Columbo: Expanding Abbreviated Column Names for Tabular Data Using Large Language Models

Ting Cai, Stephen Sheen, AnHai Doan
University of Wisconsin-Madison
{tcai35, sheen2, ahdoan}@wisc.edu

Abstract

Expanding the abbreviated column names of tables, such as "esal" to "employee salary", is critical for many downstream NLP tasks for tabular data, such as NL2SQL, table QA, and keyword search. This problem arises in enterprises, domain sciences, government agencies, and more. In this paper, we make three contributions that significantly advance the state of the art. First, we show that the synthetic public data used by prior work has major limitations, and we introduce four new datasets in enterprise/science domains, with real-world abbreviations. Second, we show that accuracy measures used by prior work seriously undercount correct expansions, and we propose new synonym-aware measures that capture accuracy much more accurately. Finally, we develop Columbo, a powerful LLM-based solution that exploits context, rules, chain-of-thought reasoning, and token-level analysis. Extensive experiments show that Columbo significantly outperforms NameGuess, the current most advanced solution, by 4-29%, over five datasets. Columbo has been used in production on EDI, a major data lake for environmental sciences.

1 Introduction

Tabular data is ubiquitous in companies, domain sciences, government agencies, and others (Zhang et al., 2023; Hanson, 2025). Using this data well, however, has been difficult. A major reason is that the names of the tables and columns are often abbreviated, appearing quite cryptic (see Figure 1 and Appendix A.1). This makes it hard for downstream NLP tasks to process the tables.

In particular, a recent work (Zhang et al., 2023) shows that using abbreviated column names significantly reduce accuracy for three NLP tasks, NL2SQL, schema-based relation detection, and table QA, by 10.54%, 40.5%, and 3.83%, respectively. That work also argues that expanding column names improves the readability of the tables, enables data integration, and improves keyword search (to discover relevant tables). Another recent work (Luoma and Kumar, 2025) shows that abbreviated column names significantly reduce the accuracy of NL2SQL. Our own experience working with a major data lake (edirepository.org) also shows that expanding column names improves the accuracy of keyword search, schema matching (i.e., finding columns that are semantically the same), and column annotation with the concepts in a given ontology (see "Columbo in Production" in the experiment section).

```
EMPS(eName, eSal, eDTPh, ...)
Ref_Sectors_GICS(GICS_IND_GRP_CD, ...)
1997lgNutSExt(Date, canWt, hclWt, corWetWt, ...)
```

Figure 1: Examples of abbreviated table and column names in companies and domain sciences.

Thus, table/column name expansion is a core challenge for the fast growing direction of NLP tasks for tabular data, and has received growing attention (Zhang et al., 2023; Luoma and Kumar, 2025; Sawant and Sonawane, 2024; Anonymous, 2025; Singh et al., 2025). The goal is to expand table/column names into meaningful English phrases, such as "eSal" to "Employee Salary", "eDTPh" into "Employee Day Time Phone", "1997lgNutSExt" into "1997 Long-term Nutrient Study Experiment", and so on. Clearly, this can enormously help downstream applications. For example, given a user query "employee phone", a keyword search application can correctly return the table "EMPS(eName, eSal, eDTPh, ...)" if it knows that "eDTPh" means "Employee Day Time Phone".

As far as we can tell, the most advanced work for this challenge is NameGuess in EMNLP-2023 by Amazon AWS (Zhang et al., 2023). That work focuses on column name expansion and frames it as a natural language generation problem. It proposes a solution that uses LLMs to obtain 69.3%

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 24774–24792
November 4-9, 2025 ©2025 Association for Computational Linguistics

exact-match accuracy (using GPT4), in contrast to 43.4% accuracy obtained by human. NameGuess has clearly showed the promise of using LLMs to expand abbreviated column names. But it has three major limitations, as we discuss below. In this paper we describe Columbo, which addresses these limitations and significantly advances the state of the art.

First, NameGuess experimented with just one dataset, which is *public data* from the cities of San Francisco, Chicago, and Los Angeles (a.k.a. Open City Data). We show later that LLMs achieve lower accuracy on *enterprise data* (coming from companies) and *domain science data*. Thus, we believe that *a good benchmark for column name expansion cannot contain just public data*. In this work we introduce four more datasets that come from enterprises and domain sciences.

Another problem is that NameGuess *synthetically* creates the abbreviated column names, e.g., by randomly dropping, shuffling, or replacing characters from English phrases. So many abbreviated column names in its dataset look "unrealistic", e.g., "r" for "area", "mmj" for "medical", and LLMs struggle to correctly expand such names. Given that we do not yet have good methods to synthetically create abbreviated column names, we believe that *a good benchmark for column name expansion should also contain abbreviated column names that come from real data*. The four datasets introduced in this paper contain column names abbreviated by humans. Later we show that LLMs indeed can leverage their vast knowledge about real-world abbreviation patterns to correctly expand these names.

The second limitation is that NameGuess computes accuracy in a restricted way. The most important accuracy measure, exact match, computes the fraction of columns where the name predicted by LLMs *exactly matches* the "gold" name. This penalizes cases of minor variations, e.g., "geography identifier" vs "geographical identifier", "photo credit" vs "picture credit", etc. To solve this, we introduce a new accuracy measure called "synonym-aware exact match". We show that this new measure captures the performance of column name expansion solutions much more accurately.

The third limitation is that NameGuess uses a rather basic LLM solution. It simply asks the LLM to expand the names of the columns (given a few expansion examples). We have developed a significantly more powerful solution. Our solution provides a lot of context information to the LLM, e.g., the name of the target table and the topics of similar tables (we infer these topics using LLMs). We ask the LLM to follow a set of rules (that help generate the correct expansions) and use chain-of-thought reasoning. Finally, we reason about column name expansion at *the token level*, i.e., we translate each column name into a sequence of tokens, expand each token into an English phrase, then combine the phrases to obtain the column name expansion. Together, these features help our solution significantly improve accuracy compared to NameGuess. In summary, we make the following contributions:

- We show that synthetic public data is not sufficient for evaluating solutions for column name expansion. We introduce four new non-public datasets with real-world abbreviated column names.
- We show that computing accuracy via exact string matching is problematic. We introduce a new measure that captures accuracy much more accurately.
- We develop Columbo, an LLM-based solution that exploits context, rules, chain-of-thought reasoning, and token-level analysis.
- We provide extensive experiments that show that Columbo outperforms NameGuess on all five datasets, improving the absolute accuracy by 4-29%, and the relative accuracy by 4-46%.

Columbo has been used in production on EDI, a major data lake for environmental sciences. We briefly describe this experience in Section 6. The code and datasets (except Finance and University, for which we do not have permission to release) are available at github.com/anhaidgroup/columbo.

2 Problem Definition

Similar to NameGuess, given a set of tables (e.g., those in a data lake), we seek to expand the column names. Expanding the table names is more complicated, as we discuss in Section 6, and hence is deferred to future work.

We assume that each column name c can be represented as a sequence $t_1 d_1 \ldots d_{n-1} t_n$, where each d_i is a *delimiter* (i.e., a special character such as '_', '-', the space character, or the empty character) and each t_i is a *token* that can be expanded into a meaningful English phrase $e(t_i)$. Phrase $e(t_i)$ must contain all characters of token t_i, in that order.

For example, column name "eSal" can be tokenized into tokens "e" and "Sal", separated by the empty-character delimiter. Token "e" expands to "Employee" and token "Sal" expands to "Salary". Other expansion examples are "Rm" $\rightarrow$ "Room" and "CD"$\rightarrow$ "Certificate Deposit". The expansion $e(c)$ of column c is then the concatenation of the expansions of its tokens.

In many application contexts, we cannot access the data tuples of the tables, for reasons of privacy, compliance, performance, etc. (Lobo et al., 2023). So here we consider the input to be just the (abbreviated) table names and column names.

NameGuess shows that state-of-the-art hosted LLMs such as GPT-4 achieves the highest accuracy for column name expansion (Zhang et al., 2023). As a result, here we focus on these LLMs, specifically on GPT-4o. In future work we will consider open-source LLMs that can be deployed in-house.

3 New Datasets

We now make the case that a good benchmark for column name expansion cannot contain just public data with synthetic abbreviations. It must also contain non-public data with real-world abbreviations. We then introduce four new such datasets.

