost over $900k with GPT-4o ($5.00/1M tokens) [4, 7].

Batch prompting is an effective way for LLM cost reduction. To be specific, several questions can be consolidated in a single prompt for LLMs to perform inference, thereby eliminating redundant demonstrations and task descriptions. Batch prompting has been preliminarily explored in recent works [14, 24]. The method in [14] proposes to randomly combine questions into groups, each with the same number of questions and a fixed demonstration set. As Random Grouping in Figure 1 shows, this method randomly divides six questions into two groups, sharing the same demonstration set $\{d_4, d_5, d_6\}$. Batcher [24] targets only the entity resolution task. It clusters diverse questions into groups with a fixed group size. As shown in Figure 1, Batcher clusters questions with size 3. After the question grouping, it selects demonstrations that are the most relevant to questions within the group, e.g., it selects d_1 for the first group since it is the most relevant to all three questions $\{q_1, q_2, q_3\}$. While these works have preliminarily demonstrate the effectiveness of batch prompting on reducing LLM cost, several challenges remain unresolved.

Challenge 1: There lacks a general solution to the problem of LLM batch prompting. Existing works rely on fixed grouping strategies and are specifically designed for certain tasks. However, different tasks may require different grouping strategies. For instance, grouping similar questions works well for data transformation task since similar questions can help LLMs learn transformation rules for data of the same type. However, in entity resolution task, a better strategy is to group related questions (e.g., electronic products) but avoid putting questions with high similarity (e.g., iPhone 14 and iPhone 14 plus) in one group in order to reduce the ambiguity [24]. It therefore calls for an LLM batch prompting approach that is generally applicable to typical data management tasks.

Challenge 2: There is no adaptive question grouping solution for batch prompting. Existing methods use a fixed group size, i.e., the number of questions in all groups remains constant. However, it is always difficult or even infeasible to choose an appropriate group size. Further, a fixed size may degrade the performance (e.g., accuracy and cost) of LLMs. To be specific, if the number is set too large, unrelated questions may be grouped together, negatively affecting the accuracy of LLMs, e.g., the second group in the Batcher example in Figure 1. On the contrary, if the group size is small, the proliferation of groups would increase the overall cost. Therefore, a better strategy should be to adaptively group questions according to the number of related questions. A simple way is to form groups by clustering all related questions into one group. However, this may produce groups that have too many questions (e.g., group 1 in Simple Grouping in Figure 1), which could simply exceed the input token limit of LLMs, or result in low accuracy due to large input context [31, 34, 39, 42]. Thus, an optimized strategy should avoid large group sizes. For example in Figure 1, the first group in Simple Grouping should be further divided into two groups, i.e, $\{q_1, q_2\}$ and $\{q_3, q_4\}$ in the Question Grouping part. How to adaptively cluster related questions into groups while ensuring low cost and high accuracy is thus challenging.

Challenge 3: There is no effective strategy for demonstration selection. Existing methods either use a fixed demonstration set or select the most similar demonstrations to the questions for each group. Although the latter is regarded as more effective [24], it often results in groups containing an excessive number of questions with insufficient demonstrations, thereby impairing the reasoning ability of LLMs. For instance in Figure 1, the first group of Simple Grouping has only one demonstration (i.e., d_1), potentially compromising the accuracy of LLMs. It would be beneficial to add more demonstrations, such as d_7 and d_8 for better LLM accuracy, as outlined in the Demonstration Selection part. Thus, how to effectively select demonstrations in each group while balancing the LLM performance and the overall cost is the third challenge.

In this paper, we investigate batch prompting techniques for data management tasks that consist of numerous interdependent questions, with the goal of minimizing the overall LLM cost while maintaining the resulting accuracy. We first formalize this problem as a constrained optimization problem. Specifically, we consider four factors affecting the accuracy, including Question Group Affinity (relationship between questions), Question Demonstration Affinity (relationship between questions and demonstrations), Group Length (the number of input tokens in a group) and Demonstration Coverage (balance between questions and demonstrations). We further demonstrate that this overall problem is NP-hard. To resolve this problem, we propose a framework for data management tasks called **O**ptimized **B**atch **P**rompting (OBP) with exact and approximation solutions. In detail, we develop a question clustering method, and then introduce a three-staged approach for adaptively grouping questions and effectively selecting demonstrations. Last, we further propose two optimizations to enhance the efficiency of the framework, which filters out unnecessary demonstrations that are covered by others, and reduce question affinity computations by means of the triangle inequality property, respectively.

In summary, our main contributions are as follows:

- We investigate batch prompting for data management tasks, and formalize it as an optimization problem. We demonstrate it is NP-hard in general with a theoretical analysis.
- We propose the Optimized Batch Prompting (OBP) framework that adaptively groups questions and selects demonstrations, and generates results with low cost while ensuring the accuracy.
- We propose two optimizations to accelerate the computation, which filter unnecessary demonstrations and reduce the number of affinity calculations respectively.
- We conduct extensive experiments across three tasks with 14 datasets to evaluate the effectiveness and efficiency of our proposed method. Results show that our method reduces the cost by up to 35% compared to the state-of-the-art LLM and non-LLM based baselines. Meanwhile, our method outperforms the baselines on almost all datasets in terms of accuracy.

The rest of the paper is organized as follows. We formalize the batch prompting problem in Section 2. We then propose our framework in Section 3. In Section 4, we provide an extensive set of

Table 1: Notations used in this paper.

Notations	Definition
N	The number of questions in one batch.
M	The number of total demonstrations in the pool.
K	The number of optimal groups for a batch of N questions, which is unknown in advance.
Q	$(q_i)_{i=1}^{N}$, questions in the batch.
D	$(d_j)_{j=1}^{M}$, demonstrations in the pool.
G	$(g_l)_{l=1}^{K}$, optimal groups after batch prompting.
c_T	The cost of the task description.
c_{q_i}	The cost of question q_i.
c_{d_j}	The cost of demonstration d_j.
τ_0	The question group affinity threshold for each group.
τ_1	The question demonstration affinity threshold.
τ_2	The threshold of group length.
τ_3	The coverage threshold of each demonstration.
$aff_q(g_l)$	The function of computing the question group affinity of group g_l.
$aff_p(q_i, q_{i'})$	The function of computing the question pair-wise affinity between questions q_i and $q_{i'}$.
$aff_d(q_i, d_j)$	The function of computing the question demonstration affinity between q_i and d_j.
$c(g_l)$	The cost of the group g_l.
$cov(d_j)$	The function of computing demonstration coverage of demonstration d_j.

experiments to validate the effectiveness and efficiency of our approach for the batch prompting. Finally, we discuss the related work in Section 5 and conclude the paper in Section 6.

2 PROBLEM FORMULATION

In this section, we first formulate the batch prompting problem. We then present a rigorous theoretical analysis of the problem and prove the hardness.

2.1 Problem Statement

For a batch of questions $Q = \{q_1, q_2, ..., q_N\}$ of certain data management tasks, 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. Assuming the optimal number of final groups is K, i.e., $\mathcal{G} = \{g_1, g_2, ...g_l, ..., g_K\}$, each group g_l consists of a task description $\mathcal{T}$, a subset of questions $Q_l \in Q$ and a subset of demonstrations $\mathcal{D}_l \in \mathcal{D}$. Thus, g_l can be written as $g_l = \{\mathcal{T}, \mathcal{D}_l, Q_l\}$.

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 cost, denoted by c_{q_i} and c_{d_j}. The cost of the task description is denoted as c_T. The cost of each group includes c_T and the cost 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.

Given a batch of questions Q and a pool of demonstrations $\mathcal{D}$, our goal is to cluster all questions in Q into groups with minimizing the overall cost of all groups. i.e., min $\sum_l c(g_l)$. However, merely minimizing the cost can result in low accuracy. Therefore, we need to minimize the cost without compromising the accuracy.

From a careful analysis, we observe the following four factors mostly affect the LLM accuracy in the batch prompting problem.

(1) Question Group Affinity (QGA). In the batch prompting setting, multiple questions are consolidated in a group. For questions in the group, unrelated ones often span distinct contexts or domains, creating noise and making it challenging for LLMs to focus on relevant information and make accurate inferences. Therefore, it is crucial to consider the relationships between questions in batch prompting, as related questions allow LLMs to draw on common patterns which leads to more coherent and accurate responses. Thus, the first factor we consider for enhancing accuracy is the relationship between questions within each group. We define a function aff_q to quantify this relationship for each group, formulated as follows:

$$aff_q(g_l) = \max_{\forall q_i, q_{i'} \in g_l} aff_p(q_i, q_{i'})$$

where $aff_p(q_i, q_{i'})$ represents the affinity between pair-wise questions q_i and $q_{i'}$. To compute the affinity, the first step is to generate representations (i.e., embeddings) for questions and demonstrations, and the second step is to calculate the affinity based on the embeddings. In our framework, we do not restrict the specific implementation method of affinity computation, which is orthogonal to our method. In our implementation, the BERT-based encoders [50] are exploited to tokenize questions into embeddings. Then, the affinity is calculated according to the distance metrics, i.e., Euclidean distance, between the embeddings.

(2) Question Demonstration Affinity (QDA). In the setting of ICL, when given a question, LLMs rely on demonstrations to perform few-shot learning. Therefore, demonstrations that are relevant to the question are crucial because they help LLMs better understand the question and provide accurate inferences. To guide our method to select suitable demonstrations to boost accuracy, we define a function $aff_d(q_i, d_j)$ to measure the relationship between question q_i and demonstration d_j. The process of calculating this affinity is akin to that used for aff_p.

(3) Group Length (GL). To ensure high accuracy of LLMs, the input length cannot be too large. First, LLMs have a token limit, and exceeding this limit can lead to incomplete information, which adversely affects accuracy. Second, overly long inputs make it difficult for LLMs to identify and extract relevant information. Therefore, it is important to consider input length when designing our batch prompting method for accurate responses. In our batch prompting scenario, the input for LLMs is a group containing a task description, multiple questions and demonstrations. We define the function $c(g_l)$ as the total token count of group g_l, so as to constrain the input length of our method.

