Tailoring the Shapley Value for In-context Example Selection towards Data Wrangling

Zheng Liang[1], Hongzhi Wang[1,✉], Xiaoou Ding[1], Zhiyu Liang[1], Chen Liang[1], Yafeng Tang[1], Jianzhong Qi[2]
[1] School of Computer Science, Harbin Institute of Technology, Harbin, China
[2] School of Computing and Information Systems, University of Melbourne, Melbourne, Australia
{lz20, wangzh, dingxiaoou,zyliang,chenliang}@hit.edu.cn, tangyf@stu.hit.edu.cnjianzhong.qi@unimelb.edu.au

***Abstract*—Data wrangling (DW) is a fundamental step to prepare data for downstream mining tasks. Recent studies explore large language models (LLMs) to form a lightweight DW paradigm. Such studies typically require prompting an LLM with a DW task together with a few examples as task demonstrations (i.e., in-context learning). A problem yet to be explored is how to select the examples, to maximize task effectiveness given constraints on the size of the examples. To fill this gap, we introduce the *constrained Shapley value* (CSV), a tailored variant of the Shapley value with a constraint on the LLM prompt size, to guide example selection. We show that CSV has desirable properties in example importance estimation. Using CSV directly for LLM-based DW is still computationally intractable. We further propose *activated contribution* (ACSV) as an unbiased estimation for CSV and sample allocation algorithms with approximation guarantees. Empirical results show that, compared with DW examples manually selected by experts, CSV improves the effectiveness of LLMs for DW tasks including schema mapping, entity matching, error detection, and missing value imputation by 5.90% averagly in F1 score, demonstrating the general applicability of CSV for in-context learning example selection towards DW tasks.**

***Index Terms*—Data wrangling, In-context example selection, Shapley value**

I. Introduction

Data wrangling (DW) [1], [2] covers tasks such as transforming data into usable format, detecting/fixing errors in data, matching similar records/attributes from different sources, and cleansing dirty data. As a fundamental step, it helps enrich individual data records and reduce data quality issues for downstream mining tasks.

LLM-based DW. Due to its importance, DM has been extensively studied. Latest studies use large language models (LLMs), exploiting the commonsense knowledge acquired by LLMs to obtain lightweight DW solutions over datasets from a variety of domains. A pioneering work formulates each data instance for wrangling as a *task question* to prompt an LLM [3] – see an entity matching example in Figure 1(a). Following the in-context learning paradigm, a couple of studies [3], [4] boost LLM-based DW by adding a few examples as part of the prompts (i.e., *few-shot learning*; see an example in Figure 1(b)). Such solutions are lightweight (assuming a pre-built LLM), requiring negligible efforts on data labeling [4].

In-context example selection. A gap in the LLM-based DW studies is how to select the examples to prompt the LLMs, which is referred to as the in-context example selection problem, or *demonstration engineering*, in the literature [6], [7]. Popular solutions for this problem include *retrieval augmented generation* (RAG) methods, *topic model* methods, and *influence-based* methods.

RAG methods [8]–[10] measure the similarity (e.g., embedding distance [8], [9] or mutual information [10]) between candidate examples and the task question, and select the most similar candidates as the task examples. Topic model methods [11] view LLMs as topic models that can infer a task-related latent concept variable θ from a few demonstrations, based on which an answer is generated. Given task questions, they select examples that are the most likely to guide the LLM to infer the latent concept variable θ learned from training data. Influence-based methods [12]–[14] select the example that yields the largest performance gap with and without using the example on a validation dataset. A full discussion of these methods and other related works is included in Section V.

Limitations of existing solutions. The RAG and topic model methods only consider the relevance between the candidate examples and the task question. They do not concern the actual impact of the selected examples on the effectiveness of the LLMs. While the influence-based methods examine such impacts, they are costly to run (especially when there are many examples and a large validation set to test) and may lead to unstable test performance [13].

Example 1: *Take entity matching over the Beer dataset [5] in Figure 1 as an example. Suppose that the training, validation, and test subsets each contains 150 entity pairs to be matched, while each LLM question answering (QA) turn takes at most 8 examples (as done by Fan et al. [7]) for in-context learning. Using influence-based methods [12]–[14], to compute LLM-based DW performance for all example sets takes* $150 \cdot \sum_{i=0}^{8} \binom{150}{i}$ *questions, which translates to over 170,000 years if each question takes 1 second to run, and $0.22 billion if each example costs $0.00004 (ChatGPT rate [15]).*

The Shapley value. In this paper, we fill the gap by studying cost-efficient example selection for LLM-based DW. We exploit a technique called the *Shapley value* (SV) [16], which originates from the cooperative game theory, to help evaluate the joint impact of a subset of candidate examples to prompt an LLM.

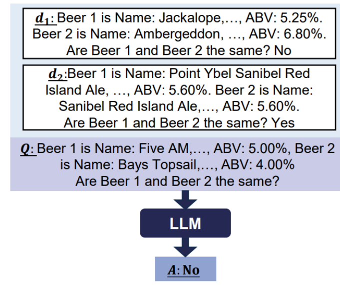
Given a utility function U and a set of players $D =$

✉ Hongzhi Wang is the corresponding author.

Beer name	Brew factory name	Style	ABV
Point Ybel Sanibel Red Island Ale	Point Ybel Brewing	Irish Ale	5.60%
Jackalope	Odd Side Ales	American Amber/Red Ale	5.25%
Five AM	BrewDog	American Amber/Red Ale	5.00%

Beer name	Brew factory name	Style	ABV
Bays Topsail	Bays	Bitter	4%
Sanibel Red Island Ale	Point Ybel Brewing	American Amber/ Red Ale	5.60%
Ambergeddon	Ale Asylum	American Amber/Red Ale	6.80%

(a) Example of entity matching



(b) Few-shot LLM-based entity matching

Fig. 1: An example of LLM-based entity matching on the Beer dataset [5]. The aim is to determine records from two tables that refer to the same beer product. Every row in a table refers to a beer with four attributes. Two entity pairs d_1 and d_2 are selected as 2-shot examples from a training dataset, while Q denotes the entity matching question asked to an LLM. In practice, all attributes are included in the input to the LLM. We only show the name and ABV attributes here for simplicity.

$\{d_1, d_2, \ldots, d_n\}$, the SV of a player d_i is defined as:

$$SV(d_i) = \frac{1}{n} \sum_{S \subset D \setminus \{d_i\}} \frac{U(S \cup \{d_i\}) - U(S)}{\binom{n-1}{|S|}}. \quad (1)$$

Intuitively, each player can be thought of as a candidate example. The utility function $U(S)$ refers to the reward if a set (or coalition) S of players cooperate in a game, which can be thought of as the accuracy (or some other performance metric like the F1 score; same below) of an LLM when the set S of candidate examples are used as the demonstrations to prompt the LLM for some DW task. Here, a validation set is used to compute the accuracy. Then, $SV(d_i)$ models the additional contribution to the LLM accuracy if candidate example d_i is added to the set of examples to prompt the LLM. We can prompt the LLM with k examples that have the largest SVs, where k is an application oriented constraint, e.g., input size limit of the LLM or user budget constraint[1].

We note that SV comes with an approximation error guarantee when the computation cost is constrained [17]. The SV technique is also orthogonal to the RAG methods, i.e., one can use RAG to perform an initial filtering of the candidate examples and SV to refine the selection of the examples. These properties prompt us to adapt SV for example selection in LLM-based DW tasks.

Challenges. Direct computation of the SV for n players (i.e., candidate examples) is #P-hard [18]. This calls for effective and efficient approximations. LLM-based DW poses additional challenges [17]:

(i) *Excessive selected examples*: Conventional SV approximation methods may select hundreds of examples, which may be too large to fit LLM prompting scenarios [19] or may lead to high LLM API call costs and/or decay in LLM effectiveness [18].

(ii) *High example selection costs*: The state-of-the-art (SOTA) SV approximation technique, $(\epsilon, \delta)-$approximation, takes $O(n \log n)$ (recall that n is the number of candidate examples) utility function (i.e., U in Equation 1) computations [20], each of which requires asking the LLM questions with all validation data, i.e., $O(mn \log n)$ LLM API calls are needed given m validation records, which can be associated with non-trivial costs when m is large.

Contributions. We address these challenges and tailor SV for example selection in LLM-based DW, making contributions as follows:

(i) We model LLM-based DW as a problem of selecting the top-k examples for prompting an LLM, given constraints on the number of examples k. We propose to guide example selection with the *constrained Shapley value* (CSV), which restricts the SV computation in Equation 1 by imposing $|S| < k$. To the best of our knowledge, this is the first work that extends SV to LLM-based DW.

(ii) We adapt the $(\epsilon, \delta)-$approximation algorithm [20] for CSV approximation, resulting in an algorithm named *MCSV*. Due to the restricted size of examples, MCSV reduces the number of utility function computation from $O(mn \log n)$ to $O(m \log n)$. We further propose a method named *activated contribution* that enables sharing a sample subset S for computing the CSV for different candidate examples. We propose two stratified subset sampling strategies. These lead to an algorithm named *ACSV* with $O(mn)$ utility function computation in theory but more efficient than MCSV in practice due to the smaller hidden constants.

(iii) Empirically, our ACSV algorithm outperforms all non-batched prompting algorithms, by up to 5.90% in F1 score on DW tasks over commonly used data wrangling benchmark datasets. Combining with batch prompting, we introduce BCSV, which further boosts the F1 score of the SOTA batch prompting-based DW solution by 4.43%.

II. Preliminaries

Data wrangling (DW) is an overall term to describe the process of transforming raw data into a more usable form.

[1]LLM APIs such as ChatGPT API are charged by the size of both input and output.

TABLE I: Notation

Symbol	Meaning
T_1, T_2	A pair of tables for entity matching
$A_1, A_2, \dots$	The attribute set of tables
t_1, t_2	A pair of tuples from T_1 and T_2
U	The evaluation metric for an EM task
$D = \{d_1, d_2, \dots, d_n\}$	Demonstration candidates (training records)
Q	Test records
V	Validation records
k	The maximum demonstration size
D^*	The optimal demonstration subset
n	The size of D
m	The size of V
$CSV(d_i)$	The constrained Shapley value of d_i
S, S'	Some subset (coalition) of D
S_j	All subsets of size j
$S_{i,j}$	All size j subsets containing sample i
w_j	Number of samples allocated to S_j
$w_{i,j}$	Number of samples allocated to $S_{i,j}$
B	The budget (in number of tokens allowed)
$AC(S, d_i)$	The activated contribution for $CSV(d_i)$

Following a recent LLM-based study [3], we focus on four DW tasks defined below.

