CENTS: A Flexible and Cost-Effective Framework for LLM-Based Table Understanding

Guorui Xiao
University of Washington
grxiao@cs.washington.edu

Dong He
University of Washington
donghe@cs.washington.edu

Jin Wang
Arizona State University
jinwang18@asu.edu

Magdalena Balazinska
University of Washington
magda@cs.washington.edu

ABSTRACT

Large Language Models (LLMs) have recently shown impressive capabilities in a variety of applications including table understanding tasks such as column type annotation. Existing LLM-based solutions for table understanding, however, focus on developing specific framework for each individual task, or do not consider the cost-effectiveness tradeoff. In this paper, we present **CENTS**, a unified and cost-effective framework for LLM-based solutions for table understanding tasks. **CENTS**'s key capability is an efficient and effective approach to compress the tabular LLM input in a way that reduces input token cost while improving performance compared with state-of-the-art methods. Experiment results show that **CENTS** outperforms other LLM-based baselines on a variety of table understanding tasks at the same or lower cost.

PVLDB Reference Format:
Guorui Xiao, Dong He, Jin Wang, and Magdalena Balazinska. **CENTS**: A Flexible and Cost-Effective Framework for LLM-Based Table Understanding. PVLDB, 18(11): 4574 - 4587, 2025.
doi:10.14778/3749646.3749714

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/grxiao137/CENTS.

1 INTRODUCTION

Structured Web tables, which are relations extracted from the Web, have long been recognized as an important data source. Many approaches have been developed to construct large-scale table corpora [3, 7, 15, 24, 51] and use Web tables in data management tasks including data discovery [1, 10], data integration [21, 43] and question answering [52].

Web tables, however, frequently lack headers describing their semantic content, limiting or at least complicating their usefulness for these tasks. To address this limitation, the field of *Table Understanding* has developed techniques to complete the missing header information through approaches that include Column Type Annotation [3, 7, 47, 50, 51] (a.k.a, labeling a column with its semantic type, such as "ticketPrice" or "itemWeight"), Relation Extraction [7, 32, 39] (a.k.a., annotating a pair of columns with their semantic relation such as "ratingOf" or "employeeOf"), or Schema Augmentation [7] (a.k.a., proposing additional columns for a table such as "Hall Capacity"). Table Understanding has a long history with early approaches using manually created feature-based solutions [16, 49, 51], more recent methods fine-tuning Pre-trained Language Models (PLMs) [7, 32, 39, 42], and most recent state-of-the-art methods leveraging Large Language Models (LLMs) [11, 20, 22, 34, 45].

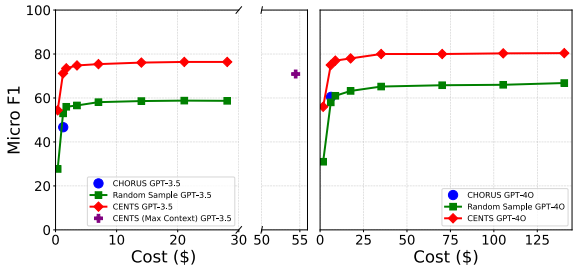


Figure 1: Cost-effectiveness trade-off for LLM-based table understanding methods. Contextual information (e.g., table vs single-column) enhances performance but increases costs. CENTS with GPT-3.5, outperforms random sampling with GPT-4O, while incurring only 1.25% of its cost. CENTS with a limited budget also outperforms CENTS (Max Context), which uses the full context window. Dollar cost computed as product of total token counts and per-token API price.

LLMs are auto-regressive models pre-trained on very large text corpora that can generate a sequence of tokens as output based on input to the model, which is also known as a *prompt*. Recent approaches to LLM-based table understanding [11, 20, 22, 34, 45] have developed effective prompting strategies to support various table understanding tasks using LLMs: They construct prompts for the LLM with the provided table or partial table (e.g., a column) together with instructions that describe the task.

Existing LLM-based solutions, however, have important limitations related to their cost-effectiveness and, in some instances, their lack of generalizability: Some existing LLM-based methods provide the entire table to an LLM in the prompt without considering the associated cost [22], which can be substantial. For example, tables in the SOTAB dataset [23] range in size from 200 tokens to over 14 million tokens. Submitting a single such table to an LLM like GPT-4O in its prompt can thus cost up to $35 (USD). Executing

Proceedings of the VLDB Endowment, Vol. 18, No. 11 ISSN 2150-8097.
doi:10.14778/3749646.3749714

one table understanding query over the entire SOTAB benchmark by including the entire table in the prompt each time costs over $750 (USD). [1] Such a high cost impacts the practical application of these approaches. Additionally, LLMs have a fixed context window limit [2] (e.g., 16,385 tokens for GPT-3.5-turbo). Tables larger than this limit need to be reduced. To reduce costs, other LLM-based methods assume that including only the target columns in a prompt is sufficient [11, 20], but recent table understanding solutions [10, 22] have shown that providing full table information improves performance. These methods thus sacrifice performance to save on cost. Interestingly, as shown in other settings [25, 30] and as we confirm in this paper, judiciously selecting the context to provide to an LLM, can increase F1 scores compared with using the full context window (Figure 1). A second challenge of some LLM-based methods is their lack of general applicability, when solutions focus on task-specific frameworks [11, 22], which can only answer one type of table understanding query.

In this paper, we design, implement, and evaluate **CENTS** [2], a cost-effective framework for table understanding tasks. **CENTS** addresses the above challenges by reducing the cost of LLM-based solutions for table understanding tasks compared to those that submit entire tables in their prompts, while maintaining and even exceeding the result quality of state-of-the-art methods. **CENTS** is designed primarily with hosted LLMs in mind, given their high performance and high costs, but we also discuss **CENTS**'s applicability to locally-deployed LLMs. **CENTS** is also generally applicable: It is not specialized for any one table understanding task, but is applicable to various such tasks. Figure 1 illustrates **CENTS**'s cost-effectiveness. Compared with the CHORUS [20] state-of-the-art method or including randomly selected subsets of a table in the prompt, **CENTS** significantly improves performance.

As discussed above, one main driver of cost in LLM-based table understanding is the inclusion in the prompt of the table to be annotated (also referred to as the *Context Data*, discussed in detail in Section 3). Our approach, which applies to a variety of table understanding tasks, is to *carefully select a subset of the Context Data* using a *score-and-solve* paradigm. In this paradigm, **CENTS** first assigns scores to elements in the *Context Data* and then carefully selects a subset of these elements given a budget. As part of **CENTS**, we develop (1) effective element scoring strategies; (2) effective and efficient element selection strategies; and (3) effective budget allocation strategies. **CENTS** can consider tables at different levels of granularity (rows, cells, or words in a cell), and (4) we experimentally evaluate which granularity leads to the best performance. Developing these four strategies raises two important challenges:
(1) Effectively reducing costs. One challenge is to reduce *Context Data*, a main part of the prompt. An effective reduction requires identifying the most important content in a table. Therefore, developing a scoring strategy that accounts for both the context within target columns and their surrounding columns and integrates syntactic and semantic information is essential yet challenging.
(2) Improving Execution Throughput. The other challenge is efficiently executing the data reduction. As we show in this paper, high-quality data reduction can easily lead to slow execution speeds. Even approximation techniques remain too slow. To address this challenge and ensure **CENTS** is a practical tool, we develop an efficient approximation solution that accelerates the data reduction process with minimal impact on the end-to-end performance.

Another challenge arises when the number of labels that LLMs must choose from (also referred to as the *Task Data*) is large, and the search space—the set of possible labels—is randomly distributed within the prompt. A large and scattered *Task Data* typically degrades the performance of LLM-based applications in general [30, 37]. Thus, we provide an optimization to reduce the search space and rerank the reduced *Task Data* to address this challenge. An additional benefit this brings is also the reduced monetary cost.

Although there are some efforts [4, 18, 53] in the machine learning field that aim to reduce inference costs of LLMs, they do not directly benefit table understanding tasks and thus are orthogonal to our work. We further discuss related works in Section 7.

In summary, this paper makes the following main contributions:

(1) We identify the problem of reducing prompt content specifically for LLM-based solutions to table understanding tasks to lower costs while achieving high performance and propose our *unified* and *cost-effective* framework, **CENTS** that works across a variety of such tasks.
(2) We design the context reduction component with *score-and-solve* paradigm, which consists of an effective scoring method and an efficient solver method; we further design the task data reduction component to effectively reduce a possibly large search space as an optimization.
(3) We provide various options within the *score-and-solve* paradigm, namely different scoring granularities and budget allocation strategies. We discuss their trade-offs and select the best-performing combinations.
(4) We conduct experiments using **CENTS** on popular benchmarks and show that it outperforms LLM-based baselines by up to 11.2 Micro F1, 15 Micro F1, and 17.5 Mean Average Precision across three table understanding tasks.

Overall **CENTS** is an important step in making LLM-based table understanding a practical and cost-effective approach.

2 BACKGROUND

Table Understanding Tasks. We consider three table understanding tasks, illustrated in Figure 2. While we evaluate **CENTS** on these tasks, **CENTS** can conceptually be applied to other tasks as well. Column Type Annotation (CTA) [7, 32, 39] assigns one or more semantic labels to a column (see example in Figure 2). Relation Extraction (RE) [7, 32, 39] assigns one or more *semantic relationships* between columns within the same table. Schema Augmentation (SA) [3, 7, 47, 50, 51] aims to augment a partial table by *recommending* new columns.
Large Language Models (LLMs). Recently, Large Language Models (LLMs) [40] have shown promising results for many data management tasks. For Table Understanding, state-of-the-art LLM prompts have three parts: (1) ***Instruction*** I (e.g., "Annotate the column...") that specify the task the LLM needs to complete; (2) A relational table, ***Context Data*** $T(E, H)$, with n columns and m rows. The bag of all $n * m$ cells is denoted with $E = (e_{11}, \ldots, e_{ij}, \ldots, e_{nm})$. The

[1] Cost calculated by multiplying the aggregated token counts for the entire benchmark by the per-token price from https://openai.com/api/pricing/.
[2] *C, E* in **CENTS** stands for Cost Effective; the name together means doing table understanding with little cost.

