HERMES: Interactive Spreadsheet Formula Prediction via Hierarchical Formulet Expansion

Wanrong He[1]*, Haoyu Dong[2]†, Yihuai Gao[1], Zhichao Fan[2], Xingzhuo Guo[1], Zhitao Hou[2], Xiao Lv[2], Ran Jia[2], Shi Han[2], Dongmei Zhang[2]
[1]Tsinghua University, [2]Microsoft Research Asia
{hwr19,gao-yh18,gxz19}@mails.tsinghua.edu.cnn,
{hadong,v-zhichaofan,zhith,xilv,raji,shihan,dongmeiz}@microsoft.com

Abstract

We propose HERMES, the first approach for spreadsheet formula prediction via HiEraRchical forMulet ExpanSion, where hierarchical expansion means generating formulas following the underlying parse tree structure, and Formulet refers to commonly-used multi-level patterns mined from real formula parse trees. HERMES improves the formula prediction accuracy by (1) guaranteeing correct grammar by hierarchical generation rather than left-to-right generation and (2) significantly streamlining the token-level decoding with high-level Formulet. Notably, instead of generating formulas in a pre-defined fixed order, we propose a novel sampling strategy to systematically exploit a variety of hierarchical and multi-level expansion orders and provided solid mathematical proof, with the aim of meeting diverse human needs of the formula writing order in real applications. We further develop an interactive formula completion interface based on HERMES, which shows a new user experience in https://github.com/formulet/HERMES.

1 Introduction

Hundreds of millions of people use spreadsheets, such as Excel and Google Sheets, for data storage and management. The semi-structured context infers potential relations among data in cells. Users can add formulas in their spreadsheets to process and analyze data based on such relations. Although spreadsheet formula language targets general users and is much simpler than programming languages like C++, it is still difficult for most Excel users without any programming experience to master. To write a formula for a cell, users need to find appropriate formula functions, such as `IF` and `SUBTOTAL`, compose them with correct grammar, and fill in proper values and cell references. This process could be time-consuming and error-prone.

For computers, spreadsheet formula prediction is also a challenging task, which requires understanding both textual and numerical data, diverse table structures, and relationships between cells. Thanks to the advances in language model (LM) pretraining, natural language (NL) understanding, and spreadsheet table understanding and reasoning, researchers have developed systems that could predict the formula of a selected cell in table (Chen et al., 2021; Cheng et al., 2022). Those systems transform users from "writers" into "decision makers", greatly reducing the required efforts for writing formulas. We have to argue, though, directly applying autoregressive sequence generation to formula generation has fundamental weaknesses.

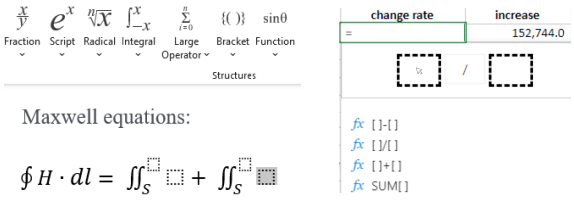
First, they fail to utilize the inner structures in spreadsheet formulas. Spreadsheet formulas have well-defined grammar and structure, including the usage of operators, parenthesis, constant values, cell references, and their combinations. Without taking the structure into consideration, a minor error made where the model is uncertain during the left-to-right decoding process might cause subsequent grammar mistakes (Guo et al., 2021; Yin and Neubig, 2017). Previous works have achieved promising results by changing the pure left-to-right fashion by decomposing formula generation into two stages, formula sketch prediction and cell range prediction (Chen et al., 2021; Cheng et al., 2022). But the underlying parse trees of formula sketches have been long neglected.

Second, the goal of spreadsheet formula prediction is to help spreadsheet users create desired formulas. Existing works generate entire formula sketches at once, but lack sufficient interaction with users. As users are only able to accept or reject the final predictions, and thus any mismatch in users' intents might lead to unpleasant post-editing and even from-scratch rewriting. How to interactively assist users in the process of creating formulas are well desirable to be explored.

*Work done during Wanrong's internship at Microsoft Research Asia.

†Corresponding author.

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 8356–8372



(a) Microsoft Word Equation (b) HERMES Interface

Figure 1: Interactive interface with variable input order.

In this work, we propose HERMES, the first approach to generate formulas through hierarchical Formulet expansion. Unlike existing works, we exploit the inherent tree structure of formulas, which guarantees that the generated formulas are grammatically correct. Furthermore, we find that many commonly-used formula patterns well reflect high-level reasoning, e.g., total and percentage change, so we extract Formulet, a set of 1-to-3 level expansions from real formula parse trees. To this end, we train the model to hierarchically expand nodes using Formulet, which avoids overly small granularity and too many steps to expand the entire parse tree, making the generation process concise and robust (Section 3.2.1). The full pipeline contains three stages. We first generate formula sketches that do not contain constant textual/numerical values nor cell references through hierarchical Formulet expansion, then generate textual/numerical values, and finally predict cell references.

Inspired by the user interface for creating equations in Word documents (Figure 1 (a)), we develop an interactive experience (Figure 1 (b)) that allows users to expand parse tree nodes in any order he/she prefers (Section 5). Unlike existing formula prediction systems that can only generate formulas in a fixed order, e.g., left-to-right, users here can drive the generation on their demands and potentially learn how to compose grammar-correct unfamiliar formulas. To support this experience, we propose a novel and systematic sampling strategy with solid mathematical proof to exploit various hierarchical expansion orders for training the HERMES decoder. For evaluation, we further introduce a metric: interaction upper bound (IUB) accuracy (Section 4.3), and we find that the IUB accuracy of our model is significantly higher than the accuracy of generating entire formulas at once, showing that the interactive framework can help users complete many more formula tasks.

2 Preliminaries

2.1 Formula Language and Parse Tree

The spreadsheet formula language used in this work comes from Microsoft Excel, consisting of operators, functions, cell references, and constant values. For a program P in a program language with known context-free grammar (CFG) $\mathcal{G}$, P could be parsed into abstract syntax tree (AST) according to $\mathcal{G}$ (Shin et al., 2019). Similar to programming languages, the spreadsheet formula also has its own context-free grammar and thus could be represented as a parse tree, which shows its hierarchical structure (Aivaloglou et al., 2015). We use XLParser (Aivaloglou et al., 2015) to obtain the raw parse tree of each formula. An example raw parse tree is shown in Appendix Figure 6. But we find that the parse tree derived by XLParser is too detailed and deep to be be the prediction target with many non-terminal nodes like `[Formula]`, `[Arguments]`, `[Argument]`, `[Reference]`, `[ReferenceFunctionCall]`, etc.

2.2 Compressed Parse Tree

We compress the raw parse tree by keeping `[Formula]` node as the only non-terminal and folding the other non-terminal nodes (Appendix B.1). Finally, we rename `[Formula]` as `[NonTerminal]` since it has became the only non-terminal node, and substitute all numerical values with `[NumberToken]`s, text values with `[TextToken]`s, and cell references with `[CellToken]`s to obtain the compressed parse tree. An example compressed parse tree is shown in figure 3. In the following sections, if not otherwise specified, parse tree refers to compressed parse tree.

Then we decompose the spreadsheet formula into formula sketch, textual and numerical values, and cell references. The formula sketch is formally defined as the formula with textual and numerical values replaced by `[TextToken]`s and `[NumberToken]`s, and cell references replaced by `[CellToken]`s. Then we link all the leaf nodes and bracket those nodes with the least common ancestors to form a sequence with a uniform bracketing rule. For example, `(SUM(A1:B2)/4)+1` has the formula sketch `( ( ( SUM ( [CellToken] : [CellToken] ) ) / [NumberToken] ) + [NumberToken] )`, textual and numerical values `4`,`1`, and cell references `A1`,`B2`. Since our model generates formula sketches by expanding nodes in the parse tree, the generated formula is guaranteed to be grammatically correct.

