NameGuess: Column Name Expansion for Tabular Data

Jiani Zhang, Zhengyuan Shen, Balasubramaniam Srinivasan, Shen Wang, Huzefa Rangwala, George Karypis
Amazon Web Services
{zhajiani, donshen, srbalasu, shenwa, rhuzefer, gkarypis}@amazon.com

Abstract

Recent advances in large language models have revolutionized many sectors, including the database industry. One common challenge when dealing with large volumes of tabular data is the pervasive use of abbreviated column names, which can negatively impact performance on various data search, access, and understanding tasks. To address this issue, we introduce a new task, called NAMEGUESS, to expand column names (used in database schema) as a natural language generation problem. We create a training dataset of 384K abbreviated-expanded column pairs using a new data fabrication method and a human-annotated evaluation benchmark that includes 9.2K examples from real-world tables. To tackle the complexities associated with polysemy and ambiguity in NAMEGUESS, we enhance autoregressive language models by conditioning on table content and column header names – yielding a fine-tuned model (with 2.7B parameters) that matches human performance. Furthermore, we conduct a comprehensive analysis (on multiple LLMs) to validate the effectiveness of table content in NAMEGUESS and identify promising future opportunities. Code has been made available at https://github.com/amazon-science/nameguess.

1 Introduction

Tabular data is widely used for storing and organizing information in web (Zhang and Balog, 2020) and enterprise applications (Leonard, 2011). One common practice when creating tables in databases is to use abbreviations for column headers due to character length limits in many standard database systems. For example, the maximum length for column names in an SQL database is 256 bytes, making it inconvenient for representation and use in code, they can cause confusion, especially for those unfamiliar with the particular tables or subject matter. Column headers are essential for many table-related tasks (Xie et al., 2022), and using abbreviations makes it challenging for end users to search and retrieve relevant data for their tasks.

Abbreviated column names can negatively impact the usefulness of the underlying data. For example, in the text2SQL semantic parsing task, which converts natural language into formal programs or queries for retrieval, abbreviations can lead to a mismatch with the terms used in the natural language queries. In fact, in the human-labeled text2SQL spider dataset (Yu et al., 2018), 6.6% of column names are abbreviations. Figure 2 shows an example containing abbreviated column names like "c_name" and "acc_bal" in tables, which mismatch the terms “the name of all customers” and “account balance”. Simple changes in using abbreviated column names in the spider dataset result in a performance degradation of over ten percentage points, with the exact match score of 66.63% (Xie et al., 2022) dropping to 56.09% on the T5-large model (Raffel et al., 2020). The effect of the abbreviated column names on table question answering (QA) (Yin et al., 2020), and column relation dis-
The terms “the name of all customers” and “account balance” do not match the abbreviated column names c_name and acc_bal. Instead, they match with the column names customer_ID and account_type.

This work addresses the task of expanding abbreviated column names in tabular data. To the best of our knowledge, this is the first work to introduce and tackle this problem. Unlike previous textual abbreviation expansion works that formulated the task as a classification problem with a predefined set of candidate expansions (Roark and Sproat, 2014; Gorman et al., 2021), we formulate NAMEGUESS as a natural language generation problem. Acquiring extensive candidate expansions can be laborious, as pairs of abbreviated and expanded column names are seldom present in the same table. Conversely, abbreviation-expansion pairs can be gleaned from textual data through co-occurrence signals, such as parenthetical expressions. Moreover, abbreviated headers may exhibit ambiguity and polysemy arising from developer-specific naming conventions and domain-related variations in expansions.

