SEMANTIC ENRICHMENT OF DATABASE COLUMNS USING GENERATIVE LANGUAGE MODELS FOR ADVANCED QUERY SEARCHES

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Dylan James Miska

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

April 2024

Fargo, North Dakota

North Dakota State University
Graduate School

Title

SEMANTIC ENRICHMENT OF DATABASE COLUMNS USING GENERATIVE LANGUAGE MODELS FOR ADVANCED QUERY SEARCHES

By

Dylan James Miska

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Jeremy Straub
Chair

Dr. Juan Li

Dr. Megan Orr

Approved:

May 11, 2024
Date

Dr. Simone Ludwig
Department Chair

ABSTRACT

This study introduces a novel application of natural language generation (NLG) models to improve database table retrieval. Unlike previous works primarily utilizing embeddings and natural language processing (NLP) models, this work explores using NLGs to generate database column descriptions to enhance search accuracy. The evaluation involves two main aspects: firstly, assessing the accuracy of AI-generated column descriptions compared to ground truth descriptions; secondly, examining the impact of these descriptions when integrated into existing search models to evaluate accuracy improvements. Results indicate improved semantic alignment when comparing generated descriptions to ground truth over column names alone and improved scores for established work.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Straub for his support throughout navigating the paper. His advice on direction with the paper helped make this work possible. I would also like to thank my committee members, Dr. Li and Dr. Orr, for their interest in my work and willingness to serve on my committee. Additionally, this work used resources of the Center for Computationally Assisted Science and Technology (CCAST) at North Dakota State University, which were made possible in part by NSF MRI Award No. 2019077. I am grateful for the access to these resources, as they were instrumental in the completion of this research.

TABLE OF CONTENTS

ABSTRACT iii

ACKNOWLEDGMENTS iv

LIST OF TABLES vii

LIST OF FIGURES viii

1. INTRODUCTION 1

2. BACKGROUND 2

2.1. Embedding Explanation and Its Influence on NLP Models 4

2.2. Transitioning to Transformers and BERT 4

2.3. Generative Models: The Emergence of GPT 5

3. RELATED WORK 7

4. APPROACH 9

4.1. Bert Encoding 9

4.2. TaBERT and StruBert Tabular Processing 10

4.3. Column Name Enhancement 11

4.4. Experiments 12

4.5. Setup 12

5. EXPERIMENTS AND RESULTS 15

5.1. Comparing Models 15

5.2. Comparing Configurations 19

5.2.1. Custom Function 19

5.2.2. Adjusting Temperature 22

5.2.3. Max Token Setting 22

5.3. Comparing Prompts 23

5.4. Examining Context Significance 26

5.5. Additional Analysis 28
5.6. Impact on Related Work 29
6. CONCLUSIONS AND FUTURE WORK 32
REFERENCES 33

LIST OF TABLES

Table		Page
1.	Example Generated Descriptions.	11
2.	Model Mean Scores by Metric.	16
3.	Example Generated Descriptions by Model.	17
4.	Model Mean Scores by Metric for Custom Function.	21
5.	Custom Function Example Generated Descriptions.	21
6.	Model Mean Scores by Metric for Temperature.	22
7.	Model Mean Scores by Metric for Max Tokens.	23
8.	Prompt Mean Scores by Metric.	26
9.	Context Mean Scores by Metric for Context Significance.	28
10.	StruBert Mean Metric Scores.	31
11.	StruBert P Value Metric Scores.	31

LIST OF FIGURES

Figure		Page
1.	Baseline Prompt.	15
2.	Metric Distributions by Model.	18
3.	Custom Function Prompt.	19
4.	Custom Function Open API Spec.	20
5.	Three Shot – Example 1 Prompt.	23
6.	Three shot – Example 2 Prompt.	24
7.	Brief Column Descriptions Prompt.	24
8.	For Unfamiliar Individuals Prompt.	24
9.	Database Format Prompt.	25
10.	Additional Context Prompt.	25
11.	Documenter Role Prompt.	25
12.	No Table Names or Datatypes Prompt.	26
13.	Single Column Given Prompt.	27
14.	Constraint Context Prompt.	27
15.	Example Constraint Context Provided.	27
16.	Embedding Score vs Change in Position Score.	29
17.	WIKITables Prompt.	30

1. INTRODUCTION

Over the past several decades, locating relevant data within expansive databases, also known as data discovery, has grown increasingly challenging. As these databases expand, they often encompass a vast number of tables and columns, which are not always well-annotated or documented. This issue has drawn significant attention from data scientists, corporations, and researchers, underscoring a pressing need for effective solutions.

This paper aims to investigate an unexplored avenue in addressing this challenge. Recent years have witnessed the rising popularity of generative language models, acclaimed for their versatility in solving a broad spectrum of knowledge problems. This study aims to utilize this capability to enhance the search retrieval accuracy of relevant database tables in response to user queries. Specifically, the paper explores the performance of various generative language models in accurately generating database columns. These generated descriptions will then be utilized instead of column names in an existing work to evaluate accuracy improvement.

2. BACKGROUND

As data discovery has become a more significant problem, various solutions and avenues have emerged. Overall, the common problem remains the same: finding relevant data from source(s) based on some form of search criteria. Some of these strategies include text-to-SQL [1], keyword driven search [2], dataset driven search, table union search, join-correlation search, ontology-based search [3], and several others. Each of these strategies, and how they address the problem, are now discussed.

Keyword-driven Search: Keyword-driven search, a foundational approach in data retrieval, involves analyzing a user's search query to identify relevant data sources [2]. This strategy resembles a search engine's functioning but is specifically tailored for tabular datasets. For example, it may involve matching keywords in a query to those in table headers or column descriptions, allowing for retrieving tables that best match the query terms. This technique is used in **[**2, 4, 5, 6].

Dataset-driven Search: Dataset-driven search differs from keyword-driven search by using an entire dataset as the input for finding relevant data sources rather than a user-entered query [3]. This approach often involves comparing datasets' structures, schemas, or contents to identify similarities and relevancies, effectively allowing for the discovery of datasets that share characteristics with the input dataset. This technique is used in **[**2, 5].

