Towards Generating Financial Reports From Table Data Using Transformers

Clayton Leroy Chapman, Lars Hillebrand, Marc Robin Stenzel, Tobias Deusser, Christian Bauckhage, Rafet Sifa

Fraunhofer IAIS,
Sankt Augustin, Germany
Email: Rafet.Sifa@iais.fraunhofer.de

Abstract—Financial reports are commonplace in the business world, but are long and tedious to produce. These reports mostly consist of tables with written sections describing these tables. Automating the process of creating these reports, even partially has the potential to save a company time and resources that could be spent on more creative tasks. Some software exists which uses conditional statements and sentence templates to generate the written sections. This solution lacks creativity and innovation when compared to recent advancements in NLP and deep learning. We instead implement a transformer network to solve the task of generating this text. By generating matching pairs between tables and sentences found in financial documents, we created a dataset for our transformer. We were able to achieve promising results, with the final model reaching a BLEU score of 63.3. Generated sentences are natural, grammatically correct and mostly faithful to the information found in the tables.

I. INTRODUCTION

A big part of running a successful business is record keeping. There are many different areas that must be overseen and documented, but arguably one of the most important involves finances. Finance is a broad term, but can be mostly summarized as anything having to do with the creation and management of capital or money. Even a small company can have hundreds or even thousands of records involving finances, so it makes sense that this information should be available in a presentable, easily understandable format. This is formally known as a financial report or financial statement. Such reports are necessary when discussing the state of a company, and allow investors and stakeholders to make more informed decisions. These lengthy documents contain financial data for the company from current and previous years, and usually include balance sheets, income or revenue, and cash flow.

To make sense of these numbers and tables, the reports also contain written sections. The written sections can contain detailed information about the company, decisions made by management, or relevant changes since the last report. Scattered within the written sections are also summaries and explanations of the various tables and spreadsheets. The cells referenced in the text have been highlighted for our convenience. While these texts are usually similar from year to year, from company to company, they still require a considerable amount of time to be addressed. As they say in business: “time is money”, and creating a financial report can be costly for a company. Automating this process, even partially, has the potential to reduce the amount of man hours spent on written

sections of these reports each quarter/year, thus saving the company money. There are, of course, a few different ways to go about automating this.

A very rudimentary and outdated method is to have generic templates for sentences. Certain words or values, such as currency amounts or dates, are intentionally left out. An algorithm then parses a table looking for these numbers, and inserts them in place of the missing words from the sentences. Multiple variations of the same sentence are necessary to account for differences in numbers. For example when describing profit or loss for a certain year, a sentence for each scenario has to be accounted for. This will be further referred to in this paper as the “parsing and pasting” method. In terms of grammar and correctness, this method does quite well, but this is to be expected since this method is essentially an intelligent form of copy and paste. The shortcoming of this strategy lies within the near complete lack of creativity and adaptability, since all sentences and expected outcomes must be written in and accounted for in advance. Also, not just anyone can write these template sentences. At least some amount of knowledge regarding finances is needed. Another shortcoming is that the amount of time needed to create this model is, in some cases, greater than the time needed to simply write the report. This strategy is only logistical for large-scale financial report text generation.

Going further to the idea of automation, neural networks can be used to generate text. A popular model for generating text is the Recurrent Neural Network (RNN). Unlike traditional networks that operate using feed forward operations, RNNs utilize a form of “memory” where some outputs of a layer can be fed back as input to a previous layer. RNNs can learn and predict sequences, and are useful for generating text since a sentence is essentially a sequence of words. In comparison to the “parse and paste” method, time is also needed to create and train the model, but with the advantage that the model can train on its own. Such a model can also be designed and trained by someone with little knowledge of finance, assuming examples for training already exist. While RNNs are not without their own shortcomings, they certainly are a step up from the previous strategy in terms of efficiency and saving time.

Recently, a new type of model has piqued the interest of many in the Natural Language Processing (NLP) community, known as the Transformer Network. This network is similar to RNNs in that it uses a sort of “memory” and can work

with sequential data. It excels at sequence-to-sequence and text generation tasks, among other things. Such a network could be used by transforming table data into an input sequence, and generating a sequence of text as its output, making it an ideal candidate for generating text in financial reports. This work will go on to describe a brief history and explanation of Transformer Networks, as well as to determine the legitimacy of this method as a way to generate text for financial documents.

II. BACKGROUND

A. Financial Reports

Financial reports or financial statements are formal records kept by either a business, person or other similar entity that serve to document the financial activities of the company. These records contain pertinent information for existing and potential investors and shareholders, and as such must be organized and easy to understand. For the most part, these documents typically have 4 sections which contain various information from a given time period:

- 1) The balance sheet, which details the company's assets, liabilities, and equity.
- 2) The income statement, which lists the company's income, expenses, and profits, as well as losses.
- 3) The statement of equity, which reports on the changes in equity: the value of shares issued by the company.
- 4) The cash flow statement, which reports on the cash flow of the company: the amount of money being transferred into or out of the company. This also includes operating, investing, and or financing activities of the company.

B. Neural Networks

The Neural Network (NN), also known as Multi-Layer Perceptron (MLP) and Deep Feed-forward Network, is the foundation of modern deep learning. The goal of these networks is function approximation, and they can be structured to solve a wide variety of problems. They are referred to as neural networks because of their loose resemblance to how neurons within the brain look and function. A NN is made up of several feed-forward layers of neurons. Each neuron takes input from input vectors or from the output(s) of one or many neurons from the previous layer. Except for the first layer, also known as the input layer, each input has an associated weight vector which influences the value of that input. Each neuron then sums its input values, and the output is determined using some nonlinear activation function. This continues until the final layer, or output layer, where the output values of each neuron make up the output vector. A NN needs to train with some dataset before it can be used to solve tasks. The training dataset contains input vectors as well as their corresponding expected output vectors. This type of training is known as supervised training. The network trains by adjusting its weight vectors/matrix, such that the output of the network is as close as possible to the expected output vector for all input vectors.