To tackle NAMEGUESS, we first built a large dataset consisting of 163,474 tables with 384,333 column pairs and a human-annotated benchmark with 9,218 column pairs on 895 tables. We then proposed a method to produce training data by selectively abbreviating well-curated column names from web tables using abbreviation look-ups and probabilistic rules. Next, we enhanced autoregressive language models with supervised fine-tuning, conditioned on table content and column headers, and conducted extensive experiments to evaluate state-of-the-art LLMs. The overall model performance is shown in Figure 3. While GPT-4 exhibited promising performance on NAMEGUESS, the deployment of such LLMs comes with much larger memory and computation overheads. Our findings indicate that supervised fine-tuning of smaller 2.7B parameter models achieves close to human performance, and including table contents consistently improved performance. However, all models found the task challenging, with the best only achieving 54.7% accuracy on the extra-hard examples, indicating room for improvement in expanding abbreviated column names. Our main contributions are:

1. Introduced a new column name expansion task, named NAMEGUESS, as a natural language generation problem.
2. Developed a large-scale training dataset for the NAMEGUESS task using an automatic method that largely reduces human effort.
3. Created a human-annotated evaluation benchmark with various difficulty levels, which provides a standard for comparing results.
4. Performed a comprehensive evaluation of LMs of different sizes and training strategies and compared them to human performance on the NAMEGUESS task.
Table 1: The effect of abbreviated column names on three table understanding tasks. The performance drops on all the tasks.

### 2 Problem Formulation

We formulate the NAMEGUESS task as a natural language generation problem: given a query name $x$ from table $t$, generate a logical name $y$ that describes the column. A table contains table content and various schema data, like a table name, column headers, and data types. Let $f_\theta$ be a generator with parameters $\theta$, the formulation becomes $y = f_\theta(x | t)$. Note that the query column names within a table may take different forms, including abbreviations and full names. The output logical names are expanded column names and should be easily understandable without additional knowledge of the table content. See Figure 1 for example inputs and outputs in the "Employee_Salary_2022" table, where "SAL" stands for “Salary” and "COMM" stands for “Commission”.

### 3 Dataset Creation

We created a training dataset comprising 384,333 columns spanning 163,474 tables and a human-labeled evaluation benchmark containing 9,218 examples across 895 tables for NAMEGUESS. The main challenge in obtaining abbreviated-expanded column name pairs is the rare co-occurrence of these two names within the same table or database. Therefore, we employed the strategies of converting well-curated column names to abbreviated names that align with the naming convention of database developers and annotating the abbreviated column names based on the information in the input table. Figure 4 illustrates the main steps for creating the training and evaluation datasets. The details of the source tables are discussed in Section 3.1. The training and evaluation datasets are constructed in Section 3.2 and Section 3.3.
3.2.1 Logical Name Identification

Identifying high-quality column names from relational tables is essential, as further abbreviating a vague term can lead to even more ambiguity. Algorithm 2 in Appendix A.2 shows the detailed algorithm with a vocabulary-based strategy. We regarded a column name as well-curated only if all the tokens in the column name can be found from a pre-defined vocabulary. Here, we used the WordNinja package to split the original column headers, which allowed us to check whether individual tokens are included in the vocabulary.

To construct this pre-defined vocabulary, we used WordNet open-class English words (Fellbaum, 1998) followed by a set of filtering criteria (such as removing words with digits, punctuation, and short words that are abbreviations or acronyms) so that the classifier achieves high precision for detecting the logical names, which further used as ground truth labels.

3.2.2 Abbreviation Generation

After obtaining well-curated names, we used an abbreviation generator to produce the abbreviated names. Table 3 summarizes four abbreviation schemes from logical to abbreviated names. We adopted word-level abbreviation and acronym extraction methods and limited word removal and word order change cases, because specifying rules for word removal or order change is an open-ended and difficult-to-scale problem. Our abbreviated name generator employs a probabilistic approach to determine the specific method used for word-level abbreviation. The method is chosen from the following three options, with the selection probability determined by pre-defined weights:

Method 1 (keep): Left as-is. This is trivial but very common, especially when the column header contains fewer words or words that cannot be further shortened without creating ambiguity.

Method 2 (lookup): Replaced with an abbreviation from an expansion-abbreviation look-up table. This is mainly for producing commonly-used abbreviations with enhanced diversity in naming style that is hard to obtain using the above four reformatting methods. Examples include abbreviations based on pronunciations (e.g., transaction → txn and end-to-end → end2end), and symbolic conversions (e.g., second → 2nd, number → no./#, and at → @). The look-up dictionary contains common abbreviations for 23,110 English words or terms. In cases where multiple candidate abbreviations are available, the abbreviation is chosen randomly.