Our goal is to select a few examples to guide an LLM to process each of these tasks with few-shot learning. Our solution is versatile, which does not involve task-specific optimizations. Thus, the task definitions are included just to make the paper self-contained.

Definition 1. *(Entity Matching, EM): Consider a pair of tables T_1 and T_2 with a unified schema of l columns $\{A_1, A_2, \dots, A_l\}$, where each row in the tables corresponds to an entity, and A_i refers to the i-th column, i.e., an attribute of an entity. Entity matching is a binary classification task that determines whether a pair of entities $(t, t') \in T_1 \times T_2$ refer to the same real-world entity.*

Definition 2. *(Schema Mapping, SM): Consider two groups of tables $\mathcal{T}_1$ and $\mathcal{T}_2$, and any random pair of tables $T_1 \in \mathcal{T}_1$ and $T_2 \in \mathcal{T}_2$ with schemata (i.e., columns) $\{A^1_1, A^1_2, \dots, A^1_l\}$ and $\{A^2_1, A^2_2, \dots, A^2_r\}$, respectively. Schema mapping is a binary classification task that determines whether two attributes A^1_i and A^2_j from the two tables refer to the same real-world property.*

Definition 3. *(Error Detection, ED): Consider a table T with l columns $\{A_1, A_2, \dots, A_l\}$. Error detection is a binary classification task that determines whether the value of some attribute $t.A_i$ of a tuple $t \in T$ contains errors, e.g., a value that deviates from the truth.*

Definition 4. *(Missing Value Imputation, MVI): Consider a table T with l columns $\{A_1, A_2, \dots, A_l\}$. Missing value imputation aims to determine the most likely value for an attribute $t.A_i$ of $t \in T$, which has been originally missing.*

We summarize the frequently used symbols in Table I.

A. Problem Statement

We study DW using LLMs in a few-shot learning setting, where the core challenge is to select the examples to form the input prompt given to an LLM, e.g., ChatGPT.

Problem 1. ***(LLM Example Selection for DW, LESD)****: Given a set of n candidate examples (of some DW task) $D = \{d_1, d_2, \dots, d_n\}$, a budget B, and a constant k ($k \geq 1$), the LESD problem aims to find the optimal subset of candidate examples $D^* \subset D$, where:*

$$D^* = \text{argmax}_{D'} U(D') \ s.t. \ D' \subset D, |D'| \leq k. \quad (2)$$

Here, U is a utility function that measures LLM effectiveness for the task. Each utility function call incurs some cost, and the total cost of function calls during the optimization process is constrained by B.

Ideally, U should be computed on the task test set. In practice, only a training or validation set is available at model design stage. Thus, we compute U on a validation set V, which returns the F1 score (or accuracy, depending on the task).

Each call of function U is a question run on some LLM, and commercial LLMs like ChatGPT charge a fee for every call to their APIs. Thus, we define B as a budget to make LLM API calls.[2]

Computing U requires asking the LLM $m = |V|$ (i.e., the size of V) questions. Each question serializes a record from V with a set of selected examples D' to be fed to the LLM (as Figure 1(b) shows). A brute-force enumeration over all subsets of D of size up to k takes $O(mn^k)$ questions.

To reduce the number of questions, we may select examples with the largest SVs. However, as discussed earlier, a naive adoption of the SV technique does not offer control over the size of the example set D^* selected, while there are also cost issues to compute SV. To tackle these issues, we define a "constrained" version of SV.

B. CSV-Based Problem Formulation

Definition 5. *(Constrained Shapley Value, CSV): Given a set of n candidate examples $D = \{d_1, d_2, \dots, d_n\}$, a utility function U, and a constant k ($k \geq 1$), the constrained Shapley value (CSV) for candidate example d_i is defined as follows:*

$$CSV(d_i) = \frac{c}{n} \sum_{S \subset D \setminus \{d_i\}, |S| < k} \frac{U(S \cup \{d_i\}) - U(S)}{\binom{n-1}{|S|}}, c > 0 \quad (3)$$

The intuition of CSV is as follows. We aim to find the top-k examples to prompt an LLM. As such, we only need to find candidate examples that, when added to the set of k examples from LLM prompting, leads to the maximum utility gain. Thus, we restrict the size of the set S in Equation 3 to be $k-1$ (such that adding d_i to it makes a size-k subset). In Equation 3, c can take any non-zero constant value. Finding the top-k examples with CSV does not concern the exact value of c, as long as the same c value is used for CSV calculation across all examples. We set $c = 1$ for simplicity.

[2]We do not define B as the maximum number of API calls because LLMs like ChatGPT charge access fees by the number of tokens communicated rather than by the number of API calls.

Properties of CSV. To find the top-k examples to prompt an LLM for DW, we need a *value function* $F(d_i)$ to evaluate the contribution of a candidate example d_i to the utility function U. We use $CSV(\cdot)$ as this value function, for its following attractive properties for *reward allocation* (i.e., the process of assigning the contribution of each candidate example d_i in a subset $S \cup \{d_i\} \subset D$ to the utility $U(S \cup \{d_i\})$). We show that $CSV(\cdot)$ is the only value function that satisfies all such properties[3], when $|S \cup \{d_i\}| \leq k$.

Proposition 1. *(Constrained Shapley Value Uniqueness): For any "game" (D, U), where U is a utility function that maps a subset S of players $D = \{d_1, d_2, \ldots, d_n\}$ to a real number: $U(S) \rightarrow \mathbb{R}$, if U can only take coalitions (i.e., subset S of D) containing at most k players (i.e., candidate examples) as input, $CSV(\cdot)$ is the* only *value function $F(d_i)$ that satisfies the following properties for reward allocation:*

Symmetry: Any two candidate examples with equal marginal contributions to every subset S receive the same reward. Formally, $\forall d_i, d_j \in D$, if $\forall S \subset D, |S| < k : U(S \cup \{d_i\}) = U(S \cup \{d_j\})$, then $F(d_i) = F(d_j)$, where $F(d_i)$ and $F(d_j)$ are the rewards of d_i and d_j.

Additivity: The utility function value on all players $U(D)$ can be fully divided among the candidate examples, i.e., $U(D) = \sum_{d_i \in D} F(d_i)$.

Balance: For any player $d_i \in D$ playing any two games (D, F_1) and (D, F_2) getting reward $F_1(d_i)$ and $F_2(d_i)$, respectively; its reward allocation for the game $(D, F_1 + F_2)$ is $F_1(d_i) + F_2(d_i)$.

Zero element: A candidate example with zero contribution to the reward of every subset of D with up to k elements has a reward of 0. Formally, $\forall d_i \in D$, if $\forall S \subset D, |S| < k$: $U(S \cup \{d_i\}) = U(S)$, then $F(d_i) = 0$.

Proof. Due to space limit, we put the proof in an online technical report [21]. Same for the rest of the propositions. □

We note that other example contribution evaluation methods cannot satisfy all four properties above simultaneously. For example, influence-based methods [12]–[14] only satisfy the zero element property [22], which may produce misleading example contribution estimation [22] and hence suboptimal example selections.

Using CSV, solving the LESD problem can be approximated by solving the following top-k CSV selection problem.

Problem 2. ***(Top-k CSV selection)**: Given a set of n candidate examples $D = \{d_1, d_2, \ldots, d_n\}$, a budget B, and a constant size k ($k \geq 1$), the top-k CSV selection problem aims to find a subset $D^* \subset D$:*

$$D^* = \{d \in D | \forall d' \in D \setminus D^*, CSV(d) \geq CSV(d')\} \\ s.t.\ |D^*| = k \tag{4}$$

The calculation of $CSV(d)$ also calls the utility function U. The total costs incurred by the utility function calls is constrained by B.

[3]We note that the original SV also satisfies these four properties.

The utility function U computed with a machine learning model (e.g., an LLM in our case) is not typically concave, which means that (D, U) is not a superadditive game. As a result, solutions of Problem 2 above are usually only approximations for Problem 1. Theoretical analysis on the solution quality using SV for example selection when U is non-concave is an open problem [23]. Similarly, analyzing the theoretical solution quality of Problem 2 for non-concave utility functions itself is worth a separate work. Thus, we leave it for future studies.

Next, we focus on solving the top-k CSV selection problem. Computing an exact solution for the problem is still computational intractable, which requires checking all subsets of size 1 to k, which takes $O(mn^k)$ time. We propose to leverage sampling-based techniques to solve the problem. Our algorithms, to be detailed in Section III, obtain (ϵ, δ)-approximation (in terms of CSV calculation, not solution quality for example selection which is an open problem as mentioned above) in $O(m \log n)$ or $O(mn)$ time. Here, ϵ and δ are parameters controlling the quality of the approximate solutions, which will be introduced together with the algorithms.

C. Basic Solution Steps

Our top-k CSV selection algorithms follow the overall steps below.

Step 1. We take an iterative approach to estimate the CSV of the candidate examples. In each iteration, we examine LLM performance using a sampled subset $S \subset D$ as in-context examples on the validation set. The result is used to update the estimation of $CSV(d_i)$ for all $d_i \in D$.

Step 2. Each iteration in Step 1 incurs costs to call the LLM API as outlined earlier. Such costs are accumulated. When the total costs exceed the cost budget B, we terminate the algorithm and return the top-k examples with the largest estimated CSV values as D^*.

Step 3. The set D^* returned by Step 2 are used for the DW task on the test set to produce the final task output.

The procedure is an iterative process based on sampling subsets $S \subset D$, and more sampled subsets can lead to higher approximation quality (see Propositions 2, 5, and 6). Meanwhile, there is a budget B that constrains the number of iterations (and hence the number of sampled subsets). Therefore, the key challenge here is how to better exploit each sampled subset to maximize the estimation quality. In the next section, we will describe our algorithms to generate the subset S for each iteration in Step 1 for effective CSV estimation.