(4) Demonstration Coverage (DC). As mentioned above, in the ICL setting, LLMs rely on demonstrations to generate accurate responses for questions. If the number of demonstrations is small, LLMs may struggle to comprehensively understand the question. Additionally, a limited number of demonstrations prevent LLMs from capturing

nuances among questions within each group, thereby affecting the reasoning ability of LLMs. Thus, we define a function $cov(d_j)$ for each demonstration d_j in the group to represent questions it can cover. Here, a question covered by a demonstration d_j means that this question is affinitive to d_j. For each demonstration d_j in the group, its number of covered questions, i.e., $|cov(d_j)|$, cannot be too large to ensure sufficient demonstrations for better reasoning ability of LLMs.

After taking into consideration the above four factors, we propose the following constraints to ensure accuracy of our batch prompting method.

- *QGA constraint*: For each group $g_l \in \mathcal{G}$, the questions in g_l have to meet the affinity requirement constrained by the affinity threshold τ_0, i.e., $aff_q(g_l) \leq \tau_0$.
- *QDA constraint*: Under the setting of ICL, each question in the group should have one affinitive demonstration as its context, i.e., $\exists d_j \in \mathcal{D}_l, aff_d(q_i, d_j) \leq \tau_1, \forall q_i \in \mathcal{Q}_l$.
- *GL constraint*: For each group $g_l \in \mathcal{G}$, the total token counts of the group, denoted as $c(g_l)$, cannot exceed the pre-defined maximum length, i.e., $c(g_l) \leq \tau_2$.
- *DC constraint*: For each demonstration d_j, the number of questions it covers cannot exceed a threshold, i.e., $|cov(d_j)| \leq \tau_3$.

To sum up, given a batch $\mathcal{Q}$ of questions and a pool $\mathcal{D}$ of demonstrations, the objective of batch prompting is to group all questions and the corresponding demonstrations into K groups, i.e., $\mathcal{G} = \{g_1, g_2, ...g_l, ..., g_K\}$ where $g_l = \{\mathcal{T}, \mathcal{D}_l, \mathcal{Q}_l\}$, such that the overall cost of all groups is minimized, with the constraints of LLM accuracy satisfied, which is formulated as:

$$\begin{aligned} \min \quad & \sum_{l}^{K} c(g_l) \\ s.t. \quad & aff_q(g_l) \leq \tau_0, && \forall l \in [1, K] \\ & aff_d(q_i, d_j) \leq \tau_1, && \forall q_i \in \mathcal{Q}_l, \exists d_j \in \mathcal{D}_l \\ & c(g_l) \leq \tau_2, && \forall l \in [1, K] \\ & |cov(d_j)| \leq \tau_3, && \forall d_j \in \mathcal{D}_l \end{aligned} \tag{1}$$

2.2 Theoretical Analysis

Theorem 1. *The optimization problem of batch prompting shown in Equation 1 is NP-hard.*

Proof. To prove NP-hardness, we create an instance of this problem by a reduction from a variant of set cover [11] problem, i.e., weighted set cover [16]. 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 cost to cover all questions. For each demonstration d_j, several similar questions are all affinitive to d_j, 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 [32], *batch prompting* is also NP-hard. □

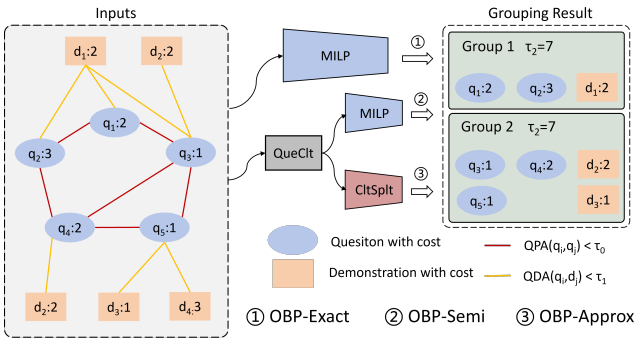


Figure 2: The framework for batch prompting.

3 THE OBP FRAMEWORK

In this section, we present our framework Optimized Batch Prompting (OBP) for adaptive grouping, which is shown in Figure 2. The input of our framework is a batch of questions $\mathcal{Q}$ and a demonstration pool $\mathcal{D}$. The OBP framework offers three versions of solutions. (1) OBP-Exact (Section 3.1): we transform the batch prompting problem into MILP (Mixed Integer Linear Programming) formulation, which can be solved via the MILP solver (e.g., GUROBI [1]) to get the exact solution. Several optimizations are also proposed to accelerate the computations of the MILP solver. (2) OBP-Semi (Section 3.2): Considering the expensive computation of the MILP solver, it is infeasible to have exact solution when the number of questions is large. To this end, we design question clustering method (*QueClt*), which first constructs a graph to represent the question relationship and then decompose it into clusters, so that each cluster can be solved by the MILP solver individually, which is significantly faster than directly solving the original problem. (3) OBP-Approx (Section 3.3): To further improve the efficiency, we design an approximation solution (*CltSplt*) to replace the MILP solver and produce grouping result for the clusters with a larger number of questions.

3.1 The OBP-Exact Solution

In this section, we discuss an MILP-based method, which can produce the exact solution to the batch prompting problem formalized in Section 2. Specifically, we transform the abstract format in Equation 1, such that it can be solved by MILP solvers. Let $x(g_l, i)$ and $y(g_l, j)$ denote a boolean decision variable of whether a question q_i and a demonstration d_j belong to group g_l or not, respectively. The following equation shows the transformation:

$$\min \quad \sum_{l}^{N}(\sum_{i}^{N} c_{q_i} x(g_l, i) + \sum_{j}^{M} c_{d_j} y(g_l, j) + u(l)c_T) \tag{2a}$$

$$s.t. \quad \sum_{l}^{N} x(g_l, i) = 1, \qquad \forall i \in [1, N] \tag{2b}$$

$$aff_q(q_i, q_{i'})x(g_l, i)x(g_l, i') \leq \tau_0, \qquad \forall l, i, i' \in [1, N] \tag{2c}$$

$$\sum_{l}^{N}\sum_{j}^{M}(aff_d(q_i, d_j)x(g_l, i)y(g_l, j)) = 1, \qquad \forall i \in [1, N] \tag{2d}$$

$$\sum_{i}^{N} c_{q_i} x(g_l, i) + \sum_{j}^{M} c_{d_j} y(g_l, j) + c_T \leq \tau_2, \qquad \forall l \in [1, N] \tag{2e}$$

$$u(l) \leq \sum_{i}^{N} x(g_l, i) + \sum_{j}^{M} y(g_l, j), \qquad \forall l \in [1, N] \tag{2f}$$

$$u(l)L \geq \sum_{i}^{N} x(g_l, i) + \sum_{j}^{M} y(g_l, j), \qquad \forall l \in [1, N] \tag{2g}$$

Equation 2a is the objective of our optimization problem, which minimizes the total cost of all groups. Equation 2b constrains that each question can only belong to one group. Equations 2c and 2d are the constraints about the *QGA* and *QDA*. To demonstrate the *QDA constraint*, we can intuitively express as: $\sum_{l}^{N}\sum_{j}^{M} \mathcal{I}(\tau_1 - aff_d(q_i, d_j)x(g_l, i)y(g_l, j)) \geq 1$. 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 $aff_d(q_i, d_j)$, we rewrite it as 1 if it is less than τ_1, 0 otherwise. Thus, the format is shown in Equation 2d. Meanwhile, to maintain the *DC constraint*, for each demonstration d_j, we sort all $aff_d(q_i, d_j)$, and assign the first τ_3 questions as 1 (whose aff_d is less than τ_1), 0 otherwise. While this pre-processing is an approximation method[1], it exhibits excellent accuracy in practice [10]. 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 $aff_p(q_i, q_{i'})$ into 0 or 1. The *GL constraint* is shown in Equation 2e. Note that a total of K groups $(g_1, g_2, ..., g_K)$ 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 2a, 2f and 2g to identify the number of groups. Note that L in Equation 2g is a pre-defined positive number that can be considered as infinity.

Once the MILP is formalized as shown in Equation 2, we exploit a general-purpose solver to solve it (e.g., GUROBI [1]). We take

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

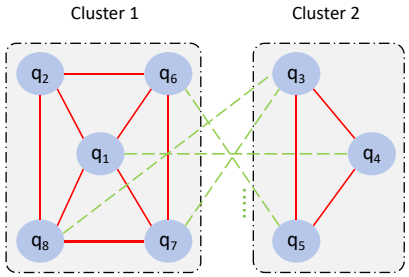


Figure 3: An example of Question Affinity Graph. The red lines denote positive edges while the green lines are negative edges. Note that we only draw partial green edges.

Figure 2 as our example. The inputs consist of five questions and four demonstrations with their cost, e.g., the cost of q_1 is 2. The edge colored red indicates that the two connecting questions are affinitive, i.e., the aff_p between them is less than τ_0; the yellow edge indicates that the question and the corresponding demonstration are affinitive, i.e., the aff_d between them is less than τ_1. 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.

3.2 The OBP-Semi Solution

The optimal solution presented in Section 3.1 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 split the original question set Q into multiple clusters, making each cluster as independent as possible. After the splitting, unrelated questions are not grouped within the same cluster, allowing each cluster to generate groups independently without being affected by others. Then, combining these results gives the final grouping. Specifically, each cluster can formalize an optimization problem without the *QGA constraint* (i.e., the Equation 2c), 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.

Next, we describe the question clustering method named *QueClt*. 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 an undirected (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 $aff_p(q_i, q_{i'})$ between q_i and $q_{i'}$, whose weight is either 1 (i.e., these pair-wise questions are affinitive to each other, denoted as the positive edge) or -1 (otherwise, denoted as the negative edge).*

Figure 3 presents an example *QAG*. The next step is to decompose the graph into subgraphs that are as independent as possible, ensuring that they do not affect each other. 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 [8]. 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 related, and $e_{uv} \in E^-$ if u and v are unrelated. A clustering of G is a non-overlapping partition of its node set. The goal of the 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 nodes 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 [6, 8] for correlation clustering. Among these, the algorithm proposed in [8] has the well-known optimality bound, which can help achieve a feasible solution within a reasonable time frame. We thus select it for implementation. The approximation ratio of it is 3 [8], which bounds deviation from the optimal solution. For example in Figure 3, after running the correlation clustering algorithm, two clusters are produced respectively: $\{q_1, q_2, q_6, q_7, q_8\}$ and $\{q_3, q_4, q_5\}$.