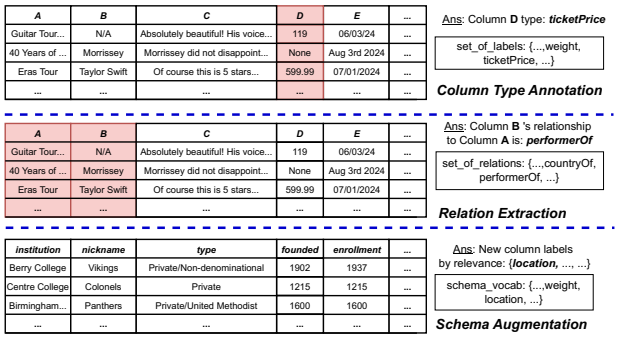


Figure 2: Table understanding tasks CENTS supports. Column type annotation labels column D with type "*ticketPrice*" from the given set of labels; relation extraction annotates the relationship between columns B and A as "*performerOf*"; schema augmentation ranks the given schema vocabulary from most to least relevant to augment the existing table, and in this case "*location*" is ranked first.

table may *optionally* have a header, H, with column labels, but not necessarily the semantic types. For CTA and RE tasks, the header is typically missing [23], while it's available for SA tasks [7]. Thus in this paper, the *Context Data* for CTA and RE is just the bag of cells E while *Context Data* for SA contains both E and H, and we further assume that no other metadata is given (e.g., no domain information, no caption) [23]; (3) ***Task Data*** L that consists of task-specific labels that the LLM should choose from. In this paper, L denotes different sets of values for three tasks: In CTA, L is the set of possible semantic types (e.g., $\{\ldots, \text{weight}, \text{ticketPrice}, \ldots\}$ in Figure 2); in RE it is the set of possible semantic relationships between columns (e.g., $\{\ldots, \text{contryOf}, \text{performerOf}, \ldots\}$); and in SA it is the set of new column headers to recommend. Figure 4 shows an example prompt for CTA. Table 1 summarizes the notation.

3 PROBLEM STATEMENT

Formal Table Understanding Task Definitions.

Column Type Annotation. Given a table T without a header, a column A in T to be annotated, and a set of semantic type labels L, *CTA* aims to find a semantic type from L such that all cells in A have that type.

Relation Extraction. Given a table T without a header, a pair of columns S and O in T, and a set of semantic relationship labels L, *RE* aims to find a semantic relationship label from L such that (S, O) have that semantic relationship.

Schema Augmentation. Given a table T and a schema vocabulary list L, *SA* aims to recommend a ranked list of additional columns from L to augment T, in order of relevance.

Prompt Construction. Given a table $T = (E, H)$, CENTS aims to reduce the inference cost of LLM-based table understanding by applying a token budget B to the table cells E and, when needed, applying a Top-K threshold to the label set L to improve performance when L is large. CENTS selects $E' \subseteq E$ such that $tokenCount(E') \leq B$, where *tokenCount* returns the total number of tokens in a given bag of values. CENTS aims to select a representative sub-table independently of a specific table understanding task, but ensures that each column contains at least one cell after reduction. Similarly, CENTS computes a reduced label set $L' \subseteq L$, such that $|L'| \leq K$. CENTS generates the reduced table $T' = (E', H)$ and combines it with L' and I to form the final prompt $P = (I \cup T' \cup L')$. When sending the prompt to the LLM, CENTS serializes the bag of cell values in E' column by column, with delimiters between columns and between cells within columns. Note that although B is used to reduce E in a single table instance, one may also define a global token budget for all instances in a benchmark and allocate an individual budget for each instance accordingly.

Symbol	Description
I	Task instructions in natural language.
$T = (E, H)$	A relation comprising a bag of cell-values, E, and optionally a header H.
E	Bag of cells grouped into columns (*Context Data*).
$E' \subseteq E$	Subset of cell values used in the prompt.
B	Token budget for E'.
L	Set of candidate labels (*Task Data*).
$L' \subseteq L$	Subset of candidate labels used in the prompt.
K	Top-K label threshold for L'.
P	Final prompt $(I \cup T' \cup L')$.

Table 1: Notation summary.

4 CENTS FRAMEWORK

In this section, we present CENTS's architecture and the details of its components. CENTS consists of a *Context Data Reducer* and a *Task Data Reducer* (Figure 3). The *Context Data Reducer* takes as input *Context Data*, $T(E, H)$ and a budget, B and generates a reduced *Context Data*, $T'(E', H)$, as output; while the bottom *Task Data Reducer* takes as input *Context Data* $T(E, H)$, *Task Data* L, and a threshold, K, and generates a reduced *Task Data* L'. CENTS merges both outputs to form the prompt, P. An example P for CTA is shown in Figure 4. CENTS serializes the *Context Data* column by column to prompt the LLM to interpret the table context on a per-column basis, thereby minimizing semantic distortion after CENTS's reduction. Finally, CENTS sends the generated prompt to the LLM and returns the LLM response.

4.1 Context Data Reduction - Processor

The Context Data Reducer has three sub-components: A Processor, a Scorer, and a Solver. We discuss the Processor here and the Scorer and Solver in the following sections. While we present the Processor for completeness, the pre-processing it performs resembles that done in prior work [11, 20], and we do not claim contribution here.

The input table $T(E, H)$ for the Processor comprises various types of columns, including numerical/datetime (e.g., column D and E in table A in Figure 5), which we refer to as non-textual columns; and textual columns (e.g., column A, B, C in table A in Figure 5). The Processor identifies, and samples non-textual columns, passing only the textual columns to the Scorer for further processing in the form of a refined table, $T^r(E^r, H)$ (e.g., table B in Figure 5).

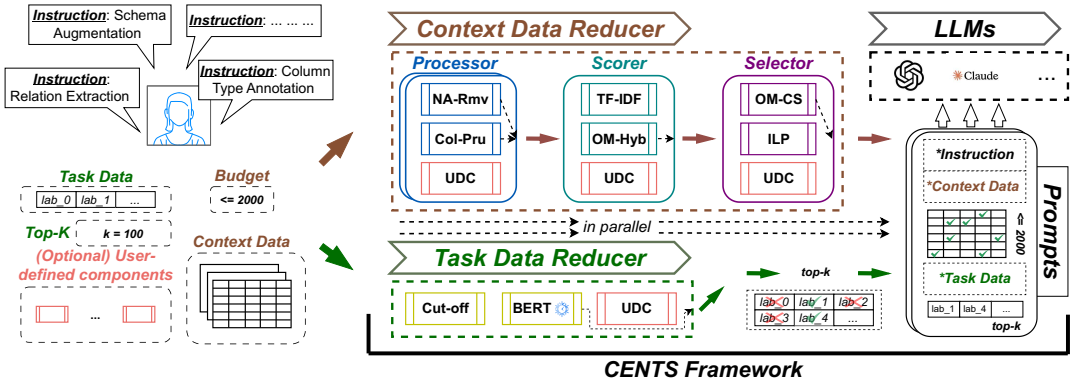


Figure 3: Overview of CENTS. CENTS consists of two main components, namely the *Context Data Reducer* and *Task Data Reducer*, which reduce the input *Context Data*, and the input set of task-specific *Task Data*, respectively. CENTS then combines the two outputs and the task instructions to create a prompt for the LLM. The output of the LLM is returned to the user/application.

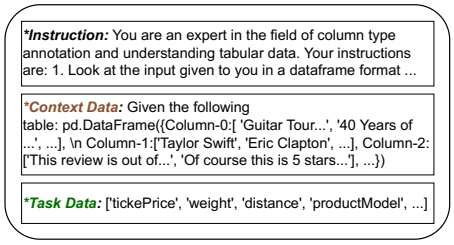


Figure 4: An example final prompt P after reducing *Context Data* and *Task Data* for CTA.

4.2 Context Data Reduction - Scorer

Motivated by Retrieval Augmented Generation (RAG)'s goal of retrieving the most relevant documents from a large corpus [26], CENTS 's *score-and-select* paradigm reduces a large table to a carefully chosen, semantically important and diverse sub-table. Non-textual columns are simply sampled by the Processor. For textual columns, CENTS employs a flexible *score-and-solve* paradigm: it assigns scores to the textual content and then selects a subset under a given token budget. This paradigm enables CENTS to optimize some key decisions: (1) determining the basic unit to be scored (e.g., a row, a cell, or words within a cell), (2) selecting a suitable scoring strategy, (3) distributing the token budget (e.g., varying budgets for each column), and (4) devising an effective and efficient selection method. We discuss the first two aspects in this section and the latter two in Section 4.3. While CENTS provides implementations for all the above, its flexible design conceptually allows a user to substitute any of the above components.

Scoring granularities. We consider three granularities:

Row. Under this strategy, each row in T^r is treated as an item, and a single score is assigned to it. While this strategy reduces the number of items for scoring and selection, it also limits flexibility. For example, if rows are the basic scoring unit, a valuable piece of information such as *"Taylor Swift is an American artist"* could appear in a row that is otherwise considered unimportant. Thus the entire row would receive a low score, and this important cell could be overlooked during selection.

Sub-cell. At the other extreme, one could split a cell into lists of *words* using heuristic methods such as whitespace splitting. Each *word* is then treated as the basic unit to score. This granularity offers maximum flexibility for scoring but has a significant drawback: after scoring each sub-cell word and reconstructing them into a list, the sub-cell words lose their contextual meaning within the sentence. For example, if a cell contains *"Taylor Swift is an American artist"*, after CENTS makes the selection, the reconstructed cell may appear as *"[Taylor, American]"*, loosing important context information.

Cell. We posit and experimentally show that treating each cell as the item to score achieves a balance between flexibility and contextual integrity, avoiding the drawbacks of the aforementioned alternatives. Thus, for the discussion of scoring strategies below, we assume that cell granularity is used. However, CENTS provides the other two options if needed. We show how scoring granularity strategies affect the end-to-end performance in Table 5.

We omit column granularity because tables have often small numbers of columns and removing entire columns would select data at only a course granularity. It might also drop important task information (e.g., dropping one column from a pair when prompting the LLM for RE would make the task infeasible.)

Scoring functions. The Scorer assigns a utility score to each item to reflect its importance within the table. The input to the scoring function is the refined table $T^r(E^r, H)$, and the output is the score table $T^c(E^c, H)$, where each cell $e^c_{ij} \in E^c$ contains a score. We consider three scoring strategies.