Text-to-SQL: Text-to-SQL takes a unique approach by converting a user's natural language query into an SQL query [1]. This method abstracts the user from the technicalities of query formulation, directly translating their intent into a database query. For instance, a user asking 'show me sales data from the last quarter' would have their request automatically converted into the corresponding SQL command. This technique is used in **[**1**].**

Table Union Search: Table union search adds a layer of abstraction beyond basic search retrieval by combining relevant columns from different tables to assemble the requested dataset [3]. For example, suppose a query asks for customer information and purchase history. In that case, this method might find the relevant data from multiple sources and stitch it together before presenting it as the response. This technique is used in **[**7, 2, 8].

Join Correlation Search: Join correlation search is similar to table union search in that it can stitch together tables to display results. However, it can also manipulate and integrate data from various tables to create a dataset that meets the user's complex requirements, such as correlating product IDs with sales figures across multiple tables [3]. This technique is used in **[**7, 2**].**

Ontology-based Search: Ontology-based search involves preprocessing data into an ontology – a structured representation of the data's concepts and relationships [3]. This method typically uses model-driven annotations to type columns and datasets, leading to efficient data retrieval. For example, an ontology might categorize data by topics or themes, enabling users to retrieve data that fits within a specific conceptual framework. This technique is used in **[**3].

While the strategies above yield varying results, they all depend on finding relevant datasets initially. This comparison process can utilize several methods [9]. Keyword bag-of-word analysis simplifies text into a set of keywords without considering word order; statistical methods might involve frequency analysis or other quantitative metrics. However, the overwhelmingly popular method today appears to be embeddings. Embeddings, often used in machine learning, represent items in a high-dimensional space, allowing for more nuanced similarity assessments.

Consequently, recent research has increasingly focused on leveraging embeddings for data discovery techniques. This work aims to explore a nuanced approach to enhancing the use of

embeddings, offering a novel contribution to the field. The specifics of Embeddings and the models that produce them are discussed in the next section.

2.1. Embedding Explanation and Its Influence on NLP Models

Regarding the field of NLP, embeddings have revolutionized how AI can process language [9]. Embeddings transform words or phrases into numerical vectors. They map words or phrases to vectors in a high-dimensional space where similar meaning terms are grouped, encapsulating richer details about their context and relationships. Among their many use cases, they can be directly compared to each other [10] to determine word or phrase similarity, which is essential for ranking similar results. These advancements have allowed for more sophisticated text processing, allowing AI to analyze language with a nuance closer to human understanding.

2.2. Transitioning to Transformers and BERT

The development of advanced embeddings beyond simpler models such as word-to-vec [11] has paired closely with the advancements of NLP models, particularly the transformer model introduced in [12]. This model's self-attention mechanism processes words by considering their full sentence context, drastically enhancing the interpretation of language subtleties. This marked a departure from traditional, sequential text processing, allowing for models that capture deeper linguistic context.

Building on this, Google's bidirectional encoder representations from transformers (BERT) [13] represents another significant advancement. BERT's bidirectional approach processes the context of words from both directions in a sentence rather than linearly. This is possible thanks to dynamic embeddings, which adapt according to the word's surrounding context. Such embeddings have been essential in BERT's ability to deeply interpret language, which is crucial for applications such as database table ranking [10].

In the context of this research, BERT's capabilities are leveraged for their proficiency in interpreting and processing the context and meaning of columns and tables. This is the key advantage it has over other methods in this space. The dynamic nature of BERT's embeddings allows for a more accurate alignment of natural language queries with relevant datasets, addressing the crucial need for understanding the nuanced meanings within database columns.

2.3. Generative Models: The Emergence of GPT

Following the advancements brought forth by models like BERT, the field of NLP has witnessed the emergence of generative models, a notable example being generative pre-trained transformer (GPT) [14], developed by OpenAI. Unlike Bert, which is primarily designed to understand language (making it an encoder), GPT is built to generate text (functioning as a decoder). GPT was a new application of NLP models capable of producing human-like text, often referred to as NLG models. These models allowed for new applications, such as content creation and conversational AI.

The critical difference between GPT (NLG) and models like BERT (NLP) lies in their training and intended use. GPT is trained on an extensive range of text, allowing it to generate coherent and contextually relevant text based on a given prompt. This ability to generate text makes GPT suitable for tasks that require creative language generation. In contrast, BERT's bidirectional understanding of context makes it ideal for tasks involving language interpretation, such as sentiment analysis, question-answering, and information retrieval.

It is believed that the application of NLG models is largely yet to be explored for dataset retrieval. One exception is Text-to-SQL, which has incorporated NLG for transforming relevant table data to SQL queries, but finding the relevant tables is left to embeddings.

A potential limitation in prior methodologies, particularly those relying on pre-trained models such as BERT, is their intrinsic focus on interpreting text as-is, without the added layer of generating new context or extrapolating beyond the given data. While BERT is adept at contextual understanding, its architecture is not designed for creative text generation or conjecture. In contrast, NLG models such as GPT are not only capable of interpreting text but excel at generating new content that can reflect implications or hypotheses based on the available data. This distinction suggests that generative models could offer a more comprehensive understanding of database columns, not just by analyzing existing descriptions but also by suggesting enriched, contextually relevant interpretations.

3. RELATED WORK

In the area of database table retrieval, several recent studies have suggested novel methods. Several relevant recent contributions and how they relate to the work presented here are now discussed.

Zhang and Balong [4] proposed a novel approach to handling table retrieval tasks using embeddings. Their approach was based on utilizing embeddings to compare the semantic meaning between tables and input queries. They used multiple representations and similarity measures in a supervised learning model, substantially improving retrieval performance over prior methods primarily relying on lexical matching.

TABERT, introduced by Yin, Neubig, Yih, and Riedel [15], represents a novel effort in interpreting both textual and tabular data. TABERT, which is a variant of the BERT model, was trained on the WIKITABLES dataset to understand and interpret tables along with input queries for enhanced retrieval performance. It utilized new methods to parse and handle larger contexts of tables using "snapshots" created by encoding and manipulating portions of table data.

Gao, et al. [1] explored utilizing an LLM for prompt engineering in text-to-SQL tasks. The study evaluated the effectiveness of optimizing LLM prompts to generate accurate SQL queries from retrieved relevant tables. While Gao, et al. focused on generating SQL queries from tables, the focus of this work is on the underlying table retrieval, which NLG did not enhance in their study. Instead, their work utilized more traditional methods for this process, including embeddings. Similarly to their work, this paper investigates prompt designs for accuracy.

