Spreadsheet Formula Prediction through 3-Stage Decoding[*]

He Wanrong
2019012405
IIIS, Tsinghua University
`hwr19@mails.tsinghua.edu.cn`

Guo Xingzhuo
2019012383
IIIS, Tsinghua University
`gxz19@mails.tsinghua.edu.cn`

Abstract

We propose a novel model for spreadsheet formula prediction based on the Seq2seq method. We use a TUTA encoder that extracts headers and learns hierarchical structures for table contextual embedding. The decoding process is divided into three stages: generating formula sketch, generating values, and predicting cell references. We introduce a scoring model between two stages for reordering to deal with distribution shift. Our model achieves the state-of-the-art in the spreadsheet formula prediction field to the best of our knowledge.

1 Introduction

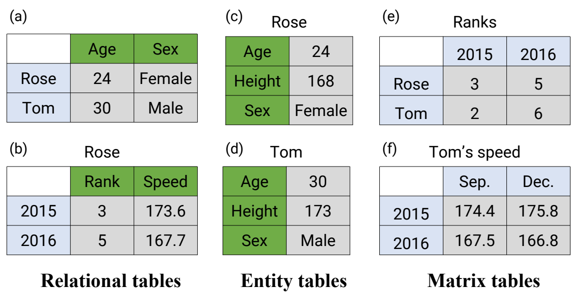
Hundreds of millions of people use spreadsheets, such as Excel, for data storage. The featured storage structure infers potential relations among data in cells. Users can add formulas in their spreadsheets to process and analyze data based on such relations. Users specify some cells, pick a formula function in a spreadsheet language, such as `SUM` and `AVG`, and wait for the software to calculate the data result automatically.

Although spreadsheet formula language targets general users and is much simpler than programming languages like C++, it is still difficult for Excel users to master. The formulas written by users could contain errors. Furthermore, to write a formula, users need to choose which formula function to use, which numbers or strings to fill, and which cell references to specify. This process could be time-consuming and error-sensitive, which might affect user experience.

Researchers have developed various formula auto-completion systems with different methods for these reasons. However, these systems are based on strong prior assumptions, such as the independence of rows, while actual spreadsheets are less limited. Another problem is the ignorance of headers. Headers are usually written in natural language and provide solid references for cell values and formulas. For example, if a header cell contains "total", then the following formula cell is likely to be a `SUM` function.

[*]Supervised by Haoyu Dong, Senior Researcher, MSRA

However, it is challenging to capture cell information and header information comprehensively. There are two main reasons: complex data structures and complex data contents. Firstly, unlike codes or other sequential languages, tables are two-dimensional but semi-structured. There are different types of tables, including relational tables, entity tables, and matrix tables, as shown in 1, which provides information on cell relations. Moreover, headers information is even harder to use for the hierarchical structure. Secondly, data in spreadsheets are recorded as natural languages, which requires strong understanding and relevance.



(a)

	Age	Sex
Rose	24	Female
Tom	30	Male

(b) Rose

	Rank	Speed
2015	3	173.6
2016	5	167.7

Relational tables

(c) Rose

Age	24
Height	168
Sex	Female

(d) Tom

Age	30
Height	173
Sex	Male

Entity tables

(e) Ranks

	2015	2016
Rose	3	5
Tom	2	6

(f) Tom's speed

	Sep.	Dec.
2015	174.4	175.8
2016	167.5	166.8

Matrix tables

Figure 1: Different forms of tables.

Thanks to the development of deep learning and pretrained models, such two challenges can be solved. Modern models, especially Transformers (Vaswani *et al.*, 2017), have a strong understanding ability both in the structure and in the content. The pretrained model supports fewer training data and faster convergence. Benefiting from the developments of these techniques, spreadsheet formula prediction using Transformers and other deep learning methods becomes possible.

In our work, we design a model to automatically generate potential spreadsheet formulas given table context and target cell. Our encoder-decoder model can understand the table context, especially the header and related contents in other cells, and smartly generate formulas via multi-stage decoding, including generating the formula sketch, generating text and number values, and predicting cell references.

2 Problem Setup

This section discusses the setup of our spreadsheet formula prediction problem.

2.1 Input Specification

Inspired by the hierarchical relationship of spreadsheet tables and the corresponding data storage methods, the table input is defined as the tabular information and header information. Tabular information includes the natural language in each cell arranged in tabular form. The header information includes the natural language in headers and the position of the header. Meanwhile, the position of the target cell is specified, telling the model where the formula is predicted. Note that the content of the target cell is replaced by a special token `[FORM]`, since the content is what we need to predict.