Specifically, NameGuess uses just one dataset obtained from the Open Data Portals of San Francisco, Chicago, and Los Angeles, covering business, education, health, etc. To evaluate expansion solutions, NameGuess needs both abbreviated column names and the gold (i.e., correct) expansions. To do this, NameGuess uses heuristic rules to find column names that are meaningful English phrases. It abbreviates these names, e.g., by randomly shuffling, dropping, or replacing characters. Finally, it applies solutions to the abbreviated names to see if they can recover the original column names.

Our experiments with the NameGuess dataset revealed two problems. First, we found that LLMs achieve a much higher accuracy on this dataset compared to the four enterprise/science datasets (e.g., 81.5% vs 63.2-73.8% EM accuracy, see Table 3). Prior work has observed the same phenomenon for other data tasks (Demiralp et al., 2024), presumably because today LLMs have been trained on a lot more public data than enterprise and science data. Put differently, we speculate that the NameGuess dataset covers popular concepts (e.g., transportation, education, etc.) that are ubiquitous online, whereas the remaining four datasets cover "rarer" concepts (e.g., specific to a vertical). Thus, if we want to apply expansion solutions to non-public data, we cannot rely on experiments with just public data, as the results can be misleading.

To address this problem, in this paper we use the five datasets described in Table 1. "NG" is the NameGuess dataset. "Finance" and "University" are two datasets obtained from companies, covering the finance and academic domains. "AW" is a variation of the AdventureWork dataset released by Microsoft, and "EDI" is a dataset from the environmental science domain. Appendix A.2 discusses how we generated these datasets.

The second problem with the NameGuess dataset is that it is difficult to accurately mimic human's abbreviation patterns. Thus, many synthetic abbreviated column names look "unrealistic", and LLMs struggle to expand these. Table 3 shows that on the NameGuess dataset, LLMs can only improve EM accuracy from 81.5% to 85.2%. In contrast, the four new datasets have real-world column names already abbreviated by their human creators. Here Table 3 shows that LLMs can exploit their vast knowledge about real-world abbreviation patterns to achieve high accuracy, improving EM accuracies from 63.2-73.8% to 87.6-93.3%. We conclude that a good benchmark for this problem should also contain real-world abbreviated column names, as our four new datasets do.

4 New Accuracy Measures

NameGuess uses three accuracy measures. EM computes the fraction of columns where the predicted expansion x exactly matches the gold expansion g. Word-level F_1 is $2PR/(P+R)$, where P is the fraction of tokens in x that occur in g, and R is the fraction of tokens in g that occur in x. BERT-score F_1 is computed similarly, except that two tokens are considered equivalent if the cosine similarity score of their BERT-based embedding vectors is high (Zhang et al., 2023).

Among these three measures, EM is most intuitive, but as defined, it is too restrictive. It regards cases of minor variations, e.g., "geography location" vs "geographical location", "picture credit" vs "photo credit", as not matched. To address this problem, we examined the five datasets described in Section 3, and created synonym pairs, e.g., "geography" = "geographical", "picture" = "photo". Given a gold expansion g, we use these synonym pairs to create all gold variations, e.g.,

Datasets	Type	# Tables	# Columns	# Columns with Gold	# Columns / Table (max)	# Columns / Table (avg)	# Columns / Table (min)
NG	Public	895	9196	8881	69	10.3	1
Finance	Enterprise	23	443	443	138	19.3	2
University	Enterprise	118	1563	1561	146	13.3	1
AW	Enterprise	101	825	824	35	8.2	2
EDI	Science	251	3830	3481	258	15.3	2

Table 1: Statistics of the 5 datasets used in our experiments.

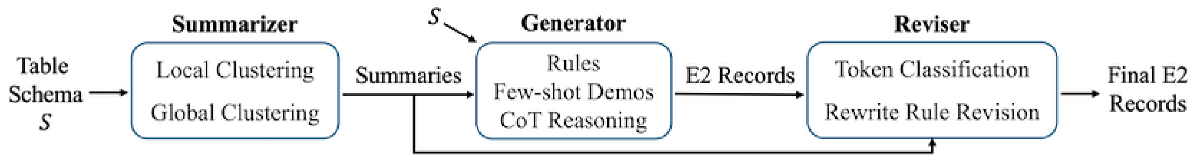

Figure 2: The overall architecture of Columbo.

Datasets	Without Synonyms			With Synonyms		
	EM	F_1	Bert-F_1	EM	F_1	Bert-F_1
NG	70.5	83.6	85.5	81.5	88.8	92.0
Finance	54.4	82.8	78.5	73.8	93.3	88.0
University	50.6	81.0	72.8	63.2	86.3	77.3
AW	64.7	84.3	85.9	72.6	88.3	88.3
EDI	52.5	77.4	68.5	65.3	85.2	79.8

Table 2: NameGuess accuracies without and with synonyms.

creating “geography location” from “geographical location”. Then we declare that a predicted expansion x matches g if it matches any variation of g. We call this new measure *synonym-aware EM*. We modify word-level F_1 and BERT-score F_1 similarly to be synonym aware.

One may wonder if the BERT-based F_1 score captures ideas similar to the synonym-based EM approach. We found that this is not the case, because there are many synonyms that are very specific to a particular vertical. Being trained mostly on public data, BERT is not aware of such synonyms. Examples include "time 0" vs "initial", "site" vs "hub", "log" vs "logarithm", "met" vs "meteorological" in EDI, "prime" vs "principal" in University, and "sedol" vs "stock exchange daily official list" in Finance.

Table 2 shows the accuracies of applying the NameGuess solution to all five datasets. Clearly, the accuracy measures without synonyms significantly underestimate the true accuracies. For example, they show 50.6-70.5% EM, whereas the measures with synonyms show 63.2-81.5% EM. Furthermore, the relative ranking of dataset difficulty as measured by the two EM measures does change. Consider Finance and AW. Table 2 shows that the EM measure with no synonyms ranks Finance harder than AW (54.4% vs 64.7%). But the synonym-aware EM measure ranks Finance easier than AW (73.8% vs 72.6%). For the rest of this paper, we use the synonym-aware accuracy measures.

5 The Columbo Solution

We now describe the Columbo solution, which improves upon NameGuess. Given a table T, NameGuess sends batches of 10 column names from T to the LLM. It provides several examples of column name expansion, e.g., “c_name”→ “customer name”, then asks the LLM to expand the above 10 column names. NameGuess does not exploit any additional information, e.g., the table name. In contrast, Columbo exploits the names of table T and related tables, rules, chain-of-thought reasoning, token-level analysis, and more.

Specifically, the key insight behind Columbo is that, to solve this problem well, we should use all available information and reason at a deeper level. Consequently, we use LLMs because to expand a column name correctly, we need a lot of domain knowledge. LLMs have been trained on tons of data and can be viewed as very large stores of domain knowledge. So applying LLMs to this problem is promising. Second, we observed that when expanding column names, LLMs keep making similar mistakes. This is why we formulate a set of

1 **Inputs**: Set S of table schemas
2 **For** each batch of k schemas in S **do**
3 Use LLM to summarize and cluster schemas
4 Get cluster summaries $\{(G_i, d_{G_i})\}_{i=1}^{m}$, where
5 - $G_i = \{(T_j, d_{t_j})\}_{j=1}^{n}$, where d_{t_j} is the summary for table T_j
6 - d_{G_i} is the summary for cluster G_i
7 **For** any clusters G_i and G_j in all clusters **do**
8 **If** $d_{G_i} = d_{G_j}$ **then** Merge G_i and G_j into one cluster

9 **For** each table T in S **do**
10 **For** each batch of p column names in T **do**
11 Use LLM to generate E2 records for column names
12 Get E2 record (t, r) for each column name c_i, where
13 - $t = \{t_1, \cdots, t_u\}$ is a sequence of tokens of c_i
14 - $r = \{t_j \rightarrow e_j\}_{j=1}^{u}$ are expansion rules for the tokens

15 **For** each unique token t in all E2 records **do**
16 **If** t satisfies filtering rules **then**
17 Use LLM to decide if t has a unique expansion e
18 **If** e exists **then**
19 Apply e to all expansion rules of t in all E2 records
20 **Return** expanded column names for all tables in S

Figure 3: The pseudo code of Columbo.

general rules (in the prompt) telling LLMs not to make these mistakes. Third, we observed that there is a well-known chain-of-thought (CoT) process that a human typically follows to expand a column name: first expand each token, then combine these token-level expansions to obtain the column-level expansion. This suggests that CoT can be well matched to this problem. Finally, we perform token-level analysis by identifying tokens with potentially incorrect expansion rules and fixing those.