III. CSV Approximation

We first adapt a classic SV approximation technique for CSV approximation, resulting in an algorithm named *MCSV* that takes $O(m \log n)$ LLM API calls, in Section III-A. Then, we present the *activated contribution* technique, along with two subset sampling strategies for activated contribution-based CSV approximation, which leads to an algorithm named *ACSV* with $O(mn)$ LLM API calls in theory but highly efficient in practice, in Section III-B. Finally, we present the *BCSV*

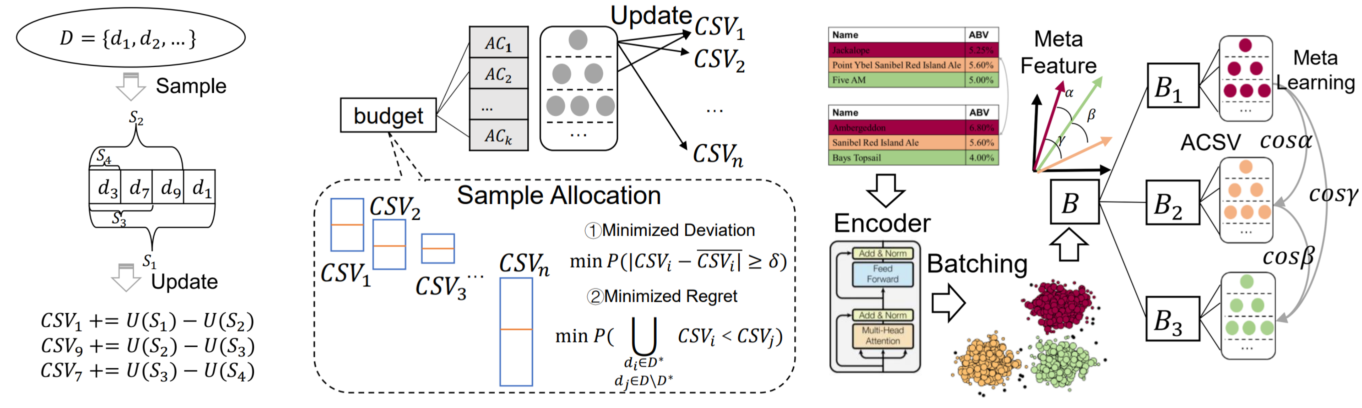


Fig. 2: Proposed CSV approximation algorithms for LLM-DW. The examples with top-k estimated CSV values returned by these algorithms are to be used for LLM in-context learning for DW tasks.

algorithm for batch processing scenarios, i.e., when the testing records can be clustered into batches and prompt examples are selected for each batch, in Section III-C.

A. MC-Based Approximation

A classic SV approximation technique uses the marginal contributions [24]. We adapt it for CSV approximation and named the adapted algorithm MCSV. The algorithm iteratively and *randomly samples* subsets of D, and it updates the estimated CSV for the sampled candidate examples. When the algorithm terminates, the top-k candidate examples with the largest estimated CSV are returned.

Rewriting $CSV(d_i)$ based on permutation. We first introduce the estimation goal of MCSV. Suppose π is a permutation of $\{1, \ldots, n\}$, denoted by $\pi = \{\pi_1, \ldots, \pi_n\}$, where $\pi_i \in [1, n], \pi_i \neq \pi_j, \forall i, j \in [1, n]$, we can rewrite $CSV(d_i)$ as:

$$CSV(d_i) = \frac{c}{n!} \sum_{\pi \subset \Pi} U(\pi^i \cup \{d_i\}) - U(\pi^i), c > 0 \quad (5)$$

In Equation 5, Π are all the $n!$ permutations of $\{1, \ldots, n\}$ each corresponding to a sequence of candidate examples in D, π^i is a permutation (i.e., the corresponding candidate examples) taken from the first $|\pi^i|$ elements in Π where the next element is i. The equivalence between Equation 3 and Equation 5 comes from the result of a previous work [25]. We omit the proof here due to space limit.

The MCSV algorithm. Algorithm 1 summarizes MCSV, where the estimated CSV of d_i is denoted by $\overline{CSV(d_i)}$, the number of times that d_i has been sampled is denoted by $count(d_i)$, and the accumulated cost triggered for all the evaluation of the utility function U is denoted by $cost$. At start, both $\overline{CSV(d_i)}$ and $count(d_i)$ are set to 0 for all $d_i \in D$, while $cost$ is also initialized to 0 (Line 1).

The algorithm then proceeds to sample D and update the CSV estimations until $cost$ reaches budget B (Line 2). In each iteration, we take π' as a random permutation of the IDs of examples in D, and π is the first k IDs in π' (Line 3). Figure 2(a) shows an example, where $k = 4$ and $S = \{d_1, d_3, d_7, d_9\}$, while $\pi = 3, 7, 9, 1$. Instead of using S to update the CSV estimation of only one candidate example, we update $k - 1$ estimations progressively (Lines 4 to 8), exploiting the term $U(S \cup \{d_i\}) - U(S)$ in Equation 3. In iteration j, $U(S \cup \{d_i\})$ is calculated, which becomes $U(S)$ of the next iteration (Line 5). As a special case, when $j = 1$, $U(\{d_{\pi_1}, d_{\pi_2} \ldots, d_{\pi_j}\}) = U(\emptyset)$, which corresponds to a zero-shot LLM call. The steps above save one utility function (i.e., $U(S)$) evaluation (which triggers LLM API calls) in each iteration. We follow the order of permutation π during this iterative process. Using the sample subset in Figure 2(a), the CSV estimations will be updated for d_7, d_9, and d_1.

Here, by using the permutation order instead of, e.g., the

Algorithm 1 Marginal Contribution-Based Approximation (MCSV)

Input: Candidate examples $D = \{d_1, d_2, \ldots, d_n\}$, cost constraint B
Output: Examples with top-k CSV: $topk(D, \overline{CSV}) \subseteq D$

1: $\overline{CSV(d_i)} \leftarrow 0, count(d_i) \leftarrow 0$ for $1 \leq i \leq n$, $cost \leftarrow 0$;
2: **while** $cost \leq B$ **do**
3: $\quad \pi \leftarrow random\ permutation\ of\ k\ examples\ in\ D$;
4: $\quad$ **for** $j = 1$ to k **do**
5: $\quad\quad \overline{CSV(d_{\pi_j})} \leftarrow \overline{CSV(d_{\pi_j})} + U(\{d_{\pi_1}, d_{\pi_2} \ldots, d_{\pi_j}\}) - U(\{d_{\pi_1}, d_{\pi_2} \ldots, d_{\pi_{j-1}}\})$;
6: $\quad\quad$ update($cost$);
7: $\quad\quad count(d_{\pi_j}) \leftarrow count(d_{\pi_j}) + 1$;
8: $\quad$ **end for**
9: **end while**
10: **for** $i = 1$ to n **do**
11: $\quad \overline{CSV(d_i)} \leftarrow \overline{CSV(d_i)} / count(d_i)$;
12: **end for**
13: **return** $topk(D, \overline{CSV})$;

order of the candidate example IDs, we achieve an unbiased estimation of $CSV(d_i)$. Also, according to Equation 5, we only accumulate $U(S \cup \{d_i\}) - U(S)$ but omit the denominator $\binom{n-1}{|S|}$ from Equation 3 in CSV estimation.

When the CSV estimation is updated for a candidate example d_{π_j}, we increase $count(d_{\pi_j})$ by 1, and we update $cost$ to accumulate the LLM API cost triggered by evaluating U (Lines 6 and 7). When budget B is exhausted, the sampling and CSV estimation process is terminated. We average the CSV estimation $\overline{CSV(d_i)}$ by $count(d_i)$ (Lines 10 to 12). Finally, the top-k candidate examples in D with the largest estimated CSV values are returned, which is denoted by $topk(D, \overline{CSV})$ (Line 13).

Algorithm anlaysis. Due to random sampling and permutation, Algorithm 1 ensures that the estimation is bounded near the exact value, as shown in the following proposition.

Proposition 2. *(Monte Carlo Marginal Contribution Approximation Quality [26]) According to Hoeffding's inequality, given the range $r = max(CSV(d_i)) - min(CSV(d_i))$ of $CSV(d_i)$, a CSV estimation error bound ϵ, and a confidence level $1 - \delta$, Algorithm 1 requires $\frac{mr^2 avl(D)k^2}{4\epsilon^2} \log \frac{2n}{\delta}$ API token costs and $\frac{mr^2}{2\epsilon^2} k \log \frac{2n}{\delta}$ queries to an LLM to ensure $P(|\overline{CSV(d_i)} - CSV(d_i)| \geq \epsilon) \leq \delta$, where $avl(D)$ denotes the average number of tokens in serialized EM examples.*

We note that the main difference (i.e., our adaptation) between Algorithm 1 and the classic marginal contributions algorithm [24] is at Line 3, where we use a permutation of size k instead of n in the original algorithm, which reduces the running time costs from $O(mn \log n)$ to $O(mk \log n)$. Recall that the $O(m)$ term comes for evaluating U over a validation set of size m. The term $O(\log n)$ comes from the number of permutations to ensure (ϵ, δ)-approximation, while the term $O(n)$ (and $O(k)$) comes from the length of each permutation. Considering k as a constant, MCSV achieves a time complexity of $O(m \log n)$.

Discussion. An issue with the MCSV algorithm is that each call of the utility function $U(S)$ over a subset $S \subset D$ can only be used for CSV estimation of two candidate examples (Line 5). As such, $U(S)$ could be *repeatedly computed* for the same set of S whenever it is part of the sample from Line 3. We introduce an algorithm to remove such repetitive computation in the next subsection.

Another classic SV approximation technique is called complementary contributions (CC) [17], which computes *pairwise sampling* $U(D \setminus S) - U(S)$ instead of the marginal contribution (i.e., $U(S \cup \{d_i\}) - U(S)$) at Line 5 of Algorithm 1. This modification enables CC sharing among all n candidate examples (Lines 9 to 11). The issue with the CC algorithm and *pairwise sampling* is that D is typically much larger than S (e.g., $|D| = 450$ and $|S| \leq 5$ in the smallest dataset used our experiments), such that $|D \setminus S|$ can also be much larger than k, which can be difficult to fit an LLM's input constraint or for the LLM to follow, or can break the budget constraint B prematurely. As such, we do not consider CC further.

B. AC-Based Approximation

To further reduce the approximation costs, we propose the concept of activated contribution (AC) and the ACSV algorithm based on this concept, as illustrated by Figure 2(b).