As previously mentioned, a NN is used to approximate functions, and a better way to understand them is by comparing

them to other methods of function approximation. If a problem is deterministic, meaning that for each input there exists a guaranteed, reproducible output, it can be approximated by a function. A very simple function approximation technique is to use linear models, such as logistic and linear regression. These models can fit data efficiently and reliably, but have the obvious limitations of only being useful for simple linear problems with a single independent variable. Using a linear model to approximate nonlinear functions requires more advanced techniques, such as kernels or transformations. Consider a transformation, $\phi(x)$, that would ideally map a nonlinear x into a new, linear input that the model could then work with. There are a few ways to choose how to map the function x , such as a generic mapping. This usually only works for generalizing local smoothness, and is inadequate for solving more advanced problems containing priori or for generalizing new, unseen data. Another way to create a mapping, similar to the "parsing and pasting" method of generating financial text, is to manually engineer it. This approach requires enormous amounts of time and will be extremely specific to the current task at hand, meaning that a mapping that works for one problem cannot be used or adapted for another problem. The final approach is what deep learning does while training.

$$y = \phi(x; \theta)^\top \omega \quad (1)$$

$$h^t = f(x^t; \theta) \quad (2)$$

The idea is to use the model 1, with parameters θ to learn ϕ from a broad class of functions, and parameters ω that map from $\phi(x)$ to the desired output [9]. This is an example of deep learning using NNs, where ϕ also represents the hidden layer(s), defined as eq. (2). This approach can benefit from both previous approaches by being generic, like selecting from a broad family of functions, and by benefiting from human knowledge, like finding the right general function family instead of the precise, exact function needed.

C. Recurrent Neural Networks

Certain tasks within the field of deep learning deal with specific sequences of data, where the output is highly dependent on the order and appearance of subsequences within the input data. Language processing and text generation is one of these tasks, where a sequence of words depends on what words have come before the current input. For a NN, it has no way of knowing such things because of its cyclical nature, but by making some modifications, a NN can be made into a Recurrent Neural Network (RNN).

$$h^t = f(h^{(t-1)}, x^t; \theta) \quad (3)$$

A RNN is similar to a NN in the sense that it has input, output, and hidden layers made up of neurons that pass information to each other, but in RNNs there exist feedback connections which allows them to be temporally dynamic. This means that in the recurrent layer(s) of the RNN, the output of a node is also an additional input to the node working on the next time step of data. The function from a NN to describe the

hidden state changes from eq. (2) to eq. (3) in the RNN [9]. Updating this hidden state representation also allows the RNN to essentially work with “variable” length inputs, whereas a NN is confined to fixed-length inputs. This simply means the network can better understand inputs that are, for example, padded with 0s if the input vector is shorter than the maximum sequence length that the RNN was trained on. The network is able to learn and recognize that these padded entries have little to no influence on the sequence. This is useful for analyzing sequential data that is not always available in a fixed length like audio, speech, and text. With this form of “short-term memory”, the network is able to represent the training data in more complex ways, which is also useful when working with text generation.

The extent of this memory is limited, however, and information cannot be stored across different input sequences. In terms of text generation, this means a RNN will eventually lose track of the current subject or context, and subsequent sentences may begin to repeat previously generated ideas or lose track of them entirely. With particularly long sentences, this can be a problem when the sentence will start and end with different trains of thought. Common sense would solve this problem by simply increasing the size of the network so that it can work with longer and longer sequences. Doing this introduces another problem which regularly affects RNNs: exploding or vanishing gradients. Because of the structure of RNNs, the error gradient calculated during back propagation is summed across each time step. This has the unfortunate side effect of the gradient either tending towards 0 or overflowing when there are a large number of time steps. This hinders training and the overall performance of the network [4]. While there do exist variations on these networks, such as Long Short-Term Memory (LSTM) [12] and Gated Recurrent Units (GRU) [5], that help to solve these shortcomings, RNNs remain complex and tedious networks to train. Since the output of each previous time step is included in the input for each subsequent time step, the training is sequential instead of parallel. Sequential training cannot take full advantage of a GPU, increasing the amount of time needed to train.

D. Transformer Networks

Currently, this task of generating text for financial documents requires a few things to be done successfully. The model should be able to read in data of variable length, often dealing with samples not present in the training data, and needs to be capable of handling long sequences while also having long-term memory. Above all, the model needs to be fast and efficient, which means it should ideally take advantage of parallelization. Surprisingly enough, a recently developed model known as the Transformer Network [20] tackles all of these obstacles. The transformer is a deep learning architecture that utilizes multi-headed attention mechanisms, allowing it to process sequential data in parallel and has theoretically infinitely long windows of memory. The transformer has had great success in the fields of NLP, NLG, and more recently even in CV. It is slowly being recognized as a successor to RNNs and LSTMs, with the speculation that they could even

one day succeed the CNN [7]. Since then, several successful transformer architectures have emerged and dominated the field of NLP. BERT from Google is one of the more popular architectures and has achieved state-of-the-art results on several NLP benchmarks [6]. T5, also from Google, has proven itself to be an incredibly capable general-purpose architecture. A single T5 model can be used in place of several different models, where each model is trained on a specific task. T5 sometimes performs almost as well as humans for certain tasks [18].

The basic architecture of a transformer consists of an encoder and a decoder. Each are made up of 6 encoder and 6 decoder layers, respectively, and together they follow the Seq2Seq approach. Seq2Seq models transform an input sequence into an output sequence, which works well for machine translation tasks. The job of the encoder is to map the input sequence into a higher dimensional space, while the decoder transforms this abstract vector into the output sequence. Traditionally, RNN layers were used to encode and decode this information. This means, however, that all features must be sequentially encoded, then decoded. Long sentences and training times remain a problem. To differentiate among the features and determine which are significant within a sequence, attention is used instead. Attention in deep learning is similar to cognitive attention in humans. For example, when a person looks at a photo featuring a subject in front of a background, their attention is first drawn to the subject and not the background. In deep learning, attention operates similarly and helps the model to focus on “more important” features within the input data while ignoring the “less important” ones. Instead of transforming the entire input sequence to a higher-dimensional, fixed-size vector, a list of context vectors from words within the input sequence are created. The model then computes and learns attention weights by determining which of these vectors are relevant for the output sequence, thereby learning to focus only on more important features of the sequences [3]. Attention also allows a network to process many features simultaneously. This benefit of parallelization translates to reductions in time needed to train a model.