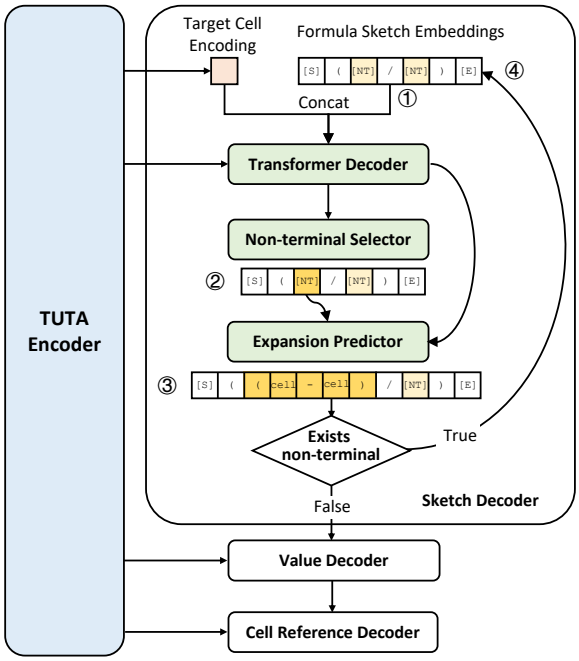


Figure 2: The overall architecture, including the TUTA Encoder and the three-stage decoder. The detail implementation of the Sketch Decoder and the iterative decoding process during inference: ① From a semi-finished formula sketch ② Select a `[NonTerminal]` token (represented as `[NT]`) ③ Predict an expansion ④ Replace selected `[NonTerminal]` with predicted expansion and get a new sketch. `[S]` represents the special token `[FormulaStart]`, and `[E]` represents the special token `[FormulaEnd]`.

3 HERMES Model

We present details of HERMES for spreadsheet formula prediction in this section. Figure 2 shows an overall architecture.

3.1 TUTA as Encoder

We follow the input specification of FORTAP (Cheng et al., 2022) (more details in Appendix B.3). The content of the target formula cell $\tau_{(s,t)}$ from table τ is replaced by a special token `[FORM]`, in order to indicate the model where to predict the formula. We use TUTA (Wang et al., 2021) – the first LM(BERT)-based spreadsheet pretraining method – as the encoder of HERMES to compute contextual toke and cell embeddings. TUTA leverages the hierarchical structure in table headers through tree embedding and tree attention and has a number encoding layer that utilizes magnitude, precision, and the first and last digit of numerical data. Understanding hierarchical contextual information from both header and data cells and understanding numerical data are important for formula prediction in semi-structured data, so we use TUTA as our encoder. Whereas HERMES is portable to be integrated into other encoders.

3.2 Three-stage Decoder

In our model, the decoding process includes three stages. First, we iteratively expand a formula sketch from initial `[NonTerminal]` using Sketch Decoder (SD). Second, we predict the textual and numerical values that correspond to the `[ValueToken]`s in the formula sketch using Value Decoder (VD). Third, we predict the cell references corresponding to `[CellToken]`s in the formula sketch using Cell Reference Decoder (CRD) and obtain the prediction of the complete formula. Note that the three decoders share the same Transformer decoder structure (Vaswani et al., 2017), but have different linear projectors.

3.2.1 Sketch Decoder (SD)

The Sketch Decoder aims to predict a single-step expansion given the encoding of the spreadsheet table and the current semi-finished formula sketch, which contains `[NonTerminal]` token(s). We obtain contextual encoding of the table from TUTA encoder, concatenate the encoding of `[FORM]` token that represents the target cell and the embedding of the current semi-finished formula sketch, then apply a shared Transformer Decoder (Vaswani et al., 2017) to obtain the sequence of representations of the tokens in semi-finished formula sketch $(h_1, \cdots, h_n)$. We use Non-terminal Selector and Expansion Predictor - two linear projectors - to predict where and how to expand, respectively.

To be more specific, the Non-terminal Selector takes the representation of all `[NonTerminal]`s in the semi-finished sketch $H = (h_{NT_1}, \cdots, h_{NT_k})$ as input, and predicts a probability distribution over each `[NonTerminal]` using

$$\mathbb{P}(NT_i) = Softmax(u^T H) \quad (1)$$

where $u \in \mathbb{R}^d$ is the vector parameter of Non-terminal Selector's linear projector, projecting h's into scalar logits. d is the dimension of h's. The Non-terminal Selector enables our model to expand nodes in the formula parse tree in arbitrary order instead of only fixed orders such as Depth First Search or Breadth First Search, and empowers our model to explore the optimal expansion order in formula generation.

Given a selected [NonTerminal] NT_i, the Expansion Predictor takes its representation h_{NT_i} as input, and predicts a probability distribution over expansions e in Formulet using

$$\mathbb{P}(e) = Softmax(Wh_{NT_i}) \quad (2)$$

where $W \in \mathbb{R}^{|\text{Formulet}| \times d}$ is the matrix parameter of Expansion Predictor's linear projector. Formulet is a set of common expansions collected in Section 3.2.1.

For inference, the generation process starts from a single [NonTerminal] token and applies beam search to iteratively select a [NonTerminal] and replace the [NonTerminal] with the predicted expansion until there is no [NonTerminal] left in the current formula sketch. The Sketch Decoder architecture and decoding process is shown in Figure 2.

Details for training Sketch Decoder with multi-level expansion are discussed in the following parts.

Multi-level expansion. Each non-leaf node in the compressed parse tree contains one or multiple [NonTerminal] tokens, each corresponds to a child node, where the child node is the node-representation of 1-level expansion of that [NonTerminal] token in the parent node, and the sequence of tokens that forms the child node is the sequence-representation of 1-level expansion of that [NonTerminal] token. Here we define n-level expansion recursively: the node-representation of n-level expansion of a root [NonTerminal] token contains the node-representation of (n-1)-level expansion of the root [NonTerminal] token, and additionally contains the 1-level expansion nodes for each unexpanded [NonTerminal] token in the (n-1)-level expansion. The sequence-representation of n-level expansion of a root [NonTerminal] token R can then be obtained by substituting each unexpanded [NonTerminal] token N in the sequence-representation of the (n-1)-level expansion by the sequence-representation of 1-level expansion of N.

Formulet. To avoid overly small expansion granularity and too many steps to expand the entire parse tree, we construct Formulet by extracting the sequence-representation of all 1-level expansions and top 90% frequent 2/3-level expansions from the parse trees in diverse high-quality datasets of Enron (Hermans and Murphy-Hill, 2015), Fuse (Barik et al., 2015), and Euses (Fisher and Rothermel, 2005), forming a set of 2,223 possible expansions of multiple levels. Formulet contains common patterns people used when writing spreadsheet formulas, like SUM ([CellToken] : [CellToken]), ([CellToken] - [CellToken]) / [CellToken], and IF (([CellToken] < [CellToken]), [NumberToken], [NumberToken]) . During generation, the Expansion Predictor is flexible to select a high-level expansion from Formulet in a single step, as well as take multiple steps of composing low-level expansions. The former one enables efficient generation for commonly-used formulas, while the latter has more flexibility to compose unseen formulas. Dynamically selecting expansions in multiple abstraction levels requires both low-level and high-level reasoning ability.