Method 3 (rule): Processed by one of the word-level abbreviation rules:

Rule 1: Keep the first \(k\) characters, \(k \in [1, 5]\) (e.g. abbreviation \(k=4\) → abbr);

Rule 2: Keep removing non-leading vowels until the threshold \(k \in [1, 5]\) or all non-leading vowels have been removed (e.g. abbreviation \(k=5\) → abbrvtn, doodle \(k=5\) → doodl);

Rule 3: Specifically, while the length of the input string is longer than a specified threshold value \(k \in [1, 5]\), the following steps are applied: 1) neighboring duplicate characters are removed, 2) vowels are removed randomly until no vowel remain, and 3) consonants are removed randomly until no consonant remains. (e.g. abbreviation \(k=4\) → abrv). This is to emulate the real-world data and varying preferences of database developers.

Once a rule is selected from the above choices, it is applied to all words within the same column header. It is important to note that these rules do not apply to non-alphabetical words. In the case of numerical words representing a four-digit year, we shorten it to a two-digit representation with a 50%
Algorithm 1 Abbreviation generation

**Inputs:**
A lookup dictionary \( D \) and a string \( x \)

**Initialize:**

- \( \text{Method}_X \leftarrow \text{select}\_\text{method}() \)
- \( \text{Rule}_X \leftarrow \text{select}\_\text{rule}() \)
- \( \text{abbr}\_\text{words} \leftarrow \emptyset \)
- \( x \leftarrow \text{tokenize}(x) \) ▷ Input: well-curated

for \( x_i \) in \( x \) do
  
  if Method\_X is keep then:
    \( \tilde{x}_i \leftarrow x_i \)
  
  if Method\_X is lookup then
    if \( x_i \in D \) then
      \( \tilde{x}_i \leftarrow D[x_i] \)
    else
      \( \tilde{x}_i \leftarrow \text{Rule}_X(x_i) \)
  
  if Method\_X is rule then
    \( \tilde{x}_i \leftarrow \text{Rule}_X(x_i) \)
  
  \( \text{abbr}\_\text{words} \leftarrow \tilde{x}_i \)

\( \tilde{x} \leftarrow \text{combine}(\text{abbr}\_\text{words}) \) ▷ Output: abbreviated

**Hybrid Method.** To simulate the naming convention for database tables, a probability of 0.5 is assigned for converting patterns subject to common acronyms in the well-curated column headers. The entire or a part of the well-curated form could be replaced by an acronym, such as Employee Date of Birth \( \rightarrow \text{EMP\_DOB} \). Furthermore, additional rules are introduced to selectively remove or switch the order of words without altering the semantics of the column name. For example, Event Name \( \rightarrow \text{Evnt} \) and Mailing Address District 2013 \( \rightarrow 2013\text{MailAddrDist} \). Note that when the same word(s) appears in different columns of the same table, we use the same abbreviation form.

**Abbreviation Combination.** As the last step of the abbreviation algorithm, the \( \text{combine()} \) function assigns an equal probability of concatenating the resulting abbreviated words into camel case, Pascal case, snake case, or simple combination which further adds diversity in the naming style.

**Overall Algorithm.** The overall algorithm is in Algorithm 1 with the probability of Method 1 (keep), 2 (lookup) and 3 (rule) set to 0.3, 0.6 and 0.1, respectively. The probabilities for Rule 1, 2, and 3 in rule are set to 0.2, 0.4, and 0.4, respectively. These assignments are undertaken to maintain statistics that resemble real-world datasets.

3.3 Evaluation Benchmark Annotation

Instead of splitting a subset of training examples for evaluation, we created a human-annotated evaluation dataset.

3.3.1 Human Annotations

The evaluation dataset was confirmed by 15 human annotators using the City Open Data from Los Angeles, San Francisco, and Chicago. Detailed instructions were provided to the annotators to ensure consistency in annotations. The instructions are outlined as follows:

1. Read the table metadata, including table category, name, and description.
2. Read and understand the original column name and sampled cell values for that column.
3. Determine if the original column name is abbreviated or well-curated. If in a well-curated form, provide only an “abbreviated variant”. Otherwise, provide a “well-curated variant” and an “abbreviated variant”.
4. When creating abbreviated names, please combine abbreviated words as suggested by the combining rules detailed in Table 3.