The transformer uses a special kind of encoding called positional encoding. Recall that a transformer is not recurrent but can still work with sequential data. Removing recurrency is what allows the transformer to use stacked layers which can more or less work independently of one another. However, the relative position of data within a sequence must be retained, especially when working with natural languages. Word order can completely change the meaning of the text, for example “The children eat chicken” vs. “The chicken eats children”. In order to accomplish this without using recurrency, the transformer uses positional encodings. The basic idea is that when each token is encoded, an additional bit of information explaining its position is encoded with it. In practice this is not solved as easily as one might think. These encodings must satisfy a few different requirements to be useful. It goes without saying that each encoding has to be unique, since multiple words cannot both have the same position. One could use encodings within the range of $[0, 1]$, but then the number of tokens in a sequence is unclear. The consequences of this

are inconsistent distances between tokens for sequences of different lengths. One could instead use a linear encoding like $(1, 2, 3, \dots, n)$, except this hurts how generalizable the model can become. The model could either never see a sequence of a certain length during training, or a sequence longer than the longest training sequence, or both. It was explored that both learned and fixed encodings can be used to solve this problem [8]. The solution chosen for the original transformer paper was to use a fixed sinusoidal based function 4. The wavelengths form a progression from 2π to $10000 * 2\pi$. The $2i$ in the denominator guarantees a unique encoding for each position. The alternating sine and cosine for even and odd indices ensures the distance between encoded tokens remains consistent across sequences of varying length. This is mentioned in the paper, because for any fixed offset k , PE_{k+1} can be represented as a linear function of PE_k . The mathematical proof found in Appendix A explains why this holds [19].

$$PE = \begin{cases} \sin(\frac{pos}{w_k}) & \text{if } i = 2k \\ \cos(\frac{pos}{w_k}) & \text{if } i = 2k + 1 \end{cases} \quad (4)$$

where $w_k = 1000^{2i/d_{model}}$

Taking a deeper look inside of the encoder and decoder layers of a transformer, we see that a special type of attention mechanism, known as multi-headed self-attention, is employed. In an attention layer, the dependencies between two sequences, for example encoder RNN output and decoder RNN output, are compared with one another. In self-attention layers, dependencies within the same sequence are compared to one another, and as such these layers can be applied many times independently within a model. This gives transformers a certain edge in NLP as they are able to extract context and syntactic functions from sentences. Self-attention also receives the advantage over attention because a self-attention layer now acts as the encoder or decoder instead of RNNs in the previous example, allowing for longer sequences and a larger reference window of memory.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{D_k}})V \quad (5)$$

The goal of this self-attention module is to learn weight matrices, and thus feature dependencies among Query, Key, and Value vectors. These vectors are the output of the previous layer and are identical to one another. The Q and K vectors are dotted to create a score matrix. The values in this matrix will eventually show how the indices from the Q vector relate to indices of the K vector as the model trains. This matrix is scaled and normalized, an optional mask can be applied, and then a softmax is performed on the matrix. Because a dot-product operation is used in combination with softmax, scaling becomes necessary. When inputs of large magnitude are given to a softmax function, the gradients can become extremely small during back propagation. To counteract this in transformers' dot-product attention, the scale of $\frac{1}{\sqrt{D_k}}$ is used. The softmax function is applied to produce probability scores, and this matrix is multiplied with the V vector. Higher probability scores weigh more important features higher, which are then

focused on more by the model. This matrix is concatenated and then fed through a final linear layer. This attention can be split into multiple heads to better attend to different representations of subspaces in different positions. For multi-headed self-attention, the Q , K , and V vectors for each module are linear projections of the original Q , K , and V vectors. The same operations from before are done, but this time in parallel and with the output of each head being concatenated and fed into the linear layer. This output is then added with the original input via residual connections, and then normalized. These residual connections help the gradient flow through the network [11], eliminating some vanishing gradient problems as well as mitigating degradation. The similarities between the encoder and decoder layers end after these multi-headed self-attention and "Add and Norm" layers.

The encoder layer is somewhat straightforward, and is made up of 4 sublayers. The first two sublayers consist of the previously discussed multi-headed self-attention, and the add and norm layers. After this, the outputs are fed through a fully connected feed-forward network, and then another residual layer adds and normalizes the output. All of these normalization layers contribute to reducing the training time needed [2]. Encoder layers can also be stacked sequentially, where the output of an encoder layer is the input for the next stacked layer. This encodes the data further and each encoder layer can learn a different attention representation.

The decoder layer is larger and more complex than the encoder layer. The input of the decoder layer is nearly identical to the expected output. The difference is that a "start" token is added to the first index of the sequence, meaning that each index must be shifted right by 1. The reason for this right shift is so that during training the decoder learns to predict the i th input based on the previous $(1, \dots, i - 1)$ ground truth inputs. This is also known as teacher forcing[9]. The target output of the decoder is the unshifted expected output sequence, except with an "end" token in the final position. During training, this token is used to ensure that the lengths of these two sequences are the same. During test time, this tells the network to stop generating for the current sequence. The decoder layer also uses the sublayers of multi-headed self-attention with add and norm, however the attention in the decoder layer uses a mask for the probability score matrix. This is to prevent the attention among tokens from learning relationships with any future or subsequent tokens. This mask transforms all elements in the score matrix above the diagonal to negative infinity, which then becomes 0 after softmax is applied. This works alongside teacher forcing to mimic the sequential learning style of RNNs, reinforcing that predictions depend on previous outputs. The next 2 sublayers are more of the multi-headed self-attention and the add and norm sublayers, except with a small difference. The vectors Q , K , and V in the encoder sublayers all shared the same input vector, whereas for the decoder sublayers the vectors K and V come from the output of the last encoder layer. The vector Q is the output of the masked multi-headed self-attention, and add and norm sublayers. This allows the decoder to be able to attend to all positions from the input for every position [20]. This output then goes through another add and norm layer,

and then, similar to the encoder, through a fully connected feed-forward network, and add and norm layer again. Decoder layers can also be stacked, but only with the same number of stacked layers as the encoder. Once the decoder is finished, the output is given to a final linear layer with an output size equal to the vocabulary size. These logits from the linear layer are softmaxed, producing probability scores for a word at the current decoding step. As mentioned earlier, this continues until the “end of sequence” token is predicted.

