CORNET: Learning Table Formatting Rules By Example

Mukul Singh
Microsoft
Delhi, India
singhmukul@microsoft.com

José Cambronero Sánchez
Microsoft
New Haven, USA
jcambronero@microsoft.com

Sumit Gulwani
Microsoft
Redmond, USA
sumitg@microsoft.com

Vu Le
Microsoft
Redmond, USA
levu@microsoft.com

Carina Negreanu
Microsoft Research
Cambridge, UK
cnegreanu@microsoft.com

Mohammad Raza*
Microsoft
Redmond, USA
moraza@microsoft.com

Gust Verbruggen
Microsoft
Keerbergen, Belgium
gverbruggen@microsoft.com

ABSTRACT

Spreadsheets are widely used for table manipulation and presentation. Stylistic formatting of these tables is an important property for presentation and analysis. As a result, popular spreadsheet software, such as Excel, supports automatically formatting tables based on rules. Unfortunately, writing such formatting rules can be challenging for users as it requires knowledge of the underlying rule language and data logic. We present CORNET, a system that tackles the novel problem of automatically learning such formatting rules from user-provided formatted cells. CORNET takes inspiration from advances in inductive programming and combines symbolic rule enumeration with a neural ranker to learn conditional formatting rules. To motivate and evaluate our approach, we extracted tables with over 450K unique formatting rules from a corpus of over 1.8M real worksheets. Since we are the first to introduce the task of automatically learning conditional formatting rules, we compare CORNET to a wide range of symbolic and neural baselines adapted from related domains. Our results show that CORNET accurately learns rules across varying setups. Additionally, we show that in some cases CORNET can find rules that are shorter than those written by users and can also discover rules in spreadsheets that users have manually formatted. Furthermore, we present two case studies investigating the generality of our approach by extending CORNET to related data tasks (e.g., filtering) and generalizing to conditional formatting over multiple columns.

PVLDB Reference Format:
Mukul Singh, José Cambronero Sánchez, Sumit Gulwani, Vu Le, Carina Negreanu, Mohammad Raza, and Gust Verbruggen. CORNET: Learning Table Formatting Rules By Example. PVLDB, 16(10): 2632 - 2644, 2023.
doi:10.14778/3603581.3603600

PVLDB Artifact Availability:
The code, data, and/or other artifacts have been made available at https://github.com/microsoft/prose-benchmarks/tree/main/ConditionalFormatting.

*Work performed while at Microsoft

Proceedings of the VLDB Endowment, Vol. 16, No. 10 ISSN 2150-8097.
doi:10.14778/3603581.3603600

1 INTRODUCTION

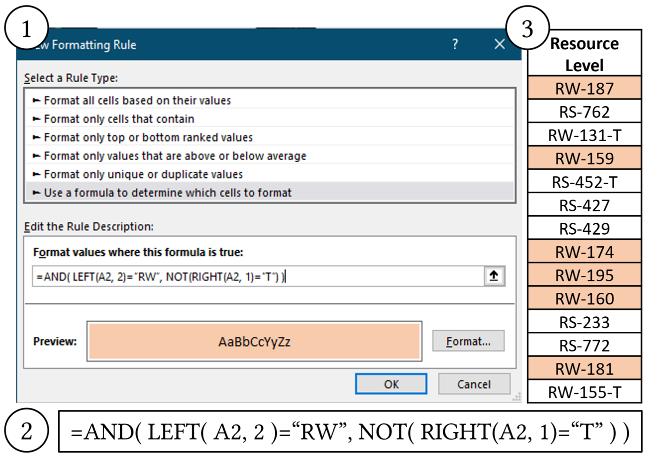
Spreadsheets are the most common table manipulation software, with around a billion monthly active users [26]. Formatting the style of cells is a fundamental and frequently used visual aid to better display, emphasize or distinguish data points in a spreadsheet. By analyzing a large public spreadsheet corpus [2, 14] we found that close to 25% of spreadsheets use some form of cell formatting.

Conditional formatting (CF) is a feature that automates table formatting based on user-defined rules. It is available in all major spreadsheet manipulation tools like Microsoft Excel, Google Sheets and Apple Numbers. All these tools support predefined templates for popular rules, such as *cell value is greater than a specific value*. In Excel and Sheets, users can also author a custom boolean-valued formula to format cells. We find that 18% of spreadsheets in a large public spreadsheet corpus [2] use conditional formatting.

We present the CORNET (**C**onditional **ORN**amentation by **E**xamples in **T**ables) system that automatically generates a formatting rule from examples of formatted cells. CORNET takes a small number of user-formatted cells as input to learn a likely formatting rule that generalizes to other cells in the column. For example, in Figure 1, after the user formats only two cells, CORNET suggests the intended rule without exposing the user to the underlying rule language.

The complexity associated with manually writing conditional formatting rules is reflected in the volume of related help forum posts on the topic. As of June 2022, more than 10,000 conditional formatting related questions were posted on the Excel tech help community alone [10]. By analyzing these posts we discovered multiple factors that contribute to the difficulty of authoring such rules manually. These factors range from fundamental logic challenges in rules to the lack of user interface support in existing platforms. We outline the most prominent factors.

First, many users are unaware of the CF feature and manually format spreadsheets, which can be highly inefficient and introduce errors. Second, even basic rule authoring requires that the user understand the syntax and logic behind conditional formatting, the predefined templates, and potentially the formula language to write more complex rules. Writing such formulas is further complicated by the absence of data type validation. For example, a user can choose numerical comparison on columns with text. This results in wrong formatting or no formatting at all. Third, when users do succeed in writing correct rules, they often write formulas that are more complex than needed to capture their intended logic.


1
w Formatting Rule
Select a Rule Type:
► Format all cells based on their values
► Format only cells that contain
► Format only top or bottom ranked values
► Format only values that are above or below average
► Format only unique or duplicate values
► Use a formula to determine which cells to format
Edit the Rule Description:
Format values where this formula is true:
Preview:
AaBbCcYyZz
Format...
OK
Cancel
2
3

=AND(LEFT(A2, 2)="RW", NOT(RIGHT(A2, 1)="T"))

=AND(LEFT(A2, 2)="RW", NOT(RIGHT(A2, 1)="T"))

Resource Level
RW-187
RS-762
RW-131-T
RW-159
RS-452-T
RS-427
RS-429
RW-174
RW-195
RW-160
RS-233
RS-772
RW-181
RW-155-T

Figure 1: Adding a CF Rule in Excel: User needs to select CF in the "Styles" portion of the home ribbon menu and select add rule from the drop-down menu. The image depicts: ① the dialog box to add a new CF rule; ② the rule the user needs to write ③ the resulting formatted column. After the user formats two cells, CORNET can automatically suggest the intended CF rule for the user.

CORNET is designed to address each of these concerns. First, CORNET can learn conditional formatting rules from as few as one example, opening up the possibility of dynamically suggesting rules to users. Because CORNET can learn rules for a wide range of tasks—about 90% of our benchmarks—users can rely on CORNET to cover a substantial amount of their conditional formatting needs. CORNET only learns rules specific to the data type at hand, removing a substantial cause of incorrect rules. Finally, we found that when users write complex custom rules, CORNET can learn a shorter rule in approximately 60% of the cases.

The key technical challenge in learning conditional formatting rules by example is that few boolean examples (formatted or not) result in a very weak specification: many candidate rules will satisfy them and there is little information to constrain the search space. To tackle this, CORNET first approximates a stronger specification by considering other cells using semi-supervised clustering and assigning them to formatted and unformatted groups—providing a richer signal for pruning the search space. This clustering uses a similarity between cells computed over the same properties that are used to learn candidate rules.

While the approximate specification allows us to navigate the search space, some examples can be noisy and there can still be multiple rules that satisfy them. To handle potentially noisy examples in this approximation, CORNET uses the concept of *soft examples*, which are given less weight during learning. To compensate for the greediness of the learner, in addition to soft examples, CORNET learns multiple candidate rules and ranks them. We propose two rankers: a lightweight ranker based on manual features (and learned weights) and a slightly better but more costly neural ranker.

To train and evaluate CORNET, we created a benchmark of 105K real user tasks from public Excel spreadsheets. We use 80K of these tasks for training and 25K for evaluation. We found that CORNET can learn CF rules from as few as two examples and outperforms existing and custom symbolic and neural baselines that were adapted for this task. In addition, we perform two case studies where we explore how CORNET generalizes to other data tasks and how it generalizes to conditional formatting over multiple columns.

This paper makes the following key contributions:

- Based on the observation that users often struggle to format tabular data, we introduce the novel problem of learning conditional formatting (CF) rules from examples.
- We propose CORNET, a system that learns CF rules from examples over tabular data.
- We create a dataset of 105K real formatting tasks extracted from public spreadsheets. We release this dataset to encourage future research.[1]
- We evaluate CORNET extensively on 25K CF tasks, compare to existing and custom baselines, and show that CORNET outperforms both symbolic and neural baselines by 20% on our benchmark.

