MATE: Multi-view Attention for Table Transformer Efficiency

Julian Martin Eisenschlos[1], Maharshi Gor[2][*], Thomas Müller[3][*], William W. Cohen[1]

Google Research[1]

{eisenjulian,wcohen}@google.com

Dept. of Computer Science, University of Maryland[2]

mgor@cs.umd.edu

Symanto Research, Valencia, Spain[3]

thomas.mueller@symanto.com

Abstract

This work presents a sparse-attention Transformer architecture for modeling documents that contain large tables. Tables are ubiquitous on the web, and are rich in information. However, more than 20% of relational tables on the web have 20 or more rows (Cafarella et al., 2008), and these large tables present a challenge for current Transformer models, which are typically limited to 512 tokens. Here we propose MATE, a novel Transformer architecture designed to model the structure of web tables. MATE uses sparse attention in a way that allows heads to efficiently attend to either rows or columns in a table. This architecture scales linearly with respect to speed and memory, and can handle documents containing more than 8000 tokens with current accelerators. MATE also has a more appropriate inductive bias for tabular data, and sets a new state-of-the-art for three table reasoning datasets. For HYBRIDQA (Chen et al., 2020b), a dataset that involves large documents containing tables, we improve the best prior result by 19 points.

1 Introduction

The Transformer architecture (Vaswani et al., 2017) is expensive to train and run at scale, especially for long sequences, due to the quadratic asymptotic complexity of self-attention. Although some work addresses this limitation (Ainslie et al., 2020; Kitaev et al., 2020; Zaheer et al., 2020), there has been little prior work on scalable Transformer architectures for *semi-structured* text.[1] However, although some of the more widely used benchmark tasks involving semi-structured data have been restricted to moderate size tables, many semi-structured documents are large: more than 20% of relational tables on the web have 20 or more rows (Cafarella et al.,

---

[*]Work done at Google Research.

[1]The term "semi-structured text" refers to text that has structure that does not reflect a known data schema. Typically semi-structured text is organized as an HTML tree or variable length lists and tables.
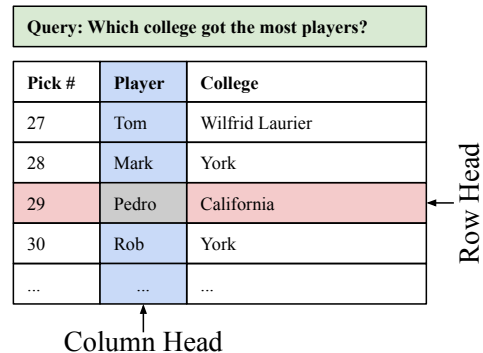

Figure 1: Sparse self-attention heads on tables in MATE are of two classes: *Row heads* attend to tokens inside cells in the same row, as well as the query. *Column heads* attend to tokens in the same column and in the query. Query tokens attend to all other tokens.

2008), and would pose a problem for typical Transformer models.

Here we study how efficient implementations for transformers can be tailored to semi-structured data. Figure 1 highlights our main motivation through an example: to obtain a contextual representation of a cell in a table, it is unlikely that the information in a completely different row and column is needed.

We propose the MATE architecture[2] (Section 3), which allows each attention head to reorder the input so as to traverse the data by multiple points of view, namely column or row-wise (Figure 2). This allows each head to have its own data-dependent notion of locality, which enables the use of sparse attention in an efficient and context-aware way.

This work focuses on question answering (QA) and entailment tasks on tables. While we apply our model to several such tasks (see section 6), HYBRIDQA (Chen et al., 2020b) is particularly interesting, as it requires processing tables jointly with long passages associated with entities mentioned in the table, yielding large documents that may not fit in standard Transformer models.

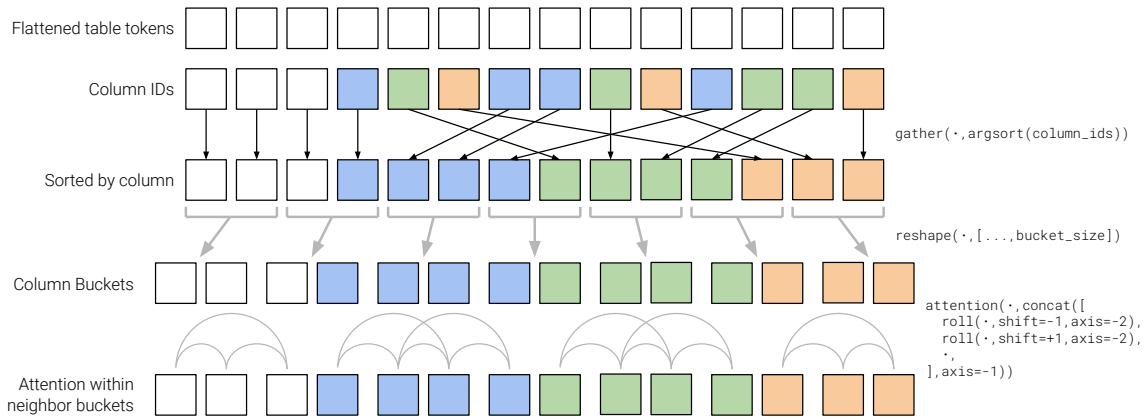[2]Pronounced *mah-teh*, as in mate tea.

Figure 2: Efficient implementation for MATE. Each attention head reorders the tokens by either column or row index and then applies a windowed attention mechanism. This figure omits the global section that attends to and from all other tokens. Since column/row order can be pre-computed, the method is linear for a constant block size.

Overall, our contributions are the following:

i) We show that table transformers naturally focus attention according to rows and columns, and that constraining attention to enforce this improves accuracy on three table reasoning tasks, yielding new state-of-the-art results in SQA and TABFACT.

ii) We introduce MATE, a novel transformer architecture that exploits table structure to allow running training and inference in longer sequences. Unlike traditional self-attention, MATE scales linearly in the sequence length.

iii) We propose POINTR (Section 4), a novel two-phase framework that exploits MATE to tackle large-scale QA tasks, like HYBRIDQA, that require multi-hop reasoning over tabular and textual data. We improve the state-of-the-art by 19 points.

All the code is available as open source.[3]

2 Related Work

Transformers for tabular data Traditionally, tasks involving tables were tackled by searching for logical forms in a semantic parsing setting. More recently Transformers (Vaswani et al., 2017) have been used to train end-to-end models on tabular data as well (Chen et al., 2020a). For example, TAPAS (Herzig et al., 2020) relies on Transformer-based masked language model pre-training and special row and column embeddings to encode the table structure. Chen et al. (2021) use a variant of ETC (Ainslie et al., 2020) on an open-domain version of HYBRIDQA to read and choose an answer span from multiple candidate passages and cells, but the proposed model does not jointly process the full table with passages.

