NL2FORMULA: Generating Spreadsheet Formulas from Natural Language Queries

Wei Zhao[1*] **Zhitao Hou**[2] **Siyuan Wu**[1] **Yan Gao**[2] **Haoyu Dong**[2] **Yao Wan**[1†]
Hongyu Zhang[3] **Yulei Sui**[4] **Haidong Zhang**[2]
[1]Huazhong University of Science and Technology, [2]Microsoft
[3]Chongqing University, [4]University of New South Wales
{mzhaowei,sy_wu022,wanyao}@hust.edu.cn, hyzhang@cqu.edu.cn
{zhith,yan.gao,hadong,haizhang}@microsoft.com, y.sui@unsw.edu.au

Abstract

Writing formulas on spreadsheets, such as Microsoft Excel and Google Sheets, is a widespread practice among users performing data analysis. However, crafting formulas on spreadsheets remains a tedious and error-prone task for many end-users, particularly when dealing with complex operations. To alleviate the burden associated with writing spreadsheet formulas, this paper introduces a novel benchmark task called NL2FORMULA, with the aim to generate executable formulas that are grounded on a spreadsheet table, given a Natural Language (NL) query as input. To accomplish this, we construct a comprehensive dataset consisting of 70,799 paired NL queries and corresponding spreadsheet formulas, covering 21,670 tables and 37 types of formula functions. We realize the NL2FORMULA task by providing a sequence-to-sequence baseline implementation called fCODER. Experimental results validate the effectiveness of fCODER, demonstrating its superior performance compared to the baseline models. Furthermore, we also compare fCODER with an initial GPT-3.5 model (i.e., `text-davinci-003`). Lastly, through in-depth error analysis, we identify potential challenges in the NL2FORMULA task and advocate for further investigation.[1]

1 Introduction

It is a widespread practice among users to engage in data analysis by composing formulas within spreadsheet applications such as Microsoft Excel and Google Sheets. While spreadsheet formula languages (e.g., Microsoft Excel Formula) are relatively simpler than general-purpose programming languages for data analysis, formulating these formulas on spreadsheets remains burdensome and error-prone for end-users (Gulwani, 2011; Cheung et al., 2016). To address this challenge, numerous approaches and tools (e.g., FlashFill (Gulwani, 2011) and SPREADSHEETCODER (Chen et al., 2021)) have been proposed to automatically generate spreadsheet formulas.

Building upon substantial progress in spreadsheet formula generation, this paper goes beyond the existing efforts by introducing a novel Natural Language (NL) interface capable of generating spreadsheet formulas from a user's NL query (short for NL2FORMULA). We believe that, for the majority of end-users, expressing their intentions in NL is more accessible than working with formulas when performing data analytics on spreadsheets.

Figure 1 presents two representative running examples to illustrate the task of NL2FORMULA. This task involves generating the corresponding spreadsheet formula automatically, given a spreadsheet table and an NL query input from an end-user. The resulting formula is intended for execution in spreadsheet applications, such as Microsoft Excel. In this paper, we focus the spreadsheet application only on Microsoft Excel, where spreadsheet formulas can take on various forms, offering a wide range of possibilities for exploration. Specifically, we present two primary categories of spreadsheet formulas. The first category is the `Analysis Query` (Figure 1 (a)), typically comprising Excel formula functions utilized for data analysis. The second category is the `Calculation` (Figure 1 (b)), consisting of basic numerical operations used for straightforward calculations.

It is important to note that NL2FORMULA shares similarities with the well-studied task of TEXT2SQL, which involves translating an NL description into a SQL query grounded on a database table (Yaghmazadeh et al., 2017; Yu et al., 2018; Zhong et al., 2017). However, it differs in two

[*] Work was done while Wei Zhao was pursuing a master degree at Huazhong University of Science and Technology, and during an internship at Microsoft.

[†] Yao Wan is the corresponding author.

[1]All the experimental data and source code used in this paper are available at https://github.com/timetub/NL2Formula.

Findings of the Association for Computational Linguistics: EACL 2024, pages 2377–2388
March 17-22, 2024

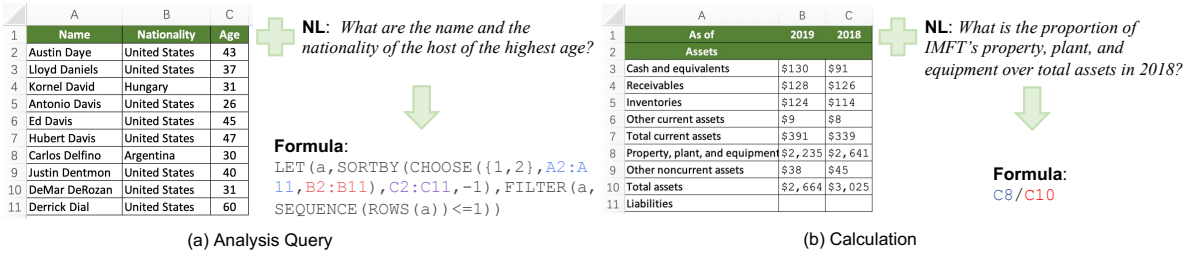

Figure 1: Two running examples from our created dataset for NL2FORMULA.

fundamental aspects. *(1) The structure of a spreadsheet table is more flexible than that of a database table.* Unlike fixed patterns in databases, the metadata (e.g., headers and orientation) of tables in a spreadsheet is optional, and the placement of the table in the layout is highly flexible. This flexibility presents significant challenges when it comes to representing the data. *(2) The formula is typically expressed by the index of data location.* In the process of generating formulas, it becomes crucial not only to determine which columns in the table should be selected but also to identify the exact position of the cell containing these values. Additionally, the expression of formulas can change with the placement of the table in the layout.

