FormaT5: Abstention and Examples for Conditional Table Formatting with Natural Language

Mukul Singh
Microsoft
Delhi, India
singhmukul@microsoft.com

José Cambronero
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

Elnaz Nouri
Microsoft Research
Redmond, USA
elnaz.nouri@microsoft.com

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

Gust Verbruggen
Microsoft
Keerbergen, Belgium
gverbruggen@microsoft.com

ABSTRACT

Formatting is an important property in tables for visualization, presentation, and analysis. Spreadsheet software allows users to automatically format their tables by writing data-dependent conditional formatting (CF) rules. Writing such rules is often challenging for users as it requires understanding and implementing the underlying logic. We present FormaT5, a transformer-based model that can generate a CF rule given the target table and a natural language description of the desired formatting logic. We find that user descriptions for these tasks are often under-specified or ambiguous, making it harder for code generation systems to accurately learn the desired rule in a single step. To tackle this problem of under-specification and minimise argument errors, FormaT5 learns to predict placeholders though an abstention objective. These placeholders can then be filled by a second model or, when examples of rows that should be formatted are available, by a programming-by-example system. To evaluate FormaT5 on diverse and real scenarios, we create an extensive benchmark of 1053 CF tasks, containing real-world descriptions collected from four different sources. We release our benchmarks to encourage research in this area. Abstention and filling allow FormaT5 to outperform 8 different neural approaches on our benchmarks, both with and without examples. Our results illustrate the value of building domain-specific learning systems.

PVLDB Reference Format:
Mukul Singh, José Cambronero, Sumit Gulwani, Vu Le, Carina Negreanu, Elnaz Nouri, Mohammad Raza, and Gust Verbruggen. FormaT5: Abstention and Examples for Conditional Table Formatting with Natural Language. PVLDB, 17(3): 497 - 510, 2023.
doi:10.14778/3632093.3632111

[*]Work performed while at Microsoft

Proceedings of the VLDB Endowment, Vol. 17, No. 3 ISSN 2150-8097.
doi:10.14778/3632093.3632111

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

1 INTRODUCTION

Conditional formatting (CF) allows users to automatically format data by writing *rules* that describe when specific formatting styles should be applied to cells. CF is commonly used for data analysis and presentation, and is supported by spreadsheet software (like Microsoft Excel and Google Sheets), data analysis software (like Pandas), and data visualization software (like Tableau).

A recent analysis [46] shows that CF is present in 18% of spreadsheets from two popular public corpora [3, 16]. Unfortunately, users struggle to write CF rules as it requires knowledge of the underlying data logic and rule language. The aim of FormaT5 is to allow users to go from *their intended formatting* to *a CF rule that achieves this formatting* as intuitively as possible. We therefore study how to obtain CF rules from natural language descriptions of the desired formatting (utterances).

Example 1. *An example table and the intended formatting is shown in Figure 1 (a). Logical conjunction and string comparison are required to write the rule. Our system takes the table (a) and the natural language instruction (b) and generates the rule (c).*

An important challenge in parsing utterances is inferring constant values required in rules. Recent work has explored learning formatting rules *by example* [46]. We show how these complementary forms of intent—natural language and examples—can be combined to learn rules with the correct constants.

Our approach, called FormaT5, uses an encoder-decoder transformer to generate the rule from a flat representation of the input data (table, utterance). There are three main challenges: (1) teaching the model link tabular data with code, (2) using examples when appropriate, and (3) obtaining training data. We now describe how we tackle these challenges.

Name	Marks	ID
Student 1	73	A-Elem-293
Student 2	68	A-Mid-124
Student 3	81	Elem-394
Student 4	77	Mid-493

(a)

"Highlight Students who have Middle School ID and have scored above Average"

(b)

```
AND(GreaterThan("Marks", AVERAGE("Marks")),
    TextContains("ID", "Mid"))
```

(c)

Figure 1: A real conditional formatting task from an Excel tutorial. (a) shows the table with an optionally highlighted row (as a single example) that should be formatted, (b) is a natural language utterance for the desired formatting rule, (c) is the target formatting rule, which would also correctly color Student 4's row.

To address the first challenge, FormaT5 first learns to relate tabular data and code in a pre-training step, and then learns to incorporate natural language in a fine-tuning step. Text-to-code systems [21, 37, 44] often ignore the data programs are meant to execute on (typically due to lack of availability). Table models [23, 53] have been developed to answer natural language questions over tables, but are not trained on code. SpreadsheetCoder [7] or FORTAP [9] encode table data to generate formulas, but do not include natural language.

To address the second challenge—using examples and learning when examples are required—we allow the model to generate placeholder tokens in the predicted rule and use a separate approach to populate the placeholders. An abstention loss objective teaches the model to only predict placeholders when tokens cannot be inferred from the utterance and table information. Based on the available information, we either use a separate model to generate constant values (given utterance only) or adapt the example-based rule learning system from Cornet [46] to generate plausible candidates (given utterance and examples).

To address the third challenge, we use pre-trained large language models (Codex and GPT-3) to bootstrap training examples from a few manually annotated code snippets. Additionally, pre-training on table and rule pairs and then fine-tuning the model to incorporate natural language (see the first challenge) allows us to leverage the availability of large amounts of rules without natural language utterances, as opposed to the comparatively small amount of rules with utterances.

To evaluate FormaT5, we gather 1053 tasks from four different sources (crowdsourced annotation, Stack Overflow posts, online tutorials, and sampled spreadsheets we annotate). Each task consists of an utterance, a table and a formatting rule. We release these benchmarks to encourage future research in this area.

We compare FormaT5 to a wide range of neural baselines, including text models, text-to-code models, table models and a table-to-code model. Our experiments show that FormaT5 outperforms the next best system by 2.4–10.3% using an exact match criterion, depending on the type of task. We also demonstrate the effectiveness of some design choices, including placeholder insertions, pre-training objectives, and constrained decoding [37].

Learning conditional formatting rules is related to row selection and learning database queries (like SQL) from natural language, but there are three key differences. First, conditional formatting allows specialized formatting operators (such as data bar, colour scale and icon sets). Second, natural language utterances in the conditional formatting domain are often underspecified. Third, conditional formatting formulas are focused on (aggregate) numerical properties and specific string properties (like prefixes and suffixes). Specializing to these, FormaT5 avoids a more complex search space and can learn from few (or no) user-provided examples.

We make the following contributions:

- We present FormaT5, a CF rule generation system trained on code, natural language and tables.
- We construct a benchmark set of 1053 table formatting problems from various sources and of varying difficulty. We release this benchmark for future research.
- We carry out an extensive evaluation against existing text-to-code and table models, showing that FormaT5 outperforms prior approaches across our benchmark tasks, both without and with examples.

2 RELATED WORK

Table Formatting. Despite its popularity, there is little work on generating table formatting rules. [46] proposes Cornet, a system that learns table formatting rules by examples. Our work is complementary, as it focuses on natural language and uses examples sparingly—when there is ambiguity in the utterance. CellGAN [13] focuses on borders and alignment of cells to learn hierarchical headers and data groups in tables. It uses an end-to-end approach to learn formatting directly from a large number of formatted sheets. Other work [8, 24, 30] focuses on formatting cells based on table structure (like headers and partitions) and cell sizes. Neither of these approaches focus on conditional formatting rules.

Text-to-Code Systems. A substantial body of work has explored the use of transformer-based models for general code tasks. For example, CodeBERT (BERT), CodeT5 (T5) and Codex (GPT-3) have been fine-tuned for text-to-code tasks for multiple domains including SQL, pandas and Python. In our work, we use a similar (T5) approach to learn to generate conditional formatting rules, but we also consider the input table in addition to the utterance.

Comparison to Text-to-SQL. CF rules are related to row selection and may seem similar to query languages like SQL. However, three key differences. First, CF supports specialized formatting operators (such as data bar, colour scale and icon sets [14]) that account for 24.35% of our corpus of 410K rules – these are unsupported by traditional text-to-SQL systems [37, 44]. Simply fine-tuning text-to-SQL methods on our dataset, which includes these new operators, does not solve the challenge as we demonstrate experimentally. Second, utterances in CF are often under-specified and ambiguous, for example, an utterance may simply state to "highlight" rows but not specify the stylistic properties. FormaT5 introduces abstention along with learning from examples as a complementary modality to tackle these underspecified utterances. Third, conditional formatting formulas are not as complicated as general where clauses in SQL. For example, we found that many conditions in formatting are focused on (aggregate) numerical properties and specific string properties (like prefixes and suffixes) [46], while SQL where clauses can include general subqueries. Specializing FormaT5 to conditional formatting allows us to avoid a more complex search space and learn from few (or no) user-provided examples. Despite these differences, because row selection (SQL) is a related problem, our evaluation considers 5 text-to-SQL baselines: TAPAS[23], TaBERT[53], ValueNet[5], PICARD[44] (which achieved state-of-the-art top-1 execution accuracy on the Spider benchmark), and Synchromesh[37]. FormaT5's results demonstrate the value of building domain-specific learning system for the task of conditional formatting.

Learning Queries for Databases. In the field of databases, initial efforts to create natural language based systems involved ontologies, intermediate languages, and various heuristics for selecting join paths [27, 43, 45]. However, with the emergence of transformer models [48], the process of adding such capabilities to databases has become simpler. The Spider leaderboard [2] presents a comparison of machine learning-based approaches for text-to-SQL conversion, which are evaluated based on the Spider dataset [54]. These approaches can be broadly classified into three categories: customized ML models like ValueNet [5], pre-trained language models such as GPT-3 [37, 38], and fine-tuned large language models [44, 49].

