

TABULARIS FORMATUS: Predictive Formatting for Tables

Mukul Singh

Microsoft

Redmond, USA

singhmukul@microsoft.com

José Cambronero Sánchez

Microsoft

Washington DC, USA

josepablocam@gmail.com

Sumit Gulwani

Microsoft

Redmond, USA

sumitg@microsoft.com

Vu Le

Microsoft

Redmond, USA

levu@microsoft.com

Gust Verbruggen

Microsoft

Keerbergen, Belgium

gverbruggen@microsoft.com

Abstract

Spreadsheet manipulation software are widely used for data management and analysis of tabular data, yet the creation of conditional formatting (CF) rules remains a complex task requiring technical knowledge and experience with specific platforms. In this paper we present TAFO, a neuro-symbolic approach to generating CF suggestions for tables, addressing common challenges such as user unawareness, difficulty in rule creation, and inadequate user interfaces. TAFO takes inspiration from component based synthesis systems and extends them with semantic knowledge of language models and a diversity preserving rule ranking. Unlike previous methods focused on structural formatting, TAFO uniquely incorporates value-based formatting, automatically learning both the rule trigger and the associated visual formatting properties for CF rules. By removing the dependency on user specification used by existing techniques in the form of formatted examples or natural language instruction, TAFO makes formatting completely predictive and automated for the user. To evaluate TAFO, we use a corpus of 1.8 Million public workbooks with CF and manual formatting. We compare TAFO against a diverse set of symbolic and neural systems designed for or adapted for the task of table formatting. Our results show that TAFO generates more accurate, diverse and complete formatting suggestions than current systems and outperforms these by 15.6%–26.5% on matching user added ground truth rules in tables.

CCS Concepts

- Do Not Use This Code → Generate the Correct Terms for Your Paper:** *Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.*

Keywords

Do, Not, Us, This, Code, Put, the, Correct, Terms, for, Your, Paper

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference acronym 'XX, Woodstock, NY

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-XXXX-X/2018/06

<https://doi.org/XXXXXXX.XXXXXXX>

ACM Reference Format:

Mukul Singh, José Cambronero Sánchez, Sumit Gulwani, Vu Le, and Gust Verbruggen. 2018. TABULARIS FORMATUS: Predictive Formatting for Tables. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 14 pages. <https://doi.org/XXXXXXX.XXXXXXX>

1 Introduction

Millions of users perform their data management and analysis in spreadsheet software [26]. To facilitate analysis and improve presentation, spreadsheet platforms typically allow users to write small, data-dependent formatting rules—also called *conditional formatting (CF) rules*—that can help visualize their data and identify important takeaways. These rules are often based on data present in the table. Figure 1 shows an example CF rule added by a user to the table.

Previous work introduced the tasks of automatically learning conditional formatting rules from examples [40] and natural language + examples [36]. These systems while useful for automating tabular formatting, still impose two challenges for users: knowing *what* to format—which requires understanding the data—and knowing *how* to format it—which requires understanding the semantics associated with certain visual properties. For example, the rule shown in Figure 1 applies a green fill color to Project ID cells where Budget is under Cost by over 1000. In such a case, the green color fill clearly carries semantics (positive). Other cases, such as coloring losses or failure rates with red or making the maximum value in a column bold, represent more common examples of formatting-based semantics. Further, these systems rely on accurate user intents and [36] shows users struggle with providing complete and accurate specifications and the system has to ask for clarification.

In this paper, we go one step further in helping users with formatting their data. We introduce TABULARIS FORMATUS or TAFO, which predictively suggests conditional formatting rules without the need for communicating any intent. Given only a table and a target column, TAFO automatically suggests relevant conditional formatting rules and the associated formatting properties.

Predicting conditions and their formatting properties raises three main challenges: (1) suggesting contextually relevant rules requires knowledge about the semantics of the data; (2) generated suggestions need to cover a diverse set of operations and executions (3) the learned rule formats need to be consistent with the data semantics and need to align with existing formatting in the table.

1 User Table

| Project ID | Cost | Budget | Status |
|------------|--------|--------|------------|
| A123 | 6012.4 | 9567.2 | Incomplete |
| A124 | 6584.7 | 9383.2 | Incomplete |
| A125 | 7037 | 8955.9 | Incomplete |
| B123 | 1262.8 | 3979.1 | Incomplete |
| A126 | 2593.9 | 4695 | Incomplete |
| A127 | 4268 | 5268.3 | Incomplete |
| A128 | 315.8 | 4465.5 | Incomplete |
| B124 | 9624.5 | 9947.7 | Complete |
| A129 | 7389.1 | 9662 | Complete |
| A130 | 6631.8 | 7569.7 | Complete |
| A131 | 9567.2 | 9012.4 | Complete |
| B125 | 383.2 | 2584.7 | Complete |
| A132 | 2955.9 | 3037 | Incomplete |
| A133 | 1979.1 | 2262.8 | Incomplete |
| A134 | 3695 | 4593.9 | Incomplete |
| A123 | 268.3 | 1268 | Complete |
| A124 | 4465.5 | 5315.8 | Complete |
| A125 | 5947.7 | 7624.5 | Incomplete |

2 User Added Formatting Rule

$[@Budget] - [@Cost] > 1000 \rightarrow \text{Fill("Red")}$

Figure 1: Sample data based formatting rule added by the user. (1) The user table in which the user wants to format the *Project ID* column (highlighted in yellow). (2) The formatting rule added by the user to highlight *Project ID* cells which are under budget (budget – cost) by over 1000, in red color.

TAFO generates natural rules by combining a purely symbolic generator, a purely neural generator, and a neuro-symbolic generator. The purely symbolic generator enumerates rules by constructing predicates and combining them into rules with beam search and a trained heuristic. The purely neural generator consists of an LLM Chain-of-Thought [49] prompt, which incorporates basic table properties (like most common values and fraction of duplicates). The neuro-symbolic learner leverages both symbolic and neural reasoning by extracting building blocks from the neural generator and using these in the predicate combination step with a higher weight. To learn the formatting properties, TAFO mines a corpus for similar conditional formatting rules applied on columns and aligns them with the current sheet.

For training and evaluating TAFO, we use a corpus of 1.8 Million spreadsheet [40] containing tables with CF rules and manual formatting. We found that TAFO can automate over 50% of user formatting tasks for both CF and manually formatted tables with just 3 suggestions per task. We also compare TAFO to current automatic table formatting systems [36, 40] as well as extend state-of-the-art language and table models to the task of CF suggestions. We find that TAFO consistently outperforms all baselines and has 15.6%–26.5% higher execution match accuracy on our benchmark.

In this work we make the following contributions:

- (1) We introduce the task of predictive conditional formatting suggestions, consisting of both the rule condition and its formatting properties.

- (2) We introduce TAFO, a neuro-symbolic system that provides conditional formatting suggestions.
- (3) We evaluate TAFO extensively on data from 1.8 Million spreadsheets, comparing it to symbolic and neural baselines, and show that TAFO outperforms these by
- (4) We analyze the suggestions generated by TAFO for completeness, diversity, coverage and complexity.

2 Related Work

Early work on rule based table formatting in spreadsheets has been relatively limited despite the large userbase of spreadsheet manipulation software. [29] explains the functionality and usage of conditional formatting in Excel. [1] discusses rule based formatting and its application for demonstrating mathematical concepts.

Recent progress in CF has seen development of automated tools that learn CF rules based on user specification. The specification can be in the form of either examples [40] or natural language instruction [47] or a mixture of both [36]. This line of work focuses on learning the CF rule specification through user supervision via multiple modalities. Unlike this, TAFO focuses on learning these rules purely predictively and does not require any supervision.

Another line of work on tabular formatting tackles automatic table formatting work which does not use data dependent rules. These include systems that rely on structural information from the table to learn formatting. [5] proposes CellGAN, a conditional Generative Adversarial Network model which learns the hierarchical headers and data groups in tables and can format these to visualize the data segmentation and improve presentation. Similarly, [14, 21, 33] focus on formatting cells based on table segmentation and cell classification. These systems solely rely on the structure of the table and are targeted towards improving data presentation, they do not generate data dependent rules which are often essential to understand and bring out insights from the data. In contrast, TAFO learns data dependent rules for the tables without supervision.

TAFO uses static data analysis to generate symbolic insights about the data which guide the component based synthesis and also assist the LLM in its reasoning. Such analysis has previously been used by [50] for learning data transformations. Techniques like this are also popular in other tabular tasks and have made their way into commercial spreadsheet software through systems like FlashFill [10] and FlashExtract [17]. These systems, which are available in Excel, learn string transformation and data extraction programs using static analysis over string properties.

TAFO is a neurosymbolic system. It leverages language models to extract semantic knowledge relevant to the table to be used for symbolic CF synthesis. Previous neurosymbolic work have used LLMs as part of the generation pipeline [36], for repair [15] for ranking [13]. All of these systems either use the symbolic reasoning to feed into the LLM or use the LLM to provide insights for the symbolic engine. Unlike these, TAFO is the first system where the symbolic reasoning is used by the LLM and the LLM reasoning is used for guiding the component based synthesis of rules.

Past work on generating predictive suggestions on databases have shown great success in the domain of querying [7, 18, 25, 38] and data understanding and cleaning [8]. TAFO builds upon these systems to solve the problem of table data formatting. In

terms of ranking, previous work has explored program analysis [6, 42], rule execution [27], diversity [43] or semantic relevance [13]. In this work, TAFO combines all of these to design a ranker that can generate relevant and diverse suggestions that are valid and executable.

Language models have revolutionized code generation [16, 19, 22, 30, 35, 37, 39, 45, 47] and tabular analysis [11, 44, 52]. Specialized systems like [4] and [15] are trained over millions of spreadsheets to generate formulas and other related artifacts. Models like TaBERT [52] and TAPAS [11] are popular Question Answering systems that use a neural model to encode the table and query. TUTA [48] is a weakly supervised model for cell and table type classification tasks. Unlike these systems, TAFO uses a neurosymbolic technique that combines static data analysis and semantic information from language models for the task of learning formatting rules.