Trabelsi, et al. [5] offered a novel approach for combining structural and textual table data. Building on the work of TABERT, they used the base model, structured the tabular data as row and column-based sequences, and applied horizontal and vertical self-attention to capture the relationships within the table's data. In addition, they utilized a model they called MiniBert to

aggregate the results and determine table relevance to input data, including queries and other tables. StruBERT is utilized as a baseline for comparison in this work.

Each of these recent papers contributed to the field of table data retrieval. Prior work primarily focused on utilizing embeddings with the BERT model or variations to compare and rank relevant table results. The work presented here makes a contribution by utilizing NLG models to enhance existing results that rely on embeddings for relevant table retrieval tasks, such as the prior work explained here.

4. APPROACH

The approach used for this work differs from prior work in utilizing generative AI to enhance existing methods. To better understand how and why this was used, current methods to encode database tables using embeddings are now discussed. First, how the BERT model creates embeddings is considered. This is followed by a discussion on how the current state-of-the-art model, StruBert, processes tabular data before utilizing the BERT model.

4.1. Bert Encoding

The Bert encoding process occurs in a sequence of steps. Starting from a given sentence, for example, the following steps are performed:

First, tokenization is performed. Bert utilizes WordPiece tokenization to break up an input into tokens. These are trained words or sections of words that the model understands and has been trained on. In addition, special tokens are added to the token sequence. These include CLS and SEP, which indicate the start and separation points in the token sequence.

Second, tokens are converted into embedding vectors that the model can process and use. These embeddings are not the final embeddings but are a base that will be processed. These base embeddings then go through attention layers and are contextualized, meaning that they represent the token and the context from the tokens surrounding it. Surrounding token context is essential because a word can mean different things in different contexts.

Finally, after processing, the BERT model outputs a sequence of contextualized embeddings corresponding one-to-one to the input tokens. These embeddings can now be utilized as needed.

4.2. TaBERT and StruBert Tabular Processing

Next, the use of embeddings by the StruBert and TaBert (which StruBert builds on) models are discussed. One approach to handling the encoding process of tabular data is to attempt encoding the entire table; however, BERT has a restrictive token limit and may not be able to interpret the meaning of the data well. To handle the latter, TaBERT was trained on the WIKITABLES dataset to interpret and understand tabular data and context better. StruBert uses this model to interpret, process, and formulate the data into contextual embeddings. To process the data StruBert follows a series of steps:

First, row and column processing is performed. StruBERT creates sequences from a table's rows and columns. For example, a three-by-three table would be turned into a total of nine token sequences. Each token sequence includes common table data redundantly. Each sequence starts with context, like the table name and metadata, and then moves through each cell that is contained in each row and column. Special tokens are also added for separation purposes.

Second, TaBert processing is performed. Once the sequences are created, they are processed with the TaBert model to produce embedding sequences, as explained previously with BERT. The output embeddings contain contextualized information for each token in the row and column input token sequences. The special CLS token is utilized as a summarization for each row and column.

Third, MiniBERT is used. Once the row and column embeddings have been created, they are processed again by another model Stubert calls MiniBERT. This model interprets the output embeddings from TaBERT and produces a relevance score for a table to query or another table.

4.3. Column Name Enhancement

The main contribution of this work is an enhancement of previous work. Previous work generated embeddings based on table metadata, column names, and table data. One potential limitation, however, is that column names can be problematic within the dataset or databases in general. For example, within the WIKITABLES dataset, column names can be missing, abbreviated, shorthanded, challenging to interpret, and potentially not have been within training data. This paper proposes a process to enhance the data before utilizing an NLG model. By enhancing the column names with full column descriptions, the NLP model can better interpret the meaning of the columns.

Table 1 shows how full English descriptions have been generated to allow the BERT model to interpret the columns' meanings more readily instead of relying on table data for the missing link in context.

Table 1. Example Generated Descriptions.

Column Name	Actual Description	Generated Descriptions
KCODE	Accounting type code for PPE and IA	The code associated with the account kind.
ATTACHEDDOCS	Number of documents attached	The number of attached documents for the advanced report.
ANLPLAN5_ID	5th Order Analytics	The ID of the fifth analysis plan associated with the batch.

Several models and methods can be used to generate these descriptions. The technique differs slightly depending on the model; however, there is a general process. First, a prompt is designed. The prompt combines context and instructions for the model to follow. The prompt is then sent to the model to be processed. Finally, a response is returned and parsed according to the requested format.

4.4. Experiments

To add additional column context, a process is needed to generate descriptions for each column in each table based on a prompt. Once the descriptions have been generated, they may be utilized in table retrieval tasks. To determine the best model and methods to generate descriptions, several experiments were performed to compare generated descriptions to ground truth. Testing was then performed on the generated descriptions within the StruBert model.

4.5. Setup

Similarity comparison to ground-truth descriptions was used to determine the best language models, configurations, and prompts to generate accurate descriptions. This was obtained from a large open-source ERP project called Millenium BSA [16]. While the columns were in English, the descriptions were in Russian, however, so they were translated to English for ease of use and to ensure optimal functionality with embeddings. A feedback loop was created to test multiple models, configurations, and prompts.

The process consists of 3 main steps. These are now discussed.

First, description generation is performed. Descriptions are generated with a script that first defines and connects to the model being tested. Some models vary on prompt setup, but all models need at least a base prompt to tell the model to generate descriptions for a set of column names. The script requests columns by each table so that each column may have the context of the surrounding columns. In addition, the table name, metadata, and column data types are used as additional context. The results are then parsed and extracted from the response from the model. One note is that due to the size of some tables, it was not possible to request descriptions for an entire table, and batching was implemented to request only up to 10 columns at a time.

Second, once the descriptions are generated, an embedding was created for each ground truth description, generated description, and column name. OpenAI Ada 2 embeddings were used for this process. These embeddings differ from BERT embeddings in that they encompass an entire phrase instead of a single token. This allows for a direct comparison between ground truth and generated description embeddings.

Third, evaluation is performed. For evaluation, four metrics were used: Bilingual Evaluation Understudy (BLEU), Recall-Oriented Understudy for Gisting Evaluation (ROUGE), embedding comparison score, and relative position score. A final script calculates the scores for each metric and results are recorded.

BLEU is a standard evaluation metric to measure machine translation quality and focuses on precision. ROUGE is a standard evaluation metric to measure text summarization and focuses on recall. Note that BLEU and ROUGE serve as additional information about the exact words used. This work does not necessarily gain by having the exact words as the ground truth descriptions.