Generating Placeholders. Learning to insert placeholders has previously been found to improve code completion [19]. We use a similar approach to insert placeholders in rules when the utterance is underspecified or the model fails to learn. We show how these placeholders can be resolved by a dedicated deep learning model or a symbolic programming-by-example approach.

Learning over Tables. Table transformer models like TaBERT [53], TAPAS [23] and TAPEX [31] learn a joint representation of table data and query, and use this to predict the answer of a query. These systems are designed to extract answers from tables in the form of individual cells or aggregation over a range of cells. These systems are not designed for code generation. TUTA [50] is another table pre-training system that uses tree-based attention to encode tables and has been fine tuned for table type classification and cell type classification tasks. [35] performs data tasks like cleaning and imputation by framing these as natural language tasks over tables. They extend foundation models for data tasks by serializing tables and framing data tasks into *Yes* and *No* text generation prompts.

Programming By Example. FormaT5 integrates NL-based generation with programming-by-example (PBE). PBE has been popularized by the likes of FlashFill and FlashExtract [18, 26]. FlashFill learns string transformation programs from few input-output examples while FlashExtract is a general framework for tabular data extraction by examples. Because of their ease of use, they have been integrated into commercial software—FlashFill and FlashExtract are available in Excel. Popper [10] is another popular inductive logic programming (ILP) framework for learning programs by specifying examples and constraints. [46] is recent work that uses PBE to learn CF rules. FormaT5 can use a PBE system (such as Cornet) to fill the placeholders inserted when the utterance is underspecified or the model fails to learn.

Multi-modal Synthesis. Recent work in code generation has explored combining NL and examples to design multimodal systems. [51] provides examples to LLM in the prompt while [40] uses NL to generate components with an LLM and then uses bottom-up synthesis to generate programs. In contrast to these, FormaT5 uses NL to learn a partial program that has placeholders for under-specification or ambiguity, leveraging examples or a second NL-based value filling model to fill these holes, generating the complete program.

3 PRELIMINARIES

A CF task consists of the input table T, an utterance Q, optionally some examples E and the target formatting rule R. Given input (T, Q, E) the goal is to find R.

The input table T consists of a collection of columns that are described by their header and data (list of cells). Each cell has a value and type, where the type is one of date, number or string. We define the type of the column as the most general type in the column, where string $\supset$ number $\supset$ date. For example, a column where cells have type number or type date has column type number.

Examples E are given in the form of a (possibly empty) list of row indices where each example is treated as a *positive instance*—the given example is expected to be formatted by the correct rule.

Rules R consist of logical propositions that are evaluated at the level of the row—a rule cannot partially format a row.

4 FORMAT5

FormaT5 builds on the encoder–decoder architecture of T5, which has been shown to be effective for code generation and code understanding [39]. We pre-train FormaT5 on (T, R) tuples without an NL utterance—of which we have an abundance—and then fine-tuned on (T, R, Q) triples—which are harder to obtain.

4.1 Encoding T, R and Q

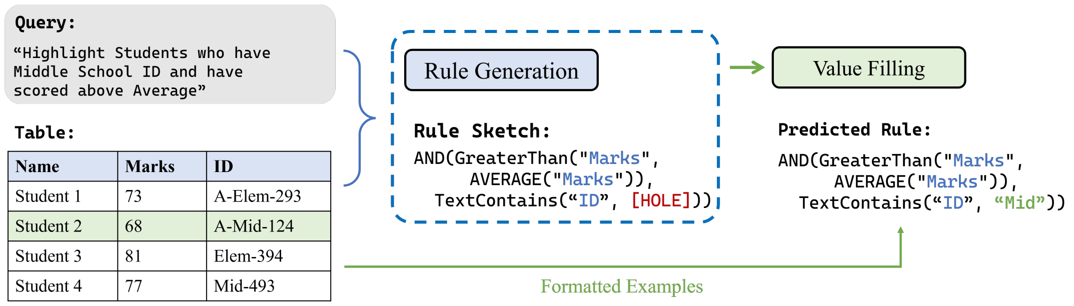
We encode the table and rule by building a flat, textual representation and then using the T5 tokenizer. Rather than encode the whole table, we represent it with its headers and column type information.[1] Columns are encoded from left to right and concatenated by a `[COL]` separator token. For example, the table in Figure 1 is flattened to "`Name string [COL] Marks number [COL] ID string`". All special tokens (between square brackets) are added as special tokens to the tokenizer.

A rule is interpreted as a string according to the Excel syntax, with boolean connectives in prefix notation (`AND(a, b)`). The utterance is directly provided to the T5 tokenizer. The table and utterance are joined by a `[SEP]` token.

4.2 Pre-training on T and R

We gather a large corpus of 410K table and rule pairs from a collection of public Excel worksheets. These pairs do not have a natural language utterance, so we use them to pre-train FormaT5 on the following objectives.

[1]Experiments show that including data values does not improve performance–see Table 7.



Query:

```
“Highlight Students who have
Middle School ID and have
scored above Average”
```

Table:

Name	Marks	ID
Student 1	73	A-Elem-293
Student 2	68	A-Mid-124
Student 3	81	Elem-394
Student 4	77	Mid-493

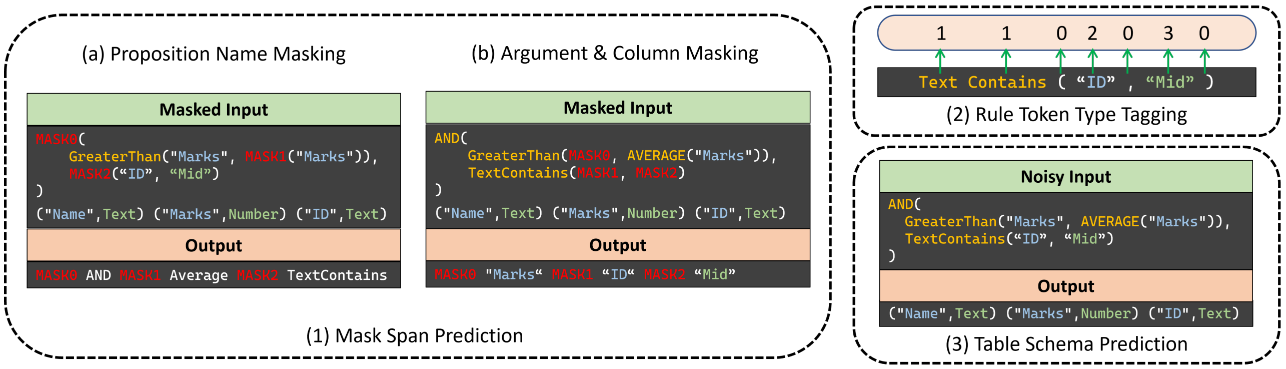
Rule Sketch:

```
AND(GreaterThan("Marks",
        AVERAGE("Marks")),
   TextContains(“ID”, [HOLE]))
```

Predicted Rule:

```
AND(GreaterThan("Marks",
        AVERAGE("Marks")),
   TextContains(“ID”, “Mid”))
```

Figure 2: FormaT5 architecture summary. Given a natural language description and the associated table, FormaT5 uses an abstention based generation that generates a sketch rule that can have holes for constants. In the value filling step, these holes are filled using examples, when available or using an NL-only model that leverages the original description. In this case, the user provided a single row as an example (Student 2) and the generated rule correctly colors remaining rows (Student 4).


(a) Proposition Name Masking
(b) Argument & Column Masking
(1) Mask Span Prediction
(2) Rule Token Type Tagging
(3) Table Schema Prediction
1 1 0 2 0 3 0

Masked Input

```
MASK0(
    GreaterThan("Marks", MASK1("Marks")),
    MASK2(“ID”, “Mid”)
)
("Name",Text) ("Marks",Number) ("ID",Text)
```

Output

```
MASK0 AND MASK1 Average MASK2 TextContains
```

Masked Input

```
AND(
    GreaterThan(MASK0, AVERAGE("Marks")),
    TextContains(MASK1, MASK2)
)
("Name",Text) ("Marks",Number) ("ID",Text)
```

Output

```
MASK0 "Marks“ MASK1 “ID“ MASK2 “Mid”
```

```
Text Contains ( “ID” , “Mid” )
```

Noisy Input

```
AND(
  GreaterThan("Marks", AVERAGE("Marks")),
  TextContains(“ID”, “Mid”)
)
```

Output

```
("Name",Text) ("Marks",Number) ("ID",Text)
```

Figure 3: FormaT5's pre-training tasks: (1) two variants of Mask Span Prediction—recover (a) masked proposition names and (b) masked arguments and column names; (2) Rule token type tagging—predict the indices representing different classes {0: syntax tokens, 1: proposition name, 2: column name, 3: argument}; (3) Table schema prediction—recover the table schema.

Masked Span Prediction (MSP). We use two variations of the masked span prediction (MSP) objective during pre-training, which has been shown to be effective for sequence-to-sequence tasks [25] like code generation [17]. In one variant, we randomly mask 50% of tokens associated with proposition names (like TextStartsWith). In the other variant, we randomly mask 50% of tokens associated with proposition arguments (constants or references to table column names). As usual in generative models, masked tokens are predicted by teaching the model to generate (mask token, replacement) pairs. This is shown in Figure 3 (a) and (b).