Rewriting CSV based on activated contribution. Intuitively, computing $CSV(d_i)$ in Equation 3 requires utility function $(U(S))$ computation for all possible subsets $S \subset D, 0 \leq |S| \leq k$. Based on this observation, we propose an efficient CSV approximation algorithm (Algorithm 2), following another rewrite of Equation 3. We first define a weighted version of the utility function called *activated contribution*.

Definition 6. *Given a set of candidate examples $D = \{d_1, d_2, \ldots, d_n\}$ and a subset $S \subset D$, the activated contribution of S to d_i is defined as folllows:*

$$AC(S, d_i) = U(S) \cdot f(S, d_i), \tag{6}$$

$$f(S, d_i) = \begin{cases} 0, & \text{if } d_i \notin S, |S| = k \\ -1, & \text{if } d_i \notin S, |S| < k \\ \frac{n}{|S|} - 1, & \text{if } d_i \in S \end{cases} \tag{7}$$

Here, function $f(S, d_i)$ mimics the activation function in neural networks (and hence the name of "activated contribution"). The "weight" $f(S, d_i)$ for the three different cases are derived from the definition of CSV (Equation 3) to guarantee equivalence between the CSV and the average of AC (see our technical report [21] for a full proof of equivalence).

Proposition 3. *(Effectiveness of Activated Contribution Approximation) Given a set of candidate examples $D = \{d_1, d_2, \ldots, d_n\}$, the CSV of d_i can be computed by the average of activated contribution:*

$$CSV(d_i) = \sum_{S \subset D, 0 \leq |S| \leq k} \frac{AC(S, d_i)}{\binom{n}{|S|}} \tag{8}$$

Compared to the marginal contribution [27], the activated contribution can share the computation of $U(S)$ when computing $AC(S, d_i)$ for all n candidate examples instead of repeatedly computing $U(S)$ for every d_i (i.e., Line 5 of 1), while it does not rely on pairwise sampling like the CC technique [17] discussed above. This property makes activated contribution attractive for the input constraint of utility function $U()$, and it is the key property leading to cost reduction for CSV approximation.

The ACSV algorithm. We give an AC-based algorithm to approximate CSV as shown in Algorithm 2. In each iteration, the algorithm randomly samples a subset S of candidate examples. The sampling process is conducted with uniform probability on all subsets (Line 3), and we update the estimated CSV for every candidate example with AC based on Equations 6 and 7 (Lines 4 to 7). The algorithm terminates when $cost$ exceeds B, returning examples with top-k largest approximated CSV values as the result (Lines 8 to 11).

The subset sampling process (Line 3) can be done with uniform weights on each possible subset. This ensures unbiased sampling, as CSV is an average of AC. Algorithm 2

Algorithm 2 Activated Contribution-Based Approximation (ACSV)

Input: Candidate examples $D = \{d_1, \ldots, d_n\}$, cost constraint B

Output: Demonstrations with top-k CSV: $S = \{d_{\pi_1}, ..., d_{\pi_k}\} \subseteq D$

1: $\overline{CSV(d_i)} \leftarrow 0$ for $1 \leq i \leq n$; $\overline{CSV_{i,j}}, m_{i,j} \leftarrow 0$ for $1 \leq i, j \leq n$; $cost \leftarrow 0$;
2: **while** $cost \leq B$ **do**
3: $\quad S \leftarrow random \quad subset \quad of \quad D$;
4: $\quad$ **for** i=1 to n **do**
5: $\quad\quad \overline{CSV_{i,|S|}} \leftarrow \overline{CSV_{i,|S|}} + AC(S, d_i); m_{i,|S|} \leftarrow m_{i,|S|} + 1; update(cost)$;
6: $\quad$ **end for**
7: **end while**
8: **for** i=1 to n **do**
9: $\quad \overline{CSV(d_i)} \leftarrow \sum_{j=1}^{k} \frac{\overline{CSV_{i,j}}}{m_{i,j}}$;
10: **end for**
11: **return** $topk(\overline{CSV})$;

then gives an unbiased estimation of $CSV(d_i)$. We note that existing intuitive influence-based sampling methods [12]–[14] are all biased estimations of CSV.

Proposition 4. *(Unbiased Estimation of Activated Contribution Approximation) Given a set of candidate examples $D = \{d_1, d_2, \ldots, d_n\}$, Algorithm 2 gives an unbiased estimation of $CSV(d_i)$ for every $d_i \in D$, i.e., $\forall d_i \in D : E(\overline{CSV(d_i)}) = CSV(d_i)$, where $\overline{CSV(d_i)}$ is the estimation of $CSV(d_i)$ computed by Algorithm 2.*

The uniform sampling in Line 3 can still suffer in efficiency. Following the stratified sampling idea [28], we adapt the sample probability of the coalitions to reduce $U(S)$ computation while retaining the same approximation upper bound. We introduce two sampling strategies below.

Sample allocation w.r.t. deviation minimization. We first adapt a stratified Shapley value sampling strategy [28] for the CSV, which randomly selects m_i samples from S_j to minimize $|\overline{CSV(d_i)} - CSV(d_i)|$. Using Hoeffding's inequality, with probability $1 - \delta$, we have [29]:

$$|\overline{CSV(d_i)} - CSV(d_i)| \leq \sum_{j=0}^{k-1} |\overline{CSV_{i,j}} - CSV_{i,j}| \leq \sum_{j=0}^{k-1} 2r(j+1)\sqrt{\frac{-\log\frac{\delta_i}{2}}{2m_{i,j}}} \tag{9}$$

$m_{i,j}$ is the number of j-sized subsets containing test record d_i during sampling process, i.e., the sample budget allocated to the j-th layer of d_i [28].

The Deviation Minimization problem is formulated as:

$$\min \sum_{j=0}^{k-1} \frac{j+1}{\sqrt{m_{i,j}}}, \quad s.t. \sum_{j=0}^{k-1} \sum_{p}^{m_{i,j}} \mathrm{len}(d_{\pi_p}) = B \tag{10}$$

Here, $\mathrm{len}(\cdot)$ is a function that returns the number of tokens LLM cost within d_{π_p}. Directly solving the above problem is difficult, as it is hard to know $m_{i,j}$ when the samples are entangled for all candidate examples. We first relax $m_{i,j}$ using its expectation $E(m_{i,j})$. By definition of CSV, we have $E(m_{i,j}) = E(m_j | d_i \in S, |S| = j+1)$. Due to uniform sampling, the expectation of a candidate example being in a sample can be computed as $E(m_{i,j}) = \frac{j}{n} m_{j+1}$. For a further relaxation, note that the LLM budget is considered as the sum of input and output tokens. Recall in our LESE problem definition, if we need j examples for each single sample, the expected token cost of a sample of size j can be estimated with $(j+1) \cdot avl(D)$.

Overall, the relaxed Deviation Minimization problem becomes as follows:

$$min \sum_{j=1}^{k} \frac{j+1}{\sqrt{m_j}} \sqrt{\frac{n}{j}}, \quad s.t. \sum_{j=1}^{k} m_j (j+1) avl(D) = B \tag{11}$$

Using the method of Lagrange multipliers, we obtain:

$$m_j = \frac{B\sqrt[3]{\frac{1}{j}}}{avl(D)\sum_{j'=1}^{k}(j'+1)\sqrt[3]{\frac{1}{j'}}} \tag{12}$$

Proposition 5. *(AC-based Minimized Deviation Approximation Quality) With sample allocation for minimized deviation, Algorithm 2 requires $\frac{2r^2 \log\frac{2}{\delta}\sqrt{n}}{\epsilon^2}$ QA cost (queries to an LLM) and $\frac{2r^2 avl(D) \log\frac{2}{\delta}\sqrt{n}}{\epsilon^2} \sum_{j=1}^{k} \frac{(j+1)}{\sqrt[3]{j}}$ API token costs to ensure $P(|\overline{CSV(d_i)} - CSV(d_i)| \geq \epsilon) \leq \delta$.*

Sample allocation w.r.t. regret minimization. The goal of top-k CSV selection problem is to select samples with top-k CSV values. We now model top-k CSV selection as a Multiple Arm Identification Problem [29], [30], where each CSV is considered as an arm from a multi-armed bandit, and the agent needs to identify a subset of the arms corresponding to some criterion [30]. We adapt the SAR algorithm [30] for our problem. The challenge here is mapping the sample allocation of the SAR algorithm to activated contribution, yielding an applicable CSV approximation algorithm. The mapping result is shown in the sample allocation, where m_j is the number of j-sized subsets during sampling.

$$m_j = \left\lceil \frac{n(n-k)}{\overline{\log}k(k^2+k)} \right\rceil \tag{13}$$

Proposition 6. *(AC-based Regret Minimizing Approximation Quality) With sample allocation towards regret minimization, the error probability of Algorithm 2 satisfies the following inequality.*

$$e_n = P(\cup_{i \leq k \leq j} CSV(d_i) < CSV(d_j)) \leq 2k^2 exp(-\frac{n-k}{8H\overline{\log}k})$$

where $H = max_{i \in \{1,..,K\}} i \cdot (|CSV(d_i) - CSV(d_{i+1})|)^{-2}$, $\overline{\log}K = \frac{1}{2} + \sum_{i=2}^{K} \frac{1}{i}$

C. The BCSV Algorithm

We further propose an algorithm named BCSV for batch processing scenarios. The main idea of BCSV is that, in each LLM query, we ask the LLM to process β (instead of one) test records (still with k examples), saving the input cost of feeding k examples to the LLM for β times.

Following a Retrieval Augmented Generation-based method named BatchER [7], we use a text embedding-based batching method to generate test record batches. For each batch, we compute the L_2 distance between the embeddings (generated by an open-sourced language model RoBERTa [31]) of the test records and every candidate example. Afterwards, we take the top-K ($K > k$) candidate examples that are the most relevant for each batch, i.e., the top-K nearest neighbor examples to the test records in a batch. Finally, these top-K candidate examples are filtered using the ACSV algorithm to derive the top-k examples for the batch.

On top of this, our BCSV algorithm uses the meta-learning paradigm to enhance CSV approximation for a new batch based on CSV results from historical batches.