TF-IDF Scoring. Following Starmie [10], we develop a TF-IDF-based scoring function to assign scores. It first splits each cell into a list of tokens. Then it treats each column of T^r as a document to compute the Term Frequency (TF) of tokens within that column and the entire set of columns within the table as the collection of documents for calculating the Inverse Document Frequency (IDF). The final score for cell e^c_{ij} is the average of the TF-IDF scores of its tokens. A limitation is that it focuses solely on *syntactic* importance and does not account for the *semantic* importance.

Clustering Scoring. As an alternative approach, to take semantic information into account, we propose to cluster cell values in an embedding space, and select medoids as representative cells. We

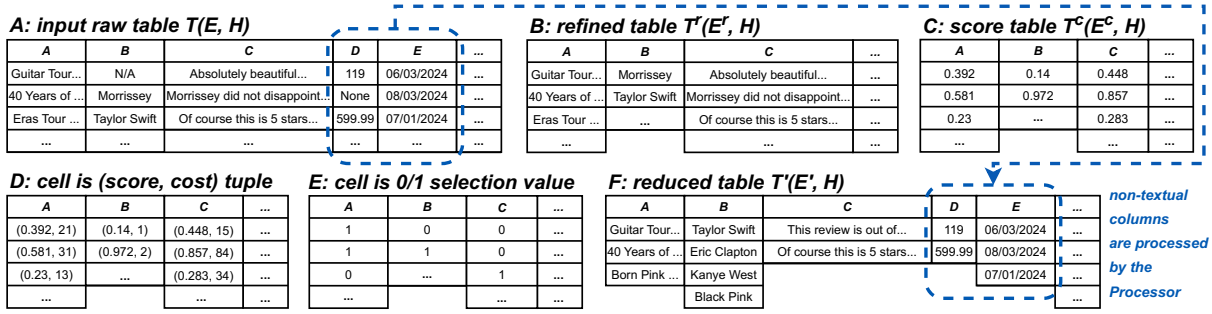


Figure 5: Example of *Context Data Reducer* workflow. Column D, E, containing only numerical and datetime values, are sampled by the Processor directly and inserted back into the final table *T'(E', H)*

compute cell embeddings using FastText[3] with a dimension of 100[4]. Alternatively, BERT-based table understanding models [7, 39] can also be used to compute cell embeddings. However, these models share a 512 token context window and cannot process entire tables at once. Although serializing each row avoids this limit, it has substantial overhead: for a token budget of 2000 for CTA without GPU, FastText generates embeddings in 129s, whereas Dodduo [39] requires 12,724s—roughly 100× slower. To make **Cents** lightweight, we therefore use FastText for embeddings. Considering each textual column separately, we cluster cell values using KMeans. To determine the number of clusters, we follow approaches such as [19] and set k_i for each column i as $\lfloor\sqrt{m_i}\rfloor$, where m_i is the number of non-empty cells in column i. For each cluster, we then select the medoid, which is the cell whose embedding vector is closest to the cluster centroid. We assign a score of 1.0 to that cell and a score of 0 to all other cells in the same cluster. Next, during the selecton phase, those scores will be combined with cell weights such that the most cost-effective cells are selected. After selecting k_i medoids for column i, if we still have budget for that column, the clustering scoring adopts an iterative approach to gradually increase k_i and re-run the KMeans to select new medoids from cells and calculate their costs until reaching the budget.

OM-Hyb. Because the TF-IDF and clustering methods have complementary benefits, we further propose a hybrid method. In this method, *OM-Hyb*, shown in Algorithm 1, considers medoids within columns *and* TF-IDF across columns. The intuition is that its output captures both syntactic importance and semantic importance. Cells with rare terms are emphasized through TF-IDF, while medoids identify representative and contextually meaningful cells. *OM-Hyb* first computes the TF-IDF score of each cell e^c_{ij} as in the pure TF-IDF scoring method, as shown from Line 1 to Line 13. The TF-IDF score of a cell is computed as the average of its per-token scores. Starting from Line 14, the algorithm computes medoids by generating cell embeddings. In our prototype implementation, we use the same library (FastText), vector dimension (100), and the number of clusters ($\lfloor\sqrt{m_i}\rfloor$) as in the clustering scoring method. Finally, **Cents** multiplies the medoid TF-IDF scores by a scale factor, C, to make the medoids more likely to be selected, although other approaches are possible. *OM-Hyb* empirically sets $C = 2$. Finally, we re-normalize all scores within the column, resulting in changes to the scores of

Algorithm 1: *OM-Hyb*

Input: Refined table $T^r(E^r, H)$ with n columns and m rows; Scale factor C
Output: Score table $T^c(E^c, H)$ with score for each cell

1 **begin**
2 **for** $i \leftarrow 1$ **to** n **do**
3 $\mathcal{V}_i \leftarrow \emptyset$;
4 **for** $j \leftarrow 1$ **to** m **do**
5 $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \text{Tokenize}(e^r_{ij})$;
6 $\text{TF}_i(t) \leftarrow \text{TF}(t, \mathcal{V}_i)$ for all $t \in \mathcal{V}_i$;
7 $\mathcal{V} \leftarrow \bigcup_{i=1}^{n} \mathcal{V}_i$;
8 $\text{IDF}(t) \leftarrow \text{IDF}(t, \mathcal{V})$ for all $t \in \mathcal{V}$;
9 **for** $i \leftarrow 1$ **to** n **do**
10 **for** $j \leftarrow 1$ **to** m **do**
11 **foreach** *token* $t \in d_{ij}$ **do**
12 $\text{TF-IDF}_{ij}(t) \leftarrow \text{TF}_i(t) \times \text{IDF}(t)$;
13 $e^c_{ij} \leftarrow \frac{1}{|d_{ij}|} \sum_{t \in d_{ij}} \text{TF-IDF}_{ij}(t)$;
14 **for** $i \leftarrow 1$ **to** n **do**
15 $m_i \leftarrow$ number of non-empty cells in col_i;
16 $k_i \leftarrow \lfloor\sqrt{m_i}\rfloor$;
17 $\mathcal{M}_i \leftarrow \text{ComputeColMedoids}(k_i, e^r_{i1}, e^r_{i2}, \ldots, e^r_{im_i})$;
18 **for** $i \leftarrow 1$ **to** n **do**
19 **for** $j \leftarrow 1$ **to** m **do**
20 **if** $e^r_{ij} \in \mathcal{M}_i$ **then**
21 $e^c_{ij} \leftarrow C \times e^c_{ij}$;
22 **return** $T^c(E^c, H)$;

non-medoids as well as the medoids, and ensuring all scores are in $[0, 1]$. With this approach, medoids are most likely going to be selected, unless they have a very low TF-IDF score. Furthermore, medoids with higher TF-IDF scores will be favored over those with lower scores. However, the selection algorithm presented in the next section also takes into account the weight of each cell (i.e., its token count) when selecting.

4.3 Context Data Reduction - Solver

The role of the Solver is to select a subset of scored items to maximizes the total score subject to staying within budget constraints. The Solver needs to do this in a way that leads to a good solution and that is computationally efficient. The Solver takes as input the scored table, $T^c(E^c, H)$, and a budget B. It uses the tokenization function *tokenCount* (see Section 3) to measure the token cost of each cell thus obtaining a (score, cost) tuple for each cell. I.e., each element $e^c_{ij} \in E^c$ becomes a tuple consisting of a score and a weight,

[3] https://fasttext.cc/docs/en/python-module.html
[4] We tested both 300 and 100 dimensions, the difference was minimal, so we chose 100.

$e^c_{ij} = (s_{ij}, w_{ij})$, where s_{ij} is the score and the w_{ij} the token cost of cell at ith column and jth row. An example of such table is shown as table D in Figure 5. Given the structure of tabular data, we have two sub-problems: how to allocate the budget and how to find the optimal or near-optimal combination of scored items.

Budget allocation. One budget allocation method is to use the budget for the entire table $T^c(E^c, H)$. However, this approach can lead to over-selection of cells in certain columns while neglecting others, as shown in Table 6. An alternative way is to divide the budget equally among all columns, but this fails to account for the varying distribution of content within and across columns.

To address these challenges, we leverage the concept of *empirical entropy* for column values from [5] to develop an adaptive budget allocation strategy. The intuition is that columns with a more "spread-out" distribution of content require a larger budget to select relevant information. The empirical entropy H^i_E for column i is calculated by mapping each non-empty cell into tokens and computing the probabilities $P(t)$ of each unique token t based on its frequency relative to the total number of tokens in the column:

$$H^i_E = -\sum_{t \in T_i} P(t) \log_2 P(t), \tag{1}$$

where T_i is the set of unique tokens in column i. Given the total budget B, we then compute the allocation B_i for each column i using the empirical entropy H^i_E of the column as:

$$B_i = B \cdot \frac{H^i_E}{\sum_{j=1}^{n} H^j_E}, \tag{2}$$

where $\sum_{j=1}^{n} H^j_E$ is the total entropy across all n columns. This ensures that the budget allocated to each column is proportional to its entropy, allowing columns with more diverse distributions to receive a higher fraction of the budget. For the remainder of this section, we assume that we have individual budgets $B_1, \ldots, B_n$ for each of the n columns, with the total budget $B = \sum_{i=1}^{n} B_i$. However, Solvers in **Cents** also support other budget options discussed. We show how budget allocation affects performance in Table 5.

Scored content selection To find the best cell combination under a budget, **Cents** formalizes it as an optimization problem.

ILP. Content selection can be formalized as an Integer Linear Programming (ILP) as follows:

$$\begin{aligned} \text{Maximize} \quad & \sum_{i=1}^{n} \sum_{j=1}^{m} s_{ij} y_{ij} \\ \text{subject to} \quad & \sum_{j=1}^{m} w_{ij} y_{ij} \le B_i \quad \forall i \in \{1, \ldots, n\}, \\ & y_{ij} \in \{0, 1\} \quad \forall i, j, \end{aligned} \tag{3}$$

where y_{ij} is a binary variable indicating whether cell e_{ij} is selected, s_{ij} is the score of the cell, and w_{ij} is its token cost. We have n columns, each with a budget B_i. Directly inputting this formulation into an off-the-shelf solver[5] yields the best content selection as shown in Table 7. However, it has a high execution time as shown in Figure 9, due to the large search space(one variable per cell and many cells being selected), limiting its practicality. Alternatively, a dynamic programming-based solver yields similarly good selections but also slow execution times due to its $O(nmB)$ time complexity.