2 PROBLEM DEFINITION

Let $C = [c_i]_{i=1}^n$ be a column of n cells with each cell $c_i \in C$ represented by a tuple (v_i, t_i) of its value $v_i \in \mathcal{V}$ and its annotated type $t_i \in \mathcal{T}$. In this paper, we consider string, number, and date as possible types—these are available in most spreadsheet software. We associate a format identifier $f_i \in \mathbb{N}_0$ (or simply format) with each cell, which corresponds to a unique combination of formatting choices made by the user. A special identifier $f_\perp = 0$ is reserved for cells without any specific formatting. In this paper, we consider cell fill color, font color, font size, and cell borders.

EXAMPLE 1. *In Figure 2, which will serve as a running example, colored cells have f_1 and all other cells have $f_\perp$ as format identifiers, where f_1 corresponds to {cell color: #beaed4, font color: default, font size: 12, border: default}.*

A conditional formatting rule (or simply *rule*) is a function $r : C \to \mathbb{N}_0$ that maps a cell to a formatting identifier. Given a column C and specification, a rule r that satisfies the specification is one such that $r(c_i) = f_i$ for all $c_i \in C$

EXAMPLE 2. *Returning to Figure 2, the formatting can be described by the following rule:*

$$r_1(c) = \begin{cases} f_1 & \text{c starts with "RW" and does not end with "T"} \\ f_\perp & \text{otherwise} \end{cases}$$

Let $C_\star = \{c_i \mid c_i \in C, f_i \neq f_\perp\}$ be the cells with formatting applied. The goal of automatic conditional formatting by example is to find r given only a small, observed subset $C_{obs} \subset C_\star$. Throughout this work we will refer to the elements of C_{obs} as *formatted examples*. Any cell in $C \setminus C_{obs}$ is considered unlabelled, which includes all unformatted cells.

EXAMPLE 3. *In Figure 2, the user has provided two examples and $C_{obs} = \{RW\text{-}187, RW\text{-}159\}$). The rest of the cells in the column are unlabeled. The goal is to learn rule r_1 from Example 2.*

[1]To adhere to compliance requirements, we release the data as URLs to xlsx files, that can be downloaded, and scripts to generate benchmarks from the downloaded files.

In the remainder of this paper, we will consider the case where there is only one formatting identifier for simplicity. We then do not have to make assumptions about the order in which a user provides examples for different formats and the order in which the rules for different formats should be applied. Note that we can generalize the single format case to k different formatting identifiers by simply solving k different formatting by example problems, such that when learning the rule for a format identifier f_i, all other format identifiers are treated as $f_\perp$. This approach to multiple formats is closely aligned with popular spreadsheet software, where each format is applied using a different rule. Different rules can overlap and the order in which they are applied, as chosen by the user, determines the final color for each cell. As only 0.63% of rules in our corpus format overlapping cells, we do not consider overlapping rules and their order.

3 APPROACH

This section describes how Cornet learns formatting rules from a small number of user-provided examples. Figure 2 shows a schematic overview of Cornet's approach. Step ① enumerates properties of cells as predicates. Step ② approximates the expected output using semi-supervised clustering. Cornet then iteratively generates rules that match this output in step ③, and ranks them in step ④. The following sections describe challenges and solutions for each step.

3.1 Predicate Generation

Cornet uses cell properties to reason about the target formatting. This step enumerates a set of these properties that hold for a non-empty proper subset of the cells of the given column. Each property is encoded as a predicate—a boolean-valued function that takes a cell c along with zero or more additional arguments and returns true if the property that it describes holds for the cell c. To avoid type errors, all predicates are assigned a type and they only match cells of their type. Supported predicates are shown in Table 1. The predicates for Cornet have been chosen based on formatting rule operations supported by popular spreadsheet software.

For each predicate, we need to generate constant values for all additional (not c) arguments. Given a column of cells and a predicate, the goal is to initialize each additional argument to a constant value such that the predicate returns true for a non-empty proper subset of cells in the column. We do this by generating a set of constant values for each type, derived from the column values or common constants, and instantiating each predicate with combinations of constants of the appropriate types. Table 2 shows an overview of how the constant values are generated for predicates of each type.

Example 4. *For the topmost cell of the column in Figure 2 and TextContain(c, s), we generate three constants for s. The first is simply the whole cell value (RW-187). Splitting the cell on non-alphanumeric characters obtains tokens {RW, -, 187}. As TextContain(c, "-") is true for all cells in the column, this is not considered and dropped. We get*

{*TextContain(c,"RW-187"), TextContain(c,"RW"), TextContain(c,"187")*}

as the three generated predicates from TextContains(c, s).

Table 1: Supported predicates and their arguments for each datatype (top) and also general (bottom). The d argument in datetime predicates determines which part of the date is compared—day, month, year, or weekday. For example, greater(c, 2, month) matches datetime cells with a date in March or later for any year.

Numeric	Datetime	Text
greater(c, n)	greater(c, n, d)	equals(c, s)
greaterEquals(c, n)	greaterEquals(c, n, d)	contains(c, s)
less(c, n)	less(c, n, d)	startsWith(c, s)
lessEquals(c, n)	lessEquals(c, n, d)	endsWith(c, s)
between(c, n_1, n_2)	between(c, n_1, n_2, d)	length(c, n)
numDigits(c, n)	workDay(c)	hasDigits(c)
isInteger(c)	weekNum(c, n_4)	hasSpecial(c)
isPercentage(c)		isEmail(c)
isCurrency(c)		isUrl(c)
isEven(c)		
isOdd(c)		
	General	
isNum(c)	isError(c)	isFormula(c)
isLogical(c)	isNA(c)	isText(c)

Table 2: Overview of constants for concretizing predicates of each type. For example, we generate constants for text predicates from two token sources: delimiter-based splitting and prefixes.

Type	Arg(s)	Values
numeric	n	all numbers that occur in the column
numeric	n	summary statistics: mean, min, max, and percentiles
numeric	n	popular constants such as 0, 1 and 10^n
numeric	n_1 and n_2	use numeric generators for n and keep the ones $n_1 < n_2$
text	s	whole cell value
text	s	tokens obtained by splitting on non-alphanumeric delimiters
text	s	tokens from prefix trie
date	n and d	for available d, extract numeric value and use generator for n

3.2 Semi-supervised Clustering

Rather than immediately combine predicates into rules, we first predict the expected output of possible rules on the unformatted cells by clustering. There are 2^n ways to cluster a column of n cells in two clusters (formatted and unformatted) but 2^{2^p} unique rules can be written with p predicates.[2] In other words, many rules yield the same clustering. Clustering then allows us to leverage the relatively small search space of output configurations to find programs that generalize to similar cells. Cornet biases the predicted output towards the generated predicates by using their output to compute

[2]Conjunction and disjunction can represent all boolean functions over p inputs, and there are 2^{2^p} such functions.

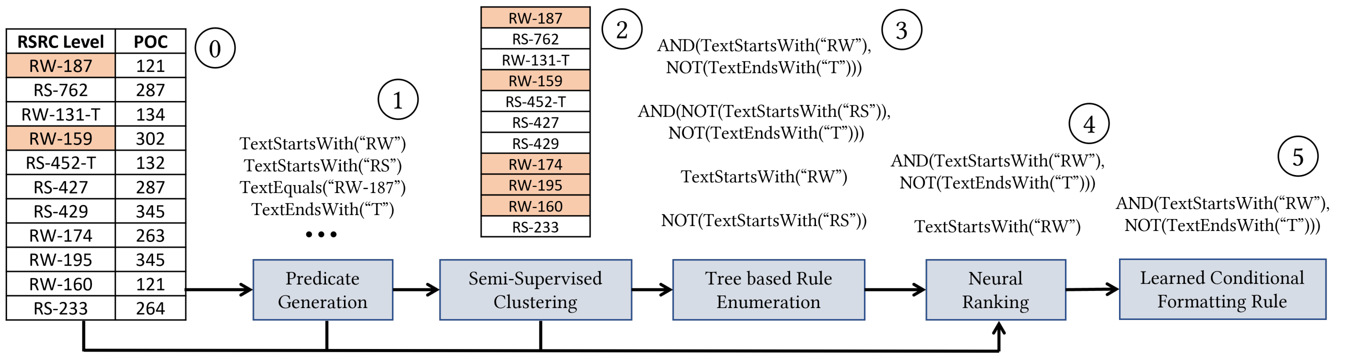


Figure 2: Cornet architecture illustrated through the example case from Figure 1: ⓪ input table with partial formatting, ① predicate generation for all cells in the table, ② semi-supervised clustering using examples and other cells to address the challenge of unlabeled cells, ③ enumerating rules based on the clustering using multiple decision trees, ④ neural ranker to score generated rules, and ⑤ final learned conditional formatting rule.