Optimization of Sketch Decoder. We propose a novel sampling strategy to systematically exploit hierarchical and multi-level expansion orders. We denote the set of nodes in the **complete** compressed parse tree as T^C. Each semi-finished formula sketch corresponds to a subset of visited nodes $T \subseteq T^C$ in the compressed parse tree, forming a smaller tree. We use "state" to denote the node set. We say a state T is reachable to state $\widetilde{T}$ if we can obtain state $\widetilde{T}$ by expanding a [NonTerminal] token in some nodes in T using Formulet. We use lattice $\mathcal{L}$ to represent the states and the reachability between them: each state T is a node in the lattice $\mathcal{L}$, and is the parent node of its reachable states. An example lattice is shown in figure 4.

As shown in the figure, we order the states so that all the states that correspond to n nodes in the compressed parse tree T^C are located on the nth row of the lattice $\mathcal{L}$. Since Formulet contains expansions in multiple levels, a state T on the ith row can directly reach to a state $\widetilde{T}$ on $(i+h)$th row, where $h \geq 1$. The initial state – the only state on the top row – corresponds to the set of a single root node in the **initial** compressed parse tree $T^I = \{\text{[NonTerminal]}\}$. The last state – the only state on the bottom row – corresponds to the complete compressed parse tree T^C.

In the view of lattice, the generation process is finding a path σ from the top root of the lattice T^I to the bottom of the lattice T^C. Let $S(\mathcal{L})$ be the set of all paths σ from T^I to T^C. Then our optimization goal for a single formula f (from table τ) with corresponding lattice $\mathcal{L}$ is to maximize

$$g(f) = \sum_{\sigma \in S(\mathcal{L})} \prod_{i=0}^{n(\sigma)-1} \mathbb{P}\left(T_{\sigma_{i+1}} | T_{\sigma_i}, \tau, \theta\right) \quad (3)$$

where $n(\sigma)$ is the length of path σ, T_{σ_i} is the state at step t in the path. $T_{\sigma_0} = T^I$ and $T_{\sigma_{n(\sigma)}} = T^C$. $\mathbb{P}\left(T_{\sigma_{i+1}}|T_{\sigma_i}, \tau, \theta\right)$ is the probability of reaching $T_{\sigma_{i+1}}$ from T_{σ_i} by selecting a `[NonTerminal]` token NT from a leaf node in T_{σ_i} and then doing an expansion e: $\mathbb{P}\left(T|T_{\sigma_i}, \tau, \theta\right) * \mathbb{P}(e)$.

It is impractical to train the model based on the exact probability by traversing through all $S(\mathcal{L})$ paths. Inspired by Shen et al., we utilize Jensen's inequality in Theorem 1 to estimate the lower bound of the optimization goal $g(f)$. We further rearrange the summation order in Theorem 2 to enable parallel optimization, which significantly boosts training speed.

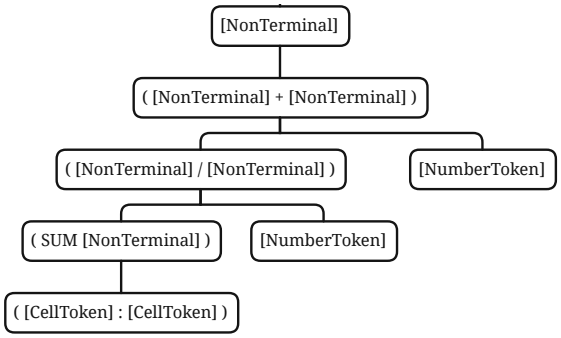


Figure 3: Compressed parse tree of `(SUM(A1:B2)/4)+1`.

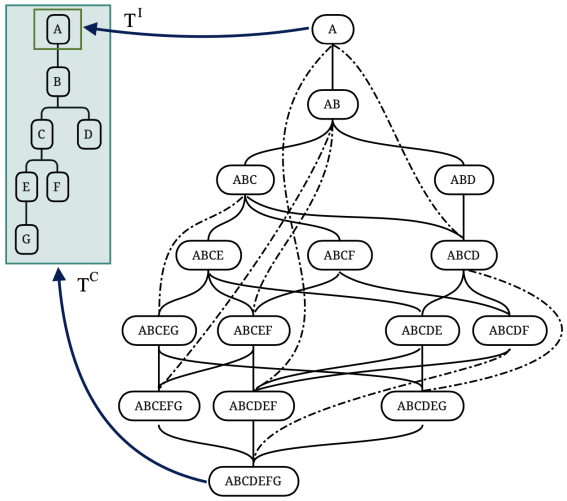


Figure 4: **Left:** each node of the compressed parse tree of `(SUM(A1:B2)/4)+1` is named with a letter. **Right:** corresponding lattice $\mathcal{L}$. Solid lines are for 1-level expansions, dotted lines are for 2/3-level expansions. The top root node `A` of the lattice corresponds to tree T^I, and the bottom node `ABCDEFG` corresponds to tree T^C.

Theorem 1. *For an arbitrary formula f and its corresponding lattice $\mathcal{L}(f)$, let $S(\mathcal{L})$ be the set of all paths σ, and T_{σ_t} be the t-th step of the path σ. $n(\sigma)$ represents the number of total steps in path σ. The logarithm of the optimization goal $g(f)$ can be lower-bounded by a summation of single step log-probabilities:*

$$\log g(f) \geq \log |S(\mathcal{L})| + \frac{1}{|S(\mathcal{L})|} \sum_{\sigma \in S(\mathcal{L})} \sum_{i=0}^{n(\sigma)-1} \log \mathbb{P}\left(T_{\sigma_{i+1}}|T_{\sigma_i}, \tau, \theta\right)$$

Theorem 2. *Let $\mathcal{C}(T)$ be the set of children of a parse tree T in lattice $\mathcal{L}$ and $R(T, T^C)$ be the number of different routes going from T to T^C. If we select a path σ in the following way:*

1. *Begin with $T_{\sigma_0} \coloneqq T^I$;*
2. *For $i = 0, 1, 2, ...$ we choose the next lattice node $T_{\sigma_{i+1}}$ among $\mathcal{C}(T_{\sigma_i})$, based on the distribution $\frac{R(T_{\sigma_{i+1}}, T^C)}{R(T_{\sigma_i}, T^C)}$;*
3. *End with $T_{\sigma_{n(\sigma)}} \coloneqq T^C$;*

all paths are sampled with equal probability. Then we optimize

$$\sum_{\widetilde{T} \in \mathcal{C}(T)} \frac{R(\widetilde{T}, T^C)}{R(T, T^C)} \log \mathbb{P}(\widetilde{T}|T, \tau, \theta)$$

for each parse tree T in the selected path σ, where $\mathbb{P}(\widetilde{T}|T, \tau, \theta)$ is the estimated probability of predicting $\widetilde{T}$ from T. In the expectation perspective, this is equivalent to optimize

$$\log |S(\mathcal{L})| + \frac{1}{|S(\mathcal{L})|} \sum_{\sigma \in S(\mathcal{L})} \sum_{i=0}^{n(\sigma)-1} \log \mathbb{P}\left(T_{\sigma_{i+1}}|T_{\sigma_i}, \tau, \theta\right)$$

, which is the lower bound of $g(f)$ in Theorem 1.

The efficient optimization process for Sketch Decoder and other modules is illustrated in Algorithm 1, which is based on the theoretical foundation in Theorem 1 and Theorem 2. We refer interested readers to Appendix A for detailed proof of the effectiveness of this algorithm.