3 Problem Definition

Let $E = [e_i^j]_{i=1 \rightarrow n}^{j=1 \rightarrow m}$ be a table with n columns and m rows. We will write E_i and E^j to denote column i and row j , respectively. Each column E_i is annotated with a column name (or header) H_i . A cell e is defined as a triplet (v, t, f) of its value $v \in \mathcal{V}$, its annotated type $t \in \mathcal{T}$ and a mapping $f : F \rightarrow \mathcal{F}_F$ of formatting identifiers F to one of their allowed values \mathcal{F}_F . We call f the format of cell e , and write $v(e)$, $t(e)$ and $f(e)$ to respectively denote its value, type and format. We consider text, numeric and date as types, since these are the most common types used in spreadsheets [40]. We consider fill color, font color, bold, italics, and underline as formatting identifiers, as these are the most popular [41]. $f_\perp = \emptyset$ denotes no formatting.

EXAMPLE 1. In Figure 1, the first cell in the “Status” column is represented as $e_4^1 = ("Incomplete", \text{text}, f_\perp)$ as it is not formatted. We write $v(e_4^1) = "Incomplete"$.

We define a condition C as a function $(\mathcal{V} \times \mathcal{T})^n \rightarrow \mathbb{B}$ that verifies some criterion on a row of n cell values and types. To ease notation in conditions, we will write $[@H_i]$ to refer to the implicit value or type of the cell in column i of the given row.

EXAMPLE 2. In Figure 1, the condition

$$(v(R_3) - v(R_2))$$

is written as

$$(@Budget) - (@Cost)$$

because $H_2 = Cost$ and $H_3 = Budget$ and the Subtract (-) function is implicitly defined on the value of a cell.

A conditional formatting rule $R = (C, f)$ associates a condition with a format f to apply to cells that satisfy the condition in a target column j . Given a table E and a target column j , the goal is then to suggest a list of *desirable* conditional formatting rules. Section 5.2 describes the metrics that we use to measure whether suggestions are desirable with respect to ground truth conditional formatting rules mined from a large corpus of spreadsheets.

4 Methodology

TAFO first learns multiple conditions C from (E, j) by generating and ranking candidate conditions, and then learns the format f from each (E, j, C) . We propose three different candidate generation strategies: purely symbolic, purely neural based on large

language models, and a neuro-symbolic combination of them with bi-directional feedback. An overview of this approach is shown in Figure 2. The following five sections respectively describe the three candidate generation strategies, ranking, and format learning.

4.1 Learning conditions: symbolic

On a high level, the symbolic generator first extracts signals in the form of static properties from the columns in the table, uses these properties to enumerate predicates, and then combines these predicates into complex conditions with beam search and a trained heuristic. An overview is shown in Figure 3 and the following paragraphs describe the three main steps.

Extracting properties. One way to generate properties is to symbolically construct them from base symbols. We use a set of templates for this construction curated specific to formatting tasks, taking inspiration from other table tasks like cleaning [38], transformation [36] and visualization [24]. We compute properties over columns that are useful to reason about the formatting condition. Table 1 summarizes the supported properties considered by TAFO. Properties like NumBlanks and NumFormulas counting the occurrence of a specific condition. Properties like Average and Median compute statistical properties. Properties like Categorical and Free-Text are heuristically computed to determine if the column contains categorical data and free text respectively.

One limitation of symbolically extracting properties is that the system has no semantic knowledge. Because of this lack of semantic knowledge, it does not perform arithmetic columns to compute aggregate constants—training the ranker to understand which operations make sense based on only column names would require a huge amount of data. For example, in Figure 2, the symbolic learner would never obtain $[@Budget] - [@Cost]$ to reason about profits or losses. LLMs, on the other hand, have been shown to capture much semantic knowledge about tables—they are increasingly used to suggest insights or transformations to users [12, 43].

To leverage LLMs for generating conditions, we follow a multi-step reasoning approach, which is outlined in Figure 4. We use the column properties as table context in the prompt, along with headers, type info and sample values. The model is instructed to generate the suggestions in four steps. First, it identifies relevant columns it that should be formatted in the table. Second, it generates useful predicates and functions that would be used in generating suggestions. Third, it lists important constants that are relevant to the predicates and table. Finally, using the previous steps, it generates a list of formatting rule suggestions. To enforce the reasoning path, we provide three static examples in the prompt from the training corpus, which were manually annotated for the reasoning steps. A sample of the prompt template is shown in Figure 5 and a sample of the full prompt is in the Appendix.

The generated rules are symbolically broken to generate base properties in the form of column names and constants and combined with properties and constants generated in the reasoning steps previously. This pooled set of properties is added back to the symbolically extracted properties. Further, the neural generations also yields predicates (for example, $NOT(BLANKS([@COLUMN]))$) which are components constructed over properties (BLANKS, COLUMN) for better semantic knowledge distillation.

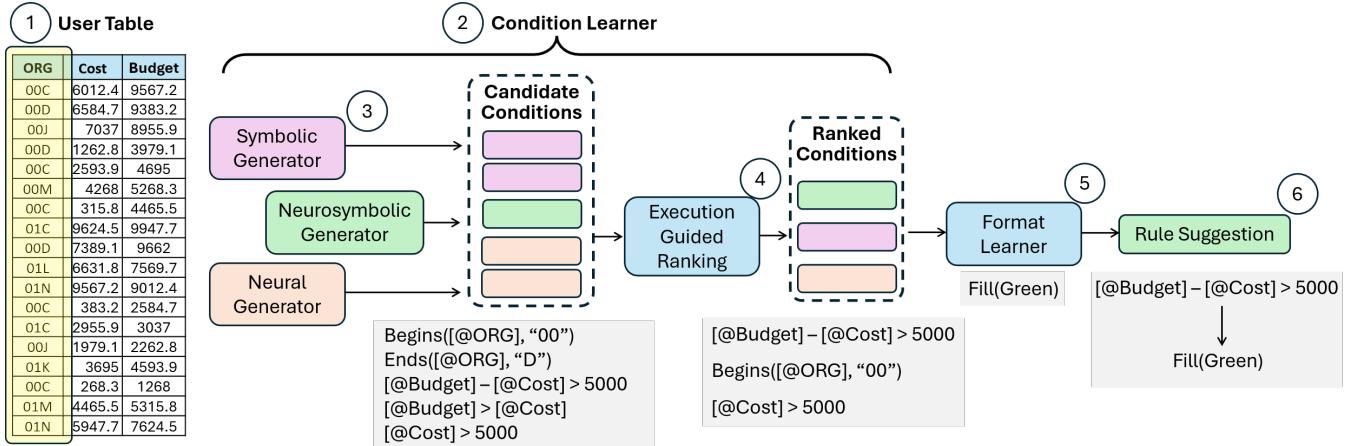


Figure 2: Summary of TAFO on a user table (1). First, TAFO learns multiple conditions for the tables (2) by pooling candidate conditions from multiple generators (3) and ranking them using an execution based ranker (4). After the condition, TAFO generates the associated format for the learned conditionals (5) generating the final suggestion (6).

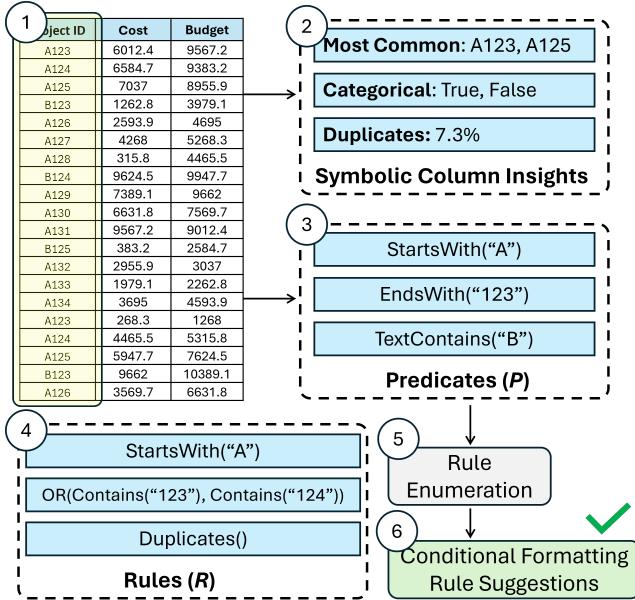


Figure 3: Symbolic rule synthesis overview. The input table (1) is used to extract table properties (2) that yield predicates (3) that are combined (4) into the final rules (5).

EXAMPLE 3. For the table shown in Figure 2, the “ORG” column will have a text property, `MostCommonValues(3)` with value {“00C”, “00D”, “00E”} and a general property `NumBlanks` with value 0.

Enumerating predicates. Predicates are boolean-valued function that takes a cell e and with zero or more additional arguments, and return true if the property that it describes holds for the cell e . All predicates are assigned a type and they only match cells of their type. We use a synthesis based predicate generator that

Table 1: Supported properties and their arguments for each datatype (top) and also general (bottom). The k argument in properties determines the maximum number of items to be considered. For example, `Formulas(5)` has at most 5 formulas from the table. By default these are sorted by frequency.

| Numeric | Datetime | Text |
|-------------------|-------------|-------------------------|
| AverageValue | InLastWeek | MostCommonValues(k) |
| MedianValue | InNextWeek | DuplicatesValues(k) |
| 90PercentileValue | InThisWeek | Categories |
| 75PercentileValue | InLastMonth | FreeText |
| 25PercentileValue | InNextMonth | AverageLength |
| 10PercentileValue | InThisMonth | |
| Skew | Today | |
| | Year | |
| General | | |
| NumErrors | NumBlanks | Formulas(k) |
| NumLogicals | NumNA | NumDuplicates |
| NumUniques | NumDate | NumFormatted |

does bottom up construction over a collection of base properties and components to generate candidate predicates. Synthesis based predicate generators have shown good performance on formatting tasks CORNET [40]. Unlike previous work, we do two modifications that impact the quality of the enumerations – (1) our synthesis is component based so the smallest element in the search space can be a combination of properties for example, `NOT(BLANKS())` is a possible component; (2) the nodes in the synthesis process are ranked using a heuristic over the properties of the node. For properties that return multiple values (like `MostCommonValues(k)`) each output is considered as an individual constant.