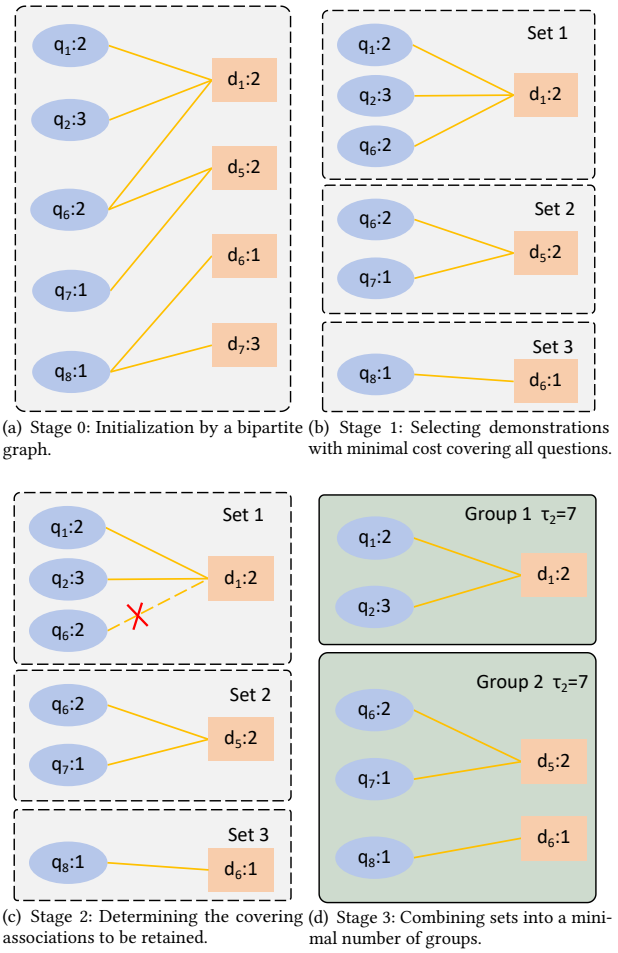
3.3 The OBP-Approx Solution

While we can use the MILP solver to obtain the optimization solution for each cluster, it is still intractable for large clusters with substantial questions. To further accelerate the computation, we design an efficient approximate method Cluster Splitting (*CltSplt*).

We illustrate the procedure of *CltSplt* with an example shown in Figure 4. Specifically, the cluster in the example has five questions and four demonstrations, which can be presented as a bipartite graph (Figure 4(a)). The blue nodes on the left represent the questions and the orange nodes on the right represent the demonstrations. The edge between a question and a demonstration indicates that they are affinitive. Next, we introduce the whole procedure of *CltSplt* based on this example.

Our solution includes three stages: (1) For each cluster, we select affinitive demonstrations with minimal cost to cover all questions. Each demonstration and its associated questions form a set. This problem is an instance of weighted set cover problem[2], so we can use the common-use approximation algorithm [15] to solve it, the approximation ratio of which is $\ln k$ [15]. Note that k indicates the number of questions in the cluster. As the example in Figure 4(b) shows, $\{d_1, d_5, d_6\}$ are the selected demonstrations that can cover all questions with minimal cost, and they together with the associated questions form three sets.

(2) After stage 1, there may be questions that are covered by multiple demonstrations in different sets, e.g., in Figure 4(b), q_6 is covered by both d_1 in Set 1 and d_5 in Set 2. Since one question only belongs to one group in the final grouping, it needs to break the tie by retaining only one connecting demonstration for each question. However, achieving an effective method for high accuracy is not straightforward. A random method may lead to cases where a



(a) Stage 0: Initialization by a bipartite graph.
(b) Stage 1: Selecting demonstrations with minimal cost covering all questions.
(c) Stage 2: Determining the covering associations to be retained.
(d) Stage 3: Combining sets into a minimal number of groups.

Figure 4: Illustration of the *CltSplt* procedure.

demonstration covers a large number of questions. When merging the sets into groups, the group that contains this demonstration has numerous questions but few demonstrations. As discussed in Section 2, this imbalance can affect the reasoning of LLMs. To this end, we propose a metric for retaining the covering associations between demonstrations and questions, and a retention algorithm in Section 3.3.1, with the aim to balance the number of questions covered by each demonstration. For example, in Figure 4(c), q_6 is removed from Set 1 and only retained in Set 2, making the number of questions covered by these demonstrations more balanced;

(3) The final stage is to combine as many sets as possible to generate the the minimum number of groups, while satisfying the *GL constraint*. This is to reduce the overall cost since for every additional group there will be an extra task description cost c_T. In Figure 4(d), three sets are finally combined into 2 groups. We shall elaborate on the combining strategy in Section 3.3.2.

[2]The proof is similar to Section 2.

3.3.1 Retention method. Recall that the retention method is applied when there are questions that are covered by multiple demonstrations in different sets in stage 2 of *CltSplt*. The aim is to retain the covering association that can lead to a high-quality grouping.

However, achieving effective retention to ensure high accuracy is not straightforward. A random retention method could result in one demonstration covering a large number of questions. Note that one set consists of the demonstration and its associated questions. When these sets are merged into groups, it will result in groups with numerous questions but few demonstrations. Such imbalances prevent LLMs from fully comprehending the questions, resulting in a decline in performance. To avoid this situation, we shall balance the number of questions in each set as much as possible while performing the retention. We denote all sets in the cluster after stage 1 as $\mathcal{S}$, where each set $s_i \in \mathcal{S}$ consists of one demonstration and its associated questions, e.g., Set 1 in Figure 4(b) contains $\{d_1\}$ and $\{q_1, q_2, q_6\}$. A direct indicator of the balance level of $\mathcal{S}$ is the maximum number of questions in its all sets. Formally, we define the balance level of $\mathcal{S}$ as:

$$s_{max} = \max_{s_i} |s_i|, \forall s_i \in \mathcal{S} \tag{3}$$

Note that $|s_i|$ denotes the number of questions in set s_i. A smaller s_{max} value implies a more balanced set $\mathcal{S}$. Thus, our goal is to minimize s_{max} when determining which covering association to be retained. Consider the example in Figure 4(c). Removing q_6 from Set 1 results in a smaller s_{max}, which is better than removing q_6 from Set 2. To this end, we propose the following two retention methods.

The first method is a dynamic programming (DP) based algorithm. Specifically, we first sort the sets in $\mathcal{S}$ based on the number of questions, and define O_i as the overlapping questions in the first i sets. An overlapping question means that it appears in multiple sets. Then, we define $DP(O_i)$ as the retention result of the first i sets with minimizing s_{max} as the objective. Thus, the final retention result is $DP(O_n)$, assuming that the total number of sets in $\mathcal{S}$ is n. The recursive formula for dynamic programming can be computed as follows:

$$|DP(O_i)| = \min \begin{cases} |DP(O_i)| \\ \max(|DP(O_{i-1})|, |s_i \setminus o_i^{'}|), o_i^{'} \subset o_i \end{cases} \tag{4}$$

Note that o_i denotes set of the overlapping questions in s_i. We need to calculate $\max(|DP(O_{i-1})|, |s_i \setminus o_i^{'}|)$ for $perm(o_i)$ times and $perm(o_i)$ denotes the number of permutation of o_i. Thus, the time complexity of this DP algorithm is $O(perm(O_i)perm(o_i)n)$.

The computation time is exponential for the number of overlapping questions and the number of sets, making it challenging to obtain the optimal solution for large number of overlapping questions.

To this end, we propose a heuristic method which is shown in Algorithm 1. We begin by sorting $\mathcal{S}$ in descending order based on $|s_i|$. If two sets have the same number of questions, the set with a smaller $|o_i|$ is ranked higher (line 1). Note that $|o_i|$ denotes the number of overlapping questions in the set s_i. The rationale behind this is to prioritize removing overlapping questions from sets with larger $|s_i|$. When the number of questions in the sets (i.e., $|s_i|$) is equal, the set with fewer overlapping questions should be removed first, in order to balance the number of questions across sets. Then, we calculate the average number of non-overlapping questions in each set, denoted as d_{avg} (lines 2-7). Note that the number of questions in the largest set of the optimal solution cannot be less than d_{avg}. Thus, for each set s_i, if the number of questions $|s_i|$ exceeds d_{avg}, we remove its overlapping questions using the function $RemoveWithRestrict(o_i)$ (lines 8-12). To be specific, for a set s_j that has overlapping questions with s_i, if $|s_j| > d_{avg}$, the function $RemoveWithRestrict$ removes $\frac{|c_{ij}|}{2}$ overlapping questions from both s_i and s_j; otherwise, it removes $|c_{ij}|$ overlapping questions from s_i. Note that $|c_{ij}|$ denotes the number of overlapping questions between s_i and s_j. The rationale behind this is to remove more overlapping questions from the set with a larger number of questions. Furthermore, assume that set s_i and h other sets have overlapping questions, denoted as $c_{i1}, c_{i2}, ..., c_{ih}$, we have $\sum_{j=1}^{h} |c_{ij}| = |o_i|$ [3]. Next, we iterate sets whose question count is below d_{avg} and apply $RemoveWithoutRestrict(o_i)$ (lines 13-17). Unlike $RemoveWithRestrict(o_i)$, $RemoveWithoutRestrict(o_i)$ removes $\frac{|c_{ij}|}{2}$ overlapping questions from both s_i and s_j without any restrictions. Finally, the processed sets are returned (line 18). Next, we give a theoretical analysis about this heuristic algorithm.

Algorithm 1 Retention Method

Input: Sets $\mathcal{S}$ after Stage 1.
Output: Retention Sets $\hat{\mathcal{S}}$.

1: $\hat{\mathcal{S}} = Sort(\mathcal{S})$, $Q = Set()$.
2: **for** s_i in $\mathcal{S}$ **do**
3: **for** q_i in s_i **do**
4: $Q.add(q_i)$.
5: **end for**
6: **end for**
7: $d_{avg} = \frac{|Q|}{|\mathcal{S}|}$.
8: **for** s_i in $\hat{\mathcal{S}}$ **do**
9: **while** $|s_i| \geq d_{avg}$ and Overlap(o_i) **do**
10: $RemoveWithRestrict(o_i)$.
11: **end while**
12: **end for**
13: **for** s_i in $\hat{\mathcal{S}}$ **do**
14: **while** $|s_i| \leq d_{avg}$ and Overlap(o_i) **do**
15: $RemoveWithoutRestrict(o_i)$.
16: **end while**
17: **end for**
18: Return $\hat{\mathcal{S}}$.