For simplicity, given the target cell, we assume the tabular information only depends on cells within the same row or column as the target cell. Therefore, instead of taking the whole tabular as input, we reduce the tabular input into only the corresponding row and column.

2.2 Formula Language

Our formula language comes from Excel[1], consisting of operators, functions, referenced cells and constant values.

A formula sketch is defined as the nesting of basic operators and functions with values and referenced cells positions replaced with special tokens. For example, formula `(SUM(A1:B2)/4)+1` has the formula sketch `( ( ( SUM ( [CellToken] : [CellToken] ) ) / [NumberToken] ) + [NumberToken] )`, which is the nesting of `SUM ( [CellToken] : [CellToken] )`, `FORMULA / [NumberToken]`, and `FORMULA + [NumberToken]`.

[1]Excel formulas tutorial: https://support.microsoft.com/en-us/office/Formulas-and-functions-294d9486-b332-48ed-b489-abe7d0f9eda9#ID0EAABAAA=Functions

To obtain the formula sketch, we first use XLParser (Aivaloglou *et al.*, 2015) a highly-compatible formula parser to obtain the raw parse tree of the formula. An example raw parse tree of formula `(SUM(A1:B2)/4)+1` is shown in figure 2. We then compress the raw parse tree by linking the terminals and bracketing the linked terminals with the least common ancestors to form a sequence with a uniform bracketing rule. Finally, we substitute all number values with `[NumberToken]`s, text values with `[TextToken]`s and referenced cells positions with `[CellToken]`s to obtain the formula sketch.

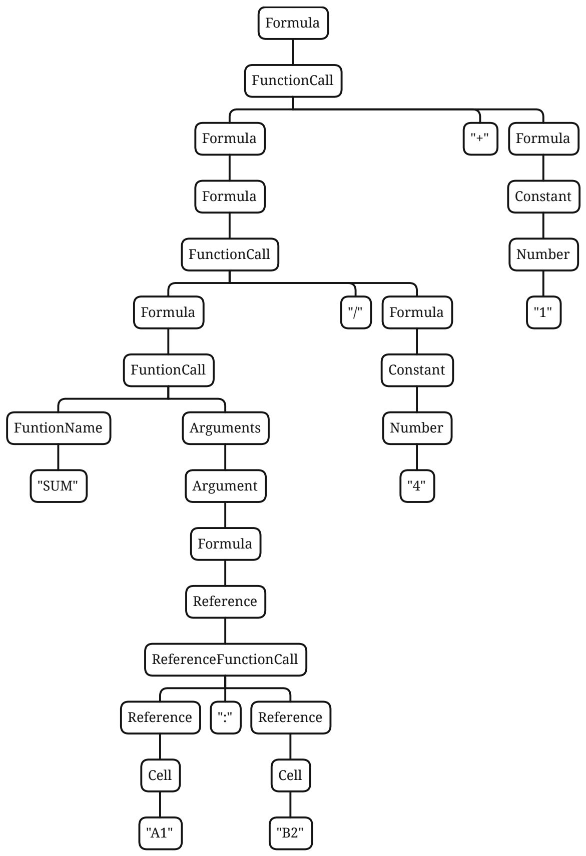


Figure 2: Raw parse tree of `(SUM(A1:B2)/4)+1`. Terminals are wrapped in “”.

2.3 Overall Setup

The overall setup is that, given a spreadsheet table and the target cell, we take the header and the corresponding row and column as input and predict the formula written in formula language.

3 Model

In this section, we present our model for spreadsheet formula prediction. An overall flow pass is shown in 3.