A pilot study was first implemented to produce a small number of examples, followed by a quality audit to ensure the guidelines were well-understood. Note that the column names that were found to be unclear or difficult to interpret even with the provided metadata and column cell values were discarded from the dataset. Finally, the annotations underwent another audit process from a separate group of annotators. We employed a criterion where if two out of three annotators agreed, the annotation was considered to pass this agreement measure. Overall, the annotations achieved an agreement rate of 96.5%.
3.3.2 Difficulty Breakdown

We divide the data samples into four difficulty levels to gain deeper insight into the model performance. This classification is based on the edit distance between characters in abbreviated names and ground truth labels. Both names will first be tokenized and have underscores replaced with spaces, and numbers/punctuations were discarded when calculating the edit distance. Four categories are established: (1) 1,036 (11%) easy examples, (2) 3,623 (39%) medium examples, (3) 3,681 (40%) hard examples, and (4) 878 (10%) extra-hard examples. Figure 5 shows one representative example from each level. The difficulty breakdown we employ is significantly different from dataset difficulty breakdown or dataset cartography approaches in literature (Swayamdipta et al., 2020; Ethayarajh et al., 2022), since the column name expansion task is formulated as a generation task as opposed to classification tasks considered in prior literature.

4 Methods

Recent advances in pre-trained LMs (Radford et al., 2019; Brown et al., 2020) have shown a strong ability to generate fluent text and the “emergent” performance boost when scaling up LMs (Wei et al., 2022). Therefore, we evaluated the performance of both small and large LMs for NAMEGUESS. We adopted two learning paradigms. First, we employed prompt-based learning with LLMs without tuning model parameters. Second, we fine-tuned small LMs. In particular, we utilized the supervised fine-tuning (Schick and Schütze, 2021a,b) paradigm with task-specific prompts.

Training. We fine-tuned pre-trained LMs by contextualizing column query names with table content, incorporating sampled cell values and table schema data. To limit the sequence length of a linearized table, we selected N cell values in the corresponding column (after removing duplicates) for each query name. We truncated cell values when they had more than 20 characters. Moreover, instead of predicting individual query name separately, we jointly predicted K query names. The columns are stacked together to form the table context data t'. This structured input is then serialized and combined with a task prompt q. Specifically,

- \( t' = \text{"Column names: } \{x_1\}, \ldots, \{x_K\} <\text{SEP} > \text{row 1: } \{c^1_1\}, \ldots, \{c^1_K\} <\text{SEP}> \text{row 2: } \{c^2_1\}, \ldots, \{c^2_K\} <\text{SEP}> \ldots \text{row } N: \{c^N_1\}, \ldots, \{c^N_K\}". \)

- \( q = \text{"As abbreviations of column names from a table, } \{x_1\}|...|\{x_K\} \text{ stand for } \{y_1\}|...|\{y_K\}". \)

Here \( K \) represents the number of query names, and \( N \) is the number of sampled rows in a table. \( \{x_i\}_{i=1}^K \) refers to the abbreviated column names from the same table. \( \{y_i\}_{i=1}^K \) represents the ground-truth logical name. And \( \{c^i_j\}_{i=1}^K, j=1,...,N \) denotes the sampled cell values of each query name. We set \( K \) and \( N \) to 10 based on the ablation study results after testing different \( K \) and \( N \) values in Appendix A.6 Table 7. A table with more than ten columns is split into multiple sequences. The prompt can also be designed in different ways. These prompts were selected by testing various template templates on the GPT-2 XL model and choosing the ones that yielded the best outcomes. We utilized decoder-only models for training, where the entire prompt sequence was employed to recover logical names in an autoregressive manner.