E. BLEU Score

Typically, a NN will be used to solve a task that has one output for each input. This makes evaluating the performance of the network through the cost function simple to calculate since there is only one correct answer to learn for each sample. However, not all tasks in the deep learning space fit this description, many of which belong to NLP and NLG. A typical example of machine translation is translating a sentence from a source language to a target language. Although language has defined rules for how words and sentences can be created, they largely vary from person to person. For example, if you ask multiple people to translate “This is a dog.” into a different language, such as German, the translations will more or less all be the same. On the other hand, more complex sentences containing direct and indirect objects, verbs, conjugations, etc, can be translated into a few different sentences without losing any context or information from the original sentence. This becomes more complicated to evaluate because two or more translations can differ while still being “correct”. One proposed method to measure this correctness is the Bilingual Evaluation Understudy (BLEU) Score.

$$\text{Unigram precision } P = \frac{m}{w_t} \quad (6)$$

$$\text{Brevity Penalty } \rho = \begin{cases} 1 & \text{if } c > r \\ e^{1 - \frac{r}{c}} & \text{if } c \leq r \end{cases} \quad (7)$$

$$\text{BLEU} = \rho e^{(\sum_{n=1}^N (\frac{1}{N} * \log P_n))} \quad (8)$$

In order to calculate the BLEU score (8) for machine translation, one or more reference translations are needed. Assuming there is already a working model outputting some translation, the BLEU score can then be calculated. The basic idea is to count the number of matches between the n -grams of the generated sentence with the n -grams from the reference sentences. A precision is calculated using eq. (6), where m is the number of n -gram matches, and w is the total n -grams of the predicted sentence. Of course, one glaring issue with calculating precision this way is if we have a prediction sentence which contains a word that is also present in one or more reference sentences, repeated many times. For example, a predicted sentence like “The the the the the the the.” with the reference sentence “The cat is sleeping on the sofa”. This sentence would have a perfect precision of $\frac{7}{7}$ for 1-grams. To counter this, BLEU instead calculates precision using a clipped match, m_{max} , which limits the number of matches for a certain n -gram with the maximum occurrence of said n -gram

from the reference sentences. Now, our example sentence only has a precision of $\frac{2}{7}$, because our reference sentence only has two instances of “the”. The precision scores for each reference sentence are summed, and these summed n -gram precision scores for each n are averaged. This averaged precision is then multiplied by a brevity penalty (7), which penalizes predicted sentences shorter than the reference sentences. According to the original paper [15], using n -grams up to $N = 4$ yielded the “highest correlation with monolingual human judgements”. The inclusion of this range over n -grams from 1 to 4 captures what makes a translation “good”, namely, the adequacy and fluency of the translated text. 1-grams and 2-grams check the adequacy of one-to-one word translations, whereas 3-grams and 4-grams measure how fluent and grammatically correct the translation is.

III. RELATED AND PREVIOUS WORK

For many years now, text generation has been a popular use for RNNs [10]. Advancements in the model architecture led to GRUs [5] and LSTMs [12] which helped address the problems that RNN networks had. Recently, a new type of model has emerged that is starting to dominate the field of NLP, known as the transformer network [20]. Transformers were already being used in a few papers to generate text from table datasets. One popular dataset is called ToTTo [16]. The dataset consists of 120,000 samples of tabular data from Wikipedia articles with an accompanying sentence. Each sample has a list of “highlighted” cells from the table, from which the sentence is based on. Another popular dataset is called ROTOWIRE [21]. This dataset contains 4,853 samples of tabular data from basketball games with a written summary of the game. The papers that use the ToTTo and Rotowire datasets have shown that transformers are an appropriate solution to generating text from tables.

In terms of previous work regarding financial text generation, the master thesis of Alisa K. [13] has been a large motivation for this work. In her work, she attempted to solve the same problem of generating financial documents using tables. Her work began at simple Markov Chains and ended with using RNNs to generate sentences that one might find within a financial report. Unfortunately for her, the data available was not as task-specific as ToTTo or Rotowire. The data she used was taken directly out of financial reports. It was difficult to match blocks of text with their respective tables, as well as differentiating which sentences contained table-specific context. Her solution was to generate sentences from tables using rule-based generation and then train the network on these generated sentences. The RNN would then only be able to generate text given a starting sequence. Another interesting detail is that all numbers were replaced with <NUM> tokens. <UNK> tokens were also used, representing “unknown” words that appeared less than a certain N number of times, meaning that they were not present in the model’s vocabulary. Both of these tokens were a result of the memory constraints of RNNs. The inclusion of these tokens also meant that postprocessing was needed to replace these tokens with reasonable values.

Fig. 1. A visual representation of the working pipeline.

IV. IMPLEMENTATION

Originally, we wanted to continue with the work done by Alisa Khativa [13]. Her work involved generating text from financial documents using RNNs and Markov Chains. However, there were certain changes that we wanted to improve upon for this experiment. She had removed all numbers, percents, and dates in favor of special tokens to get around memory constraints of RNNs. The sentences were generated using an initial sentence produced manually, instead of using information found in a table or report. The sentences were then given to professional financial workers to compare with text produced by humans and scored.

Working with this idea and data as a starting point, our goal was to go further by making the process more intelligent and automated. With the aid of a relatively new type of network, the transformer network, we wanted to generate text directly from tables found within reports, such as budgets, profits and losses, and so on.