Algorithm 1: HERMES Training

Initialize model parameters θ
$\forall T \in \mathcal{L}$, calculate $R(T, T^C)$ by enumertaion
repeat
  Sample a training example (table τ, formula f), where f has corresponding lattice $\mathcal{L}$
  $T \leftarrow T^I$
  $\sigma \leftarrow [T]$ `// We are going to uniformly sample a path` $\sigma \in S(\mathcal{L})$ `from` T^I `to` T^C
  while $T \neq T^C$ **do**
    $\widetilde{T} \leftarrow$ a child node of T sampled with probability $\frac{R(\widetilde{T}, T^C)}{R(T, T^C)}$
    Append $\widetilde{T}$ to σ
    $T \leftarrow \widetilde{T}$
  $\text{loss}_{\text{SD}} \leftarrow 0$
  foreach $T \in \sigma$ **do**
    $\text{loss}_{\text{SD}} \leftarrow \text{loss}_{\text{SD}} - \sum_{\widetilde{T} \in \mathcal{C}(T)} \left[\frac{R(\widetilde{T}, T^C)}{R(T, T^C)} \log \mathbb{P}(\widetilde{T} | T, \tau, \theta) \right]$
    `//` $\log \mathbb{P}(\widetilde{T} | T, \tau, \theta)$ `can be calculated by summing the output log probabilities of both the Non-terminal Selector and the Expansion Predictor`
  $\text{loss} \leftarrow \text{loss}_{\text{SD}} + \text{loss}_{\text{VD}} + \text{loss}_{\text{CFD}}$
  Update θ by gradient descent
until *Convergence*;

3.2.2 Value Decoder (VD)

Different from SpreadsheetCoder (Chen et al., 2021) and FORTAP (Cheng et al., 2022) which regard textual and numerical values as a part of the sketch, we separate the value from the sketch decoded by expansion-based Sketch Decoder. Given table encoding from TUTA encoder, the Value Decoder considers fully expanded formula sketch as already decoded tokens, and uses the shared Transformer decoder in a Seq2seq fashion (Sutskever et al., 2014) to further decode the textual and numerical values corresponding to each `[TextToken]` and `[ValueToken]` in formula sketch. Specifically, let $v_1, v_2, ..., v_n$ be the values corresponding to `[TextToken]`s and `[ValueToken]`s in the same order, where n is the total number of `[TextToken]`s and `[ValueToken]`s, then the target for the Value Decoder is `[ValueStart]` v_1 `[SEP]` v_2 `...` `[SEP]` v_n `[ValueEnd]`. $loss_{VD}$ is aggregated on each token's cross entropy. We use sketch-with-value to denote the concatenation of generated formula sketch and the decoded textual and numerical values, which could be equivalently transformed into a formula without cell reference by replacing special tokens with corresponding values.

3.2.3 Cell Reference Decoder (CFD)

Cells in HERMES are represented by the encoding of corresponding `[SEP]`s in the input sequence. Different from SpreadsheetCoder (Chen et al., 2021) that directly decodes cell addresses, we apply full attention to the sketch-with-value using the shared Transformer decoder, then for each `CellToken`, we measure the cosine similarity of its embedding with embeddings of all input cells and pick the most similar cell as the cell reference prediction. $loss_{CFG}$ represents the aggregated loss.

4 Experiments

4.1 Dataset

We construct our dataset from the Enron Corpus (Hermans and Murphy-Hill, 2015), a database containing over 17K spreadsheets with diverse table structures and rich formula types. We follow the data processing pipeline of Cheng et al. (2022) but additionally extracted formula parse trees. More implementation details are discussed in Appendix B.2. We collect 106K table-formula samples to form Enron dataset. We split them into train and test sets in the ratio of 7:3. We have also constructed a dataset from the FUSE Corpus (Barik et al., 2015) using the same pipeline (only reserve English spreadsheets) to further evaluate the model's generalizability, which contains 30K table-formula samples.

4.2 Baselines

We adopt SpreadsheetCoder and FORTAP (Chen et al., 2021; Cheng et al., 2022) as our baselines. SpreadsheetCoder is based on BERT (Devlin et al., 2019) and uses the first row of the table as header. FORTAP is state-of-the-art method for spreadsheet formula prediction. FORTAP is based on TUTA (Wang et al., 2021) and leverages spreadsheet formulas for table pretraining. Both of the models apply two-stage decoder for formula generation, first decoding sketch-with-value defined in Section 3.2.2 using an LSTM and then predicting cell references. Since SpreadsheetCoder has not publicly released its model and code with training details, we use the alternative implementation by Cheng et al..

4.3 Metrics

We evaluate the following metrics: (1) Sketch accuracy, (2) Sketch-with-value accuracy, (3) Cell reference accuracy, and (4) Formula accuracy, which

measures the percentage of correctly predicted formula sketch, sketch-with-value, all cell references in the formula, and the entire formula. Note that the sketch accuracy defined in (Chen et al., 2021; Cheng et al., 2022) corresponds to sketch-with-value accuracy defined in our work. Formula is only correct when sketch, sketch-with-value, and cell-reference are correct. Sketch accuracy $\geq$ sketch-with-value accuracy $\geq$ formula accuracy in the three-stage decoding process.

We propose a new metric: (5) interaction upper bound (IUB) accuracy, which provides a new aspect for evaluating how well an algorithm performs with potential human interaction. We consider that the evaluation of AI models should be human-centered. In other words, how well can a model help humans complete their tasks, e.g., writing a formula. Existing works all generate entire formulas at once, directly showing users the top-1 or top-n results without intermediate interactions, but it may cause more failed cases without users' inputs.

IUB accuracy is formally defined as the formula prediction accuracy when a user is able to select from the model recommendations in each round of interactions, e.g., the top-5 predicted expansions in each step, with a limited total number of rounds no bigger than 5. To calculate IUB accuracy, we simulate a user who actively interacts with the model by always choosing the best recommendation, which is the recommendation that potentially leads to a correct formula generation.

4.4 Experimental Details

Encoders of our models are initialized from the pre-trained TUTA (Wang et al., 2021). All our models have first trained $1M$ steps for Sketch Decoder on Enron dataset, keeping Value Decoder and Cell Reference Decoder frozen, then trained $1M$ steps for the whole model on Enron dataset. The beam size is 10 for both Sketch Decoder and Value Decoder. Hyperparameters are presented in Appendix B.4. Experiment results are from single runs.

4.5 Results

Table 1 summarizes the performance of different models when predicting sketch and sketch-with-value on the Enron test set, and Table 2 summarizes the performance of predicting cell references and entire formula. As shown, although without pre-training, HERMES outperforms FORTAP – current state-of-the-art – by 2.3% on top-1 sketch-with-value accuracy, and 4.5% on top-1 formula accuracy, showing the effectiveness of the three-stage decoding process and generating formula sketch by hierarchical expansion. Importantly, our model achieves **96.9%** in IUB sketch accuracy, significantly outperforms the top-5 sketch accuracy of 87.6%, showing that our interactive framework can help users to complete much more formula sketches than just selecting from a final generation list. However, the IUB formula accuracy drops to 69.0%, showing that there still exists a challenge in cell reference and constant value prediction.

Model	Sketch		Sketch-w-value	
	Top-1	**Top-5**	**Top-1**	**Top-5**
SpreadsheetCoder	-	-	59.6%	73.6%
FORTAP	-	-	70.8%	79.4%
HERMES	**78.8%**	**87.6%**	**73.1%**	**80.1%**

Table 1: Sketch accuracy and sketch-with-value accuracy on Enron.