Prediction. Given \( K \) column names \( x_1, \ldots, x_K \) in a table \( t \), we used LM to predict the corresponding logical names \( y_1, \ldots, y_K \). To generate predictions, we concatenated the linearized table context \( t' \) and the task prompt \( q \) as the input sequence. We used the same query prompt as in training during inference, except that we removed the ground truth. The modified query prompt becomes "As abbreviations of column names from a table, \( \{x_1\}|...|\{x_K\} \) stand for". For the non-finetuned LLMs, we provided a single demonstration before the query prompt to ensure that the model can generate answers in a desired format, i.e., "As abbreviations of column names from a table, c_name | pCd | dt stand for Customer Name | Product Code | Date." We extracted the answer in the predicted sequence when the <EOS> token or the period token is met and then split the answers for each query name with the | separator.

5 Experiments

We performed a comprehensive set of experiments to answer (1) Can NAMEGUESS be solved as a natural language generation task? (2) How does fine-tuning and scaling the number of parameters help the model handle NAMEGUESS? (3) Can table contents aid disambiguation?
5.1 Evaluation Metrics

We use three metrics for evaluation: exact match accuracy, F1 scores based on partial matches, and Bert F1 scores based on semantic matches.

Exact Match (EM). Similar to the exact match for question answering, the exact match is computed based on whether the predicted column name is identical to the ground truth after normalization (ignoring cases, removing punctuations and articles).

F1 Score. Computed over individual normalized tokens in the prediction against those in the ground truth, as $2 \cdot \text{precision} \cdot \text{recall}/(\text{precision} + \text{recall})$, where precision and recall are computed by the number of shared tokens in the prediction and ground truth label, respectively.

BertScore. The similarity score for each token in the predicted phrase with that in the reference phrase computed using pre-trained contextual embeddings (Zhang et al., 2020b). It demonstrates better alignment with human judgment in summarization and paraphrasing tasks than existing sentence-level and system-level evaluation metrics. We use rescaled BertScore F1 and roberta_large (Liu et al., 2019) for contextual embeddings.

5.2 Representative Methods

We fine-tuned GPT-2 (Radford et al., 2019) and GPT-Neo (Black et al., 2021) models from Hugging Face. We conducted preliminary experiments using the pre-trained small LMs without fine-tuning but consistently obtained incorrect results. We also evaluated non-finetuned LLMs, including Falcon-40B-Instruct (Almazrouei et al., 2023), LLaMA-65B (Touvron et al., 2023), and GPT-4 (OpenAI, 2023) using the same prompt. Furthermore, we collected human performance from 6 annotators on 1,200 samples from the evaluation set, including 300 from each difficulty level. The annotators had access to 10 sampled cell values for each column. Detailed setups are in Appendix A.5.

5.3 Results and Discussion

The main results, including models with and without table context data, are in Table 4. Table 5 reports the EM results on four hardness levels, and the F1 and Bert-F1 results are in Appendix Table 6. The responses of Falcon-instruct, LLaMA, and GPT-4 models may not match the demonstration example, resulting in potential answer extraction failures. In total, 92% of Falcon-instruct-40B, 96% of LLaMA-65B, and 99% of GPT-4 examples have successfully extracted answers. The scores in the tables are calculated based on the predictions with successful extractions.

Effect of Model Size. From Table 4 we see that among the fine-tuned models, the GPT-2-124M model exhibits particularly poor performance, while the fine-tuned GPT-Neo-2.7B model achieves the highest overall EM, F1, and Bert F1 scores. With a similar number of parameters, the GPT-Neo-1.3B model has a 3.8% higher overall EM score than GPT-2-1.5B. With a much larger size, GPT4 achieves 29.6% higher EM than the best fine-tuned model.

Effect of Fine-Tuning. Without fine-tuning, the small to medium models achieve <1% overall EM scores, producing almost random predictions. However, the NAMEGUESS training data allows a series of fine-tuned small to medium-sized models to approach human performance with a much cheaper inference cost than LLMs. Still, a big gap exists between these fine-tuned models and LLMs.