Using pre-trained transformers from HuggingFace [1] and datasets similar to this problem (ToTTo and ROTOWIRE), an initial pipeline would be coded that could receive table data and produce text similar to those written by a person. There are two possible strategies which could be useful for this experiment. The first strategy uses the ToTTo dataset. The input data would consist of “highlighted” cells, their associated column and row headers, the name of the table and similar metadata, and in some instances the name of the subtable where these cells were located. The output data is a caption or sentence that describes only the highlighted cells from the table. The second strategy uses the ROTOWIRE dataset,

except all cells are considered “highlighted”, meaning that the entire table is looked at. The output data for these tables is a rather lengthy paragraph describing the entire table. This can sometimes be problematic, as certain information found in long texts was not always in the table, so the network could hallucinate or struggle with producing sentences that properly describe the entire table. This strategy may work for a very specific context, such as basketball games in the original paper, but might become overwhelmed by the breadth of topics covered by financial reports. Testing both strategies, initial experiments showed that the pipeline is at least successful at transforming table data into some grammatically correct text. Looking through the table data available from financial documents, as well as the quality of the texts, the first strategy of using highlighted cells to produce a single sentence makes more sense.

Going further with this idea, the next step is to build an appropriate dataset. The documents at hand had already been parsed into a JSON format and contained Blobs of different types. The JSON data is parsed again and the Blobs without the type “text”, and “table” are removed. The intention would then be to take these Blobs and try and match each text with a table that contains the same or similar words, as well as numbers, percentages, and dates.

The basic matching algorithm is: Each text Blob is tokenized and iterated through. For each token, all of the tables are parsed and checked whether the token appears and if so, it is recorded as a match. A counter tracks the matches, and after all tokens are checked, the table with the highest number of matches is paired with the text Blob. Each match represents a highlighted cell. The table is fed to the transformer along with special tokens to separate rows and cells, and mark which cells are highlighted. The transformer being used is Bert2Bert. For each cell, one of the following tokens: </section_title>, </row>, </cell>, </row_header>, or </col_header> is appended to its tokenized value, depending on what type of cell it is. This is to give the transformer a better awareness of how the table was structured in the hope that it can better interpret it.

We had access to the financial reports which were used for Khativa’s work, so the algorithm is written according to their structure. Our work uses data from German financial reports from the entire dataset, as opposed to just the 200 banking reports used by Khativa. Text blobs are broken into sentences instead of being left as paragraphs to reduce complexity. The algorithm is adjusted to only check for matches on numbers, specifically currency values, instead of for every token in the sentence. This cuts down on noise and lowers the chance that a sentence will be matched with an incorrect table. Samples are also filtered based on the word count in sentences and the number of unique highlighted cells in tables.

Our final version of this matching algorithm takes some inspiration from the ToTTo paper. Their dataset collection process had 3 different strategies, and their final dataset was randomly sampled from the sets produced from these strategies. Their first strategy was matching tables and sentences with numbers that were at least 3 nonzero digits and not dates. Their second was to match tables and sentences that had more than 3 unique token matches in a single row of a table.

TABLE I
RESULTS AFTER TRAINING.

Approach	BLEU	Training time
Bert2Bert (subtables)	63.3	22.7 hours
Bert2Bert (full-tables)	29.6	49 hours
Seq2Seq RNN (512 nodes)	4.7	1.3 hours
Seq2Seq RNN (256 nodes)	8.7	1.5 hours

Their third was to match sentences and tables that contained respective hyperlinks to each other. We already have a similar strategy to their first one, namely matching currency tokens. Next is to employ a similar strategy to their second one, in order to eliminate as much noise as possible while also looking for the most context-rich pairs.

An upper bound of 30 words is used to preprocess the samples and remove outliers. Next, only unique matches in a row are counted when determining the best table. This is accomplished by keeping a list of matched tokens for each row and ignoring any matches already present in the list. For the highlighted cell postprocessing, in order for a table-sentence pair to become a sample, the number of unique highlights per row needs to be 2 or more. After preprocessing, the sample size is 181,010, and after postprocessing drops to 35,790.

Most tables in our dataset contain empty cells, which heavily contribute to noise in the input data because they are still being included as `</cell>` tokens. Other sources of noise present in the table come in the form of irrelevant cells. When describing the differences in budgets between 2 years, it makes very little sense to also give the transformer information about taxes paid or cash flow, for example. According to the ToTTo paper, they found greater success when limiting the size of the table for their input data. For this reason, only subtables made up of highlighted cells and their column and row headers are used instead of entire tables.

V. RESULTS

Fig. 2. BLEU scores of validation set during train time.

The experiments were carried out on a GPU cluster, where enough RAM was available to accommodate a batch size of

up to 16. Although our best model was trained on subtable samples, another model was trained on full tables for comparison. With 10,000 samples, one epoch would typically take 10 to 15 minutes to complete. Our final model was trained on 50 epochs, but it managed to reach 50 without triggering early stopping. Although the model could have theoretically trained for more epochs, the metric score showed very little improvement after 30 epochs. We determined it to be unlikely that a significantly higher score would have been achieved after 50 epochs. In a typical setting, a model could be gauged using a validation loss or F1 score. Benchmarking natural languages, however, requires a different metric. There were a few choices on which metric to use. Popular metrics for NLP include BLUE, NIST, and ROGUE, each of which has advantages and disadvantages. We decided to use BLEU as our metric, as the BLEU score is a standard metric for text generation and was also used in the ToTTo paper. Final scores are reported as sacrebleu scores in order to ensure consistency across different systems [17].