Model	**Cell**	**Formula**	**IUB Formula**
SpreadsheetCoder	67.7%	40.4%	-
FORTAP	78.8%	55.8%	-
HERMES	**82.4%**	**60.3%**	**69.0%**

Table 2: Cell reference accuracy, top-1 formula accuracy and IUB formula accuracy on Enron.

We also test our model on FUSE dataset without further finetuning (Table 3). Top-1 sketch accuracy only drops by 8.2%, indicating generalizability since Formulet is the formula structure shared in different spreadsheet tables. More evaluation results on FUSE are discussed in Appendix C.

Ablation study results in Table 4 demonstrate the importance of different modules in Sketch Decoder. When restricting the model to only use 1-level expansion in the Formulet, the model performance significantly decreases. As shown in Table 5, multi-level expansion reduces the steps of expansions required to generate a formula sketch, increases the probability of expanding the formula sketch

Testset	**Top-1**	**Top-5**
Enron	78.8%	87.6%
FUSE	70.6%	74.3%

Table 3: Sketch accuracy on Enron and FUSE.

in a single step and reduces the uncertainty in the sketch generation process. In addition, we observe that adding Non-terminal Selector brings performance gain regarding top-5 accuracy, but slightly decreases top-1 accuracy. We could consider Non-terminal Selector as a route planner in a distribution: it tends to select an "easier" order to predict expansions of nodes in the compressed parse tree, which corresponds to a route in the lattice, but at the same time, all feasible orders are possibly chosen because it has been assigned multiple ways in the training phase through our sampling strategy. We conjecture that adding Non-terminal Selector improves the flexibility when selecting expansion orders, thus improving the possibility of generating a correct formula sketch when given more chances. But in the top-1 setting, the model might have explored too many variants while lacking sufficient training in a specific order.

Approach	**Top-1**	**Top-5**
Full Model	78.8%	**87.6%**
- Non-terminal Selector	**81.3%**	84.9%
- Multi-level expansion	40.5%	68.7%

Table 4: Sketch accuracy of model variants.

5 Interactive Formula Generation

We propose interactive formula generation, allowing users to participate in the formula generation process, which potentially helps users to complete more tasks as indicated in Section 4.5. As shown in Figure 5 (a), users could start the interaction process by simply typing "=" in the target cell, then ten recommended (semi-finished) formula sketches will popup in a tooltip for users to select. They are generated by running the Expansion Predictor and may include different levels of expansions. Users need to select one of the sketches from the tooltip, and it will appear in the top bar of the tooltip as the updated (semi-finished formula) sketch.

Sequence	**1-level exp**	**Multi-level exp**
4.40±1.83	2.54±0.82	2.31±0.81

Table 5: Average required steps for sketch completion by different methods. Compared to sequence generation ("Sequence"), where required steps are the number of tokens, expansion-based ("exp") methods greatly reduce the required steps for generating sketches. Formulet-based multi-level expansion achieves minimal steps.

If the updated semi-finished formula sketch contains one or more blank boxes (`[NonTerminal]`s) (Figure 5 (b)), users can select one blank box for the next round of Expansion Prediction, then again select an expanded sketch from the updated tooltip, and repeat this process until the formula sketch does not contain any blank boxes (Figure 5 (c)). Note that the `[NonTerminal]` blank box with the highest probability predicted by Non-terminal Selector is selected by default, and corresponding Expansion Predictions are automatically shown, so it's not necessary for users to make blank box selection if they do not want to change the expansion order. Finally, the interactive system will complete the formula with constant values and cell references.

This interface is developed upon (Srinivasa Ragavan et al., 2022) using JavaScript. We implement table detection, header detection, and feature extraction following (Wang et al., 2021) in the frontend as inputs, and the HERMES model is wrapped to be a backend service using FastAPI.

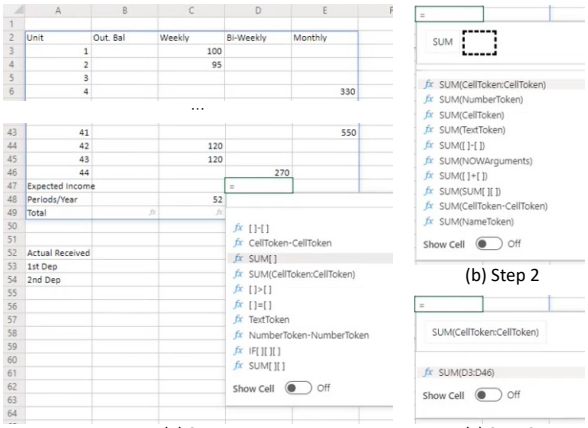


Figure 5: Demonstration of the interactive formula generation interface. For simplicity and easy understanding, a `[NonTerminal]` is represented as a blank dotted box (in semi-finished formula in the top bar) or a "[]" (in sketch recommendations).

6 Related Works

We introduced SpreadsheetCoder and FORTAP in Section 4.2. We believe that table pertaining methods in FORTAP can be combined with our advanced model structure to achieve even greater gains. While TabT5 (Andrejczuk et al.) is a concurrent work that has been evaluated on spreadsheet formula prediction using a pre-trained language model for left-to-right sequential decoding, we have not included it as a baseline due to the

lack of publicly available source code, pre-trained model, and the dataset processing details.

Recent works in code generation have explored leveraging grammar trees for structured code generation, showing advantages over left-to-right sequential decoding (Yin and Neubig, 2017; Brockschmidt et al., 2018). For example, (Shin et al., 2019) generates code based on high-level patterns (idioms), but in a fixed depth-first order. (Guo et al., 2021) recently proposes to generate code completions in flexible orders while using only low-level granularity. HERMES explores controllable hierarchical expansion orders based on both high-level and low-level expansions in Formulet.

More broadly speaking, structured generation leverages inherent structural information of tasks such as grammar and semantics. (Dong and Lapata, 2016) proposed to generate the logical forms of input utterances with a hierarchical tree decoder. (Li et al., 2020) additionally considers input objects as graphs and generates tree outputs without specific assumption on downstream tasks.

Existing works on flexible generation mainly investigates methods for generating sequences in arbitrary order. (Stern et al., 2019) generate sequences based on token insertions during decoding. (Shen et al., 2020) proposes to generate sequences by iteratively inserting and filling in blanks, further allowing users to specify where to insert. In spreadsheet formula generation, HERMES continues the idea of (Shen et al., 2020) where users can instruct the model on where and what to expand given the formula sketch.

We are the first to leverage the tree structure in spreadsheet formula generation. We are also the first work in code generation to combine idiom-based generation (Shin et al., 2019) with variable hierarchical generation orders. We have derived an optimization algorithm that applies systematic sampling to deal with the idiom-based hierarchical generation which may contain skip-level expansions, and provided solid mathematical proof. We believe our approach is a novel exploration of code generation and can inspire future work in this field.

7 Discussion

`Is the grammar generated by HERMES 100% correct?`

If we ignore the variable type, we can prove that the formula sketch generated by HERMES (Sketch Decoder) is correct by simple mathematical induction: we have that the expansions (including formula sketches and terminals) in Formulet are grammatically correct, and the initial (semi-finished) formula sketch "[NonTerminal]" is also correct. Given a correct semi-finished formula sketch, its grammar is still correct by expanding a non-terminal in it into either a formula sketch or a terminal. Thus the formula sketch is grammatically correct.

Even if we take variable type into consideration, we can still easily avoid grammar mistakes: each non-terminal in the formula sketch corresponds to a specific variable type And the calculation result of each expansion in Formulet also has a fixed variable type (except for very few functions such as VLOOKUP).

