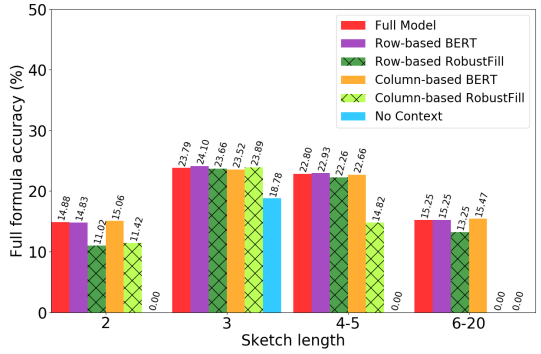

Figure 8. Top-1 formula accuracies for different sketch lengths, excluding headers in the context.

A. An Extended Discussion of Related Work

Various neural network approaches have been proposed for the FlashFill benchmark (Parisotto et al., 2017; Devlin et al., 2017; Vijayakumar et al., 2018). Specifically, both R3NN (Parisotto et al., 2017) and RobustFill (Devlin et al., 2017) are purely statistical models, and RobustFill performs better. In a RobustFill model, each formula is executed on a single data row, thus each row is independently fed into a shared encoder. Afterwards, at each decoding step, a shared LSTM decoder generates a hidden state per data row, which are then fed into a max pooling layer. Finally, the pooled hidden state is fed into a fully-connected layer to predict the formula token. On the other hand, in (Vijayakumar et al., 2018), they design a neural network to guide the deductive search performed by PROSE (Polozov & Gulwani, 2015), a commercial framework for input-output program synthesis. A recent work proposes neural-guided bottom-up search for program synthesis from input-output examples, and they extend the domain-specific language of FlashFill to support more spreadsheet programs (Odena et al., 2020).

Besides formula prediction, some previous work has studied other applications related to spreadsheets, including smell detection (Hermans et al., 2012a; Cheung et al., 2016; Singh et al., 2017; Azam et al., 2019), clone detection (Hermans et al., 2013; Dou et al., 2016; Zhang et al., 2020), and structure extraction for spreadsheet tables (Dong et al., 2019a;b). Our proposed encoder architecture could potentially be adapted for these spreadsheet tasks as well, and we leave it for future work.

B. More Experimental Results

For the setting where the model input does not include headers, corresponding to Table 3 in Section 4.3.2, we present the sketch and range accuracies in Table 4, and the breakdown accuracies on formulas of different sketch lengths in Figure 8. We observe that the performance degradation is more severe for formulas of sketch lengths 2–3.

Table 4. Breakdown accuracies on the test set, excluding headers in the context.

(a) Sketch accuracy.

	Approach	Top-1	Top-5	Top-10
	Full Model	28.33%	**62.55%**	72.89%
–	Column-based BERT	28.40%	61.60%	**74.92%**
–	Row-based BERT	27.71%	60.84%	73.43%
–	Pretraining	**28.78%**	62.37%	74.61%
	Row-based RobustFill	25.78%	42.66%	50.17%
	Column-based RobustFill	26.15%	47.78%	57.72%
	No context	25.19%	47.08%	52.70%

(b) Range accuracy.

	Approach	Top-1	Top-5	Top-10
	Full Model	22.60%	47.11%	53.84%
–	Column-based BERT	22.82%	**47.76%**	**54.98%**
–	Row-based BERT	22.47%	46.14%	54.51%
–	Pretraining	**23.48%**	47.27%	54.59%
	Row-based RobustFill	21.01%	38.21%	43.89%
	Column-based RobustFill	21.27%	37.80%	43.77%
	No context	11.80%	25.54%	38.07%

C. More Dataset Details

Although in principle, our model could generate formulas using any operator in the spreadsheet language, some kinds of value references are impossible to predict from local context, thus we remove formulas with such values from our dataset. Specifically, we exclude formulas that use the `HYPERLINK` function with a literal URL, since those are merely "stylistic" formulas that perform no computation beyond presenting a URL as a clickable link. As discussed in Section 2, we also filtered out formulas with cross-references from other tabs or spreadsheets. In total, the formulas filtered out after these two steps constitute around 40% of all formulas. We further filtered out formulas with cell references farther than 10 rows or columns from the target cell in either direction, and formulas with absolute cell ranges. In this way, about 45% of the original set of formulas are kept in our dataset.