Theorem 2. $R(\mathcal{S}) \leq OPT(\mathcal{S}) + \frac{|o_{\arg\max(|s_i| - \frac{|o_i|}{2})}|}{2}$

Proof. We use $R(\mathcal{S})$ to denote the number of questions in the largest set of the solution generated by Algorithm 1, and $OPT(\mathcal{S})$ denote the number of questions in the largest set of the optimal solution. We have:

[3] If s_i contains an overlapping question that appears in multiple sets, this overlapping question is counted only once to ensure that the summation formula $\sum_{j=1}^{h} |c_{ij}| = |o_i|$ holds.

$$
\begin{aligned}
R(\mathcal{S}) &\leq \max_{i<n}(|s_i| - \frac{|o_i|}{2}) \\
&= \max_{i<n}(|s_i| - |o_i|) + \max_{i<n}(|s_i| - \frac{|o_i|}{2}) - \max_{i<n}(|s_i| - |o_i|) \\
&\leq OPT(\mathcal{S}) + \max_{i<n}(|s_i| - \frac{|o_i|}{2}) - \max_{i<n}(|s_i| - |o_i|) \\
&\leq OPT(\mathcal{S}) + (|s_{\arg\max(|s_i|-\frac{|o_i|}{2})}| - |\frac{o_{\arg\max(|s_i|-\frac{|o_i|}{2})}}{2}|) \\
&-(|s_{\arg\max(|s_i|-\frac{|o_i|}{2})}| - |o_{\arg\max(|s_i|-\frac{|o_i|}{2})}|) \\
&= OPT(\mathcal{S}) + \frac{|o_{\arg\max(|s_i|-\frac{|o_i|}{2})}|}{2}
\end{aligned} \tag{5}
$$

For the above equation, $R(\mathcal{S})$ is guaranteed to be generated in sets with more than d_{avg} questions, so it can be derived based on the *RemoveWithRestrict* function, i.e., $R(\mathcal{S}) = \max_{i<n}(|s_i| - \sum_{|s_j|\geq d_{avg}} \frac{|c_{ij}|}{2} - \sum_{|s_j|<d_{avg}} |c_{ij}|)$. By amplifying the last term, we have $R(\mathcal{S}) \leq \max_{i<n}(|s_i| - \sum \frac{|c_{ij}|}{2}) = \max_{i<n}(|s_i| - \frac{|o_i|}{2})$. Additionally, we have $\max_{i<n}(|s_i| - |o_i|) \leq OPT(\mathcal{S})$. This is because $OPT(\mathcal{S})$ does not exceed the maximum value of these sets after all overlapping questions have been removed from each set.

□

Given the two proposed retention methods, how to select the appropriate one is determined by the number of overlapping questions. When the number of overlapping questions is large, the heuristic method is efficient. Otherwise, the DP-based method is better, which yields more effective results. Empirical results indicate that when the number of overlapping questions exceeds 180, the runtime of the DP-based method exceeds one hour.

3.3.2 Packing demonstrations and questions. In stage 3, we combine sets produced in stage 2 into groups with the objectives to minimize the number of groups and at the same time satisfy the *GL constraint*. This problem can be reduced to the well-known bin packing problem (BPP) [30], in which items of different sizes must be packed into a finite number of bins, each with a fixed given capacity. The goal is to minimize 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, we utilize the first-fit bin packing algorithm [22] to implement it, which guarantees a bounded deviation from the optimal solution. The approximation ratio of this approximation algorithm is 1.7 [22].

3.4 Time Complexity of the Approximation Method in OBP

In this section, we analyze the time complexity for the approximation method in OBP. Specifically, our approximation method contains two steps, namely, *QueClt* and *CltSplt*.

For the *QueClt* step, given N questions, it runs in time $O(NC)$, i.e., the runtime for the approximation algorithm of correlation clustering [6]. Note that C is the number of generated clusters.

After the *QueClt* step, for each cluster with k questions and m demonstrations, it is input into the *CltSplt* step. This step involves

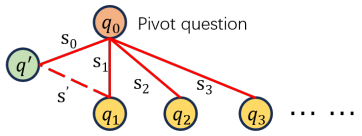


Figure 5: An illustration for Lemma 2.

three stages. Stage 1 leverages weighted set cover to select demonstrations. Note that each demonstration corresponds to one set, and the total number of elements equals the number of questions in the cluster, i.e. k. Thus, its time complexity is $O(mk)$, i.e., the runtime for approximate set cover algorithm [15]. Note that n sets are produced by Stage 1, and subsequently are input into Stage 2. Stage 2 includes a heuristic algorithm (i.e., Algorithm 1) to balance the number of questions between sets. This algorithm first sorts n sets, which takes $O(nlogn)$ time. Then, it iterates over all n sets. For each set, it removes $|o_i|$ questions. Therefore, the runtime for this step is $O(n|o_i|)$. As a result, the overall time complexity of Algorithm 1 is $O(nlogn + n|o_i|)$. Then, the sets without overlapping questions are input into Stage 3. Stage 3 exploits the first-fit bin packing [22] to complete the final grouping, whose runtime is $O(nlogn)$. Thus, the time complexity of the *CltSplt* step is $O(mk + n|o_i| + nlogn)$ for each cluster.

In conclusion, the time complexity of the overall algorithm is $O(NC + (mk + n|o_i| + nlogn)C)$.

3.5 Optimization for Efficient Computation

In this section, we propose two optimization strategies to further speed up the computation.

3.5.1 Pruning ineffective demonstrations . The demonstration pool may comprise a large number of demonstrations, causing expensive computations. Intuitively, we can prune the demonstrations that are not affinitive to any question in the batch $\boldsymbol{Q}$. Let the remaining demonstrations be referred as *valid* demonstrations. Meanwhile, we can rule out those valid demonstrations that are *dominated*, as given in the Lemma 1 below. Here, we define that a demonstration d_a is dominated by a demonstration d_b (denoted as $d_a \prec d_b$), if $cov(d_a) \subset cov(d_b)$ and $c_{d_a} \geq c_{d_b}$.

Lemma 1. *Pruning Dominated Demonstrations. Given two valid demonstrations d_a and d_b, d_a can be safely pruned if $d_a \prec d_b$ holds.*

Proof. Given that $cov(d_a) \subset cov(d_b)$ and $c_{d_a} \geq c_{d_b}$, if d_a is selected in certain group, d_a can always be replaced with d_b without violating the constraints in Equation 1 or increasing the overall cost. □

3.5.2 Reducing affinity computations. Our method requires pair-wise aff_p computations for questions, and pair-wise aff_d computations for questions and demonstrations, which is time-consuming especially when the affinity calculation is performed over high-dimensional embeddings. In the following, we propose an optimization method to reduce the number of computations by means of triangle 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 be 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 most common metrics, e.g., Euclidean distance and Cosine similarity, all satisfy this property. Through the computation of aff_p, for each question q_i, we need to obtain questions whose affinity with q_i is below τ_0, denoted by $N_{\tau_0}(q_i)$. We have the following lemma:

Lemma 2. *For any two questions* $q_i, q_j \in \mathcal{Q}$, *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 2 to reduce some computations for aff_p. Specifically, in Lemma 2, questions q_i, q_j and q_r form a triangle. We denote q_r as the pivot question and assume that $dist(q_i,q_r)$ and $dist(q_j,q_r)$ have been calculated. To calculate the affinity between q_i and q_j, if $dist(q_i,q_r) - dist(q_j,q_r) > \tau_0$, we can directly conclude that $dist(q_i,q_j) > \tau_0$ without further affinity computations between q_i and q_j. To illustrate the rationale of Lemma 2 and Algorithm 2, we give an example shown in the Figure 5. Assume that we have calculated the affinity between the pivot question q_0 and all other questions, and sort them by the affinity value, i.e., $s_1 < s_2 < s_3$. When calculating the affinity between a new question q' and other questions, we can reduce the number of calculations by means of Lemma 2. Specifically, to determine whether the affinity between q' and q_1 is less than τ_0 (i.e., whether q_1 belongs to the set $N_{\tau_0}(q')$[4]), we exploit the triangle inequality property among the pivot question q_0, questions q' and q_1 (i.e., $s' > s_1 - s_0$). If $s_1 - s_0 > \tau_0$, we can directly determine that the affinity value between q_1 and q' is greater than τ_0 (i.e., $s' > \tau_0$), so no further affinity calculation between q' and q_1 is needed. Subsequently, for questions q_2 and q_3, they both can form triangles with q_0 and q', if $s_2 - s_0 > \tau_0$ and $s_3 - s_0 > \tau_0$, we can avoid calculating the affinity between q' and q_2, q' and q_3. Furthermore, if $s_1 - s_0 > \tau_0$, we can directly conclude that $s_2 - s_0 > \tau_0$ and $s_3 - s_0 > \tau_0$, because $s_1 < s_2 < s_3$. We propose an efficient method, which is shown in Algorithm 2. First, we select a pivot question q (e.g., the first question), and then calculate the affinity between q and all other questions (line 1). For each question q_i except q, we use the Lemma 2 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 aff_d, the procedure is similar (lines 7-12).

Algorithm 2 RAC

Input: A pivot question q, a batch of questions $\mathcal{Q}$, all demonstrations $\mathcal{D}$, threshold τ_0 and τ_1.
Output: aff_p and aff_d.
1: Calculate $aff_p(q,\_)$=$aff_p(q,q_i), \forall q_i \in \mathcal{Q}$ and sort them.
2: **for** q_i in $\mathcal{Q}$ **do**
3: s=BinarySearch($aff_p(q,\_)$, $aff_p(q,q_i) + \tau_0, >$).
4: e=BinarySearch($aff_p(q,\_)$, $aff_p(q,q_i) - \tau_0, <$).
5: aff_p=$aff_p(\mathcal{Q}^e_s, q_i)$.
6: **end for**
7: Calculate $aff_d(q,\_)$ =$aff_d(q,d_j), \forall d_j \in \mathcal{D}$ and sort them.
8: **for** d_j in $\mathcal{D}$ **do**
9: s=BinarySearch($aff_d(q,\_)$,$aff_d(q,d_j) + \tau_1, >$)
10: e=BinarySearch($aff_d(q,\_)$,$aff_d(q,d_j) - \tau_1, <$)
11: aff_d=$aff_d(\mathcal{D}^e_s, d_j)$
12: **end for**
13: Return aff_p and aff_d.