In this paper, we pioneer the effort to formulate and benchmark the task of NL2FORMULA. One main challenge lies in the lack of well-labeled data for training. To tackle this issue, we construct a novel dataset comprising paired NL queries and their corresponding formulas, grounded on specific spreadsheet tables. As manual labeling would require extensive human effort and time, we opt for an indirect transformation approach using an existing dataset of TEXT2SQL (i.e., Spider (Yu et al., 2018)), which is composed of 10,181 NL descriptions along with their corresponding SQL queries. We devise a set of conversion rules by analyzing the grammar of SQL and Excel formulas. By applying the formulated conversion rules, we convert SQL queries from the established TEXT2SQL datasets into formulas suitable for NL2FORMULA. Additionally, to augment the dataset, we engage in the manual collection of labeled data following a set of predefined rules. As a result, we produce a comprehensive dataset comprising 70,799 paired NL queries and formulas, associated with a total of 21,670 tables.

Furthermore, we establish a benchmark for NL2FORMULA. In this benchmark, we also present *f*CODER, a sequence-to-sequence framework based on the pre-trained language model T5 (Raffel et al., 2020). As a baseline model, we adapt FORTAP (Cheng et al., 2021), originally designed for synthesizing spreadsheet formulas, for comparison. We conduct comprehensive experiments and analysis to assess the effectiveness of our proposed *f*CODER. The experimental results demonstrate that *f*CODER achieves the highest performance with 70.6% *Exact Matching Accuracy* and 77.1% *Accuracy* based on the results of running formulas on a specific engine (i.e., Microsoft Excel). After conducting a comprehensive analysis of the experimental results, we have identified potential areas for improvement and future directions that warrant further exploration.

In summary, the key contributions of this paper are three-fold. (1) We are the first to formulate a new task of NL2FORMULA, that can serve as an interface allowing users to effortlessly translate input NL queries into spreadsheet formulas. (2) We introduce a novel dataset that comprises 70,799 paired NL queries and their corresponding formulas, associated with 21,670 tables. (3) We benchmark several models for the task of NL2FORMULA, including our designed *f*CODER that is based on pre-trained T5, as well as FORTAP (Cheng et al., 2021) that is adapted from TUTA (Wang et al., 2021).

2 Background and The Problem

Spreadsheet Formula. Spreadsheets, which are formulated as a two-dimensional grid of cells, play a vital role in our daily lives, especially for data analysis. Typically in a spreadsheet, rows are numbered sequentially from top to bottom, beginning at 1, while columns are designated alphabetically from left to right using the base-26 system, with 'A' to 'Z' as the digits.

We can perform various computing, data processing, and operational tasks using pre-defined formulas within the spreadsheet. In a formula, we can refer to a cell by combining its column and row

numbers, as shown by the notation (e.g., `B2`). Additionally, we have the option to use a range operator "`:`" to create a rectangular range between two cells, with the top-left and bottom-right corners specified. For instance, the formula `=SUM(A1:B5)` encompasses all cells in columns `A` and `B`, ranging from row 1 to row 5. In general, a formula is composed of constant values, arithmetic operations, function calls, and references to cells. Formally, the Microsoft Excel formula studied in this paper can be defined by the extended BNF grammar, referred to Appendix A. Figure 2 shows a detailed example of the Excel formula.

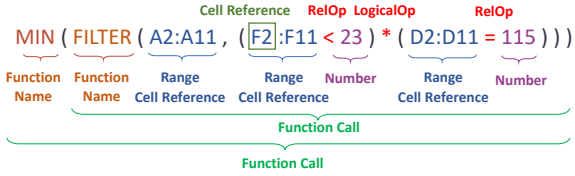

Figure 2: An example of the Excel formula.

Problem Statement. Let N denote the NL query composed of a sequence of tokens $\{q_1, q_2, \ldots, q_L\}$, and T denote the corresponding tabular context composed of a collection of cells $\{c_1, c_2, \ldots, c_M\}$. Let F denote the corresponding formula to predict that is denoted a sequence of tokens $\{y_1, y_2, \ldots, y_K\}$. Inspired by previous semantic parsing tasks, we formulate the task of NL2FORMULA as a sequence-to-sequence problem, where the source sequence is the NL query and its tabular contexts, while the target sequence is the formula. More specifically, the NL2FORMULA problem is expressed as follows: given a source NL sequence N, as well as the tabular context T, the goal is to learn a mapping function f to map the input $\{N, T\}$ into a formula F, i.e., $F = f_\theta(N; T)$, where θ is the parameters of model f.

3 NL2FORMULA: The Dataset

3.1 Dataset Construction

Constructing a paired dataset of NL queries and spreadsheet formulas poses considerable challenges. One approach to tackle this is by inviting experts to generate corresponding NL queries and spreadsheet formulas based on the tabular content. However, this method is time-consuming and labor-intensive, demanding significant human effort. Hence, it drives us to explore alternative ways of indirectly creating the NL2FORMULA dataset.

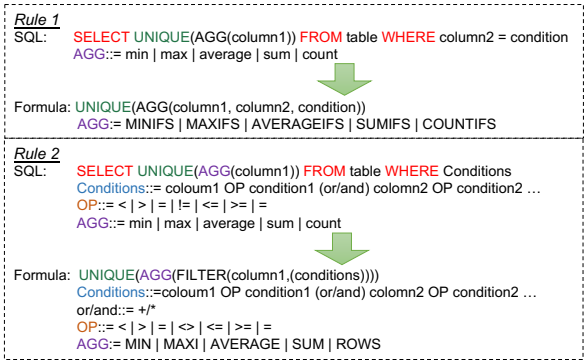

Figure 3: Two simple examples of conversion rules to translate SQL queries into formulas.