Meta-learning. Meta-learning, or learning to learn, can capture the task structure and map different task input (datasets) to certain hyper-parameters or model structures so as to enable few-shot learning [32]. We aim to reuse CSV estimation results on historical DW batches to recommend in-context examples adaptable to a new EM task, which follows the intuition of meta-learning [33]. To compute CSV for a target dataset, we combine the CSV estimation results on historical DW batches using a weighted sum following the idea of the meta-features [34]. A meta-feature is a vector composed of statistics of a dataset, such as the entropy or maximal value of an attribute. The cosine similarity between two meta-features reflects the similarity of the two corresponding datasets for a machine learning task, which is, in our case, selecting top-k CSV examples for LLM-based DW.

Meta-feature-based similarity expectation. Using meta-features and entity pair embeddings generated by any encoder (e.g., Ditto [35]), the expected CSV of a candidate example d_j in D is computed as follows:

$$\widehat{CSV(d_j)} = \frac{\sum_{d_i \in D'} \cos\langle \vec{D}, \vec{D'} \rangle \cdot l_2\langle e_{d_i}, e_{d_j} \rangle \cdot \overline{CSV(d_i)}}{\sum_{d_i \in D'} \cos\langle \vec{D}, \vec{D'} \rangle \cdot l_2\langle e_{d_i}, e_{d_j} \rangle}. \tag{14}$$

Here, $\vec{D}$ is a meta-feature vector of dataset D, $l_2\langle e_{d_i}, e_{d_j}\rangle$ is the L_2 distance between the embeddings of d_i and d_j, while $\overline{CSV(d_i)}$ is the estimated CSV on example d_i.

Modeling as a meta-learned muti-arm identification problem. Given any target dataset, Equation 14 stays as a linear projection. We model the sampling-based top-k CSV selection process as a Meta-learned Muti-Arm Identification (MMI) problem. Given α k-armed visible bandits $\{MAB_1, MAB_2, \ldots, MAB_\alpha\}$, an invisible bandit MAB, a weight vector $\mathbf{w} \in \mathbb{R}^\alpha$, the aim of MMI is to find an optimal sampling strategy on $\{MAB_1, MAB_2, \ldots, MAB_\alpha\}$ within a fixed budget B, such that the regret possibility e_N in Proposition 7 is minimized.

The BCSV algorithm. We propose an algorithm named BCSV (Algorithm 3), using the ACSV algorithm for each single batch with uniform budget allocation and meta-learning-based sample combination. In Algorithm 3, Lines 1 to 3 describe the offline stage, where Line 2 runs the ACSV algorithm for α times with a uniform budget. In the online stage (Lines 4 to 6), the algorithm uses meta-feature mapping to compute each arm of the similarity expectation of MAB (Line 6), and selects the top-k as the result (Line 7).

Algorithm 3 Batch CSV Approximation (BCSV)

Input: DW record batches $\{D_1, D_2, \ldots\}$, budget B
Output: Demonstrations with Top-K $\widehat{CSV}$
1: **for** $D_i \in D$ **do**
2: $\quad ACSV(D_i, B/\alpha)$
3: $\quad$ **for** $d_j \in D$ **do**
4: $\quad\quad$ compute $\widehat{CSV}_{d_j}$ with Equation 14
5: $\quad$ **end for**
6: **end for**
7: **return** $TopK(\widehat{CSV})$

We analyze the error probability of BCSV as follows.

Proposition 7. *The error probability of BCSV satisfies:*

$$e_N \leq 2\alpha K^2 \exp\left(-\frac{n - \alpha K}{2\alpha \cdot \overline{\log} K \cdot H_\alpha}\right) \tag{15}$$

where $H(i) = \max_{i \in \{1,2,\ldots,n\}} i \cdot (|CSV_{\pi_i} - CSV_{\pi_{i+1}}|)^{-2}$*, and* $H_\alpha = \max_{1 \leq i \leq \alpha} H(i)$.

Algorithm time costs. From Equation 15, we have:

$$2aK^2 \exp\left(-\frac{N - aK}{2a\overline{\log}K \cdot H(a)}\right) = \delta$$

$$N = 2a\overline{\log}K \cdot H(a) \cdot \log\left(\frac{2aK^2}{\delta}\right) + aK$$

This implies that BCSV requires $N = an + 2a\overline{\log}K \cdot H \cdot \log a = O(n)$ offline token cost to obtain comparable error bound with the online scenario.

TABLE II: Data Wrangling Datasets

Type	Dataset	Size	# Attr.	SampleSize
Entity Matching	Fodors-Zagats (FZ)	946	6	68
	iTunes-Amazon (IA)	540	8	98
	Beer	450	4	144
	DBLP-ACM (DA)	12363	4	244
	DBLP-GoogleScholar (DG)	28707	4	54
	Amazon-Google (AG)	11460	3	93
	Walmart-Amazon (WA)	10242	5	215
Data Imputation	Buy	651	4	234
	Restaurant	865	5	254
Error Detection	Adult	11001	13	143
	Hopstital	1001	19	285
Schema Mapping	Synthea	29638	8	50

IV. EXPERIMENTS

We test the effectiveness of our algorithms using **gpt-3.5-turbo** [36] as the LLM to handle four DW tasks including entity matching, error detection, missing value imputation, and schema matching.

A. Experimental Settings

All experiments were run on a Ubuntu 20.04 server with an Intel Xeon (2.50GHz) CPU and 32 GB memory, and a Tesla M40 GPU.

Datasets. We follow a previous study [37] and use the following popular benchmark datasets: (1) the **Magellan** benchmark [38] with seven entity matching datasets; (2) **Adult** and **Hospital** [39] for error detection; (3) **Buy** and **Restaurant** [40] for missing value imputation; and (4) **Synthea** [41] for schema mapping. Each labeled dataset is split into training, validation, and test sets with ratio 3:1:1, following an existing DW study [37]. The details of DW datasets and the corresponding sample size are listed in Table II. Note that the sample size (i.e., number of LLM API calls within budget) is related to not only the size and attributes but also the length of questions after serialization as described in our technical report [21].

Competitors. We compare our algorithms **MCSV**, **ACSV** (with the regret-minimizing sampling strategy), and **BCSV** with five LLM-based algorithms, and four task-specific SOTA algorithms, one for each data wrangling task. The LLM-based competitor algorithms include: (1) **Zero** (zero-shot learning with LLM), which prompts a pre-trained LLM with a task question without examples; (2) **Manual**, which prompts a pre-trained LLM with examples selected by experts [3]; (3) **SC** [3], which starts with a zero-shot LLM-DW at the training stage, clusters wrongly predicted candidate examples with DBSCAN [42], and samples (with probability proportional to the cluster sizes) k examples from the clusters to perform few-shot learning on the testing data; (4) **BatchER** [7] (SOTA LLM-based entity matching algorithm), which runs few-shot learning with the top-k candidate examples chosen as the kNNs for each batch of test EM instances. We adapt BatchER for Schema Mapping, Data Imputation, and Error Detection benchmarks with question prompt templates from [3]; and (5) **CondAcc** [12], [14], which selects the top-k examples using an "influence" metric (detailed in Section V). These algorithms share the same pre-trained LLM with our algorithms, i.e., gpt-3.5-turbo. The task question (detailed in our technical report [21]) used for each DW task is also shared among the algorithms, with the only difference being the examples used.

The task-specific SOTA algorithms (**Task SOTA** in the result tables) include: (1) entity matching: **Ditto** [35], (2) error correction: **Baran** [43], (3) missing value imputation: **IPM** [40], and (4) schema mapping: **SMAT** [41]. These algorithms use highly sophisticated deep learning models (e.g., RoBERTa [31]) specifically tuned for each task. They typically have higher accuracy than LLM-based algorithms on their target tasks, although the LLM-based algorithms are more versatile and can be applied across different tasks.

Evaluation metrics. We report the F1 score for the error detection, schema mapping, and entity matching tasks, and accuracy for the missing value imputation task (where F1 score is irrelevant). As for efficiency evaluation, we report the algorithm running time, Number of Tokens (**NoT**) the LLM takes for input and output, and the API costs the LLM takes.

Parameter setting. As mentioned Section I, our primary goal is to design a fine grained prompt example selection method, e.g., to power example fine-tuning for RAG. Thus, the prompt examples in the our CSV-based algorithms are selected from 20 candidate examples sampled using the RAG-based **SC [3]** algorithm to reduce the LLM API call costs.

By default, we set k as 5, which is an economic choice consistent with the optimal parameter setting in previous LLM prompt example selection studies [3], [7], [12], [14]. Also, we use US$10 per dataset as the example selection budget. This translates to 50 to 285 iterations depending on the example and LLM output size, i.e., we can compute $U()$ with an LLM API call for 50 to 285 times. In each iteration, to save costs, we only ask the LLM to make DW inference on *200 random examples* to produce an estimation of the performance over the full training dataset, following the setting of a previous study [37].

B. End-to-End Performance Results

Comparison against LLM-based algorithms. We first compare our CSV-based algorithms with the LLM-based DW algorithms. ACSV outperforms the two automatic example selection algorithms, Zero and SC on all four DW tasks and almost all datasets, with the only exception being the DA dataset. ACSV also outperforms Manual on average over all four tasks, e.g., 67.27 vs. 52.50 in F1 score for the error detection task. This suggests that our ACSV algorithm can even outperform human expert in the example selection task. ACSV also yields better example selection than CondAcc which is an influence-based algorithm. On all twelve datasets, ACSV has equal or higher F1 (accuracy) scores, where the maximum gap is again observed on the error detection task.

MCSV also outperforms Zero, while it is close to Manual on average. This further verifies the effectiveness of CSV-based example selection. It is outperformed by ACSV, because it is less effective under a budget constrained setting to exploit the sampled candidate example subsets to obtain accurate CSV estimations. BatchER has reported strong results for the entity matching task, because it follows the RAG paradigm to choose the top-k candidate examples as the kNNs for each batch of test instances. Our BCSV algorithm further improves upon BatchER by replacing its intuitive example coverage method via task performance-based top-k CSV selection. As shown, BCSV has F1 scores that are at least as high as those of BatchER, meeting the promise that our CSV-based example selection algorithms can be plugged into existing example selection algorithms to further improve their effectiveness.

Comparison against task SOTA. We further compare with the task SOTA algorithms Ditto, Baran, IPM, and SMAT. We see that ACSV matches the SOTA performance on the missing

TABLE III: Overall algorithm performance results in F1 score and Accuracy, where the *average rank* is computed by taking performance with N/A as 0 (best results are in boldface and second best ones are underlined).