In order to overcome the limitations on sequence length Eisenschlos et al. (2020) propose heuristic column selection techniques, and they also propose pre-training on synthetic data. Krichene et al. (2021) propose a model based cell selection technique that is differentiable and trained end-to-end together with the main task model. Our approach is orthogonal to these methods, and can be usefully combined with them, as shown in Table 4.

Recently, Zhang et al. (2020) proposed SAT, which uses an attention mask to restrict attention to tokens in the same row and same column. SAT also computes an additional histogram row appended at the bottom of the table and encodes the table content as text only (unlike TAPAS). The proposed method is not head dependent as ours is, which prevents it from being implemented efficiently to allow scaling to larger sequence lengths. Controlling for model size and pre-training for a fair comparison, we show that our model is both faster (Table 4) and more accurate (Table 6) than SAT.

Efficient Transformers There is prior work that tries to improve the asymptotic complexity of the self-attention mechanism in transformers. Tay et al. (2020) review the different methods and cluster them based on the nature of the approach. We cover some of the techniques below and show a theoretical complexity comparison in Table 1.

The LINFORMER model Wang et al. (2020) uses learned projections to reduce the sequence length axis of the keys and value vectors to a fixed length. The projections are then anchored to a specific input length which makes adapting the sequence

---

[3] github.com/google-research/tapas

Model	Complexity	Class
Transformer-XL	$\mathcal{O}(n^2)$	RC
REFORMER	$\mathcal{O}(n \log n)$	LP
LINFORMER	$\mathcal{O}(n)$	LR
BIGBIRD	$\mathcal{O}(ng + nb)$	FP+RP+M
ETC	$\mathcal{O}(ng + nb)$	FP+M
MATE (Ours)	$\mathcal{O}(ng + nb)$	FP

Table 1: Comparison of transformer models following Tay et al. (2020). Class abbreviations include: FP = Fixed Patterns, RP = Random Patterns, M = Memory, LP = Learnable Pattern, LR = Low Rank and RC = Recurrence. The block size for local attention is denoted as b and g the size of the global memory. For our MATE model, a $n \log n$ sorting step can be pre-computed before training or inference for known tables so it is omitted.

length during pre-training and fine-tuning challenging, and makes the model more sensitive to position offsets in sequences of input tokens.

REFORMER (Kitaev et al., 2020) uses locality sensitive hashing to reorder the input tokens at every layer in such a way that similar contextual embeddings have a higher chance of being clustered together. We instead rely on the input data structure to define ways to cluster the tokens. Although the limitation can be circumvented by adapting the proposed architecture, REFORMER was originally defined for auto-regressive training.

Ainslie et al. (2020) introduce ETC, a framework for global memory and local sparse attention, and use the mechanism of relative positional attention (Dai et al., 2019) to encode hierarchy. ETC was applied to large document tasks such as Natural Questions (Kwiatkowski et al., 2019). The method does not allow different dynamic or static data re-ordering. In practice, we have observed that the use of relative positional attention introduces a large overhead during training. BIGBIRD (Zaheer et al., 2020) presents a similar approach with the addition of attention to random tokens.

3 The MATE model

Following TAPAS (Herzig et al., 2020), the transformer input in MATE, for each table-QA example, is the query and the table, tokenized and flattened, separated by a [SEP] token, and prefixed by a [CLS]. Generally the table comprises most of the the input. We use the same row, column and rank embeddings as TAPAS.

To restrict attention between the tokens in the table, we propose having some attention heads limited to attending between tokens in the same row (plus the non-table tokens), and likewise for columns. We call these *row heads* and *column heads* respectively. In both cases, we allow attention to and from all the non-table tokens.

Formally, if $\mathbf{X} \in \mathbb{R}^{d \times n}$ is the input tensor for a Transformer layer with sequence length n, the k-th position of the output of the i-th attention head is:

$$\text{Head}_k^i(\mathbf{X}) = \mathbf{W}_V^i \mathbf{X}_{\mathcal{A}_k^i} \sigma \left[\left(\mathbf{W}_K^i \mathbf{X}_{\mathcal{A}_k^i} \right)^{\top} \mathbf{W}_Q^i \mathbf{X}_k \right]$$

where $\mathbf{W}_Q^i, \mathbf{W}_K^i, \mathbf{W}_V^i \in \mathbb{R}^{m \times d}$ are query, key and value projections respectively, σ is the softmax operator, and $\mathcal{A}_k^i \subseteq \{1, \cdots, n\}$ represents the set of tokens that position k can attend to, also known as the *attention pattern*. Here $\mathbf{X}_{\mathcal{A}_k^i}$ denotes gathering from $\mathbf{X}$ only the indexes in $\mathcal{A}_k^i$. When $\mathcal{A}_k^i$ contains all positions (except padding) for all heads i and token index k then we are in the standard dense transformer case. For a token position k, we define $r_k, c_k \in \mathbb{N}_0$ the row and column number, which is set to 0 if k belongs to the query set Q: the set of token positions in the query text including [CLS] and [SEP] tokens.

In MATE, we use two types of attention patterns. The first $h_r \geq 0$ heads are *row heads* and the remaining h_c are *column heads*:

$$\mathcal{A}_k^i = \begin{cases} \{1, \cdots, n\} & \text{if } k \in Q, \text{ else} \\ Q \cup \{j : r_j = r_k\} & \text{if } 1 \leq i \leq h_r \\ Q \cup \{j : c_j = c_k\} & \text{otherwise.} \end{cases}$$

One possible implementation of this is an attention mask that selectively sets elements in the attention matrix to zero. (Similar masks are used for padding tokens, or auto-regressive text generation.) The ratio of row and column heads is a hyperparameter but empirically we found a $1 : 1$ ratio to work well. In Section 6 we show that attention masking improves accuracy on four table-related tasks. We attribute these improvements to better inductive bias, and support this in Section 7 showing that full attention models learn to approximate this behavior.