The embedding comparison score compares the ground truth description embedding to the generated description embedding using cosine similarity, which is the normalized dot product between x and y. The distance between the embedding vectors is thought to compare the semantic meaning of the text.

The relative position score refers to the ranking of the results. For each ground truth description, all the generated descriptions were ranked using cosine similarity and the position of the correct description in the list was noted. For both the embedding comparison and the position score, the column name was compared to the ground truth description as a baseline. This is useful because prior work uses the column name to allow the embeddings to interpret the meaning of the

column. The goal is to allow the embeddings to interpret the column as closely to the ground truth description embeddings as possible.

5. EXPERIMENTS AND RESULTS

The subsequent sections describe the observations from each of the experiments, including the model, configuration, prompt, and context inclusion comparisons. These are followed by a brief analysis of the impact of the description replacement. Finally, an experiment showing the results of replacing the column names with generated descriptions will be discussed.

5.1. Comparing Models

Four models were evaluated to identify performance differences. These were OpenAI's GPT 3.5 and GPT 3.5 0613, Meta's LLAMA2, and Google's PALM BISON. OpenAI's GPT3.5 0613 is an updated version of the 3.5 model. The same prompt was used for all models to observe the performance equally for each. This prompt is shown in Figure 1. From here on, it will be referred to as the baseline prompt.

Figure 1 shows the baseline prompt which provides the model with the schema, table name, and a list of the columns for which descriptions are to be generated.

```
lambda column_names, table_name, schema: \
f"Please provide descriptions for the following columns: {column_names}\n\
Please format your responses as 'Column: example_column, Description: Example column description'\n\
The table name and full schema are below\n\
Table name: {table_name}\n\
Schema:\n\
Column name, Data type\n\
{schema}"
```

Figure 1. Baseline Prompt.

Table 2 shows the results for the various models. The Palm model scored the highest on the BLUE, ROUGE, and embedding scores, while GPT3.5 0613 scored highest on the position score. Since the BLEU and the ROUGE scores measure exact words, it follows that the column name baseline would score highest in those categories. This also indicates that PALM may be

returning results that are closer in alignment with the actual column name rather than the desired descriptions. To further investigate, example description results from each model were examined.

Table 2. Model Mean Scores by Metric.

	BLEU Score	ROUGE Score	Embedding Score	Position Score
GPT3.5	0.1088	0.1523	0.8222	0.9309
GPT3.5 0613	0.1171	0.1629	0.8225	*0.9359
LLAMA2 13B	0.1320	0.1831	0.8213	0.9208
PALM BISON	*0.2539	*0.3252	*0.8515	0.9251
Column Name	0.3157	0.3838	0.8439	0.9250

As shown in Table 3, PALM consistently generated shorter and more conservative descriptions that were not much different from the column name. This explains the favorable BLEU and ROUGE scores, which often punish length and reword for the exact words used. GPT3.5 0613, however, generated longer descriptions and moved further away from the column name. This resulted in descriptions that often added considerably more context compared to PALM. The distributions between the models were reviewed to further analyze the differences between the models.

Table 3. Example Generated Descriptions by Model.

Column Name	Actual Description	PALM Description	GPT3.5 0613 Description
KCODE	Accounting type code for PPE and IA	Key code	This column stores a code that represents a specific category or type of account kind
RESTDOCDATE	Document date for balance / overspend	RestDoc date	This column stores the date associated with a document related to the restocking process
ATTACHEDDOCS	Number of documents attached	Number of the attached documents	The number of attached documents related to the advanced reports
ANLPLAN5_ID	5th Order Analytics	ANLPLAN5 ID	This column may store an identifier that represents the fifth level of an analytical plan or categorization system used within the database

The distribution graph presented in Figure 2, shows that the PALM model is closely associated with the column name baseline while the other models have similar distributions. All models and the baseline column name generally had position scores greater than ninety percent, while the GPT models were slightly elevated above the rest.

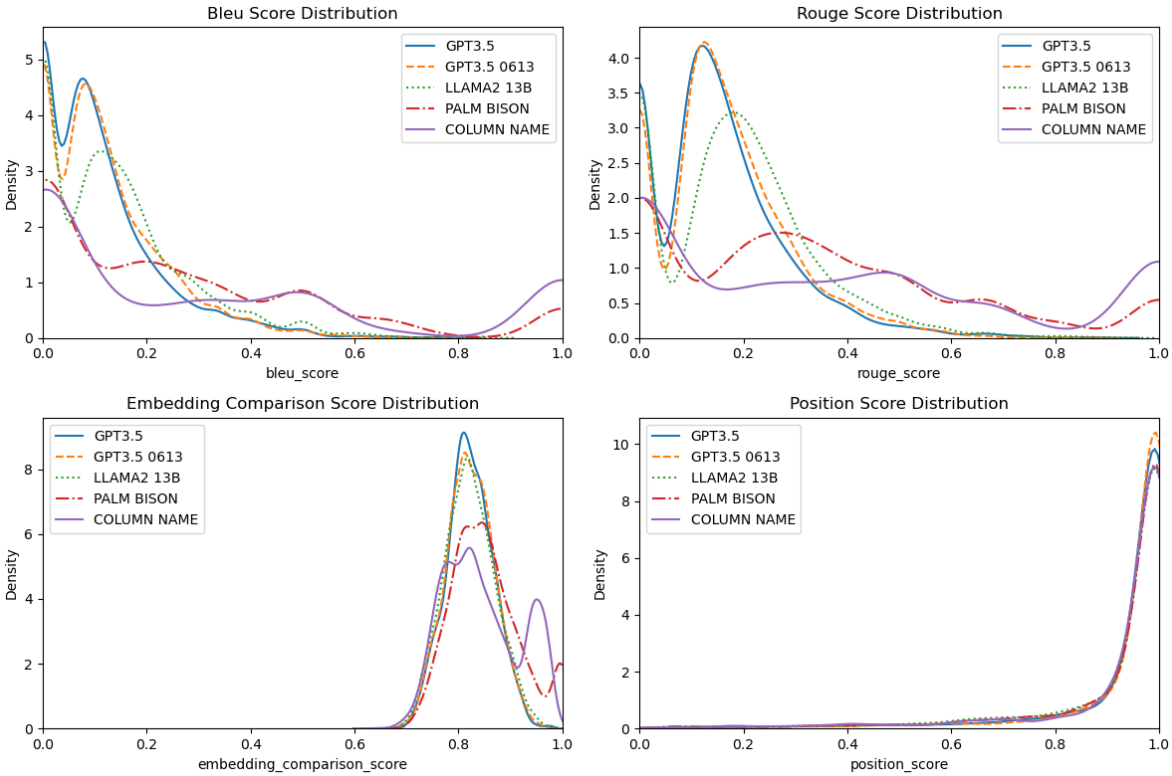

Figure 2. Metric Distributions by Model.