8 Conclusion

HERMES is the first framework for hierarchical spreadsheet formula expansion. It streamlines token-level decoding with multi-level Formulets and devises a systematic sampling strategy to exploit various hierarchical multi-level expansion orders. Experiment results show significant improvements in prediction accuracy. Importantly, it enables a new interactive formula completion experience in our developed interface. We hope for HERMES to stimulate more advances in leveraging underlying structures for spreadsheet formula prediction.

Ethics Statement

In this work, we propose an interactive framework for spreadsheet formula prediction via hierarchical Formulet expansion. The datasets we use are all collected from public spreadsheet datasets, Enron, FUSE, and Euses, and we follow SpreadsheetCoder and FORTAP to use Enron for formula prediction training and testing. We additionally use FUSE for training and testing to check the model's generalizability. The data we used are all in English and has no privacy issue. It does not contain any information that names or uniquely identifies individual people or offensive content. We do not collect additional spreadsheet data in this work. The TUTA model we used in this research is open source under the MIT License. Our code will be released to the public under the MIT License. We think that the interactive formula generation can reduce the affection on user experience when faced with challenging cases and incorrect prediction occurs. Our framework has been deployed at GridBook (Srinivasa Ragavan et al., 2022) for internal testing, and we hope it can stimulate new experiences in real products such as Excel and Google Sheets to benefit a large number of real spreadsheet users. Although our model predicts expansions with high accuracy in each step, it is still possible that our model cannot meet the users' specific needs, requiring the users to write the expansions or formulas by themselves.

References

Efthimia Aivaloglou, David Hoepelman, and Felienne Hermans. 2015. 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.

Ewa Andrejczuk, Julian Martin Eisenschlos, Francesco Piccinno, Syrine Krichene, and Yasemin Altun. Table-to-text generation and pre-training with tabt5.

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*, pages 486–489.

Marc Brockschmidt, Miltiadis Allamanis, Alexander L Gaunt, and Oleksandr Polozov. 2018. Generative code modeling with graphs. *arXiv preprint arXiv:1805.08490*.

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

Zhoujun Cheng, Haoyu Dong, 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)*, pages 1150–1166, Dublin, Ireland. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany. Association for Computational Linguistics.

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*, pages 1–5.

Daya Guo, Alexey Svyatkovskiy, Jian Yin, Nan Duan, Marc Brockschmidt, and Miltiadis Allamanis. 2021. Learning to complete code with sketches. In *International Conference on Learning Representations*.

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

Shucheng Li, Lingfei Wu, Shiwei Feng, Fangli Xu, Fengyuan Xu, and Sheng Zhong. 2020. Graph-to-tree neural networks for learning structured input-output translation with applications to semantic parsing and math word problem. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2841–2852, Online. Association for Computational Linguistics.

Tianxiao Shen, Victor Quach, Regina Barzilay, and Tommi Jaakkola. 2020. Blank language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5186–5198, Online. Association for Computational Linguistics.

Eui Chul Shin, Miltiadis Allamanis, Marc Brockschmidt, and Alex Polozov. 2019. Program synthesis and semantic parsing with learned

code idioms. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Sruti Srinivasa Ragavan, Zhitao Hou, Yun Wang, Andrew D Gordon, Haidong Zhang, and Dongmei Zhang. 2022. Gridbook: Natural language formulas for the spreadsheet grid. In *27th International Conference on Intelligent User Interfaces*, pages 345–368.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: Flexible sequence generation via insertion operations. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5976–5985. PMLR.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

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

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

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696*.

A Proof of The Optimization Process for Sketch Decoder

In this section, we first prove Theorem 1 to get a lower bound of the optimization goal $g(f)$, then introduce the following lemmas to prove Theorem 2.

Theorem 1. *For an arbitrary formula f and its corresponding lattice $\mathcal{L}(f)$, let $S(\mathcal{L})$ be the set of all paths σ, and T_{σ_t} be the t-th step of the path σ. $n(\sigma)$ represents the number of total steps in path σ. The logarithm of the optimization goal $g(f)$ can be lower-bounded by a summation of single step log-probabilities:*

$$\log g(f) \geq \log |S(\mathcal{L})| + \frac{1}{|S(\mathcal{L})|} \sum_{\sigma \in S(\mathcal{L})} \sum_{i=0}^{n(\sigma)-1} \log \mathbb{P}\left(T_{\sigma_{i+1}} | T_{\sigma_i}, \tau, \theta\right)$$

Proof.

$$\begin{aligned}
&\log \left(\sum_{\sigma \in S(\mathcal{L})} \prod_{i=0}^{n(\sigma)-1} \mathbb{P}\left(T_{\sigma_{i+1}} | T_{\sigma_i}, \tau, \theta\right) \right) \\
=& \log S(\mathcal{L}) + \\
& \log \frac{1}{|S(\mathcal{L})|} \sum_{\sigma \in S(\mathcal{L})} \prod_{i=0}^{n(\sigma)-1} \mathbb{P}\left(T_{\sigma_{i+1}} | T_{\sigma_i}, \tau, \theta\right) \\
\geq& \log S(\mathcal{L}) + \\
& \frac{1}{|S(\mathcal{L})|} \sum_{\sigma \in S(\mathcal{L})} \log \prod_{i=0}^{n(\sigma)-1} \mathbb{P}\left(T_{\sigma_{i+1}} | T_{\sigma_i}, \tau, \theta\right) \\
=& \log S(\mathcal{L}) + \\
& \frac{1}{|S(\mathcal{L})|} \sum_{\sigma \in S(\mathcal{L})} \sum_{i=0}^{n(\sigma)-1} \log \mathbb{P}\left(T_{\sigma_{i+1}} | T_{\sigma_i}, \tau, \theta\right)
\end{aligned}$$

Since log is a concave function, the lower bound in the second step is provided by Jensen's inequality. □

Lemma 1. *If we follow the path selection strategy in Theorem 2, the sample probability for each path is uniformly distributed, i.e. $\forall \sigma \in S(\mathcal{L}), \mathbb{P}(\sigma) = \frac{1}{|S(\mathcal{L})|}$.*

Proof. Note that all steps of $\sigma = \{T_{\sigma_i}\}_{i \geq 0}$ forms a Markov chain, i.e.

$$p(T_{\sigma_i} | T_{\sigma_{i-1}} \cdots T_{\sigma_0}) = p(T_{\sigma_i} | T_{\sigma_{i-1}})$$

where

$$p(T_{\sigma_i} | T_{\sigma_{i-1}}) = \frac{R(T_{\sigma_{i+1}}, T^C)}{R(T_{\sigma_i}, T^C)}$$

can be calculated before the training process, and is different from the target distribution $\mathbb{P}$. According to the definition, we have $N(T^I) = |S(\mathcal{L})|$ and $N(T^C) = 1$. Then

$$\begin{aligned}
p(\sigma) &= p(T_{\sigma_{n(\sigma)}} \cdots T_{\sigma_0}) \\
&= p(T_{\sigma_0}) \prod_{i=1}^{n(\sigma)} p(T_{\sigma_i} | T_{\sigma_{i-1}} \cdots T_{\sigma_0}) \\
&= \prod_{i=1}^{n(\sigma)} p(T_{\sigma_i} | T_{\sigma_{i-1}}) \\
&= \prod_{i=1}^{n(\sigma)} \frac{R(T_{\sigma_i}, T^C)}{R(T_{\sigma_{i-1}}, T^C)} \\
&= \frac{R(T_{\sigma_{n(\sigma)}}, T^C)}{R(T_{\sigma_0}, T^C)} \\
&= \frac{R(T^C, T^C)}{R(T^I, T^C)} \\
&= \frac{1}{|S(\mathcal{L})|}
\end{aligned}$$