Fortunately, we discovered a related task called TEXT2SQL, which has already undergone extensive study. Leveraging this, we develop a converter from the TEXT2SQL dataset to the NL2FORMULA dataset. The underlying intuition is that both SQL queries and spreadsheet formulas specify the required data in a similar fashion.

Rule-Based SQL to Formula. By analyzing SQL grammar and Excel formula grammar, we manually define several conversion rules to convert the SQL queries into Excel formulas. For example, in certain conditions that necessitate single operations (e.g., `MAX`) in SQL, we can utilize the corresponding `MAXIFS` function in a spreadsheet formula. In more intricate scenarios involving multiple conditions and operators in SQL (e.g., `MIN` and `AND`), we can replace them with equivalent Excel formulas (e.g., `MIN` and `FILTER`). In situations requiring sorting and combination operations, we need to employ a combination of various Excel formula functions (e.g., `HSTACK`, `UNIQUE`, and `SORT`). We present two straightforward examples of conversion rules in Figure 3.

In practice, we primarily utilize two TEXT2SQL datasets: WikiSQL (Zhong et al., 2017) and Spider (Yu et al., 2018). WikiSQL is an extensive dataset consisting of 80,654 instances of paired NL queries and SQL queries, derived from 24,241 tables sourced from Wikipedia. This dataset exclusively comprises single tables and simple SQL queries. However, our objective is to create a more challenging dataset that encompasses a wider range of formula functions and categories. To achieve this, we integrate the Spider dataset, with the potential to enhance the diversity of formulas. Spider is a complex and cross-domain TEXT2SQL dataset annotated by 11 graduate students. It comprises

10,181 NL queries and 5,693 unique complex SQL queries derived from 200 databases containing multiple tables across 138 different domains. Due to the constraints posed by existing models regarding input data length, we select tables with 3 to 20 rows and 3 to 10 columns. As a result, we obtain approximately 19,789 candidate tables.

Data Augmentation. Based on our investigation, all the formulas converted from TEXT2SQL are analysis-oriented, commonly referred to as `Analysis Query`. In other words, these formulas predominantly consist of formula functions such as `AVERAGE` and `MAXIFS`. Notably, simple numerical operations such as addition ($+$), subtraction ($-$), multiplication ($\times$), and division ($/$) (also referred to as `Calculation`) are excluded from the converted formulas. To complement this, we manually augment the data by incorporating a question-answering benchmark named TAT-QA (Zhu et al., 2021), which includes numerous numerical operation formulas.

3.2 Data Statistics and Analysis

We finally obtain 70,799 pairs of NL queries and spreadsheet formulas, covering 21,670 tables. The tables are randomly split into a training set (75%), validation set (10%), and test set (15%). The basic statistics of each split are shown in Table 1. The length of a formula is defined by the number of its keywords. We can observe that the average formula length is about 10, indicating the difficulty in predicting these formulas.

To better comprehend the performance of models on various formulas, we categorize the formulas into two groups: `Analysis Query` and `Calculation`. In particular, `Analysis Query` formulas encompass 37 types of formula functions, while `Calculation` formulas consist of addition, subtraction, division, and composition. Moreover, for `Analysis Query`, we have tailored the division standards of hardness levels, which are classified into 3 categories: $Simple$, $Medium$, and $Complex$. Specifically, the division standard is based on the number of formula components, selections, and conditions. For instance, we define a formula as $Simple$ if it typically represents a short-length query with 1-2 functions, $Medium$ for 3-4 functions, and any formula with more than 4 functions is considered $Complex$ and falls into the long-length category.

Figure 4 depicts the hardness distribution of the dataset. It is evident that the majority of formulas consist of medium-level analysis queries, accounting for 49.1%.

Table 1: Statistics of the NL2FORMULA dataset.

Statistics	Train	Val.	Test
# of tabular contexts	16,791	1,743	3,136
# of NL queries	55,165	5,523	10,111
Avg. # of table rows	10.8	10.8	10.8
Avg. # of table columns	6.0	6.0	5.9
Avg. length of NL	11.2	11.6	11.4
Avg. length of formula	10.2	10.1	10.0

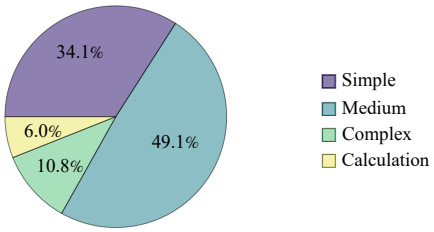

Figure 4: Distribution of formulas in NL2FORMULA dataset, including `Analysis Query` of three hardness levels ($Simple$, $Medium$, $Complex$), and `Calculation`.

3.3 Data Quality Assessment

To ensure the quality of our NL2FORMULA dataset, we follow a rigorous process. Initially, we randomly sample 5% of the original data and convert it from SQL queries to formula queries. Subsequently, we input these queries into a spreadsheet to assess their smooth execution. Based on the execution results, we make necessary adjustments to the conversion rules for formula queries that fail to execute successfully. To guarantee accuracy and reliability, we engage five verifiers with extensive experience in NLP and familiarity with spreadsheet formulas. Each verifier is tasked with checking and approving 500 pairs of NL queries and formula queries, randomly selected from the dataset. Their expertise ensures meticulous scrutiny of the data. Finally, in cases where we identify faulty formulas, we verify their formula patterns and search the dataset for all instances of such patterns, making the necessary modifications to rectify the situation.