4 EVALUATION

In this section, we present the experimental evaluation of our method 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, GPT-4-Turbo of OpenAI [4], Claude 3 Opus of Anthropic [3] and Qwen-Long of Aliyun [2] as our backend LLMs. Meanwhile, our MILP solver is GUROBI [1]. For simplicity, we use OBP to represent the approximation method OBP-Approx when the context is clear.

[4] $N_{\tau_0}(q')$ denotes questions whose affinity with q' is below τ_0.

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
DG	AssertTrue (ATr)	179	44971	40/936
	AssertNotNull (ANN)	329	87635	22/1370
	AssertEquals (AE)	301	74108	54/887
	AssertThat (ATh)	75	18696	62/856

Datasets. We evaluate our OBP framework using three representative tasks in data management, i.e., entity resolution (ER), data transformation (DT) and data generation (DG). In the following, we detail the corresponding datasets and Table 2 presents the detailed statistics of these datasets.

(1) Entity Resolution (ER). We use the well-adopted datasets from Magellan [20], 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 according to the existing ER studies [43, 53]. Each sample has two items with attributes and corresponding values. In this task, LLMs are used 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 [18, 35]. We use the datasets introduced by [5, 62], which are collected from real-world tables. Each sample contains the source format and the target format, e.g., "Norm Adams" and "n.adams". In this task, LLMs are used to generate the target format given the source format.

(3) Data Generation (DG). We use the dataset from ATLAS [55] to evaluate our method, which is to generate code assertions for JAVA code. Each sample contains the context of a unit test followed by the instruction, and the LLMs need to generate a single assertion for the given test method. In this paper, we 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.

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

LLM-based baselines: We compare with 3 LLM-based methods.

(1) Batcher [24]. 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 a group (in their experiments, all the group sizes are set to 8 by default). After generating groups, they leverage set cover to select demonstrations for each group.

(2) 1demo [24]. In this baseline, they use the same method in Batcher to group questions. Then, they select the most similar demonstration for each question in the group. If the most similar demonstration of a certain question has already been selected, they choose the next most similar demonstration. Thus, the number of questions is the same as the number of demonstrations in each group.

(3) Single [47]. This method asks LLMs to resolve one question at a time, with one demonstration that is most similar to the question.

Non-LLM Baselines. We compare against the non-LLM SOTA methods for each task.

(1) For entity resolution, we compare against Ditto [43], the current SOTA deep learning-based approach which finetunes BERT [19].

(2) For data transformation, we compare against DTT [18], a transformer based example-driven solution.

(3) For data generation, we compare against ATLAS [55], a neural machine translation-based approach.

Metrics. We conduct the evaluation 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 DG. For ER, we use $F1$ score to measure the accuracy of an approach, which is a common metric for existing ER works [24, 43, 53]. 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 DG, we use exact match accuracy, which denotes the percentage of numbers where the predicted output matches lexically with the expected output.

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

Table 3: Comparison between OBP and baselines on 14 datasets. The LLM-based results are under GPT-3.5-Turbo. The unit of Accuracy is %, and the Cost refers to token counts (k). Note that the best results are bolded and the second best results are underlined.

Dataset	OBP		Batcher		1demo		Single		non-LLM
	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc
FZ	**95.2**	**23.7**	**95.2**	30.2	90.5	38.2	90.0	53.2	91.7
Beer	**92.3**	**9.5**	88.9	11.9	85.7	14.2	88.0	20.3	90.4
iA	**94.1**	**7.9**	92.3	10.4	92.3	11.8	87.7	19.4	82.1
DA	93.0	**400.3**	**93.5**	522.7	93.0	612.5	91.3	766.9	87.5
WA	**83.7**	**223.7**	82.8	280.3	83.0	341.4	75.8	475.2	74.8
AB	**82.0**	**127.2**	79.5	171.2	79.4	208.0	80.7	323.1	73.1
AG	57.8	**175.5**	57.4	193.7	**59.4**	231.9	49.4	346.5	57.8
DS	**88.5**	**471.8**	78.5	501.6	77.2	588.5	78.3	778.7	85.7
WTS	**87.7**	**22.7**	87.6	32.7	83.7	51.9	82.1	136.5	61.3
KBWT	**97.2**	**2.2**	92.8	3.4	96.6	5.1	91.1	17.5	33.3
ATr	**88.8**	**87.4**	78.3	91.3	76.7	107.8	82.7	112.6	72.7
ANN	**92.1**	**174.6**	78.7	190.6	91.5	206.7	82.6	225.3	90.8
AE	**87.7**	**117.6**	79.7	133.8	87.4	175.2	79.4	176.8	73.2
ATh	**81.3**	**25.0**	61.3	35.6	74.7	44.7	78.7	46.4	63.9

4.2 Overall Efficacy Comparison

In this section, we compare our OBP framework with both LLM-based baselines and non-LLM SOTAs on 14 real-world datasets.

4.2.1 Comparison between OBP and LLM-based baselines. The comparison results are shown in Table 3. For Batcher, we use the default group size 8 in ER and DT, while the group size in DG is 2, which performs the best. Furthermore, when computing aff_q, we use the reciprocal of the Euclidean distance as the affinity metric for the ER task, which is consistent with Batcher [24]. Meanwhile, we use the Euclidean distance as the affinity metric for DT and DG tasks, as it yields the best performance.

Cost. Table 3 shows that our method incurs the least cost over all 14 datasets. Compared with Batcher, OBP can reduce the cost by up to 35% (i.e., KBWT). The cost of Batcher is the local optimum, and our method OBP can reach the global optimum approximately. To be specific, our method can adaptively adjust the number of groups and the number of questions per group based on data. Except for OBP, Batcher outperforms 1demo on cost. This is because for each group, the number of demonstrations of Batcher may be less than the number of questions due to demonstration sharing, and 1demo selects a fixed number of demonstrations, which is the same as the number of questions in the group.

Meanwhile, single prompting method (i.e., Single) incurs higher cost than batch prompting methods. Even for 1demo, the batch prompting method with the highest cost, it can reduce the cost by up to 30% compared to Single. While the demonstration number of Single and 1demo are the same (each question corresponds to

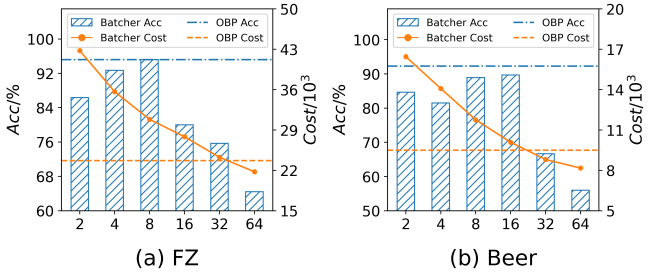


Figure 6: Comparison between OBP and Batcher with different group sizes on the entity resolution task. The group sizes of Batcher are varying from 2 to 64.

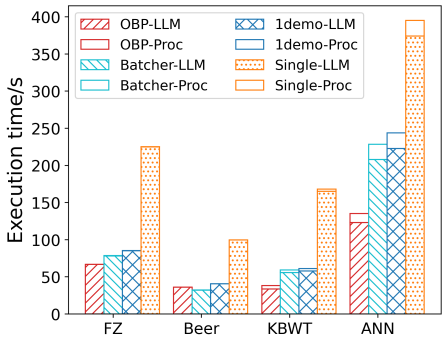


Figure 7: Evaluation on execution time.

a demonstration), Single treats each question as a group to query LLMs, resulting in higher task description cost.

Accuracy. From Table 3, we can observe that OBP performs the best for almost all datasets, and only performs the second best with only a slight difference in 2 datasets out of 14 datasets under GPT-3.5-Turbo, which indicates that our method can produce higher-quality results. Batcher and 1demo both use a fixed group size, which may result in some questions being put in the same group inappropriately. In contrast, our approach is able to perform adaptive grouping, leading to more accurate results.

Additionally, we observe that batch prompting methods (i.e., OBP, Batcher and 1demo) outperform single prompting method (i.e., Single), which validates the idea of batch prompting in leveraging LLMs for data management. It not only significantly reduces the cost, but also improves the result accuracy since more related context is provided in a batch to facilitate the reasoning ability of LLMs.

4.2.2 Comparison between OBP and non-LLM SOTAs. In this section, we compare OBP with non-LLM SOTA methods of each task, and the results are shown in the "non-LLM" column of Table 3. The results show that LLM-based methods outperform non-LLM SOTA methods on almost all datasets, and a significant performance gap can be observed on some datasets. For example, OBP under GPT-3.5-Turbo achieves 97.2% accuracy on dataset KBWT whereas the non-LLM SOTA only reaches 33.3%. We find that non-LLM baselines generally perform well on some datasets in simple tasks (e.g., the dataset FZ in entity resolution) but performs poorly on more challenging ones (e.g., the dataset KBWT in data transformation). Additionally, the LLM-based method is few-shot, requiring little training data (i.e., demonstrations).

4.2.3 Comparison between OBP and Batcher with different group sizes. Batcher clusters questions with a fixed group size (i.e., the number of questions in each group is constant). The default group size is suggested to be 8 [24]. In this section, we compare OBP with Batcher with group sizes varying from 2 to 64 to study the robustness of our method. We conduct experiments on entity resolution, which is the target task of Batcher. Due to space constraint, Figure 6 only shows the results on FZ and Beer datasets. We observe similar results on other datasets. We find that OBP consistently outperforms Batcher in terms of accuracy regardless of the group size, demonstrating its superior performance. Meanwhile, OBP incurs relatively low cost. For Batcher, we can observe that group size significantly affects the accuracy, and the optimal group size varies for different datasets. For instance, the optimal group size of dataset FZ is 8. But for dataset Beer, it is 16. Additionally, we find that the accuracy drops when the group size is set too large. For example, the accuracy of dataset Beer drops from 89% to 67% when the group size changes from 16 to 32, and the accuracy is only 56% when group size is set to 64. This is because when the group size is too large, one group accommodates more questions, leading to some questions improperly grouped together, negatively affecting the reasoning of LLMs. When the group size is set too small, the proliferation of groups incurs a significantly high cost. For instance in dataset Beer, the cost increases by 70% when the group size is changed from 16 to 2. In summary, OBP can adaptively cluster affinitive questions in one group, enhancing the overall accuracy while ensuring low cost.