□

Lemma 2. *Let $\zeta(T)$ be an arbitrary function of a parse tree T and use $\mathbb{E}_\sigma$ to denote an expectation based on uniformly distributed $\sigma \in S(\mathcal{L})$, then we have*

$$\mathbb{E}_\sigma \left[\sum_{i=0}^{n(\sigma)-1} \zeta(T_{\sigma_i}) \right] = \frac{1}{|S(\mathcal{L})|} \sum_{T \in \mathcal{L}} N(T) \zeta(T)$$

Proof.

$$\begin{aligned}
&\mathbb{E}_\sigma \left[\sum_{i=0}^{n(\sigma)-1} \zeta(T_{\sigma_i}) \right] \\
=& \sum_{\sigma \in S(\mathcal{L})} p(\sigma) \sum_{i=0}^{n(\sigma)-1} \zeta(T_{\sigma_i}) \\
=& \frac{1}{|S(\mathcal{L})|} \sum_{\sigma \in S(\mathcal{L})} \sum_{T \in \mathcal{L}} \mathbb{1}_{\{T \in \sigma\}} \zeta(T) \\
=& \frac{1}{|S(\mathcal{L})|} \sum_{T \in \mathcal{L}} \left(\sum_{\sigma \in S(\mathcal{L})} \mathbb{1}_{\{T \in \sigma\}} \right) \zeta(T) \\
=& \frac{1}{|S(\mathcal{L})|} \sum_{T \in \mathcal{L}} N(T) \zeta(T)
\end{aligned}$$

□

Lemma 3. *Let $\mathcal{C}(T)$ be the set of child nodes of a parse tree T, $R(T_1, T_2)$ be the number of routes that begin with T_1 and end with T_2, and $N(T), N(T \to \widetilde{T})$ denote the number of all paths $\sigma \in S(\mathcal{L})$ passing though T and the single expansion step $T \to \widetilde{T}$, respectively. Then we have*

$$\sum_{T \in \mathcal{L}} N(T) \sum_{\widetilde{T} \in \mathcal{C}(T)} \left[\frac{R(\widetilde{T}, T^C)}{R(T, T^C)} \log \mathbb{P}(\widetilde{T} | T, \tau, \theta) \right]$$
$$= \sum_{\sigma \in S(\mathcal{L})} \sum_{i=0}^{n(\sigma)-1} \log \mathbb{P}\left(T_{\sigma_{i+1}} | T_{\sigma_i}, \tau, \theta\right)$$

Proof. According to the definition of $N(T)$ and $N(T \to \widetilde{T})$, we have the following identity:

$$N(T) = R(T^I, T) R(T, T^C)$$

$$N(T \to \widetilde{T}) = R(T^I, T) R(\widetilde{T}, T^C)$$

Then we have

$$\sum_{T \in \mathcal{L}} N(T) \sum_{\widetilde{T} \in \mathcal{C}(T)} \left[\frac{R(\widetilde{T}, T^C)}{R(T, T^C)} \log \mathbb{P}(\widetilde{T} | T, \tau, \theta) \right]$$
$$= \sum_{T \in \mathcal{L}} R(T^I, T) R(T, T^C) \cdot \sum_{\widetilde{T} \in \mathcal{C}(T)} \left[\frac{R(\widetilde{T}, T^C)}{R(T, T^C)} \log \mathbb{P}(\widetilde{T} | T, \tau, \theta) \right]$$
$$= \sum_{T \in \mathcal{L}} R(T^I, T) \sum_{\widetilde{T} \in \mathcal{C}(T)} \left[R(\widetilde{T}, T^C) \log \mathbb{P}(\widetilde{T} | T, \tau, \theta) \right]$$
$$= \sum_{T \in \mathcal{L}} \sum_{\widetilde{T} \in \mathcal{C}(T)} \left[R(T^I, T) R(\widetilde{T}, T^C) \log \mathbb{P}(\widetilde{T} | T, \tau, \theta) \right]$$
$$= \sum_{T \in \mathcal{L}} \sum_{\widetilde{T} \in \mathcal{C}(T)} \left[N(T \to \widetilde{T}) \log \mathbb{P}(\widetilde{T} | T, \tau, \theta) \right]$$
$$= \sum_{T, \widetilde{T} \in \mathcal{L}} \left[\mathbb{1}_{\{\widetilde{T} \in \mathcal{C}(T)\}} N(T \to \widetilde{T}) \log \mathbb{P}(\widetilde{T} | T, \tau, \theta) \right]$$
$$= \sum_{T, \widetilde{T} \in \mathcal{L}} \sum_{\sigma \in S(\mathcal{L})} \mathbb{1}_{\{\widetilde{T} \in \mathcal{C}(T)\}} \mathbb{1}_{\{T, \widetilde{T} \in \sigma\}} \log \mathbb{P}(\widetilde{T} | T, \tau, \theta)$$
$$= \sum_{\sigma \in S(\mathcal{L})} \sum_{i=0}^{n(\sigma)-1} \log \mathbb{P}\left(T_{\sigma_{i+1}} | T_{\sigma_i}, \tau, \theta\right)$$

where $\mathbb{1}$ denotes the characteristic function. □

With the above lemmas proved, we are fully equipped to prove Theorem 2.

Theorem 2. *Let $\mathcal{C}(T)$ be the set of children of a parse tree T in lattice $\mathcal{L}$ and $R(T, T^C)$ be the number of different routes going from T to T^C. If we select a path σ in the following way:*

1. *Begin with $T_{\sigma_0} \coloneqq T^I$;*
2. *For $i = 0, 1, 2, ...$ we choose the next lattice node $T_{\sigma_{i+1}}$ among $\mathcal{C}(T_{\sigma_i})$, based on the distribution $\frac{R(T_{\sigma_{i+1}}, T^C)}{R(T_{\sigma_i}, T^C)}$;*
3. *End with $T_{\sigma_{n(\sigma)}} \coloneqq T^C$;*

all paths are sampled with equal probability. Then we optimize

$$\sum_{\widetilde{T} \in \mathcal{C}(T)} \frac{R(\widetilde{T}, T^C)}{R(T, T^C)} \log \mathbb{P}(\widetilde{T} | T, \tau, \theta)$$

for each parse tree T in the selected path σ, where $\mathbb{P}(\widetilde{T} | T, \tau, \theta)$ is the estimated probability of predicting $\widetilde{T}$ from T. In the expectation perspective, this is equivalent to optimize

$$\log |S(\mathcal{L})| + \frac{1}{|S(\mathcal{L})|} \sum_{\sigma \in S(\mathcal{L})} \sum_{i=0}^{n(\sigma)-1} \log \mathbb{P}\left(T_{\sigma_{i+1}} | T_{\sigma_i}, \tau, \theta\right)$$

, which is the lower bound of $g(f)$ in Theorem 1.

Proof. According to Lemma 1, probability of choosing any one of the valid paths from T^I to T^C is

$$p(\sigma) = \frac{1}{|S(\mathcal{L})|}$$

We let

$$\zeta(T) \coloneqq \sum_{\widetilde{T} \in \mathcal{C}(T)} \frac{R(\widetilde{T}, T^C)}{R(T, T^C)} \log \mathbb{P}(\widetilde{T} | T, \tau, \theta)$$

Since all paths are uniformly sampled, by applying Lemma 2 the parallel optimization of $\zeta(T)$ for all $T \in \sigma$ is equivalent to optimizing

$$\sum_{T \in \mathcal{L}} N(T) \sum_{\widetilde{T} \in \mathcal{C}(T)} \left[\frac{R(\widetilde{T}, T^C)}{R(T, T^C)} \log \mathbb{P}(\widetilde{T} | T, \tau, \theta) \right]$$

in expectation, which is exactly

$$\sum_{\sigma \in S(\mathcal{L})} \sum_{i=0}^{n(\sigma)-1} \log \mathbb{P}\left(T_{\sigma_{i+1}} | T_{\sigma_i}, \tau, \theta\right)$$

according to Lemma 3.