Neural properties and components are added to the rule enumeration step with higher weight, and constants and columns are used to boost the weight of symbolically generated predicates by adding

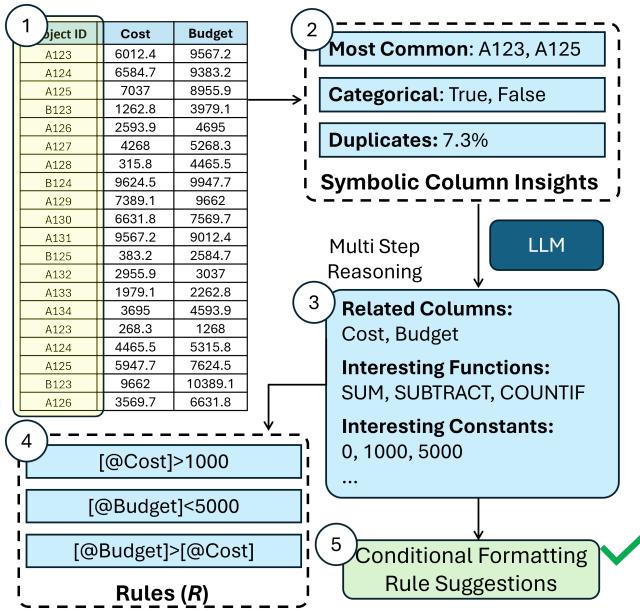


Figure 4: Overview of the LLM based Rule Generation. The input table (1) is used to extract table properties (2) which are used as context in the LLM prompt (3) followed by a multi step reasoning (4) and the final suggested rules (5).

a reward to the node scores which use these columns and constants. The reward is computed as 10% of the original score of the node. Figure 6 shows the overview of the neuro-symbolic generator on the example task.

EXAMPLE 4. For the “ORG” column in Figure 2, two generated predicates are `TextStartsWith("00C")` and `TextEndsWith("D")`.

Combining predicates. We represent complex conditions as propositional formulas in disjunctive normal form over predicates. One challenge with enumerating rules is handling the huge number of candidates. Prior work like CORNET [40] and FORMAT5 [36] use an initial user specification—in the form of input-output examples and natural language descriptions, respectively—to guide this enumeration and prune the number of candidates. We do not have any specification or user intent in our task. Even with type constraints and syntax pruning, we end up with over 100K candidates per table. We therefore train a ranker to score candidate rules—partial rules are valid rules—and use this ranker in a beam search to enumerate candidates. We use the table properties (Table 1) along with execution properties of the rule. The properties of the partial program used for ranking are: percentage of table highlighted, type of rule, category of rule, and argument length. We generate positive examples by mining partial rules from conditions in our corpus. We generate negative examples by generating random rules over tables in our corpus. The ranker is a dense network with three layers over the table and rule features, that is trained with logistic regression objective and used by considering the predicted logit as a score.

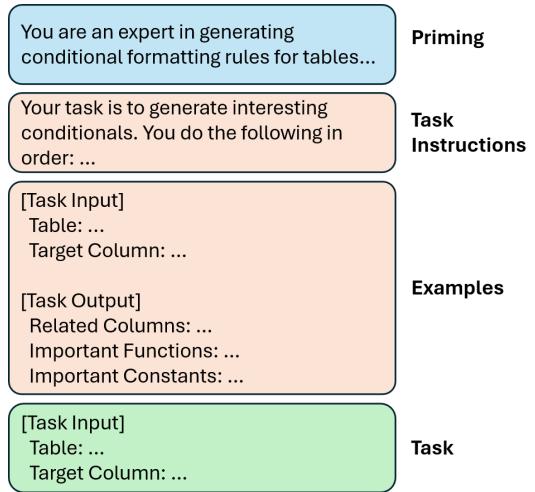


Figure 5: Prompt structure used to generate neural candidates for conditionals. The prompt uses a multi-step reasoning approach in a few-shot setting with task priming.

4.2 Why we need neurosymbolic generation

We can consider a pure symbolic systems that extracts properties using templates and does rule based enumeration for predicate construction, or a pure neural system that prompts a language model to directly to generate predicates as alternatives to the neurosymbolic learner. Upon examining the candidates generated by pure symbolic and neural systems—see Figure 3 and Figure 4—we find that none of these generate the actual user added condition, $[@Budget]-[@Cost]>1000$. Symbolic candidates contain diverse operations, but they lack semantic knowledge to know that “Budget” and “Cost” are related. Neural candidates have semantic knowledge, but they lack the diversity due to lack of execution. For example, the top neural candidate $[@Budget]>@Cost$ evaluates to true for every row in the table and upon execution formats the entire column.

Our experiments show that considering the neural, symbolic and neuro-symbolic approaches separately and then ranking still improves the performance over only using the neuro-symbolic approach (Section 6.3). This happens due to the increased diversity, which significantly impacts performance (Section 6.2).

4.3 Ranking conditions

The generated conditions need to be ranked based on their relevance to the table. Previous work has used symbolic features [6], LLMs [13] and custom models [40] for ranking suggestions. One key difference in formatting is that its a visual task—the execution of the rules play a crucial role in determining its score. Furthermore, we also want the generated suggestions to be diverse and cover different parts of the rule grammar and the table.

To solve these competing objectives of diversity and quality, we use execution-based ranking. Let C be the collection of learned conditions. First, we use the node ranker described in Section 4.1 to assign a score s_i for all conditions $C_i \in C$. Next, we execute all candidate rules $C_i \in C$ on the table E to get the output as boolean vectors $O_i \in \mathbb{B}^n$. The candidates are then clustered into semantic

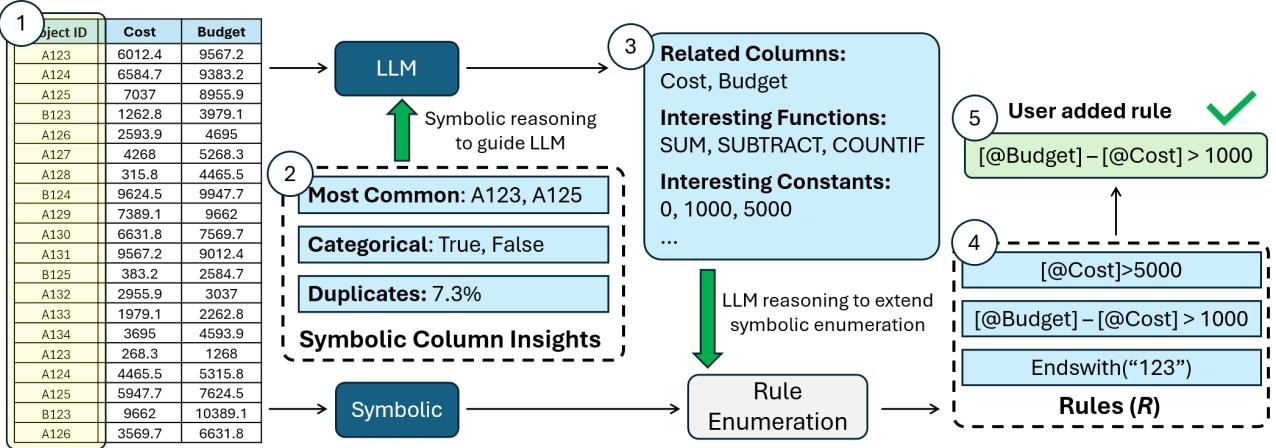


Figure 6: Figure summarizing the neuro-symbolic condition generator in TAFO for sample table (1). The symbolic column insights (2) are used as context by the LLM to generate conditions. The multi-step reasoning generated by LLM (3) is used by the symbolic rule enumerator to generate the final set of rules (4). This generates the user added condition (5) in the candidates.

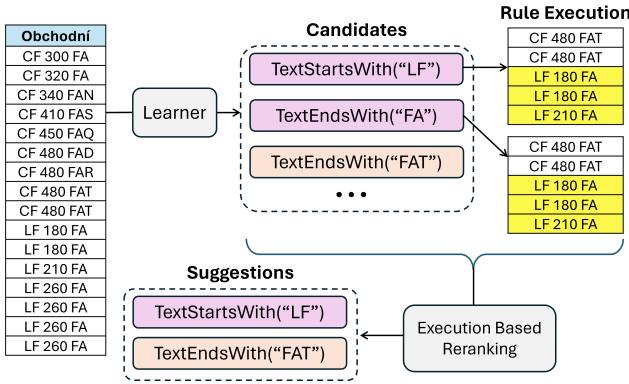


Figure 7: Suggestion ranking system. The learner generates a ranked set of candidate rules based on the applicability of the rule. To ensure a diverse selection of suggestions shown to the user, TAFO executes the candidates rules and clusters the candidates into equivalence classes based on the output formatting they generate. The clusters are then ranked with the top ranked rule from each cluster being selected.

equivalence classes EC_k such that $\forall C_i, C_j \in EC_k : O_i = O_j$. Each equivalence class is a unique execution on the column.

Each cluster is scored based on the average scores of the rules present in the cluster $S_i = \text{Avg}(s_r \in EC_i)$. To generate the final list of rules we do a round robin sampling of the rules in order of the rule scores s_r from clusters in order of the cluster scores S_i for all clusters $\bigcup_n EC_i$. This ensures that the rules belonging to different execution equivalence classes are prioritized. Figure 7 summarizes the overview of the execution guided ranking setup.

EXAMPLE 5. Consider the table shown in Figure 7, the learner generates candidates which are then clustered into semantically equivalent classes denoted by orange and pink highlighting. The top rule from each semantic class is picked as the final ranked subset.