3.1 Efficient implementation

Although row- and column-related attention masking improves accuracy, it does not improve Transformer efficiency—despite the restricted attention, the Transformer still uses quadratic memory and time. We thus also present an approximation of row and column heads that can be implemented

more efficiently. Inspired by Ainslie et al. (2020), the idea is to divide the input into a global part of length G that attends to and from everything, and a local (typically longer) part that attends only to the global section and some radius R around each token in the sequence. ETC does this based on a fixed token order. However, the key insight used in MATE is that the notion of locality can be configured *differently for each head*: one does not need to choose a specific traversal order for tokens ahead of time, but instead tokens can be ordered in a data-dependent (but deterministic) way. In particular, row heads can order the input according to a row order traversal of the table, and column heads can use a column order traversal. The architecture is shown in Figure 2.

After each head has ordered its input we split off the first G tokens and group the rest in evenly sized buckets of length R. By reshaping the input matrix in the self-attention layer to have R as the last dimension, one can compute attention scores from each bucket to itself, or similarly from each bucket to an adjacent one. Attention is further restricted with a mask to ensure row heads and column heads don't attend across rows and columns respectively. See model implementation details in Appendix D. When G is large enough to contain the question part of the input and R is large enough to fit an entire column or row, then the efficient implementation matches the mask-based one.

As observed in Ainslie et al. (2020), asymptotic complexity improvements often do not materialize for small sequence lengths, given the overhead of tensor reshaping and reordering. The exact break-even point will depend on several factors, including accelerator type and size as well as batch size. In the experiments below the best of the two functionally equivalent implementations of MATE is chosen for each use case.

3.2 Compatibility with BERT weights

The sparse attention mechanism of MATE adds no additional parameters. As a consequence, a MATE checkpoint is compatible with any BERT or TAPAS pre-trained checkpoint. Following Herzig et al. (2020) we obtained best results running the same masked language model pre-training used in TAPAS with the same data but using the sparse attention mask of MATE.

For sequence lengths longer than 512 tokens, we reset the index of the positional embeddings at

Split	Train	Dev	Test	Total
In-Passage	35,215	2,025	20,45	39,285
In-Table	26,803	1,349	1,346	29,498
Missing	664	92	72	828
Total	62,682	3,466	3,463	69,611

Table 2: Statistics for HYBRIDQA. *In-Table* and *In-Passage* groups mark the location of the answer. Missing denotes answers that do not match any span and may require complex computations.

the beginning of each cell. This method removes the need to learn positional embeddings for larger indexes as the maximum sequence length grows while avoiding the large computational cost of relative positional embeddings.

3.3 Universal approximators

Yun et al. (2020a) showed that Transformers are universal approximators for any continuous sequence-to-sequence function, given sufficient layers. This result was further extended by Yun et al. (2020b); Zaheer et al. (2020) to some Sparse Transformers under reasonable assumptions.

However, prior work limits itself to the case of a single attention pattern per layer, whereas MATE uses different attention patterns depending on the head. We will show that MATE is also a universal approximator for sequence to sequence functions.

Formally, let $\mathcal{F}$ be the class of continuous functions $f : \mathbb{D} \subset \mathbb{R}^{d \times n} \to \mathbb{R}^{d \times n}$ with $\mathbb{D}$ compact, with the p-norm $\| \cdot \|_p$. Let $\mathcal{T}_{\text{MATE}}$ be any family of transformer models with a fixed set of hyper-parameters (number of heads, hidden dimension, etc.) but with an arbitrary number of layers. Then we have the following result.

Theorem 1. *If the number of heads is at least* 3 *and the hidden size of the feed forward layer is at least* 4*, then for any* $f \in \mathcal{F}$ *and* $\epsilon \in \mathbb{R}_+$ *there exists* $\hat{f} \in \mathcal{T}_{\text{MATE}}$ *such that* $\|\hat{f} - f\|_p < \epsilon$.

See the Appendix C for a detailed proof, which relies on the fact that 3 heads will guarantee at least two heads of the same type. The problem can then be reduced to the results of Yun et al. (2020b).

4 The POINTR architecture

Many standard table QA datasets (Pasupat and Liang, 2015; Chen et al., 2020a; Iyyer et al., 2017), perhaps by design, use tables that can be limited to 512 tokens. Recently, more datasets (Kardas et al., 2020; Talmor et al., 2021) requiring parsing larger semi-structured documents have been released.

Original Input Table:

Pos	No	Driver	Constructor	Time	Gap
1	2	Rubens Barrichello	Ferrari	1:10.223 -	
2	1	Michael Schumacher	Ferrari	1:10.400	+0.177
3	10	Takuma Sato	Honda	1:10.601	+0.378
4	9	Jenson Button ⋆	Honda	1:10.820	+0.597

Entity Descriptions (some are omitted for space):

• *Rubens Barrichello* is a Brazilian racing driver who competed in Formula One between 1993 and 2011.
• *Jenson Alexander Lyons Button* MBE is a British racing driver. He won the 2009 F1 World Championship.
• *Scuderia Ferrari S.p.A.* is the racing division of luxury Italian auto manufacturer Ferrari. Ferrari supplied cars complete with V8 engines for the A1 Grand Prix series from the 2004 season.
• *Honda Motor Company, Ltd* is a Japanese public multinational conglomerate manufacturer of automobiles, motorcycles, and power equipment, headquartered in Minato, Tokyo, Japan.

⋮

Question:

The driver who finished in position 4 in the 2004 Grand Prix was of what nationality? **British**

⇓

Expanded Table with k Description Sentences Most Similar to the Question:

| Pos	No	Driver	Constructor	Time	Gap
1	2	Rubens Barrichello	Ferrari (Ferrari supplied cars complete with V8 engines for the A1 Grand Prix series from the 2004 season.)	1:10.223 -	
2	1	Michael Schumacher	Ferrari (Ferrari supplied cars complete with V8 engines for the A1 Grand Prix series from the 2004 season.)	1:10.400	+0.177
3	10	Takuma Sato	Honda	1:10.601	+0.378
4	9	Jenson Button (Jenson Alexander Lyons Button MBE is a **British** racing driver.) ⋆	Honda	1:10.820	+0.597

POINTR inference pipeline:
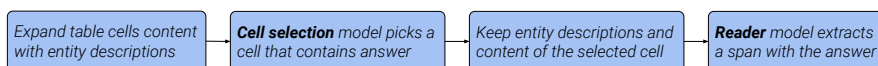
Figure 3: An example from the HYBRIDQA dataset processed by POINTR. The first paragraph in the Wikipedia page for each underlined entity was available to the dataset authors. We expand the text in the cells with this descriptions for the top-k most relevant sentences, as shown in the second table, and train a model to find the cell containing or linking to the answer (marked here with a ⋆). The goal is to provide the model with all the context needed to locate the answer. A second model extracts a span from the selected cell content and linked text.