Figure 2 describes the Columbo architecture, which consists of 3 modules: Summarizer, Generator, and Reviser. We now describe these modules, then discuss the rationales behind the design decisions.

The Summarizer: This module creates two kinds of summaries to be used by subsequent modules (see Lines 1-8 of the algorithm in Figure 3). Let S be the set of table schemas for which we want to expand the column names. We first send batches of k table schemas from S to the LLM, and ask it to cluster the k tables into groups (currently $k = 30$). For each group G, we ask the LLM to provide a group summary d_G, which is a short English phrase that best summarizes G, and similarly, for each table T in group G, we also ask for a table summary d_T that best summarizes T.

Appendix A.3 shows the prompt for a sample batch and the output of the LLM. For example, it shows that 3 tables BusinessEntity(ModDate, bID,rID), BusinessEntityAddress(ModDate, aID, aTypeID, bID, rID), BusinessEntityContact(ModDate, PersonID, bID, cTypeID, rId) have been clustered into a group, with group summary "Business Entity Structure", and that Table BusinessEntity(ModDate, bID,rID) has the summary "Represents a generic business entity that can be a person, vendor, or customer".

After processing all tables in S (via batches of up to k tables), we perform a "global merge" that merges all groups with the same summary. Thus, the Summarizer produces a clustering of all input tables into groups, where each group G has a summary d_G, and each table T has a summary d_T.

The Generator: This module expands the column names (see Lines 9-14 of the algorithm in Figure 3). Specifically, for each table T (in the set S), we send batches of p columns of T to the LLM, and ask it to expand the p column names (currently $p = 10$). We structure the LLM prompt for each batch as follows.

First, we provide the context, which is the name of table T, as well as the names and table summaries of up to q tables (randomly sampled) from the same group as T (currently $q = 100$).

Second, we specify a set of rules, e.g., "expand all abbreviations in a column name", "do not expand numbers", "do not add extra words or explanations", etc.

Finally, unlike NameGuess which just asks the LLM for the expansion $e(c)$ of a given column name c, we ask the LLM to provide a chain-of-thought reasoning that leads to the expansion. Specifically, we ask the LLM to parse column name c into a sequence of tokens, then provide the expansion for each token, then concatenate these expansions into $e(c)$. We provide a few examples of such reasoning in the prompt.

Thus, for each input column name c, this module produces a sequence of tokens $t_1 \ldots t_n$, and expansion rules for the tokens: $t_1 \rightarrow e_1, \ldots, t_n \rightarrow e_n$. We call this output the E2 record for column c (where "E2" stands for "expansion & explanation"). The expansion of c is then the string $e_1 \ \ldots \ e_n$. Appendix A.4 shows the prompt for a sample set of columns and the output of the LLM.

The Reviser: This module improves upon the output of the Generator, by identifying tokens with potentially incorrect expansion rules, then trying to fix those (see Lines 15-20 of the algorithm in Figure 3).

Specifically, we first process the E2 records (output by the Generator) to identify the set P of all tokens that have more than one expansion rule, e.g., $dt \rightarrow date$ and $dt \rightarrow data$. Next, for each token $x \in P$, we ask the LLM if x should have just one expansion rule, and if so, to identify that rule.

To help the LLM make the above decisions, we provide it with the summaries of all groups, and all expansion rules of token x (that we have identified from the E2 records). For each expansion rule of x, we also provide its frequency (i.e., the number of column names in which that expansion rule is used), and a sample table schema in which that expansion rule is used.

We optimize the above process by using a set of rules to decide which tokens in P to send to the LLM. For example, if a token $x \in P$ has too few characters (currently set to 1), then we do not send x to the LLM, because it is likely that x has more than one correct expansion, e.g., $e \rightarrow employee$ and $e \rightarrow electronic$.

Suppose the LLM has identified a set Q of pairs (x, e), where token x should always be expanded into e. Then we use Q to modify the E2 records (produced by the Generator). The Reviser outputs the modified E2 records as final records, and the expansions of the column names can be quickly obtained from these E2 records. Appendix A.5 shows the prompt for a sample token and the output of the LLM.

Discussion: We now discuss the rationales behind the major design decisions. Consider the Generator. This module asks the LLM to perform chain-of-thought reasoning in which it translates the column name into a sequence of tokens, then finds the expansion of each token. We found that this improves the LLM's accuracy. Further, this gives us the tokens and their expansion rules, which enable token-level analysis, such as the one carried out by the Reviser. We also found that just supplying examples of column name expansion was not enough. LLMs were still prone to producing incorrect output, e.g., adding extra words, explanations, changing word orders, etc. Adding rules asking the LLM not to do so helps improve accuracy (see the experiments).

Intuitively, providing context can help expand column names. For example, the LLM may incorrectly expand column RUSS_CD to "Russian Code". But being told this column is in Table RUSSELL_INDEX, it can correctly expand the column to "Russell Code". So we provide the LLM with the name of the target table T. We also cluster all the input tables so that we can find the tables related to T, and provide a subset of these tables to the LLM, as additional context. To provide this subset of tables, we can just send their schemas to the LLM. But it turns out that many of these tables can have a large number of columns (e.g., 200+), making their schemas too large. This is why in the Summarizer we ask the LLM to provide for each table a short table summary. Then instead of sending the full table schema, we just send the (shorter) table name and summary.

Now consider the Reviser. If we can provide the LLM with information about the entire dataset, i.e., the entire set of table schemas S, it can more accurately decide if a token x has just one expansion in S (e.g., RUSS should always be expanded to "Russell"). But sending all table schemas in S to the LLM is impractical. So in the Summarizer we ask the LLM to provide a short group summary for each group of tables, then send the LLM just the summaries of all groups.

Datasets	Columbo			NameGuess		
	EM	F_1	Bert-F_1	EM	F_1	Bert-F_1
NG	**85.2**	91.9	94.5	81.5	88.8	92.0
Finance	**87.6**	96.9	95.6	73.8	93.3	88.0
University	**92.2**	98.2	97.4	63.2	86.3	77.3
AW	**93.3**	97.6	98.2	72.6	88.3	88.3
EDI	**90.7**	95.3	93.6	65.3	85.2	79.8

Table 3: The accuracy of Columbo vs. NameGuess

6 Experiments

We now evaluate Columbo, using the five datasets described in Table 1 (see Section 3). We conducted all experiments using GPT-4o, version gpt-4o-2024-08-06, with temperature 0, max completion tokens 6000, and default values for all other parameters.

Overall Performance: Table 3 compares Columbo with NameGuess, the state-of-the-art solution, using the three synonym-aware accuracy measures described in Section 4. In what follows we focus on the EM accuracy measure, as it is most intuitive. First, the table shows that Columbo significantly outperforms NameGuess on all five

Datasets	Columbo	−co	−t	−r	−cot	−to
NG	85.2	83.7	85.6	84.2	84.8	86.9
Finance	87.6	88.5	83.1	85.3	88.0	87.1
University	92.2	76.6	83.5	82.7	89.2	80.7
AW	93.3	90.4	83.1	85.4	91.0	93.5
EDI	90.7	89.8	89.4	63.6	90.4	90.3

Table 4: Ablation studies for Columbo.

Datasets	ColumboNG	Columbo	EM-diff1	NameGuess	EM-diff2
NG	83.9	85.2	1.3	81.5	2.4
Finance	84.4	87.6	3.2	73.8	10.6
University	69.3	92.2	22.9	63.2	6.1
AW	73.1	93.3	20.2	72.6	0.5
EDI	87.1	90.7	3.6	65.3	21.8

Table 5: Exact Match accuracy of ColumboNG vs. NameGuess

datasets, improving the absolute EM accuracy by 4-29% and the relative EM accuracy by 4-46%.

Second, NameGuess achieves lower EM accuracies on the enterprise (Finance, University, AW) and science (EDI) datasets, compared to the public (NG) dataset: 63.2-73.8% vs 81.5%. This suggests that LLMs perform worse on non-public data (prior work (Demiralp et al., 2024) has reached the same conclusion).

Finally, Columbo is able to improve the EM accuracy of NameGuess on the non-public datasets by a large amount, from 63.2-73.8% to 87.6-93.3%, but it "struggles" to improve the EM accuracy of NameGuess on the public dataset NG, from 81.5% to just 85.2%. We believe this is because many column names in NG look "unrealistic", as they are synthetically abbreviated (as we discussed in Section 4). So the LLM fails to expand many such columns, resulting in a small EM improvement. Overall, the results in Table 3 suggests that a good benchmark for column name expansion should contain non-public data with real-world abbreviated column names.