[5]https://github.com/mosek

Algorithm 2: Convert

Input: An optimal solution y^*_C for column C for relaxed LP problem
Output: A feasible solution y^{**}_C for column C to Equation 3

begin
  $R \leftarrow$ FormCellSets(y^*)
  while $|\{cs \in R \mid cs \text{ is non-0/1}\}| > 1$ **do**
    $cs_\alpha, cs_\beta \leftarrow$ GetTwoCellSets(R); // $\beta > \alpha$
    $R \leftarrow$ ReduceCellSets(R, cs_α, cs_β);
  $y^{**}_C \leftarrow$ RoundCellSets(R);
  return y^{**}_C;

Algorithm 3: RoundCellSets

Input: Current set of cell-sets R, Current two cell-sets cs_α, cs_β
Output: A new set of cell-sets R^* with exactly one fewer cell-set

begin
  $W^{cs}_\alpha, W^{cs}_\beta \leftarrow$ GetCost(cs_α, cs_β);
  $S^{cs}_\alpha, S^{cs}_\beta \leftarrow$ GetValue(cs_α, cs_β);
  $cs_o, cs_p, cs_k, cs_l \leftarrow$ GetNeighborCellSets(cs_α, cs_β, R);
  $r \leftarrow \frac{S^{cs}_\alpha}{S^{cs}_\beta}$;
  $flag \leftarrow W^{cs}_\alpha - \frac{S^{cs}_\alpha W^{cs}_\beta}{S^{cs}_\beta}$;
  if $flag \ge 0$ **then**
    $\epsilon_\alpha \leftarrow \min\left(\alpha - o, \frac{S^{cs}_\beta}{S^{cs}_\alpha}(l - \beta)\right)$;
    $\alpha' \leftarrow \alpha - \epsilon_\alpha$;
    $\beta' \leftarrow \beta + r\epsilon_\alpha$;
  else
    $\epsilon_\alpha \leftarrow \min\left(p - \alpha, \frac{S^{cs}_\beta}{S^{cs}_\alpha}(\beta - k), \frac{S^{cs}_\beta}{S^{cs}_\alpha + S^{cs}_\beta}(\beta - \alpha)\right)$;
    $\alpha' \leftarrow \alpha + \epsilon_\alpha$;
    $\beta' \leftarrow \beta - r\epsilon_\alpha$;
  $cs_{\alpha'}, cs_{\beta'} \leftarrow$ ChangeValue($cs_\alpha, cs_\beta, \alpha', \beta'$);
  $R^* \leftarrow$ RemoveCellSets(R, cs_α, cs_β);
  $R^* \leftarrow$ InsertCellSets($R^*, cs_{\alpha'}, cs_{\beta'}$);
  return R^*;

OM-CS. We observe that the slow execution time of an ILP solver is due to its use of the branch-and-bound method, which repeatedly solves the Linear Programming (LP) relaxation of the problem. This LP relaxation gives fractional selection values (e.g., 0.43) between 0 and 1 for each cell instead of binary values. Simply rounding the fractional solutions from the LP solver can lead to either infeasible solutions that violate constraints or sub-optimal outcomes. To address this challenge, we propose a solver, *OM-CS*, specialized for our setting. *OM-CS* runs LP *once* to ensure fast execution. Unlike naive rounding, once *OM-CS* has the optimal solution generated by LP, it iteratively groups cells together and rounds their selection values toward 0 or 1. The key intuition for *OM-CS* to be effective for our problem is that it eventually returns binary but not fractional solutions for cell selection while *carefully maintaining the best total score without violating budget constraints* by adjusting two groups of selection values *simultaneously*.

4.3.1 Step 1: Relax and solve the LP Problem. *OM-CS* begins by relaxing the ILP in Equation 3 from $y_{ij} \in \{0, 1\}$ to $y_{ij} \in [0, 1]$. Using an LP solver, *OM-CS* solves the relaxed IP problem to obtain the optimal fractional solution y^*.

4.3.2 Step 2: Convert LP solution. Given the optimal fractional solution y^*, *OM-CS* converts y^* to y^{**}, a feasible solution to Equation 3. Given that each column has its own budget, *OM-CS* rounds its per-column LP solution, y^*_C, independently. Each cell in y^*_C is of the form $e^*_{ij} = (y_{ij}, s_{ij}, w_{ij})$.

Algorithm 2 shows the approach. Within y^*_C, *OM-CS* aggregates cells with the same selection value $y_{ij} = \alpha$ into a *cell-set* cs_α (Line 2). This is to reduce the search space. For example, as shown in Figure 6, *OM-CS* groups the tuples in the leftmost column by their y_{ij} values into cell-sets shown in the middle state D-A. We define the $\leq$ operator for cell-sets as: $cs_\alpha \leq cs_\beta$ if $\alpha \leq \beta$. After construction, we have the following cell-sets in R, where $v = 0$ and $u = 1$:

$$cs_v \leq ... \leq cs_o < cs_\alpha \leq cs_p ... \leq cs_k < cs_\beta < cs_l \leq ... \leq cs_u \quad (4)$$

We further define a few more symbols for convenience. We call *neighbors*, cell sets that are consecutively ordered in R. We use W^{cs}_α to denote the total cost of cells in cell-set cs_α and S^{cs}_α the total score of cells in cell-set cs_α. For example, in state D-A in Figure 6, $W^{cs}_{\alpha=0.42} = 6+4+4+6 = 20$, $S^{cs}_{\alpha=0.42} = 0.7+0.61+0.55+0.45 = 2.31$. After constructing cell-sets following Equation 4, within the for loop of Algorithm 2, *OM-CS* slowly adjusts the selection values of cell sets to gradually move to a solution with only three cell-sets and selection values $\{0, 1, \alpha\}$. For this, *OM-CS* randomly samples two non-0/1 cell-sets that satisfy the constraint $\beta > \alpha$. The *OM-CS* then does three things: (1) It merges one of the two selected cell-sets with its neighbor by adjusting its selection value, α or β, thus reducing the total number of cell sets; (2) It adjusts the selection value of the other selected cell-set in the opposite direction and by an amount that ensures the total score remains the same and the total cost does not increase. At the end, only cells with selection value 1 are returned. Algorithm 3 summarizes the approach.

We now explain the intuition behind Algorithm 3 using Figure 6 as an example: We begin by getting the total scores and total costs of cs_α and cs_β (e.g., in Figure 6, we randomly pick $\alpha = 0.42$, $\beta = 0.52$, and thus we have $W^{cs}_\alpha = 20$, $S^{cs}_\alpha = 2.31$, $W^{cs}_\beta = 21$, $S^{cs}_\beta = 1.23$). Then on Line 4, we get the neighbor cell-sets for $cs_{\alpha=0.42}$ and $cs_{\beta=0.52}$ as defined in Equation 4 (e.g., in the example, the neighbors of cs_α are cs_o, cs_p with selection value $o = 0.39$, $p = 0.468$, and the neighbors of cs_β are cs_k, cs_l with selection value $k = 0.47$, $l = 0.83$). *OM-CS* will increase the selection value of one of the two cell-sets (either cs_α or cs_β) and decrease the selection value of the other, such that it eventually merges one of the two cell-sets with a neighbor. *OM-CS* cannot simply move the selection values of the two cell-sets by the same amounts. It needs to take their costs and scores into account to remain within budget and maximize the total score of the final solution. *OM-CS* proceeds as follows. We first define r in Line 5 as the score ratio of cs_α and cs_β (e.g., in the example, $r = \frac{2.31}{1.23}$). *OM-CS* then needs to decide whether to increase α and decrease β or vice-versa. To do so, it computes a *flag* value, shown in Line 6. Let ϵ_α and ϵ_β be the amounts by which *OM-CS* will update α and β. We set $\epsilon_\beta = \frac{S^{cs}_\alpha}{S^{cs}_\beta}\epsilon_\alpha$, which ensures that the total score remains fixed as *OM-CS* updates selection values. In the first scenario (increasing α, decreasing β), the change in total cost is $W^{cs}_\alpha \epsilon_\alpha - W^{cs}_\beta \epsilon_\beta = \epsilon_\alpha\big(W^{cs}_\alpha - \frac{S^{cs}_\alpha W^{cs}_\beta}{S^{cs}_\beta}\big)$. In the second scenario (decreasing α, increasing β), the change in total cost is $-\epsilon_\alpha\big(W^{cs}_\alpha - \frac{S^{cs}_\alpha W^{cs}_\beta}{S^{cs}_\beta}\big)$. Here, since the total cost changes in these two scenarios are inverse of each other, we can pick the scenario in which the total cost does not increase. To make such a decision, we define $flag = W^{cs}_\alpha - \frac{S^{cs}_\alpha W^{cs}_\beta}{S^{cs}_\beta}$. (e.g., in the example, $flag = 20 - \frac{2.31\times21}{1.23} \approx -19 < 0$, we would increase α by ϵ_α and decrease β by ϵ_β.) Algorithm 3 decides how much to adjust selection value of each cell-set on Line 12 based on the constraint that the total score shall not be changed. For example, when adjusting one of the cell-set selection values to its neighboring cell-sets (e.g., subtract $r \times \frac{1.23}{2.31} \times 0.05 = 0.05$ from $\beta = 0.52$ down to 0.47 in the state D-B), we adjust the other cell-set correspondingly (e.g., add $\frac{1.23}{2.31} \times 0.05$ to $\alpha = 0.42$ up to 0.4466 in the state D-B). Adjusting the two cell-sets lowers the total cost to 127.48, and maintains the total score, thereby successfully moving from state D-A to state D-B in Figure 6 and eliminating one cell-set.

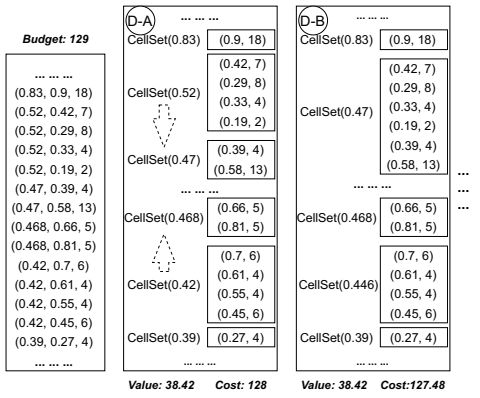


Figure 6: Workflow of *OM-CS* on a single column. Leftmost column is solution from relaxed LP problem, middle column shows process of forming cell-set, and rightmost column shows result after one iteration of Algorithm 3.