Among them, we focus on HYBRIDQA (Chen et al., 2020b). It uses Wikipedia tables with entity links, with answers taken from either a cell or a hyperlinked paragraph. Dataset statistics are shown in Table 2. Each question contains a table with on average 70 cells and 44 linked entities. Each entity is represented by the first 12 sentences of the Wikipedia description, averaging 100 tokens. The answer is often a span extracted from the table or paragraphs but the dataset has no ground truth annotations on how the span was obtained, leaving around 50% of ambiguous examples where more than one answer-sources are possible. The total required number of word pieces accounting for the table, question and entity descriptions grows to more than $11,000$ if one intends to cover more than 90% of the examples, going well beyond the limit of traditional transformers.

To apply sparse transformers to the HYBRIDQA task, we propose POINTR, a two stage framework in a somewhat similar fashion to open domain QA pipelines (Chen et al., 2017; Lee et al., 2019). We *expand* the cell content by appending the descriptions of its linked entities. The two stages of POINTR correspond to (Point)ing to the correct *expanded cell* and then (R)eading a span from it. See Figure 3 for an example. Full set-up details are discussed in Appendix A.

4.1 POINTR: Cell Selection Stage

In the first stage we train a cell selection model using MATE whose objective is to select the expanded cell that contains the answer. MATE accepts the full table as input; therefore, expanding all the cells with their respective passages is impractical. Instead, we consider the top-k sentences in the entity descriptions for expansion, using a TF-IDF metric against the query. Using $k = 5$, we can fit 97% of the examples in 2048 tokens; for the remaining examples, we truncate the longest cells uniformly until they fit in the budget.

The logit score S for each cell c is obtained by

mean-pooling the logits for each of the tokens t inside it, which are in turn the result of applying a single linear layer to the contextual representation of each token when applying MATE to the query q and the expanded table e.

$$S(t) = \text{MLP}(\text{MATE}(q, e)[t])$$
$$S(c) = \text{avg}_{t \in c} S(t)$$
$$P(c) = \frac{\exp(S(c))}{\sum_{c' \in e} \exp(S(c'))}$$

We use cross entropy loss for training the model to select expanded cells that contain the answer span. Even though the correct span may appear in multiple cells or passages, in practice many of these do so only by chance and do not correspond to a reasoning path consistent with the question asked. In Figure 3 for instance, there could be other British divers but we are only interested in selecting the cell marked with a *star* symbol ($\star$). In order to handle these cases we rely on *Maximum Marginal Likelihood* (MML) (Liang et al., 2013; Berant et al., 2013). As shown by Guu et al. (2017) MML can be interpreted as using the online model predictions (without gradients) to compute a soft label distribution over candidates. For an input query x, and a set $\mathcal{C}$ of candidate cells, the loss is:

$$\mathcal{L}(\Theta, x, \mathcal{C}) = \sum_{z \in \mathcal{C}} -q(z) \log p_\Theta(z|x)$$

with $q(z) = p_\Theta(z|x, z \in \mathcal{C})$ the probability distribution given by the model restricted to candidate cells containing the answer span, taken here as a constant with zero gradient.

4.2 POINTR: Passage Reading Stage

In the second stage we develop a span selection model that reads the answer from a single *expanded* cell selected by the POINTR Cell Selector. In order to construct the expanded cell for each example, we concatenate the cell content with all the sentences of the linked entities and keep the first 512 tokens.

Following various recent neural machine reading works (Chen et al., 2017; Lee et al., 2019; Herzig et al., 2021), we fine-tune a pre-trained BERT-uncased-large model (Devlin et al., 2019) that attempts to predict a text span from the text in a given table cell c (and its linked paragraphs) and the input query q. We compute a span representation as the concatenation of the contextual embeddings of the first and last token in a span s

	TABFACT	WIKITQ	SQA
Examples	118,275	22,033	17,553
Tables	16,573	2,108	982

Table 3: Statistics for SQA, WIKITQ and TABFACT.

and score it using a multi-layer perceptron:

$$h_{start} = \text{BERT}_r(q, c)[\text{START}(s)]$$
$$h_{end} = \text{BERT}_r(q, c)[\text{END}(s)]$$
$$\text{S}_{\text{read}}(q, c) = \text{MLP}([h_{start}, h_{end}])$$

A softmax is computed over valid spans in the input and the model is trained with cross entropy loss. If the span-text appears multiple times in a cell we consider only the first appearance. To compute EM and F1 scores during inference, we evaluate the trained reader on the highest ranked cell output predictions of the POINTR Cell Selector using the official evaluation script.

5 Experimental Setup

We begin by comparing the performance of MATE on HYBRIDQA to other existing systems. We focus on prior efficient transformers to compare the benefits of the table-specific sparsity. We follow Herzig et al. (2020); Eisenschlos et al. (2020) in reporting error bars with the interquartile range.

5.1 Baselines

The first baselines for HYBRIDQA are Table-Only and Passage-Only, as defined in Chen et al. (2020b). Each uses only the part of the input indicated in the name but not both at the same time. Next, the HYBRIDER model from the same authors, consists of four stages: entity linking, cell ranking, cell hopping and finally a reading comprehension stage, equivalent to our final stage. The first three stages are equivalent to our single cell selection stage; hence, we use their reported error rates to estimate the retrieval rate. The simpler approach enabled by MATE avoids error propagation and yields improved results.

We also consider two recent efficient transformer architectures as alternatives for the POINTR Cell Selector, one based on LINFORMER (Wang et al., 2020) and one based on ETC (Ainslie et al., 2020). In both cases we preserve the row, column and rank embeddings introduced by Herzig et al. (2020). LINFORMER learns a projection matrix that reduces the sequence length dimension of the keys

and values tensor to a fixed length of 256 (which performed better than 128 and 512 in our tests.) ETC is a general architecture which requires some choices to be made about how to allocate global memory and local attention (Dai et al., 2019) Here we use a 256-sized global memory to summarize the content of each cell, by assigning each token in the first half of the global memory to a row, and each token in the second half to a column. Tokens in the input use a special relative positional value to mark when they are interacting with their corresponding global row or column memory position. We will refer to this model as TABLEETC.

Finally we consider two non-efficient models: A simple TAPAS model without any sparse mask, and an SAT (Zhang et al., 2020) model pretrained on the same MLM task as TAPAS for a fair comparison. For the cell selection task TAPAS obtains similar results to MATE, but both TAPAS and SAT lack the efficiency improvements of MATE.