Rule Token Type Tagging. We adapt the identifier prediction objective [49] to predict the type of each token in a tokenized rule. Possible types are proposition name, proposition arguments, column types and syntax tokens (like parentheses and quotes). A classification head is added to the encoder and we train with standard cross entropy loss. This is shown in Figure 3 (2).

Table Schema Prediction. Given a rule, we task the model with generating the table schema for the columns that it references. The schema generated corresponds to (column name, column type) tuples. This is shown in Figure 3 (3).

4.3 Bootstrapping Data

As conditional formatting rules with natural language utterance annotations are scarce, we generate synthetic data using Codex (code-davinci-002) [6] and GPT-3.5 (text-davinci-003) [4]. First, we use Codex (code-davinci-002) to translate rules into utterances, and then we use GPT-3.5 (text-davinci-003) to paraphrase the utterances. Bootstrapping training data using language models [32] and paraphrasing for diversity has been studied in prior work [22, 33].

We collected 50 rules that jointly contain at least one instance of each proposition name from the rule language. We annotate each rule with an associated natural language utterance. We then prompt Codex to generate utterances for rules without an utterance by using examples from the manually annotated set. We provide 3 examples which are selected using cosine similarity over CodeBERT [15] embeddings of the rules.

We find that even with multiple runs at a high temperature, the generated utterances lack diversity and have similar sentence structure for rules that use the same propositions. To induce more diversity in our dataset, we use GPT-3.5 to paraphrase the utterances that were originally generated by Codex.

We find that the paraphrased queries are much more diverse and realistic. After annotation and paraphrasing, we finally are able to generate a total of 105K (T, Q, R) triples. These are used for fine-tuning FormaT5 and all baselines.

4.4 Fine-tuning on T, R and Q with Abstention

We describe teaching FormaT5 to insert placeholders using the abstention loss objective and describe the fine tuning process.

4.4.1 Abstention Loss. We want to teach FormaT5 to predict a placeholder rather than an incorrect token. During generation, FormaT5 may be unable to capture parts of the intended R, either due to under-specification in Q or because of the model fails to learn. Code generation with placeholders over grammars by reinforcement learning was previously also used to generate code completion sketches [19].

Example 2. *Given the utterance* "Highlight the students who have middle school ID and have scored above Average" *from Example 1, the rule generation model struggles to identify the constant associated with* "middle school ID" *without further user information.*

We address this issue by extending the vocabulary with a special [?] token that represents a placeholder in the rule which needs to be filled. However, we cannot train such a model in a supervised setting as our training data does not have these [?] tokens. One solution to this is to use a weighted cross-entropy loss and reduce the weight of [?] tokens, such that the loss penalizes incorrect predictions more heavily as compared to [?] predictions. The modified loss L_h (simplified for a single placeholder) is shown below with α controlling the penalty for placeholder generation.

$$L_h = -(1-\alpha)\left(\sum_{t\in T}^{t\neq h} y_t \log(p_t)\right) - \alpha \cdot \log(p_h) \quad (1)$$

This appears to solve the problem, but it has two major limitations: (1) with this loss the model produces a lot more false-positive placeholders, and (2) it is difficult to tune the parameter α.

To address these issues, we employ the abstention loss objective introduced for standard multi-class classification in [47]. For each token c_i in the rule, this objective combines a standard cross-entropy loss function and the modified abstention loss

$$L(i) = (P(c_h) - 1)\log\left(\frac{P(c_i)}{1-P(c_h)}\right) + \alpha \log\left(\frac{1}{1-P(c_h)}\right),$$

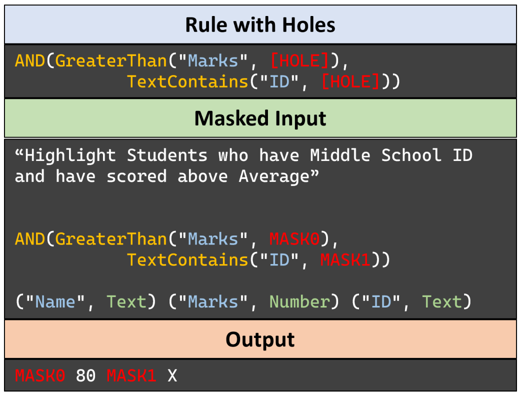
where c_h = [?] and $P(\cdot)$ denotes the token probability from the model. The first term computes a modified cross-entropy loss (renormalized for non-placeholder token probability mass) and weights it by the complement of the placeholder probability. The second term computes a placeholder penalty adjusted by a factor α.

Example 3. *Given the utterance "Highlight the students who have middle school ID and have scored above Average," FormaT5 with abstention generates a rule*

And(GreaterThan(Average("Marks")), TextContains([?]))

with a (single) placeholder [?].

4.4.2 Fine-tuning FormaT5. We fine-tune FormaT5 on the 105K bootstrapped samples. We use an Adam optimizer with a constant learning rate of $1e-3$. We train for 100 epochs over batches of 32 samples. We set the maximum sequence length of the model to 128


Rule with Holes

```
AND(GreaterThan("Marks", [HOLE]),
         TextContains("ID", [HOLE]))
```

Masked Input

```
“Highlight Students who have Middle School ID
and have scored above Average”


AND(GreaterThan("Marks", MASK0),
         TextContains("ID", MASK1))

("Name", Text) ("Marks", Number) ("ID", Text)
```

Output

```
MASK0 80 MASK1 X
```

Figure 4: Input and target for value filling model used for filling placeholders. The value filling task is implemented as a mask span prediction where the placeholders in the rule sketch are the masks that the model needs to predict.

and use the standard T5 [PAD] token to fill batches. We use the same tuning procedure described in [47] to set a value for α during training by starting from 0 and linearly ramping up the value of α. This ensures that the model learns abstention only for cases that need them and the regular training is not affected. We also explored replacing T5 with BART and found similar performance.

4.5 Filling placeholders

To fill placeholders, we distinguish between the cases where examples are not available and when examples are available.

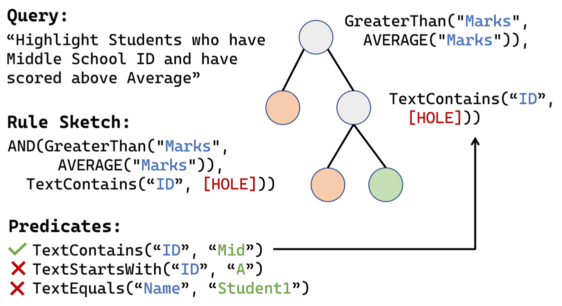
4.5.1 Without examples. When examples are not provided, we reuse FormaT5 with standard cross-entropy loss (without placeholders) in a mask span prediction setting. The input contains the NL utterance, table schema, and the rule with placeholders replaced by [MASK] tokens. Figure 4 shows an example input and target for the model. We show that first learning a sketch and then populating placeholders performs better than directly predicting the whole rule without placeholders in Section 7.3.

4.5.2 With examples. When examples are available, we leverage the Cornet system for learning CF rules by examples [46]. In summary, Cornet (1) clusters a column to obtain positive, negative and soft negative examples, (2) iteratively learns decision trees on those clustered examples, and (3) ranks the learned decision trees. Each decision tree corresponds to one CF rule, where each node corresponds to one predicate. FormaT5 uses the same three steps to learn CF rules from the given examples, but constrains the learned decision trees to match the generated sketch. To do this efficiently, the rule with placeholders is converted to a decision tree with placeholders in nodes and these placeholders are populated—using Cornet predicate generation—to maximize the splitting criterion. Note that both FormaT5 and Cornet can explicitly state a grammar and can be made fully compatible.

Example 4. *Figure 2 shows a column with highlighted cell and an utterance. FormaT5 first generates a rule with a placeholder:*

And(GreaterThan(Average("Marks")), TextContains([?]))

The associated decision tree with placeholders is shown in Figure 5. We only need to compute the splitting criterion over CORNET *predicates for one node to complete the tree with* TEXTCONTAINS*("MID").*



Query:
"Highlight Students who have Middle School ID and have scored above Average"
Rule Sketch:
AND(GreaterThan("Marks", AVERAGE("Marks")), TextContains("ID", [HOLE]))
Predicates:
✓ TextContains("ID", "Mid")
✗ TextStartsWith("ID", "A")
✗ TextEquals("Name", "Student1")
GreaterThan("Marks", AVERAGE("Marks")),
TextContains("ID", [HOLE]))

Figure 5: Example of a CORNET decision tree, where nodes contain CF predicates, with a placeholder in a node. To learn the split for this node, we only need to consider the predicate generation step of TextContains over the ID column.

4.6 Constrained Semantic Decoding

To generate syntactically valid rules that can be successfully executed, we use constrained semantic decoding [37, 44]. During the decoding process, we verify that (1) the predicted token satisfies the rule grammar, (2) that any column references are present in the table and have the appropriate type for their proposition, and (3) that placeholders are only generated for argument tokens (i.e. constants). Figure 6 shows how the grammar and table schema constrain the possible next tokens for the prefix of a rule.