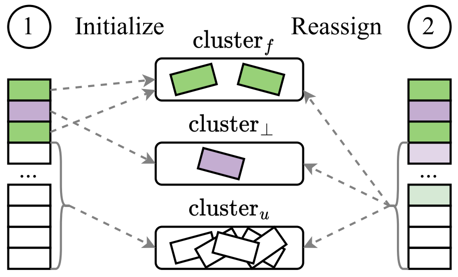


Figure 3: Schematic overview of the initialize and reassign steps for clustering cells into formatted (f), unformatted ($\perp$) and unassigned (u) clusters. Only initially unassigned cells are reassigned and obtain a noisy label when this happens.

the similarity between cells. FlashProfile [30] uses the same concept with regular expressions to learn syntactic profiles of data.

More concretely, we assign a (potentially noisy) formatting label $\hat{f}_i$ to each unobserved cell $c_i \notin C_{obs}$ by building on two insights. First, tables are typically annotated by users from top to bottom, which implies that there is positional information available. In particular, cells $c_i \notin C_{obs}$ such that there exist $c_j, c_k \in C_{obs}$ for which $j < i < k$ are likely intended to have no formatting associated with them. We refer to this set of c_i as *soft negative examples* [33]. Second, user-provided examples C_{obs} should be treated as *hard positive examples*—we assume that the user does not make errors, which is a common assumption in PBE [15]. During learning, hard examples *have* to be correctly classified. The learner tries to optimize the accuracy on soft examples, subject to a threshold on the maximum number of nodes in the tree to account for rule complexity.

We perform iterative clustering over three clusters of formatted (cluster$_f$), unformatted (cluster$_\perp$) and unassigned (cluster$_u$) cells. Some supervision is introduced by initializing each cell $c_i \in C_{\text{obs}}$ to cluster$_f$ and soft negative example cells to cluster$_\perp$. These cells are never assigned to another cluster. The remaining cells C_u are assigned to cluster$_u$. Taking inspiration from k-medoids [19] we iteratively reassign $c_u \in C_u$ to a new cluster. Figure 3 shows a schematic overview of initialization and reassignment. Instead of computing a cluster medoid, however, we average the minimal and maximal distance to any element of the cluster, which was found to perform well in practice. The distance between two cells is the size of the symmetric difference between the sets of predicates that hold for either cell. When clusters become stable or a maximum number of iterations is reached, cells c_i from cluster$_u$ and cluster$_\perp$ are assigned $\hat{f}_i = f_\perp$ (soft negative examples). Cells c_i from cluster$_f$ are assigned $\hat{f}_i = f$ (soft and hard positive examples).

3.3 Candidate Rule Enumeration

After clustering, we have a target formatting label $\hat{f}_i$ for each c_i in C. We now learn a rule r such that $r_f(c_i) = \hat{f}_i$ for all $c_i \in C_{obs}$ (hard examples) and $\sum[r_f(c_i) = \hat{f}_i]$ for $c_i \notin C_{obs}$ is greedily maximized (soft examples). We define the space of rules and a search procedure in the following two subsections.

3.3.1 Predicates to Rules. A rule in Cornet for a column C and format f is a function $r_f : C \to \mathbb{B}$ that takes a cell and returns whether the cell should be formatted as f or not. Cornet supports r_f that can be built as a propositional formula in disjunctive normal form over predicates. In other words, every r_f is of the form

$$(p_1(c) \wedge p_2(c) \wedge \dots) \vee (p_j(c) \wedge p_{j+1}(c) \wedge \dots) \vee \dots$$

with p_i a generated predicate or its negation. Our goal is to strike a balance between expressiveness and simplicity.

3.3.2 Enumerating Rules. We greedily enumerate candidate rules by iteratively learning decision trees that predict the noisy label $\hat{f}_i$ for each c_i from their predicate outputs. Each decision tree then corresponds to a rule in disjunctive normal form [3]. We identify and address three challenges: variety in rules, simplicity of rules, and coping with noisy labels. To induce variety in rules, during iteration we remove the predicate associated with the root node of the current tree from possible predicates, so any subsequent trees learned cannot use it. To induce simplicity of rules, we limit the depth of learned decision trees to λ_n or fewer nodes. To cope with our noisy (cluster-based) labels, we 1) weigh labeled cells twice as much as unlabeled ones to bias our tree towards perfectly classifying user-provided examples (and we perform a check after the tree is learned to ensure this property), and 2) we allow misclassifications for soft-labeled (i.e. originally unassigned) cells by allowing accuracy to fall up to λ_a. This learning procedure is schematically shown in Figure 4.

3.3.3 Setting parameters. λ_n limits the complexity of generated rules. We set $\lambda_n = 10$ as only 17 rules in our 105K corpus had more than 10 predicates. λ_a filters rules with lower accuracy. We set $\lambda_a = 0.8$ as CF rules tend to have lower noise tolerance.

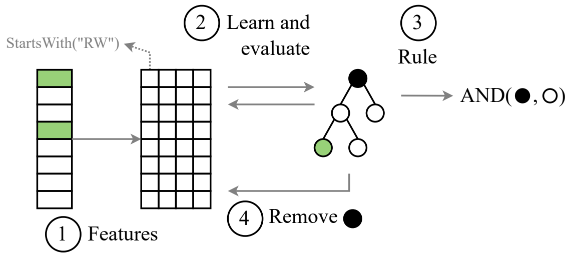


Figure 4: Schematic overview of iterative rule learning. Steps ② until ④ are repeated as long as the decision tree achieves the desired accuracy and there are features remaining.

3.4 Candidate Rule Ranking

To choose a final rule from candidates generated by the iterative learner we rank candidates. Prior work has proposed ranking programs based on output features [27] or rule features [9]. We build on these approaches and develop a neural ranker that combines information from both types of features.

Information about the rule is captured by handpicked features: depth of rule, count of predicates used, number of arguments, mean length of arguments, number of cells and data type of column, percentage of cells that satisfy the rule, accuracy with respect to approximated labels ($\hat{f}$) and source of constant arguments (cell value, statistical value, popular constant, delimiter or trie).

Information about the column data is captured by turning the column into a sequence of words and using a pre-trained language model (BERT) [6] to obtain cell-level embeddings. These embeddings are augmented with information about the execution of the rule through cross-attention [21].

Both the rule and data representations previously described are concatenated and passed to a linear layer with sigmoid activation to produce a single score. This score thus combines both syntactic (rule) and semantic (data and execution) information. Figure 5 shows an overview of our ranking architecture.

We train the ranking model by treating this problem as binary classification of the correctness of learned rules and we use the output of the final linear layer after sigmoid activation as the rule score. To generate training data we apply CORNET up to the rule enumeration step using 1, 3, and 5 examples on a held-out dataset of columns with ground-truth conditional formatting rules. We keep rules that do *not* match the user rule as negative samples and rules that *do* match the user rule as positive examples. Additionally, we apply user rules on other columns to obtain both positive (by construction) and negative (by the procedure above) examples. This process results in 174K examples for our ranking model.

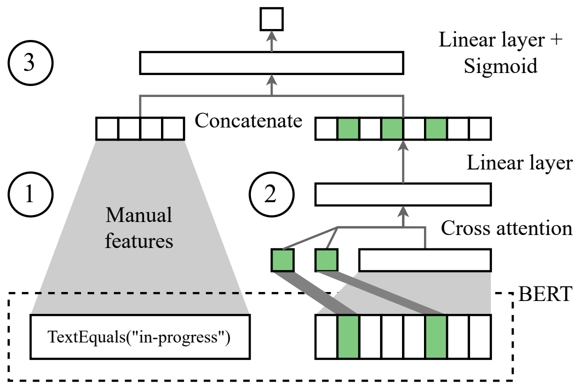


Figure 5: Ranking model: ① model inputs - rule to be scored and the corresponding data column; ② the column encoding model pools BERT token embeddings, passes them through cross attention with the rule's execution outputs (formatted or not), and then through a linear layer; and ③ the resulting embedding is concatenated with manually-engineered rule features and fed into a final linear layer which outputs the score after applying a sigmoid activation.

4 BASELINES

As we are the first to introduce the problem of learning conditional formatting rules from examples, there are no existing systems that tackle this task. We therefore adapt a variety of approaches related to this problem. Six approaches are symbolic, five of which are able to generate rules. Three neural approaches cast conditional formatting as cell classification and we consider different baseline models and cross-attention mechanisms. The following sections describe these baselines in more detail. We focus on the case where we have a single format identifier.