Lemma 4.1. *Given an optimal solution y^*_C to the relaxed LP problem for column C, Algorithm 2 and Algorithm 3 can always reduce the solution to cell-sets that contain only one of the three values $\{0, 1, \alpha\}$, while maintaining it as the optimal solution.*

Proof sketch (by induction). If at most one fractional cell-set remains with α, we are done. Otherwise, Algorithm 2 picks two different fractional sets cs_α and cs_β and runs Algorithm 3 to adjust their selection values in opposite directions by ϵ_α and ϵ_β. By doing so, it eliminates one cell-set, retains the same total score, and does not increase the total cost. Since each round reduces the number of fractional values by one, after finitely many steps only the three values $\{0, 1, \alpha\}$ remain. □

We refer interested readers to the full proof details [6].

At the end, for each column C, *OM-CS* is left with exactly one non-0/1 cell-set. It discards this to obtain y^{**}_C because including it would increases the total score of the solution the LP solver already gives the optimal solution y^*, and thus by contradiction we know that including this last cell-set yields an infeasible solution.

[6] http://drive.google.com/drive/folders/1pEuK01i0hAp7mI1xQ0GSwjRW4Cuo40eT

4.4 Task Data Reduction

There are two motivations for *Task Data Reducer*: first, a larger number of classes generally leads to more complex problems; second, the ordering of classes in prompts can affect LLM performance [37]. Consequently, *Task Data Reducer* processes the original $T(E, H)$ and the set of original *Task Data* L (e.g., the schema_vocab LLM needs to choose from at the bottom of Figure 2) to generate a reduced label set L', ranked by likelihood. *Task Data Reducer* reduces the label space for LLMs, thereby enhancing their performance over a variety of table understanding tasks.

A naive way is to truncate L to a shorter, L', with K elements, which is not effective when *Task Data* is a list of labels, since the correct label may not be in L', as shown in Section 5. Instead, following the idea in Doduo [39], we leverage a pre-trained BERT Language Model (LM) [8] with *frozen* weights and *only fine-tune a linear layer* on top of it to reduce L in a lightweight way.

Specifically, *Task Data Reducer* first serializes the input table T by concatenating all cell values, inserting a dummy token [CLS] before each column, and appending a final [SEP] token at the end. The serialization process is defined as:

$$\text{ser}(T) = [\text{CLS}] + \text{ser}(e_{11}) + \ldots + [\text{CLS}] + \text{ser}(e_{21}) + \ldots + [\text{SEP}] \quad (5)$$

where $\text{ser}(e_{ij})$ denotes the serialization of the cell e_{ij}. The *Task Data Reducer* feeds $\text{ser}(T)$ into BERT to get context-aware embeddings for each token. Then it extracts the embedding of the final [SEP] token and apply a dense linear layer.

For training, the *Task Data Reducer* treats this task as a multi-class multi-label classification problem since each table can be associated with multiple labels (i.e., multiple headers to recommend) and use the binary cross-entropy loss function. During inference, the *Task Data Reducer* ranks the logits generatedx from highest to lowest and selects top K labels based on this ranking.

5 EVALUATION

In this section, we evaluate **Cents** on popular table understanding benchmarks and show that **Cents** successfully lowers the token cost, while improving F1 scores compared with vanilla LLM-based baselines. We then study the detailed components of **Cents** and their impact on the framework's performance. We evaluate the *Context Data Reducer* on CTA and RE due to their large *Context Data* sizes whereas we evaluate the *Task Data Reducer* on SA due to its large *Task Data* size, as shown in Table 2.

Implementation: We implement **Cents** using Python and Pytorch.[7] We use GPT-3.5-turbo and GPT-4O to evaluate the end-to-end performance of three tables understanding tasks: CTA, RE, and SA. We use TikToken[8] as the *tokenCount* function to count token cost in this paper as it is the standard tokenizer used by OpenAI. We conduct our experiments on a server with a single L40S GPU with 24 cores and 128GB RAM.

Datasets: For CTA and RE, we evaluate **Cents** over the SOTAB[9] dataset [23], a popular benchmark used by recent table understanding related research [6, 11, 22]. For SA, we evaluate **Cents** over the TURL dataset [7], another popular benchmark used by [32, 39, 45], which consists of web tables from Wikipedia. For these these datasets, we follow ArcheType [11]'s approach and reduce the semantic overlap among labels to let LLMs discriminate between semantically different labels. The statistics are shown in Table 2.

Evaluation Metrics: When evaluating CTA and RE, we follow existing work [7, 11, 20, 39] and use Micro F1. For SA, we follow TURL's setting [7] and use mean Average Precision (MAP).

Baselines: For CTA, we compare **Cents** with five baselines. The first is CHORUS [20], an LLM-based system designed for tasks such as table-class detection, CTA, and join-column prediction. The second is ArcheType [11], a method specifically tailored for CTA. To the best of our knowledge, these are the state-of-the-art LLM-based methods for CTA. As it has already been evaluated on SOTAB27CTA, we directly take the confusion matrix provided by the authors and calculate the corresponding Micro F1. Third, we use the *Context Data* Reduction Random Sample as a baseline method, in which we sample rows up to the given budget. For our fourth baseline, **Cents** (Max Context), we include as much table context as possible until the final LLM's context window is close to full. We set the token budget to 15,000, slightly below the GPT-3.5-turbo limit of 16,385[10] to leave room for *Instruction*, *Task Data*, and output tokens. We then run **Cents** to evaluate its performance when budget is not tightly constrained, and understand whether table reduction at a low budget using **Cents** remains useful in such case. This leads to $54.5 (USD), as shown in Figure 1. Finally, we apply the state-of-the-art prompt compression method LLMLingua [18] that requires using LLM Phi-2 [17] for compression. This method requires *significantly more computing resources* than **Cents** as an LLM needs to be deployed locally.

For RE, since CHORUS and ArcheType do not support RE, we compare **Cents** with three baselines: *Context Data* Reduction Random Sample, **Cents** (Max Context), and LLMLingua [18].

Lastly, for SA, which neither CHORUS nor ArcheType support, we compare **Cents** with the following baseline: given that the average number of tokens per table, as shown in Table 2, is small but the number of labels is large, we serialize the full table and populate the prompt with K labels selected at random. We discuss the choice of K later in this section.

5.1 Main Results

The main results are shown in Table 3 and Table 4 and we highlight the best-performing methods in bold. ArcheType and CHORUS both use a single column as the context provided to the LLM. However, they do not share the same budget, and ArcheType outperforms CHORUS due to its carefully designed CTA-specific framework that includes components such as rule-based label filtering (e.g., provides only numerical-related labels for numerical columns) and label re-sampling (e.g., call the LLM multiple times to get diverse answers), which also makes it also hard to count the tokens used by this method. We report the best result for all methods for fair comparison and discuss the budget impact in Section 5.2.6.

CTA: For CTA, we provide the full *Task Data* given its relatively small size. Compared with CHORUS and ArcheType, both of which uses a single column as context, **Cents** achieves higher Micro F1 with additional contextual data. It outperforms ArcheType by 11.2

[7] https://pytorch.org/

[8] https://github.com/openai/tiktoken

[9] Note that there is another dataset with a similar name SOTAB V2, but the dataset used in this paper is adopted from [11, 23].

[10] https://platform.openai.com/docs/models/gpt-3.5-turbo

Table 2: Dataset statistics for three table understanding tasks: CTA, RE, and SA.

Dataset	# of tables	# of test instances	avg. columns	avg. rows	avg. tokens per table	# of labels
SOTAB27CTA [11, 23]	7026	15040	8.4	200	40863	27
SOTAB41RE [23]	6480	23155	9.3	249	46360	41
TURL1748SA [7]	4646	4646	5.8	20	2106	1748

Table 3: Main result for SOTAB27CTA and SOTAB41RE.

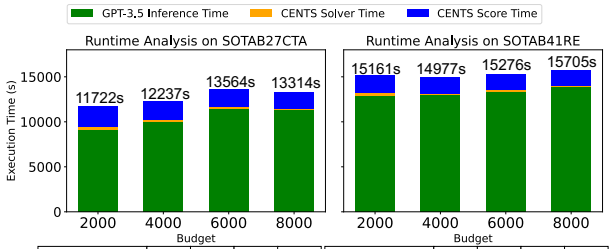
Benchmark	Method	Micro F1
SOTAB27CTA	ArcheType	65.2
	CHORUS GPT-3.5-turbo	46.7
	Random Sample GPT-3.5-turbo	58.8
	Cents (Max Context) GPT-3.5-turbo	70.9
	LLMLingua GPT-3.5-turbo	60.2
	Cents GPT-3.5-turbo	**76.4**
	CHORUS GPT-4O	59.4
	Random Sample GPT-4O	65.8
	Cents GPT-4O	**80.4**
SOTAB41RE	Random Sample GPT-3.5-turbo	53.4
	Cents (Max Context) GPT-3.5-turbo	57.6
	LLMLingua GPT-3.5-turbo	46.9
	Cents GPT-3.5-turbo	**68.4**
	Random Sample GPT-4O	72.4
	Cents GPT-4O	**79.1**

Table 4: Main result for TURL1748SA.

Benchmark	Method	MAP
TURL1749SA	Max Labels GPT-3.5-turbo	38.4
	Cents GPT-3.5-turbo	**55.9**
	Max Labels GPT-4O	39.9
	Cents GPT-4O	**61**

Micro F1 and CHORUS by 29.7 when using GPT-3.5-turbo. **Cents** outperforms Random Sampling by 17.6 by selecting context data at the granularity of cells. **Cents** also outperforms **Cents** (Max Context) by 5.5 Micro F1, showing that including excessive context leads to performance drop, in line with recent findings [25, 30]. This shows that even when the budget is not constrained, **Cents** is useful with a lower budget as it filters out excessive information to increase performance. Moreover, despite using significantly less computing resources, **Cents** outperforms LLMLingua by 16.2 Micro F1 because LLMLingua is not specifically designed for tables.