Overall, the results show that the GPT3.5 0613 and PALM models performed the best based on the metrics tested. However, it is important to note that the PALM model correlated much more closely with the baseline and provided descriptions that differed little from the column name alone. One interesting note is that the embedding score and the position score deviated from the results, given that the position score relies on embedding comparisons to rank results for each column. One possibility is that while the generated descriptions were closer to the actual descriptions for the PALM model, the GPT model differentiated the meanings of the columns more, resulting in a more accurate ranking.

5.2. Comparing Configurations

In this section, configuration options that the GPT model offers were investigated to see how they impact the results. The options tested included using a custom function call, increasing/decreasing the temperature parameter, and utilizing a max token limit. Custom functions are a feature offered by Open AI's GPT models that can force the model to return a response in a specific format, specified as Open API JSON format. Temperature refers to the variation or randomness the model may introduce in its responses. Finally, the max token limit is a limitation that can be introduced to force the model to respond in a limited response size.

5.2.1. Custom Function

A slightly different prompt was used for the custom function option, and an Open API spec was used for the response. Figure 3 shows the prompt that was used to instruct the model to generate the descriptions.

```
lambda column_names, table_name, schema: \
f"Please update the following columns with the update_column_descriptions function.\n\
The table name and partial schema are below\n\
Table name: {table_name}\n\
Schema:\n\
Column name, Data type\n\
{schema}"
```

Figure 3. Custom Function Prompt.

It is unnecessary to specify the format in which the model should respond in the prompt. The model was also told that it updates descriptions instead of providing descriptions. This is because this specific variation of the GPT model was designed to provide JSON-formatted responses to perform actions. No actions are actually performed, but this format is used to generate descriptions more reliably.

Figure 4 shows how the model was instructed to return a response in the format of an array of object pairs containing each column name and associated description. The goal is that the model will interpret this as though it is directly updating column descriptions in a database. This is an attempt for the model to avoid speculative responses. Many responses from the baseline were given in a speculative format that differs from those that would commonly be found in a database.

```
{
    "name": "update_column_descriptions",
    "description": "Update the passed list of columns with new descriptions to be added to db metadata.",
    "parameters": {
        "type": "object",
        "properties": {
            "descriptions": {
                "type": "array",
                "items": {
                    "type": "object",
                    "properties": {
                        "column_name": {
                            "type": "string",
                            "description": "The name of the column to update the description of."
                        },
                        "column_description": {
                            "type": "string",
                            "description": "The new proposed description of the column."
                        }
                    },
                    "required": ["column_name", "column_description"]
                }
            }
        },
        "required": ["descriptions"],
    }
}
```

Figure 4. Custom Function Open API Spec.

Table 4 shows the custom function results. It shows improvements in all metrics by utilizing this custom function configuration. As explained previously, the BLEU and ROUGE scores reward exact word usage and similar length results. The results could indicate that this configuration has resulted in closer results to the ground truth column descriptions. To further examine this, examples from the results were examined. They are shown in Table 5.

Table 4. Model Mean Scores by Metric for Custom Function.

	BLEU Score	ROUGE Score	Embedding Score	Position Score
GPT3.5	0.1088	0.1523	0.8222	0.9309
GPT3.5 0613	0.1171	0.1629	0.8225	0.9359
GPT3.5 0613 Custom Function	*0.1600	*0.2168	*0.8362	*0.9423

Table 5 shows that the generated descriptions are more concise and less speculative. They would be more like what would be found in a database and less like a description that may be given in conversation. This configuration accomplishes the task better than the parsing method.

Table 5. Custom Function Example Generated Descriptions.

Column Name	Actual Description	GPT3.5 0613 Description	GPT3.5 0613 Custom Function Description
KCODE	Accounting type code for PPE and IA	This column stores a code that represents a specific category or type of account kind	The code associated with the account kind.
RESTDOCDATE	Document date for balance / overspend	This column stores the date associated with a document related to the restocking process	This date of the restocking document.
ATTACHEDDOCS	Number of documents attached	The number of attached documents related to the advanced reports	The number of attached documents for the advanced report.
ANLPLAN5_ID	5th Order Analytics	This column may store an identifier that represents the fifth level of an analytical plan or categorization system used within the database	The ID of the fifth analysis plan associated with the batch.

5.2.2. Adjusting Temperature

Open AI offers temperature as a parameter for its GPT models, ranging from 0 to 2. This setting adjusts the variability of the responses. More variability can lead to less restrictive responses. For this test, decreased and increased values were specified, and the results were examined. The baseline prompt previously shown was used for this test. Table 6 shows a slight improvement with a lower temperature compared to the base model. Using a higher temperature, on the other hand, results in considerably lower scores.

Table 6. Model Mean Scores by Metric for Temperature.

	BLEU Score	ROUGE Score	Embedding Score	Position Score
GPT3.5	0.1088	0.1523	0.8222	0.9309
GPT3.5 Temperature .5	*0.1170	*0.1556	*0.8223	*0.9353
GPT3.5 Temperature 1.5	0.0796	0.1210	0.8096	0.8995

5.2.3. Max Token Setting

Max tokens is a parameter that Open AI offers in its GPT models that specifies the maximum number of tokens that the model can output. This can be helpful in restricting the model from generating long outputs that may not be optimal as a database description. However, one challenge with this parameter is that it includes the token count for all the output that is given back on results. As with previous experiments, batch sizes of ten were used. A maximum size of 250 tokens was utilized, which is shared across all ten column descriptions the model generates descriptions for. Tables with less than ten columns or the final batch of tables with more than ten columns are not as restricted in their description size. Table 7 shows that this setting produced a minor increase in scores, as compared to the base model.

Table 7. Model Mean Scores by Metric for Max Tokens.

	BLEU Score	ROUGE Score	Embedding Score	Position Score
GPT3.5	0.1088	0.1523	0.8222	0.9309
GPT3.5 Max 250 Tokens	*0.1110	*0.1533	*0.8233	*0.9325

5.3. Comparing Prompts

As there has been evidence of differences in results with NLG models with different prompts [17], several prompts that may alter the generated descriptions' results were examined.