4.1 Symbolic

4.1.1 Decision trees. We fit a decision tree with formatted and unformatted cells as positive and negative examples, respectively. We consider two variations of encoding cells. In the first one, raw cell values are passed to the decision tree, where text columns are categorically encoded. This encoding does not allow learning rules that involve partial strings, summary statistics for numbers or date parts. In the second encoding, we therefore use the outputs of our generated predicates as features for cells. In the latter case, we perform an additional improvement by allowing the splitting criterion to use our ranker when impurity is equal across different predicates. There are then three decision tree baselines in total.

4.1.2 ILP. We cast conditional formatting as an inductive logic programming (ILP) problem over the same grammar of rules as CORNET. This requires examples (both positive and negative) and background knowledge as input and learns a program that satisfies the examples using the background knowledge. In our setting, the background knowledge consists of the grammar and the constants extracted from the column. Again, we consider two variants by using raw cell values and by augmenting the grammar to use our generated predicates. We use POPPER [5], a state-of-the-art ILP tool.

Example 5. *Consider a numerical column with values [7, 6, 3, 4]. An excerpt of the background knowledge is*

```
LessThan(A, B) :- A < B.
const1(7). const2(6). const3(3). const4(4).
```

where the first line defines a predicate and the second line defines constants that the predicate can use. We define col (A) *as the predicate to be learned and give* col (3) *and* col (6) *as a positive and negative example, respectively. The program produced by* POPPER *is*

```
col(A)  :=  LessThan(A, B).
B       :=  const4(4).
```

4.1.3 *Constrained Clustering.* Conditional formatting can be treated as a constrained (cell) clustering problem where clusters must respect the provided formatted examples. COP-KMeans is a k-means based clustering strategy that supports linkage constraints for clusters [37]. Besides a distance function between cells and the number of clusters, it also takes *must-link* e^+ and *cannot-link* e^- constraints as input. We use the size of the symmetric difference between the sets of predicates that hold for two cells to measure their distance. The formatted examples and the implicit negative examples are used to populate e^+ and e^-. All pairs of formatted cells and pairs of negative cells are in e^+. All pairs consisting of a formatted and an implicit negative example are in e^-. For example, in Figure 5, e^+ contains the positive pair (RW-187, RW-159) and the negative pair (RS-762, RW-131-T). The mixed pair (RW-187, RS-762) is in e^-.

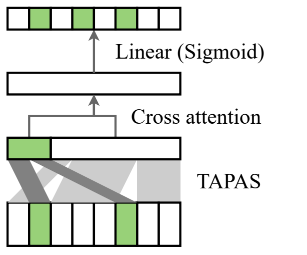
4.2 Neural

To build neural baselines, we frame conditional formatting as a table/cell classification problem and pick state-of-the-art models from this domain. Two of these neural approaches are based on table embedding models and one is built on top of a language model. All neural baselines are fine-tuned on our 80K training tasks.

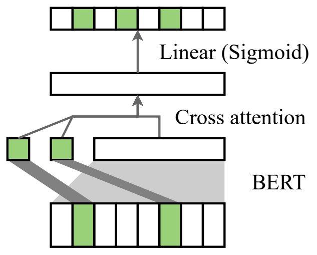
4.2.1 *TAPAS.* TAPAS [17] is a table encoding model trained for sequential question answering (SQA). We apply it to conditional formatting by using it to encode the input column and getting an embedding for each cell and applying cross-attention between the formatted cells and the rest of the column. A linear layer followed by a sigmoid activation is used to make a prediction (formatted or unformatted) for each cell. Figure 6 (a) describes the architecture.

4.2.2 *TUTA.* TUTA [38] is a tree based transformer model that is pre-trained on multiple table-related objectives. One of the downstream tasks it has been fine-tuned for is cell type classification (CTC). TUTA uses cell values in a table along with their position, data type and formatting information to predict the role of a cell. By considering formattings as cell types, we fine-tune it to predict the format of each cell from a partially annotated column.

4.2.3 *BERT.* Finally, we use an architecture similar to the TAPAS baseline, but use the BERT language model [6] to produce column embeddings. Each cell in a column is tokenized, the tokens for different cells are concatenated with a separator token in between,



(a) Baseline using TAPAS table embedding model.



(b) Neural baseline with BERT. Cells are tokenized, embedded and average pooled.

Figure 6: Two custom neural baseline architectures in our evaluation. We cast conditional formatting as cell classification. Green cells represent formatted examples. TAPAS directly encodes all formatted cells to a single embedding while BERT does a cell level embedding.

this sequence of tokens is embedded, and cell-level embeddings are obtained by average pooling. Tokenization and average pooling is also used to obtain individual cell embeddings for the positive examples. A cross attention layer, where the full column provides queries (Q) and formatted cells provide keys (K) and values (V), is used to combine these embeddings—a thorough discussion on attention in transformers is given in [36]. Finally, a linear layer followed by a sigmoid activation converts the cross embedding output to predictions for each cell. Figure 6 (b) shows the architecture.

5 EVALUATION

We perform experiments to answer the following questions:

- **Q1.** Is CORNET able to quickly and correctly learn conditional formatting rules from few examples?
- **Q2.** How do our design decisions (clustering, iterative learning and ranking) impact learning time and correctness?
- **Q3.** How do properties of the input table (number of examples, row order and column type) impact learning?
- **Q4.** Can CORNET learn rules that are shorter than those authored by users?
- **Q5.** Can CORNET learn rules for spreadsheets that users formatted manually?

Additionally, we investigate two dimensions of generality – generalizing to related data tasks and generalizing to complex multi-column conditional formatting scenarios – through case studies:

- **CS1.** We apply CORNET to two related problems: data filtering and learning the conditions for conditional formulas (`IF`).
- **CS2.** We evaluate CORNET on the rare cases where the condition spans multiple columns (0.9% of cases in our CF rule corpus).

5.0.1 *Benchmarks.* To train and evaluate CORNET, we leveraged a corpus of 1.8 million publicly available Excel workbooks from the web. Among these, 236.5K workbooks contain at least one CF rule added by users. In total, we extracted 410.6K CF rules and their corresponding cell values and formatting. We deduplicate files by filename, sheets by column headers and rules by exact syntactic

Table 3: Average properties of benchmark problems divided by type. 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
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**

match. Furthermore, we remove rules that operate on less than five cells, format the entire column or only format a single cell. After deduplicating and filtering, we retain 105K tasks where a task consists of a (formatted) column and the associated CF rule. Table 3 shows a summary of the benchmarks. We split the 105k tasks into a train set of 80K, which we use for training CORNET and baselines, and a test set of 25K tasks, which we use for evaluation. [3]

5.0.2 Evaluation Metrics. To evaluate learned rules against user-written rules, we consider three metrics: exact match, execution match and cell match. *Exact match* is a syntactic match between a learned rule and the user-written rule, with tolerance for differences arising from white space and alternative argument order, which do not impact execution. *Execution match* consists of executing two rules and comparing the formatting produced —there is an execution match if the formatting are identical. In addition to capturing the fact that different rules can produce the same formatting, execution match allows us to evaluate against baselines that directly predict formatting rather than produce rules. This distinction between exact and execution match is also made in related areas, such as natural language to code [23, 31]. We also report cell-level precision and recall over the predicted and the ground truth formatting. These are micro-averaged over all tasks. We refer to these two metrics as *Cell Match.*

EXAMPLE 6. *Because they are equivalent after removing spaces and swapping (equivalent) argument order, OR(Equals(10), Equals(20)) and OR(Equals(20),Equals(10)) are an exact match. On the other hand, TextStartsWith("D12") and TextContains("D12") are not an exact match because the rules are not equivalent. These may be an execution match on a column that only has "D12" at the start of values.*

5.1 Q1. Performance

Table 4 presents an overview of our results. CORNET outperforms symbolic and neural baselines on both exact and execution match metrics. Both POPPER and decision tree methods perform worse than CORNET even when provided with CORNET's predicates. TUTA is the only neural model that is competitive with symbolic methods—likely due to being trained for cell type classification. However, TUTA does not do well at capturing syntactic patterns. and as a result does not perform as well as CORNET. CORNET is the only approach that achieves, on average, both high recall (97.8%) and high precision (92.2%). Symbolic methods achieve high precision but low recall, indicating poor generalization. Neural methods achieve high recall, but low precision, indicating too aggressive generalization.