5.2 Other datasets

We also apply MATE to three other datasets involving tables to demonstrate that the sparse attention bias yields stronger table reasoning models. SQA (Iyyer et al., 2017) is a sequential QA task, WIK-ITQ (Pasupat and Liang, 2015) is a QA task that sometimes also requires aggregation of table cells, and TABFACT (Chen et al., 2020a) is a binary entailment task. See Table 3 for dataset statistics. We evaluate with and without using the intermediate pre-training tasks (CS) (Eisenschlos et al., 2020).

6 Results

In Figure 4 we compare inference speed of different models as we increase the sequence length. Similar results showing number of FLOPS and memory usage are in Appendix A. The linear scaling of LINFORMER and the linear-time version MATE can be seen clearly. Although LINFORMER has a slightly smaller linear constant, the pre-training is 6 times slower, as unlike the other models, LIN-FORMER pretraining must be done at the final sequence length of 2048.

Table 5 shows the end-to-end results of our system using POINTR with MATE on HYBRIDQA, compared to the previous state-of-the-art as well as the other efficient transformer baselines from Section 5. MATE outperforms the previous SOTA HYBRIDER by over 19 points, and LINFORMER, the next best efficient-transformer system, by over
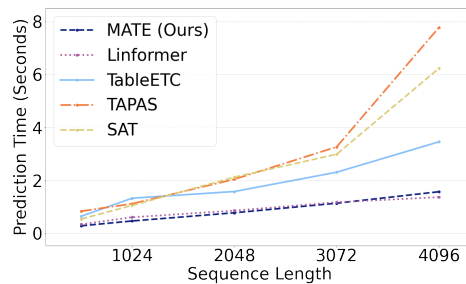

Figure 4: Comparison of inference speed on a cloud VM with 64GB. At a sequence length of 2048, MATE is nearly twice as fast as TAPAS.

Model	SQA ALL	SQA SEQ	WIKITQ	TABFACT
TAPAS	$67.2_{\pm 0.5}$	$40.4_{\pm 0.9}$	$42.6_{\pm 0.8}$	$76.3_{\pm 0.2}$
MATE	$71.6_{\pm 0.1}$	$46.4_{\pm 0.3}$	$42.8_{\pm 0.8}$	$77.0_{\pm 0.3}$
TAPAS + CS	$71.0_{\pm 0.4}$	$44.8_{\pm 0.8}$	$46.6_{\pm 0.3}$	$81.0_{\pm 0.1}$
MATE + CS	$71.7_{\pm 0.4}$	$46.1_{\pm 0.4}$	$51.5_{\pm 0.2}$	$81.4_{\pm 0.1}$

Table 4: Test results of using MATE on other table parsing datasets show improvements due to the sparse attention mechanism. Using *Counterfactual + Synthethic* pretraining (CS) in combination with MATE achieves state-of-the-art in SQA and TABFACT. Errors are estimated with half the interquartile range over 5 runs.

2.5 points, for both exact-match accuracy and F1.

We also applied MATE to three tasks involving table reasoning over shorter sequences. In Table 4 we see that MATE provides improvements in accuracy, which we attribute to a better inductive bias for tabular data. When combining MATE with *Counterfactual + Synthetic* intermediate pre-training (CS) (Eisenschlos et al., 2020) we often get even better results. For TABFACT and SQA we improve over the previous state-of-the-art. For WIKITQ we close the gap with the best published system TABERT (Yin et al., 2020) (51.8 mean test accuracy), which relies on traditional semantic parsing, instead of an end-to-end approach. Dev results show a similar trend and can be found in Appendix B. No special tuning was done on these models—we used the same hyper-parameters as the open source release of TAPAS.

7 Analysis

HYBRIDQA Error analysis We randomly sample 100 incorrectly answered examples from the development set. 55% of the examples have lexical near-misses—predictions have the correct information, but have slightly different formatting (e.g. (Q)uestion: *In what round was the Okla-*

Model	Dev						Test					
	In-Table		In-Passage		Total		In-Table		In-Passage		Total	
	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
Table-Only	14.7	19.1	2.4	4.5	8.4	12.1	14.2	18.8	2.6	4.7	8.3	11.7
Passage-Only	9.2	13.5	26.1	32.4	19.5	25.1	8.9	13.8	25.5	32.0	19.1	25.0
HYBRIDER (τ=0.8)	54.3	61.4	39.1	45.7	44.0	50.7	56.2	63.3	37.5	44.4	43.8	50.6
POINTR + SAT	$66.5_{\pm0.33}$	$71.8_{\pm0.28}$	$60.3_{\pm0.11}$	$69.2_{\pm0.04}$	$61.2_{\pm0.29}$	$68.7_{\pm0.31}$	64.6	70.1	59.6	68.5	60.1	67.4
POINTR + TAPAS	$68.1_{\pm0.33}$	$73.9_{\pm0.37}$	$62.9_{\pm0.25}$	$72.0_{\pm0.21}$	$\mathbf{63.3}_{\pm0.25}$	$\mathbf{70.8}_{\pm0.12}$	67.8	73.2	62.0	70.9	62.7	70.0
POINTR +TABLEETC	$36.0_{\pm1.26}$	$42.4_{\pm1.13}$	$37.8_{\pm1.19}$	$45.3_{\pm1.53}$	$36.1_{\pm1.30}$	$42.9_{\pm1.36}$	35.8	40.7	38.8	45.7	36.6	42.6
POINTR + LINFORMER	$65.5_{\pm0.78}$	$71.1_{\pm0.55}$	$59.4_{\pm0.59}$	$69.0_{\pm0.68}$	$60.8_{\pm0.68}$	$68.4_{\pm0.63}$	66.1	71.7	58.9	67.8	60.2	67.6
POINTR + MATE	$68.6_{\pm0.37}$	$74.2_{\pm0.26}$	$62.8_{\pm0.25}$	$71.9_{\pm0.20}$	$\mathbf{63.4}_{\pm0.16}$	$\mathbf{71.0}_{\pm0.17}$	66.9	72.3	62.8	71.9	$\mathbf{62.8}$	$\mathbf{70.2}$
Human											88.2	93.5

Table 5: Results of different large transformer models on HYBRIDQA. In-Table and In-Passage subsets refer to the location of the answer. For dev, we report errors over 5 runs using half the interquartile range. Since the test set is hidden and hosted online, we report the results corresponding to the model with the median total EM score on dev.

homa athlete drafted in? (G)old answer: "second", (P)redicted: "second round"). While around 30% of such misses involved numerical answers (eg: "1" vs "one"), the predictions for the rest of them prominently (58% of the near misses) either had redundant or were missing auxiliary words (e.g., Q: *What climate is the northern part of the home country of Tommy Douglas?* G: "Arctic" P: "Arctic climate"). The inconsistency in the gold-answer format and unavailability of multiple gold answers are potential causes here.