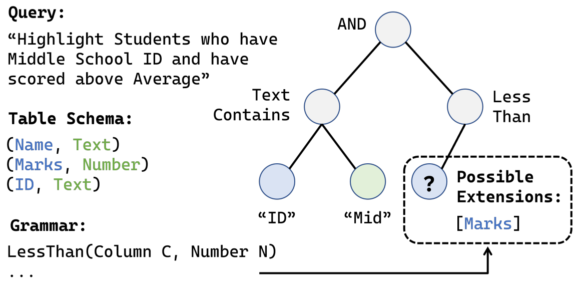


Figure 6: CSD of formatting rules for a prefix And(TextContains(ID, "Mid"), LessThan(. The only possible extension is Marks, as LessThan is a numerical operation and Marks is the only number column in the schema.

5 BASELINES

As NL-to-CF is a new problem, we adapted various systems designed for code generation or table conditioned tasks. These baselines were chosen by considering the state-of-the-art systems from three different categories: code generation models (like Codex and T5), constrained generation models (like Synchromesh and PICARD) and table based models (like TAPAS and TaBERT). Five of the eight baselines approaches (TAPAS, TaBERT, ValueNet, PICARD, Synchromesh) have been used for text-to-SQL. Of these, PICARD is state-of-the-art on the popular Spider dataset [54]. We report top-1 results for FORMAT5 and all baselines.

5.1 Code Generation

We fine-tune T5-base, CodeT5-large and CodeT5+ (770M) on our data, and use GPT-3.5 variants in a few-shot setting.

5.1.1 T5, CodeT5, CodeT5+. T5 is the base model of FORMAT5 and uses an encoder-decoder architecture. CodeT5/CodeT5+ also use T5 as the base model and pre-train on code generation and understanding tasks. We train T5, CodeT5, and CodeT5+ to generate formatting rules from (utterance, table) pairs. The input is the same as that of FORMAT5 fine-tuning (Section 4.4). The models are optimized on standard cross-entropy loss with Adam optimizer at $1e{-}4$ learning rate for 100 epochs.

5.1.2 GPT Variants. We use multiple variants of GPT-3.5 [1]: *code-davinci-002*, *text-davinci-003*, and *gpt-3.5-turbo*. We use all in a few-shot setting. For each test, we select five examples from the training corpus based on cosine similarity of SentenceBERT [41] embedding over the NL utterance. We evaluated using temperature values between 0 to 1 with a step size of 0.1 and report the best result. All other parameters are set at their default value.

5.1.3 CodeLlama and StarCoder. We use the pre-trained versions of CodeLlama [42] and StarCoder [29] in a few-shot setting. We use the inference as described in the documentation at [52].

5.2 Constrained Generation

Synchromesh, PICARD and ValueNet [5] are three popular constrained generation systems trained for text-to-SQL.

5.2.1 Synchromesh. Synchromesh [37] uses Codex but enhances it with the addition of target-sample-tuning (TST) and constrained semantic decoding (CSD). We fine-tune TST using mean-pooled CodeBERT embeddings [15]. The CSD architecture is the same as the one described in Section 4.6. We use Synchromesh in a few-shot setting using the same prompt template as Codex (Section 5.1.2).

5.2.2 PICARD. PICARD [44] is a constrained decoding system that improves the generations of large language models when generating syntactically structured information like code. We use PICARD with T5, as done by the original authors. The training of T5 is the same as described in Section 4.4. The constrained decoding does not need training as its used at inference

5.2.3 ValueNet. ValueNet [5] is an end-to-end system for generating SQL queries from natural language specifications. ValueNet leverages full table information by extracting relevant value candidates from the table, and a syntax based decoding to generate SQL queries. We fine-tune ValueNet end-to-end for our conditional formatting rule generation task.

5.3 Table Based

TaBERT [53] and TAPAS [23] are models trained for table question-answering, and thus capture properties about tables and natural language. SpreadSheetCoder [7] is a predictive formula generation model that generates formulas from table context.

5.3.1 *TAPAS and TaBERT.*

TAPAS and TaBERT are table encoding models pre-trained for question answering over tables. Since these models are not generative and only produce an encoding, we add a decoder on top of these to convert these to an encoder-decoder model. TAPAS produces joint table and query embedding, so we directly pass that to a decoder. TaBERT produces separate embeddings, one each for the query and the table, which need to be concatenated before passing to the decoder. We tried different setting for number of decoders and attention heads in each decoder, and report results for the best performing model.

5.3.2 *SpreadsheetCoder.*

SpreadsheetCoder [7] is a predictive Excel formula generation model that uses contextual information, like table headers and neighbouring values, to suggest formulas for cells. Since it only takes as input a table, we adapt the table to include the utterance by adding a dummy column whose header is the utterance. We then predict the formula for the first cell in the dummy column and train this model end-to-end.

6 EVALUATION SETUP

We first describe the system specifications used for carrying out experiments, benchmark statistics, and evaluation metrics.

6.1 Hardware Specifications

All experiments and studies have been carried out using Python (version 3.8.7). The system used to run the experiments uses an Intel Core i7 processor (base at 1.8 GHz) along with a K80 GPU, a 64-bit operating system, and 32 GB RAM. FormaT5 took 12 hours to pre-train and 6 hours to fine-tune on our dataset. We implemented FormaT5 using the HuggingFace `transformers` library [52] on top of `pytorch` [36] and the existing Cornet PBE system [46].

6.2 Benchmarks

We collect benchmarks from a variety of sources to capture the diverse nature of CF rules and perform a domain-agnostic evaluation.

Instruct Excel (IE) Prior work [34] crowdsourced 5K Excel spreadsheets where annotators were asked to perform operations of their choice (like formatting, pivoting and plotting) and provide a description of the operations. Out of the 5K collected tasks, 660 involved conditional formatting. We took the associated workbook and NL description for these.

StackOverflow (SO) We scraped 1K posts on formatting data in spreadsheets which had an accepted answer. We filter out posts not requiring conditional formatting rules. For remaining posts we use the NL description and associated table to write the ground truth CF rule. After removing duplicate rules, we get 125 tasks. For posts that do not have tables, we generate tables aligned with the post description.

Tutorials (T) We collected 33 URLs containing blogs, tutorials, quizzes and help pages on how to use conditional formatting rules in Excel. We extracted NL descriptions and ground truth rules, and generated tables that align with those in the tutorial. After rule deduplication, we end up with 117 tasks.

Annotated (A) We randomly sampled a subset of 100 spreadsheets with at least 1 CF rule from our corpus of public spreadsheets. We manually annotate these rules with NL utterances by describing the formatted table without looking at the rule. This gives us 151 tasks.

To characterize the number of columns used in a CF rule and the number of columns in the associated table, we label tasks with a tuple (R, T), where R denotes the number of columns in the CF rule and T the number of columns in the table. R (and T) can take values of S, for single, and M, for multiple. With this, we label benchmark tasks as one of three categories: *(S, S)*, *(S, M)*, and *(M, M)*. We classify benchmark tasks into these three subcategories in Table 1.

Table 1: Properties of benchmark tasks. *Rows* is mean number of rows in table, *Cols* is mean number of columns in Rule (R) and Table (T). Tasks denote the number of tasks broken by task type (R, T), where R denotes columns in rule and T denotes columns in table, which can be S (single) or M (multiple). Depth is the AST rule depth in our grammar.

Source	Tasks			Rows	Cols		Depth
	(S, S)	(S, M)	(M, M)		R	T	
IE	97	524	39	172.6	1.3	11.4	1.4
SO	54	39	32	34.7	4.1	5.7	2.3
T	4	67	46	39.5	3.2	8.6	2.7
A	47	92	12	143.2	1.9	13.6	1.5
Total	**202**	**722**	**129**	**67.2**	**1.9**	**10.7**	**2.1**

6.3 Evaluation Metrics

We consider three metrics: sketch, exact, and execution match. *Sketch match* is an argument agnostic match between the generated and ground truth rule, equivalent to comparing abstract syntax trees and ignoring leaf nodes. Sketch match aims to evaluate the structure of the generated rule and is a popular metric in code generation systems [7]. *Exact match* is a syntactic match between the generated and ground truth rule, with tolerance for differences arising from white space and alternative argument order. *Execution match* consists of executing two rules and comparing the produced formattings—there is an execution match if the formattings are identical. Execution match captures the fact that different rules can produce the same formatting outcomes. This distinction between exact and execution match is also made in the text-to-code domain [44].