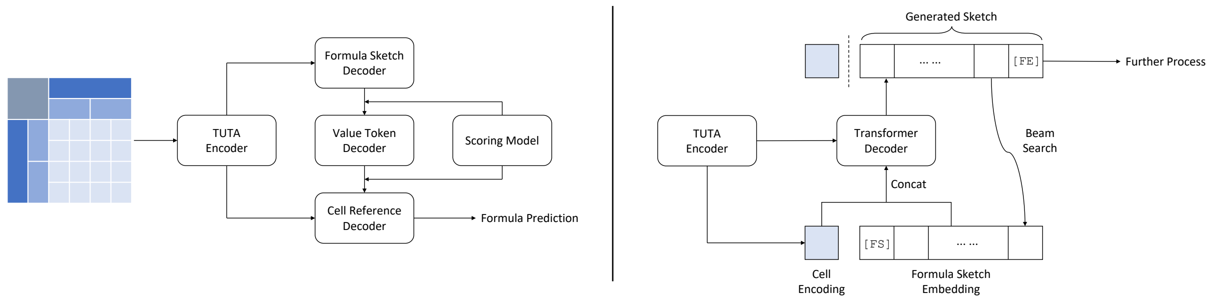


Figure 3: Overall flow pass of our model. **Left:** The overall architecture, including a TUTA encoder and a three-stage decoder. **Right:** The detailed implementation of the formula sketch decoder.

3.1 TUTA Encoder

3.1.1 Input Tokenization

We flatten the header and input tabular cells to a cell sequence. We tokenize each cell of the sequence with a `[SEP]` token and the tokenization of the data following. We concatenate the tokenization of cells for the further encoding process. Note that the data for the target cell is replaced by a `[EMP]` token, as discussed in 2.1.

For numbers in spreadsheets, we not only consider them as plain text but also consider their numerical features. We extract numerical features including magnitude, precision, the top digit, and the low digit. After we tokenize the plain texts and obtain their embeddings, we concatenate the embeddings with the corresponding value feature vectors for further processing. For consistency, we set a default value feature for non-numeric words, representing the token is not a number. Such processing can handle various meanings of numbers in spreadsheets, such as amounts, scores, rates, times, years, and extract comprehensive information.

3.1.2 TUTA Encoder

We use TUTA (Wang *et al.*, 2021) encoder to compute contextual embedding for each token in the context. TUTA builds a tree structure through the hierarchy of headers together with corresponding tree embedding and tree attention. TUTA encoder enables us to extract hierarchical contextual information from both the header and cells. The weights of our TUTA encoder are initialized from TUTA pretrained model (Wang *et al.*, 2021). The hierarchical structure of tabular is reflected as structural neighborhood information in TUTA tree attentions.

3.2 Three-stage Decoder

In our model, we apply a coarse-to-fine decoding process which includes three stages. Firstly, we generate a formula sketch from scratch given the output cell encodings of the TUTA encoder. Secondly, we predict all the unfilled constant values in the generated formula sketch. Thirdly, we predict the cell references and obtain the prediction of the full formula.

3.2.1 Formula sketch decoder

The formula sketch decoder aims to generate the formula sketch with values and cells unfilled, occupied with `[ValueToken]`s and `[CellToken]`s. The target formula sketches are expressed in the infix form. We use a Transformer decoder in Seq2seq-style, and the decoder takes the contextual embedding as the encoding input. We add `[FormulaStart]` and `[formuland]` tokens to the tokenization of the target formula sketches representing the starting and the ending. Specifically, the input of the decoder is the concatenation of the encoding of the `[SEP]` token of the target cell and the embedding of the current sketch prefix that starts with `[FormulaStart]`. The output corresponding to the encoding of `[SEP]` will be removed since the encoding acts as an instruction signal and will not be further used.

3.2.2 Value token decoder

Unlike SpreadsheetCoder, we separate the generation of values from the generation of sketch and use another Transformer decoder for the Seq2seq-style generation of the value tokens. In our model, we fill the `[ValueToken]`s in a formula sketch after the end of sketch generation. Specifically, we add `[ValueStart] v1 [SEP] v2 [SEP] ... [ValueEnd]` after the formula sketch to represent the first value, the second value, etc., respectively. We use sketch-with-value to denote the concatenation of sketch and filled values. Note that if there is no `[ValueToken]`, the `[ValueStart]` and `[ValueEnd]` will not be added.

We believe the sketch-value separation can enhance the model performance since sketches and values follow different distributions. Without the sketch-value separation, generating incorrect value tokens during sketch generation could harmfully affect token generation afterward. We prove the advantage of the coarse-to-fine generation process with experiments.

3.2.3 Cell reference decoder

The cell reference is predicted implicitly. We use a decoder on the sketch-with-value. Unlike the previous two decoders with Seq2seq style and autoregressive masks, the cell reference decoder aims to learn the embeddings of `CellTokens`, and the attention mechanism of this decoder is almost fully-connected. For each `CellToken`, after we obtain the embedding, we measure the cosine similarity with the contextual embedding of all cells. We pick the most similar cell as the cell reference prediction. We add autoregressive mask among all `[CellToken]`s in order to distinguish different `[CellToken]`s in a formula sketch with values. This mechanism makes sure that previous `[CellToken]`s cannot see future ones, which is also in line with the human habit of filling in cell references from left to right. An example is shown in 4.