Among the non near-misses, the majority predictions were either numerically incorrect, or were referencing an incorrect entity but still in an relevant context—especially the questions involving more than 2 hops. (e.g. Q: *In which sport has an award been given every three years since the first tournament held in 1948-1949?* G: "Badminton", P: "Thomas Cup"). Reassuringly, for a huge majority (> 80%), the entity type of the predicted answer (person, date, place, etc.) matches the type of the gold answer. The observed errors suggest potential gains by improving the entity (Xiong et al., 2020) and numerical (Andor et al., 2019) reasoning skills.

Ablation Study In Table 6 we compare architectures for cell selection on HYBRIDQA. Hits@k corresponds to whether a cell containing an answer span was among the top-k retrieved candidates. As an ablation, we remove the sparse pre-training and try using only row/column heads. We observe a drop also when we discard the ambiguous examples from training instead of having MML to deal with them. Unlike the other datasets, TAPAS shows comparable results to MATE, but without any of the theoretical and practical improvements.

Model	Hits@1	Hits@3	Hits@5
HYBRIDER	68.5	-	-
SAT	77.9	87.4	90.3
TAPAS	**80.1**	**89.5**	91.4
TABLEETC	51.1	72.0	78.9
LINFORMER	77.1	86.5	90.0
MATE	**80.1**	89.2	**91.5**
MATE (– row heads)	78.3	87.7	90.3
MATE (– col heads)	77.8	87.1	90.0
MATE (– sparse pretrain)	75.5	86.5	89.9
MATE (– ambiguous)	76.7	84.2	86.6

Table 6: Retrieval results over HYBRIDQA (dev set) for models used in POINTR Cell Selection stage. Efficient transformer models are grouped together. HYBRIDER results are obtained from Chen et al. (2020b) by composing the errors for the first components.

Observed Attention Sparsity Since we are interested to motivate our choices on how to sparsify the attention matrix, we can inspect the magnitude of attention connections in a trained dense TAPAS model for table question answering. It is important to note that in this context we are not measuring attention as an explanation method (Jain and Wallace, 2019; Wiegreffe and Pinter, 2019). Instead we are treating the attention matrix in the fashion of magnitude based pruning techniques (Han et al., 2015; See et al., 2016), and simply consider between which pairs of tokens the scores are concentrated.

Given a token in the input we can aggregate the attention weights flowing from it depending on the position of the target token in the input (CLS token, question, header, or table) and whether the source and target tokens are in the same column or row, whenever it makes sense. We average scores across all tokens, heads, layers and examples in the development set. As a baseline, we also compare against the output of the same process when using

Type	Same column	Same row	Attention / Uniform	Attention %	Uniform %
[CLS]			44.46	12.45	0.28
Question			10.13	37.08	3.66
Header	✗		1.32	3.89	2.94
	✓		3.37	1.72	0.51
Table	✗	✗	**0.34**	23.59	68.18
	✗	✓	0.87	3.51	4.02
	✓	✗	0.70	13.16	18.84
	✓	✓	2.93	4.60	1.57

Table 7: Average attention flow from a token in the table to other token types. We compare to an uniform attention matrix as a baseline. Attention to tokens in different rows and columns is the relative smallest with one third of the baseline. Computed on WIKITQ Dev.

a uniform attention matrix, discarding padding.

In Table 7, we show the obtained statistics considering only table tokens as a source. We use the WIKITQ development set as a reference. While we see that 23% of the attention weights are looking at tokens in different columns and rows, this is only about one third of the baseline number one would obtain with a uniform attention matrix. This effect corroborates the approach taken in MATE.

8 Conclusion

We introduce MATE, a novel method for efficiently restricting the attention flow in Transformers applied to Tabular data. We show in both theory and practice that the method improves inductive bias and allows scaling training to larger sequence lengths as a result of linear complexity. We improve the state-of-the-art on TABFACT, SQA and HYBRIDQA, the last one by 19 points.

Ethical Considerations

Although one outcome of this research is more efficient Transformers for table data, it remains true that large Transformer models can be expensive to train from scratch, so experiments of this sort can incur high monetary cost and carbon emissions. This cost was reduced by conducting some experiments at relatively smaller scale, e.g. the results of Figure 4. To further attenuate the impact of this work, we plan release all the models that we trained so that other researchers can reproduce and extend our work without re-training.

All human annotations required for the error analysis (Section 7) are provided by authors, and hence a concern of fair compensation for annotators did not arise.

References

Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. 2020. ETC: Encoding long and structured inputs in transformers. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 268–284, Online. Association for Computational Linguistics.

Daniel Andor, Luheng He, Kenton Lee, and Emily Pitler. 2019. Giving BERT a calculator: Finding operations and arguments with reading comprehension. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 5947–5952, Hong Kong, China. Association for Computational Linguistics.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA. Association for Computational Linguistics.

Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. Uncovering the relational web. In *WebDB*.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to answer open-domain questions. In *Proceedings of the Association for Computational Linguistics*, pages 1870–1879, Vancouver, Canada. Association for Computational Linguistics.

Wenhu Chen, Ming-Wei Chang, Eva Schlinger, William Yang Wang, and William W. Cohen. 2021. Open question answering over tables and text. In *International Conference on Learning Representations*.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2020a. Tabfact: A large-scale dataset for table-based fact verification. In *Proceedings of the International Conference on Learning Representations*.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. 2020b. HybridQA: A dataset of multi-hop question answering over tabular and textual data. In *Findings of the Association for Computational Linguistics: EMNLP*, pages 1026–1036, Online. Association for Computational Linguistics.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. 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.

Julian Eisenschlos, Syrine Krichene, and Thomas Müller. 2020. Understanding tables with intermediate pre-training. In *Findings of the Association for Computational Linguistics: EMNLP*, pages 281–296, Online. Association for Computational Linguistics.

Kelvin Guu, Panupong Pasupat, Evan Liu, and Percy Liang. 2017. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *Proceedings of the Association for Computational Linguistics*, pages 1051–1062, Vancouver, Canada. Association for Computational Linguistics.