Example 5. *because they are equivalent after removing spaces and swapping (equivalent) argument order, Or(Equals(Value, 10), Equals(Value, 20)) and Or(Equals(Value, 20), Equals(Value, 10)) are an exact match. On the other hand, TextStartsWith(Code, "D12") and TextContains(Code, "D12") are not an exact match because the rules are not equivalent. These may be an execution match if the Code column only has "D12" at the start of values.*

7 RESULTS AND DISCUSSION

We perform experiments to evaluate the performance of FormaT5 and answer the following research questions:

Table 2: **Comparison of FormaT5 with baselines on the task of NL based rule generation. We report sketch (SM), exact (EM) and execution match (ExM) of the generated rules for the different task categories. "Model" column denotes the underlying base model used by the system. FormaT5 outperforms all baselines in sketch, execution and exact rule match.**

System description			(S,S)			(S,M)			(M,M)		
Method	**Model**	**Param**	**EM**	**SM**	**ExM**	**EM**	**SM**	**ExM**	**EM**	**SM**	**ExM**
T5	T5	770M	74.8	86.8	77.7	63.3	84.7	70.9	48.9	68.2	54.6
CodeT5	T5	770M	75.4	88.0	78.0	65.1	83.8	72.5	50.4	69.3	55.8
CodeT5+	T5	770M	75.6	88.1	78	65.3	83.8	72.4	50.9	70.5	56.4
code-davinci-002	GPT-3.5	175B	68.5	82.1	70.8	62.9	79.7	70.4	45.1	62.4	50.8
text-davinci-003	GPT-3.5	175B	68.7	82.4	71.5	62.9	79.8	70.6	45.7	62.8	52.1
gpt-3.5-turbo	GPT-3.5	Unknown	67.9	81.8	71.2	61.4	79.2	70.1	44.8	61.9	50.5
CodeLlama	Llama 2	7B	65.6	78.2	70.1	60.1	77.9	69.1	41.3	57.5	48.2
StarCoder	GPT-2	15.5B	65.8	78.3	70.4	60.3	78.2	69.8	41.5	57.8	48.5
PICARD	T5	770M	75.8	88.7	78.3	67.4	85.6	75.0	52.4	73.5	58.2
Synchromesh	GPT-3	175B	74.5	87.5	76.9	66.0	83.5	73.3	50.9	69.8	56.7
ValueNet	BERT	110M	71.4	79.9	73.9	60.4	74.8	67.6	42.4	63.2	47.9
TAPAS	BERT	110M	73.3	84.7	75.5	65.2	72.6	72.4	43.4	62.6	49.0
TaBERT	BERT	110M	69.6	81.5	72.6	61.6	70.9	68.5	41.8	60.5	47.3
SpreadsheetCoder	BERT	110M	46.7	52.4	47.5	32.4	37.8	33.9	21.3	26.7	23.4
FormaT5	**T5**	770M	**78.2**	**90.7**	**81.2**	**70.7**	**86.2**	**78.0**	**58.6**	**79.5**	**64.3**

Q1. Can FormaT5 generate CF rules from table and utterance?
Yes, FormaT5 scores 2.4–6.2 points higher (exact match) than the second best baselines.

Q2. Are additional examples useful when generating rules?
Yes, providing one and two examples increases the exact match of FormaT5 by 7% and 9% respectively.

Q3. Does FormaT5 appropriately insert placeholders when generating rules with abstention?
Yes, 97.6% of arguments correctly predicted without placeholders are still correct with placeholders, and 90.8% of arguments incorrectly predicted without placeholders are correctly predicted as placeholders.

Q4. Do pre-training & constrained decoding improve FormaT5?
Yes, both pre-training objective and constrained decoding improve exact match performance by +.5%.

We also perform a case study on supporting rule grammars in other spreadsheet systems (**CS1**).

7.1 Generation from Utterances and Tables (Q1)

Table 2 presents the results of FormaT5 and baselines on the benchmarks. FormaT5 outperforms all baselines on sketch, exact and execution match accuracy. We use the value filling model without examples (section-4.5.1) to fill placeholders.

Pre-training on rules and tables lets FormaT5 leverage table semantics more than text-to-code models (CodeT5, Codex). Constrained generation attempts to capture these table semantics during generation. While PICARD is better than other text-to-code models, it performs worse than FormaT5, which learns these semantics during training rather than only impose them during decoding.

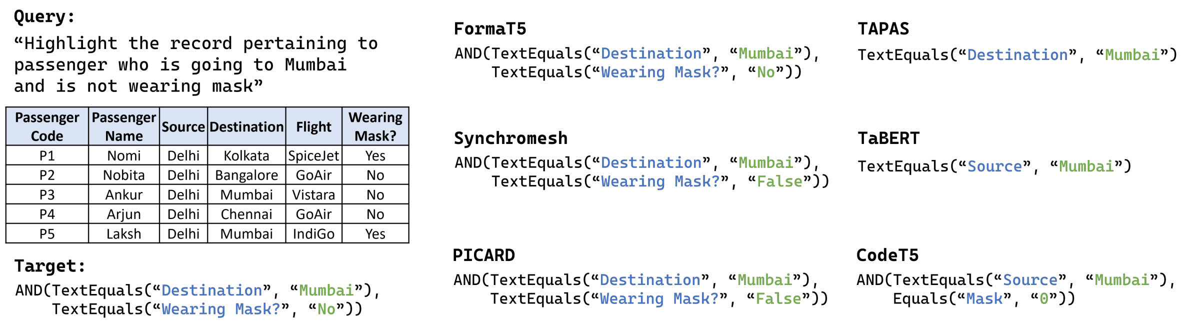
Table querying models like TAPAS and TaBERT perform worse than all other baselines. Even though these models consider full table information in their encoding, they are not trained to generate code and perform poorly at a generation task.

All models have a relatively high sketch accuracy compared to full match or execution match accuracy. Systems are better at predicting the right rule structure and are struggling more in picking the correct arguments This is mainly due to (1) underspecification in utterances and (2) LLM-based code generation models struggle with arguments, as described in [5].

7.1.1 Success Case of FormaT5. We look at cases where FormaT5 is able to generate the correct rule and baselines systems are not. We find that these are mostly where FormaT5 is able to extract information better from tables and abstention allows it to fill ambiguous arguments more accurately in a second step. Figure 7 shows an example from the benchmarks where the user wanted to highlight passengers flying to Mumbai who are not wearing a mask. The correct rule for this is `AND(TextEquals("Destination", "Mumbai"), TextEquals("Wearing Mask?", "No"))`. FormaT5 is able to generate the correct rule with just the NL utterance. TAPAS and TaBERT get the sketch wrong as they miss the condition that the passenger is not wearing a mask. CodeT5 gets the column name wrong and uses a numeric rule on a text type column. Synchromesh and PICARD get the structure right but get the arguments wrong, by predicting `False` instead of `No`. FormaT5 does not make this mistake as it can leverage the column name *"Wearing Mask?"* to predict `No` as the argument for the rule.

7.1.2 Failure Case of FormaT5. While FormaT5 consistently outperforms baselines, there are a few cases where Codex learns the correct rule and FormaT5 does not. For example, given a query *"Color all rows with a successful and valid return code"*, Codex generates the rule Equals(Codes, 200) whereas FormaT5 generates Equals(Codes, 100). Since Codex is trained on much more data (over 1B lines of code) it knows 200 is the *successful return code* in HTTP responses.

7.1.3 Effect of Rule Complexity. We study the effect of rule complexity on performance by analyzing exact match as a function



Query:

```
“Highlight the record pertaining to
passenger who is going to Mumbai
and is not wearing mask”
```

Passenger Code	Passenger Name	Source	Destination	Flight	Wearing Mask?
P1	Nomi	Delhi	Kolkata	SpiceJet	Yes
P2	Nobita	Delhi	Bangalore	GoAir	No
P3	Ankur	Delhi	Mumbai	Vistara	No
P4	Arjun	Delhi	Chennai	GoAir	No
P5	Laksh	Delhi	Mumbai	IndiGo	Yes

Target:

```
AND(TextEquals(“Destination”, “Mumbai”),
    TextEquals(“Wearing Mask?”, “No”))
```

FormaT5

```
AND(TextEquals(“Destination”, “Mumbai”),
    TextEquals(“Wearing Mask?”, “No”))
```

TAPAS

```
TextEquals(“Destination”, “Mumbai”)
```

Synchromesh

```
AND(TextEquals(“Destination”, “Mumbai”),
    TextEquals(“Wearing Mask?”, “False”))
```

TaBERT

```
TextEquals(“Source”, “Mumbai”)
```

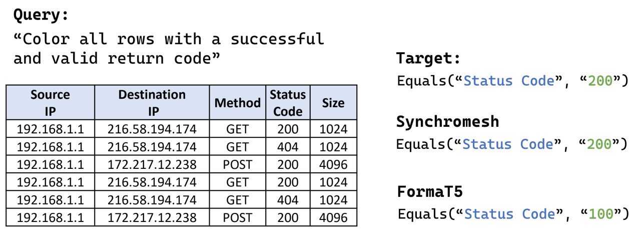
PICARD

```
AND(TextEquals(“Destination”, “Mumbai”),
    TextEquals(“Wearing Mask?”, “False”))
```

CodeT5

```
AND(TextEquals(“Source”, “Mumbai”),
    Equals(“Mask”, “0”))
```

Figure 7: Example case where FormaT5 learns the correct rule but baselines fail.



Query:

```
“Color all rows with a successful
and valid return code”
```

Source IP	Destination IP	Method	Status Code	Size
192.168.1.1	216.58.194.174	GET	200	1024
192.168.1.1	216.58.194.174	GET	404	1024
192.168.1.1	172.217.12.238	POST	200	4096
192.168.1.1	216.58.194.174	GET	200	1024
192.168.1.1	216.58.194.174	GET	404	1024
192.168.1.1	172.217.12.238	POST	200	4096

Target:

```
Equals(“Status Code”, “200”)
```

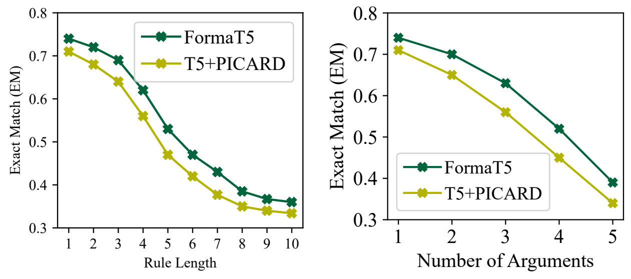
Synchromesh

```
Equals(“Status Code”, “200”)
```

FormaT5

```
Equals(“Status Code”, “100”)
```

Figure 8: Example case where Codex learns the correct rule but FormaT5 fails. This is due to the fact that FormaT5 is a much smaller model (770M) compared to Codex (175B) and doesn't relate *successful response code* to 200



(a) Rule Length **(b) Number of Arguments**

Figure 9: Exact rule match (EM) across all benchmarks of FormaT5 and best performing baseline (PICARD) plotted against rule length and number of arguments (without examples). Performance drops as rules become more complex, but FormaT5 is consistently better.

of rule length (number of non-syntax tokens) and number of arguments in the rule. Figure 9 shows FormaT5 and the next best baseline system (PICARD). As expected, exact match declines with longer rules and rules with more arguments. FormaT5 outperforms the next best system across length and argument counts.