[3]The benchmarks are released and can be accessed at github.com/microsoft/prose-benchmarks/tree/main/ConditionalFormatting

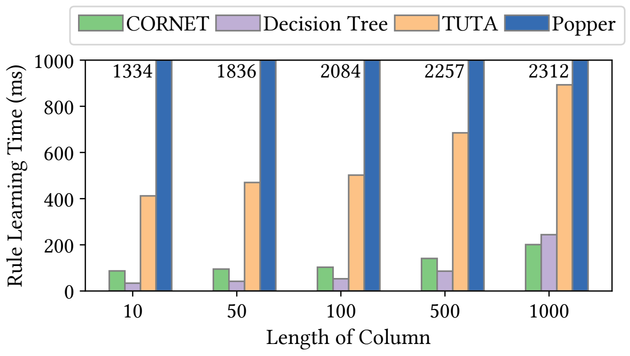


Figure 7: Rule learning time in milliseconds plotted against the number of cells in a column. We compare CORNET with the fastest and best (execution match) symbolic methods (decision tree and POPPER, respectively) and the fastest and best (execution match) neural method (TUTA). CORNET is faster than both TUTA and POPPER by over half a second.

5.1.1 Limitations of Baselines. Our symbolic baselines are limited to learning a single rule, whereas CORNET learns multiple rules and then uses more context to rank them. Neural models are heavily dependent on tokenization and mainly appear to capture semantic properties. This makes them less effective in cases that require identifying syntactic patterns, which is often the case for CF rules. In rare cases, this ability to capture the semantic meaning of text gives neural models an advantage over CORNET. For example, TUTA is able to color cells that contain *High* or *Medium* even though the single formatted example provided is *High*, whereas CORNET would only color *High.* A second advantage of neural models is that they are not bound by our conditional formatting grammar and can support some scenarios that require arbitrary Excel formulas. While CORNET does not support such cases, our analysis shows they are rare in practice (377 cases in our full corpus).

5.1.2 Execution Time. We also evaluate the time required by each system to predict formatting as a function of the number of cells in the target column. Figure 7 shows the average time taken to predict a rule as a function of the increasing number of cells in a column. We show results for CORNET, the fastest (decision tree) baseline and the symbolic (POPPER) and neural baselines (TUTA) with highest execution match. Learning multiple shallow decision trees (CORNET) is faster than learning one large one. TUTA is backed by a medium-sized neural network (110M parameter) that makes inference slow in our testing environment, which has resources beyond those that a target CF user would typically have. POPPER is the slowest out of these baselines as the hypothesis space quickly explodes as a result of predicate generation for different cells.

5.1.3 Memory Consumption. Table 5 shows the total amount of disk space required to store a system, and the average CPU and GPU memory used for prediction over benchmarks for CORNET and neural baselines. CORNET uses BERT for ranking, hence their

Table 4: Comparison of Cornet with neural and symbolic baselines. We report exact and execution match for 1, 3 and 5 user formatted examples. We report cell match using 3 formatted examples. "Rules" denotes if an approach generates symbolic rules. Cornet outperforms neural and symbolic baselines in both execution and exact rule match. Cornet achieves high precision and recall, given 3 formatted examples, compared to symbolic/neural systems which have low recall/precision, respectively.

System description			Execution match			Exact match			Cell match (3 ex.)	
Name	**Technique**	**Rules**	**1 ex.**	**3 ex.**	**5 ex.**	**1 ex.**	**3 ex.**	**5 ex.**	**Precision**	**Recall**
Decision Tree	Symbolic	Yes	47.2	58.3	63.2	20.3	27.2	31.1	93.1	69.7
Decision Tree + Predicates	Symbolic	Yes	55.5	66.9	71.7	40.2	49.1	50.6	91.3	76.4
Decision Tree + Predicates + Ranking	Symbolic	Yes	56.1	68.7	73.5	43.8	51.5	52.9	92.5	84.6
Popper	Symbolic	Yes	56.2	63.4	67.8	45.6	53.5	57.1	94.0	77.0
Popper + Predicates	Symbolic	Yes	58.3	68.9	74.1	46.1	54.2	57.8	95.0	80.4
Constrained Clustering	Symbolic	No	51.7	61.9	66.4	–	–	–	79.3	89.2
TUTA for Cell Type Classification	Neural	No	57.4	66.1	69.3	–	–	–	85.3	**92.8**
TAPAS + Cell Classification	Neural	No	44.3	55.8	59.4	–	–	–	77.2	83.5
BERT + Cell Classification	Neural	No	40.6	54.9	60.2	–	–	–	73.4	81.0
Cornet	**Neuro-symbolic**	**Yes**	**66.3**	**78.2**	**82.8**	**50.8**	**59.8**	**63.2**	**97.8**	92.2

Table 5: Total disk space and average memory used in megabytes for inference over benchmark tasks for Cornet and neural baselines. Cornet needs GPU resources because it uses a neural ranker based on BERT. Cornet with symbolic ranker is lightweight with a disk space of only 2.7MB.

System	Disk Space	CPU Memory	GPU Memory
TUTA	722.4	5.4	1335.2
TAPAS	443.6	2.3	1416.8
BERT	416.7	4.4	775.9
Cornet	419.2	21.3	804.2
Cornet Symbolic	2.7	32.4	0.0

Table 6: Execution match for the top rule with 1, 3 and 5 examples, average number of candidates and learning time (in milliseconds) for different clustering configurations.

Model	1 ex.	3 ex.	5 ex.	candidates	t (ms)
No clustering	58.7	74.4	79.3	122.7	104
No neg. examples	61.8	75.3	80.5	42.2	152
Hard neg. examples	63.7	76.6	81.9	20.1	174
Cornet	**66.3**	**78.2**	**82.8**	22.5	187

similar footprint. The overhead for the rest of Cornet is minimal—with a symbolic ranker it uses only 2.7MB of memory. Section 5.2.3 describes the symbolic ranker for low-resource environments.

5.2 Q2. Impact of Design Decisions

We discuss the impact of the three main components in Cornet: semi-supervised clustering, iterative rule learning, and ranking.

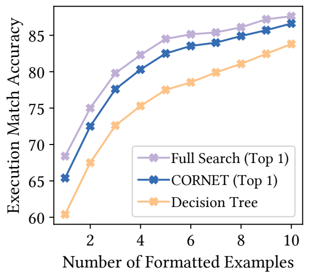
5.2.1 Clustering. First, we carry out experiments with three different versions of our clustering approach and show the results in Table 6. First, *no clustering* removes the semi-supervised clustering step altogether. It considers user formatted cells to be positive examples and *all* unlabeled cells to be negative examples. Note that this ablation can still learn rules (with worse performance) because the iterative tree learning procedure in Cornet only requires satisfying the user formatted examples and tolerates noise in other examples through the accuracy threshold during learning. Second, we consider a version of clustering where there are only two clusters: one for user formatted cells (positive examples) and one for all unassigned examples. Upon termination, all cells still in the unassigned cluster are relabeled as negative examples. We label this *no negative examples* in our results table. Third, we consider a version that only has *hard negative* examples by setting the weight of labeled and unlabeled cells equal during iterative tree learning—see Section 3.3 for details.

Table 6 shows accuracy and number of candidate rules for each of these clustering versions. We find that clustering reduces the number of candidates by 80%, which allows ranking to select a better rule. Not using negative examples drops performance with 1 example by 4.5%. Hard negatives constrain the search space too much with a 2.6% failure rate in finding desired rule with 1 example.

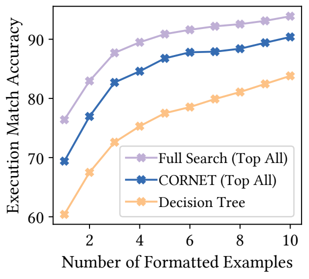
5.2.2 Iterative Rule Learning. Iterative learning allows Cornet to learn multiple candidate rules and then rank them separately. However, this iterative procedure is greedy and as a result is not complete—it only considers a subset of all possible rules. To evaluate the extent to which this impacts performance, we compared our greedy approach to an iterative full search up to tree depth 5.

In Figure 8, we compare the top-1 and top-all execution match accuracy for iterative greedy search (Cornet), a single decision tree and an exhaustive search with a maximal depth of five. As expected, Cornet is slightly less expressive and loses about 3% execution match accuracy on average against the full search, but this effect reduces as more examples are given.

In Figure 9, we compare the learning time for Cornet, single decision tree and the exhaustive search strategy as a function of the depth of the rule. Our results indicate that Cornet can be up to 40x to 80x faster than an exhaustive search, despite the small decrease in execution match accuracy as shown in Figure 8.



(a) Execution Match on Top-1



(b) Execution match on Top-All

Figure 8: (a) Top-1 and (b) top-all execution match accuracy for increasing number of examples for CORNET, a decision tree and an exhaustive search. CORNET sacrifices only 3% and 8% in top-1 and top-all execution match accuracy, respectively, compared to a depth-bounded (to 5) exhaustive search.