4 fCODER : A Reference Framework

For the task of NL2FORMULA, we adopt the encoder-decoder paradigm as the baseline approach. In this paradigm, an encoder network embeds the NL queries and tabular contexts into an

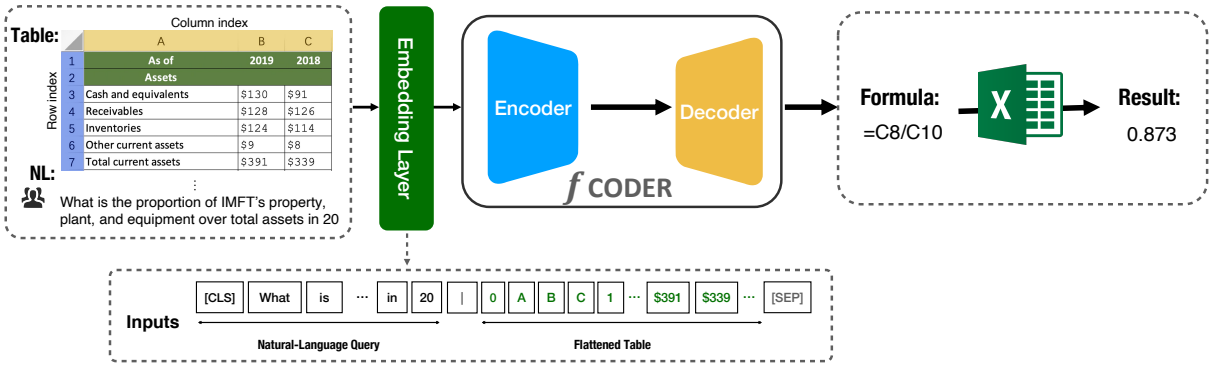

Figure 5: An overview of the fCODER, which is a reference framework for NL2FORMULA.

embedding vector, while a decoder network generates the formula based on the encoded vector. Figure 5 illustrates the overview of the encoder-decoder framework for NL2FORMULA.

Input Preparation. We represent each table using its column index, row index, and the corresponding content. Specifically, we work with two types of inputs: an NL query and tabular content. Each input is transformed into a sequence, and subsequently, the two sequences are concatenated. We employ a unique symbol | to differentiate between the sequence of NL queries and tabular content. Furthermore, we utilize a specific token `[CLS]` to mark the inception of the concatenated sequence, resulting in a hybrid representation of the two elements, as follows:

$$X = \texttt{[CLS]}, q_1, q_2, \ldots, q_L, \texttt{|}, c_1, c_2, \ldots, c_M \,.$$

For each token x_i in X, we begin by encoding it using a word embedding layer, resulting in the token embedding $\mathbf{x}_i^{token}$. Next, we incorporate a positional embedding to account for the position of each token, represented as $\mathbf{x}_i^{position}$. The ultimate embedding of each token for an input sample X is determined as follows:

$$\mathbf{x}_i = \text{Emb}(x_i) = \mathbf{x}_i^{token} + \mathbf{x}_i^{position} \,. \quad (1)$$

After processing each token as discussed above, the output sequence is represented by $\mathbf{X} = \text{Emb}(X)$, which serves as the input to the encoder network.

Encoder. We input the embedding matrix $\mathbf{X}$ into the encoder network, yielding the corresponding output $\mathbf{O}^e$ as follows:

$$\mathbf{O}^e = \text{Encoder}(\mathbf{X}) \,. \quad (2)$$

Finally, these output embeddings are passed as input to the decoders.

Decoder. At the t-th time step in the decoding process, the operations of the decoder network can be formulated as follows:

$$\mathbf{O}_t^d = \text{Decoder}(\mathbf{O}^e, \text{Emb}(ctx)) \,, \quad (3)$$

where $\mathbf{O}^d$ is the output of the decoder network, ctx denotes the current partial sequence of the generated formula, i.e., $y_0, \ldots, y_{t-1}$, which is also mapped into vector forms via an embedding layer.

We feed the output of the decoder into a Softmax layer, to map the output vector into a probability vector over the whole vocabulary, as follows:

$$p(y_t|ctx, \mathbf{X}) = \text{Softmax}(\mathbf{W}^d\mathbf{O}^d + \mathbf{b}^d) \,, \quad (4)$$

where $\mathbf{W}^d$ and $\mathbf{b}^d$ are the linear layer parameters.

Model Learning. To train the fCODER model, we employ the cross-entropy loss function, as follows:

$$\mathcal{L} = -\sum_{t=1}^{T} \log p_\theta(y_t|ctx, \mathbf{X}) \,, \quad (5)$$

where θ denotes all the model parameters, and T is the maximum step of formula generation.

5 Experimental Evaluation

5.1 Benchmarked Models

▷ **FORTAP (Cheng et al., 2021).** FORTAP, building on TUTA (Wang et al., 2021), extends table pre-training to include spreadsheet formulas for enhanced formula prediction, question answering, and cell type classification. We introduce an adaptation of FORTAP to NL2FORMULA, where the task is to predict formulas for a specified cell within a table. We embed the NL query into the table and designate the following row as the target cell. A two-stage LSTM (Hochreiter and Schmidhuber,

Table 2: Overall performance of the fCODER and baselines on the validation and test datasets, in terms of the EM and ERA metrics.

Models	Exact Match				Execution Result Assessment	
	Validation		Test		Validation	Test
	Sketch	Formula	Sketch	Formula	Formula	Formula
FORTAP	-	-	58.4	24.2	-	-
GPT3.5 (10-Shot)	-	-	-	21.4	-	25.2
fCODER-Small	97.0	65.6	96.9	65.5	71.2	70.4
fCODER-Base	97.4	70.5	97.2	69.4	73.3	75.0
fCODER-Large	97.5	71.5	97.6	70.6	76.8	77.1

1997) decoder then processes this integrated data to produce formula sketches and pinpoint reference cells, yielding the target formula.