7.1.4 Inference Time and Memory Consumption. We also evaluate the time and memory required by each system for inferring the formatting rule. Table 3 shows the average time taken, disk space used and GPU memory allocated, to predict a rule for FormaT5 and baselines. Codex and Synchromesh are not included in this table because these models are only available via APIs and we cannot run them locally. FormaT5 uses T5 as the underlying model and hence the time and memory consumption are comparable to CodeT5. TAPAS and TaBERT uses a smaller model with 110M parameters and are thus faster and more lightweight. However, these models are limited in their learning capabilities. PICARD is the slowest out of these as its constrained decoding is expensive as noted in [44].

7.2 Generation with Examples (Q2)

We employ two different strategies to integrate examples in our baselines. *Chain* generates the rule in two steps: first generate the sketch using the utterance and then predict values for the sketch using utterance and examples. This strategy has been shown to be successful for query generation [28]. Both the sketch generation and the value filling step use the same baseline architecture and are fine tuned separately. *Examples in Prompt* directly adds examples in prompt. These are encoded along with the utterance and table, separated by a [SEP] token. We fine tune the systems by augmenting the data with examples obtained from ground truth execution. To ensure that the generated rule satisfies the given examples, we generate 50 rules via beam search and take the highest ranked rule satisfying the examples.

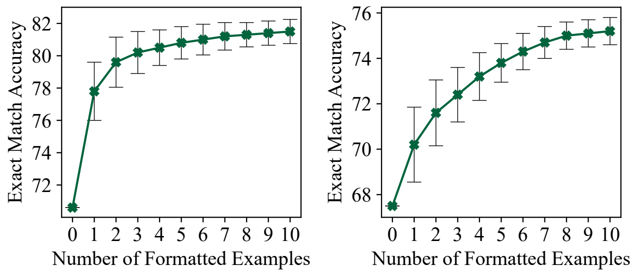
Table 4 shows results with utterance and a single example for FormaT5 and the best performing example strategy for each baseline system. All systems perform better compared to only using natural language utterances (Table 2). FormaT5 benefits the most from using examples and achieves the highest increase in performance (+7.3% versus +3.2% for PICARD).

Figure 10 shows the average exact match accuracy for FormaT5 and PICARD over 25 shuffled runs for an increasing number of examples. FormaT5 is generally able to learn the correct rule with just 2 examples—further examples provide diminishing returns.

Figure 11 shows a breakdown of FormaT5 results with a single example based on (a) type of rule and (b) number of placeholders in the rule (before filling). We find that text rules have the biggest improvement with examples and numeric rules have the smallest. This is consistent with the observation made in Cornet that numeric rules are harder to generate by example due to possibly infinite space of arguments. Rules with more placeholders are harder to complete with few examples.

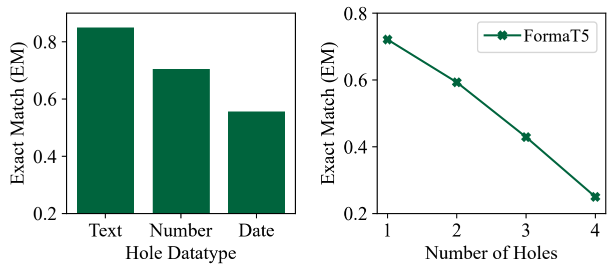
Table 3: **Table comparing rule inference time in milliseconds, model disk space in megabytes and GPU memory used in megabytes, averaged over all benchmarks for FormaT5 and baselines with comparable size (< 1B parameters). Codex and Synchromesh are not included since these models can only be accessed via API and not run locally. FormaT5 has comparable inference time and memory usage to other systems.**

System	#param	Time(ms)	Disk(MB)	Memory(MB)
CodeT5	770M	1632.4	897.3	914.5
PICARD	770M	1896	901.3	918.7
ValueNet	110M	996.1	563.7	645.4
TAPAS	110M	852.9	443.6	1416.8
TaBERT	110M	734.5	193.2	334.5
FormaT5	770M	1783.1	897.3	915.2



(a) FormaT5 **(b) PICARD**

Figure 10: Exact match accuracy across all benchmarks with increasing number of examples for FormaT5 and PICARD. We report average of 25 runs with shuffled rows. FormaT5's performance stabilizes with just 2 examples.



(a) Hole Datatype **(b) Number of Holes**

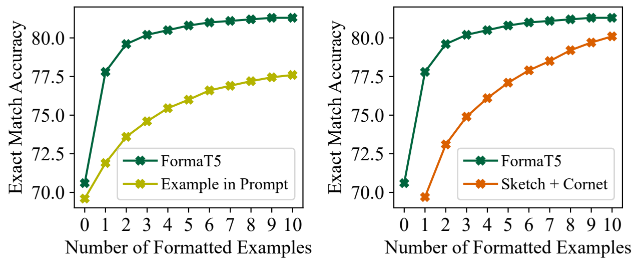
Figure 11: (a) Exact match for FormaT5 based on the placeholder type in the rule. If a rule has multiple placeholders of different types, it is counted towards each type. (b) Exact match as a function of number of placeholders in the rule.

7.3 Abstention (Q3)

To evaluate the quality of predicted placeholders, we train FormaT5 with and without placeholders and compare the argument predictions for both models. For example, if the model without placeholders correctly predicts an argument, the placeholder model can change it to (1) a placeholder, (2) the same argument, or (3) an incorrect argument. Table 5 summarizes these outcomes. We can see that FormaT5 rarely changes correct arguments to placeholders (2.4% cases) and converts most incorrect arguments (90.8% cases) to placeholders—both desirable outcomes. Furthermore, we find that FormaT5 generates placeholders in only approximately 11% – 22% of rules (depending on task category) and that these rules usually have only one placeholder (mean 1.07).

To understand the benefit of predicting placeholders, we compare FormaT5 to two variants: (1) generating a sketch with all arguments as placeholders and using Cornet to fill them, and (2) adding examples to the prompt and generating the rule in a single step (Examples in Prompt strategy in section-7.2) Figure 12 compares FormaT5 against both variants for increasing number of examples. Adding examples in prompt performs worse, as it is hard for a neural model to extract syntactic constraints from few examples. Full sketches blow up the number of placeholders (to 3.64 per rule) that are difficult to fill with few examples. FormaT5, on the other hand, only generates placeholders in places of under-specification.

FormaT5 uses a symbolic, example-based method to fill placeholders. We compare this with two other methods: (1) a T5 based model that takes utterance, examples and masked rule as input to predict masked arguments, and (2) a decision tree that uses Cornet predicates and is constrained to the generated rule sketch. Figure 13 shows the exact match accuracy for these different value filling methods. FormaT5 performs better than both variants.



(a) FormaT5 with examples in prompt **(b) FormaT5 with sketch plus value filling**

Figure 12: Exact match accuracy for FormaT5 compared with (a) adding examples directly in prompt (direct generation), and (b) sketch generation (all arguments become placeholders) followed by value filling with Cornet. FormaT5 performs better than both variants.

Figure 14 shows cases from benchmarks where FormaT5 predicted a hole in the rule. It can be seen that FormaT5 adds placeholders where the provided NL utterance are either under-specified or ambiguous. For the examples shown, FormaT5 fills all placeholders correctly and generate the target rule with just a single example.

7.4 Ablations (Q4)

FormaT5 uses (1) table-aware pre-training and (2) constrained generation. We analyze the impact of both of these.

Table 4: Comparison of FormaT5 with baselines for generating rules with utterance and examples. We report exact (EM) and execution match (ExM) for the different benchmarks with 1 example given. We consider *Chaining (Chain)* and *Examples in Prompt (Prompt)* stratergy for example selection and report the best strategy for each method. FormaT5 outperforms all baselines in execution and exact match.

System description			(S, S)		(S, M)		(M, M)	
Method	Model	Strategy	EM	ExM	EM	ExM	EM	ExM
T5	T5	Chain	79.0	81.2	66.7	72.4	52.5	56.3
CodeT5	T5	Chain	79.7	81.6	68.7	74.3	54.1	57.4
CodeT5+	T5	Chain	79.9	81.9	68.7	74.8	55.2	58.2
code-davinci-002	GPT-3.5	Prompt	74.0	75.1	65.9	73.1	50.3	53.4
text-davinci-003	GPT-3.5	Prompt	74.4	75.7	66.3	73.4	51.4	55.1
gpt-3.5-turbo	GPT-3.5	Prompt	74.1	76.0	65.3	73.1	50.6	55.2
CodeLlama	Llama 2	Chain	72.1	75.3	64.2	72.3	47.3	53.2
StarCoder	GPT-2	Prompt	72.4	75.5	64.5	72.9	47.5	53.4
PICARD	T5	Chain	80.2	81.7	71.4	77.3	55.6	59.8
Synchromesh	GPT 3	Prompt	80.3	81.9	70.7	75.6	54.7	58.6
ValueNet	BERT	Chain	74.9	75.6	63.0	68.7	46.5	49.80
TAPAS	BERT	Prompt	76.6	78.1	67.5	73.1	47.8	50.4
TaBERT	BERT	Prompt	72.2	74.7	64.4	69.9	45.3	48.9
FormaT5	**T5**	**FormaT5**	**87.4**	**89.1**	**78.2**	**84.8**	**65.9**	**69.4**

Table 5: Argument outcomes under FormaT5 without (left) and with (top) abstention. Most correct arguments remain correct and most incorrect arguments become placeholders.