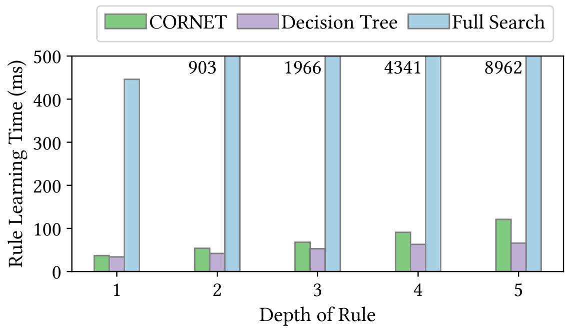


Figure 9: Rule learning time in milliseconds for increasingly deeper rules. We compare CORNET with a single decision tree and a bounded depth exhaustive search. CORNET is much faster than the exhaustive search and scales better as the depth of the target rule grows.

5.2.3 Ranking. Finally, we compare the neural ranker with two ablated versions: a purely symbolic ranker that only uses a linear combination of the handpicked features, and a purely neural ranker that replaces the handpicked features with a CodeBERT [13] encoding of the formatting rule. Table 7 shows that combining both sources of information outperforms both ablated versions. Note that the symbolic ranker is only about 4% worse than the combined ranker while requiring significantly less computation and memory, as it does not use a large neural model. This symbolic ranker is a good alternative in resource constrained domains.

5.3 Q3. Impact of Input Configuration

The exact input to CORNET has an effect on its performance. We thus study how different properties of this input, like the number of formatted examples, order of examples, and number of unformatted cells, affect the performance of CORNET.

First, the number of examples that a user provides influences the accuracy. Ideally, this influence diminishes after a certain number of examples. Figure 10 shows this dependency on the provided number of examples, which varies significantly across data types. For text, two examples is sufficient for more than 90% of the cases. For numbers, performance steadily improves until 15 examples are provided. We hypothesise that more examples are needed in the numeric cases because constants in numeric rules are harder to learn—examples close to the decision boundary are needed, which might only appear lower in the column. When suggesting rules to users, we can thus be more conservative in numeric columns. Note that rules for text columns are on average longer than those for numbers (2.9 predicates versus 1.6) and we can more quickly suggest rules in cases that may be harder for the user.

Table 7: Execution match within top-k candidates with 3 formatted examples for different ranking models. Top-all represents the performance of an oracle ranker. #pm shows the number of trainable parameters in the model. CORNET outperforms both ablated versions.

Ranker	#pm	top-1	top-3	top-5	top-10	top-all
Symbolic	45	73.4	74.4	75.2	75.8	84.3
Neural	124M	74.4	76.1	76.9	79.4	84.3
CORNET	1.7M	**78.2**	**80.4**	**81.8**	**82.8**	**84.3**

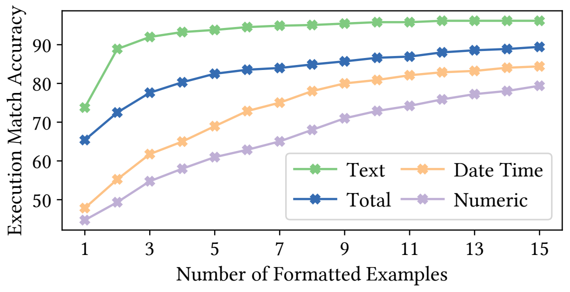


Figure 10: Execution match over the number of formatted examples for different column data types. CORNET has higher accuracy for Text and DateTime columns. Numeric columns require more examples, given the larger search space.

Second, we investigate the impact of the number of unformatted cells on performance. Less data, and thus fewer unformatted cells, might be available when deploying systems like CORNET in browsers or on mobile devices. Our aim is to estimate the minimum number of unformatted cells needed for acceptable performance. Figure 11 shows how accuracy increases with the number of unformatted cells for different numbers of formatted cells. Performance gains diminish after more than 20 unformatted cells, across settings which provide 1, 3, and 5 formatted examples.

Third, we evaluate the effect of the order in which the user provides examples. To do so, we take each formatting task and randomly shuffle the formatted (positive) rows in the column five times to create five random orderings. For each shuffled task, we apply CORNET to an increasing number of formatted examples to learn a rule. We compute three statistics from this. First, we compute an *all-shuffles* execution match accuracy, which is the fraction of tasks where CORNET achieves execution match in all

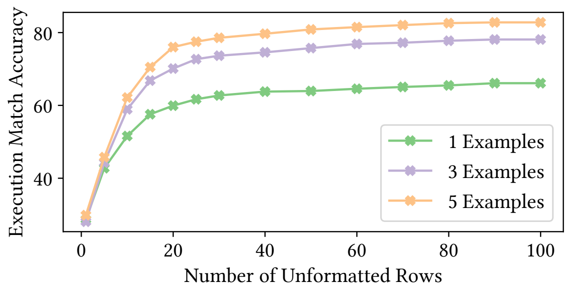


Figure 11: Execution match over the number of unformatted rows for different number of formatted examples given. Cornet is able to generalize with as few as 20 unformatted examples after which performance stabilizes.

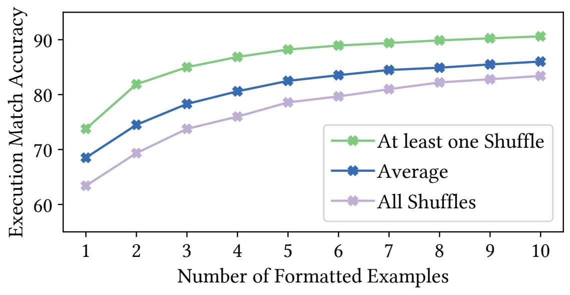


Figure 12: Execution match in our shuffling experiments. We report execution match for tasks where Cornet achieves execution match in *all shuffles, at least one shuffle*, and on average. We find that formatted example order impacts execution match accuracy, but the average performance is comparable to that achieved with the original user's cell order.

five shuffled orderings. Second, we compute an *at-least-one-shuffle* execution match accuracy, which is the fraction of tasks where Cornet achieves execution match in at least one shuffled ordering. Finally, we report an *average* execution match accuracy where we simply report the fraction of tasks and orderings where Cornet learns a rule with the correct execution match.

Figure 12 reports the results over these shuffling experiments. We found that there is a 9% difference between the *all-shuffles* and *at-least-one-shuffle* execution match accuracy at three formatted examples, showing that there can indeed be an effect in the ordering of formatted examples. However, the original example order—used in all other experiments for this work—roughly aligns with the average accuracy found in these shuffling experiments.

5.4 Q4. Simplicity of Rules

When comparing execution match and exact match, we find that these metrics are roughly 20% apart for any given amount of examples. This suggests that Cornet learns rules that are syntactically different from rules that users write, while resulting in the same formatting. Our experiments show that often, Cornet actually learns a simpler rule. We use rule length as a proxy for simplicity, as shorter

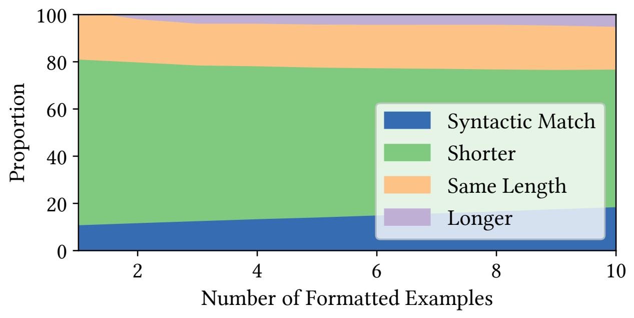


Figure 13: Comparing the rules learned by Cornet against user rules for tasks where the user wrote a custom conditional formatting formula (rather than choose a predefined template), we find that Cornet produces shorter rules in approximately 60% of the cases.

rules are easier to interpret, write, and maintain. This notion of length-based simplicity has been used in prior PBE systems [8].

We treat all functions, operators and arguments as individual tokens and define the length of the rule as the associated count of tokens. For example, IF(A1="Not Applicable", TRUE, FALSE) consists of tokens {IF, =, "Not Applicable", TRUE, FALSE} and thus has a length of 5. Similarly, GreaterThan(10) has a length of 2.

In Figure 13, we consider all tasks where the user wrote a custom conditional formatting formula—not a predefined template—and we compare lengths of these formulas with the rules learned by Cornet. We find that in the majority of cases (~60%) Cornet learns shorter rules, while maintaining execution match. As more examples are given, Cornet seems to learn comparatively longer rules. This happens because tasks that need more examples to be solved are more likely to require a (longer) complex rule.

We also found that reductions in formula length can be substantial: for complex rules, where we need up to 5 examples to learn a rule, the Cornet rule can be on average up to 65% shorter than the user-written rule. Figure 14 shows the average formula length reduction as a function of the length of the original user formula. In cases where Cornet requires more examples, rules are more complex and Cornet can provide greater reductions. This suggests that Cornet can be used for rule refactoring as well.