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Proceedings of Advances in Neural Information Processing Systems*, volume 28, pages 1135–1143. Curran Associates, Inc.

Jonathan Herzig, Thomas Müller, Syrine Krichene, and Julian Eisenschlos. 2021. Open domain question answering over tables via dense retrieval. In *Conference of the North American Chapter of the Association for Computational Linguistics*, pages 512–519, Online. Association for Computational Linguistics.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. In *Proceedings of the Association for Computational Linguistics*, pages 4320–4333, Online. Association for Computational Linguistics.

Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 2017. Search-based neural structured learning for sequential question answering. In *Proceedings of the Association for Computational Linguistics*, pages 1821–1831, Vancouver, Canada. Association for Computational Linguistics.

Sarthak Jain and Byron C. Wallace. 2019. Attention is not Explanation. In *Conference of the North American Chapter of the Association for Computational Linguistics*, pages 3543–3556, Minneapolis, Minnesota. Association for Computational Linguistics.

Marcin Kardas, Piotr Czapla, Pontus Stenetorp, Sebastian Ruder, Sebastian Riedel, Ross Taylor, and Robert Stojnic. 2020. AxCell: Automatic extraction of results from machine learning papers. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 8580–8594, Online. Association for Computational Linguistics.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. In *Proceedings of the International Conference on Learning Representations*.

Syrine Krichene, Thomas Müller, and Julian Eisenschlos. 2021. DoT: An efficient double transformer for NLP tasks with tables. In *Findings of the Association for Computational Linguistics: ACL*, pages 3273–3283, Online. Association for Computational Linguistics.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association of Computational Linguistics*.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. In *Proceedings of the Association for Computational Linguistics*, pages 6086–6096, Florence, Italy. Association for Computational Linguistics.

Percy Liang, Michael I. Jordan, and Dan Klein. 2013. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the Association for Computational Linguistics*, pages 1470–1480, Beijing, China. Association for Computational Linguistics.

Abigail See, Minh-Thang Luong, and Christopher D. Manning. 2016. Compression of neural machine translation models via pruning. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 291–301, Berlin, Germany. Association for Computational Linguistics.

Alon Talmor, Ori Yoran, Amnon Catav, Dan Lahav, Yizhong Wang, Akari Asai, Gabriel Ilharco, Hannaneh Hajishirzi, and Jonathan Berant. 2021. Multimodal{qa}: complex question answering over text, tables and images. In *Proceedings of the International Conference on Learning Representations*.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient transformers: A survey.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity.

Sarah Wiegreffe and Yuval Pinter. 2019. Attention is not not explanation. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 11–20, Hong Kong, China. Association for Computational Linguistics.

Wenhan Xiong, Jingfei Du, William Yang Wang, and Veselin Stoyanov. 2020. Pretrained encyclopedia: Weakly supervised knowledge-pretrained language model. In *Proceedings of the International Conference on Learning Representations*.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the Association for Computational Linguistics*, pages 8413–8426, Online. Association for Computational Linguistics.

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank Reddi, and Sanjiv Kumar. 2020a. Are transformers universal approximators of sequence-to-sequence functions? In *Proceedings of the International Conference on Learning Representations*.

Chulhee Yun, Yin-Wen Chang, Srinadh Bhojanapalli, Ankit Rawat, Sashank Reddi, and Sanjiv Kumar. 2020b. O(n) connections are expressive enough: Universal approximability of sparse transformers. In *Proceedings of Advances in Neural Information Processing Systems*.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. In *Proceedings of Advances in Neural Information Processing Systems*, volume 33.

Hongzhi Zhang, Yingyao Wang, Sirui Wang, Xuezhi Cao, Fuzheng Zhang, and Zhongyuan Wang. 2020. Table fact verification with structure-aware transformer. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 1624–1629, Online. Association for Computational Linguistics.

Appendix

We provide all details on our experimental setup to reproduce the results in Section A. In Section B we show the development set results for our experiments. The proof for Theorem 1 is described in Section C and in Section D we include the main code blocks for implementing MATE efficiently on a deep learning framework.

A Experimental setup

A.1 Pre-training

Pre-training for MATE was performed with constrained attention with a masked language modeling objective applied to the corpus of tables and text extracted by Herzig et al. (2020). With a sequence length of 128 and batch size of 512, the total training of 1 million steps took 2 days.

In contrast, for LINFORMER the pre-training was done with a sequence length of 2048 and a batch size of 128, and the total training took 12 days for 2 million steps. For TABLEETC we also pre-trained for 2 million steps but the batch size had to be lowered to 32. In all cases the hardware used was a 32 core **Cloud TPUs V3**.

A.2 Fine-tuning

For all experiments we use Large models over 5 random seeds and report the median results. Errors are estimated with half the interquartile range. For TABFACT, SQA and WIKITQ we keep the original hyper-parameters used in TAPAS and provided in the open source release. In Figure 5 we show the floating point operation count of the different Transformer models as we increase the sequence length, as extracted from the execution graph. We also measure the memory doing CPU inference in figure 6. The linear scaling of LINFORMER and MATE can be observed. No additional tuning or sweep was done to obtain the published results. We set the global size G to 116 and the radius R for local attention to 42. We use an Adam optimizer with weight decay with the same configuration as BERT. The number of parameters for MATE is the same as for BERT: $340M$ for Large models and $110M$ for Base Models.

In the HYBRIDQA cell selection stage, we use a batch size of 128 and train for $80,000$ steps and a sequence length of 2048. Training requires 1 day. We clip the gradients to 10 and use a learning rate of 1×10^{-5} under a 5% warm-up schedule. For the
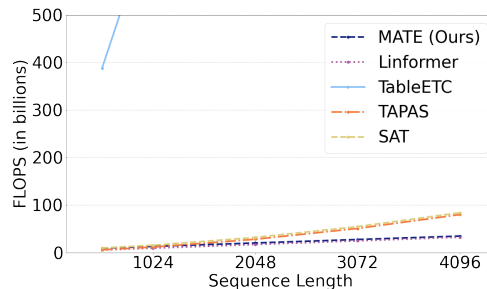

Figure 5: Comparison of inference FLOPS obtained from execution graph. While TABLEETC is linear, relative attention adds a high computation cost so keep it out of range in the figure.
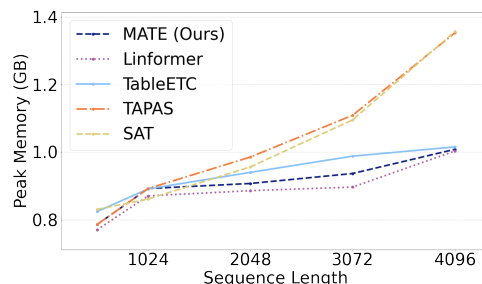

Figure 6: Comparison of the peak memory usage during CPU inference shows the linear asymptotic curve of the memory footprint of MATE.

reader stage use a learning rate of 5×10^{-5} under a 1% warm-up schedule, a batch size of 512 and train for $25,000$ steps, which takes around 6 hours.