▷ **GPT-3.5 (Brown et al., 2020).** With recent advancements in the domain of Large Language Models (LLMs), remarkable breakthroughs have been achieved in the field of NLP (Zhao et al., 2023; Kaddour et al., 2023). In this study, we compare the performance of our proposed methodology with GPT-3.5 on the NL2FORMULA dataset, utilizing the open-sourced `text-davinci-003` model. The prompt template used by GPT-3.5 is referred to Appendix B

▷ **fCODER.** We adopt the T5 model (Raffel et al., 2020) as the initial implementation of the fCODER framework. T5 converts all text-based language problems into a text-to-text format and serves as a typical sequence-to-sequence model. Some variants of the model are also included in this paper, namely fCODER-Small, fCODER-Base, and fCODER-Large, with parameter sizes of 60M, 220M, and 770M, respectively.

Additionally, we also perform a preliminary comparison between fCODER and ChatGPT (OpenAI) in the Appendix C.

5.2 Evaluation Metrics

Inspired by the evaluations in TEXT2SQL, we also employ two similar metrics: *Exact Match (EM)* and *Execution Results Assessment (ERA)*. Furthermore, we categorize the formulas into two main groups: `Analysis Query` and `Calculation`. Additionally, within the `Analysis Query` category, we further differentiate formulas into three levels, namely, *Simple*, *Medium*, and *Complex*, based on the number of functions they incorporate.

Exact Match (EM). The *Exact Match* is a widely recognized metric used to evaluate the performance of models. It demands a flawless match between the model's output formulas and standard formulas, encompassing all its components and table ranges. To provide a fine-grained analysis of the model's performance on different granularities of formulas, we present both the *Sketch EM* score and the *Formula EM* score across all models.

Execution Result Assessment (ERA). To assess the semantic equivalence of predicted formulas, we also compare their execution results in Microsoft Excel. To streamline this evaluation process, we have developed an automated Python script for large-scale batch execution.

5.3 Results and Analysis

Overall Performance We begin by analyzing and discussing the overall performance of various models, which includes the baseline FORTAP, GPT-3.5, and our proposed fCODER, on the NL2FORMULA task. Table 2 presents a comprehensive evaluation of these models on both the validation and test datasets, in terms of the EM (including *Sketch EM* and *Formula EM*) and ERA metrics.

From this table, we can observe a notable performance disparity between the baseline model FORTAP and our proposed fCODER models. The former achieves an EM accuracy of 24.2 on the test set, indicating its struggle to precisely match the ground truth answers. One possible reason is that FORTAP is not specifically designed for this task; instead, it focuses on the context of individual cells, neglecting to capture the connections between the entire table and the question. In contrast, the fCODER-Small model, despite having the smallest number of parameters, significantly outperforms FORTAP, achieving an impressive EM accuracy of 65.5 on the test dataset. These results demonstrate the ef-

Table 3: Experimental results of fCODER models across different types of formulas, with varying levels of difficulty on the test dataset.

Models	Exact Match				Execution Result Assessment			
	Simple	Medium	Complex	Calculation	Simple	Medium	Complex	Calculation
GPT3.5 (10-Shot)	8.5	25.8	0.3	55.8	17.4	26.6	0.6	59.5
fCODER-Small	39.9	73.9	54.5	62.2	58.6	82.7	56.3	64.8
fCODER-Base	44.5	76.9	53.4	71.8	63.0	87.4	56.0	74.5
fCODER-Large	45.4	76.0	58.4	76.5	64.5	88.7	61.6	79.5

fectiveness of fCODER in generating accurate formulas from tabular data.

Furthermore, we can observe that the GPT-3.5 model with a 10-shot in-context learning approach achieves an EM accuracy of 21.4 and an execution results accuracy of 25.2. GPT-3.5 model also falls short of matching the performance of the fCODER series models. This discrepancy could be attributed to the relative simplicity of the current prompt design. Due to the constraints of length of input tokens, we can only provide a prompt consisting of 10 examples at a time, which seems to be insufficient in quantity.

Performance on Varying Hardness We also evaluate the performance of models across both types of formulas, namely, `Analysis Query` and `Calculation`, encompassing varying levels of difficulty, as shown in Table 3. From this table, it is interesting to see that our fCODER models demonstrate lower performance in the *Simple* level compared to the *Medium* level, in terms of EM accuracy. Through our human inspection, we have determined that this phenomenon can be ascribed to the fact that the model has a tendency to generate diverse formula queries, primarily stemming from the ambiguity introduced by NL queries. Furthermore, it is evident that fCODER attains high performance in the ERA metric. This is attributed to the fCODER 's ability to generate diverse expressions while consistently yielding the correct result.

In comparing the performance of our model with GPT-3.5 utilizing a 10-shot context, it is evident that the GPT-3.5 model exhibits poor performance in generating formulas within the `Analysis Query` category, highlighting a considerable need for further enhancements. Nonetheless, it is intriguing to observe that the GPT-3.5 model demonstrates a comparable level of proficiency in generating formulas within the `Calculation` category.

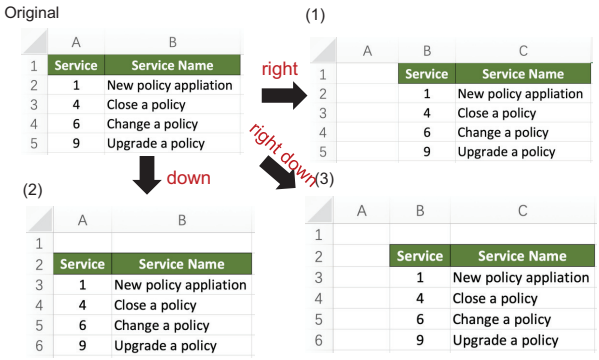

Figure 6: An example of a table as well as its three variants of movement in three different directions.