Some concrete examples of user rules and the associated Cornet rules are shown in Table 8. When Cornet learns a shorter rule, the user has often resorted to a custom formula instead of using a built-in predicate. When the length is the same, Cornet either uses the same predicate with a different constant or a different predicate with the same constant. For different constants, due to enumeration, Cornet yields less precise numbers (10 versus 10.5). For different predicates, due to ranking, Cornet is generally more conservative and yields more specific rules (Equals versus Contains).

5.5 Q5. Manual (re)Formatting

Not all users are aware of CF and manually format spreadsheets. We study the extent to which Cornet can help with discoverability of this feature. We analyze manually formatted columns. From our corpus of spreadsheets, we sample 100K columns with at least 5

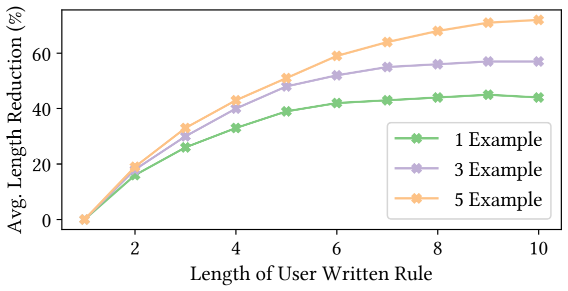


Figure 14: Average reduction length (in %) achieved by CORNET's rule, as a function of user-written rule length, given 1, 3, and 5 examples for cases with execution match. With more examples, CORNET achieves execution match for more complex rules, which it can simplify to a greater extent.

Table 8: Examples comparing rules generated by CORNET to user written rules. The cases shown are where CORNET produces the correct execution and simplifies the rule, learns a different rule of the same length or learns a longer rule.

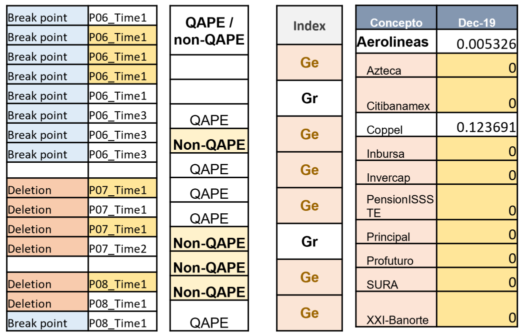
Length	CORNET	Gold Rule
Shorter	TextStartsWith("Dr")	IF(LEFT(A1,2)="Dr",TRUE,FALSE)
	GreaterThan(5)	IF(NOT(A1<=5), TRUE)
	TextContains("Pass")	ISNUMBER(SEARCH("Pass",A1))
Equal	TextEquals("Aramco")	TextContains("Aramco")
	GreaterThan(10)	GreaterThan(10.5)
	TextEndsWith("ARM")	TextContains("_ARM")
Longer	OR(Equal(0),Equal(1))	NOT(Equal(-1))
	NOT(TextEquals("OK"))	TextContains("Not")

non-empty cells, of which at least 3 have a custom background color applied without CF. Figure 15 shows some examples.

First, we provide CORNET with all formatted cells to learn a rule. If the learned rule has fewer predicates than formatted cells, we heuristically label this as a case where the user could have written a rule. We find 93.4K such columns. This distribution is shown in Figure 16a. Our results show that 80% of these learned rules have 3 or fewer predicates, which may benefit interpretability. Next, for all the columns identified, we find the minimal number of examples the user could have given to CORNET to obtain their desired formatting. The distribution of minimum number of examples needed for these cases is shown in Figure 16b. We find that CORNET learns over 90% of the rules with 4 or fewer examples.

5.6 CS1: Filtering and Conditional Formulas

Generating predicates and learning binary classifiers from few examples can be applied to related problems. We explore two such related problems in tabular data: filtering rows of data and learning the condition for conditional formulas. Filtering maps naturally to CF, as hidden and visible cells can be treated as unformatted and formatted, respectively. Similarly, we can map the true and false branches of a conditional formula to formatted and unformatted.



Break point	P06_Time1
Break point	P06_Time1
Break point	P06_Time1
Break point	P06_Time1
Break point	P06_Time1
Break point	P06_Time3
Break point	P06_Time3
Break point	P06_Time3
Deletion	P07_Time1
Deletion	P07_Time1
Deletion	P07_Time1
Deletion	P07_Time2
Deletion	P08_Time1
Deletion	P08_Time1
Break point	P08_Time1

Contains("Time1")

QAPE / non-QAPE
QAPE
Non-QAPE
QAPE
QAPE
QAPE
Non-QAPE
Non-QAPE
Non-QAPE
QAPE

Begins("Non")

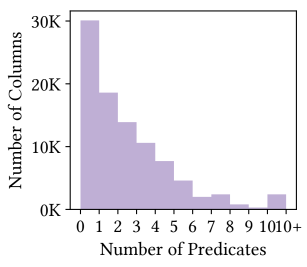
Index
Ge
Gr
Ge
Ge
Ge
Gr
Ge
Ge

Equals("Ge")

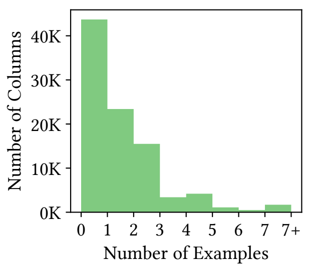
Concepto	Dec-19
Aerolineas	0.005326
Azteca	0
Citibanamex	0
Coppel	0.123691
Inbursa	0
Invercap	0
PensionISSSTE	0
Principal	0
Profuturo	0
SURA	0
XXI-Banorte	0

Equals(0)

Figure 15: Examples of columns with manual cell formatting but no CF Rules along with CORNET's learned rule (below)



(a) Predicates



(b) Examples

Figure 16: Histograms showing (a) the number of predicates in the CF rule learned by CORNET for users' manual formatting and (b) the minimum number of examples needed by CORNET to learn that CF rule. CORNET is able to learn more than 90% of these rules with fewer than 4 examples.

We consider data filtering in both Excel (from our corpus and StackOverflow) and Python (from StackOverflow). We sourced 330 filters that users created through the spreadsheet interface from a sample of 20K workbooks, 33 `FILTER` formulas from StackOverflow posts, which either use the interface or a custom `FILTER`, and 87 Python (Pandas) filtering tasks from StackOverflow.

We sampled 250 conditional formulas from the same subset of spreadsheets used to sample filtering tasks. The conditional formulas we consider are of the form `=IF(`c`,` a`,` b`)` where a and b do not contain nested `IF` operators, and where c only mentions a single column. Learning conditional (c) in formulas is an important problem studied extensively in programming-by-example [15].

Results for both problems are shown in Figure 17. We found that the filtering tasks resulted in higher performance than learning formula conditionals. We believe this is a result of our grammar, which is well aligned with the predicates available in the filtering interface and those often used in the `FILTER` function. In contrast, conditional formulas may use different functions, as well as use numeric conditions more often. Extending CORNET's grammar with such predicates could help mitigate this performance difference.

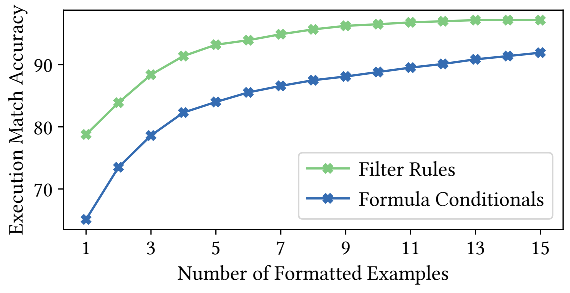


Figure 17: Case study: CORNET performance for filtering and learning conditions as a function of the number of examples. Performance is consistently higher for filtering tasks.

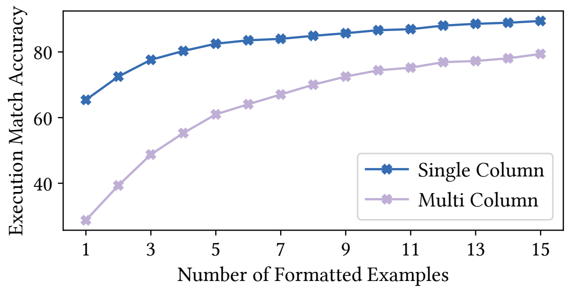


Figure 18: Case study: CORNET performance on rules over multiple columns. More examples are required for the same performance, if the rule involves multiple columns.

5.7 CS2: Conditions over Multiple columns

We extended CORNET to learn conditions over multiple columns. In this extension, predicates that accept two arguments can instantiate those arguments over cells that are in the same row but in different columns. For example, if c_i^A and c_i^B are the i^{th} cell in column A and B, respectively, then greater(c_i^A, c_i^B) is a valid predicate. We introduce an optimizing assumption to keep the number of predicates linear in the number of columns considered: that one of the cells in pair predicates *must* be in the column being formatted. Note that multi-column CF rules are relatively uncommon: in our corpus of 410.6K rules, we found 3647 (0.9%) that required more than one column.