A larger model such as GPT-4O [35] improves all results, while yielding the same cost-effectiveness trade-off. Figure 1 compares the methods across a range of budgets. CHORUS randomly samples four values from a column as context. We thus estimate its per-table token cost to be the total number of tokens used for the benchmark divided by the number of tables in the benchmark, which is 358 tokens per instance. As Figure 1 shows, even if we reduce **Cents**'s budget to match CHORUS', **Cents** achieves a higher F1 score thanks to its more careful selection of the context to include and its inclusion of more context. We do not estimate ArcheType budget



CTA Runtime %	2000	4000	6000	8000
CENTS	22.5%	18.4%	15.6%	14.6%
GPT-3.5-Turbo	77.5%	81.6%	84.4%	85.4%

RE Runtime %	2000	4000	6000	8000
CENTS	14.6%	13.3%	12.4%	11.6%
GPT-3.5-Turbo	85.4%	86.7%	87.6%	88.4%

Figure 7: Execution Time Analysis. Numbers above bars are totals. Table below shows execution time percentages.

for the reasons described at the start of this section. Interestingly, **Cents** significantly outperforms Random Sampling even when the latter uses a better model and a higher budget.

RE: For RE, the full *Task Data* is also given to all methods due to its relatively small size. The results are similar to CTA: when compared against Random Row sampling, **Cents** shows an improvement of 15 Micro F1, further demonstrating the importance of carefully selecting the context for table understanding tasks. Furthermore, **Cents** also outperforms **Cents** (Max Context) by 10.8 Micro F1 and LLMLingua by 21.5 Micro F1. These gains are due to **Cents**'s filtering of redundant and excessive context and its tailored approach for handling tabular data, as discussed earlier in the results for CTA. As mentioned in the baseline setting, we do not include CHORUS nor ArcheType as they do not support RE.

SA: For SA, on the other hand, we give all methods the full *Context Data* since the average token cost is only 2106 for each table instance. We compare *Task Data Reducer* against providing all labels to the LLM. As shown in Table 4, our *Task Data Reducer* is effective at reducing the *Task Data* in a way that not only achieves the same MAP, but increases it by 17.5. We also notice that in the SA task, using a more advanced model with the Max Labels setting does not increase the performance significantly. This is possibly due to two reasons: *Task Data Reducer* reduces the search space and crafts shorter prompts [30], and it ranks labels by likelihood [37].

5.2 Detailed Results

We discuss the effects of design choices for **Cents** components.

5.2.1 Runtime Analysis. We evaluate **Cents**'s execution time, running *OM-CS* and *OM-Hyb*. As shown in Figure 7, the LLM inference time dominates. **Cents** contributes up to 22.5% for SOTAB27CTA and 14.6% for SOTAB41RE to the total execution time. Additionally, while **Cents**'s primary goal is to reduce the dollar cost of API calls, it also reduces the LLM inference time, and is sufficiently lightweight that the total time also goes down.

Configurations	SOTAB27CTA B2000	B4000	B6000	B8000	SOTAB41RE B2000	B4000	B6000	B8000
Effect of Scoring Strategy								
Random Sample + GPT-3.5-turbo	58.1	58.6	58.8	58.7	51.5	52.2	53.2	53.4
TF-IDF + *OM-CS* + GPT-3.5-turbo	72.3	72.5	73.2	73.1	63.6	63.8	64.5	65.3
Clustering + *OM-CS* + GPT-3.5-turbo	67.9	70.2	71.1	71.3	60	61.7	62.3	62.3
OM-Hyb + *OM-CS* + GPT-3.5-turbo	**75.4**	**76.1**	**76.4**	**76.4**	**66.8**	**67**	**68.2**	**68.4**
Effect of Scoring Granularity								
OM-Hyb (Row) + *OM-CS* (Row) + GPT-3.5-turbo	71.3	72.1	72.9	73.8	62.6	63.8	64.0	64.7
OM-Hyb (Word) + *OM-CS* (Word) + GPT-3.5-turbo	52.6	51.9	51.8	50.9	41.5	39.7	39.2	38.5
OM-Hyb (Cell) + *OM-CS* (Cell) + GPT-3.5-turbo	**75.4**	**76.1**	**76.4**	**76.4**	**66.8**	**67**	**68.2**	**68.4**
Effect of Budget Allocation Strategy								
OM-Hyb + *OM-CS* (Table Budget) + GPT-3.5-turbo	72.2	73.6	73.9	74.4	62.8	63.4	63.9	64.7
OM-Hyb + *OM-CS* (Equal Budget) + GPT-3.5-turbo	73.7	74.6	74.9	75.6	63.2	65.8	66.2	66.4
OM-Hyb + *OM-CS* (Entropy Budget) + GPT-3.5-turbo	**75.4**	**76.1**	**76.4**	**76.4**	**66.8**	**67**	**68.2**	**68.4**
Effect of Budget B with Different Model								
OM-Hyb + *OM-CS* + GPT-3.5-turbo	75.4	76.1	76.4	76.4	66.8	67	68.2	68.4
OM-Hyb + *OM-CS* + GPT-4O	**80**	**80**	**80.3**	**80.4**	**78.5**	**79.1**	**79.1**	**79.1**
OM-Hyb + *OM-CS* + DeepSeek-R1-Distill-Llama-8B	49.6	49.8	50	49.8	48.2	48.8	48.9	48.3
OM-Hyb + *OM-CS* + TableGPT2	50.7	51.4	50.5	50.6	50.2	50.4	51.8	51.5

Table 5: Analysis of different design choices in CENTS for SOTAB27CTA and SOTAB41RE. For each budget B (i.e., count of tokens for reduced *Context Data* is $\leq B$, the best result is highlighted in bold. Units are Micro F1.

Benchmark	B2000	B4000	B6000	B8000
SOTAB27CTA	2889	1498	990	745
SOTAB41RE	3527	1960	1320	980

Table 6: Number of columns that CENTS fails to sample at all if CENTS keeps a budget for the whole table.

5.2.2 Effectiveness of budget allocation. The impact of the budget allocation in the *Context Data Reducer* on **CENTS**'s end-to-end performance is shown in Table 5. We evaluate three strategies: *OM-CS* (Table Budget), which assigns a single budget to the entire table; *OM-CS* (Equal Budget), which splits the budget equally among all columns; and our proposed *OM-CS* (Entropy Budget). Our results indicate that the *OM-CS* (Entropy Budget) consistently achieves the best performance across all budgets. Although the *OM-CS* (Equal Budget) performs better than the *OM-CS* (Table Budget), it is still worse than the *OM-CS* (Entropy Budget), particularly at lower budget settings because *OM-CS* (Entropy Budget) adaptively assigns budgets to columns based on their content.

5.2.3 Effectiveness of different scoring granularities. We evaluate how different scoring granularities affect **CENTS**'s end-to-end performance. We modify our *OM-CS* such that it works for each data granularity (i.e.,*OM-CS* (Word) records word locations, selects individual words, and reconstructs cell values from those words; *OM-CS* (Row) computes a score for each cell, aggregates those scores at the row level, and selects row-wise). The results in Table 5 indicate that *OM-Hyb* (Cell) achieves the best performance across various budget settings. *OM-Hyb* (Row) does not perform as well because of its inflexibility. On the other hand, *OM-Hyb* (Word) performs the worst likely due to the ill-formed context it introduces for LLMs when cells are reconstructed from lists of words.

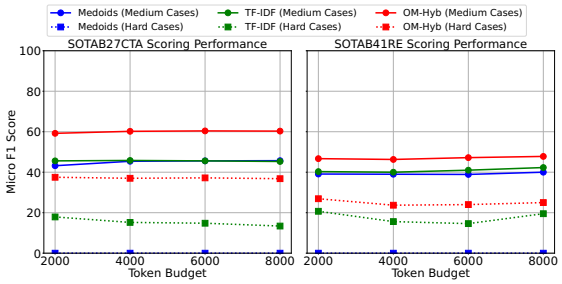


Figure 8: Analysis of different scoring function design choices in CENTS for SOTAB27CTA and SOTAB41RE.

5.2.4 Effectiveness of different scorers. We evaluate the effectiveness of scoring strategies, specifically Random Sampling, TF-IDF, Clustering, and our proposed *OM-Hyb*. As the results in Table 5 show, our *OM-Hyb* outperforms all baseline methods, showing that *OM-Hyb*'s robust performance across different tasks. We dig deeper into the differences between scorers by distinguishing their performance on tables that pose different levels of difficulty. We consider *Medium Cases*, those instances where the Random Sampling method fails and *Hard Cases*, those where both the Random Sampling and Clustering methods make errors. We ignore easy cases where even Random Sampling suffices. Figure 8 shows the performance of our *OM-Hyb* on these cases. For *Medium Cases*, we compare the solid lines of *OM-Hyb* with those of the other methods. For *Hard Cases*, we compare the dotted lines. Our results demonstrate that *OM-Hyb* consistently outperforms the other two methods in both scenarios and by a significant margin, showcasing its ability to correctly handle challenging instances where other approaches fail.

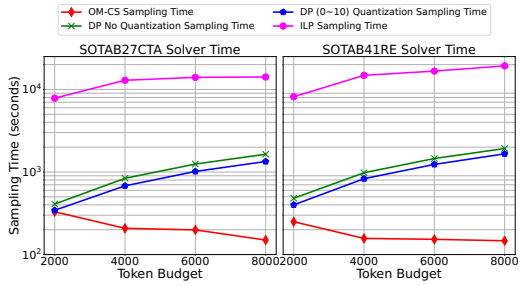


Figure 9: Analysis of execution time of different solver choices in CENTS for SOTAB27CTA and SOTAB41RE.

Configurations	SOTAB27CTA	SOTAB41RE
OM-Hyb + ILP	**77.1**	**68**
OM-Hyb + DP	76.6	67.8
OM-Hyb + DP (0 ~ 10)	73.8	64.5
OM-Hyb + *OM-CS*	76.1	67.6

Table 7: Analysis of performance, averaged over all budgets, of different solver choices in CENTS for SOTAB27CTA and SOTAB41RE with GPT-3.5-turbo.

Token Budget	SOTAB27CTA	SOTAB41RE
B2000	4956 (95.1%)	4581 (95.7%)
B4000	3821 (90.2%)	3576 (91.3%)
B6000	3094 (85.3%)	2984 (87.1%)
B8000	2625 (80.4%)	2600 (82.7%)

Table 8: Number of tables reduced per budget, with the average percentage of token reduction in parentheses.