Task	Dataset	Task SOTA				LLM-based					*Ours*		
		Ditto	Baran	IPM	SMAT	Zero	Manual	SC	BatchER	CondAcc	*MCSV*	*ACSV*	*BCSV*
EM (F1)	FZ	**100.00**	N/A	N/A	N/A	93.30	97.97	95.79	**100.00**	**100.00**	95.79	**100.00**	**100.00**
	IA	95.65	N/A	N/A	N/A	62.80	**98.11**	93.61	96.43	94.34	86.30	96.30	96.43
	Beer	94.37	N/A	N/A	N/A	85.81	92.23	92.30	**96.55**	**96.55**	92.85	**96.55**	**96.55**
	DG	**95.60**	N/A	N/A	N/A	64.60	70.44	62.36	83.70	66.67	68.70	75.02	83.70
	DA	**98.99**	N/A	N/A	N/A	93.50	94.90	93.06	94.96	83.87	72.75	86.60	94.96
	AG	**75.58**	N/A	N/A	N/A	54.30	65.40	60.66	62.16	62.16	63.41	65.17	65.40
	WA	**86.76**	N/A	N/A	N/A	72.00	82.63	78.53	80.66	84.21	82.13	85.63	85.63
	average	**92.42**	N/A	N/A	N/A	75.19	85.95	82.33	87.78	83.97	80.27	86.42	88.95
ED(F1)	Adult	N/A	**66.67**	N/A	N/A	0.00	25.00	41.03	39.25	29.83	27.40	47.87	53.80
	Hospital	N/A	**87.00**	N/A	N/A	57.14	80.00	41.38	41.39	50.00	76.00	86.67	**87.00**
	average	N/A	**76.84**	N/A	N/A	28.57	52.50	41.21	40.32	39.92	51.70	67.27	71.40
MVI (Accuracy)	Buy	N/A	N/A	**96.50**	N/A	88.91	89.12	90.47	93.33	90.59	91.25	92.30	93.00
	Restaurant	N/A	N/A	76.90	N/A	79.26	80.14	75.00	66.67	64.10	79.26	**80.87**	**80.87**
	average	N/A	N/A	86.70	N/A	84.09	85.95	82.33	80.00	77.35	85.26	86.68	**86.94**
SM (F1)	Synthea	N/A	N/A	N/A	38.50	0.50	42.86	45.20	45.20	45.20	45.20	**46.37**	**46.37**
DW	*average rank*	5.96	9.38	9.71	10.71	8.80	5.12	7.00	4.50	5.79	6.12	3.33	**2.29**

TABLE IV: Cost results in Time and Number of Tokens.

Metric	Task	Dataset	Task SOTA				LLM-based					*Ours*		
			Ditto	Baran	IPM	SMAT	Zero	Manual	SC	BatchER	CondAcc	*MCSV*	*ACSV*	*BCSV*
Time (s)	EM	FZ	53.86	N/A	N/A	N/A	165.56	166.45	179.75	115.70	7947.04	5367.15	1534.71	643.22
		IA	26.33	N/A	N/A	N/A	95.70	101.05	128.95	41.63	2875.14	10299.50	1251.98	686.50
		Beer	53.45	N/A	N/A	N/A	86.79	78.23	97.40	43.83	1188.60	1755.71	795.32	191.98
		DG	3052.18	N/A	N/A	N/A	180.02	179.52	202.37	3023.33	24173.04	10233.02	2475.58	525.20
		DA	1187.23	N/A	N/A	N/A	170.49	178.97	200.06	1361.22	18853.12	7697.81	3524.38	732.51
		AG	251.68	N/A	N/A	N/A	181.62	189.99	219.87	1214.32	17235.38	15328.93	2774.26	523.02
		WA	577.30	N/A	N/A	N/A	183.40	177.56	195.24	2175.20	15271.83	12633.89	2493.30	539.90
	ED	Adult	N/A	247.12	N/A	N/A	164.39	168.75	179.46	6600.00	40797.04	7480.65	1376.55	645.48
		Hospital	N/A	23.09	N/A	N/A	145.7	166.07	181.33	1132.78	25359.93	15129.05	2871.44	771.18
	MVI	Buy	N/A	N/A	184.86	N/A	97.74	94.06	104.86	2740.98	50762.01	2187.81	1229.91	267.67
		Restaurant	N/A	N/A	200.92	N/A	124.99	118.99	135.90	369.83	20466.31	4301.82	1053.36	632.88
	SM	Synthea	N/A	N/A	N/A	335.02	151.77	153.87	170.66	1837.74	46746.29	12583.38	3630.98	766.82
	average		743.15	135.11	192.89	335.02	158.92	161.23	181.44	1877.87	24697.79	9545.33	2273.80	629.67
NoT	EM	FZ	N/A	N/A	N/A	N/A	20626	136906	136801	31576	198643	983037	324890	167989
		IA	N/A	N/A	N/A	N/A	15037	51007	51229	10257	138784	603996	142170	77526
		Beer	N/A	N/A	N/A	N/A	4777	19999	25042	10980	99456	217428	55739	31746
		DG	N/A	N/A	N/A	N/A	18414	202551	217663	1123960	2528464	1916785	767133	267005
		DA	N/A	N/A	N/A	N/A	21026	195148	200978	534488	1350142	2692517	943799	269672
		AG	N/A	N/A	N/A	N/A	12829	99204	99809	250272	1033673	982065	202625	135874
		WA	N/A	N/A	N/A	N/A	13864	136864	141178	279657	1038156	911210	291547	182408
	ED	Adult	N/A	N/A	N/A	N/A	15167	85119	87092	3643811	2612639	1448106	568115	236313
		Hospital	N/A	N/A	N/A	N/A	2681	14858	15768	83839	170932	207025	51344	43099
	MVI	Buy	N/A	N/A	N/A	N/A	5346	52146	56090	201617	145350	526219	158831	71815
		Restaurant	N/A	N/A	N/A	N/A	5902	18226	18903	105327	254502	244986	185285	62081
	SM	Synthea	N/A	N/A	N/A	N/A	5161	21844	22978	323819	9526258	2044651	329655	80669
	average		N/A	N/A	N/A	N/A	11736	86156	89461	549967	1591417	1064835	335094	135810

value imputation task, while it even outperforms the SOTA on the schema mapping task. For the entity matching and error detection tasks, the SOTA algorithms are better, for their specifically tuned models as mentioned above. Note that the task SOTA algorithms are data hungry and are computationally expensive. Even in these tasks, ACSV performs just as well or even better than the SOTA on some of the datasets (e.g., IA and Beer). These results show the strong potential of an LLM-based solution for the DW tasks. Regarding the average rank, BCSV and ACSV are ranked significantly higher than all the others including the second best method, Manual. Task SOTA methods fail in the ranking as each of them can only be adapted for one specific DW task.

Critical difference diagram. With statistical level of 0.1, we run a critical difference diagram based on Wilcoxon test in Figure 3. Notably, on all data wrangling tasks, ACSV outperforms the Manual method by 1.5 in terms of average rank. Some task SOTA appears to be even worse than MCSV, as they are severely affected by the DW tasks that they are not designed for scenarios where the performance is 0. We also ran 4 extra Wilcoxon tests on each different DW task and record the average ranking. Due to space limit, we put the critical difference diagram of these extra Wilcoxon test in our technical report [21]. The results show that BCSV

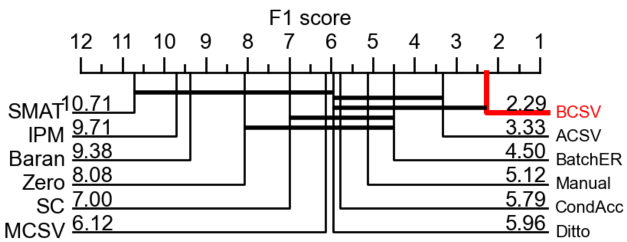


Fig. 3: Critical difference diagram on the F1 score of all data wrangling tasks under the statistical level of 0.1.

TABLE V: F1 score Comparison of ACSV and SC with Llama2-70b (‘L’), gpt-4o-mini(‘g4’) and gpt-3.5-turbo (‘g35’) on the entity matching datasets.

Dataset	FZ	IA	Beer	DG	DA	AG	WA
SC (L)	52.78	73.17	49.05	29.25	54.79	51.63	70.58
SC (g35)	95.79	93.61	92.30	62.36	93.06	60.66	78.53
SC (g4)	87.80	94.33	88.94	76.19	96.48	55.69	83.51
ACSV(L)	78.57	81.82	71.79	56.90	94.49	55.25	75.00
ACSV(g35)	100.00	96.30	96.55	75.02	86.60	65.17	85.63
ACSV(g4)	100.00	98.18	92.30	81.82	96.58	67.69	86.25

is comparable to Task SOTA, and it outperforms BatchER significantly. ACSV also approaches BatchER, while both ACSV and BCSV are much better than the naive adaption of marginal contribution, MCSV.

C. Cost Analysis and Discussion

Convergence. CSV is an anytime algorithm, such that given any API cost/Number of Tokens/Time budget, the algorithm can stop at that budget and return a feasible approximate solution. Proposition 2, 5, and 6 ensure that the upper bound of error probability decreases monotonically with increases in the budget. Therefore, we focus on the effectiveness instead of the cost above. Nevertheless, we observe that CSV converges to optimal performance on DW tasks with a few samples. In such sense, we report time costs and number of tokens to reach peak performance to show the applicability of our method.

Cost analysis. We first report the time to reach peak performance. BCSV is faster than ACSV by over $3.6\times$ and MCSV by $15.2\times$ due to sample sharing and meta-learning. We highlight that LLM APIs can answer multiple questions simultaneously. Such time can be further accelerated via parallel LLM QA techniques, which is worth another study, and we leave it as future work. In such sense, a more sensible comparison would be on the NoT costs of the LLM-DW algorithms.

Table IV shows that, on average, the NoT of BCSV is $2\times$ smaller than that of Manual, attributing to example sharing and meta-learning. Note that NoT within 3.3 million are not a concern. All reported NoT is far below our \$10 budget, as gpt-3.5-turbo costs \$3 for 1M tokens, while the latest gpt-4o-mini is even $25\times$ cheaper.