Figure 18 summarizes the performance of CORNET for single and multi-column cases. For learning rules over multiple columns, the number of examples needed for high performance increases. This is expected as the number of candidates in the search space and rule complexity is exponentially higher for multi-column cases.

6 RELATED WORK

Despite the large spreadsheet userbase, there have been relatively few formal studies on CF. [28] gives detailed coverage of how this feature works in the context of Excel. [1] discusses how CF in Excel can improve the demonstration of mathematical concepts.

Recent progress in automatic table formatting includes [7] which describes CellGAN, a conditional Generative Adversarial Network model which learns the hierarchical headers and data groups in tables. Other work like [18, 24] focus on formatting cells based on table structure and cell sizes. In contrast, CORNET targets example based data formatting, and generates the associated formatting rule.

CORNET uses a programming-by-example (PBE) paradigm, which has been popularized by systems like FlashFill [15] and FlashExtract [20]. These systems, which are available in Excel, learn string transformation and data extraction programs from few input-output examples. Popper [5] is another popular inductive logic programming (ILP) system for learning programs by specifying examples.

In terms of search techniques, [25, 32] use goal-driven top-down symbolic backpropagation. This is not applicable in our setting because the boolean signal (i.e., is a cell formatted) is too-weak to derive strong-enough constraints to navigate the search space. A popular alternative in PBE is bottom-up enumeration [12, 29, 34], which is infeasible in our setting because of the large search space. The notion of re-interpretation in [16] finds outputs and programs in separate DSLs. In contrast, CORNET first hypothesizes the outputs (cell formats) and then learns the associated rule. CORNET is the first system to take an "output-first" synthesis approach motivated by the fact that our output space is smaller than program space.

Past work on using PBE systems on databases have shown great success in the domain of querying [11, 22, 25] and data understanding and cleaning [12]. CORNET builds upon these systems to solve the problem of data formatting. Past PBE work has ranked programs using program features [9, 25] or output features [22, 27]. CORNET uses both program and output features for ranking programs.

Neural approaches have previously been applied in various table tasks. For example, TaBERT [39] and TAPAS [17] are popular Question Answering systems that use a neural model to encode the table and query. TUTA [38] is a weakly supervised model for cell and table type classification tasks. SpreadsheetCoder [4] is a predictive system for learning spreadsheet formulas from table context. TabNet [35] predicts cell types using a neuro-symbolic model. Unlike these systems, CORNET targets learning formatting rules.

7 CONCLUSION

In this paper, we introduced the novel problem of learning conditional formatting rules from user examples. We proposed CORNET, a system that learns such data-dependent rules from few examples. To evaluate CORNET, we created a benchmark of 105K CF tasks extracted from 1.8 million real Excel spreadsheets. To facilitate future research into this novel problem, we release our set of benchmarks. To effectively evaluate CORNET, we compare performance to baselines by framing the problem as an ILP task, a clustering task, a cell classification task and a table fine tuning task. We also create custom neural and symbolic baselines for a more comprehensive comparison and result analysis. We study how CORNET can help with discoverability of CF and learn shorter CF rules. Finally, we present two case studies applying CORNET to other related data tasks and multi-column formatting. CORNET opens future work like predictive CF rule learning and conditional data transformations.

ACKNOWLEDGMENTS

We would like to thank Yair Helman, Almog-Ben Kandi, Sophie Gerzie, Avital Nevo, Israela Solomon, Christian Poelitz and Yoav Hayun for their feedback on this research.

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] Titus Barik, Kevin Lubick, Justin Smith, John Slankas, and Emerson Murphy-Hill. 2015. Fuse: a reproducible, extendable, internet-scale corpus of spreadsheets. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, IEEE/ACM, Florence, Italy, 486–489.

[3] Hendrik Blockeel and Luc De Raedt. 1998. Top-down induction of first-order logical decision trees. *Artificial intelligence* 101, 1-2 (1998), 285–297.

[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)*, Marina Meila and Tong Zhang (Eds.), Vol. 139. PMLR, virtual, 1661–1672. https://proceedings.mlr.press/v139/chen21m.html

[5] Andrew Cropper and Rolf Morel. 2021. Learning Programs by Learning from Failures. *Mach. Learn.* 110, 4 (Apr 2021), 801–856. https://doi.org/10.1007/s10994-020-05934-z

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, Minneapolis, USA, 4171–4186. https://doi.org/10.18653/v1/n19-1423

[7] 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. https://doi.org/10.1145/3340531.3411943

[8] Ian Drosos, Titus Barik, Philip J. Guo, Robert DeLine, and Sumit Gulwani. 2020. Wrex: A Unified Programming-by-Example Interaction for Synthesizing Readable Code for Data Scientists. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3313831.3376442

[9] 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-learn-programs-examples-going-beyond-program-structure/

[10] Microsoft Excel. 2022. Excel Tech Help Forum. https://techcommunity.microsoft.com/t5/forums/searchpage/tab/message?q=conditional%20formatting. Last Accessed: 2022-06-30.

[11] 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. https://doi.org/10.14778/3342263.3342266

[12] 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. https://doi.org/10.1145/3448016.3452750

[13] 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. https://doi.org/10.18653/v1/2020.findings-emnlp.139

[14] Marc Fisher and Gregg Rothermel. 2005. The EUSES spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms. In *Proceedings of the first workshop on End-user software engineering*. Association for Computing Machinery, New York, NY, USA, 1–5.

[15] 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/

[16] Sumit Gulwani, Vu Le, Arjun Radhakrishna, Ivan Radicek, and Mohammad Raza. 2020. Structure interpretation of text formats. In *Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)*. ACM, Association for Computing Machinery, New York, NY, USA, 29. https://www.microsoft.com/en-us/research/publication/structure-interpretation-of-text-formats/

[17] 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/

[18] 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.

[19] Leonard Kaufman and Peter J Rousseeuw. 2009. *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, online.

[20] 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/

[21] Kuang-Huei Lee, Xi Chen, Gang Hua, Houdong Hu, and Xiaodong He. 2018. Stacked Cross Attention for Image-Text Matching. In *Computer Vision – ECCV 2018*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer International Publishing, Cham, 212–228.

[22] 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. https://doi.org/10.14778/2831360.2831369

[23] 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.

[24] Xiaofan Lin. 2006. Active layout engine: Algorithms and applications in variable data printing. *Computer-Aided Design* 38, 5 (2006), 444–456.

[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. https://doi.org/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] Erich Neuwirth and Deane Arganbright. 2003. *The Active Modeler: Mathematical Modeling With Microsoft Excel*. Duxbury Press, online.

[29] Augustus Odena, Kensen Shi, David Bieber, Rishabh Singh, and Charles Sutton. 2021. BUSTLE: Bottom-up program-Synthesis Through Learning-guided Exploration. *ArXiv* abs/2007.14381 (2021).

[30] Saswat Padhi, Prateek Jain, Daniel Perelman, Oleksandr Polozov, Sumit Gulwani, and Todd D. Millstein. 2017. FlashProfile: Interactive Synthesis of Syntactic Profiles. *CoRR* abs/1709.05725, Article 150 (2017), 28 pages. arXiv:1709.05725 http://arxiv.org/abs/1709.05725

[31] Gabriel Poesia, Oleksandr Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. 2022. Synchromesh: Reliable code generation from pre-trained language models. *CoRR* abs/2201.11227 (2022). arXiv:2201.11227 https://arxiv.org/abs/2201.11227

[32] Oleksandr Polozov and Sumit Gulwani. 2015. FlashMeta: A Framework for Inductive Program Synthesis. *SIGPLAN Not.* 50, 10 (oct 2015), 107–126. https://doi.org/10.1145/2858965.2814310

[33] Mohammad Raza and Sumit Gulwani. 2020. Web data extraction using hybrid program synthesis: A combination of top-down and bottom-up inference. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, New York, NY, USA, 1967–1978.

[34] Yanyan Shen, Kaushik Chakrabarti, Surajit Chaudhuri, Bolin Ding, and Lev Novik. 2014. Discovering Queries Based on Example Tuples. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14)*. Association for Computing Machinery, New York, NY, USA, 493–504. https://doi.org/10.1145/2588555.2593664

[35] 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

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[37] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. 2001. Constrained K-Means Clustering with Background Knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML '01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 577–584.

[38] 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 &amp; Data Mining (KDD '21)*. Association for Computing Machinery, New York, USA, 1780–1790. https://doi.org/10.1145/3447548.3467434

[39] 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*. Association for Computational Linguistics, Online, 8413–8426. https://doi.org/10.18653/v1/2020.acl-main.745