While reading through the text generated by the transformer using full tables instead of subtables, the text appears convincing. With a BLEU score of 29.6, the results are fluent and read like something you might find in a financial document. Most sentences contain a combination of dates, monetary values, and financial terms, similar to what one would find in the training data. However, a few patterns start to become noticeable and appear frequently among samples. Certain predicted sentences are identical to one another despite having different target sentences. There are also many numerical and monetary values that are formatted incorrectly with spaces between numbers, periods, and/or commas. The conclusion as to why the models up until this point performed so poorly was attributed to noise and overfitting. The input data consisting of the entire table for each sample was determined to be unnecessary and the cause of overfitting because the model would start to remember each table. It also later became apparent that for longer tables, information was being left out. This was due to the BERT2BERT's maximum input sequence length of 512 tokens. Any additional tokens after the 512th were simply dropped.

The results from only using subtables were somewhat of a surprise, namely, with the noticeable jump in the model's BLEU score to 63.3. It appeared that using subtables which only included highlighted cells and their headers was enough to produce high-quality sentences. By leaving out the unneeded information, the size of each sample shrunk, and the training time was also noticeably reduced. Sentences generated were of high quality, with correct grammar and sentence structure. There was the occasional hiccup where a word might be incorrectly assumed. An example being "TEUR" instead of "EUR", or a word article will be left out, such as "Ergebnis" instead of "Das Ergebnis".

It also appeared as if the model was learning the writing style for each company, thanks to the inclusion of the section title special token. Looking through some of the samples, target sentences between companies confirmed there were similarities. Word choice tended to be consistent, and certain sentences followed a similar structure. In some cases,

sentences had the same wording except for the year and monetary value. This explains how the model was able to produce diverse and detailed sentences while working with as few as 5 cells. This could also explain why our BLEU score of 63.3 is so high, especially when compared to ToTTo's 44.0. Our task is more focused, with a tighter vocabulary and similarly structured tables. Most tables did not actually vary much, with the majority having only 2 or 3 columns, with years as column headers. In comparison, ToTTo had to deal with a wide range of topics and worked with tables of many different shapes. This is to be expected, as the complexity of our task is comparatively lower than ToTTo.

VI. CHALLENGES AND FUTURE WORK

One of the biggest challenges we faced during this experiment was the quality of our data. We did not have access to the original documents and had to use the already parsed JSONs. Reading through the samples, one can see a mix of partial sentences, words being cut off, as well as numbers and units being incorrectly spaced. While it was possible to filter out or fix some of these issues, it was infeasible to control every single sample. Many samples contained a mix of English and German words, meaning that a spellchecker to remove samples with cut-off words might also remove legitimate samples. This mix of languages also negatively affected our tokenizer. Some words were incorrectly split into multiple tokens, and many of our number values were separated due to commas and decimal points. Fortunately, these tokens still belonged to the same cell and were wrapped by special tokens. Despite this, nearly every date, percent, and number from the generated sentences contained awkward spaces. This can of course easily be remedied by more postprocessing, but we still wonder if this spacing problem has any effect on training.

Inconsistencies within the tables also proved to be quite the challenge for us, and led to some difficulties in our parsing algorithm. There are several different formats used for dates and money within the tables. Some examples of date formats which are found in the tables:

- "30.06.2011" (DD.MM.YYYY)
- "04.30.2014" (MM.DD.YYYY)
- "01-092010" (DD-MMYYYY)
- "2015" (YYYY)
- "01.01.-31.12.2014" or "2011/2012" (Span dates)
- "Q1 2015", "Q1-15", or "1. Quartal 2015" (Quarter w/ year)

After trying to manually catch each edge case, we decided to just use a simple regular expression, which assumed that any occurrence of a 4 digit number with the pattern 18XX, 19XX, or 20XX was a date. It would be interesting to see if our results could be improved with a more elegant strategy.

We faced a similar problem with currency values. Sometimes the number would have the currency unit inside its cell, other times it would be somewhere else in the table. There were also different ways to express the currency units, such as:

- €
- Euro

- eur
- EUR
- TEUR or T EUR
- Mio. Eur

Luckily, these edge cases were less of an issue to tackle, and the model seemed to do a fine job inferring the difference between "100 Eur" and "100 Mio Eur".

Another observation was model hallucination, which was also mentioned in the ToTTo paper. This problem has also been documented in other table-to-text experiments [14], [22]. Similar with the ToTTo dataset, the tables were not unique to the samples, meaning 2 samples could reference the same table. This meant overlapping tables were a possibility. The chance of this happening further increased since we filtered all non-highlighted cells out. We suspect this to be the case, as multiple samples hallucinated words, or context which were not present in the tables' tokens. To try and avoid this from happening, the ToTTo team ran different experiments with and without overlapping tables. Their results showed a nearly 20 point increase in BLEU score when comparing the results from the overlapping table dataset with the non-overlapping table dataset. We assume this would also happen for our experiment if we had taken similar steps to filter out overlapping tables. This observation causes a dilemma. On the one hand, our model generates reasonable results similar to their reference sentences. On the other hand, the results generated are less faithful to their tables. For this reason, perhaps another metric might be more appropriate for this task.

Future work in this area would be greatly benefited from a more consistent dataset. Given more time and resources, target sentences written specifically for samples in this experiment could also be helpful. This would eliminate any shortcomings with the parsing and matching algorithms or possible mismatches between sentences and tables. Most of the samples chosen for the final model only relied on the assumption that "good" samples had multiple unique matches in the same row. It is well within the realm of possibility that many samples which could have been useful were ignored. This also limited much diversity among the samples, because many sentences referencing different rows may not have met the algorithm's criteria for being "good". Having people manually highlight cells and write something insightful for those matches could improve on the robustness of the model. The data could be further improved by having multiple reference sentences for each sample. Additionally, it is common for text generation and translation experiments to have humans rate the quality of generated text alongside automated metrics. This was unfortunately not an option available to us, and BLEU had to instead be solely relied upon. Given more time, it would have been interesting to have our generated sentences judged by humans. Predicted sentences could be compared directly with reference sentences, and scored based on whether they describe the table's highlighted cells worse, the same, or better than the reference sentence.

VII. CONCLUSION

Financial reports are a necessary part of running a business. As we have seen, a lot of time and effort goes into writing

these reports despite their content being somewhat standard. Giving a company the option to automate parts of this process would free up employees' time to work on other essential tasks. There already exists some software which can semi-automate this process but which relies on if-else statements and basic logic. This software must account for all edge cases and outcomes, but the text produced in the end still adheres to the template. There exists a possibility for a more intelligent form of automation.