Ablation Studies: We now evaluate the major components of Columbo. Table 4 shows the EM accuracies of Columbo (the 2nd column) and the five Columbo versions in which we remove a major component. (Appendix A.6 shows the full result which also contains the remaining two accuracy measures.)

First we modify Columbo to not exploit any context information, i.e., removing the Summarizer and disabling using the table and group summaries in the Generator and Reviser. The EM accuracies of this Columbo version are reported in Column "-co" in Table 4.

Second, we modify Columbo to not exploit table names. Specifically, we need table names in the Summarizer to create the summaries, so we keep the Summarizer as is. But we remove all mentions of table names in the Generator and Reviser, using only table summaries where appropriate. The results are in Column "-t".

Third, we remove all nine rules used in the Generator (see Column "-r"). Fourth, we modify the Generator to not use chain-of-thought reasoning. The results are in Column "-cot". Finally, we remove the token-level analysis by disabling the Reviser, and report the results in Column "-to".

Table 4 shows that disabling a component typically leads to a drop in accuracy, sometimes by a lot, as highlighted in blue in the table, e.g., 92.2% of Columbo vs 76.6% of "-co" on University, 90.7% of Columbo vs 63.6% of "-r" on EDI. Occasionally the accuracy increases, as highlighted in red in the table. But this increase is minimal, from 0.2-1.7%. Thus, the results suggest that the components contribute meaningfully to the overall accuracy of Columbo.

We also examined the case where Columbo uses exactly the same input information as NameGuess. To do so, we develop ColumboNG, by removing the table clustering step and token revision step of Columbo, as well as the contextual information in the prompt of the name expansion step. So ColumboNG only has access to the column names (like NameGuess), but it still uses rules, chain-of-thought reasoning, and in-context learning.

Table 5 shows the results (only for the EM accuracy, for space reasons). The table shows that compared to the original Columbo, ColumboNG reduces the EM accuracy by 1.3-22.9% (see Column "EM-diff1"). This is as expected. It suggests that exploiting new information (e.g., the target table names, the names of other tables, etc.) does help improve accuracy, in some cases significantly.

But interestingly, even though exploiting the exact same information as NameGuess, ColumboNG still improves accuracy, by 0.5-21.8% (see Column "EM-diff2"). This suggests that the innovations introduced by ColumboNG, such as rules, using chain-of-thought reasoning, and in-context learning, do help improve accuracy, in some cases significantly.

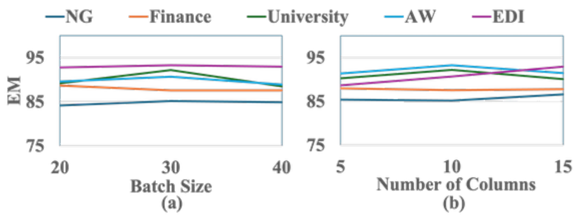

Figure 4: Accuracy of Columbo as we vary the batch size k and the number of columns p.

Sensitivity Analysis: Finally, we examine how Columbo's accuracy changes as we vary the major parameters. Recall that the Summarizer clusters the tables in batches (of size k). We want to know how the order in which the tables are processed impacts the accuracy. So we ran Columbo 3 times, using the original default table order, and 2 more orders in which we randomly shuffled the tables. The results show that the EM accuracy changes minimally across the three runs, by 0.44-3.58%. See the full results in Table 14 in Appendix A.7.

Next, we vary k, the number of tables that the Summarizer clusters in a single batch, from 20 to 40 (the default value for k is 30). Figure 4.a shows that the EM accuracy for all five datasets fluctuates, but in a small range. Finally, we vary p, the number of columns that the Generator processes in a single batch, from 5 to 15 (the default value is 10). Figure 4.b shows that the EM accuracy fluctuates but again within a small range. We conclude that Columbo is robust to small changes in the values of the major parameters.

Columbo in Production: We briefly describe how the EDI team has used Columbo in production on *edirepository.org*. EDI is an online data lake where research groups in environmental sciences submit their data, for other groups to use. EDI has 87K data packages containing 18K tables. To help researchers find desired tables on EDI, the EDI team has been trying to assign the concepts from ESCO, a large ontology that covers environmental domains, to the columns of the 18K tables.

To date, the EDI team has done such assignments manually, and progress is slow. In April 2025 they enlisted our help, asking us to assign ESCO concepts to 36K columns. We frame this problem as a string matching problem, in which given a table A of 2K concept names in ESCO and a table B of 36K column names, find all pairs $(x \in A, y \in B)$ that match.

This problem is challenging because the column names are often abbreviated and cryptic. Applying standard string matching solutions (including using embedding vectors) to this problem produced very low accuracy (less than 30% on a labeled dataset). So we applied Columbo to expand all 36K column names, before applying standard ML-based string matching solutions, achieving 83% accuracy on the labeled dataset. The EDI team examined and judged the results "immensely helpful". They have now incorporated Columbo as a part of their assignment workflow. That is, they use Columbo to expand column names of the new tables, use our ML-based matcher to match these column names with ontology concepts, then manually examine the results returned by the matcher to confirm, reject, or modify the assignments.

This experience, while anecdotal, suggests that expanding column names is critical for downstream tasks, such as assigning ontology concepts, and that Columbo is already sufficiently accurate to be useful in production of some real-world applications.

Table Name Expansion: Finally, while this paper focuses on column names, we have conducted preliminary experiments with expanding table names, on 3 datasets: Finance, University, and EDI. Asking the LLM to expand the table names, given the abbreviated column names, produces EM accuracy of 91.3, 98.31, 61.7%, respectively. Surprisingly, giving the LLM the expanded column names does not notably improve the EM accuracy, achieving 91.3, 96.6, and 64.5%, respectively.

We then examined EDI, where the EM accuracy is lowest, and asked the LLM to expand the table names, given the gold column names. Also surprisingly, this improves the EM accuracy by only 0.7%. We found that the EDI table names often use phrases that are *not* present in the column names, e.g., for table "EVRT1980_TILLER", token "EVRT" does not appear in any column name, and for table "2006_JD_SnowShrub", token "JD" is the name of the data uploader and does not appear in any column name.

The above result suggests that expanding table names may require additional information, such as from textual table descriptions. As a result, we defer this problem to future research.

7 Related Work

Abbreviation Expansion: This task has been studied extensively. Many earlier approaches formulate it as a classification problem, i.e., choosing the most likely expansion from a predefined candidate set based on surrounding text (Roark and Sproat, 2014; Gorman et al., 2021; Ammar et al., 2011; Pouran Ben Veyseh et al., 2020). Other lines of research expand abbreviations in specific domains like informal text (Gorman et al., 2021) and SMS messages (Cai et al., 2022). (Du et al., 2019) expand prefix-abbreviations in biomedical text. These works differ from ours as they often rely on different types of context (e.g., free-form text) or target different abbreviation styles than those found in tables.

The work most closely related to ours is NameGuess (Zhang et al., 2023), which specifically expand abbreviated column names in tables. NameGuess introduced a benchmark dataset (based on synthetic abbreviations) and showed the potential of LLMs, even outperforming fine-tuned models with one-shot prompting. However, NameGuess relies on the column names themselves, without incorporating table names or broader schema context, and their evaluation was limited to a single public dataset. Another relevant study (Luoma and Kumar, 2025) also investigates abbreviated column names but focuses on identifying the level of abbreviation and analyzing its impact on downstream tasks like natural language queries, rather than proposing an expansion method. Our work leverages richer schema context and more sophisticated LLM prompting techniques for improved expansion accuracy. Finally, the work (Anonymous, 2025) follows up on (Zhang et al., 2023) and focuses on generating more realistic abbreviations from English phrases.

Table Understanding and Enrichment: Expanding abbreviated column names is a specific instance of the broader goal of enriching table metadata to enhance data understanding (Fang et al., 2024), discovery (Freire et al., 2025), and usability for downstream tasks. Table-to-text (Zhao et al., 2023b,a; Kasner et al., 2023; Yang et al., 2022; Gong et al., 2019) and table question answering (Pal et al., 2023; Xie et al., 2022; Herzig et al., 2020) aim at developing models able to understand structured tabular data and natural language questions to perform reasoning and tasks across tables.