□

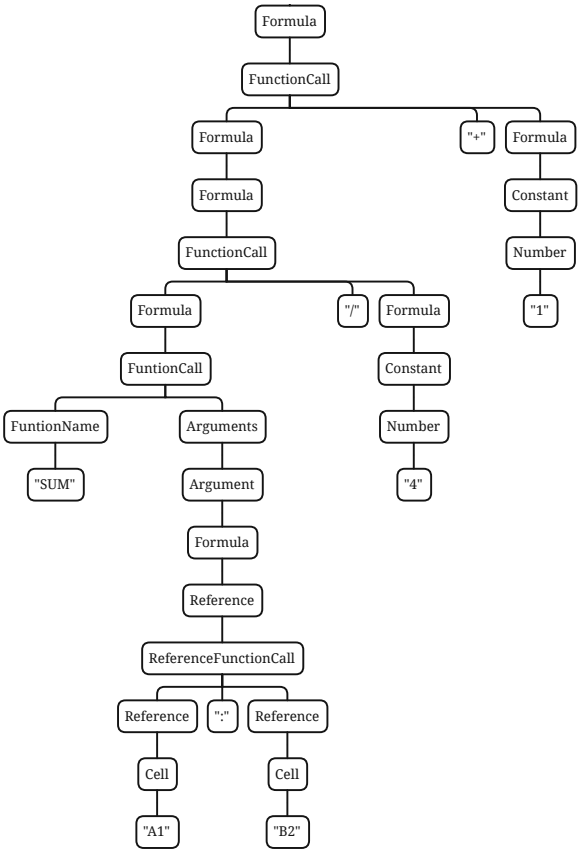


Figure 6: Raw parse tree of `(SUM(A1:B2)/4)+1` obtained from XLParser. Terminals are wrapped in "".

B Implementation Details

B.1 Compressed Parse Tree

Considering that the parse tree is too detailed and deep, we then compress the raw parse tree by keeping `Formula` node as the only non-terminal. The subtree of a `Formula` node up to terminals and other `Formula` nodes are compressed into a single node containing multiple tokens - the linking of leaf nodes in this subtree. Finally, we rename `Formula` as `[NonTerminal]` since it has became the only non-terminal token, and substitute all numerical values with `[NumberToken]`s, text values with `[TextToken]`s, and cell references with `[CellToken]`s to obtain the compressed parse tree.

B.2 Data Preprocessing

During data preprocessing, we have filtered out formulas containing special tokens such as `NameToken`, `UserDefinedFuntion`, and formulas containing cross-sheet and cross-file cell references. We also filter out spreadsheets where resulting input sequence contains more than 600 tokens. A spreadsheet may contain multiple duplicated formulas. For each spreadsheet, we only keep at most 10 formulas from a set of duplicated formulas by random sampling. We ensure no data leakage during the dataset splitting: the formulas from the same spreadsheet are either all in training or all in test set.

B.3 Input Specification

Following Wang et al. (2021), the input tokens of the model is the cell contents in the table τ flattened from left to right and from top to bottom, forming a linear sequence `[CLS] [SEP]` $\tau_{(0,0)}$ `[SEP]` $\tau_{(0,1)}$ ... `[SEP]` $\tau_{(m-1,n-1)}$, where $\tau_{(i,j)}$ represent the tokenized string content of cell in the $(i+1)^{th}$ row and $(j+1)^{th}$ column, m and n refers to the number of rows and number of columns in table τ, respectively. The data of the target formula cell $\tau_{(s,t)}$ is replaced by a special token `[FORM]`, in order to specify the position of the target cell and tell the model where to predict the formula. For other formulas cells, they are represented as formula sketch filled with textual and numerical values. Note that we blank out the cells that are same as the target formula cell with the form of relative cell references, e.g., `SUM(R[-2]C[-3], R[-1]C[0])`. Following Wang et al. (2021), numerical, positional and formatting features are further extracted from each input cells.

More than 89% formulas only reference cells that are on the same row or column as the formula cell (Cheng et al., 2022). For simplicity and as a compromise to the limitation of model memory capacity, given the target cell, instead of taking all the cells in the table as input, we only take the cells in the table header and cells on the same row/column as inputs. We assume that these cells provide enough information for formula prediction.

B.4 Model Hyperparameters

All our models are first trained $1M$ steps for Sketch Decoder on Enron dataset, keeping Value Decoder and Cell Reference Decoder freezed, then trained $1M$ steps for the whole model for together generating sketches, values and cell references on Enron dataset. Our models are trained with batch size 1 and max sequence length 600 without parallelism on Nvidia V100 GPUs with 16GB memory. We skip the samples with more than 600 tokens. Each phase is estimated to be trained for 9 epochs, i.e., each sample in our trainset are used for about 9 times in each phase. The optimizer is Adam with learning rate $2e-6$. The hyperparameters are

manually tuned by the authors. For reference, our model has a parameter size of 265,453,990, where a parameter size of 133,914,766 is inherited from the TUTA model.

C Supplementary Results

As the first work in spreadsheet formula generation to evaluate on the FUSE dataset, we mainly use FUSE as an auxiliary dataset to test whether our model is overfitting on Enron and whether it can generalize to other datasets. We have also done an experiment to test the full potential of our model on FUSE. We start from the model trained 1,000,000 steps on the Enron dataset for sketch prediction, and further train it for 1,000,000 steps on the FUSE dataset to generate the entire formula, using the same hyperparameters specified in Section B.4. The model achieves sketch accuracy of 0.710 (top 1) and 0.825 (top 5), sketch-with-value accuracy of 0.677 (top 1) and 0.782 (top 5), and formula accuracy of 0.514 (top 1) and 0.575 (top 5), which is higher than the generalization performance in Table 3 as well as the baseline performance: FORTAP only achieves top-1 formula accuracy of 0.37 and SpreadsheetCoder only achieves top-1 formula accuracy of 0.26.

D Limitations

Until now HERMES has only been evaluated on the formula prediction task, but we believe that the three-stage decoding pipeline and the sampling strategy for multi-level expansion in HERMES can further be extended to other forms of code such as SQL and Linux commands. Another limitation is how to unify the experiences of hierarchical expanding and other formula writing orders, because we think that the human reasoning order in writing formulas may be a mixture of top-down, bottom-up, left-to-right, etc. HERMES proposed new ideas on variable high-level expansion, but it is desirable to be further integrated with our experiences.

ACL 2023 Responsible NLP Checklist

A For every submission:

☐ A1. Did you describe the limitations of your work?
Left blank.

☐ A2. Did you discuss any potential risks of your work?
Left blank.

☐ A3. Do the abstract and introduction summarize the paper's main claims?
Left blank.

☐ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☐ Did you use or create scientific artifacts?

Left blank.

☐ B1. Did you cite the creators of artifacts you used?
Left blank.

☐ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Left blank.

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Left blank.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Left blank.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Left blank.

☐ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Left blank.

C ☐ Did you run computational experiments?

Left blank.

☐ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Left blank.

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☐ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Left blank.

☐ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Left blank.

☐ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Left blank.

D ☐ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
Left blank.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
Left blank.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
Left blank.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
Left blank.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
Left blank.