D. Ablation Study

Impact of budget. It’s notable that we proposed multiple sampling methods in this paper, so we also conduct the experiments to verify the design choice of CSV sampling methods. Our results are shown in Figure 4. Due to space limit, results for tasks other than entity matching in our technical report [21]. As we can see, BCSVhas better performance compared to the other two methods, MCSVand ACSV. This suggests that by more careful allocation of samples, it is possible for CSV to converge at higher performance within low cost. CSV works better when applied on each of question batches individually, and the stratified sampling converges faster than other sampling approaches, which aligns with our complexity analysis.

Impact of LLMs. To show the impact of the LLMs used, we further apply our ACSValgorithm with an open-source LLM, Llama-2-70b. For comparison purposes, we also run SC with Llama-2-70b. We use SC as a comparison since ACSV uses the output of SC as input. We report the F1 scores for entity matching tasks in Table V, and further results on other data wrangling tasks are in our technical report [21]. We see that Llama-2-70b in general is less effective than gpt-3.5-turbo for the entity matching task. Importantly, ACSV using Llama-2-70b also outperforms SC using the same LLM, which again verifies the effectiveness of our algorithm. We conducted further experiments on gpt-4o-mini as shown below. We report accuracy for MVI and F1 socre for other DW tasks. As shown, our algorithm ACSV also outperforms SC by a considerable margin in most cases.

V. Related Work

There are three lines of research closely related to our work.

Data wrangling: We give a brief discussion on entity matching task below and refer interested readers to [44] for a detailed survey on data wrangling.

Entity matching (EM) is a long-standing challenge in data integration, information retrieval, and natural language processing [1]. EM solutions started from declarative similarity-based matching rules, either pre-defined or synthesized. Using entity attribute similarity as the feature, machine learning models was later introduced to EM [45], [46]. Such models are data hungry and may become inapplicable without labeled training data. Crowd-sourcing [47], [48] and active learning [49], [50] are exploited to reduce the manual data labeling costs. More recently, advanced machine learning techniques such as domain adaption [51], [52] are proposed to address the data annotation cost issues. Another thread of studies addresses the challenges in measuring attribute similarity over long and complex textual entity attributes. Pre-trained language model (PLM)-based solutions [2], [53]–[55] are the state-of-the-art in this category, capturing textual attributes with text embeddings. With the rise of large language models (LLMs), fine tuning [56] or prompting [7] LLMs have become a new EM paradigm, where in-context learning [6], [12]–[14] is more widely adopted as a computational friendly prompting solution. A piece that is still missing in these studies is how to choose the examples used to guide the LLMs for in-context learning. Our study fills this gap.

In-context example selection. In-context learning is a learning paradigm for LLMs that guides LLMs to generate

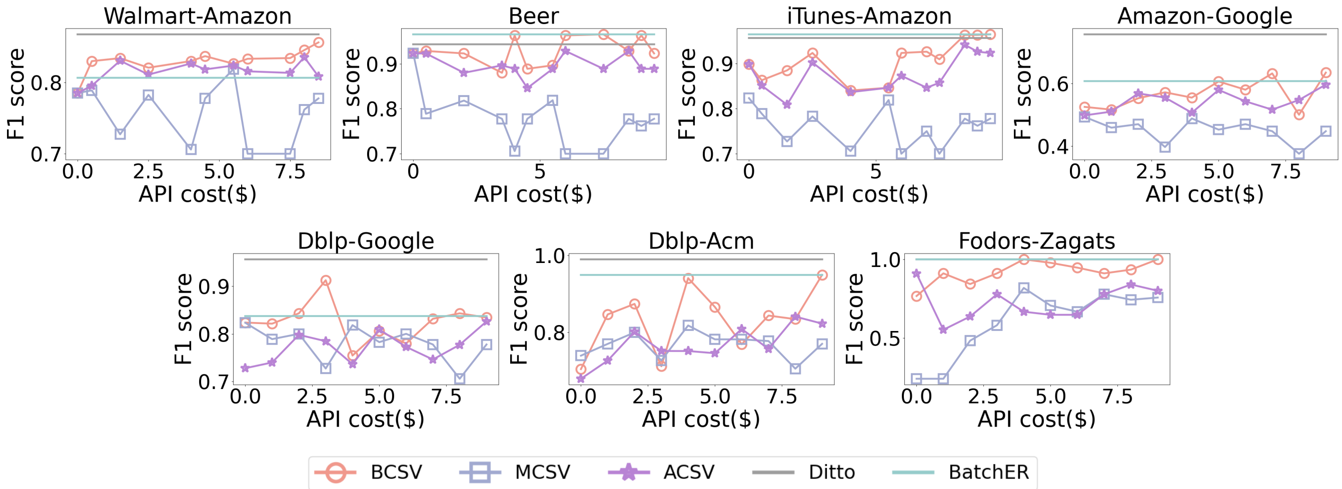


Fig. 4: Impact of LLM API call costs on F1 score (on the entity matching datasets).

answers based on the input context (typically a few task examples [6]) beyond just a question. A key issue here is how to select the task examples, i.e., the *demonstration engineering* problem [3].

Retrieval augmented generation (RAG) is a popular strategy, where candidate examples that are most relevant (e.g., kNNs in some embedding space) to the task question are selected as the task examples [8], [57]. This strategy was used in a baseline algorithm BatchER [7] compared with in our experiments.

Influence-based methods [12]–[14] selects top-k examples using an "influence" metric. The influence of an example d_i, denoted by $inf(d_i)$, is the gap between the average utility with and without d_i:

$$inf(d_i) = \sum_{S \subset D \setminus \{d_i\}, |S| \le K, d_i \in S} \frac{U(S)}{N-M} - \sum_{S \subset D \setminus \{d_i\}, |S| \le K, d_i \notin S} \frac{U(S)}{M} \tag{16}$$

Here, N is the total samples, while M is the samples without d_i, and $|S| = M$. This definition resembles CSV with differences in coefficient according to the total number size. It yielded strong accuracy results on natural language processing tasks such as sentiment analysis and intent detection, while its performance is subject to sample quality [13], [58].

Initial studies [37], [56] on LLM prompting for EM focus on handling probabilistic matching [56] or question & demonstration batching [37]. The former study does not concern in-context learning, while the latter uses RAG with heuristic-based similarities or embedding-based distances for example selection. These studies aim to reduce LLM API costs like we do, while we introduce a Shapley value-based example selection process, improving both the effectiveness and explainability of LLM-based EM.

Shapley value. *Shapley value* (SV) is a contribution evaluation metric from the cooperative game theory [59]. Due to its balance, symmetry, additivity, and zero element properties [59], SV has been widely adopted by the data management community [60]. It has also been used as a data valuation method to decide the contribution and price of data points, queries, or even prompts in data market applications [24], [61]–[65]. Another application is in model explainability for machine learning models, also known as feature selection or data debugging [16], [20], [66], [67].

Exact SV computation is known to be #P-hard [68], and hence much effort has been spent on reducing its computational costs. Random permutation [69] and stratified sampling allocation [70] are the commonly adopted efficient solutions, reporting promising efficiency results on tasks such as data debugging and detecting and training data with negative impacts in LLM models [20] and machine learning pipelines [16].

There are a few concepts similar to our constrained Shapley value, such as the *affine transformation* on the SV [71] and the *influence* as described above. These two concepts approximate a biased estimation of CSV, which do not satisfy the symmetry and additivity properties, and may lead to selecting examples with negative impacts.

VI. Conclusion

We proposed the constrained Shapley value (CSV), a technique that enables efficient evaluation of the impact of different examples on the effectiveness of in-context learning for LLMs over data wrangling tasks. CSV has attractive properties using reward allocation to guide candidate example selection, while it is computational intractable. To reduce the costs of example selection for LLMs, we further proposed to compute approximate top-k CSV, with an algorithm named ACSV that has costs linear to the size of the set of candidate examples. Experimental results on four data wrangling tasks over commonly used benchmark datasets show that LLMs using our ACSV algorithm for example selection yields higher F1 scores than those using SOTA LLM-based algorithms. Even comparing with specialised models tuned for each different task, combining our algorithms with LLM yields comparable results in general, and better results over a number of the datasets tested.

References

[1] L. Getoor and A. Machanavajjhala, "Entity resolution: theory, practice & open challenges," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 2018–2019, 2012.

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

[3] A. Narayan, I. Chami, L. J. Orr, and C. R'e, "Can foundation models wrangle your data?" *Proceedings of the VLDB Endowment*, vol. 16, no. 4, pp. 738–746, 2022.

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

[5] A. Doan, "Deepmatcher datasets," 2024, accessed: 2024-11-26. [Online]. Available: https://github.com/anhaidgroup/deepmatcher/blob/master/Datasets.md#beeradvo-ratebeer

[6] Q. Dong, L. Li, D. Dai, C. Zheng, J. Ma, R. Li, H. Xia, J. Xu, Z. Wu, B. Chang, X. Sun, L. Li, and Z. Sui, "A survey on in-context learning," in *EMNLP*, 2024.

[7] M. Fan, X. Han, J. Fan, C. Chai, N. Tang, G. Li, and X. Du, "Cost-effective in-context learning for entity resolution: A design space exploration," in *ICDE*, 2024.

[8] J. Liu, D. Shen, Y. Zhang, B. Dolan, L. Carin, and W. Chen, "What makes good in-context examples for GPT-3?" in *Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, 2022.

[9] E. Tanwar, M. Borthakur, S. Dutta, and T. Chakraborty, "Multilingual LLMs are better cross-lingual in-context learners with alignment," in *ACL*, 2023.

[10] T. Sorensen, J. Robinson, C. Rytting, A. Shaw, K. Rogers, A. Delorey, M. Khalil, N. Fulda, and D. Wingate, "An information-theoretic approach to prompt engineering without ground truth labels," in *ACL*, 2022.

[11] X. Wang, W. Zhu, M. S. Saxon, and W. Y. Wang, "Large language models are latent variable models: Explaining and finding good demonstrations for in-context learning," in *NeurIPS*, 2023.

[12] T. Nguyen and E. Wong, "In-context example selection with influences," *arXiv preprint abs/2302.11042*, 2023.

[13] Y. Zhang, S. Feng, and C. Tan, "Active example selection for in-context learning," in *EMNLP*, 2022.

[14] T.-Y. Chang and R. Jia, "Data curation alone can stabilize in-context learning," in *ACL*, 2022.