Human Performance. One important observation is that human performance on NAMEGUESS test set (also human-annotated) is far from perfect. However, the fine-tuned models can slightly exceed human performance (the fine-tuned GPT-Neo-2.7B is 0.4% higher in EM). Intuitively, expanding from query name $x$ into logical name $y$ is much more challenging than reverse since it requires a deeper understanding of the meaning and context of the abbreviations to identify and reconstruct the original phrase accordingly.

Effect of table context data. Expanding abbreviated column names may require a deep understanding of table content. For example, "E_NAME" can represent “Employer Name” instead of “Employee
Table 5: Exact match scores (%) of fine-tuned models, LLMs (non-finetuned), and human performance for four difficulty levels. * indicates fine-tuned models.

<table>
<thead>
<tr>
<th></th>
<th>Easy</th>
<th>Medium</th>
<th>Hard</th>
<th>Extra Hard</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT-2 (124M)</td>
<td>41.4</td>
<td>30.3</td>
<td>19.0</td>
<td>16.1</td>
</tr>
<tr>
<td>GPT-2 (1.5B)</td>
<td>53.8</td>
<td>42.7</td>
<td>32.3</td>
<td>21.1</td>
</tr>
<tr>
<td>GPT-Neo (1.3B)</td>
<td>54.3</td>
<td>47.2</td>
<td>36.8</td>
<td>23.6</td>
</tr>
<tr>
<td>GPT-Neo (2.7B)</td>
<td>55.3</td>
<td>50.4</td>
<td>38.0</td>
<td>27.2</td>
</tr>
<tr>
<td>Falcon-Inst.</td>
<td>62.2</td>
<td>59.3</td>
<td>49.0</td>
<td>36.5</td>
</tr>
<tr>
<td>LLaMA (65B)</td>
<td>62.2</td>
<td>62.9</td>
<td>52.3</td>
<td>37.9</td>
</tr>
<tr>
<td>GPT-4</td>
<td>71.7</td>
<td>79.9</td>
<td>72.0</td>
<td>54.7</td>
</tr>
<tr>
<td>Human</td>
<td>53.7</td>
<td>53.7</td>
<td>33.7</td>
<td>30.2</td>
</tr>
</tbody>
</table>

Name” with company names among column values. By comparing the performance of fine-tuned and non-fine-tuned models with sampled table content ($t' + q$) and without table content ($q$ only) in Table 4, we find that incorporating sampled table contents can increase the performance of all the models, with a huge boost of 15% for the smallest GPT-2 (124M) model.

**Difficulty Breakdowns.** We observe a trend of decreasing exact match scores on more difficult examples. From the difficulty breakdown, the fine-tuned small LMs can outperform humans on easy examples but are worse on extra hard examples. Conversely, GPT-4 outperforms all other models and human results, especially in medium and hard divisions. The LLMs are infused with broader knowledge regarding hard and extra hard examples. They are better at interpreting the exact meaning of the misleading abbreviations that often involve uncommon acronyms words or ambiguous ways of combining abbreviations.

6 Related Work

6.1 Table Understanding and Metadata Augmentation Tasks

Great strides have been achieved in table understanding (Dong et al., 2022). Descriptive column names are crucial for task performance, aiding the model in comprehending table semantics and conducting cell-level reasoning. Column type detection (Hulsebos et al., 2019; Zhang et al., 2020a; Suhara et al., 2022) and column relationship identification (Deng et al., 2022; Iida et al., 2021; Wang et al., 2021) involve assigning predefined types, like semantic labels or database column relationships. Semantic column type detection and NAMEGUESS, though related to columns, have distinct objectives. The former predicts types with predefined labels (classification task), while the latter refines tokens within names, often at the word level (e.g., “c_name” expands to “customer name”). Table question answering (Yin et al., 2020; Herzig et al., 2020; Yang et al., 2022; Xie et al., 2022) requires models to understand both tables and natural language questions and to perform reasoning over the tables. Other tasks, such as table description generation (Gong et al., 2020), table fact verification (Eisenschlos et al., 2021), and formula prediction (Cheng et al., 2022), also require meaningful column names to aid the model in understanding the overall semantic meaning of tables.