4.3 Evaluation on Execution Time

In this section, we compare the end-to-end execution time of OBP against the LLM-based baselines. In Figure 7, we present the processing time and the time of calling LLMs on four representative datasets from the three data management tasks. It can be seen that the processing time is negligible compared to the time of calling LLMs. However, we find that the processing time in the dataset ANN is relatively long, primarily due to the time-consuming embedding computations in this dataset. Meanwhile, we find that our method requires less LLM calling time (except for the comparison with Batch on Beer dataset). The reasons are two fold. 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 related questions and sufficient demonstrations to facilitate the reasoning of LLMs, thereby saving the inference time. We observe that OBP spends more time calling LLMs compared to Batcher on the Beer dataset. This is because our method generates more groups than Batcher (17 vs 12).

4.4 Tuning and Sensitivity Analysis of Hyperparameters

Our framework involves four hyperparameters: $\tau_0, \tau_1, \tau_2, \tau_3$. In this section, we first present the guidance for tuning these hyperparameters, followed by a comprehensive sensitivity analysis for each one.

Hyperparameter τ_0. It measures the affinity between questions. Thus, τ_0 needs to distinguish relevant and irrelevant questions, ensuring that questions within the same group are affinitive to each other. We choose the appropriate value of τ_0 in three steps: the first step is to calculate the affinity values[5] of each question with all other questions and sort them in ascending order, noting that smaller values indicating higher relevance. The second step is to identify the affinity gap for each question (see Figure 8(a) and 9(a)), which is the largest range between adjacent affinity values. Intuitively, the gap signifies a cutoff value below which questions are affinitive to the given question, and above which questions are mostly unrelated to the given question. We denote the cutoff value of the question as the endpoint of the gap. The third step is to determine a cutoff region among the cutoff values of all questions, which is an interval that encompasses most cutoff values. Finally, we can either set τ_0 as the endpoint of the cutoff region to reduce cost or tune τ_0 within the region for higher accuracy. See the example below for more details.

Figure 8(a) shows the distribution of sorted affinity values for a sample question, whose id is 200 from the ANN dataset. Note that the x-axis represents the most affinitive questions to the sample question in ascending order. As observed, we can derive the affinity gap [0.440, 0.447] and the cutoff value 0.447 from the figure. After obtaining all cutoff values for each question, we plot them as depicted in Figure 8(b). As observed, most of the cutoff values lie between 0.4 and 0.5, i.e., the cutoff region. Consequently, we set τ_0 as 0.5. We can also tune τ_0 within [0.4, 0.5] for better accuracy. This way, the search space for hyperparameter tuning can be significantly reduced. Similar guidance on determining τ_0 is also viable on other datasets, e.g., the KBWT dataset in Figure 9.

We further conduct a sensitivity analysis of τ_0, as shown in Figure 10. From the figure, we observe that τ_0 performs the best in terms of accuracy within the cutoff region. For ANN dataset, the optimal accuracy is achieved when τ_0 lies within the cutoff region, i.e., between 0.4 and 0.5. Similarly, in the KBWT dataset, the optimal accuracy falls within the cutoff region, i.e. ranging from 0.06 to 0.15. Furthermore, we observe that τ_0 is less sensitive to accuracy within this region. For values not belonging to this region, the accuracy shows a significant downgrade. Meanwhile, we observe that the cost decreases as τ_0 increases. This is because a larger τ_0 implies more questions per group, resulting in fewer groups and thus decreasing the task description cost. For a new dataset, users can first identify the appropriate range of τ_0 values based on the above guidance. If the aim is to reduce cost, they can choose the maximum value within the cutoff region as τ_0. Otherwise, they can tune τ_0 within this region, ultimately choosing the one that achieves the highest accuracy. In our implementation, we set τ_0 to the maximum value within the cutoff region to reduce cost.

[5]Please refer to Section 2 of the revised manuscript for the definitions.

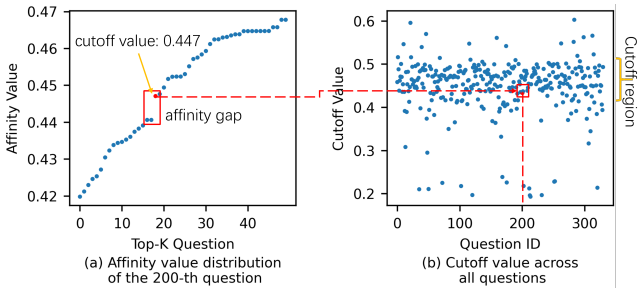


Figure 8: Affinity distribution of hyperparameter τ_0 on the ANN dataset.

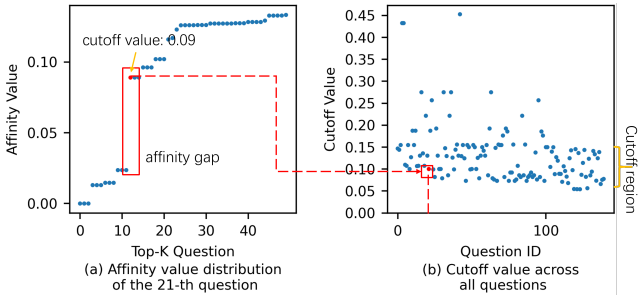


Figure 9: Affinity distribution of hyperparameter τ_0 on the KBWT dataset.

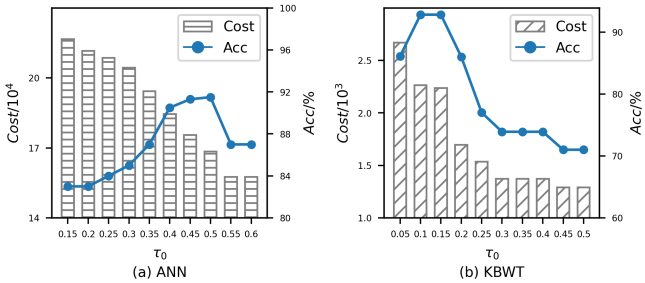


Figure 10: Sensitivity analysis of hyperparameter τ_0.

Hyperparameter τ_1. It measures the relationship between questions and demonstrations. Thus, τ_1 is used to distinguish between questions that are relevant and irrelevant to a demonstration. Similar to τ_0, we identify the appropriate value for τ_1 in the following three steps: the first step is to calculate the affinity values of each demonstration with all questions and sort them in ascending order. The second step is to identify the affinity gap and cutoff value within the first few affinity values, as a demonstration cannot cover too many questions. The third step is to determine the cutoff region of τ_1 by inspecting cutoff values across all demonstrations. Finally, we set τ_1 as the endpoint of the cutoff region to reduce cost or tune τ_1 within the region for higher accuracy.

Figure 11(a) and Figure 12(a) illustrate the distribution of sorted affinity values for a sample demonstration from the ANN and KBWT datasets respectively. Furthermore, Figure 11(b) and Figure 12(b) present cutoff values across all demonstrations. From these figures,

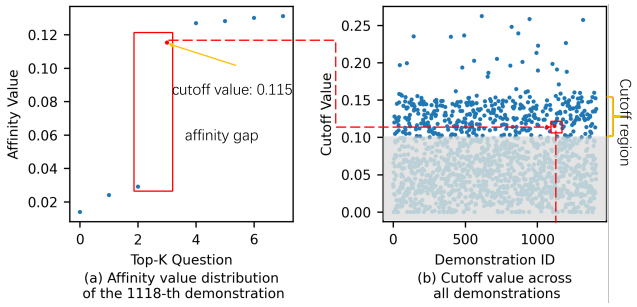


Figure 11: Affinity distribution of hyperparameter τ_1 on the ANN dataset.

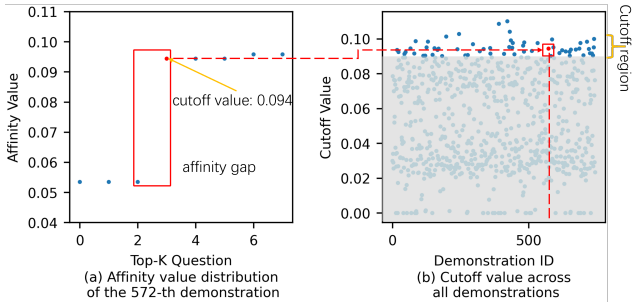


Figure 12: Affinity distribution of hyperparameter τ_1 on the KBWT dataset.

we observe that the cutoff values in the ANN dataset typically fall below 0.16, while those in the KBWT dataset generally fall below 0.1. Note that when τ_1 becomes particularly small, it leads to a lack of relevant demonstrations, resulting in no feasible solution (i.e., the grey area in the figures). For example, in the ANN dataset, when τ_1 drops below 0.1, the algorithm fails to return a result. Similarly, in Figure 11(b), it is the region below 0.09. Thus, we denote the cutoff region for the ANN dataset as [0.1, 0.16], and for the KBWT dataset as [0.09, 0.1].

We further perform a sensitivity analysis on τ_1, and the results are shown in Figure 13. We find that the best performance is achieved when τ_1 is set to values within the cutoff region. For instance, in the ANN dataset, when τ_1 is within [0.1, 0.16] the corresponding accuracy significantly outperforms that of other values. Similarly, in the KBWT dataset, when τ_1 is within [0.09, 0.1], the accuracy surpasses that of other values. In terms of cost, we find that cost decreases as τ_1 increases. This is because when τ_1 becomes larger, each demonstration can cover more questions, so fewer demonstrations are needed to cover all the questions within a group, leading to decreased cost. For a new dataset, we can achieve reduced cost if we select the maximum value within the cutoff region as τ_1. Otherwise, we can tune τ_1 within the cutoff region, selecting the one that achieves higher accuracy. Again, the search space is significantly reduced, making fine-grained tuning affordable. In our implementation, we use the maximum value within the cutoff region as τ_1 in order to reduce cost.