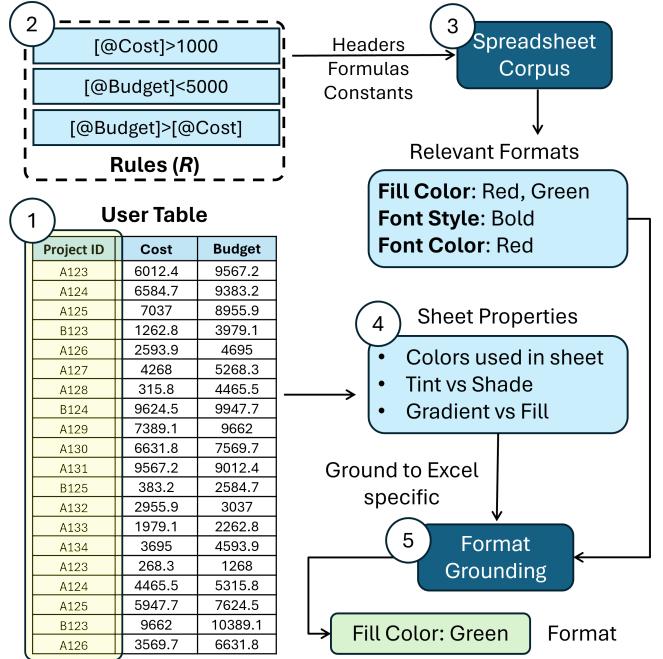


Figure 8: Summary of the format learning for TAFO. Given a user table \mathcal{T} (1) and the learned rule triggers $t \in (t, f) = r \in R$, TAFO retrieves similar $(table, rule)$ pairs from the corpus (3) and mines formats used in them. The mined formats are then transformed based on the formatting properties found in \mathcal{T} . The final formats are grounded and the final format is returned (5).

4.4 Learning formats

Since conditional formatting is a visual task, suggesting the right format is almost as crucial as learning the condition. For example,

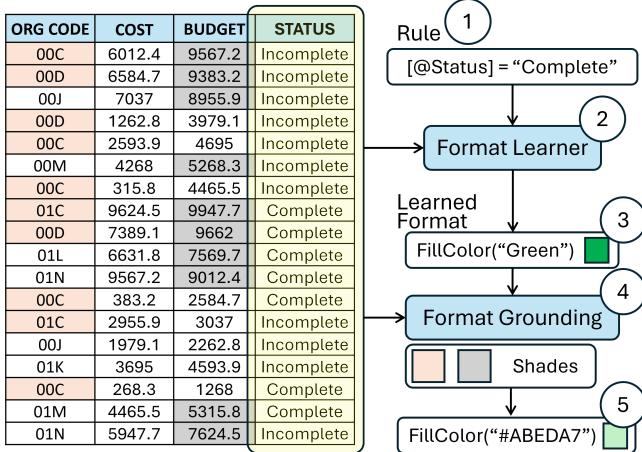


Figure 9: Figure showing the format grounding for a sample table: (1) the learned rule for which we need to learn formats; (2) TAFO format learner using corpus and table information; (3) the formats learned; (4) The table has all existing formatted cells in shade colors and not solid colors; (5) The suggested format is a shade version of the originally learned format.

a condition `TextEquals("PASS")` is more likely to be associated with a light green fill color than with a red font color. Past work [36, 40] has not looked at learning formats and only focused on learning the rule component of formatting rules. We build on the intuition shown by previous work in spreadsheet automation [53] which suggest that similar tables have similar artifacts, like formulas, charts and pivot tables. We extend this hypothesis for formats used in tables. We leverage a corpus of conditional formatting rules to suggest relevant formats for a given condition C and table E .

First, we collect candidate formats $f_c = f_e \cup f_r$ with f_e being all formats already present in the current sheet and f_r the formats from similar tables and rules in the corpus. To find these related tables, we compare tables on their headers and formulas they contain, and rules on the predicates and their constants of the condition C . We use weighted hamming distance to compute this similarity, where weights are obtained by manually annotating the similarity of 100 tables and performing linear regression over these. We mine rule-table pairs in decreasing order of their similarity until either λ_N pairs have been mined or the similarity score drops below a threshold λ_T , with λ_N and λ_T hyperparameters.

Second, we select the formats using statistical properties of the format identifiers and their values. We score format identifiers based on their frequency in the candidate set. To bias the system towards formats used in the current sheet, formats in f_e have twice the weight as those in f_r . We pick all format identifiers that have an average score greater than 0.5 in the candidate set. To pick values for the format identifiers, we use the same frequency based ranking withing the format identifier with a slight change that colors are compared using the closest HTML web color instead of direct hex codes. For each format identifier, we only keep the highest ranked value. The final format is then the union of all selected formats.

Table 2: Aggregate properties of benchmark problems divided by the column data type. # Cells is the average number of cells in the column, and # Formatted is the average number of cells which contain user formatting in the column. Rule depth is defined as the tree depth of the abstract syntax tree produced by parsing the rule using our grammar.

| Type | Rules | # Cells | # Formatted | Rule Depth |
|-----------------------------------|--------------|--------------|-------------|------------|
| (1) Tables with CF | | | | |
| Text | 13.81 K | 107.5 | 32.1 | 2.3 |
| Numeric | 9.32 K | 184.8 | 111.2 | 1.8 |
| Date | 1.87 K | 73.3 | 23.5 | 1.7 |
| Total | 25 K | 133.7 | 60.9 | 2.1 |
| (2) Tables with manual formatting | | | | |
| Text | 42.4 K | 53.6 | 17.4 | – |
| Numeric | 47.1 K | 89.2 | 21.9 | – |
| Date | 10.5 K | 34.1 | 10.2 | – |
| Total | 100 K | 68.3 | 49.5 | – |

Finally, since Excel allows using the same color with different shades, we ground the format properties based on indicators mined from the table. If at least 75% of cells has a specific shade, we suggest that shade. This allows TAFO to personalize the formats to user preferences beyond specific colors.

EXAMPLE 6. Figure 9 shows a sample table where the rule to highlight cells equal to `"Incomplete"`. The learned format for this rule is `FillColor("#008000")` which is solid green. The format grounding engine mines the shade indicator since all existing formats in the table have shade fill colors. The format grounding thus grounds the fill color to `#ABEDA7` which is a lighter shade of solid green.

5 Evaluation Setup

In this section we describe the benchmarks, evaluation metrics and baseline systems which we use for evaluating TAFO.

5.1 Benchmarks

To train and evaluate TAFO, we leverage a corpus of 1.8 million Excel workbooks scraped from public sources [40]. This corpus has previously been used to study tabular formatting tasks [36, 40]. Each task in this corpus consists of a table and a conditional formatting rule. We use the de-duplicated set as described in [40], which consists of 105K tasks. We use 80K tasks for training the ranker and rule generation engine, and the remaining 25K tasks for evaluation. Table 2 shows a summary of the tasks (1).

Additionally, we also consider non rule based table formatting. We extract tables with columns that contain manual formatting for cell fill color (without conditional formatting rules). We consider a column E^j as formatted if it has > 5 and $< |E^j|$ formatted cells. This yields another 683K tasks, out of which we randomly sample 100K for evaluation. Table 2 shows a summary of these tasks (2).

5.2 Metrics

In this section we describe the metrics used to evaluate quality and diversity of conditions and quality of formats.

5.2.1 Condition quality. To evaluate a learned rule C_L against a user-written rule C_U , we use three popular existing metrics in this domain [36, 40]: **exact match**, **sketch match**, and **execution match**. Exact match is a syntactic match between a learned rule and the user-written rule, with tolerance for white space and argument orderings that do not impact execution. Sketch match is a relaxed syntactic match that only looks at the operators and rule structure between a learned rule and the user-written rule, with tolerance for differences in arguments. Execution match consists of executing two rules and comparing the outcomes produced ($\forall E^j \in E : C_L(E^j) = C_U(E^j)$). In addition to capturing the fact that different rules can produce the same execution, execution match allows us to evaluate against baselines that directly predict formats, rather than produce rules. This distinction between exact, sketch and execution match has been made in previous work in formatting [36, 40] and related areas, such as text to code [20, 34].

EXAMPLE 7. *TextEquals("A") \wedge TextEquals("B") and TextEquals("B") \wedge TextEquals("A") are an exact match as the rules are semantically equivalent. TextStartsWith("D") and TextContains("D") are not an exact match because the rules are not equivalent. These may be an execution match on a column that only has "D" at the start of values.*

5.2.2 Condition diversity. To evaluate diversity in conditions, we consider three pairwise metrics: **average edit**, **embedding** and **execution distance**. Edit and embedding distance measure the diversity in the rule space while execution distance measures the diversity in the output format space. Edit distance is a token level edit distance between the generated conditions, averaged over all pairwise combinations of suggestions. Embedding distance is the average of all pairwise cosine similarities between embeddings of conditions rendered as code. We use CodeBERT [9] to compute the embedding. Execution distance computes diversity by considering the output of condition C on a table as a boolean vector $[C_L(E^j) | i = 1 \rightarrow n]$. We compute the pairwise Hamming distance over these vectors. These metrics for diversity have previously been used to evaluate code generation systems in literature [37].

5.2.3 Format quality. To evaluate the learner formats against the user-added formats, we consider two metrics: **color match** and **property match**. Color match is an approximate, qualitative comparison between the format suggested and the format applied by the user. To match colors, we map a color to the closest extended CSS color name [46] using euclidean distance between the RGB color values ($\in \mathbb{R}^3$) for both the predicted and ground truth color and match the CSS colors. Property match is a relaxed matching which only checks if the same format properties $\subseteq F$ have been modified, without considering their values. Property match allows us to understand the high level formatting properties and is often more indicative, since matching exact colors and fonts might be too specific and not reveal the true effectiveness of the suggestion.