A significant body of work (Deng et al., 2022; Feuer et al., 2024; Hulsebos et al., 2019; Suhara et al., 2021; Zhang et al., 2020; He et al., 2021; Hulsebos et al., 2023) focuses on inferring the semantic type of data within table columns (e.g., tagging columns as 'zip code', 'address', 'date'). While related, semantic type detection differs fundamentally from our task; it is typically framed as a classification problem (assigning a type from a predefined ontology) based on column values, whereas we focus on generating a natural language expansion based on the abbreviated column name and schema context.

LLMs have also been employed to generate natural language descriptions for tables (Gong et al., 2020; Zhang et al., 2025; Gao and Luo, 2025; Anonymous, 2024; Han et al., 2025) or individual columns (Wretblad et al., 2024). These descriptions provide valuable semantic context but do not directly address the problem of resolving cryptic abbreviations within column names themselves.

Other related tasks involve matching schemas across different tables or mapping table columns to concepts in external knowledge graphs or ontologies, often leveraging LLMs for their semantic understanding capabilities (Lobo et al., 2023; Yang et al., 2025; Vandemoortele et al., 2024).

8 Conclusion

Expanding the abbreviated column names for tabular data is critical for many downstream NLP tasks. In this paper we have significantly advanced the state of the art for this problem. First, we showed that synthetic public data used by prior work is not sufficient for experiments, and we introduced four new datasets in enterprise/science domains, with real-world abbreviations. Second, we showed that accuracy measures used by prior work undercount correct expansions, and we proposed new synonym-aware measures that capture accuracy much more accurately. Finally, we developed Columbo, a powerful LLM-based solution, and described extensive experiments, which show that Columbo outperforms prior work by 4-29% accuracy on five datasets.

For future work we will explore improving Columbo, exploiting data tuples where available, using in-house LLMs, and developing solutions to expand abbreviated table names.

Acknowledgments: We thank Suresh Bathini and Guarav Pathak for their assistance with this paper; Paul Hanson, Colin Smith, Corinna Gries, Mark Tervo, and Minh Phan for helping us with EDI; and the anonymous reviewers for their insightful comments. This paper is supported by NSF grant IIS-2504787 and a grant from Google Inc.

9 Limitations

A limitation of our work is that so far we only consider using table names and column names as the input to expand column names. In real-world datasets, tables could contain additional metadata and data tuples may be available that could provide useful context. Furthermore, some datasets include taxonomic information, which could enhance our summarization and local clustering process. We focus on table and column names because they are typically the most essential and consistently available metadata across datasets. Moreover, domain-specific abbreviations are also common and can aid in column name expansion. A potential improvement is to incorporate domain knowledge into the prompting process to enable more accurate expansion of such abbreviations.

References

Waleed Ammar, Kareem Darwish, Ali El Kahki, and Khaled Hafez. 2011. Ice-tea: in-context expansion and translation of english abbreviations. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 41–54. Springer.

Anonymous. 2024. Tabmeta: Table metadata generation with LLM-curated dataset and LLM-judges. In *Submitted to ACL Rolling Review - June 2024*. Under review.

Anonymous. 2025. Realistic training data generation and rule enhanced decoding in LLM for nameguess. In *Submitted to ACL Rolling Review - December 2024*. Under review.

Shanqing Cai, Subhashini Venugopalan, Katrin Tomanek, Ajit Narayanan, Meredith Morris, and Michael Brenner. 2022. Context-aware abbreviation expansion using large language models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1261–1275, Seattle, United States. Association for Computational Linguistics.

Çagatay Demiralp, Fabian Wenz, Peter Baile Chen, Moe Kayali, Nesime Tatbul, and Michael Stonebraker. 2024. Making llms work for enterprise data tasks. *CoRR*, abs/2407.20256.

Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2022. TURL: table understanding through representation learning. *SIGMOD Rec.*, 51(1):33–40.

Xiaokun Du, Rongbo Zhu, Yanhong Li, and Ashiq Anjum. 2019. Language model-based automatic prefix abbreviation expansion method for biomedical big data analysis. *Future Generation Computer Systems*, 98:238–251.

Xi Fang, Weijie Xu, Fiona Anting Tan, Jiani Zhang, Ziqing Hu, Yanjun Qi, Scott Nickleach, Diego Socolinsky, Srinivasan Sengamedu, and Christos Faloutsos. 2024. Large language models (llms) on tabular data: Prediction, generation, and understanding–a survey. *arXiv preprint arXiv:2402.17944*.

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):2279–2292.

Juliana Freire, Grace Fan, Benjamin Feuer, Christos Koutras, Yurong Liu, Eduardo Peña, Aécio S. R. Santos, Cláudio T. Silva, and Eden Wu. 2025. Large language models for data discovery and integration: Challenges and opportunities. *IEEE Data Eng. Bull.*, 49(1):3–31.

Yingqi Gao and Zhiling Luo. 2025. Automatic database description generation for text-to-sql. *arXiv preprint arXiv:2502.20657*.

Heng Gong, Xiaocheng Feng, Bing Qin, and Ting Liu. 2019. Table-to-text generation with effective hierarchical encoder on three dimensions (row, column and time). In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3143–3152, Hong Kong, China. Association for Computational Linguistics.

Heng Gong, Yawei Sun, Xiaocheng Feng, Bing Qin, Wei Bi, Xiaojiang Liu, and Ting Liu. 2020. TableGPT: Few-shot table-to-text generation with table structure reconstruction and content matching. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 1978–1988, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Kyle Gorman, Christo Kirov, Brian Roark, and Richard Sproat. 2021. Structured abbreviation expansion in context. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 995–1005, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Donghee Han, Seungjae Lim, Daeyoung Roh, Sangryul Kim, Sehyun Kim, and Mun Yong Yi. 2025. Leveraging llm-generated schema descriptions for unanswerable question detection in clinical data. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 10594–10601.

Paul Hanson. 2025. Environmental Data Initiative. [link].

Yeye He, Jie Song, Yue Wang, Surajit Chaudhuri, Vishal Anil, Blake Lassiter, Yaron Goland, and Gaurav Malhotra. 2021. Auto-tag: Tagging-data-by-example in data lakes.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333, Online. Association for Computational Linguistics.

Madelon Hulsebos, Paul Groth, and Çagatay Demiralp. 2023. Adatyper: Adaptive semantic column type detection.

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*, pages 1500–1508. ACM.

Zdeněk Kasner, Ekaterina Garanina, Ondrej Platek, and Ondrej Dusek. 2023. TabGenie: A toolkit for table-to-text generation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 444–455, Toronto, Canada. Association for Computational Linguistics.

Elita Lobo, Oktie Hassanzadeh, Nhan Pham, Nandana Mihindukulasooriya, Dharmashankar Subramanian, and Horst Samulowitz. 2023. Matching table metadata with business glossaries using large language models. *arXiv preprint arXiv:2309.11506*.

Kyle Luoma and Arun Kumar. 2025. Snails: Schema naming assessments for improved llm-based sql inference. *Proc. ACM Manag. Data*, 3(1).

Vaishali Pal, Andrew Yates, Evangelos Kanoulas, and Maarten de Rijke. 2023. MultiTabQA: Generating tabular answers for multi-table question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6322–6334, Toronto, Canada. Association for Computational Linguistics.

Amir Pouran Ben Veyseh, Franck Dernoncourt, Quan Hung Tran, and Thien Huu Nguyen. 2020. What does this acronym mean? introducing a new dataset for acronym identification and disambiguation. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 3285–3301, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Brian Roark and Richard Sproat. 2014. Hippocratic abbreviation expansion. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 364–369.

Pradnya Sawant and Kavita Sonawane. 2024. Nlp-based smart decision making for business and academics. *Natural Language Processing Journal*, 8:100090.

Mayank Singh, Abhijeet Kumar, Sasidhar Donaparthi, and Gayatri Karambelkar. 2025. Leveraging retrieval augmented generative llms for automated metadata description generation to enhance data catalogs. *arXiv preprint arXiv:2503.09003*.

Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çagatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2021. Annotating columns with pre-trained language models. *CoRR*, abs/2104.01785.

Nathan Vandemoortele, Bram Steenwinckel, SV Hoecke, and Femke Ongenae. 2024. Scalable table-to-knowledge graph matching from metadata using llms.

Niklas Wretblad, Oskar Holmström, Erik Larsson, Axel Wiksäter, Oscar Söderlund, Hjalmar Öhman, Ture Pontén, Martin Forsberg, Martin Sörme, and Fredrik Heintz. 2024. Synthetic sql column descriptions and their impact on text-to-sql performance. *arXiv preprint arXiv:2408.04691*.

Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I. Wang, Victor Zhong, Bailin Wang, Chengzu Li, Connor Boyle, Ansong Ni, Ziyu Yao, Dragomir Radev, Caiming Xiong, Lingpeng Kong, and 4 others. 2022. UnifiedSKG: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 602–631, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Duo Yang, Ioannis Dasoulas, and Anastasia Dimou. 2025. Matching table metadata to knowledge graphs: A data augmentation perspective. In *ELLIS workshop on Representation Learning and Generative Models for Structured Data*.

Jingfeng Yang, Aditya Gupta, Shyam Upadhyay, Luheng He, Rahul Goel, and Shachi Paul. 2022. TableFormer: Robust transformer modeling for table-text encoding. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 528–537, Dublin, Ireland. Association for Computational Linguistics.

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):1835–1848.

Haoxiang Zhang, Yurong Liu, Aécio Santos, Juliana Freire, and 1 others. 2025. Autoddg: Automated dataset description generation using large language models. *arXiv preprint arXiv:2502.01050*.

Jiani Zhang, Zhengyuan Shen, Balasubramaniam Srinivasan, Shen Wang, Huzefa Rangwala, and George Karypis. 2023. NameGuess: Column name expansion for tabular data. In *Proceedings of the 2023*

Conference on Empirical Methods in Natural Language Processing, pages 13276–13290, Singapore. Association for Computational Linguistics.

Yilun Zhao, Zhenting Qi, Linyong Nan, Boyu Mi, Yixin Liu, Weijin Zou, SIMENG HAN, RUIZHE CHEN, Xiangru Tang, Yumo Xu, Dragomir Radev, and Arman Cohan. 2023a. QTSumm: Query-focused summarization over tabular data. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Yilun Zhao, Haowei Zhang, Shengyun Si, Linyong Nan, Xiangru Tang, and Arman Cohan. 2023b. Investigating table-to-text generation capabilities of large language models in real-world information seeking scenarios. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 160–175.

A Appendix

A.1 Sample Column Names & Gold Expansions

Figures 5-6 show sample column names and gold expansions for the datasets AW and EDI, respectively.

A.2 Creating the Datasets

We now describe how the datasets for our experiments were created.

NameGuess (NG): This dataset is provided by (Zhang et al., 2023). It comes from the tables provided by government data portals (San Francisco, Los Angeles, and Chicago). Upon using the original gold expansions provided by the authors, we manually curated some of the gold expansions where the original gold expansion did not match the abbreviated column names. For example, if the original abbreviated column is 'b_stop', but the gold expansion is 'bus stop only', we will remove 'only' and use 'bus stop' as the gold expansion.

Finance & University: They are two proprietary customer datasets from companies in the finance and university domains, they natively include table names, abbreviated column names, and detailed column descriptions, which we used to manually derive the gold expansions.

Environmental Data Initiative (EDI): This dataset comes from the (Hanson, 2025) platform, a repository rich in ecological data, containing over 18,000 tables from 27 US ecological sites. We randomly sampled 251 tables from the EDI platform, and used the extensive provided metadata (project details, table/column descriptions) to manually create gold expansions for the originally abbreviated column names.

Adventure Works (AW): This dataset is based on the Adventure Works sample database, representative of common enterprise schemas. Its original schema contains full-form names. But in a separate project the participants have manually abbreviated the column names using the table context. We manually reviewed these human-generated abbreviations and verified or corrected gold expansions.

Across all datasets, there were instances where determining a definitive gold expansion with high confidence was impossible due to ambiguity or insufficient metadata. For example, in the EDI dataset, there are column names 'x' and 'y' and no definition is provided. In such cases we cannot determine the gold expansion for these columns. As another example, in the EDI dataset, in the 'd-1cr23x-cr1000.daily.ml.data' table, there is a column named 'airtemp_hmp1_max', however there is no definition of the token 'hmp' and we cannot find other information about the token 'hmp' in other entries of the project metadata. Moreover, LLM may generate various possible expansions for this token and we cannot determine which one is correct as we are not domain experts.

Such columns were excluded from the calculation of evaluation metrics. However, they were retained as part of the input schemas provided to the models, as they still contribute valuable contextual information about the table's structure and domain.

A.3 Sample LLM Prompt and Output for the Summarizer

Figure 7 shows a sample prompt to the LLM to ask it to cluster 10 given tables. Figure 8 shows a sample output from GPT-4o for the above prompt.

A.4 Sample LLM Prompt and Output for the Generator

Figure 9 shows a sample prompt to the LLM to ask it to expand 4 column names from a table. Similar prompts are used by the Generator (but asking the LLM to expand 10 column names). Figure 10 shows a sample output from the LLM for the above prompt.

Column Name	Gold Expansion
ACity	address city
AID	address identifier
AL1	address line 1
AL2	address line 2
ARGuid	address row globally unique identifier
CDate	changed date
SID	state identifier
ZipCode	zip code

(a) Table name: Address

Column Name	Gold Expansion
Date	date
FactCCAutRSPs	fact call center automatic responses
FactCCAvgTimePerIssue	fact call center average time per issue
FactCCCalls	fact call center calls
FactCCDateKey	fact call center date key
FactCCID	fact call center identifier
FactCCIssuesRaised	fact call center issues raised
FactCCLvlOneOps	fact call center level one operators
FactCCLvlTwoOps	fact call center level two operators
FactCCOrders	fact call center orders
FactCCShift	fact call center shift
FactCCTotOps	fact call center total operators
FactCCWageType	fact call center wage type
SGrade	service grade

(b) Table name: FactCallCenter

Column Name	Gold Expansion
EmpId	employee identifier
ODate	order date
POHFrght	purchase order header freight
POHMdate	purchase order header modified date
POHShipDate	purchase order header ship date
POHStatus	purchase order header status
POHSubTtl	purchase order header sub total
POHTaxAmt	purchase order header tax amount
PurchOrdrID	purchase order identifier
RvNumber	revision number
ShipId	ship identifier
TDue	total due
VID	vendor identifier

(c) Table name: PurchaseOrderHeader

Figure 5: Sample column names and gold expansions for the AW dataset.

Column Name	Gold Expansion
LTER_site	Long-Term Ecological Research Site
Local_site	Local Site
Jday	Julian day
Year	Year
Airtemp_max	Air temperature maximum
Rh_max	Relative humidity maximum
Bp_min	Barometric pressure minimum
Flag_bp_min	Flag barometric pressure minimum
Flag_ws_max	Flag wind speed maximum
Wd	Wind direction
Solrad_avg	Solar radiation average
Soiltemp_5cm_avg	Soil temperature 5 centimeters average

(a) Table name: d-1cr23x-cr1000.daily.ml.data

Column Name	Gold Expansion
Region	region
SITE_ID	Site identifier
TOTBIO	Total biovolume
BIOV	biovolume
SOLTYP	Soil type
TP	Total phosphorus
DRY	dry
AFDM	Ash free dry mass
ORG	organic
CHLA	Chlorophyll A
CHLCON	Chlorophyll A Concentration

(b) Table name: ST_PP_Lahee_002

Column Name	Gold Expansion
Release_ID	Release identifier
Water_temp_C	Water temperature celsius
STRBAS	Striped bass
AMESHA	American shad
CHACAT	Channel catfish
CHISAL	Chinook salmon
SACPIK	Sacramento pikeninnow
SACSUC	Sacramento sucker
BLACRA	Black crappie
COMCAR	Common carp
BLUGIL	bluegill
WHICAT	White catfish
SMABAS	Smallmouth bass

(c) Table name: EDI_ASB_SampleDay_122723

Figure 6: Sample column names and gold expansions for the EDI dataset.

A.5 Sample LLM Prompt and Output for the Reviser

Figure 11 shows a sample prompt to the LLM asking if a given token has a unique expansion. Similar prompts are used by the Reviser. Figure 12 shows a sample output by GPT-4o that determines that the given token does not have a unique expansion.

A.6 Ablation Studies

Figure 13 shows the full result of the ablation studies, as we examine the effect of five major components of Columbo.