5.2.5 Effectiveness and execution time of different Solvers. We now evaluate the efficiency (i.e., execution time) of each solver: we compare *OM-CS* against an ILP solver, a DP solver, and a DP solver with quantization (DP (0 ~ 10)). For quantization, we employ the most aggressive granularity by converting scores to integers ranging from 0 to 10, resulting in the fastest DP solver available for comparison. Figure 9 shows the execution time. (note the log-scale on the y-axis). If the total token cost for a table is below the budget, the table is included without reduction. Table 8 shows the number of tables reduced for each budget. The results show that the ILP solver requires much higher execution time for both tasks. In contrast, *OM-CS* shows a decreasing execution time as the budget increases. This is because of the reduced number of tables that need to be sampled and the fact that *OM-CS* only requires running the LP solution once per table. In the SOTAB27CTA task, *OM-CS* achieves up to a 94× speedup compared to the ILP solver and a 9× speedup over DP (0 ~ 10). Similarly, for SOTAB41RE, *OM-CS* attains up to a 131× speedup over ILP and an 11× speedup compared to DP (0 ~ 10).

Table 7 shows the Micro F1 performance comparison for these solvers, averaged over all budgets. While **CENTS** with *OM-CS* leads to a small drop in performance compared with using an ILP-based solver, the drop is rather minimal, and thus justifies the two orders of magnitude improvement in execution time. The drop is even less significant compared with the DP solver.

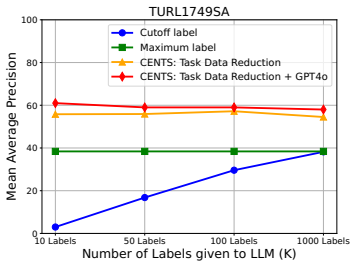


Figure 10: Analysis of different choices of *Task Data Reducer* and K in CENTS for TURL1749SA.

5.2.6 Impact of budget and model. We discuss the impact of varying the budget B and the LLM on **CENTS**'s performance. We report our findings at the bottom of Table 5. We gradually reduce the token budget from 8000 to 2000, decreasing it by 2000 each time. As shown in Table 8, each 2000 token decrease corresponds to reducing approximately 5% more tokens on average per table for both SOTAB27CTA and SOTAB41RE. The results show that reducing the budget from 8000 to 2000 does not lead to a significant performance drop for either the CTA or RE. For CTA with GPT-3.5-turbo, decreasing the budget to 2000 results in a Micro F-1 drop of 1, while for RE, the drop is only 1.6. In contrast, with GPT-4O, reducing the budget to 2000 causes a Micro F-1 drop of only 0.4 for CTA and 0.6 for RE. With Deepseek-R1-Distill-Llama-8B [12], reducing the budget to 2000 causes a Micro F-1 drop of only 0.4 for CTA and 0.6 for RE. For TableGPT2 [38], we use its standalone decoder without its special table encoder since it is not released, but it still outperforms Deepseek-R1-Distill-Llama-8B with a smaller model size.Interestingly, both Deepseek-R1-Distill-Llama-8B and TableGPT2 show large performance drops for both CTA and RE across all budgets compared to the OpenAI models. We further evaluate **CENTS**'s effectiveness under very small token budgets on SOTAB27CTA, and present our findings in Figure 1. Our results show when the token budget is extremely small (e.g., 100 token budget), **CENTS**'s performance eventually drops. These findings demonstrate three things: First, using a more advanced model improves end-to-end performance across all tasks Second, **CENTS** effectively selects the most representative information for any given budget. By contrast, random sampling and CHORUS consistently underperform at the same budget, demonstrating that **CENTS** is more cost-effective. Finally, while locally deployable LLM exist, larger hosted LLMs provide better performance on these tasks, highlighting the cost-saving benefits of **CENTS**.

5.2.7 Performance of the Task Data Reducer. We now discuss how *Task Data Reducer* strategies affect **CENTS**'s performance on the SA task in Figure 10. We observe that using *Cutoff* (i.e., selecting the first K labels given in a random order) as our *Task Data Reducer* results in suboptimal performance. This is because the actual ground-truth labels may not be included in the reduced *Task Data* provided to the LLM. Even the final point at $K = 1000$ labels performs worse than our method. We further increase K above 1000 for the *Cutoff* baseline until the context-window limit is reached. We plot this as a straight line, labeled *Maximum label* and the results are not much better than *Cutoff* with $K = 1000$. This is because the labels are not sorted. Our *Task Data Reducer* method, as shown in

the figure, effectively reduces *and* ranks the more likely labels at the front of the prompt, achieving significant improvements over the Cutoff and Maximum labels methods. The issue of the LLM becoming confused still exists as we increase K, leading to a slight decline in performance for both **Cents** with all models.

6 DISCUSSION

In this section, we discuss alternative solutions to table understanding, and how to extend **Cents** to other problem settings.

Pre-trained Language Models (PLM) for table understanding. LLMs are known to perform well even on tasks they have not been trained for [2, 36, 44]. On the other hand, PLMs rely on task-specific fine-tuning, which gives strong performance on the same task but limits their generalizability to new tasks or data distributions [8]. For this reason, they outperform purely LLM-prompting-based solutions on in-distribution data using extra training data [11, 20, 34]. In line with these prior findings, fine-tuning Doduo [39], a recent state-of-the-art PLM-based method, on our benchmarks gives a Micro F1 of 87.3, a Micro F1 of 86.6, and a MAP of 87 on SOTAB27CTA, SOTAB41RE, and TURL1748SA, respectively, outperforming the LLM-based approach. Trying to use Doduo without pre-training does not work: Using Doduo off-the-shelf without fine-tuning only produces embeddings [39]. Off-the-shelf Doduo, fine-tuned on the TURL [7] dataset, produces labels for the TURL benchmark. Using *exact string match*, there is no overlap with the labels needed for our benchmarks. CHORUS [20] further showed that by manually mapping labels to a new dataset, these PLM-based methods exhibit *significant performance drops* on new target sets. Moreover, PLM methods require well-labeled datasets and task-specific architectures *for each benchmark* [7]. Thus, while PLMs achieve higher performance on in-distribution data, LLM-based approaches can be more practical when task-specific training data is not available.

Locally deployable small LMs for table understanding We designed **Cents** with hosted LMs in mind, as those systems offer excellent performance. Our experiments in Table 5 with Deepseek-R1-llama-8B [12] and TableGPT2 [38] show that their performance fall short compared to closed-source models like GPT-3.5-turbo. Nevertheless, **Cents** can still be useful with locally deployable small LMs [9, 12, 17] because reducing the prompt length improves query throughput given the $O(n^2)$ time complexity of attention mechanisms, where n is the input sequence length [41]. And, indeed, we also saw reduction in LLM inference times in Figure 7.

Extending Cents to other tasks. **Cents**'s main contribution is a method to reduce tables. Such a method can be helpful in tasks beyond table understanding. For example, LLM-based methods have been proposed for schema matching [27, 31], in which an LLM is used to judge the similarity between query columns and candidate table columns. **Cents** could be applied to this setting by reducing the size of the candidate table in the input prompt sent to the LLM to save costs. However, for LLM-based TableQA [27], where an answer can appear anywhere in a table and the full context of the table is required, **Cents**'s reduction method may not be useful.

Few-Shot Learning. Few-shot learning with LLMs—by including task-specific demonstrations in the prompt—can enhance performance [2]. Extending **Cents** to support few-shot learning is also feasible by processing the demonstration examples with **Cents**. Due to space constraints, we leave this extension for future work.

7 RELATED WORK

Table understanding. Table understanding has been a popular topic in the data management community. Limaye et al. [29] proposed to annotate tables at different granularities (i.e., cell, column, and pairs of columns). Early approaches [16, 49] used features from tables or leveraged topic model and use machine learning-for CTA. Later studies [7, 13, 32, 39, 48] used PLMs to improve the performance of table understanding tasks. These solutions could be divided into two categories: Some approaches [7, 13, 48] pre-train a language model for tabular data that can be applied to several downstream tasks, while others [32, 39] fine-tune a pre-trained model with task specific training data. Specifically, Doduo [39] devises a multi-task learning framework to support different table annotation tasks with a shared encoder; Watchog [32] focuses on the semi-supervised where there are much fewer labeled training instances. These approaches all require separated training data for each specific task. In contrast, **Cents** prompts over LLM-based solutions and thus does not require labeled data for fine-tuning.

Recently, some early trials proposed to use LLMs for table understanding tasks. Some approaches [11, 23, 45] explored prompting strategies and optimizations specifically for CTA. CHORUS [20] developed a flexible framework that enables the synthesis of multiple table related tasks but not consider the issues of cost reduction for prompts. TableGPT [27] conducted extensive fine-tuning over GPT to accommodate for the tabular structure and support a multiple table related tasks. Our work is orthogonal to them and could be integrated with them to further improve their performance.

There has also been progress in creating table understanding benchmarks. Viznet [14] constructs a corpus with ground truth for different table-related tasks including CTA. WDC [24] is a large table corpus for dataset search and entity matching. TURL [7] provides dataset for seven downstream table understanding tasks. Gittable [15] is a Github-based large table corpus. SOTAB [23] provides annotated benchmarking tasks for CTA and RE. ArcheType [11] customized existing datasets for LLM-based solutions with label mapping mechanisms.

LLM Cost-effectiveness. There are efforts in machine learning to provide cost-effective LLM solutions. For prompt compression, some approaches [18, 28] use a smaller LM to remove tokens in natural language prompts. Others [33, 46] use task-specific datasets to train models to reduce prompts. FrugalGPT [4] provides high-level guidance and requires task-specific datasets for training to decide whether to use a cheaper LLM. PromptIntern [53] fine-tunes LLM to internalize prompt knowledge. In contrast, **Cents** does not train model to reduce input tables (our proposed label reduction is a side optimization), and focuses specifically on *tabular data for table understanding* tasks. **Cents** is thus orthogonal to these approaches.

8 CONCLUSION

In this paper, we developed **Cents**, a flexible, unified, and cost-effective framework for LLM-based table understanding solutions. **Cents** strategically reduces input tables in prompts through a novel scoring method and an efficient selector, resulting in high F1 scores at a lower cost, and with a rapid execution time. **Cents** also provide label sets reduction as an optimization to improve performance of LLM-based solutions. Experiments show that **Cents** outperforms existing methods at the same or lower cost.