EXAMPLE 8. *Fill(#FF0000) and Fill(#FF0011) are an exact match as they both map to HTML color Red. On the other hand, FontStyle(bold) and FontStyle(italic) are an intent match since they have the same formatting operator (FontStyle) but different arguments.*

5.3 Baselines

As we introduce the problem of predictively suggesting conditional formatting rules, there are no existing systems that perform this task. We therefore adapt a variety of approaches related to this problem, some of which have been previously applied to conditional formatting learning. Six approaches are symbolic, five of which are able to generate rules. Symbolic baselines only generate conditions—no formats. Three neural approaches cast conditional formatting as cell classification, and we consider different baseline models and cross-attention mechanisms. Four neural baselines cast it as text generation with large language models. Neural baselines generate both conditions and formats. The following sections describe these baselines in more detail. Implementation details are in Appendix ??.

5.4 Symbolic

5.4.1 Greedy Enumeration. Greedy enumeration uses the same predicate generation as TAFO and then performs a beam search with beam width 10 and maximum depth of 5, and use the CORNET ranker to drive the search and select the best condition.

5.4.2 CORNET. CORNET [40] is a system that learns conditional formatting rules by example. We adapt CORNET to suggest CF rules without examples by replacing its semi-supervised clustering with an unsupervised clustering among two classes (formatted and non-formatted). We did not see improvements with the iterative tree learner and therefore learn a single decision tree.

5.4.3 Clustering. Conditional formatting can be treated as a cell clustering problem where clusters denote the formatting groups. We use both state-of-the-art symbolic [23] and neural [32] clustering techniques to split the data in groups with different formatting. For neural clustering, the values are embedded before clustering. Since, CORNET also uses clustering in its pipeline and our results show that both neural and symbolic clustering alone perform significantly worse than CORNET, we omit these results from the main paper. The neural clustering results can be found in the Appendix.

5.5 Neural

We consider two categories of neural baselines, based on the base models. The first category are models that learn table representations for downstream table intelligence tasks, like question-answering or cell and table type classification. The second category are pre-trained, auto-regressive language models, which we both fine-tune and prompt to generate rules and formats over tables.

5.5.1 TAPAS and TUTA. TAPAS [11] is a table encoding model trained for sequential question answering (SQA) and TUTA [48] is a tree-based transformer model pre-trained on multiple table tasks, such cell type classification (CTC). We apply it to conditional formatting by using it to encode the input column and getting an embedding for each cell. To generate conditions, use a classification head (dense model) on top of these cell embeddings that predicts formatted or not formatted. To generate formats, use a generation head (CodeT5+ decoder) on top of the cell embeddings.

5.5.2 FORMAT5. FORMAT5 [36] is trained to generate conditional formatting rules from natural language and examples. We remove the natural language condition but keep the abstention objective,

which teaches the model to emit a placeholder token `<?>` if it cannot confidently predict a value token. A separate model is used to predict values for these placeholders. For example, the first model predicts `Fill(<?>)` and the second model predicts that `<?>` should be `#FF0000`. This yields better values predictions, because the second model can already look at the whole condition.

5.5.3 GPT-4 [31], CodeT5+ [47], StarCoder [19] and CodeLlama [35]. We use open-source code generation models (CodeT5+, StarCoder and CodeLlama) through prompting and fine-tuning, and a closed-source language model (GPT-4) through prompting. In both cases, the model is tasked with generating both the condition and the formatting. The model prompt consists of the table information and 5 few-shot examples from the training corpus picked using CodeBERT [9] similarity. The table is encoded as header, column types and sample values, similar to FORMAT5 [36]. To improve performance of prompting these models, recent work has highlighted the benefit of using chain-of-thought [49], tree-of-thought [51], and program-of-thought [3]. We find that in our experiments, program-of-thought works consistently better than the other variants and we thus only report results for program-of-thought for brevity. The other variant results can be found in the appendix.

6 Evaluation

We perform experiments to answer the following questions:

- Q1.** Do conditions and formats generated by TAFO match those applied by users?
- Q2.** Can TAFO generate sufficient, interesting and diverse rules?
- Q3.** How do different design decisions and properties impact performance of TAFO, and are its system requirements?

6.1 Q1: Suggestion Performance

We study the performance of learning conditions and formatting separately, and end-to-end performance of learning both.

6.1.1 Conditions. Table 3 presents an overview of our results for condition learning. TAFO outperforms symbolic and neural baselines on exact, sketch and execution match metrics. Symbolic methods perform worse than methods with a neural component. This neural component helps to understand the semantics of the data. CORNET obtains second highest sketch match in top-1 but scores lower in execution match, which indicates that semantics are important to determine the right constant values. TAFO combines neural and symbolic components to extract predicate with interesting constants and then ranks them to obtain best of both worlds.

In addition to evaluating TAFO on conditions written by users, we also investigate the extent to which it can learn conditions for formatting that users carried out manually. Since there is no rule, we report execution match. Figure 10 shows the execution match for TAFO with increasing number of suggestions generated (top- k). We find that TAFO could have automated 55% of column formatting for the users with just three suggestions.

6.1.2 Formatting. As formatting is a visual task, we evaluate TAFO and baselines on their ability to generate the formats that the users added in their tables. For this experiment, we only match the format and ignore the condition.

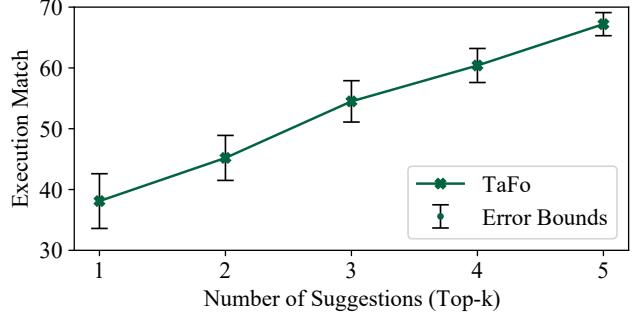


Figure 10: TAFO performance on tasks with manual formatting. We show execution match accuracy for increasing number of samples generated (top- k).

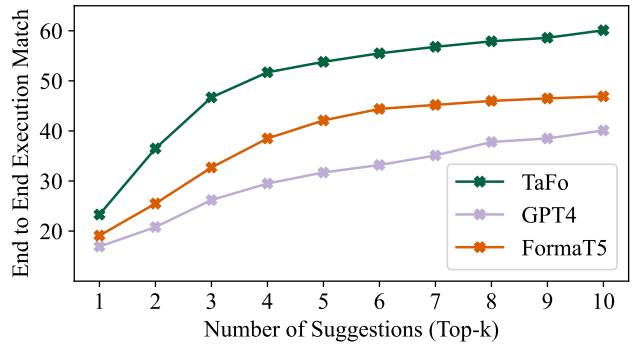


Figure 11: End to end execution match for TAFO, the best baseline (GPT-4) and the current state-of-the-art in table formatting (FORMAT5) against increasing number of suggestions (Top- k). TAFO outperforms both baselines

Table 4 shows the exact and property match for top-1, 3 and 5 suggestions for TAFO and baselines. We find that TAFO performs consistently better than baselines on both exact match (+4.8–9.1%) and property match (+9.4–23.7%) over the best baseline.

Qualitative analysis of the baseline performance reveals insights about the common mistakes made by these systems. Neural models like GPT-4 tend to prefer extreme colors like “red” and “green” for boolean operations. On examination of ground truths we find that in a lot of these cases, however, the actual user preference is a mild color like “pink” or “yellow”. TAFO is able to correctly generate the right format, as these patterns are prevalent in the corpus.

6.1.3 End-to-end. We also evaluate end-to-end performance of TAFO, which considers both the rule trigger and format together. We consider end-to-end execution match as (condition execution match) \wedge (format color match). Figure 11 shows the end-to-end execution match for TAFO and baselines on increasing number of suggestions. We find that TAFO performs consistently better than the next best baselines (GPT-4 and FormaT5). TAFO gets an end-to-end execution match of 50% with just three suggestions.

Table 3: Comparison of TAFO with neural and symbolic baselines on learning conditions. We report sketch, exact and execution match for top-1, 3 and 5 suggestions generated. “Rules” denotes if an approach generates symbolic rules. TAFO outperforms neural and symbolic baselines across all execution, sketch and exact rule match.

| System description | | | Execution match | | | Exact match | | | Sketch match | | |
|--------------------|-----------------------|------------|-----------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| Name | Technique | Rules | Top 1 | Top 3 | Top 5 | Top 1 | Top 3 | Top 5 | Top 1 | Top 3 | Top 5 |
| Greedy Enumeration | Symbolic | Yes | 11.8% | 18.5% | 25.6% | 6.0% | 9.7% | 12.0% | 36.8% | 41.2% | 53.3% |
| Clustering | Symbolic | No | 13.5% | 15.6% | 17.4% | — | — | — | — | — | — |
| CORNET | Symbolic | Yes | 24.3% | 29.9% | 39.0% | 12.3% | 15.7% | 18.3% | 51.9% | 58.9% | 65.8% |
| TAPAS | Neural | Yes | 19.6% | 23.4% | 31.5% | 9.9% | 12.3% | 14.7% | 38.8% | 46.1% | 57.0% |
| TUTA | Neural | No | 24.1% | 30.1% | 37.8% | — | — | — | — | — | — |
| FORMAT5 | Neural | Yes | 29.4% | 37.5% | 57.0% | 14.9% | 19.6% | 26.7% | 47.8% | 62.4% | 71.9% |
| CodeT5+ | Neural | Yes | 28.5% | 35.5% | 56.6% | 14.4% | 18.6% | 26.5% | 42.5% | 57.8% | 72.5% |
| StarCoder | Neural | Yes | 28.3% | 35.5% | 55.0% | 14.3% | 18.6% | 25.7% | 42.9% | 59.3% | 71.7% |
| CodeLlama | Neural | Yes | 27.1% | 35.2% | 53.1% | 13.7% | 18.4% | 24.9% | 41.3% | 57.6% | 70.4% |
| GPT4 | Neural | Yes | 31.2% | 39.1% | 58.3% | 15.8% | 20.5% | 27.3% | 46.5% | 64.2% | 75.0% |
| TAFO | Neuro-symbolic | Yes | 36.7% | 46.8% | 64.3% | 18.6% | 24.5% | 30.1% | 54.2% | 73.3% | 84.8% |