First, three shot examples were used for the prompt that display generic column names and descriptions. This prompt is shown in Figure 5.

```
lambda column_names, table_name, schema: \
f"Given the table and schema, please provide descriptions as your response for the following schema similar to these
examples:\n\
Column: employee_id, Description: Unique identifier for each employee.\n\
Column: first_name, Description: First name of the employee.\n\
Column: last_updated, Description: Timestamp of the last update to the record.\n\
The table name and schema are below:\n\
Table name: {table_name}\n\
Schema:\n\
Column name, Data type\n\
{schema}"
```

Figure 5. Three Shot – Example 1 Prompt.

Next, three shot examples from the dataset were used for the prompt, which display column names and ground truth column descriptions. This prompt is shown in Figure 6.

```
lambda column_names, table_name, schema: \
f"Given the table and schema, please provide descriptions as your response for the columns in the schema below.\n\
Ensure your response format is the same as these examples.\n\
Column: anumber, Description: Personal account number\n\
Column: sum_discount, Description: Amount with taxes included and discount/markup\n\
Column: minincome, Description: Minimum Income\n\
The table name and schema are below:\n\
Table name: {table_name}\n\
Schema:\n\
Column name, Data type\n\
{schema}"
```

Figure 6. Three shot – Example 2 Prompt.

After this, a prompt for the model to respond briefly was incorporated. This is an attempt to limit the response size for each description, similar to what may be in the ground truth descriptions. This prompt is shown in Figure 7.

```
lambda column_names, table_name, schema: \
f"We have some columns and their datatypes from the table '{table_name}' below.\n\
Could you provide brief but informative descriptions for each of them?\n\
Please format your responses as 'Column: example_column_name, Description: Example column description' each on its
own line.\n\
\n{schema}"
```

Figure 7. Brief Column Descriptions Prompt.

The next prompt asks the model to produce descriptions for users unfamiliar with the database to provide additional context. This prompt is shown in Figure 8.

```
lambda column_names, table_name, schema: \
f"Assume you are writing database column descriptions for the table '{table_name}' to assist individuals who are
unfamiliar with this project. \n\
Feel free to make educated guesses. Provide the most likely purpose for each column in a format suitable for
database documentation.\n\
Please format your responses as 'Column: example_column, Description: Example column description'\n\
The schema is as follows:\n\
{schema}"
```

Figure 8. For Unfamiliar Individuals Prompt.

Following this the model was prompted to generate descriptions in the format found in a database. This is an attempt to reduce speculative language in the generated descriptions. This prompt is shown in Figure 9.

```
lambda column_names, table_name, schema: \
f"Given the table and schema, please provide descriptions for the columns below in a definitive and professional tone.\n\
Format your responses as 'Column: example_column, Description: Direct column description'.\n\
Table name: {table_name}\n\
Schema:\n\
Column name, Data type\n\
{schema}"
```

Figure 9. Database Format Prompt.

Next, additional context about the database for which the generated descriptions are intended was provided. This prompt is shown in Figure 10.

```
lambda column_names, table_name, schema: \
f"Context: The Millennium Business Suite Anywhere (BSA) is a comprehensive enterprise management system.\n\
It's designed as a web-based ERP/CRM solution with integrated BPM.\n\
Originating from deep Russian forests, this open-source platform, based on advanced java technologies,\n\
offers functionalities to automate resources' planning (MRPII), distribution, inventory, payroll, HR, purchase, and sales.\n\
BSA can operate on various operating systems, including Unix, Linux, Solaris, Mac, and Windows.\n\
Its ultra-thin client ensures secure operations via the Internet. The system supports businesses of all sizes,\n\
from small entities to large geographically distributed holdings.\n\
Given this context, please provide descriptions for the columns below.\n\
Please format your responses as 'Column: example_column, Description: Example column description'.\n\
The table name and full schema are below:\n\
Table name: {table_name}\n\
Schema:\n\
Column name, Data type\n\
{schema}"
```

Figure 10. Additional Context Prompt.

Finally, the model was assigned a role as though it is a documenter for column descriptions. This is an attempt to gain database format in a concise format. This prompt is shown in Figure 11.

```
lambda column_names, table_name, schema: \
f"Given the table schema below provide a description for each column as if you were writing documentation.\n
Please format your responses as 'Column: example_column, Description: Example column description'\n\
The table name and full schema are below\n\
Table name: {table_name}\n\
Schema:\n\
Column name, Data type\n\
{schema}"
```

Figure 11. Documenter Role Prompt.

Table 8 shows that most prompts resulted in lower scores than the baseline prompt. The 3-shot examples 1 and 2 appeared to map closer to the examples, resulting in higher BLEU, ROUGE, and embedding scores, however they performed considerably lower in the position scores. Only

the “for unfamiliar individuals” prompt had a minimal increase for all metrics as compared to the baseline.

Table 8. Prompt Mean Scores by Metric.

	BLEU Score	ROUGE Score	Embedding Score	Position Score
3 shot – Example 1	0.1433	0.1869	0.8248	0.9159
3 shot – Example 2	0.1623	0.2099	0.8313	0.9223
Brief Column Descriptions	0.1127	0.1577	0.8218	0.9326
For unfamiliar individuals	0.1384	0.1859	0.8290	0.9333
Database Format	0.1296	0.1740	0.8245	0.9255
Additional Context	0.1120	0.1570	0.8230	0.9307
Documenter Role	0.1262	0.1711	0.8245	0.9264
GPT3.5 Baseline	0.1088	0.1523	0.8222	0.9309

5.4. Examining Context Significance

This section considers the various context options and how they affect the descriptions generated. The context options given in previous tests include the table name, other column names, and each column’s data type. The impact of when the table name and data types are removed, only one column is provided per batch, and join constraints are additionally provided are all examined.

First, the table name and datatypes were removed, and instead, the model was asked to provide descriptions with only the column names as context. This prompt is shown in Figure 12.

```
lambda column_names, table_name, schema: \
f"Please provide descriptions for the following columns: {column_names}\n\
Please format your responses as 'Column: example_column, Description: Example column description"
```

Figure 12. No Table Names or Datatypes Prompt.