[15] OpenAI, "OpenAI API pricing," 2024, accessed: 2024-11-26. [Online]. Available: https://openai.com/api/pricing/

[16] B. Karlavs, D. Dao, M. Interlandi, B. Li, S. Schelter, W. Wu, and C. Zhang, "Data debugging with Shapley importance over end-to-end machine learning pipelines," in *ICLR*, 2024.

[17] H. Liu, X. Mao, H. Xia, J. Lou, and J. Liu, "Prompt valuation based on Shapley values," *arXiv preprint abs/2312.15395*, 2023.

[18] L. S. Shapley, "A value for n-person games," in *Classics in Game Theory*. Princeton University Press, 1953, pp. 307–317.

[19] OpenAI, "ChatGPT (March 14 version)," 2023, accessed: 2024-11-26. [Online]. Available: https://chat.openai.com/chat

[20] R. Jia, D. Dao, B. Wang, F. A. Hubis, N. M. Gürel, B. Li, C. Zhang, C. J. Spanos, and D. X. Song, "Efficient task-specific data valuation for nearest neighbor algorithms," *Proceedings of the VLDB Endowment*, vol. 12, no. 11, pp. 1610–1623, 2019.

[21] Z. Liang, H. Wang, X. Ding, Z. Liang, C. Liang, and J. Qi, "Technical report for CSV," Harbin Institute of Technology, Tech. Rep., 2024. [Online]. Available: https://github.com/Lorenzo0224/csv/blob/main/CSV_ICDE_techrept.pdf

[22] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *NeurIPS*, 2017.

[23] M. Mahdavi, Z. Abedjan, R. C. Fernandez, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang, "Raha: A configuration-free error detection system," in *SIGMOD*, 2017.

[24] W. Zhou, H. Adel, H. Schuff, and N. T. Vu, "Explaining pre-trained language models with attribution scores: An analysis in low-resource settings," in *LREC-COLING*, 2024.

[25] J. Castro, D. Gmez, and J. Tejada, "Polynomial calculation of the Shapley value based on sampling," *Computers and Operations Research*, vol. 36, no. 5, pp. 1726–1730, 2009.

[26] S. Maleki, "Addressing the computational issues of the Shapley value with applications in the smart grid," Ph.D. dissertation, University of Southampton, 2015.

[27] Z. Chen, Q. Chen, B. Hou, Z. Li, and G. Li, "Towards interpretable and learnable risk analysis for entity resolution," in *SIGMOD*, 2020.

[28] J. Zhang, Q. Sun, J. Liu, L. Xiong, J. Pei, and K. Ren, "Efficient sampling approaches to Shapley value approximation," *Proceedings of the ACM on Management of Data*, vol. 1, no. 1, pp. 1–24, 2023.

[29] P. Kolpaczki, V. Bengs, and E. Hüllermeier, "Identifying top-k players in cooperative games via Shapley bandits," in *Lernen, Wissen, Daten, Analysen*, 2021.

[30] S. Bubeck, T. Wang, and N. Viswanathan, "Multiple identifications in multi-armed bandits," in *ICML*, 2013.

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

[32] S. Barhom, V. Shwartz, A. Eirew, M. Bugert, N. Reimers, and I. Dagan, "Revisiting joint modeling of cross-document entity and event coreference resolution," in *ACL*, 2019.

[33] H. Nie, X. Han, B. He, L. Sun, B. Chen, W. Zhang, S. Wu, and H. Kong, "Deep sequence-to-sequence entity matching for heterogeneous entity resolution," in *CIKM*, 2019.

[34] T. Mu, H. Wang, C. Wang, Z. Liang, and X. Shao, "Auto-CASH: A meta-learning embedding approach for autonomous classification algorithm selection," *Information Sciences*, vol. 591, pp. 344–364, 2022.

[35] P. Wang, W. Zheng, J. Wang, and J. Pei, "Automating entity matching model development," in *ICDE*, 2021.

[36] OpenAI, "Chatgpt (gpt-3.5-turbo)," 2023, accessed: 2024-11-26. [Online]. Available: https://www.openai.com

[37] H. Li, L. Feng, S. Li, F. Hao, C. J. Zhang, Y. Song, and L. Chen, "On leveraging large language models for enhancing entity resolution," in *ICDE*, 2024.

[38] P. Konda, S. Das, A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. Naughton, S. Prasad *et al.*, "Magellan: toward building entity matching management systems over data science stacks," *Proceedings of the VLDB Endowment*, vol. 9, no. 13, pp. 1581–1584, 2016.

[39] A. Heidari, J. McGrath, I. F. Ilyas, and T. Rekatsinas, "HoloDetect: Few-shot learning for error detection," in *SIGMOD*, 2019.

[40] Y. Mei, S. Song, C. Fang, H. Yang, J. Fang, and J. Long, "Capturing semantics for imputation with pre-trained language models," in *ICDE*, 2021.

[41] J. Zhang, B. Shin, J. D. Choi, and J. Ho, "SMAT: An attention-based deep learning solution to the automation of schema matching," in *ADBIS*, 2021.

[42] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *KDD*, 1996.

[43] M. Mahdavi and Z. Abedjan, "Baran: Effective error correction via a unified context representation and transfer learning," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 1948–1961, 2020.

[44] T. Furche, G. Gottlob, L. Libkin, G. Orsi, and N. Paton, "Data wrangling for big data: Challenges and opportunities," in *EDBT*, 2016.

[45] S. Chaudhuri, B.-C. Chen, V. Ganti, and R. Kaushik, "Example-driven design of efficient record matching queries." in *VLDB*, 2007.

[46] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom, "Swoosh: a generic approach to entity resolution," *The VLDB Journal*, vol. 18, pp. 255–276, 2009.

[47] C. Chai, G. Li, J. Li, D. Deng, and J. Feng, "A partial-order-based framework for cost-effective crowdsourced entity resolution," *The VLDB Journal*, vol. 27, pp. 745–770, 2018.

[48] S. Das, P. S. G. C., A. Doan, J. F. Naughton, G. Krishnan, R. Deep, E. Arcaute, V. Raghavendra, and Y. Park, "Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services," in *SIGMOD*, 2017.

[49] R. Wu, S. Chaba, S. Sawlani, X. Chu, and S. Thirumuruganathan, "ZeroER: Entity resolution using zero labeled examples," in *SIGMOD*, 2020.

[50] V. V. Meduri, L. Popa, P. Sen, and M. Sarwat, "A comprehensive benchmark framework for active learning methods in entity matching," in *SIGMOD*, 2020.

[51] J. Tu, J. Fan, N. Tang, P. Wang, C. Chai, G. Li, R. Fan, and X. Du, "Domain adaptation for deep entity resolution," in *SIGMOD*, 2022.

[52] J. Tu, X. Han, J. Fan, N. Tang, C. Chai, G. Li, and X. Du, "DADER: hands-off entity resolution with domain adaptation," *Proceedings of the VLDB Endowment*, vol. 15, no. 12, pp. 3666–3669, 2022.

[53] R. Peeters and C. Bizer, "Dual-objective fine-tuning of BERT for entity matching," *Proceedings of the VLDB Endowment*, vol. 14, no. 10, pp. 1913–1921, 2021.

[54] U. Brunner and K. Stockinger, "Entity matching with transformer architectures-a step forward in data integration," in *EDBT*, 2020.

[55] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, and N. Tang, "Distributed representations of tuples for entity resolution," *Proceedings of the VLDB Endowment*, vol. 11, no. 11, pp. 1454–1467, 2018.

[56] D. Vos, T. Döhmen, and S. Schelter, "Towards parameter-efficient automation of data wrangling tasks with prefix-tuning," in *NeurIPS Table Representation Workshop*, 2022.

[57] O. Rubin, J. Herzig, and J. Berant, "Learning to retrieve prompts for in-context learning," in *NAACL*, 2022.

[58] Y. Lu, M. Bartolo, A. Moore, S. Riedel, and P. Stenetorp, "Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity," in *ACL*, 2022.

[59] S. Schoch, H. Xu, and Y. Ji, "CS-Shapley: Class-wise Shapley values for data valuation in classification," in *NeurIPS*, 2022.

[60] L. E. Bertossi, B. Kimelfeld, E. Livshits, and M. Monet, "The Shapley value in database management," *ACM SIGMOD Record*, vol. 52, no. 2, pp. 6–17, 2023.

[61] J. Zhang, H. Xia, Q. Sun, J. Liu, L. Xiong, J. Pei, and K. Ren, "Dynamic Shapley value computation," *ICDE*, 2023.

[62] H. Xia, J. Liu, J. Lou, Z. Qin, K. Ren, Y. Cao, and L. Xiong, "Equitable data valuation meets the right to be forgotten in model markets," *Proceedings of the VLDB Endowment*, vol. 16, no. 11, pp. 3349–3362, 2023.

[63] A. Agarwal, M. A. Dahleh, and T. Sarkar, "A marketplace for data: An algorithmic solution," in *ACM Conference on Economics and Computation*, 2019.

[64] L. Chen, P. Koutris, and A. Kumar, "Model-based pricing for machine learning in a data marketplace," in *SIGMOD*, 2019.

[65] J. Liu, J. Lou, J. Liu, L. Xiong, J. Pei, and J. Sun, "Dealer: An end-to-end model marketplace with differential privacy," *Proceedings of the VLDB Endowment*, vol. 14, no. 6, pp. 957–969, 2021.

[66] R. H. L. Sim, X. Xu, and K. H. Low, "Data valuation in machine learning: "ingredients", strategies, and open challenges," in *IJCAI*, 2022.

[67] Z. Zhou, X. Xu, R. H. L. Sim, C.-S. Foo, and K. H. Low, "Probably approximate Shapley fairness with applications in machine learning," in *AAAI*, 2022.

[68] X. Deng and C. H. Papadimitriou, "On the complexity of cooperative solution concepts," *Mathematics of Operations Research*, vol. 19, no. 2, pp. 257–266, 1994.

[69] J. Castro, D. Gómez, and J. Tejada, "Polynomial calculation of the Shapley value based on sampling," *Computers & Operations Research*, vol. 36, no. 5, pp. 1726–1730, 2009.

[70] J. Neyman, "On the two different aspects of the representative method: the method of stratified sampling and the method of purposive selection," in *Breakthroughs in Statistics: Methodology and Distribution*. Springer New York, 1992, pp. 123–150.

[71] J. Derks and H. Peters, "A Shapley value for games with restricted coalitions," *International Journal of Game Theory*, vol. 21, pp. 351–360, 1993.