B Development set results for SQA, WIKITQ and TABFACT

We show in Table 8 the dev set results for all datasets we attempted, which show consistent results with the test set reported in the main paper.

C Proof of Theorem 1

In this section we discuss the proof that MATE are universal approximators of sequence functions.

Theorem. *If the number of heads is at least* 3 *and*

Model	SQA ALL	SQA SEQ	WIKITQ	TABFACT
TAPAS	$64.9_{\pm 0.5}$	$40.0_{\pm 1.0}$	$41.6_{\pm 1.0}$	$76.9_{\pm 0.4}$
MATE	$67.0_{\pm 0.1}$	$43.2_{\pm 0.2}$	$42.9_{\pm 0.6}$	$77.5_{\pm 0.3}$
TAPAS + CS	$\mathbf{68.0}_{\pm 0.2}$	$\mathbf{45.8}_{\pm 0.3}$	$46.2_{\pm 0.2}$	$81.0_{\pm 0.1}$
MATE + CS	$\mathbf{68.0}_{\pm 0.4}$	$44.9_{\pm 0.4}$	$\mathbf{50.1}_{\pm 0.7}$	$\mathbf{81.3}_{\pm 0.1}$

Table 8: Dev results of using MATE on other table parsing datasets. Errors are estimated with half the interquartile range over 5 runs.

the hidden size of the feed forward layer is at least 4, then for any $f \in \mathcal{F}$ and $\epsilon \in \mathbb{R}_+$ there exists $\hat{f} \in \mathcal{T}_{\text{MATE}}$ such that $||\hat{f} - f||_p < \epsilon$

Proof. When the number of heads is at least 3, there are at least 2 heads of the same type. Fixing those two heads, we may restrict the value of the projection weights W_V to be 0 for the rest of the heads. This is equivalent to having only those two heads with the same attention pattern to begin with. This restriction only makes the family of functions modelled by MATE smaller. In a similar way, we can assume that the hidden size of the feed-forward layer is exactly 4 and that the head size is 1.

Note that the attention pattern of the two heads, regardless of its type contains a token (the first one) which attends to and from every other token. We also have that every token attends to itself. Then Assumption 1 of Yun et al. (2020b) is satisfied. Hence we rely on Theorem 1 of Yun et al. (2020b), which asserts that sparse transformers with 2 heads, hidden size 4 and head size 1 are universal approximators, which concludes the proof. □

D   TensorFlow Implementation

In figure 7 we provide an approximate implementation of MATE in the TensorFlow library. For the sake of simplicity we omit how attention is masked between neighbor buckets for tokens in difference columns or rows. We also omit the tensor manipulation steps to reorder and reshape the sequence into equally sized buckets to compute attention across consecutive buckets. The full implementation will be part of the open source release.

```python
import dataclasses
import tensorflow as tf

@dataclasses.dataclass
class MultiViewEmbedding():
  """Results of sorting and reshaping an embedding tensor.

  Different views of the tensor created to facilitate attention across tokens
  from global/long parts. First two dimensions are 'batch_size' and 'num_heads':

  Attributes:
    full: <float32>[..., seq_length, embedding_size].
      Original tensor without any bucketing.
    global: <float32>[..., global_length, embedding_size].
    long: <float32>[..., long_length/radius, radius, embedding_size]
    window: <float32>[..., long_length/radius, 3*radius, embedding_size]
      Same as 'long' but also a rotation to the left and right, in order
      to achieve attention to the previous and next bucket.
  """
  full: tf.Tensor
  global: tf.Tensor
  long: tf.Tensor
  window: tf.Tensor

def multi_view_attention(
    Q: MultiViewEmbedding,
    K: MultiViewEmbedding,
    V: MultiViewEmbedding,
    embedding_size: int,
    global_length: int,
    long_length: int,
    num_heads: int,
):
  # <float32>[batch_size, num_heads, global_length, sequence_length]
  attention_prob_from_global = tf.nn.softmax(tf.einsum(
      'BHFE,BHTE->BHFT', Q.global, K.full) / sqrt(embedding_size))
  # <float32>[batch_size, num_heads, long_length, global_length]
  attention_score_to_global = tf.einsum('BHNFE,BHTE->BHNFT',
      Q.long, K.global) / sqrt(embedding_size)
  # <float32>[batch_size, num_heads, long_length, 3 * radius]
  attention_score_to_window = tf.einsum('BHNFE,BHNTE->BHNFT',
      Q.long, K.window) / sqrt(embedding_size)
  # <float32>[batch_size, num_heads, long_length, global_length + 3 * radius]
  attention_prob_from_long = tf.nn.softmax(tf.concat(
      [attention_score_to_global, attention_score_to_window], axis=-1))
  attention_prob_to_global = attention_prob_from_long[..., :global_length]
  attention_prob_to_window = attention_prob_from_long[..., global_length:]

  # <float32>[batch_size, num_heads, global_length, embedding_size]
  context_layer_from_global = tf.einsum('BHFT,BHTE->BHFE',
      attention_prob_from_global, V.full)
  # <float32>[batch_size, num_heads, long_length / radius, radius, embedding_size]
  context_layer_to_global = tf.einsum('BHNFT,BHTE->BHNFE',
      attention_prob_to_global, V.global)
  # <float32>[batch_size, num_heads, long_length / radius, radius, embedding_size]
  context_layer_to_window = tf.einsum('BHNFT,BHNTE->BHNFE',
      attention_prob_to_window, V.window)

  context_layer_from_long = tf.reshape(
      context_layer_to_first + context_layer_to_window,
      [-1, num_heads, long_length, embedding_size])
  return tf.concat(
      [context_layer_from_global, context_layer_from_long], axis=-1)
```

Figure 7: Implementation of MATE in TensorFlow. The creation of `MultiViewEmbedding` is ommited and relies on `tf.gather` for ordering the input. We also omit the use of the input mask and column and row index to further mask the sparse attention matrix.