FormaT5	with placeholders		
without placeholders	[?]	Correct	Incorrect
Correct Argument	2.4	97.6	0
Incorrect Argument	90.8	3.4	5.8

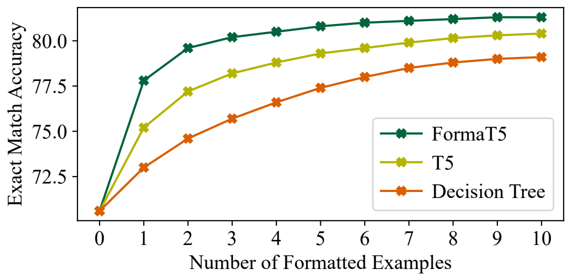
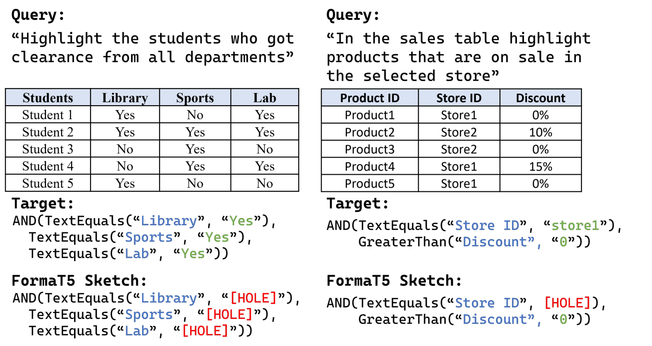


Figure 13: Exact match accuracy for different value filling models plotted against the number of examples. The examples are provided in top down order. FormaT5 performs better than both T5 and Decision Tree based value filling.

7.4.1 Pre-training. We evaluate the impact of the pre-training objectives by ablating them. The first three rows of Table 6 show the exact match accuracy for FormaT5 on different benchmark tasks, removing one pre-training objective at a time. We find that all objectives contribute to FormaT5 performance, with table recovery objective being the most important. The table recovery objective allows FormaT5 to learn rule and column type associations.



Query:
"Highlight the students who got clearance from all departments"

Students	Library	Sports	Lab
Student 1	Yes	No	Yes
Student 2	Yes	Yes	Yes
Student 3	No	Yes	No
Student 4	No	Yes	Yes
Student 5	Yes	No	No

Target:
```
AND(TextEquals("Library", "Yes"),
  TextEquals("Sports", "Yes"),
  TextEquals("Lab", "Yes"))
```

FormaT5 Sketch:
```
AND(TextEquals("Library", "[HOLE]"),
  TextEquals("Sports", "[HOLE]"),
  TextEquals("Lab", "[HOLE]"))
```

Query:
"In the sales table highlight products that are on sale in the selected store"

Product ID	Store ID	Discount
Product1	Store1	0%
Product2	Store2	10%
Product3	Store2	0%
Product4	Store1	15%
Product5	Store1	0%

Target:
```
AND(TextEquals("Store ID", "store1"),
    GreaterThan("Discount", "0"))
```

FormaT5 Sketch:
```
AND(TextEquals("Store ID", [HOLE]),
    GreaterThan("Discount", "0"))
```

Figure 14: Examples of tasks where FormaT5 generates a hole due to underspecification or ambiguity in the NL utterance. FormaT5 resolves the holes with one example. The scenarios are from Tutorials and InstructExcel, respectively

7.4.2 Constrained Semantic Decoding (CSD). We use CSD to ensure reliability of generated rules. The fourth row in Table 6 shows the performance of FormaT5 without CSD. We find that CSD improves performance by eliminating impossible candidates and forcing the model to generate a syntactically valid rule.

7.4.3 Encoding Tables. One of the key factors behind the performance of FormaT5 is its ability to leverage the table structure along with the NL utterance. Prior work on integrating NL and Tables have tried to encode the entire table [23], a subset of rows [53] and header names [7]. To evaluate FormaT5's table encoder, in addition to FormaT5 table encoder, we also implement the following encoding techniques and compare performance on the benchmarks.

- **Utterance Only:** We only use the utterance.
- **Header Only:** We only use table column names.

Table 6: Exact match on all benchmark datasets using the NL-only approach. M denotes full FormaT5, "–" means the corresponding component is removed.

Model	(S,S)	(S,M)	(M,M)
M – Rule MSP Objective	77.9	70.3	58.1
M – Rule Tagging Objective	77.7	70.0	58.0
M – Table Objective	77.3	69.5	57.2
M – CSD	77.5	69.7	57.5
M (FormaT5)	**78.2**	**70.7**	**58.6**

Table 7: Exact match on all benchmark categories for different table encoders. FormaT5 table encoder performs better than all variants for our task.

Table Encoder	(S, S)	(S, M)	(M, M)
Utterance Only	74.3	36.2	8.4
Header Only	75.6	68.0	55.9
Schema (FormaT5)	78.2	70.7	58.6
Sample Values	78.0	70.5	58.3
Row Subset	75.8	68.3	56.2

- **Schema:** FormaT5 table encoder
- **Sample Values:** We use the schema along with sample values from the column. The table in Figure 14 will be encoded as `(Students, Text, 90) [COL] (Library, Text, Yes)...`
- **Subset of Rows:** We use the header and the first 3 rows. The table in Figure 14 will be encoded as `Students, Library [COL] Student1, Yes [COL] Student2, No [COL] Student3, No`

Table 7 summarizes the exact match accuracy on benchmarks with different table encoding strategies. FormaT5 beats all variants as it extracts necessary information from the table while restricting context size. Performance is worst with only utterance as the table contains useful type and column information.

7.5 FormaT5 on Different Platforms (CS1)

FormaT5 can be applied to any platform that supports rule-based formatting based on a fixed grammar. To evaluate the generality of FormaT5, we evaluate its performance when restricting the output operators to those available on two additional spreadsheet platforms: Google Sheets and Apple Numbers.

The predicates used to train FormaT5, drawn from Microsoft Excel, are a superset of formatting rule trigger operations supported in Google Sheets and Apple Numbers. We restrict FormaT5 to platform-specific operator subsets at inference time using constrained decoding. To execute on a new platform, one would only need a translator from our DSL to the corresponding runtime.

We see that the performance is comparable when using Google Sheets operators. Apple Numbers has a limited formatting rule grammar and hence has the worst performance. Because FormaT5 was trained on Excel data, it is possible that retraining on data from other platforms would perform better (as users in those domains may express CF rules differently).

Table 8: Exact match (top 1 with 1 example) on all benchmark categories for platform-specific CF rule grammars.

Platform	(S,S)	(S,M)	(M,M)
Google Sheets	80.5	71.3	57.5
Apple Numbers	73.8	63.8	40.6
Microsoft Excel	87.4	78.2	65.9

8 LIMITATIONS

Availability of Tables: FormaT5 assumes it has access to the target table as a collection of named columns. Standard tables can be easily represented this way. However, spreadsheets can contain more complex structures (like multi-level headers and summary rows). Structures vary substantially so even detecting tables in the grid is an active area of research [12]. Extending FormaT5 to support more complex table structures is left to future work.

Multiple Tables: FormaT5 does not currently support rules that span multiple tables. One way of supporting this is to (1) predict which tables are relevant from the given utterance, (2) propositionalize them into one large table with fuzzy joins, and (3) apply FormaT5. While possible, our corpus shows that such rules are rare in practice: only 314 of 410.6K rules referenced more than 1 table.

Language Use: FormaT5 currently only supports natural language utterances in English. Ambiguous utterances are mitigated with a combination of constrained decoding and the use of examples. FormaT5 assumes that the user did not provide an example with a mistake. This is a common assumption in PBE, and is reasonable as users only needs to provide few examples (a single example for results in Table 4). Prior work in PBE [11, 20] has learned from noisy examples. FormaT5 could improve robustness by replacing our current symbolic PBE learner [46] with one of these alternatives.

Hardware Requirements: As Table 3 shows, the main resource for FormaT5 is the associated GPU memory needed. We believe that this requirement is mitigated by the increasing ease of running model inference on consumer processors (like Apple's M1).

9 CONCLUSION

We introduce the problem of learning spreadsheet formatting rules from natural language. We propose FormaT5, a transformer-based model that takes a table, natural language utterance, and optional examples, and generates a conditional formatting rule. FormaT5 combines domain pre-training, fine-tuning with abstention, constrained decoding, and both NL and example-based abstention resolution. We evaluate on 1053 tasks and show that FormaT5 performs better across task types and metrics, and is more effective at using examples. This paper opens future work such as predictive table formatting, generation over ambiguous NL and table specific pre-training and designing domain-specific systems for other specialized data tasks like filtering and cleaning. A similar approach can be explored for generating spreadsheet formulas from natural language, as these also contain a high proportion of constant values and cell references.