A.7 Sensitivity Analysis

Figure 14 shows the full result of the sensitivity analysis with respect to the ordering of tables for the Summarizer.

```
//Below are 10 table schemas that we ask LLM to cluster
Given the following abbreviated table schemas from the same dataset:
AWBuildVersion ['DbVersion', 'ModDate', 'SysInformationID', 'VerDate’]
Address ['ACity', 'AID', 'AL1', 'AL2', 'ARGuid', 'CDate', 'SID', 'ZipCode’]
AddressType ['ATChangeDate', 'ATRGuid', 'ATypeID', 'Type’]
AdventureWorksDWBuildVersion ['DBVers', 'VerDate’]
BillOfMaterials ['Bid', 'BomLvl', 'ModDate', 'PAssemblyQt', 'PassemblyId', 'cId', 'eDate',
'sDate', 'uMCode’]
BusinessEntity ['ModDate', 'bId', 'rId’]
BusinessEntityAddress ['ModDate', 'aId', 'aTypeId', 'bId', 'rId’]
BusinessEntityContact ['ModDate', 'PersonId', 'bId', 'cTypeId', 'rId’]
ContactType ['CTypeID', 'ContactName', 'ContactTypeMDate’]
CountryRegion ['CRCode', 'CRMdate', 'CRName’]

Your job is to:

1. Determine the overall context and domain of the dataset based on column names and
structures.
2. Explain how you identified the context and grouped the tables together. Group them into
categories based on their function or thematic role.
3. For each table, write a clear and concise topic that summarizes the purpose of the table
(not a direct name expansion). a. These topics should help explain what kind of data is in each
table and how it's used.
4. At the end, output the grouped table topics, a JSON object where:
        i. Each key is a group topic
        ii. Each value is a dictionary where keys are table names and values are table topics.
```

Figure 7: A sample prompt to the LLM to ask it to cluster 10 tables from the AW dataset. Similar prompts are used by the Summarizer (but sending 30 tables to the LLM).

```
Based on the structure and naming of the columns and tables, here's the
  analysis:
1. Overall Context and Domain
The dataset belongs to the AdventureWorks Data Warehouse, a well-known
  sample database from Microsoft. The domain is Enterprise Resource
  Planning (ERP) and Customer Relationship Management (CRM) for a
  manufacturing or retail company. It contains data on addresses,
  business entities, products, materials, contacts, and metadata
  related to system builds.
2. How Context and Groupings Were Identified
• Naming conventions such as "AW", "DW", "Entity", "Contact",
  "BuildVersion" indicate metadata and system tracking.
• Entity relationships are inferred from shared ID fields (e.g., bId,
  aId, aTypeId), showing how tables relate via business entities or
  address IDs.
• Table roles (e.g., master data vs. transactional or reference data)
  are inferred by the type of columns they have.
• Suffixed IDs (Id, Code, TypeId) denote keys and foreign keys used for
  linking.
• Date fields hint at versioning, modification, or validity of records.
//Below are 5 groups output by the LLM
3. Table Topics
System Metadata
• AWBuildVersion: Tracks the version and modification date of the
  AdventureWorks database build.
• AdventureWorksDWBuildVersion: Contains version information for the
  AdventureWorks Data Warehouse environment.
Geographic and Address Data
• Address: Stores full physical address details used for customers,
  employees, and business locations.
• AddressType: Defines the type of an address (e.g., billing, shipping)
  with metadata and change tracking.
• CountryRegion: Provides a list of countries or regions used for
  address normalization and geographic grouping.
Business Entity Structure
• BusinessEntity: Represents a generic business entity that can be a
  person, vendor, or customer.
• BusinessEntityAddress: Links business entities to physical addresses
  with a specific address type.
• BusinessEntityContact: Connects business entities to contacts and
  contact types (like roles or responsibilities).
Reference and Lookup
• ContactType: Lists types of contacts (e.g., decision-maker, buyer)
  for categorizing contact records.
Product and Manufacturing
• BillOfMaterials: Defines component relationships in manufacturing,
  showing how parts are assembled into products.
//Below is the same output but in JSON format
4. Grouped Table Topics (JSON Object)
{ "System Metadata":
       { "AWBuildVersion": "Tracks the version and modification date of
                            the AdventureWorks database build.",
       "AdventureWorksDWBuildVersion": "Contains version information
                                        for the AdventureWorks Data
                                        Warehouse environment." },
//3 more groups omitted for readability
  "Product and Manufacturing":
       { "BillOfMaterials": "Defines component relationships in
                             manufacturing, showing how parts are
                             assembled into products." }
}
```

Figure 8: A sample output by GTP-4o, in response to the prompt in Figure 7. Note how the LLM groups the 10 input tables into 5 groups, with 2, 3, 3, 1, 1 tables in each group, respectively. The first group has the summary "System Metadata" and has 2 tables. The second group has the summary "Geographic and Address Data" and has 3 tables, and so on. Note also that the LLM produces for each table a short summary.

```
//Below are the names and summaries of some related tables
Given a list of table schemas in the same dataset:
[['AWBuildVersion', 'Metadata tracking for build versions of the source
  system'], ['AdventureWorksDWBuildVersion', 'Data warehouse versioning
  and deployment date tracking'], ['DatabaseLog', 'Historical log of
  database-level operations for auditing and debugging']]

Your task is to expand abbreviated column names into full-form English
  phrases.
You should return the tokens of each column name and their associated
  expansions.
First reason step by step and then return your final answer.
//Below are the rules we ask the LLM to follow
Follow the guidelines below when you expand:
1. Expand all abbreviations in the column names.
2. Expand chemical symbols and units of measure to their full names.
3. Do not expand or mutate numbers.
4. Do not add extra words or explanations.
5. Maintain the original order of tokens in the expansion.
6. If the column name is already in full-form English words, return the
  column name.
7. The tokens should be as concise and simple as possible.
8. Only provide 1 expansion for each token, even if you are uncertain,
  output the most possible one.
Do not put ambiguous or need context after your expansion.
9. If the token is not abbreviated, the expansion should be itself, do
  not paraphrase or add other words around it.
//Below are examples of CoT reasoning for the LLM
[Question]
As abbreviations of column names from a table named Prchs_info, c_name
  | pCd | dt stand for? Think step by step
[Answer]
### Reasoning
1. **Identify the table name context:**
   - The table is named **"Prchs_info"**, which likely stands for
  **"Purchase Information"**.
   - This suggests that the column names are related to purchase
  details.

2. **Break down each column name into tokens and infer meanings:**
   - **c_name** → `c`, `name`
     - `c` is commonly used for **Customer** in business contexts.
     - `name` clearly refers to **Name**.
     - Together, `c_name` most likely means **Customer Name**.

   - **pCd** → `p`, `Cd`
     - `p` is frequently used for **Product** in sales or inventory
  tables.
     - `Cd` is a common abbreviation for **Code**.
     - Together, `pCd` most likely means **Product Code**.

   - **dt** → `dt`
     - `dt` is a standard abbreviation for **Date**.
     - In the context of a purchase table, it likely refers to
  **Purchase Date**.

### Final Answer
c_name: c → Customer, name → Name
pCd: p → Product, Cd → Code
dt: dt → Date
//3 more examples omitted for readability.
//Below are the columns for which we ask LLM for expansion.
//Note that we are also supplying the table name
[Question]
As abbreviations of column names from a table named AWBuildVersion,
  SysInformationID | DbVersion | VerDate | ModDate stand for
[Answer]
```

Figure 9: A sample prompt to the LLM to ask it to expand 4 column names of a table. Similar prompts are used by the Generator, but asking to expand up to 10 column names. Note how we provide the names and summaries of some related tables, the rules, examples of CoT reasoning. Then we finally provide the 4 abbreviated column names (and the table name) in the last three lines of the prompt, and ask the LLM to expand those column names.

```
### Reasoning
1. **Identify the table name context:**
   - The table is named **"AWBuildVersion"**, which is related to **"Metadata
  tracking for build versions of the source system"**.
   - This suggests that the column names are related to system and version
  information.

2. **Break down each column name into tokens and infer meanings:**
   - **SysInformationID** → `Sys`, `Information`, `ID`
     - `Sys` stands for **System**.
     - `Information` is already in full form.
     - `ID` stands for **Identifier**.
     - Together, `SysInformationID` means **System Information Identifier**.

   - **DbVersion** → `Db`, `Version`
     - `Db` stands for **Database**.
     - `Version` is already in full form.
     - Together, `DbVersion` means **Database Version**.

   - **VerDate** → `Ver`, `Date`
     - `Ver` stands for **Version**.
     - `Date` is already in full form.
     - Together, `VerDate` means **Version Date**.

   - **ModDate** → `Mod`, `Date`
     - `Mod` stands for **Modified**.
     - `Date` is already in full form.
     - Together, `ModDate` means **Modified Date**.

### Final Answer
SysInformationID: Sys → System, Information → Information, ID → Identifier
DbVersion: Db → Database, Version → Version
VerDate: Ver → Version, Date → Date
ModDate: Mod → Modified, Date → Date
```