The Impact of Table Position. As previously mentioned, the spreadsheet table is flexible. Therefore, we further explore the performance of the model in generating formulas under different table placements. Specifically, the position of the original tables in our dataset starts from the first row and the column "`A`". We empirically move these tables in the following three ways, as shown in Figure 6: (1) Moving one column to the right, i.e., the starting position of tables is changed to "`B1`". (2) Moving one row down, i.e., the starting position of tables is changed to "`A2`". (3) Moving down and right, i.e., the starting position of tables is changed to "`B2`". In this scenario, the formulas will also be changed. For example, a formula in the original scenario, `SORTBY(B2:B5, B2:B5, 1)`, would be transformed to `SORTBY(C3:C6, C3:C6, 1)` in scenario (3). Initially, we use the fCODER-Base trained in the original position to verify the three scenarios. We explore whether the model can adapt to different table placements in spreadsheets, which were not seen during training. However, the performance of the model is poor, achieving only an average EM accuracy of 6.7%. We find that most of the errors are caused by the fact that our model fails to infer the cell index accurately.

5.4 Case Study and Error Analysis

Figure 7 presents an illustrative example of the prediction formula, which differs from the golden formula, yet yields identical results when executed in the spreadsheet. The table in `A1:J6` contains the NL description "*What is the lowest number of laps in the 5th position?*" provided in the 8th row. The given golden formula is `MINIFS(G2:G6, J2:J6, "5th")`, and the resulting value after executing this formula in Excel is "3", displayed in cell `A9`. On the other hand, the model prediction formula, `MIN(FILTER(G2:G6, J2:J6="5th"))`, produces the same result, which is demonstrated in cell `C9`.

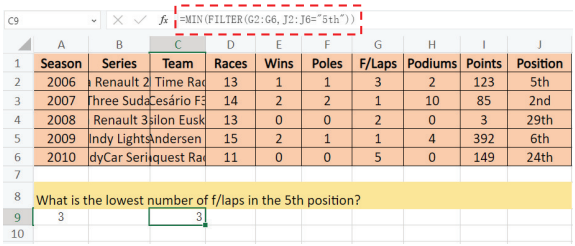

Figure 7: An example of the prediction formula, which is different from the ground-truth formula but the execution results in the spreadsheet are the same.

To gain a comprehensive insight into the effectiveness of our constructed model on NL2FORMULA, we conduct a detailed examination of the *f*CODER-Large, specifically focusing on instances where errors occur. We randomly sample 200 error instances from the test dataset (50 per level). We classify them into four categories, as shown in Figure 8: (1) Wrong Evidence: The model obtains incorrect supporting evidence or infers the wrong cell index from the table. Additionally, the example of the formula demonstrates the model's failure to identify the correct evidence from the NL query. (2) Missing Evidence: The model fails to extract complete supporting evidence from the table to arrive at the correct answer. (3) Wrong Intent Inference: The model is unsuccessful in understanding the intent expressed by the NL query. (4) Wrong Calculation: The model correctly infers the intention from the NL query and accurately locates the cell index in the table. However, the model fails to compute the answer using the correct evidence. We find that most of these errors stem from the model's inability to accurately infer or extract the correct evidence from the tables and NL queries.

6 Related Work

Semantic Parsing. Semantic parsing is a task to transform NL queries into structured representations that can be understood and processed by machines. So far, many datasets for semantic parsing have benn built with different query formats, such as ATIS (Price, 1990), Geo-Query (Zelle and Mooney, 1996), and JOBS (Tang and Mooney, 2001). Their output format is logic forms and has been studied extensively (Dong and Lapata, 2016; Berant and Liang, 2014; Reddy et al., 2014; Zettlemoyer and Collins, 2012; Wong and Mooney, 2007). In recent years, using SQL queries as programs in semantic parsing is more popular, and many datasets have been built, including Restaurants (Popescu et al., 2003), Academic (Li and Jagadish, 2014), Yelp and IMDB (Yaghmazadeh et al., 2017), Scholar (Iyer et al., 2017), WikiSQL (Zhong et al., 2017), Spider (Yu et al., 2018), and CoSQL (Yu et al., 2019).

Formula Synthesis. Formula synthesis is a branch of program synthesis that has been studied in many works. FlashFill (Gulwani, 2011; Gulwani et al., 2012) utilizes input-output examples to help end-users automatically synthesize string transformation tasks in spreadsheets. Recent studies have explored various neural architectures for learning programs from examples (Kalyan et al., 2018; Parisotto et al., 2017), but they do not consider context-specific information from spreadsheet tables. FORTAP (Cheng et al., 2021) and SPREEDSHEETCODER(Chen et al., 2021) are the prior approaches for synthesizing spreadsheet formulas from tabular context. Our work provides a standardized benchmark for evaluating and comparing future formula generation work, fostering advancement and understanding of the field.

Tabular Data Processing. Several studies have pretrained Transformers on tables. TABLEBERT (Chen et al., 2020) linearized tables as sentences so that tables can be directly processed by the pre-trained BERT model. TUTA (Wang et al., 2021) is the first effort to pre-train Transformers on variously structured tables. FORTAP (Cheng et al., 2021) use formulas for numerical-reasoning-aware table pre-training. To improve the representation of utterances and tables for neural semantic parsing, several works joined contextual representations of utterances and tables, such as TAPAS (Herzig et al., 2020) and TABERT (Yin