Next, a similar prompt to the baseline was used, but the grammar was changed slightly to reflect that only a single column was provided as context, unlike the previous batches of 10. This prompt is shown in Figure 13.

```
lambda table_name, schema: \
f"Please provide a description for the column in the schema below.\n\
Please format your response as 'Column: example_column, Description: Example column description'\n\
The table name and full schema are below\n\
Table name: {table_name}\n\
Schema:\n\
Column name, Data type\n\
{schema}"
```

Figure 13. Single Column Given Prompt.

Following this, the standard prompt from baseline was used, but it was modified slightly to provide constraints as additional context shown in Figure 14. The constraints are SQL statements that can provide additional context to the model about column purposes. Figure 15 displays an example of this.

```
lambda column_names, table_name, schema, constraints: \
f"Please provide descriptions for the columns in the schema below.\n\
Please format your responses as 'Column: example_column, Description: Example column description'\n\
The table name, partial schema, and relevant constraints(if any) are below\n\
Table name: {table_name}\n\
Schema:\n\
Column name, Data type\n\
{schema}\n\
Constraints:\n\
{format_constraints(constraints)}"
```

Figure 14. Constraint Context Prompt.

```
ALTER TABLE ALG_CUSTOM_USER_ACTION_PERMISS ADD CONSTRAINT FK_CUSTOM_USER_ACT_PERM_ROL

                FOREIGN KEY (ROLE_ID) REFERENCES SEC_GROUPS (ID)
```

Figure 15. Example Constraint Context Provided.

Table 9 shows that removing the table name and data types slightly reduced the BLEU and ROUGE scores, but improved the position score, implying that the table name and data types had minimal impact on the description accuracy. Removing additional columns from the request significantly reduced all scores, implying that the additional columns play a significant role in

producing accurate descriptions. Finally, the constraint context produced BLEU and ROUGE scores similar to the baseline, but they reduced the embedding and position scores. This could be due to too much context being provided or due to simply confusing the model and drawing its attention away from more meaningful elements.

Table 9. Context Mean Scores by Metric for Context Significance.

	BLEU Score	ROUGE Score	Embedding Score	Position Score
No Table Name/No Data Types	0.0957	0.1382	0.8223	0.9360
Single Column Given	0.0737	0.1123	0.8060	0.9010
Constraint Context	0.1097	0.1558	0.8154	0.9160
GPT3.5 Baseline	0.1088	0.1523	0.8222	0.9309

5.5. Additional Analysis

Additional analysis was performed to understand how the generated columns affected the position score. The change was mapped from the baseline to the gpt3.5 custom function configuration across the embedding score between the column name and the actual description. This shows how the score changes, given how discrepant the column name is from the actual description or meaning of the column.

Figure 16 shows the results of this. The graph shows several vertical lines, most notably near the .94 embedding score mark. Additional analysis showed that these were duplicate columns, specifically the ID columns from all the tables. This shows a limitation of using the position score as a metric, because there were many duplicate ID columns, and the ranking for these would be random. However, this would have been applied to all measurements, so it should not have biased experiments.

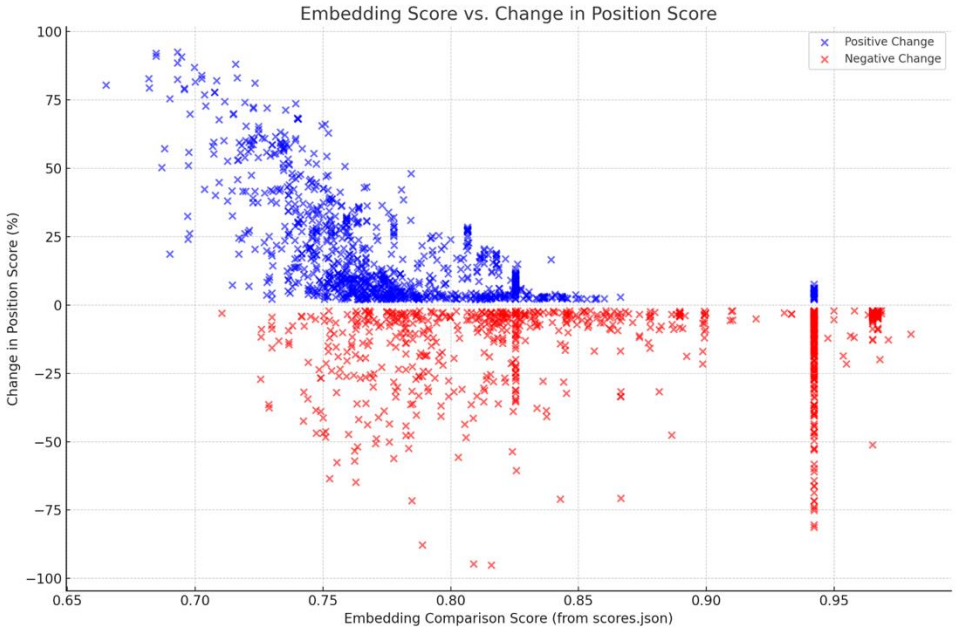

Figure 16. Embedding Score vs Change in Position Score.

Apart from these lines, the graph shows a fanning out as the embedding score decreases. This could indicate that, as column names become less correlated with the actual meaning of the column, the model's estimations of the column's meaning increase in variation. The graph also shows how the model's estimations are not always an improvement. Many generated descriptions were a regression, as compared to simply using the column name. However, the results of the previous experiments showed that the mean of the position scores is higher than the column name alone, meaning that, on average, the model is potentially more correct than wrong when adding additional context.

5.6. Impact on Related Work

To test if the generated descriptions add valuable context for use in existing models, the column descriptions were substituted for the column names in the StruBert model. The StruBert model utilizes the WikiTables dataset for its testing, which does not include column descriptions. As such, comparing the generated descriptions to identify accuracy compared to ground truth is

impossible. However, the goal is to determine if the added context improves search accuracy when retrieving relevant database tables to a user query. The Wikitables dataset includes additional context, which was utilized in prompts to generate the column descriptions.

Figure 17 shows the prompt provided to the model to generate descriptions for the Wikitables dataset. The model was provided with the table name, second table title, table caption, column IDs, schema, and sample table data. Then, batches of 10 columns were used.

```
lambda table_name, table_second_title, table_caption, column_chunk_ids_str, schema, sample_data: \
f"Please update descriptions for only columns with the following ids: {column_chunk_ids_str}.\n\
The table meta data, sample data, and schema are below\n\
Table name: {table_name}\n\
Table second title: {table_second_title}\n\
Table caption: {table_caption}\n\
Sample data rows:\n\
{sample_data}\n\
\n\
Schema:\n\
Column id, Column name, Data type\n\
{schema}"
```

Figure 17. WIKITables Prompt.