REFERENCES

[1] Dan Brickley, Matthew Burgess, and Natasha F. Noy. 2019. Google Dataset Search: Building a search engine for datasets in an open Web ecosystem. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*. ACM, 1365–1375.

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 159, 25 pages.

[3] Michael J. Cafarella, Alon Y. Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. WebTables: exploring the power of tables on the web. *Proc. VLDB Endow.* 1, 1 (2008), 538–549.

[4] Lingjiao Chen, Matei Zaharia, and James Zou. 2023. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. *CoRR* abs/2305.05176 (2023).

[5] Xingguang Chen and Sibo Wang. 2021. Efficient Approximate Algorithms for Empirical Entropy and Mutual Information. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. ACM, 274–286.

[6] Tianji Cong, Madelon Hulsebos, Zhenjie Sun, Paul Groth, and H. V. Jagadish. 2023. Observatory: Characterizing Embeddings of Relational Tables. *Proc. VLDB Endow.* 17, 4 (2023), 849–862.

[7] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: Table Understanding through Representation Learning. *Proc. VLDB Endow.* 14, 3 (2020), 307–319.

[8] 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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, 4171–4186.

[9] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. 2024. The Llama 3 Herd of Models. *CoRR* abs/2407.21783 (2024).

[10] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J. Miller. 2023. Semantics-aware Dataset Discovery from Data Lakes with Contextualized Column-based Representation Learning. *Proc. VLDB Endow.* 16, 7 (2023), 1726–1739.

[11] Benjamin Feuer, Yurong Liu, Chinmay Hegde, and Juliana Freire. 2024. ArcheType: A Novel Framework for Open-Source Column Type Annotation using Large Language Models. *Proc. VLDB Endow.* 17, 9 (2024), 2279–2292.

[12] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).

[13] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. TaPas: Weakly Supervised Table Parsing via Pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*. Association for Computational Linguistics, 4320–4333.

[14] Kevin Zeng Hu, Snehalkumar (Neil) S. Gaikwad, Madelon Hulsebos, Michiel A. Bakker, Emanuel Zgraggen, César A. Hidalgo, Tim Kraska, Guoliang Li, Arvind Satyanarayan, and Çagatay Demiralp. 2019. VizNet: Towards A Large-Scale Visualization Learning and Benchmarking Repository. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*. ACM, 662.

[15] Madelon Hulsebos, Çagatay Demiralp, and Paul Groth. 2023. GitTables: A Large-Scale Corpus of Relational Tables. *Proc. ACM Manag. Data* 1, 1 (2023), 30:1–30:17.

[16] Madelon Hulsebos, Kevin Zeng Hu, Michiel A. Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César A. Hidalgo. 2019. Sherlock: A Deep Learning Approach to Semantic Data Type Detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*. ACM, 1500–1508.

[17] Mojan Javaheripi, Sébastien Bubeck, Marah Abdin, Jyoti Aneja, Sebastien Bubeck, Caio César Teodoro Mendes, Weizhu Chen, Allie Del Giorno, Ronen Eldan, Sivakanth Gopi, et al. 2023. Phi-2: The surprising power of small language models. *Microsoft Research Blog* (2023).

[18] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 13358–13376. https://doi.org/10.18653/V1/2023.EMNLP-MAIN.825

[19] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Trans. Big Data* 7, 3 (2021), 535–547. https://doi.org/10.1109/TBDATA.2019.2921572

[20] Moe Kayali, Anton Lykov, Ilias Fountalis, Nikolaos Vasiloglou, Dan Olteanu, and Dan Suciu. 2024. CHORUS: Foundation Models for Unified Data Discovery and Exploration. *Proc. VLDB Endow.* 17, 8 (2024), 2104–2114.

[21] Aamod Khatiwada, Roee Shraga, Wolfgang Gatterbauer, and Renée J. Miller. 2022. Integrating Data Lake Tables. *Proc. VLDB Endow.* 16, 4 (2022), 932–945.

[22] Keti Korini and Christian Bizer. 2023. Column Type Annotation using ChatGPT. In *Joint Proceedings of Workshops at the 49th International Conference on Very Large Data Bases (VLDB 2023), Vancouver, Canada, August 28 - September 1, 2023 (CEUR Workshop Proceedings)*, Vol. 3462. CEUR-WS.org.

[23] Keti Korini, Ralph Peeters, and Christian Bizer. 2022. SOTAB: The WDC Schema.org Table Annotation Benchmark. In *SemTab@ISWC*. https://api.semanticscholar.org/CorpusID:255943681

[24] Oliver Lehmberg, Dominique Ritze, Robert Meusel, and Christian Bizer. 2016. A Large Public Corpus of Web Tables containing Time and Context Metadata. In *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11-15, 2016, Companion Volume*. ACM, 75–76.

[25] Quinn Leng, Jacob Portes, Sam Havens, Matei Zaharia, and Michael Carbin. 2024. Long context rag performance of large language models. *arXiv preprint arXiv:2411.03538* (2024).

[26] Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *NeurIPS*.

[27] Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2024. Table-GPT: Table Fine-tuned GPT for Diverse Table Tasks. *Proc. ACM Manag. Data* 2, 3 (2024), 176.

[28] Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. 2023. Compressing Context to Enhance Inference Efficiency of Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 6342–6353. https://doi.org/10.18653/V1/2023.EMNLP-MAIN.391

[29] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. 2010. Annotating and Searching Web Tables Using Entities, Types and Relationships. *Proc. VLDB Endow.* 3, 1 (2010), 1338–1347.

[30] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the Middle: How Language Models Use Long Contexts. *Trans. Assoc. Comput. Linguistics* 12 (2024), 157–173.

[31] Yurong Liu, Eduardo Pena, Aecio Santos, Eden Wu, and Juliana Freire. 2024. Magneto: Combining Small and Large Language Models for Schema Matching. *arXiv preprint arXiv:2412.08194* (2024).

[32] Zhengjie Miao and Jin Wang. 2023. Watchog: A Light-weight Contrastive Learning based Framework for Column Annotation. *Proc. ACM Manag. Data* 1, 4 (2023), 272:1–272:24.

[33] Jesse Mu, Xiang Li, and Noah D. Goodman. 2023. Learning to Compress Prompts with Gist Tokens. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/3d77c6dcc7f143aa2154e7f4d5e22d68-Abstract-Conference.html

[34] Avanika Narayan, Ines Chami, Laurel J. Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *Proc. VLDB Endow.* 16, 4 (2022), 738–746.

[35] OpenAI. 2023. GPT-4. Available at https://openai.com.

[36] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 2011, 15 pages.

[37] Pouya Pezeshkpour and Estevam Hruschka. 2024. Large Language Models Sensitivity to The Order of Options in Multiple-Choice Questions. In *Findings of the Association for Computational Linguistics: NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, Kevin Duh, Helena Gómez-Adorno, and Steven Bethard (Eds.). Association for Computational Linguistics, 2006–2017. https://doi.org/10.18653/V1/2024.FINDINGS-NAACL.130

[38] Aofeng Su, Aowen Wang, Chao Ye, Chen Zhou, Ga Zhang, Guangcheng Zhu, Haobo Wang, Haokai Xu, Hao Chen, Haoze Li, Haoxuan Lan, Jiaming Tian, Jing Yuan, Junbo Zhao, Junlin Zhou, Kaizhe Shou, Liangyu Zha, Lin Long, Liyao Li, Pengzuo Wu, Qi Zhang, Qingyi Huang, Saisai Yang, Tao Zhang, Wentao Ye, Wufang Zhu, Xiaomeng Hu, Xijun Gu, Xinjie Sun, Xiang Li, Yuhang Yang, and Zhiqing Xiao. 2024. TableGPT2: A Large Multimodal Model with Tabular Data Integration. arXiv:2411.02059 [cs.LG] https://arxiv.org/abs/2411.02059

[39] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çagatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2022. Annotating Columns with Pre-trained Language Models. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. ACM, 1493–1503.

[40] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. *CoRR* abs/2302.13971 (2023).

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NIPS*. 5998–6008.

[42] Daheng Wang, Prashant Shiralkar, Colin Lockard, Binxuan Huang, Xin Luna Dong, and Meng Jiang. 2021. TCN: Table Convolutional Network for Web Table Interpretation. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*. ACM / IW3C2, 4020–4032.

[43] Runhui Wang, Yuliang Li, and Jin Wang. 2023. Sudowoodo: Contrastive Self-supervised Learning for Multi-purpose Data Integration and Preparation. In *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023*. IEEE, 1502–1515.

[44] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent Abilities of Large Language Models. *Trans. Mach. Learn. Res.* 2022 (2022).

[45] Lindsey Linxi Wei, Guorui Xiao, and Magdalena Balazinska. 2024. RACOON: An LLM-based Framework for Retrieval-Augmented Column Type Annotation with a Knowledge Graph. In *NeurIPS 2024 Third Table Representation Learning Workshop*. https://openreview.net/forum?id=unlufkzs7v

[46] Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2024. RECOMP: Improving Retrieval-Augmented LMs with Context Compression and Selective Augmentation. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=mlJLVigNHp

[47] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. 2012. InfoGather: entity augmentation and attribute discovery by holistic matching with web tables. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*. ACM, 97–108.

[48] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*. Association for Computational Linguistics, 8413–8426.

[49] Dan Zhang, Yoshihiko Suhara, Jinfeng Li, Madelon Hulsebos, Çagatay Demiralp, and Wang-Chiew Tan. 2020. Sato: Contextual Semantic Type Detection in Tables. *Proc. VLDB Endow.* 13, 11 (2020), 1835–1848.

[50] Li Zhang, Shuo Zhang, and Krisztian Balog. 2019. Table2Vec: Neural Word and Entity Embeddings for Table Population and Retrieval. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019*. ACM, 1029–1032.

[51] Shuo Zhang and Krisztian Balog. 2017. EntiTables: Smart Assistance for Entity-Focused Tables. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, Noriko Kando, Tetsuya Sakai, Hideo Joho, Hang Li, Arjen P. de Vries, and Ryen W. White (Eds.). ACM, 255–264.

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

[53] Jiaru Zou, Mengyu Zhou, Tao Li, Shi Han, and Dongmei Zhang. 2024. PromptIntern: Saving Inference Costs by Internalizing Recurrent Prompt during Large Language Model Fine-tuning. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*. Association for Computational Linguistics, 10288–10305.