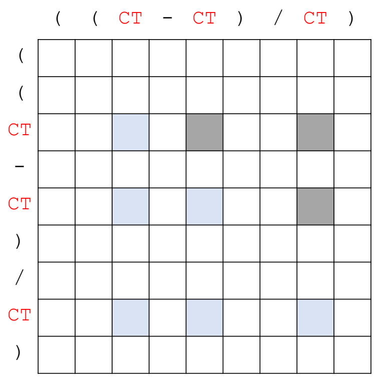


Figure 4: An example of cell mask in the cell reference decoder. "CT" stands for `CellToken`. A gray square is an attention mask, indicating that the position corresponding to the row cannot see that to the column. A white or light blue square indicates no mask.

3.3 Scoring Model

To handle the distribution shift between different decoding processes, we design a scoring model to score the sketches and sketch-with-values. We apply the scoring model after the formula sketch decoder and the value token decoder for reordering. The scoring model is simply another Seq2seq-style Transformer language model. The scoring model is trained on the ground truth sketches and sketch-with-values with equal weight and aims to minimize the negative log-likelihood for the shifted inputs.

4 Experiments

4.1 Dataset

We constructed our dataset from the Enron Corpus (Hermans and Murphy-Hill, 2015), and the FUSE corpus (Barik *et al.*, 2015). For spreadsheets that contain more than ten formulas with the same sketch and values, we randomly selected ten formulas to include in our dataset. During the data preprocessing, we have filtered out formulas containing unique tokens such as `NameToken`, `UserDefinedFuntion`, and formulas containing cross-sheet and cross-file cell references. Finally, we collected 106K training samples to form the Enron dataset and 30K training samples to form the FUSE dataset and split them into training and test set in the ratio of 7:3, ensuring formulas from the same spreadsheet are all in either training or test set.

As discussed in 2, each sample includes a formula, the cells in the header rows and columns of its corresponding spreadsheet, and the cells within the same row or column of the target cell. Note that we replace the target cell with an special token, and exclude the cells that contains the same content as the target cell. Each formula is represented by formula sketch, value tokens and cell references. For example, the formula `(SUM(A1:B2)/4)+1` will have sketch: `( ( ( SUM ( [CellToken] : [CellToken] ) ) / [NumberToken] ) + [NumberToken] )`, value tokens: `4`, `1` and cell references: `A1`, `B2`. Note that some formulas might not have value tokens and cell references.

4.2 Evaluation Setup

Metrics We evaluate the following metrics: (1) Sketch accuracy: the percentage of the generated sketches that are the same as the sketch of ground truth formula. (2) Sketch-with-value accuracy: the percentage of generated sketch-with-values (the concatenation of sketch and filled values) that are the same as the sketch-with-value of the ground truth formula. Note that the sketch-with-value accuracy in our setting is comparable with sketch accuracy in the setting of SpreadsheetCoder. (3) Full formula accuracy: the percentage of generated formulas (including sketch, value, and cell reference) that are the same as the ground truth formula. Obviously, sketch accuracy $>$ sketch-with-value accuracy $>$ full formula accuracy in the three-stage decoding process.

Note that the evaluated accuracy of our metric could be lower than semantic accuracy because different spreadsheet formulas could be semantically equivalent (Chen *et al.*, 2021).

Experimental Details The encoder of all our models are initialized from the pre-trained TUTA (Wang *et al.*, 2021).

All Transformer decoders in our model use 8 attention heads, 6 layers, and a hidden size of 768. When generating sequences of tokens, we use a beam size of 10. All the models are trained for 1,000,000 steps.

4.3 Results

In this section, we compare our results with the state-of-the-art method and compare the results using different variants of our model.

We have reproduced state-of-the-art method SpreadsheetCoder(Chen *et al.*, 2021) on our dataset. We present the detailed comparison of our method and SpreadsheetCoder on multi-stage accuracy in Table 1, where top-k accuracy measures how often the ground truth appears in the top-k predictions. Note that when the scoring model is available, predictions are sorted by output score of the scoring model, while without the scoring model, predictions are sorted in the beam search process.