Previous work using RNNs to try and solve this task improved the variety of sentences generated. This method demonstrated that these deep learning networks could fill this potential niche, but that they also had several shortcomings. RNNs are tedious to train and require a large amount of data, time, and resources. RNNs are plagued by exploding and vanishing gradients. This puts an effective upper bound on the length of the sequences they can process. Being sequential also means that RNNs have limited memory. If a feature must travel through several different cells to where it is needed, the chance that this feature is corrupted or forgotten increases.

Recent research in attention-based models and the emergence of transformer networks has given new hope to many NLP and NLG tasks. The transformer network addresses most of the shortcomings of RNNs. Because of multi-headed self-attention, the transformer can better utilize the GPU. This results in faster training time and more efficient use of resources to train large models. Transformer networks are not affected by exploding or vanishing gradients to the same degree as RNNs. As such, they can work with longer sequences. Through their use of attention, transformer networks have a much larger window of memory when compared to RNNs. Many pre-trained transformer networks are available to download and can be immediately trained on downstream tasks. For these reasons we decided to experiment with using transformers to generate text from tables found in financial reports.

Using the ToTTo paper as a guide, we used the BERT2BERT transformer network. This network uses special tokens to separate sentences within an input sequence, and implementing custom special tokens based on a table was trivial. Our special tokens were: `</section_title>`, `</row>`, `</cell>`, `</row_header>`, and `</col_header>`.

Financial reports already parsed as JSON were used for the experiment. Within a JSON, Blobs containing tables and text were collected, and parsing and matching algorithms were used to generate table-text pairs. Each table-text pair kept track of matches between cells and words, and these cells were designated as highlighted. Tables were then broken down into highlighted cells with their respective section titles, and column and row headers. These subtables were tokenized, encoded, and given to the transformer to train. Our final model achieved a BLEU score of 63.3.

Despite our many obstacles along the way, the final model demonstrates itself as capable. Generated sentences are mostly grammatically correct and fluent, with word choices appropriate for a financial setting. These sentences also reach reasonable conclusions based on their encoded tables. The model sometimes struggled with hallucinations, an already documented phenomenon from previous table-to-text studies.

Despite all this, such a model could be used in place of a person to at least partially write financial documents, which in the end still saves time.

REFERENCES

- [1] Huggingface: The ai community building the future. <https://huggingface.co/>.
- [2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.
- [3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.
- [4] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.
- [5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.
- [6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.
- [7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.
- [8] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning, 2017.
- [9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [10] Alex Graves. Generating sequences with recurrent neural networks, 2014.
- [11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.
- [12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- [13] Alisa Khativa. Methods of financial reports generation, 2019.
- [14] Remi Lebrete, David Grangier, and Michael Auli. Neural text generation from structured data with application to the biography domain, 2016.
- [15] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.
- [16] Ankur P. Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. Totto: A controlled table-to-text generation dataset, 2020.
- [17] Matt Post. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium, October 2018. Association for Computational Linguistics.
- [18] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2020.
- [19] Vinit Ravishankar and Anders Søgaard. The impact of positional encodings on multilingual compression, 2021.
- [20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.
- [21] Hongmin Wang. Revisiting challenges in data-to-text generation with fact grounding. *Proceedings of the 12th International Conference on Natural Language Generation*, 2019.
- [22] Sam Wiseman, Stuart M. Shieber, and Alexander M. Rush. Challenges in data-to-document generation, 2017.

APPENDIX

TABLE II

PARAMETERS FOR FINAL BERT2BERT TRANSFORMER MODEL. ANY ADDITIONAL PARAMETERS WHICH AREN'T LISTED CAN BE ASSUMED TO BE THEIR DEFAULT VALUES FROM HUGGINGFACE.

Encoder	bert-base-german-dbmdz-cased	Decoder	bert-base-german-dbmdz-cased
Tokenizer	bert-base-german-dbmdz-cased	Vocabulary size	30,000
Max sequence length	512 (tokens)	Batch size	8
Output max length	30 (tokens)	Output min length	5 (tokens)
Beams	1	Top-k	50
Repetition penalty	0.8	Length penalty	1
Optimizer	AdamW	Learning rate	1e-5
Weight decay	0.01	Correct bias	false
Learning rate scheduler	lin_warmup	Learning rate warmup	0.1
BLEU n-gram weights	[0.25, 0.25, 0.25, 0.25]	Validation metric	+sacrebleu

TABLE III: Results across various BLEU scores, along with target sentence and encoded table. One can see the hallucination referenced in the results section.