Figure 10: A sample output from GPT-4o that provides the E2 records for the 4 column names mentioned in the prompt of the previous figure.

```
//Below is the summaries of all groups of tables
The tables from the dataset are clustered into different groups. The
  topics of all groups are:
['Vegetation & Biomass Monitoring',
 'Seed, Cone & Reproduction Studies',
 […omitted for readability…]
 'Biogeochemical Fluxes']

Now, there are ambiguous tokens from each column name. They are
  ambiguous because LLM expands them into different expansions. For
  example:
//Below we ask if this token has an unique expansion
ambiguous token: cond
//Below are the current expansion rules and their frequencies
possible expansion and count: ['condition', 10]
//Below are one sample table schema for each expansion rule
here is a sample table schema having this expansion:
  mp_tallgrass_25years.csv ['animal_id', 'sex', 'reprod_cond',
  'weight', 'recapture', 'trapping_session']
possible expansion and count: ['conductance', 12]
here is a sample table schema having this expansion:
  MEL_Toolik_Tussock_Pfertilized.csv ['canopy fraction', 'VPD', 'Avail.
  Water fraction', 'P bal Apatite weathering', 'P bal PO4
  precipitation', 'P bal Non-occl weathrng', 'P bal occld weathering',
  'P bal non-occld stablz', 'soil potential', 'canopy cond max']



Based on the context of the whole dataset, does this token appear to be
  a commonly used special token in this dataset or just a random
  abbreviation whose meaning may vary in different tables? If it is a
  common token, what is the most possible expansion of this token?
  Understand the whole dataset first and reason. Justify your answer.
  At the end, after '### Final Answer': output your answer for this
  ambiguous token in the format of token: answer, the answer is either
  the most possible expansion of this token or 'vary based on context".
```

Figure 11: A sample prompt asking the LLM if the token "cond" has a unique expansion.

To determine whether the ambiguous token **"cond"** is a commonly used special token with a consistent meaning or a context-dependent abbreviation, we must consider the context of the dataset as a whole, the nature of the table topics, and the examples provided.

Step 1: Understand the Dataset Scope

The dataset spans a broad range of environmental and ecological monitoring themes, including:

- Vegetation and biomass studies
- Soil, water, and atmospheric chemistry
- Climate and weather data
- Biodiversity monitoring (plants, animals, microbes)
- Aquatic systems and hydrology
- Animal and faunal trapping
- Remote sensing and physiological traits

This strongly suggests a **multi-disciplinary environmental science dataset** involving both **biotic and abiotic** factors, field measurements, and sensor data.

Step 2: Analyze Token Usage

We are given two sample expansions for the token cond:

1.'condition' (10 counts)
Example schema:
['animal_id', 'sex', 'reprod_cond', 'weight', 'recapture', 'trapping_session']
Interpretation:
In this context, reprod_cond likely refers to **reproductive condition** — a biological trait in animal studies.

2.'conductance' (12 counts)
Example schema:
['canopy fraction', 'VPD', 'Avail. Water fraction', ..., 'canopy cond max']
Interpretation:
Here, canopy cond max likely refers to **canopy conductance maximum,** a physiological or hydrological parameter related to water or gas flux.

Step 3: Dataset Context Alignment

We now evaluate how these two expansions align with the dataset themes:

- **'Conductance'** is a key measurement in plant physiology, soil hydrology, and gas flux studies.
  It fits well under:
  - *Remote Sensing & Photosynthesis Modeling*
  - *Ecophysiological Plant Traits*
  - *Gas, Hydrology & Water Chemistry*
  - *Atmospheric & Climate Flux*
  - *Soil Temperature & Moisture Monitoring*
- **'Condition'** (especially reproductive condition or health condition) is important in faunal monitoring and ecological health assessments.
  It aligns with:
  - *Wildlife Monitoring (Rodent Trapping)*
  - *Fish Monitoring & Trapping*
  - *Faunal Monitoring and Tagging*
  - *Nesting and Reproduction in Birds*

Thus, **both interpretations are valid in their respective subdomains,** and neither dominates in importance across all dataset topics. Furthermore, the counts are close (10 vs. 12), suggesting no strong bias in favor of one meaning.

Step 4: Semantic Ambiguity

Given the diversity of domains in the dataset and the fact that **the same token expands to different meanings based on context,** this suggests **"cond" is not a consistent, domain-specific shorthand,** but rather a **context-sensitive abbreviation.**

Final Answer:

- **token: vary based on context**

Figure 12: A sample output from GPT-4o stating that token "cond" does not have a unique expansion rule (see the last line of the sample output.

Datasets	Columbo			Columbo - Context		
	EM	F_1	Bert-F_1	EM	F_1	Bert-F_1
NG	**85.2**	91.9	94.5	83.7	90.5	93.1
Finance	87.6	96.9	95.6	**88.5**	96.9	95.4
University	**92.2**	98.2	97.4	76.6	91.9	88.0
AW	**93.3**	97.6	98.2	90.4	95.3	96.0
EDI	**90.7**	95.3	93.6	89.8	94.6	92.4

(a) The ablation study where Columbo does not exploit contexts.

Datasets	Columbo			Columbo – Table Name		
	EM	F_1	Bert-F_1	EM	F_1	Bert-F_1
NG	85.2	91.9	94.5	**85.6**	91.7	94.4
Finance	**87.6**	96.9	95.6	83.1	96.0	94.4
University	**92.2**	98.2	97.4	83.5	95.4	93.3
AW	**93.3**	97.6	98.2	83.1	93.5	95.7
EDI	**90.7**	95.3	93.6	89.4	94.8	92.9

(b) The ablation study where Columbo does not exploit table names.

Datasets	Columbo			Columbo - Rules		
	EM	F_1	Bert-F_1	EM	F_1	Bert-F_1
NG	**85.2**	91.9	94.5	84.2	91.1	92.5
Finance	**87.6**	96.9	95.6	85.3	95.9	94.5
University	**92.2**	98.2	97.4	82.7	96.4	94.6
AW	**93.3**	97.6	98.2	85.4	95.2	94.0
EDI	**90.7**	95.3	93.6	63.6	80.8	76.8

(c) The ablation study where Columbo does not exploit rules.

Datasets	Columbo			Columbo - CoT		
	EM	F_1	Bert-F_1	EM	F_1	Bert-F_1
NG	**85.2**	91.9	94.5	85.0	91.1	93.6
Finance	87.6	96.9	95.6	**88.0**	96.5	95.0
University	**92.2**	98.2	97.4	89.2	97.0	96.0
AW	**93.3**	97.6	98.2	91.0	96.5	96.5
EDI	**90.7**	95.3	93.6	90.4	95.1	93.6

(d) The ablation study where Columbo does not exploit CoT reasoning.

Datasets	Columbo			Columbo – Token Analysis		
	EM	F_1	Bert-F_1	EM	F_1	Bert-F_1
NG	85.2	91.9	94.5	**86.9**	92.1	94.5
Finance	**87.6**	96.9	95.6	87.1	96.8	95.5
University	**92.2**	98.2	97.4	80.7	94.8	92.2
AW	93.3	97.6	98.2	**93.5**	97.6	98.2
EDI	**90.7**	95.3	93.6	90.3	95.0	93.6

(e) The ablation study where Columbo does not exploit token-level analysis.

Figure 13: The Ablation studies of Columbo

Datasets	Columbo			Columbo Shuffled 1			Columbo Shuffled 2		
	EM	F_1	BERTScore-F_1	EM	F_1	BERTScore-F_1	EM	F_1	BERTScore-F_1
NG	85.16	91.88	94.50	**85.60**	91.82	94.30	85.56	91.10	94.08
Finance	87.58	96.92	95.55	**88.71**	97.28	95.82	88.04	97.09	95.96
University	**92.18**	98.16	97.36	88.60	96.64	95.59	88.60	96.95	95.86
EDI	**90.69**	95.27	93.61	88.97	94.35	92.66	89.54	94.75	93.25
AW	**93.33**	97.56	98.20	91.26	96.89	97.94	92.96	97.33	98.09

Figure 14: The accuracy of Columbo as we try three different orderings of the tables, for the Summarizer.