We observe that our method significantly outperforms SpreadsheetCoder in both sketch-with-value accuracy and full formula accuracy. We have improved top-1 full formula accuracy by about 10 percentage points.

We have also done an ablation study to show the significance of the scoring model and the 3-stage decoding process. The result is presented in 2. It shows that adding a scoring model enhances the model performance because the ordering result of beam search could be inappropriate. Reordering according to the score enables the correct prediction to have higher priority, making the top-1 accuracy closer to top-5 accuracy.

The sketch-value separation avoids the negative effect of generating incorrect value tokens when generating sketches. Our coarse-to-fine process also allows attending to the previously generated sketch of high confidence when generating values, providing the value generation with more information, thus improving the sketch-with-value accuracy.

Table 1: Detailed comparison with SpreadsheetCoder (denoted as Coder). Sketch-w-val denote sketch-with-value. Note that the underlined values are accuracies on the part of the test set.

Method	**Our Model**		**Coder**	
Stage	Top-1	Top-5	Top-1	Top-5
Sketch	$\underline{83.2\%}$	$\underline{86.8\%}$	-	-
Sketch-w-val	$\underline{72.0\%}$	$\underline{78.6\%}$	59.6%	64.3%
Full formula	$\underline{50.3\%}$	$\underline{54.5\%}$	40.4%	43.6%

Table 2: Sketch-with-value accuracy on the test set. “-” means the corresponding component is removed from our full model. Note that the underlined values are accuracies on the part of the test set.

Approach	**Top-1**	**Top-5**
Full Model	$\underline{72.0\%}$	$\underline{78.6\%}$
- Scoring Model	$\underline{28.4\%}$	$\underline{78.3\%}$
- Sketch-value separation	65.1%	69.8%
- TUTA encoder	64.3%	71.2%
SpreadsheetCoder	59.6%	64.3%

5 Related Works

Predicting and thus smartly generating user suggestions based on contextual understanding of language is an important task inspired by the success of natural language processing. These tasks require an encoder to encode contextual information of the target position and a decoder to complete prediction or generation works from corresponding latent. IntelliCode Compose (Svyatkovskiy *et al.*, 2020) is an application on code as a sequential language. Motivated by large transformer models in NLP tasks, they propose Transformer GPT-C for codes, which is trained on for complete code snippets and used for auto-completion.

In contrast, table languages are much more complex than sequential languages. Besides relational tables where each column is homogeneous, tables can have more complicated structures where positional information is crucial. TUTA (Wang *et al.*,

2021) is a unified pre-training architecture for understanding generally structured tables that extract both spatial, hierarchical, and semantic information. TUTA establishes a bi-dimensional coordinate tree through the hierarchy of header and proposes tree embedding and tree attention, which can be used as an encoder to extract table information based on transformer structure.

SpreadSheetCoder (Chen *et al.*, 2021) proposes a two-stage decoding process for formula prediction with a BERT-based model as an encoder. They divide a formula into sketch and concrete ranges. In their definition, a formula sketch includes every token in the prefix representation of the parse tree of the spreadsheet formula, except for cell references which are replaced with a placeholder `RANGE` token. The sketch in their definition corresponds to sketch-with-value in our definition. References can be either a single cell or a range of cells, expressed as row and column coordinate relative to the target cell and using a split token for cases of a range. SpreadSheetCoder first generates the formula sketch and then predicts the concrete ranges as a token sequence using a shared LSTM but with different output layers.

6 Conclusion

This paper proposes a model for spreadsheet formula prediction based on the Seq2seq method. A novel tokenization method for numbers is designed. We applied the TUTA encoder to extract hierarchical contextual embeddings. We divide decoding into three stages with a Transformer decoder for each. We introduce a scoring model for reordering generation results. Our model achieves the state-of-the-art in the spreadsheet formula prediction field.

References

Efthimia Aivaloglou, David Hoepelman, and Felienne Hermans. A grammar for spreadsheet formulas evaluated on two large datasets. In *2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 121–130, 2015.

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

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

Felienne Hermans and Emerson Murphy-Hill. Enron's spreadsheets and related emails: A dataset and analysis. In *37th International Conference on Software Engineering, ICSE '15*, 2015. to appear.

Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. Intellicode compose: Code generation using transformer. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1433–1443, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. Tuta: Tree-based transformers for generally structured table pre-training. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1780–1790, 2021.