REFERENCES

[1] [n.d.]. *OpenAI Platform Documentation.* https://platform.openai.com/docs/model-index-for-researchers Accessed on December 3, 2023.

[2] [n.d.]. The Spider leaderboard. https://yale-lily.github.io/spider.

[3] 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, 486–489.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[5] Ursin Brunner and Kurt Stockinger. 2021. ValueNet: A Natural Language-to-SQL System that Learns from Database Information. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. 2177–2182. https://doi.org/10.1109/ICDE51399.2021.00220

[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. https://doi.org/10.48550/ARXIV.2107.03374

[7] Xinyun Chen, Petros Maniatis, Rishabh Singh, Charles Sutton, Hanjun Dai, Max Lin, and Denny Zhou. 2021. SpreadsheetCoder: Formula Prediction from Semi-structured Context. In *International Conference on Machine Learning*.

[8] Zhe Chen and Michael Cafarella. 2014. Integrating spreadsheet data via accurate and low-effort extraction. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1126–1135.

[9] Zhoujun Cheng, Haoyu Dong, Ran Jia, Pengfei Wu, Shi Han, Fan Cheng, and Dongmei Zhang. 2022. FORTAP: Using Formulas for Numerical-Reasoning-Aware Table Pretraining. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Dublin, Ireland, 1150–1166. https://doi.org/10.18653/v1/2022.acl-long.82

[10] 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

[11] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. 2017. Robustfill: Neural program learning under noisy i/o. In *International conference on machine learning*. PMLR, 990–998.

[12] Haoyu Dong, Shijie Liu, Shi Han, Zhouyu Fu, and Dongmei Zhang. 2019. Tablesense: Spreadsheet table detection with convolutional neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 69–76.

[13] 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*. 305–314.

[14] Microsoft Excel. 2023. Color scales, data bars and icon sets. https://support.microsoft.com/en-us/office/use-data-bars-color-scales-and-icon-sets-to-highlight-data-f118d0a6-5921-4e2e-905b-fe00f3378fb9. Last Accessed: 2023-04-30.

[15] 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 *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics, Online, 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

[16] 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*. 1–5.

[17] Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen-tau Yih, Luke Zettlemoyer, and Mike Lewis. 2022. InCoder: A Generative Model for Code Infilling and Synthesis. https://doi.org/10.48550/ARXIV.2204.05999

[18] Sumit Gulwani. 2011. Automating String Processing in Spreadsheets using Input-Output Examples. In *PoPL'11, January 26-28, 2011, Austin, Texas, USA.* https://www.microsoft.com/en-us/research/publication/automating-string-processing-spreadsheets-using-input-output-examples/

[19] Daya Guo, Alexey Svyatkovskiy, Jian Yin, Nan Duan, Marc Brockschmidt, and Miltiadis Allamanis. 2022. Learning to Complete Code with Sketches. In *International Conference on Learning Representations*. https://openreview.net/forum?id=q79uMSC6ZBT

[20] Shivam Handa and Martin C Rinard. 2020. Inductive program synthesis over noisy data. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 87–98.

[21] Moshe Hazoom, Vibhor Malik, and Ben Bogin. 2021. Text-to-SQL in the Wild: A Naturally-Occurring Dataset Based on Stack Exchange Data. In *Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021)*. Association for Computational Linguistics, Online, 77–87. https://doi.org/10.18653/v1/2021.nlp4prog-1.9

[22] Chaitra Hegde and Shrikumar Patil. 2020. Unsupervised Paraphrase Generation using Pre-trained Language Models. https://doi.org/10.48550/ARXIV.2006.05477

[23] 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)*. Seattle, Washington, United States. https://www.aclweb.org/anthology/2020.acl-main.398/

[24] Nathan Hurst, Kim Marriott, and Peter Moulder. 2005. Toward tighter tables. In *Proceedings of the 2005 ACM symposium on Document engineering*. 74–83.

[25] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2020. SpanBERT: Improving Pre-training by Representing and Predicting Spans. *Transactions of the Association for Computational Linguistics* 8 (2020), 64–77. https://doi.org/10.1162/tacl_a_00300

[26] Vu Le and Sumit Gulwani. 2014. FlashExtract: a framework for data extraction by examples. In *2014 Programming Language Design and Implementation*. ACM, 542–553. https://www.microsoft.com/en-us/research/publication/flashextract-framework-data-extraction-examples/

[27] Fei Li and Hosagrahar V Jagadish. 2014. NaLIR: An Interactive Natural Language Interface for Querying Relational Databases. 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, 709–712. https://doi.org/10.1145/2588555.2594519

[28] Hao Li, Chee-Yong Chan, and David Maier. 2015. Query from examples: An iterative, data-driven approach to query construction. *Proceedings of the VLDB Endowment* 8, 13 (2015), 2158–2169.

[29] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, 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 Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nourhan Fahmy, Urvashi Bhattacharyya, W. Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jana Ebert, Tri Dao, Mayank Mishra, Alexander Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean M. Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. StarCoder: may the source be with you! *ArXiv* abs/2305.06161 (2023). https://api.semanticscholar.org/CorpusID:258588247

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

[31] Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2021. Tapex: Table pre-training via learning a neural sql executor. *arXiv preprint arXiv:2107.07653* (2021).

[32] Yu Meng, Jiaxin Huang, Yu Zhang, and Jiawei Han. 2022. Generating Training Data with Language Models: Towards Zero-Shot Language Understanding. https://doi.org/10.48550/ARXIV.2202.04538

[33] Yu Meng, Martin Michalski, Jiaxin Huang, Yu Zhang, Tarek Abdelzaher, and Jiawei Han. 2022. Tuning Language Models as Training Data Generators for Augmentation-Enhanced Few-Shot Learning. https://doi.org/10.48550/ARXIV.2211.03044

[34] Swaroop Mishra, Justin Payan, Carina Negreanu, Christian Poelitz, Chitta Baral, Subhro Roy, Rasika Chakravarthy, Ben Van Durme, and Elnaz Nouri. 2023. InstructExcel: A Benchmark for Natural Language Instruction in Excel. *arXiv preprint* (2023).

[35] Avanika Narayan, Ines Chami, Laurel Orr, Simran Arora, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? arXiv:2205.09911 [cs.LG]

[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[37] 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. *ArXiv* abs/2201.11227 (2022).

[38] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. arXiv:2304.11015 [cs.CL]

[39] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21, 140 (2020), 1–67. http://jmlr.org/papers/v21/20-074.html

[40] Kia Rahmani, Mohammad Raza, Sumit Gulwani, Vu Le, Daniel Morris, Arjun Radhakrishna, Gustavo Soares, and Ashish Tiwari. 2021. Multi-Modal Program Inference: A Marriage of Pre-Trained Language Models and Component-Based Synthesis. *Proc. ACM Program. Lang.* 5, OOPSLA, Article 158 (oct 2021), 29 pages. https://doi.org/10.1145/3485535

[41] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 3982–3992. https://doi.org/10.18653/v1/D19-1410

[42] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy 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. 2023. Code Llama: Open Foundation Models for Code. arXiv:2308.12950 [cs.CL]

[43] Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R. Mittal, and Fatma Özcan. 2016. ATHENA: An Ontology-Driven System for Natural Language Querying over Relational Data Stores. 9, 12 (2016). https://doi.org/10.14778/2994509.2994536

[44] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 9895–9901. https://doi.org/10.18653/v1/2021.emnlp-main.779

[45] Jaydeep Sen, Chuan Lei, Abdul Quamar, Fatma Özcan, Vasilis Efthymiou, Ayushi Dalmia, Greg Stager, Ashish Mittal, Diptikalyan Saha, and Karthik Sankaranarayanan. 2020. ATHENA++: Natural Language Querying for Complex Nested SQL Queries. 13, 12 (2020). https://doi.org/10.14778/3407790.3407858

[46] Mukul Singh, José Cambronero Sánchez, Sumit Gulwani, Vu Le, Carina Negreanu, Mohammad Raza, and Gust Verbruggen. 2023. Cornet: Learning Table Formatting Rules By Example. *Proceedings of the VLDB Endowment* 16, 10 (2023), 2632–2644.

[47] Sunil Thulasidasan, Tanmoy Bhattacharya, Jeff A. Bilmes, Gopinath Chennupati, and Jamal Mohd-Yusof. 2019. Combating Label Noise in Deep Learning using Abstention. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, 6234–6243. http://proceedings.mlr.press/v97/thulasidasan19a.html

[48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[49] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 8696–8708. https://doi.org/10.18653/v1/2021.emnlp-main.685

[50] 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* (Virtual Event, Singapore) *(KDD '21)*. Association for Computing Machinery, New York, NY, USA, 1780–1790. https://doi.org/10.1145/3447548.3467434

[51] Eli Whitehouse, William Gerard, Yauhen Klimovich, and Marc Franco-Salvador. 2022. Programming by Example and Text-to-Code Translation for Conversational Code Generation. *ArXiv* abs/2211.11554 (2022).

[52] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Online, 38–45. https://doi.org/10.18653/v1/2020.emnlp-demos.6

[53] 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

[54] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium, 3911–3921. https://doi.org/10.18653/v1/D18-1425