Table 4: (Formatting Identifier) Comparison of TAFO with neural and symbolic baselines. We report exact and property match for top-1, 3 and 5 suggestions generated. TAFO outperforms all baselines accross both exact and property match.

| System | Exact match | | | Property match | | |
|-------------|--------------|--------------|--------------|----------------|--------------|--------------|
| | Name | Top 1 | Top 3 | Top 5 | Top 1 | Top 3 |
| TAPAS | 4.3% | 7.1% | 9.0% | 8.6% | 17.7% | 23.4% |
| FormaT5 | 9.5% | 14.3% | 16.2% | 19.1% | 35.7% | 42.1% |
| CodeT5+ | 8.1% | 10.1% | 11.9% | 16.3% | 25.2% | 31.0% |
| StarCoder | 7.7% | 9.9% | 11.5% | 15.5% | 24.7% | 29.9% |
| CodeLlama | 7.1% | 9.3% | 10.8% | 14.3% | 23.2% | 28.1% |
| GPT4 | 8.4% | 10.5% | 12.2% | 16.9% | 26.2% | 31.7% |
| TAFO | 13.2% | 18.9% | 21.3% | 26.5% | 47.2% | 55.4% |

6.2 Q2: Condition Analysis

It is important that we generate various (completeness and diversity) and interesting (complexity and coverage) conditions. We study these properties in the following paragraphs.

Completeness. Figure 12a shows the number of tasks for which at least $> k$ suggestions are made for increasing k , for both TAFO and GPT-4. This corresponds to an upper bound of the recall after k suggestions. Because we combine different methods of generation, among which exhaustively generating predicates, TAFO obtains a higher upper bound on recall.

Coverage. Figure 12b shows percentage of cells in a column that are covered by any condition for increasing number of suggestions, for TAFO and GPT-4. Because TAFO ranks suggestions based on their execution, it covers significantly more cells. This is especially noticeable after the second suggestion, which upon execution, affects completely different cells than the first suggestion. With just 2 suggestions, TAFO achieves a column coverage of over 80%.

Complexity. Figure 13 shows how the complexity evolves for generating more conditions based on two metrics: (a) the number of tokens in the condition, with tokens defined as predicates and

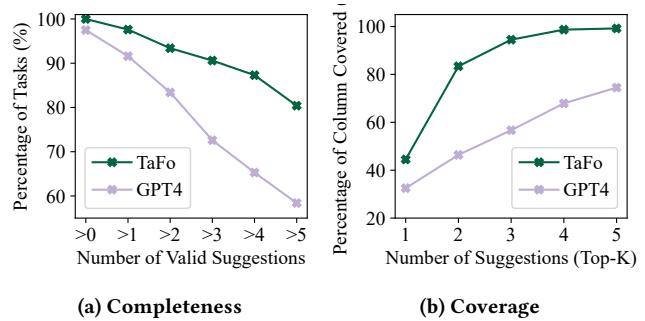


Figure 12: (a) Completeness: Percentage tasks with greater than k suggestions; and (b) Coverage: Percentage of the column covered by at least one rule in the suggestions for TAFO and GPT-4. We find that TAFO generates three or more suggestions for over 90% of the benchmark tasks and achieves a column coverage of over 80% with just two suggestions.

arguments; and (b) depth of the syntax tree of the condition. As opposed to GPT-4, TAFO yields more complex conditions as the number of conditions increases. This happens for two reasons: recombining simpler conditions (with predicates at the lowest level) into more complex conditions and ranking these based on execution to ensure that complex conditions are also chosen.

Diversity. Table 5 summarizes condition diversity metrics of TAFO and baselines over the top 5 suggestions. We only consider the best performing baseline from each category—the remainder of results is in the Appendix. Because of ranking with execution, TAFO achieves highest diversity across all metrics. Interestingly, neural models (GPT-4, CodeT5+) struggle with diversity, despite good overall performance (Table 3). Low-diversity generations is a known issue in large language models [28, 37]. FORMAT5 and CORNET also ranks candidates and achieves high rule diversity, but they do not consider differences in execution across rules.

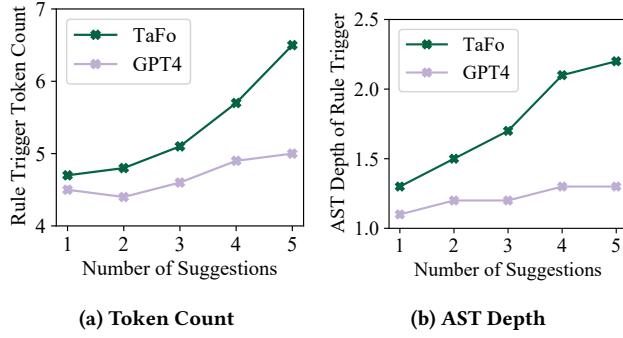


Figure 13: Rule trigger t complexity for suggestions generated by TaFo and the best baseline (GPT4). We show (a) Token Count: average number of tokens in the suggestions; and (b) AST Depth: average depth of rule trigger in the AST. We find that the complexity of conditions generated by TaFo increases with the number of suggestions as most of the low complexity rules have similar execution.

Table 5: Diversity metrics for top-5 suggestion generation for TaFo and baselines. We report average pairwise edit distance, embedding similarity and Hamming distance of the output formatting on execution. Arrows in the header indicate if higher (\uparrow) or lower (\downarrow) value is better.

| System | Diversity Metric | | |
|-------------|------------------------------|----------------------------|--------------------------|
| | Edit Distance (\uparrow) | Embedding (\downarrow) | Execution (\uparrow) |
| TAPAS | 6.2 | 0.95 | 0.27 |
| CORNET | 9.2 | 0.94 | 0.33 |
| FORMAT5 | 10.4 | 0.91 | 0.37 |
| CodeT5+ | 6.5 | 0.96 | 0.25 |
| GPT-4 | 7.6 | 0.94 | 0.31 |
| TaFo | 14.3 | 0.87 | 0.65 |

6.3 Q3: Design Choices and Input Properties

We evaluate the impact of different design decisions and input properties on the performance of TaFo.

Symbolic \leftrightarrow neural \leftrightarrow neuro-symbolic. The neural and symbolic components of TaFo can be decoupled and work independently—see Section 4.1. Figure 14 shows the execution match accuracy of condition learning for TaFo and its purely neural, symbolic and neurosymbolic ablations. We find that TaFo outperforms all ablations by 4.5%–16.7% and 5.8%–21.8% respectively showing that different condition learners specialize on different tasks thus indicating the value of having multiple condition learners. Further, the neurosymbolic ablation outperforms both the purely neural and symbolic, highlighting the value of sharing reasoning paths.

Condition ablations. We ablate different components to understand their impact on the performance of TaFo. We ablate execution guided ranking by considering the original order of rules returned by the trigger learner. We ablate symbolic property extraction by directly running enumeration and LLM generation without

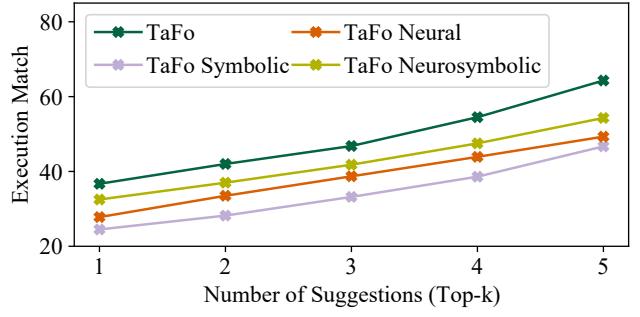


Figure 14: Execution match for condition learning for TaFo and its neural, symbolic and neurosymbolic only variants for increasing number of suggestions (Top- k). TaFo outperforms all ablations, especially with higher number of suggestions, which highlights the value of combining the neural, symbolic and neurosymbolic techniques. Further, the neurosymbolic variant outperforms both neural and symbolic variants.

Table 6: Execution match for condition learning for different ablations of TaFo. Each ablated component is denoted by ‘-’. Symbolic properties (row 2) and ranking (row 1) have the highest impact on performance. Neuro-symbolic interaction also affects performance of TaFo, improving over directly combining candidates from symbolic and neural variants.

| System | Top-1 | Top-3 | Top-5 |
|-------------------------------------|--------------|--------------|--------------|
| TaFo – execution-guided ranking | 19.8% | 34.5% | 56.4% |
| TaFo – symbolic property extraction | 17.2% | 20.7% | 53.4% |
| TaFo – LLM multi-step reasoning | 23.5% | 38.2% | 58.8% |
| TaFo – neuro-symbolic interaction | 25.8% | 40.0% | 59.6% |
| TaFo | 36.7% | 46.8% | 64.3% |

these properties. We ablate LLM generation by not using multi-step reasoning. Table 6 shows execution match results for condition learning for top-1, 3 and 5 suggestions for these ablations. We find that symbolic property extraction has the biggest impact on performance (-19.5% in top-1). This is expected since the properties provide building blocks for the symbolic and neural models to reason about the data. Execution-guided ranking significantly affects the top-5 performance (-20.4%) since not selecting diverse samples reduces coverage and causes repetition in the suggestions.

Format ablations. Table 7 shows the performance of learning formats with different components removed. We find that the corpus retrieval has the most significant impact on performance (-19.1% – 9.0%). This is consistent with recent work [2] which finds that mining formula components and other artifacts from similar tables boosts performance since similar data has similarity in associated artifacts like formulas and formats. Not grounding the format impacts performance (-4.5%). Furthermore, not using sheet properties to align formats to the current sheet also affects performance (-3.4%).