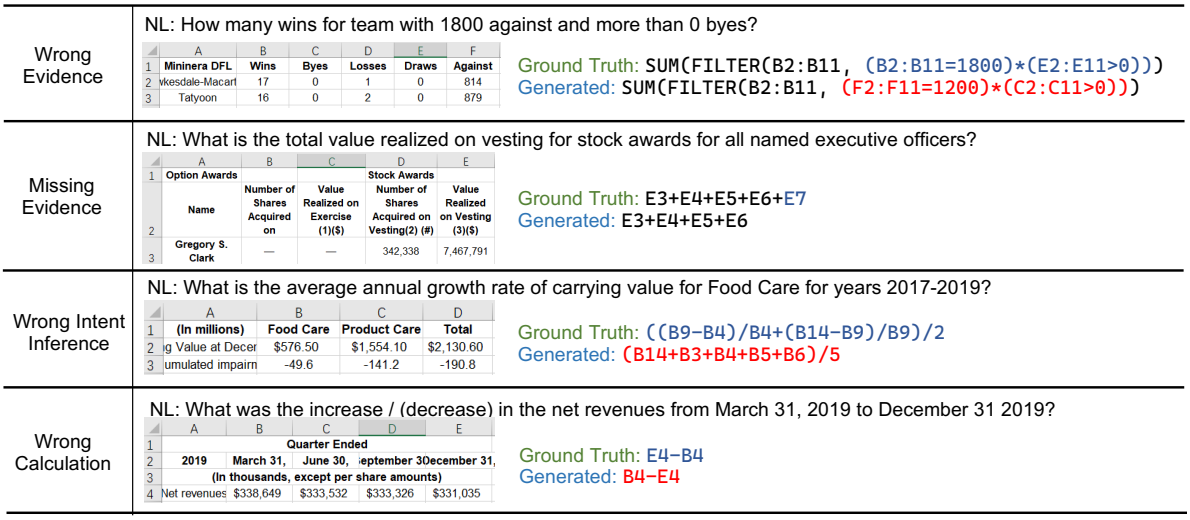

Figure 8: Case studies of error cases. (NL: Natural Language)

et al., 2020). Furthermore, Chen et al. (2021) introduced SPREADSHEETCODER, which leverages machine learning to assist in formula prediction in spreadsheets.

7 Conclusion

In this paper, we have presented a novel and challenging research problem, NL2FORMULA, and develop an accompanying dataset that includes spreadsheet tables, NL queries, and formulas. We construct a comprehensive dataset consisting of 70,799 paired NL queries and corresponding spreadsheet formulas, covering 21,670 tables and 37 types of formula functions. We also realize the NL2FORMULA task by providing a sequence-to-sequence baseline implementation called fCODER. Through in-depth error analysis, we identify potential challenges in the NL2FORMULA task and advocate for further investigation. We believe that the benchmark developed in this paper can prompote the related research in NL2FORMULA.

8 Limitations

There are several limitations of our research. One is that the formula queries in our NL2FORMULA dataset are converted from several TEXT2SQL datasets, resulting in a relatively fixed table structure. Additionally, while we made efforts to include as many formula functions and combinations as possible in our experiments, we have not yet fully covered all types of formula functions, such as the "FIND" function used for string queries. In our future work, we aim to expand the range of formula queries by incorporating additional formula functions, specifically targeting a broader array of scenarios. This expansion will include incorporating diverse data samples that utilize functions like "CONCATENATE", "LEN", and "REPLACE". These particular functions are essential for tasks related to data cleaning, preparation, and textual data manipulation. Moreover, we intend to explore the capabilities of models under multi-type tables, including horizontal and vertical tables, to simulate more realistic application scenarios. Furthermore, we aim to investigate situations involving multiple tables under the same spreadsheet.

Another limitation is the maximum length of model input, which is generally 512 characters. Despite controlling the length of rows and columns in the tables in this paper, we observed some errors caused by the model not fully encoding the table.

An additional potential limitation of our approach is the inability to directly execute custom-defined lambda functions in the current Excel environment. The DAX library, with its different grammar from Excel formulas, is used to build formulas and expressions in Excel data models like Power BI, Analysis Services, and Power Pivot. Consequently, we cannot use our execution result metric to measure the performance of custom-defined lambda functions. This limitation may impact the accuracy and comprehensiveness of our evaluation for this specific functionality.

Acknowledgements

This work is supported by the National Natural Science Foundation of China under grand No. 62102157.

References

J. Berant and P. Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Harrison Chase. 2022. LangChain.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2020. Tabfact: A large-scale dataset for table-based fact verification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Xinyun Chen, Petros Maniatis, Rishabh Singh, Charles Sutton, Hanjun Dai, Max Lin, and Denny Zhou. 2021. Spreadsheetcoder: Formula prediction from semi-structured context. In *International Conference on Machine Learning*, pages 1661–1672. PMLR.

Zhoujun Cheng, Haoyu Dong, Fan Cheng, Ran Jia, Pengfei Wu, Shi Han, and Dongmei Zhang. 2021. Fortap: Using formulae for numerical-reasoning-aware table pretraining. *In Proceedings of the Association for Computational Linguistics.*

Shing-Chi Cheung, Wanjun Chen, Yepang Liu, and Chang Xu. 2016. Custodes: automatic spreadsheet cell clustering and smell detection using strong and weak features. In *Proceedings of the 38th International Conference on Software Engineering*, pages 464–475.

L. Dong and M. Lapata. 2016. Language to logical form with neural attention. *Office for Official Publications of the European Communities,*.

Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices*, 46(1):317–330.

Sumit Gulwani, William R Harris, and Rishabh Singh. 2012. Spreadsheet data manipulation using examples. *Communications of the ACM*, 55(8):97–105.

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

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Srini Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Annual Meeting of the Association for Computational Linguistics*.

Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. 2023. Challenges and applications of large language models.

Ashwin Kalyan, Abhishek Mohta, Oleksandr Polozov, Dhruv Batra, Prateek Jain, and Sumit Gulwani. 2018. Neural-guided deductive search for real-time program synthesis from examples. *In ICLR*.

F. Li and H. V. Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proceedings of the Vldb Endowment*, 8(1):73–84.

OpenAI. ChatGPT plugins. https://openai.com/blog/chatgpt-plugins. 2023.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. 2017. Neuro-symbolic program synthesis. *In International Conference on Leaning Representations*.

A. M. Popescu, O. Etzioni, and H. Kautz. 2003. Towards a theory of natural language interfaces to databases. *International Conference on Intelligent User Interfaces*.