Meanwhile, we observe that some spreadsheets may have tens of thousands of rows including the same formula, and including all of them in the dataset could bias our data distribution. Therefore, when multiple rows in the same spreadsheet table include the same formula in the same column, we keep the first 10 occurrences of such a formula, and create one data sample per formula. In this way, we extract around 846K formulas from 20M formulas before this filtering step, and we split them into 770K training samples, 42K for validation, and 34K for testing.

In total, around 100 operators are covered in our output

vocabulary. Among all spreadsheet formulas included in our filtered dataset, we list the 30 most commonly used spreadsheet functions and operators with their types [5] as follows: `SUM` (Math), + (Operator, equivalent to `ADD`), − (Operator, equivalent to `MINUS`), * (Operator, equivalent to `MULTIPLY`), / (Operator, equivalent to `DIV`), `&` (Operator, equivalent to `CONCAT`), `AVERAGE` (Statistical), `LEN` (Text), `UPLUS` (Operator), `STDEV` (Statistical), `COUNTA` (Statistical), `MAX` (Statistical), `LEFT` (Text), `IFERROR` (Logical), `ABS` (Math), `MEDIAN` (Statistical), `UMINUS` (Operator), `CONCATENATE` (Text), `ROUND` (Math), `WEEKNUM` (Date), `AVERAGEA` (Statistical), `MIN` (Statistical), `COUNT` (Statistical), `TRIM` (Text), `COS` (Math), `SIN` (Math), `SINH` (Math), `TODAY` (Date), `IF` (Logical), `MONTH` (Date). We observe that most of these functions and operators are for mathematical calculation, statistical computation, and text manipulation. However, people also write conditional statements, and spreadsheet formulas for calculating the dates.

The spreadsheet functions and operators utilized in the Enron corpus are: + (Operator, equivalent to `ADD`), `SUM` (Math), − (Operator, equivalent to `MINUS`), `UPLUS` (Operator), * (Operator, equivalent to `MULTIPLY`), / (Operator, equivalent to `DIV`), `AVERAGE` (Statistical), `MIN` (Statistical), `MAX` (Statistical), `UMINUS` (Operator), `COUNT` (Statistical), `COUNTA` (Statistical), `ABS` (Math), `LN` (Math), `DAY` (Date), `WEEKDAY` (Date), and `STDEV` (Statistical).

D. More Discussion of the FlashFill-like Setting

Following prior work on FlashFill (Devlin et al., 2017; Parisotto et al., 2017; Vijayakumar et al., 2018), we evaluate model performance when different numbers of data rows are presented to the model as input. Specifically, when the input includes 1–11 data rows, we grow the input from the target row upward. Our full data context includes 21 data rows, with 10 rows above the target cell, 10 rows below the target cell, and 1 row where the target cell locates. Consistent with prior work, when we vary the number of input data rows during inference, we always evaluate the same model trained with the full data context including 21 data rows. Since RobustFill independently encodes each row, it supports variable number of input rows by design. For our models with the tabular input representation, we set the rows to be empty when they are out of the input scope, and apply a mask to indicate that the corresponding data values are invalid.

[5] The function types are based on the Google Sheets function list here: https://support.google.com/docs/table/25273?hl=en.

E. Implementation Details

Data preprocessing. The content in each cell includes its data type and value, and we concatenate them as a token sequence. For example, `A2` in Figure 1a is represented as `num 0`. As discussed in Section 3.1, we concatenate all cell values in the same row as a token sequence, where values of different cells are separated by the `[SEP]` token. Each data row fed into the model includes $L = 128$ tokens, and when the concatenated token sequence exceeds the length limit, we discard cells that are further away from the target cell. For column-wise representation, we produce token embeddings independently for each column-wise bundle $S_{cb} = [H_c, C_{3b-1}, C_{3b}, C_{3b+1}]$ for $b \in [-3, 3]$, where C_i is a token sequence produced by concatenating all tokens of the cells in column C_i.

Output vocabulary construction. To construct the output formula token vocabulary, we filtered out tokens that appear less than 10 times in the training set, so that the vocabulary contains 462 tokens, out of 2625 tokens before filtering. In total, around a hundred operators are covered in our output vocabulary, including 82 spreadsheet-specific functions, and other general-purpose numerical operators (e.g., +, −).

Hyper-parameters. The formula decoder is a 1-layer LSTM with the hidden size of 512. We train the model with the Adam optimizer, with an initial learning rate of 5e-5. We train models for 200K minibatch updates, with a batch size 64. We set the dropout rate to be 0.1 for training. The norm for gradient clipping is 1.0.