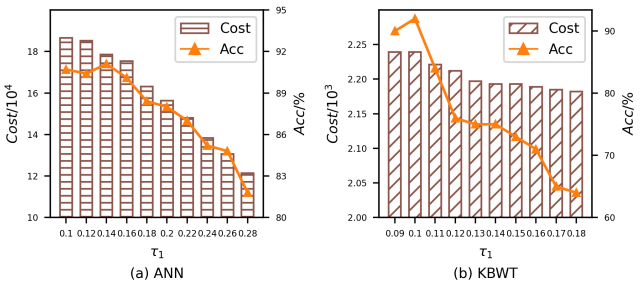


Figure 13: Sensitivity analysis of hyperparameter τ_1.

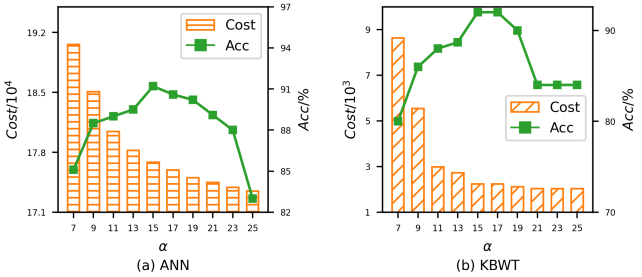


Figure 14: Sensitivity analysis of hyperparameter τ_2.

Hyperparameter τ_2. It constrains the group length, which directly affects the group size. In our setting, the group size includes both questions and demonstrations. Considering that the length of questions and demonstrations varies across tasks, we model τ_2 as $\alpha \times q_{len}$, where q_{len} is the average question length for each task, capturing task-specific characteristics. Then, we vary α to examine its impact on accuracy. The results are shown in Figure 14. From the figure, we find that when the value of α is around 15, it performs the best. For example, the average question length in the ANN dataset is 200, and the optimal τ_2 value is around 3000. Note that similar results are also observed on other datasets, e.g., the KBWT dataset in Figure 14(b). These empirically demonstrate that the optimal α value is generally applicable across datasets. In terms of cost, we observe that when τ_2 becomes larger, the cost decreases. This is because a group can accommodate more questions as τ_2 becomes larger, thereby reducing the number of groups. For a new dataset, if users focus on cost, they can adjust τ_2 starting from $\alpha = 15$ and gradually increase it. Otherwise, they can tune τ_2 around $\alpha = 15$ and selecting the value that yields the best accuracy. In our implementation, we choose the first option to reduce cost.

Hyperparameter τ_3. It limits the maximum number of questions that each demonstration can cover. This hyperparameter is typically set to a relatively small value. If it is set too large, it may lead to groups with many questions but few demonstrations, which negatively affects the reasoning capabilities of LLMs. The corresponding evaluation is shown in Figure 15. We observe that τ_3 performs the best when it is set to 3 or 4. Similar results are observed on other datasets. Taking cost into consideration, users can choose 4 as the value for τ_3. This is because when τ_3 is larger, one demonstration can cover more questions, thereby reducing the overall cost.

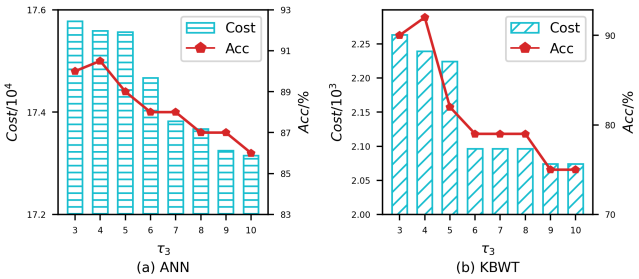


Figure 15: Sensitivity analysis of hyperparameter τ_3.

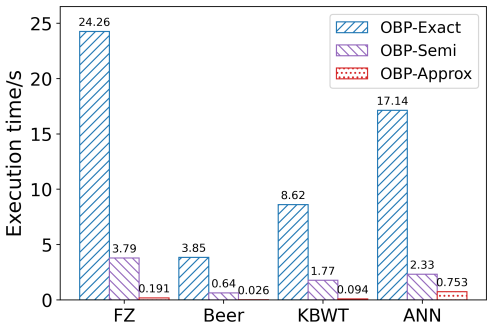


Figure 16: Evaluation on execution time of the three OBP variants.

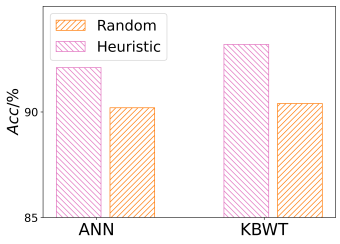


Figure 17: Comparison between random determination and our heuristic method.

4.5 Cost-Accuracy Trade-off Analysis

In this section, we analyze the cost-accuracy trade-off under varying hyperparameters, with the relevant results shown in Figures 10, 13, 14 and 15.

For τ_0, Figure 10 illustrates the cost-accuracy trade-off as τ_0 varies. Regarding cost, it decreases as τ_0 increases. This is because a larger τ_0 implies more questions per group, resulting in fewer groups and thus decreasing the task description cost. As for accuracy, it first increases and then decreases as τ_0 increases. When τ_0 is too small, most groups contain fewer demonstrations, which can impact the reasoning capability of LLMs. Conversely, when τ_0 is very large, the questions within each group tend to be randomly selected, leading to poor performance.

For τ_1, Figure 13 presents the cost-accuracy trade-off as τ_1 varies. Regarding cost, it decreases as τ_1 increases. This is because when τ_1 is larger, each demonstration may cover more questions, so fewer demonstrations are needed to cover all the questions within a group. Additionally, as τ_1 increases, the accuracy deteriorates. This is because the selected demonstration may differ significantly from the questions with a larger τ_1, which do not effectively assist in the reasoning of LLMs.

For τ_2, the corresponding trade-off evaluations are shown in Figure 14. Regarding cost, it decreases when τ_2 becomes large. This is because a group can accommodate more questions as τ_2 becomes larger, thereby reducing the number of groups, i.e., reducing the task description cost. As for accuracy, it first increases then decreases as τ_2 increases. For smaller τ_2, each group can accommodate fewer demonstrations, which can affect the reasoning capability of LLMs. When τ_2 becomes larger, the LLM performance tends to degrade due to the larger number of questions within each group.

For τ_3, the trade-off results are shown in Figure 15. For cost, when τ_3 is larger, one demonstration may cover more questions, causing fewer costs. As for accuracy, when τ_3 is relatively large, it becomes worse. This is because one group may have fewer demonstrations, resulting in LLMs acquiring less domain knowledge. As a result, the LLMs are more likely to reason incorrectly for questions.

4.6 Comparison of the Random and Heuristic Retention Methods

In this section, we compare the random retention approach with the heuristic method (i.e., Algorithm 1), the results of which are presented in Figure 17. Note that the random retention method refers to randomly removing overlapping questions from the sets. As shown in the figure, our method outperforms the random method, achieving the highest accuracy on both the ANN and KBWT datasets. For example, in the ANN dataset, our method achieves 92.1% in terms of accuracy, while the random method only achieves 90.2%. This is because the groups generated by the random method often have many questions but very few demonstrations, whereas in our method, the number of questions and demonstrations in each group are more balanced, improving the reasoning capability of LLMs. This demonstrates the effectiveness of our approach.

4.7 Ablation study on OBP

Recall that the OBP framework offers three versions of solutions, namely OBP-Exact, OBP-Semi and OBP-Approx. In this section, we conduct evaluation on the three OBP variants in terms of effectiveness and efficiency.

The comparison results on the execution time are shown in Figure 16. Note that after correlation clustering, the cluster splitting methods for each cluster can be executed concurrently. So the execution time for OBP-Semi and OBP-Approx in Figure 16 refers to the slowest time among all clusters. From the results, we can see that compared to OBP-Exact, the execution time of OBP-Semi can be reduced by around 85%. Take the dataset FZ as an example, it takes 24.26s with OBP-Exact, but only 3.79s with OBP-Semi. Generally, given n questions, OBP-Semi can eliminate n boolean variables and $O(n^2)$ constraints in Equation 2, so as to accelerate computation with the MILP solver. Furthermore, OBP-Approx, which replaces the computationally expensive MILP solver with our proposed approximation algorithm, achieves the shortest execution time. For

Table 4: Evaluation on cost and accuracy of the three OBP variants. Note that cost refers to token counts (k), and the best results are bolded.

Dataset	OBP-Exact		OBP-Semi		OBP-Approx	
	Acc (%)	Cost (k)	Acc (%)	Cost (k)	Acc (%)	Cost (k)
FZ	**94.1**	**12.8**	**94.1**	12.9	**94.1**	13.0
Beer	**91.4**	**24.4**	91.3	24.6	91.2	24.6
KBWT	**97.0**	**8.6**	96.3	8.8	**97.0**	8.8
ANN	**97.6**	**6.1**	96.5	6.3	96.0	6.4

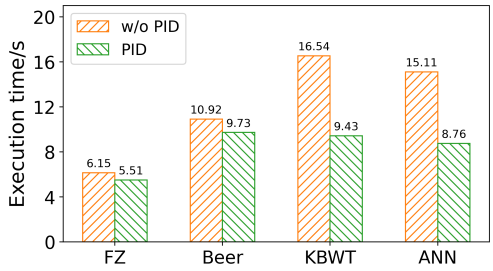


Figure 18: Evaluation on PID.

example, the runtime of OBP-Approx is only 5% of OBP-Semi on the dataset FZ.

We summarize the comparison results on cost and accuracy of the three OBP variants in Table 4. From the results, we observe that OBP-Exact achieves the best performance for all datasets, due to its optimal exact solution. However, it incurs a severe execution delay as shown in Figure 16, making it impractical for real-world applications. On the contrary, OBP-Approx is efficient, and achieves approximately optimal performance in terms of accuracy and cost. Specifically, compared with OBP-Exact, the accuracy of OBP-Approx only decreases by up to 1.6%, and the increase of cost is nearly negligible. In summary, OBP-Approx is effective and efficient for LLM batch prompting in data management.

4.8 Evaluation on Computation Optimizations

In Section 3.5, we propose two pruning strategies to reduce unnecessary computations, namely Pruning Ineffective Demonstrations (PID) and Reducing Affinity Computations (RAC). In this section, we evaluate these two optimizations.