Sample size of rows. The quality of suggestions is directly impacted by the amount of data available, however, due to practical

Table 7: Intent match for different ablations of TAFO format learner (F_{TAFO}). Each ablated component is denoted by ‘-’. Corpus retrieval has the highest impact on the overall performance of TAFO format learner showing that similar data have similar format preferences.

| System | Top-1 | Top-3 | Top-5 |
|-------------------------------|-------|-------|-------|
| F_{TAFO} – Corpus Retrieval | 17.5% | 28.1% | 34.8% |
| F_{TAFO} – Sheet Properties | 23.1% | 40.0% | 48.2% |
| F_{TAFO} – Format Grounding | 21.4% | 36.6% | 44.4% |
| F_{TAFO} | 26.5% | 47.2% | 55.4% |

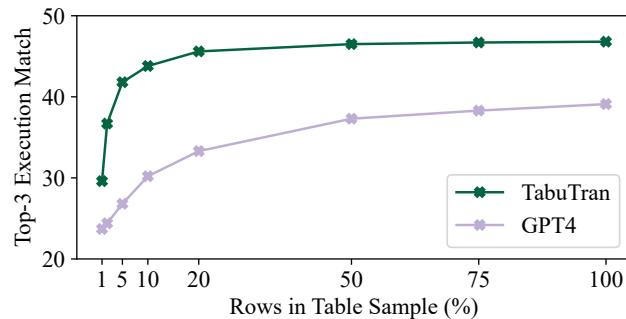


Figure 15: Top-3 suggestion execution match for condition learning for TAFO and best baseline (GPT-4) when using between 1% and 100% of rows. We sample rows randomly from tables with > 1000 rows. TAFO achieves over 40% execution match with just 10% of rows while GPT-4 only achieves 30%.

constraints, the entire table data might not be readily available. For example, spreadsheet clients often store a subset of data on the user viewport (active screen) which is updated as the viewport moves – for large spreadsheets this can be a small fraction of their total data. To be practically useful, TAFO must be able to work with table subsets without a significant drop in suggestion quality. Figure 15 shows the execution match accuracy when including from 1% to 100% of rows in the table sample. We find that the performance drops as the sample size is reduced. However, with just 5% rows, TAFO gets an execution match of 42.8% with 3 suggestions per task.

Data type of rule. We analyze the condition learning performance of TAFO based on the data types of the suggestions. Figure ?? shows the execution match for different rule data types with increasing number of generation size (Top- k). We find that TAFO has the highest performance on text columns and numeric and date columns have lower execution match with fewer suggestions and it improves at a lower rate compared to text columns on increasing the number of suggestions. This is inline with what previous work in spreadsheet formatting [40] have observed due to the numeric and date rules having an infinitely large search space for arguments.

Execution Time and Memory. We also evaluate the time and memory resources required by TAFO and baselines. Table 8 shows the average latency, total amount of disk space required to store a system, and the average CPU and GPU memory used for prediction

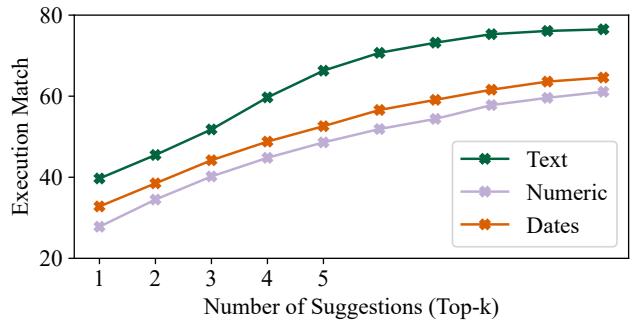


Figure 16: Execution match for condition learning for TAFO over increasing number of suggestions (Top- k) for different column data types. TAFO has higher accuracy for Text columns. Numeric and Date columns keep increasing with more suggestions, due to the larger search space of rules.

Table 8: Total disk space and average memory used in megabytes for inference over benchmark tasks for TAFO and baselines. GPU usage for TAFO is not included as it uses GPT4 via API. TAFO uses GPT4 via API and the latency reported includes network latency. Symbolic variant of TAFO is lightweight with latency of only 189.3 ms.

| System | Latency | Disk | CPU | GPU |
|--------------------|---------|-------|-------|--------|
| Greedy Enumeration | 781.3 | 3.1 | 103.4 | 0 |
| CORNET | 254.3 | 2.9 | 61.4 | 0 |
| TUTA | 2194.4 | 741.5 | 7.6 | 1432.8 |
| TAPAS | 1832.5 | 412.4 | 9.1 | 1713.2 |
| TAFO | 1218.5 | 4.5 | 8.3 | – |
| TAFO Symbolic | 189.3 | 4.5 | 8.9 | 0 |

over benchmarks for TAFO and baselines. We also report the latency and memory of TAFO for just the symbolic ablation which does not require LLM resources. Section 4.1 describes the symbolic variant of TAFO designed for low-resource environments. Since TAFO makes LLM calls over an API (GPT-4), we do not report GPU memory for TAFO and the latency reported includes network latency. Appendix contains the tokens and cost analysis for TAFO and discussion over replacing the LLM with a smaller and cheaper model.

7 Conclusion

In this paper, we introduced the novel problem of learning conditional formatting rules without user specification. We proposed TAFO, a system that learns such data-dependent formatting suggestions with rule and format based on tabular context. We evaluate TAFO, on a benchmark of 105K CF tasks extracted from 1.8 million real Excel spreadsheets and evaluate quality, completeness, diversity and complexity of the suggestions. We compare TAFO to both symbolic and neural approaches on this benchmark, for the table formatting task and also adapt techniques from other tasks in this domain, and find that TAFO performs significantly better than baselines. We evaluate TAFO on varying input properties to simulate practical challenges like availability of data and latency constraints.

We experiment with the components of TAFO and design choices made, analyzing the impact they have on overall performance. Finally, we evaluate TAFO on manually formatted tables which contain formatting but no rules and find that TAFO automates over 60% of these tasks. TAFO opens future work for other tabular predictive tasks like cleaning, transformation, visualization and querying. Furthermore, TAFO highlights the importance of corpus grounding and neurosymbolic interaction in recommender systems for tables.