Once the column descriptions were obtained, they were concatenated to the column names utilized by StruBert. StruBert utilizes its own processing, therefore it's necessary to modify its data processing slightly to perform this action. StruBert requires a training process for its MiniBert model. For this work, it was trained on a single Nvidia Quadro GP100. The environment was reproduced utilizing the instructions given on the StruBert Github page and the suggested parameters for testing were used. StruBert utilizes NDCG@5, MAP, and MRR to evaluate results. It evaluates an average result based on predefined 5-fold datasets.

Table 10 shows the results with and without the generated descriptions. It shows that using column descriptions gives an improvement across all metrics and considerably improves the MRR. Note that each test was run 20 times, and these results were the top for each series of runs. To

further identify if the results are significant, a one-sided t-test was performed on the series of results, assuming equal variance.

Table 10. StruBert Mean Metric Scores.

	NDCG@5	MAP	MRR
Strubert with column names	0.6252	0.6195	0.6450
Strubert with column descriptions	0.6263	0.6229	0.6591

Table 11 shows that all metrics are statistically significant at a sample size of 20 and a statistical significance level of 0.05.

Table 11. StruBert P Value Metric Scores.

	NDCG@5	MAP	MRR
Comparison of result sets	0.01223	0.00002	0.00001

6. CONCLUSIONS AND FUTURE WORK

The exploration of the use of NLG models for the semantic enrichment of database columns identified several limitations and areas that are available for future research. Notably, the attempt to reproduce the StruBert model did not yield the reported scores, suggesting an area for further investigation. Although improvements were observed with the incorporation of column descriptions, these did not surpass the previously reported scores.

Additionally, there are opportunities to utilize larger models. This work utilizes both NLP and NLG models, both of which could be replaced with larger options. However, this work did not utilize larger NLG models such as GPT4, LLAMA2 63B, or PALM 1.5, which could show improved description generation. In addition, one potential limitation could be the limited size of the NLP model, BERT. If the NLP model was replaced with a larger model, it could shrink the discrepancies in scores observed in this work.

Finally, utilizing NLG models to directly determine the relevance of database tables to queries presents a promising direction. Traditional reliance on embeddings could be replaced with NLG's dynamic response generation, potentially improving accuracy and reducing computational demands as models evolve to handle larger context windows efficiently.

In conclusion, this work examined the potential of NLG models to generate database column descriptions and assessed the impact of various models, configurations, prompts, and context inclusions on the accuracy of these descriptions. The application of these generated descriptions in place of column names within the state-of-the-art StruBert model demonstrated that the generated descriptions provide additional valuable context that can be utilized to improve tabular data search retrieval.

REFERENCES

[1] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou, "Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation," 2023, arXiv:2308.15363.

[2] N. W. Paton, J. Chen, and Z. Wu, "Dataset Discovery and Exploration: A Survey," ACM Computing Surveys, vol. 56, no. 4, 2023, doi: 10.1145/3626521.

[3] Y. Suhara, J. Li, Y. Li, D. Zhang, Ç. Demiralp, C. Chen, and W.-C. Tan, "Annotating Columns with Pre-trained Language Models," in Proc. of the 2022 ACM SIGMOD Int. Conf. on Management of Data, Philadelphia, PA, USA, 2022, pp. 1493-1503, doi: 10.1145/3514221.3517906.

[4] S. Zhang and K. Balog, "Ad Hoc Table Retrieval using Semantic Similarity," in Proc. of the 2018 World Wide Web Conf. on World Wide Web - WWW '18, 2018, doi: 10.1145/3178876.3186067.

[5] M. Trabelsi, Z. Chen, S. Zhang, B. D. Davison, and J. Heflin, "StruBERT: Structure-aware BERT for Table Search and Matching," in Proc. of the ACM Web Conf. 2022, Virtual Event, Lyon, France, 2022, pp. 442-451, doi: 10.1145/3485447.3511972.

[6] Z. Chen, M. Trabelsi, J. Heflin, Y. Xu, and B. D. Davison, "Table Search Using a Deep Contextualized Language Model," in Proc. 43rd Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2020, pp. 589-598, doi: 10.1145/3397271.3401044.

[7] C. Koutras, G. Siachamis, A. Ionescu, K. Psarakis, J. Brons, M. Fragkoulis, C. Lofi, A. Bonifati, and A. Katsifodimos, "Valentine: Evaluating Matching Techniques for Dataset Discovery," in 2021 IEEE 37th Int. Conf. on Data Engineering (ICDE), 2021, pp. 468-479, doi: 10.1109/ICDE51399.2021.00047.

[8] G. Fan, J. Wang, Y. Li, D. Zhang, and R. Miller, "Semantics-aware Dataset Discovery from Data Lakes with Contextualized Column-based Representation Learning," 2023, arXiv:2210.01922.

[9] U. Naseem, I. Razzak, S. K. Khan, and M. Prasad, "A Comprehensive Survey on Word Representation Models: From Classical to State-Of-The-Art Word Representation Language Models," 2020, arXiv:2010.15036.

[10] J. Lin, R. Nogueira, and A. Yates, "Pretrained Transformers for Text Ranking: BERT and Beyond," 2021, arXiv:2010.06467.

[11] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," 2013, arXiv:1301.3781.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," 2023, arXiv:1706.03762.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proc. 2019 Conf. of the North American Chapter of the Assoc. for Computational Linguistics: Human Language Technologies, vol. 1, Minneapolis, Minnesota, 2019, pp. 4171-4186, doi: 10.18653/v1/N19-1423.

[14] A. Radford and K. Narasimhan, "Improving Language Understanding by Generative Pre-Training," 2018, Available: https://api.semanticscholar.org/CorpusID:49313245.

[15] P. Yin, G. Neubig, W.-t. Yih, and S. Riedel, "TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data," 2020, arXiv:2005.08314.

[16] D. Valentino, E. Akhmetkhanov, and TheTypoMaster, "Millennium Business Suite Anywhere (MBSA): An open-source ERP/CRM platform," 2017, Available: https://github.com/vporoxnenko/mbsa.

[17] L. Reynolds and K. McDonell, "Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm," in Extended Abstracts of the 2021 CHI Conf. on Human Factors in Computing Systems, 2021, Art. no. 314, doi: 10.1145/3411763.3451760.