6.2 Abbreviation Expansion and Acronym Disambiguation

Abbreviations are widely used in social network posts (Gorman et al., 2021), biomedical articles (Yu et al., 2002), clinic notes (Wu et al., 2015) and scientific documents (Zilio et al., 2022; Pouran Ben Veyseh et al., 2020). Abbreviation expansion and acronym disambiguation tasks are typically formulated as classification problems (Ammar et al., 2011; Pouran Ben Veyseh et al., 2020), which involve selecting an expansion for an abbreviation from a set of candidates based on the context information. The ad hoc abbreviation expansion task (Gorman et al., 2021) is similar to the NAMEGUESS task but limits the per-token translation imposed on the dataset creation process and the developed solutions. Meanwhile, NAMEGUESS is a natural language generation problem suitable for a wide range of lengths of abbreviations and expansions. Regarding abbreviation expansion, our most relevant work is by Cai et al. (2022). However, this work primarily addresses text messages/SMS abbreviations, aiming to reduce message length and minimize typos. In contrast, our task focuses on generating meaningful and human-readable expansions for column name-based abbreviations.

7 Conclusion and Future Works

We introduced a new task related to expanding the commonly-used abbreviated column names in tabular data, developed and created two benchmark datasets to facilitate the study of this task, and analyzed the performance of many language modeling methods to solve this task. One future direction is to utilize similar examples that provide contextual information to solve this task, preferably feeding these examples through in-context learning.
8 Ethics Statement

The human annotations, including abbreviated/logical column names for the evaluation set, were collected through hired annotators from a data annotation service. Annotators were instructed to strictly refrain from including any biased, hateful, or offensive content towards any race, gender, sex, or religion. The annotations passed through audits, where they were examined by a separate group of annotators and reached a 96.5% agreement ratio. The human performance on the NAMEGUESS test set was collected from database and dialogue/linguistics experts.

9 Limitations

One limitation of our work is the need for real relational database tables. The training and evaluation sets we used were all public web-related tables, which generally have fewer rows and lack the metadata of primary and secondary keys. This work is just the first step to introduce this topic to the NLP community, and further research is needed to improve the performance with methods that better capture context around relational data. Moreover, the scope of handling omitted information in the original column names, like column header “glucose”, which stands for “fasting glucose”, is beyond NAMEGUESS. One possible solution is to collect and utilize more table metadata information. For example, if the table name contains the term “fasting”, then the original column name “glucose” will most likely be inferred as “fasting glucose”.

References


Figure 6: Unionable/Joinable relation extraction tasks with abbreviated column names. The column names in blue are the corrupted names. Column D_ID should match Department_ID, D_NAME with Department_Name, and E_NAME with Name for unionable column pairs. Column E_NAME matches with column Employee_Name for the joinable relation.

Figure 7: A table Question Answer (QA) task example. After corrupting the column names in the table (i.e., the names in blue), the keyword in the question ‘college’ may fail to locate the abbreviated column name Col.

A Appendix

A.1 The Effect of Abbreviated Column Names on Table Understanding Tasks

Unionable and Joinable Relation Detection. The unionable and joinable column relation extraction tasks are illustrated in Figure 6. The original recall score of the schema-based relation detection (with expanded column name) in Table 1 is cited from the Valentine paper (Koutras et al., 2021). The datasets used in Valentine were fabricated by corrupting the column names or the column cells. The results of the original datasets have the same column names in two tables, so the recall score is 1.0. The best schema-based method (i.e., Similarity Flooding (Melnik et al., 2002)) drops from 1.0/1.0 to 0.63/0.56 (an average score of 0.595) on unionable/joinable relation detection tasks with expanded and abbreviated column names. As illustrated in the Valentine study, even when other schema information, such as data types and transitive relationships, is available, the lack of descriptive column names can hinder the effectiveness of schema-based methods in identifying unionable and joinable columns.