Encoded table	[CLS] RhoenKlinikum-AnnualReport-2015.json </section_title> Zugänge aus Änderung Konsolidierungskreis </row_header> 0,2 </cell> </row> Umbuchungen </row_header> 0,2 </cell> 2,0 </cell> </row> [SEP]
Predicted sentence	2. 2 Mio. EUR (Vj. 2, 2 Mio. EUR)) hat die ML V. 0, 0 Mio. EUR (Vj. 0, 0 Mio. EUR
Target sentence	Auf nicht beherrschende Anteile entfallende Gewinnanteile sind im Vergleich zur Vorjahresperiode transaktionsbedingt um 0, 2 Mio. EUR auf 2, 0 Mio. EUR zurückgegangen.
BLEU score	10
Encoded table	[CLS] Commerzbank-QuarterlyReport-2010-Q3.json </section_title> und Sonstige </row_header> 20 </cell> </row> Sonstige </row_header> 9 </cell> 20 </cell> </row> [SEP]
Predicted sentence	Die Risikovorsorge im Privatkundengeschäft lag im dritten Quartal 2010 bei 9 Mio Euro, im Vergleich zu 20 Mio Euro im Vorjahr.
Target sentence	Hauptbestandteile des Exposures sind die Teilportfolios Büro (27 Mrd Euro), Handel (20 Mrd Euro) und Wohnimmobilien (9 Mrd Euro).
BLEU score	20
Encoded table	[CLS] FuchsPetrolub-AnnualReport-2013.json </section_title> 2013 </col_header> </row> Europa </row_header> 1.104,2 </cell> 23,5 </cell> </row> [SEP]
Predicted sentence	Europa ist im Jahr 2013 um 23, 5 % bzw. 23, 5 Mio EUR organisch gewachsen und hat sich organisch um 1. 52, 2 Mio EUR bzw.
Target sentence	Die europäischen Gesellschaften erwirtschafteten im Berichtsjahr Umsätze von 1. 104, 2 Mio EUR (1. 080, 7), das sind 2, 2 % oder 23, 5 Mio EUR mehr als im Vorjahr.
BLEU score	30
Encoded table	[CLS] Zalando-AnnualReport-2012.json </section_title> 01.01.-31.12.2012 </col_header> 01.01.-31.12.2011 </col_header> </row> Wertberichtigungen auf Forderungen </row_header> 24.145 </cell> 7.166 </row> [SEP]
Predicted sentence	Die Forderungen aus Lieferungen und Leistungen betragen 24. 145 T EUR (Vorjahr : 7. 166 T EUR).
Target sentence	Im Berichtsjahr 2012 wurden Wertberichtigungen auf Forderungen in Höhe von T EUR 24. 145 (Vorjahr : T EUR 7. 166) aufwandswirksam erfasst.
BLEU score	40
Encoded table	[CLS] ADVA_Optical-AnnualReport-2012.json </section_title> 2012 </col_header> 2011 </col_header> </row> Nettoliquidität </row_header> 41,6 </cell> 31,2 [SEP]
Predicted sentence	Zum Jahresende 2012 erreichte die Nettoliquidität mit EUR 41, 6 Millionen einen historischen Quartalsende - Höchststand und lag damit um EUR 7, 7 Millionen über den zu Ende 2011 verzeichneten EUR 31,
Target sentence	Zum Jahresende 2012 belief sich die Nettoliquidität auf ein neues Allzeithoch von EUR 41, 6 Millionen nach EUR 31, 2 Millionen zu Ende des Vorjahres.
BLEU score	50
Encoded table	[CLS] Wacker_Neuson-AnnualReport-2016.json </section_title> 2016 </col_header> 2015 </col_header> </row> Cashflow aus operativer Geschäftstätigkeit </row_header> 131,1 </cell> 131,0 </cell> </row> [SEP]
Predicted sentence	Der Cashflow aus operativer Geschäftstätigkeit stieg im Geschäftsjahr 2016 auf 131, 1 Mio. Euro (Vorjahr : 131, 0 Mio. Euro).
Target sentence	Der operative Cashflow lag zum Jahresende mit 131, 1 Mio. Euro auf Vorjahres niveau (2015 : 131, 0 Mio. Euro).
BLEU score	60

Encoded table	[CLS] MTU_Aero_Engines-QuarterlyReport-2014-Q3.json </section_title> 30.9.2014 </col_header> 31.12.2013 </col_header> </row> Summe Eigenkapital und Schulden </row_header> 4.493,1 </cell> 4.410,5 </cell> [SEP]
Predicted sentence	Die Konzernbilanzsumme ist zum 30. September 2014 mit 4. 493, 1 Mio. EUR gegenüber dem 31. Dezember 2013 mit 4. 410, 5 Mio. EUR nahezu unverändert geblieben.
Target sentence	Die Konzernbilanzsumme ist um 82, 6 Mio. EUR gestiegen - von 4. 410, 5 Mio. EUR zum 31. Dezember 2013 auf 4. 493, 1 Mio. EUR zum 30. September 2014.
BLEU score	70
Encoded table	[CLS] Deutsche_Post-QuarterlyReport-2016-Q1.json </section_title> Q1 2015 </col_header> Q1 2016 </col_header> </row> MEA (MiddleEastandAfrica) </row_header> 254 </cell> 261 </cell> </row> [SEP]
Predicted sentence	In der Region MEA (Middle East and Africa) lag der Umsatz im Berichtszeitraum mit 261 MIO EUR auf dem Niveau des Vorjahres (254 MIO EUR).
Target sentence	In der Region MEA (Middle East and Africa) stieg der Umsatz im Berichtszeitraum um 2, 8 % auf 261 MIO EUR (Vorjahr : 254 MIO EUR).
BLEU score	80
Encoded table	[CLS] STADA-AnnualReport-2011.json </section_title> 2011 </col_header> </row> Italien </row_header> 146,1 </cell> 136,8 </cell> </row> [SEP]
Predicted sentence	In Italien verzeichnete der Konzern im Geschäftsjahr 2011 eine Umsatzerhöhung von 17 % auf 146, 1 Mio. EUR (Vorjahr : 136, 8 Mio. EUR).
Target sentence	In Italien verzeichnete der Konzern im Geschäftsjahr 2011 einen Umsatzanstieg von 7 % auf 146, 1 Mio. EUR (Vorjahr : 136, 8 Mio. EUR).
BLEU score	90
Encoded table	[CLS] Jungheinrich-AnnualReport-2014.json </section_title> 31.12.2014 </col_header> 31.12.2013 </col_header> </row> Verbindlichkeiten Finanzierung Mietgeräte </row_header> FLAC </row_header> 67.625 </cell> 53.820 </cell> </row> [SEP]
Predicted sentence	Die Verbindlichkeiten aus der Finanzierung von Mietgeräten resultierten in Höhe von 67. 625 T EUR (Vorjahr : 53. 820 T EUR) aus dem Verkauf von Forderungen aus konzerninternen Mietkaufverträgen.
Target sentence	Die Verbindlichkeiten aus der Finanzierung von Mietgeräten resultierten in Höhe von 67. 625 T EUR (Vorjahr : 53. 820 T EUR) aus dem Verkauf von Forderungen aus konzerninternen Mietkaufverträgen.
BLEU score	100