References

- [1] Sergei Abramovich, Stephen Sugden, Sergei Abramovich, and Stephen J Sugden. 2004. Spreadsheet Conditional Formatting: An Untapped Resource for Mathematics Education. *Spreadsheets in Education* 1 (2004), 85105.
- [2] Sibe Chen, Yeye He, Weiwei Cui, Ju Fan, Song Ge, Haidong Zhang, Dongmei Zhang, and Surajit Chaudhuri. 2024. Auto-Formula: Recommend Formulas in Spreadsheets using Contrastive Learning for Table Representations. arXiv:2404.12608 [cs.DB]
- [3] Wenhui Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *Transactions on Machine Learning Research* (2023). https://openreview.net/forum?id=YfZ4ZPt8zd
- [4] Xinyun Chen, Petros Maniatis, Rishabh Singh, Charles Sutton, Hanjun Dai, Max Lin, and Denny Zhou. 2021. SpreadsheetCoder: Formula Prediction from Semi-structured Context. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, virtual, 1661–1672. https://proceedings.mlr.press/v139/chen21m.html
- [5] Haoyu Dong, Jinyu Wang, Zhouyu Fu, Shi Han, and Dongmei Zhang. 2020. Neural Formatting for Spreadsheet Tables. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management (Virtual Event, Ireland) (CIKM '20)*. Association for Computing Machinery, New York, NY, USA, 305–314. doi:10.1145/3340531.33411943
- [6] Kevin Ellis and Sumit Gulwani. 2017. Learning to Learn Programs from Examples: Going Beyond Program Structure. In *IJCAI 2017* (ijcai 2017 ed.). IJCAI 2017, Melbourne, Australia, 1638–1645. www.microsoft.com/research/publication/learning-program-examples-going-beyond-program-structure/
- [7] Anna Fariha and Alexandra Meliou. 2019. Example-Driven Query Intent Discovery: Abductive Reasoning Using Semantic Similarity. *Proc. VLDB Endow.* 12, 11 (jul 2019), 1262–1275. doi:10.14778/3342263.3342266
- [8] Anna Fariha, Ashish Tiwari, Alexandra Meliou, Arjun Radhakrishna, and Sumit Gulwani. 2021. CoCo: Interactive Exploration of Conformance Constraints for Data Understanding and Data Cleaning. In *Proceedings of the 2021 International Conference on Management of Data (Virtual Event, China) (SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 2706–2710. doi:10.1145/3448016.3452750
- [9] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *EMNLP 2020*. Association for Computational Linguistics, Online, 1536–1547. doi:10.18653/v1/2020.findings-emnlp.139
- [10] Sumit Gulwani. 2011. Automating String Processing in Spreadsheets using Input-Output Examples. In *PoPL'11, January 26–28, 2011, Austin, Texas, USA*. Association for Computing Machinery, New York, NY, USA, 317–330. https://www.microsoft.com/en-us/research/publication/automating-string-processing-spreadsheets-using-input-output-examples/
- [11] Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. Tapas: Weakly Supervised Table Parsing via Pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Seattle, Washington, United States, 4320–4333. https://www.aclweb.org/anthology/2020.acl-main.398/
- [12] Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhan Wu, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Wenyi Wang, Xiangru Tang, Xiangtao Lu, Xiawu Zheng, Xinbing Liang, Yaying Fei, Yuheng Cheng, Zongze Xu, and Chenglin Wu. 2024. Data Interpreter: An LLM Agent For Data Science. arXiv:2402.18679 [cs.AI]
- [13] Yupeng Hou, Junjie Zhang, Zihan Lin, Hongyu Lu, Ruobing Xie, Julian McAuley, and Wayne Xin Zhao. 2024. Large Language Models are Zero-Shot Rankers for Recommender Systems. arXiv:2305.08845 [cs.IR]
- [14] Nathan Hurst, Kim Marriott, and Peter Moulder. 2005. Toward tighter tables. In *Proceedings of the 2005 ACM symposium on Document engineering*. Association for Computing Machinery, New York, NY, USA, 74–83.
- [15] Harshit Joshi, Abishai Ebenezer, José Cambronero, Sumit Gulwani, Aditya Kanade, Vu Le, Ivan Radicek, and Gust Verbruggen. 2023. FLAME: A small language model for spreadsheet formulas. arXiv:2301.13779 [cs.PL]
- [16] Anirudh Khatry, Joyce Cahoon, Jordan Henkel, Shaleen Deep, Venkatesh Emani, Avrilia Floratou, Sumit Gulwani, Vu Le, Mohammad Raza, Sherry Shi, Mukul Singh, and Ashish Tiwari. 2023. From Words to Code: Harnessing Data for Program Synthesis from Natural Language. arXiv:2305.01598 [cs.DB] https://arxiv.org/abs/2305.01598
- [17] Vu Le and Sumit Gulwani. 2014. FlashExtract: a framework for data extraction by examples. In *2014 Programming Language Design and Implementation*. ACM, New York, NY, USA, 542–553. https://www.microsoft.com/en-us/research/publication/flashextract-framework-data-extraction-examples/
- [18] Hao Li, Chee-Yong Chan, and David Maier. 2015. Query from Examples: An Iterative, Data-Driven Approach to Query Construction. *Proc. VLDB Endow.* 8, 13 (sep 2015), 2158–2169. doi:10.14778/2831360.2831369
- [19] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Alkiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebazé, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokolov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swapayam Singh, Sasha Lucioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. StarCoder: may the source be with you! arXiv:2305.06161 [cs.CL]
- [20] Pietro Liguori, Erfan Al-Hossami, Domenico Cotroneo, Roberto Natella, Bojan Cukic, and Samira Shaikh. 2022. Can we generate shellcodes via natural language? An empirical study. *Automated Software Engineering* 29 (2022), 1–34.
- [21] Xiaofan Lin. 2006. Active layout engine: Algorithms and applications in variable data printing. *Computer-Aided Design* 38, 5 (2006), 444–456.
- [22] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. WizardCoder: Empowering Code Large Language Models with Evol-Instruct. arXiv:2306.08568 [cs.CL]
- [23] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA, 281–297.
- [24] Paula Maddigan and Teo Susnjak. 2023. Chat2VIS: Generating Data Visualisations via Natural Language using ChatGPT, Codex and GPT-3 Large Language Models. arXiv:2302.02094 [cs.HC]
- [25] Davide Mottin, Matteo Lissandrini, Yannis Velegrakis, and Themis Palpanas. 2016. Exemplar Queries: A New Way of Searching. *The VLDB Journal* 25, 6 (dec 2016), 741–765. doi:10.1007/s00778-016-0429-2
- [26] Joseph N. 2022. Number of Google Sheets and Excel Users Worldwide. https://askwonder.com/research/number-google-sheets-users-worldwide-eoskdoxav. Last Accessed: 2022-07-30.
- [27] Nagarajan Natarajan, Danny Simmons, Naren Datha, Prateek Jain, and Sumit Gulwani. 2019. Learning Natural Programs from a Few Examples in Real-Time. In *AIStats*. PMLR, online, 1714–1722. https://www.microsoft.com/en-us/research/publication/learning-natural-programs-from-a-few-examples-in-real-time/
- [28] Roberto Navigli, Simone Conia, and Björn Ross. 2023. Biases in Large Language Models: Origins, Inventory, and Discussion. *J. Data and Information Quality* 15, 2, Article 10 (jun 2023), 21 pages. doi:10.1145/3597307
- [29] Erich Neuwirth and Deane Arganbright. 2003. *The Active Modeler: Mathematical Modeling With Microsoft Excel*. Duxbury Press, online.
- [30] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. arXiv:2203.13474 [cs.LG]
- [31] OpenAI. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]
- [32] Ari Pakman, Yueqi Wang, Catalin Mitelut, JinHyung Lee, and Liam Paninski. 2020. Neural Clustering Processes. arXiv:1901.00409 [stat.ML]
- [33] Justin Payan, Swaroop Mishra, Mukul Singh, Carina Negreanu, Christian Poelitz, Chitta Baral, Subhro Roy, Rasika Chakravarthy, Benjamin Van Durme, and Elnaz Nouri. 2023. InstructExcel: A Benchmark for Natural Language Instruction in Excel. arXiv:2310.14495 [cs.CL] https://arxiv.org/abs/2310.14495
- [34] Gabriel Poesia, Oleksandra Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. 2022. Syncromesh: Reliable code generation from pre-trained language models. *CoRR* abs/2201.11227 (2022). arXiv:2201.11227 https://arxiv.org/abs/2201.11227
- [35] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémie Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade

- Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code Llama: Open Foundation Models for Code. arXiv:2308.12950 [cs.CL]
- [36] Mukul Singh, José Cambronero, Sumit Gulwani, Vu Le, Carina Negreanu, Elnaz Nouri, Mohammad Raza, and Gust Verbruggen. 2023. FormaT5: Abstention and Examples for Conditional Table Formatting with Natural Language. *Proc. VLDB Endow.* 17, 3 (2023), 497–510. <https://www.vldb.org/pvldb/vol17/p497-singh.pdf>
- [37] Mukul Singh, José Cambronero, Sumit Gulwani, Vu Le, Carina Negreanu, and Gust Verbruggen. 2023. CodeFusion: A Pre-trained Diffusion Model for Code Generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 11697–11708. doi:10.18653/v1/2023.emnlp-main.716
- [38] Mukul Singh, José Cambronero, Sumit Gulwani, Vu Le, Carina Negreanu, and Gust Verbruggen. 2023. DataVinci: Learning Syntactic and Semantic String Repairs. arXiv:2308.10922 [cs.DB]
- [39] Mukul Singh, Rahul Kumar Dubey, and Swarup Kumar. 2022. Chapter 15 - Vehicle telematics: An Internet of Things and Big Data approach. In *Artificial Intelligence and Machine Learning for EDGE Computing*, Rajiv Pandey, Sunil Kumar Khatri, Neeraj kumar Singh, and Parul Verma (Eds.). Academic Press, 235–254. doi:10.1016/B978-0-12-824054-0.00019-8
- [40] Mukul Singh, José Cambronero Sánchez, Sumit Gulwani, Vu Le, Carina Negreanu, Mohammad Raza, and Gust Verbruggen. 2023. Cornet: Learning Table Formatting Rules By Example. *Proc. VLDB Endow.* 16, 10 (jun 2023), 2632–2644. doi:10.14778/3603581.3603600
- [41] Mukul Singh, José Cambronero Sanchez, Sumit Gulwani, Vu Le, Carina Negreanu, and Gust Verbruggen. 2023. Cornet: Learning Spreadsheet Formatting Rules by Example. *Proc. VLDB Endow.* 16, 12 (aug 2023), 4058–4061. doi:10.14778/3611540.3611620
- [42] Ananya Singha, Bhavya Chopra, Amirudh Khatri, Sumit Gulwani, Austin Henley, Vu Le, Chris Parnin, Mukul Singh, and Gust Verbruggen. 2024. Semantically Aligned Question and Code Generation for Automated Insight Generation. In *Proceedings of the 1st International Workshop on Large Language Models for Code* (Lisbon, Portugal) (*LLM4Code '24*). Association for Computing Machinery, New York, NY, USA, 127–134. doi:10.1145/3643795.3648381
- [43] Ananya Singha, Bhavya Chopra, Amirudh Khatri, Sumit Gulwani, Austin Henley, Vu Le, Chris Parnin, Mukul Singh, and Gust Verbruggen. 2024. Semantically Aligned Question and Code Generation for Automated Insight Generation. In *LLM4Code Workshop at ICSE '24*. <https://www.microsoft.com/en-us/research/publication/semantically-aligned-question-and-code-generation/>
- [44] Kexuan Sun, Harsha Rayudu, and Jay Pujara. 2021. A Hybrid Probabilistic Approach for Table Understanding. *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 5 (May 2021), 4366–4374. <https://ojs.aaai.org/index.php/AAAI/article/view/16562>
- [45] Qiushi Sun, Nuo Chen, Jianing Wang, Xiang Li, and Ming Gao. 2023. TransCoder: Towards Unified Transferable Code Representation Learning Inspired by Human Skills. arXiv:2306.07285 [cs.SE]
- [46] W3C. 2020. CSS Color Module Level 3. <https://www.w3.org/TR/css-color-3/>
- [47] Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi D. Q. Bui, Junnan Li, and Steven C. H. Hoi. 2023. CodeT5+: Open Code Large Language Models for Code Understanding and Generation. arXiv:2305.07922 [cs.CL]
- [48] 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 (KDD '21)*. Association for Computing Machinery, New York, USA, 1780–1790. doi:10.1145/3447548.3467434
- [49] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain of Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*, Alice H. Oh, Alekh Agarwal, Daniela Belgrave, and Kyunghyun Cho (Eds.). https://openreview.net/forum?id=_VjQlMeSB_J
- [50] Cong Yan and Yeye He. 2020. Auto-Suggest: Learning-to-Recommend Data Preparation Steps Using Data Science Notebooks. In *International Conference on Management of Data (SIGMOD)*. ACM, 1539–1554. <https://www.microsoft.com/en-us/research/publication/auto-suggest-learning-to-recommend-data-preparation-steps-using-data-science-notebooks/>
- [51] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. arXiv:2305.10601 [cs.CL]
- [52] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TabBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 8413–8426. doi:10.18653/v1/2020.acl-main.745
- [53] Wei Zhao, Zhitao Hou, Siyuan Wu, Yan Gao, Haoyu Dong, Yao Wan, Hongyu Zhang, Yulei Sui, and Haidong Zhang. 2024. NL2Formula: Generating Spreadsheet Formulas from Natural Language Queries. arXiv:2402.14853 [cs.CL] <https://arxiv.org/abs/2402.14853>