Table Question Answering Table question answering (QA) aims to derive answers from tables for natural language questions. Typically, the task requires two steps: translating natural language questions to corresponding SQL queries and then extracting the answers from the SQL queries. In order to test the effect of the abbreviations of the column names for Table QA, we used the abbreviated method in section 3.2.2 to corrupt the column names in tables of the WikiSQL (Zhong et al., 2017) dataset and used the same training and evaluation script in UnifiedSKG (Xie et al., 2022) with the T5-large model. The accuracy drops from 84.32 to 80.49, recorded in Table 1.
A.2 The Logical Name Identification Algorithm
See Algorithm 12.

Algorithm 2 | Logical name identification
\[\text{procedure ISLOGICALNAME}(x)\]
\[\text{if } x \in V \text{ then}\]
\[\text{return 1} \triangleright \text{well-curated}\]
\[\text{else}\]
\[x \leftarrow \text{tokenize}(x)\]
\[\text{for } x_i \in x \text{ do}\]
\[\text{if isdigit}(x_i) \text{ then:}\]
\[\text{continue}\]
\[x_i \leftarrow \text{lemmatize}(x_i)\]
\[\text{if } x_i \notin V \text{ then}\]
\[\text{return 0} \triangleright \text{not well-curated}\]
\[\text{return 1} \triangleright \text{well-curated}\]

A.3 Training/Evaluation Set Details
Some variations in the table size are included in the training and evaluation set. As seen from Figure 8, most tables have fewer than 100 columns, and there could be some small tables with fewer than five columns since the columns that initially had abbreviated ambiguous names were filtered before generating abbreviated columns. We limit the number of rows per table for training and evaluation sets to less than 1000.

A.4 Annotation Interface
Figure 9 shows the interface that the annotators used to produce both the abbreviated column names (“Cryptic Variant”) and well-curated column names (“Well-Curated Variant”) based on the column contents (“Original Column Name”, “Sampled Column Values”) and table metadata (“Table Name”, “Table Category”, and “Table Description”). The “Comments” column includes some explanations or clarifications for ambiguous samples.

A.5 Experimental Setup
All fine-tuning experiments on GPT-2\(^1\), GPT-2-XL\(^2\), GPT-Neo-1.3B\(^3\), GPT-Neo-2.7B\(^4\) and inferences were run using eight NVIDIA A100 GPUs. The inference results from GPT-4 was generated using OpenAI API calls\(^5\). The model weights for LLaMA-65B\(^6\) and Falcon-Instruct-40B\(^7\) models were downloaded and hosted using Text Generation Inference (TGI) toolkit\(^8\). We used a global batch size of 128 and a learning rate of \(10^{-6}\) during the fine-tuning process. We used beam search with a beam size of 5 during inference, which performed better than decoding by sampling. The results of the evaluation benchmark reported in the experimental tables were obtained from the best checkpoint of each model on a single run.

1https://huggingface.co/gpt2
2https://huggingface.co/gpt2-xl
3https://huggingface.co/EleutherAI/gpt-neo-1.3B
4https://huggingface.co/EleutherAI/gpt-neo-2.7B
5https://platform.openai.com/docs/models/gpt-4
6https://ai.meta.com/blog/large-language-model-llama-meta-ai/
7https://huggingface.co/tiiuae/falcon-40b-instruct
8https://github.com/huggingface/text-generation-inference
Table 6: Exact match (EM), F1, and BertScore-F1 (B-F1) scores (%) of fine-tuned models, LLMs (non-finetuned), and human performance for four difficulty levels. * Fine-tuned models.

A.6 Ablation Studies

A.6.1 Difficulty Breakdowns.

The full results of all the models on four different difficulty levels are in Table 6.

A.6.2 The Effect of Different Jointly Predicted Query Column Names $K$ and Different Number of Sampled Rows $N$.

We evaluated the exact match (EM) scores on varied values of $K$ and $N$ on the LLaMA-65B model. From Table 7, we note that an increase in the values of $K$ and $N$ results in a rise in EM scores, although the rate of increase tends to diminish for higher $K$ and $N$ values. Thus, we set the values of $K$ and $N$ at 10 for other models in the experiments.

Table 7: Exact match scores (%) with different $K$ and $N$ values for the LLaMA-65B model.
Figure 9: A screenshot for the interface used by annotators.