4.8.1 PID evaluation. The demonstration pool comprises large numbers of demonstrations, causing expensive computations. In section 3.5.1, we propose PID to reduce the number of demonstrations. In Figure 18, we compare the execution time with and without PID. The execution time here refers to the computation time for batch prompting, excluding the time of calling LLMs. We can observe that PID can reduce computations by up to 42%. This is because PID filters dominated demonstrations, causing Equation 2 to have fewer variables. In summary, PID can effectively filter out unnecessary demonstrations, thereby enhancing efficiency.

4.8.2 RAC evaluation. In our OBP framework, we need to first generate embeddings for each question and demonstration, and then calculate the aff_p and aff_d. For high-dimensional embeddings, these computations are significantly slow [54], so we propose a pruning strategy called RAC in Section 3.5.2 to reduce the number of computations. In this section, we evaluate the efficiency of RAC and present the experimental results in Figure 19.

In the figure, "Ratio" shows the percentage of computation reduction after applying RAC. Additionally, since RAC is influenced by the threshold τ_0 and τ_1, we thus evaluate the efficiency with respect to different values of τ_0 and τ_1. From the figure, 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 of aff_p and aff_d are below the thresholds, so most of the values satisfy the Lemma 2. The reduction ratio decreases as the threshold increases. When the threshold is larger than 0.3, the ratio drops sharply. This is because the number of calculations for aff_p and aff_d below the thresholds is increasing. In our experiments, we set τ_0 as the top 25% aff_p quantile values, and set τ_1 as top 10% aff_d quantile values. In conclusion, our optimization method helps reduce the computations of both aff_p and aff_d.

5 RELATED WORK

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

LLMs prompting. The main trend of using LLMs is through prompting [9, 56]. This approach has changed the research paradigm. 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 [57] and soft prompt [58] respectively. Hard prompt means the same context is used for all questions, such as the task description and fixed demonstrations. Soft prompt typically refers to selecting or generating prompts dynamically with respect to the questions. For example, recent works try to retrieve demonstrations similar to the question for improving accuracy [24, 28, 47]. 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. Recent works [26, 45] start to explore the possibility of applying LLMs in data management tasks, for example, entity resolution [45], data transformation [36], data generation [47], and so on. While LLMs demonstrate promising performance on these tasks, there are still some challenges. For example, for some data management tasks, they tend to process a large number of questions, which can be very costly for LLMs. It therefore calls for developing a cost-effective method. In this paper, we propose optimizations to reduce cost for LLMs through batching questions and demonstrations without compromising accuracy. We next elaborate more on three representative data management applications: entity resolution, data transformation and data 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 [25, 52], crowd-based methods [12, 13, 23, 41] and deep learning-based methods [43, 53]. Recently, some researchers have proposed to exploit LLMs to resolve the entity resolution [45].

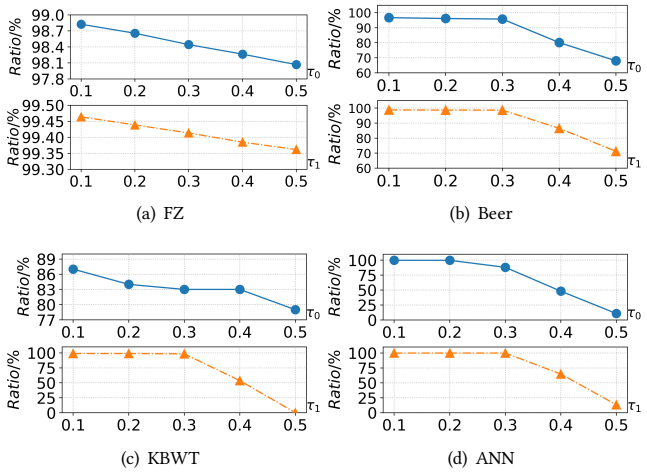


Figure 19: Evaluation on RAC.

The second application is data transformation. There are numerous studies in this application, including example-driven methods [5, 33, 35, 51, 62] and deep learning-based methods [18, 40, 49]. For example, Zhu et al. propose a method called Auto-join [62], which relies on the provided examples and a set of pre-defined string-based transformation units, such as substring and split, to describe the transformations. Meanwhile, DTT [18] exploits deep learning model to transform source values into the desired target representation. Recently, some LLM-based methods have been proposed [37, 45]. Last, data generation is a vital task in data management, such as code generation [44, 59]. LLMs are capable of adapting to code-related tasks without requiring task-specific model training or fine-tuning [29, 38, 48]. Recent works [17, 47] have leveraged LLMs for the generation of code assertions. For all these three tasks, the LLM-based methods have demonstrated their effectiveness and outperformed the non-LLM based methods. However, they mainly focus on the strategies of selecting demonstrations, and pay less attention to cost-effective LLMs . In this paper, we focus on batch prompting for these applications in order to reduce the LLM cost while achieving higher accuracy.

6 CONCLUSION

In this paper, we study the problem of batch prompting in leveraging LLMs for data management. To this end, we develop a framework named Optimized Batch Prompting (OBP) aiming to find the optimal grouping of questions and demonstrations with accuracy guarantee and minimal cost. We first formalize the batch prompting problem as a constrained optimization problem in general setting. Then, we study the hardness of this problem and demonstrate that it is NP-hard. Finally, we design efficient methods for adaptive grouping. Extensive experiments on 14 real-world datasets from three representative data management tasks confirm the superiority of our OBP compared to the state-of-the-art LLM and non-LLM based baselines in terms of both cost and accuracy.

REFERENCES

[1] 2022. Gurobi Optimizer Reference Manual. https://www.pgurobi.com.
[2] 2023. Qwen API. https://qianwen.aliyun.com/qianwen/.
[3] 2024. Claude API. https://www.anthropic.com/.
[4] 2024. OpenAI API. https://platform.openai.com/.
[5] Ziawasch Abedjan, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, and Michael Stonebraker. 2016. Dataxformer: A robust transformation discovery system. In *Proceedings 32th International Conference on Data Engineering*. IEEE, 1134–1145.
[6] Nir Ailon, Moses Charikar, and Alantha Newman. 2008. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)* 55, 5 (2008), 1–27.
[7] 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, 5 (2023), 1132–1145.
[8] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. 2004. Correlation clustering. *Machine learning* 56 (2004), 89–113.
[9] 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.
[10] Matteo Brucato, Juan Felipe Beltran, Azza Abouzied, and Alexandra Meliou. 2016. Scalable Package Queries in Relational Database Systems. *Proceedings of the VLDB Endowment* 9, 7 (2016).
[11] Alberto Caprara, Paolo Toth, and Matteo Fischetti. 2000. Algorithms for the set covering problem. *Annals of Operations Research* 98, 1 (2000), 353–371.
[12] 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.
[13] 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.
[14] Zhoujun Cheng, Jungo Kasai, and Tao Yu. 2023. Batch Prompting: Efficient Inference with Large Language Model APIs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 792–810.
[15] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms*. MIT press.
[16] Marek Cygan, Łukasz Kowalik, and Mateusz Wykurz. 2009. Exponential-time approximation of weighted set cover. *Inform. Process. Lett.* 109, 16 (2009), 957–961.
[17] 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.
[18] 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.
[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 4171–4186.
[20] 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.
[21] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234* (2022).
[22] György Dósa and Jirí Sgall. 2013. First Fit bin packing: A tight analysis. In *30th International symposium on theoretical aspects of computer science (STACS 2013)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik.
[23] Ju Fan, Guoliang Li, Beng Chin Ooi, Kian-lee Tan, and Jianhua Feng. 2015. icrowd: An adaptive crowdsourcing framework. In *Proceedings of the 2015 international conference on management of data*. 1015–1030.
[24] Meihao Fan, Xiaoyue Han, Ju Fan, Chengliang Chai, Nan Tang, Guoliang Li, and Xiaoyong Du. 2024. Cost-effective in-context learning for entity resolution: A design space exploration. In *Proceedings 40th International Conference on Data Engineering*. 3696–3709.
[25] 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.
[26] 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.
[27] 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).

[28] 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. *Proceedings of the VLDB Endowment* 17, 10 (2023), 1034–1045.

[29] 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.

[30] 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.

[31] 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).

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

[33] 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.

[34] Hongye Jin, Xiaotian Han, Jingfeng Yang, Zhimeng Jiang, Zirui Liu, Chia-Yuan Chang, Huiyuan Chen, and Xia Hu. 2024. LLM Maybe LongLM: SelfExtend LLM Context Window Without Tuning. In *Forty-first International Conference on Machine Learning*.

[35] 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.

[36] Moe Kayali, Anton Lykov, Ilias Fountalis, Nikolaos Vasiloglou, Dan Olteanu, and Dan Suciu. 2024. Chorus: Foundation Models for Unified Data Discovery and Exploration. *Proceedings of the VLDB Endowment* 17, 8 (2024), 2104–2114.

[37] Moe Kayali, Anton Lykov, Ilias Fountalis, Nikolaos Vasiloglou, Dan Olteanu, and Dan Suciu. 2024. Chorus: Foundation Models for Unified Data Discovery and Exploration. *Proceedings of the VLDB Endowment* 17, 8 (2024), 2104–2114.

[38] Junaed Younus Khan and Gias Uddin. 2022. Automatic code documentation generation using gpt-3. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–6.

[39] 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).

[40] M Lewis. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461* (2019).

[41] 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.

[42] Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. 2024. Long-context LLMs Struggle with Long In-context Learning. *CoRR* (2024).

[43] 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.

[44] Fabian Nagel, Gavin M Bierman, and Stratis D Viglas. 2014. Code generation for efficient query processing in managed runtimes. *Proceedings of the VLDB Endowment (PVLDB)* 7, 12 (2014), 1095–1106.

[45] 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.

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

[47] 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*.

[48] Julian Aron Prenner, Hlib Babii, and Romain Robbes. 2022. Can OpenAI's codex fix bugs? an evaluation on QuixBugs. In *Proceedings of the Third International Workshop on Automated Program Repair*. 69–75.

[49] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research* 21, 140 (2020), 1–67.

[50] N Reimers. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *arXiv preprint arXiv:1908.10084* (2019).

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

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] 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).

[58] 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.

[59] Lixi Zhang, Chengliang Chai, Xuanhe Zhou, and Guoliang Li. 2022. Learnedsqlgen: Constraint-aware sql generation using reinforcement learning. In *Proceedings of the 2022 International Conference on Management of Data*. 945–958.

[60] 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.

[61] 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).

[62] 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.