P. J. Price. 1990. Evaluation of spoken language systems: the atis domain. In *Proceedings of the third DARPA Speech and Natural Language Workshop*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67.

S. Reddy, M. Lapata, and M. Steedman. 2014. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics*, 2(1):377–392.

L. R. Tang and R. J. Mooney. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. *Springer, Berlin, Heidelberg*.

Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. 2021. Tuta: tree-based transformers for generally structured table pre-training. In *Proceedings of the 27th ACM SIGKDD*

Conference on Knowledge Discovery & Data Mining, pages 1780–1790.

Y. W. Wong and R. J. Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Acl, Meeting of the Association for Computational Linguistics, June, Prague, Czech Republic*.

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. Sqlizer: query synthesis from natural language. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):1–26.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Victoria Lin, Yi Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, and Dragomir Radev. 2019. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. pages 1962–1979.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *In Proceedings of EMNLP*.

J. M. Zelle and R. J. Mooney. 1996. Learning to parse database queries using inductive logic programming. *AAAI Press*.

L. S. Zettlemoyer and M. Collins. 2012. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Conference on Uncertainty in Artificial Intelligence*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A survey of large language models.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. TAT-QA: A question answering benchmark on a hybrid of tabular and textual content in finance. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3277–3287, Online. Association for Computational Linguistics.

A BNF Grammar of Formula

The extended BNF grammar of the Microsoft Excel formula studied in this paper is defined as follows:

⟨*Formula*⟩ ::= = ⟨*Expr*⟩

⟨*Expr*⟩ ::= ⟨*Term*⟩ { ⟨*AddOp*⟩ ⟨*Term*⟩ }

⟨*Term*⟩ ::= ⟨*Factor*⟩ { ⟨*MulOp*⟩ ⟨*Factor*⟩ }

⟨*Factor*⟩ ::= ⟨*Number*⟩ | ⟨*CellReference*⟩ | ⟨*FunctionCall*⟩ | (⟨*Expr*⟩)

⟨*CellReference*⟩ ::= ⟨*ColumnName*⟩ ⟨*RowNumber*⟩

⟨*ColumnName*⟩ ::= ⟨*Letter*⟩ { ⟨*Letter*⟩ }

⟨*RowNumber*⟩ ::= ⟨*Digit*⟩ { ⟨*Digit*⟩ }

⟨*FunctionCall*⟩ ::= ⟨*FunctionName*⟩ ([⟨*ArgumentList*⟩])

⟨*ArgumentList*⟩ ::= ⟨*Expr*⟩ { , ⟨*Expr*⟩ }

⟨*AddOp*⟩ ::= + | −

⟨*MulOp*⟩ ::= * | /

⟨*RelOp*⟩ ::= < | > | <= | >= | = | !=

⟨*LogicalOp*⟩ ::= + | *

⟨*FunctionName*⟩ ::= [a-zA-Z]+

⟨*Number*⟩ ::= ⟨*Integer*⟩ | ⟨*Decimal*⟩

⟨*Integer*⟩ ::= ⟨*Digit*⟩ { ⟨*Digit*⟩ }

⟨*Decimal*⟩ ::= ⟨*Integer*⟩ . ⟨*Digit*⟩ | . ⟨*Digit*⟩

⟨*Letter*⟩ ::= [a-zA-Z]

⟨*Digit*⟩ ::= [0-9]

B Prompt Template Used by GPT-3.5

We utilize a 10-shot in-context learning strategy, where for each new question and table, we dynamically select the Top-10 most similar NL-Formula pair examples from our training set. The similarity is determined based on their BLEU scores (Papineni et al., 2002). These selected examples, comprising 10 pairs of NL queries and formulas, are then integrated into a prompt to guide the model in generating its result. We use the following prompt template:

```
NL: [NL description]
Formula: [Excel Formula]
...(*10)
NL: [NL description]
Formula: [Excel Formula]
NL: [NL description]
Formula: [to be generated]
```

Table 4: Execution results of fCODER and ChatGPT, at different levels of hardness.

	Simple	Medium	Complex	Calculation	Overall
ChatGPT3.5-DirectQA	11.5	38.9	21.1	0.8	27.7
ChatGPT3.5-Agent	22.4	67.9	44.7	3.6	49.4
fCODER -Large	87.0	91.6	71.1	80.5	89.1

C Preliminary Comparison to ChatGPT

We explore the capabilities of ChatGPT for the task of NL2FORMULA. In addition to prompting LLMs to generate formulas (see Sect. 5), we also explore alternative approaches utilizing LLMs for the processing of tabular data. We leverage Langchain (Chase, 2022), a framework purposefully crafted to harness the potential of LLMs in the realm of application development. We investigate ChatGPT through two distinct approaches: (1) Direct Question-Answering (Direct-QA): We input the complete flattened table directly into the LLMs, prompting it to provide a direct answer to the NL query without any intermediate processing. (2) Langchain-Agent (Agent): We employ the Langchain CSVAgent workflow, which entails the transformation of the original spreadsheet into a Pandas data frame and the generation of Python code to extract or manipulate data to respond to the NL query.

We comprehensively evaluate ChatGPT's ability to handle tabular information and respond to NL queries. We randomly select 3,000 samples from the test dataset, which exclusively feature built-in Excel functions and exclude custom-defined lambda functions. Table 4 shows the evaluation results on the NL2FORMULA dataset. From this table, we can observe that ChatGPT exhibits moderate proficiency in processing spreadsheet data. They also unveil limitations in performing basic numerical operations within the `Calculation` subset, due to their constrained arithmetic and complex reasoning capabilities. Interestingly, the utilization of ChatGPT with Langchain CSVAgents exhibits notably superior performance when compared to the Direct-QA method. This is because the Langchain agent generates Python code for manipulating Dataframes, which closely aligns with the current *Code Interpreter* in handling tabular data.