Memory-efficient Transformers via Top-k Attention

Ankit Gupta[1*] **Guy Dar**[2] **Shaya Goodman**[2] **David Ciprut**[2] **Jonathan Berant**[2,3]

[1]IBM Research [2]Tel Aviv University [3]Allen Institute for AI

`{ankitgupta.iitkanpur, guyd1995, ishayahu156, davidciprut}@gmail.com, joberant@cs.tau.ac.il`

Abstract

Following the success of dot-product attention in Transformers, numerous approximations have been recently proposed to address its quadratic complexity with respect to the input length. While these variants are memory and compute efficient, it is not possible to directly use them with popular pre-trained language models trained using vanilla attention, without an expensive corrective pre-training stage. In this work, we propose a simple yet highly accurate approximation for vanilla attention. We process the queries in chunks, and for each query, compute the top-k scores with respect to the keys. Our approach offers several advantages: (a) its memory usage is linear in the input size, similar to linear attention variants, such as Performer and RFA (b) it is a drop-in replacement for vanilla attention that does not require any corrective pre-training, and (c) it can also lead to significant memory savings in the feed-forward layers after casting them into the familiar query-key-value framework. We evaluate the quality of top-k approximation for multi-head attention layers on the Long Range Arena Benchmark, and for feed-forward layers of T5 and UnifiedQA on multiple QA datasets. We show our approach leads to accuracy that is nearly-identical to vanilla attention in multiple setups including training from scratch, fine-tuning, and zero-shot inference.

1 Introduction

The Transformer architecture (Vaswani et al., 2017) has been successful in a wide range of natural language processing tasks, including machine translation (Edunov et al., 2018), language modeling (Roy et al., 2021), question-answering (Karpukhin et al., 2020), and many more. Transformers pre-trained on large amounts of text with a language modeling (LM) objective, have become the standard in NLP, exhibiting surprising amounts of linguistic and world knowledge (Peters et al., 2018; Devlin et al., 2019; Petroni et al., 2019; Hewitt and Manning, 2019; Roberts et al., 2020).

The contextualizing component of the Transformer is the attention layer where all positions in an input sequence of length L aggregate information from the entire sequence in parallel. At its core, given L query, key, and value vectors Q, K, V respectively, the *dot-product attention* function outputs $\mathrm{softmax}(QK^\top)V$ where the $\mathrm{softmax}$ function is applied row-wise on the matrix $QK^\top \in \mathbb{R}^{L\times L}$ of similarity scores of the query-key pairs, leading to an expensive $\Omega(L^2)$ memory requirement.

To alleviate this, past work proposed approximation methods for the computation of $\mathrm{softmax}(QK^\top)$. One major line of research focused on *sparse attention* variants, where only a few similarity scores are computed per query, and the rest are ignored. Methods differ by which query-key pairs are selected (Child et al., 2019; Ye et al., 2019; Qiu et al., 2020; Roy et al., 2021; Kitaev et al., 2020; Beltagy et al., 2020; Gupta and Berant, 2020; Vyas et al., 2020). A second line of research explored *dense* variants (Katharopoulos et al., 2020; Wang et al., 2020; Bello, 2021; Tay et al., 2020a) (cf. (Tay et al., 2020b) for a survey). For example, instead of computing the attention scores exactly for only a small number of query-key pairs, (Choromanski et al., 2021) compute an approximation of scores for all pairs.

In this work, we adopt the sparse attention approach, but rather than *approximating* the k most similar key vectors per query vector, we compute this quantity *exactly*. Specifically, we propose *top-k attention* where, for each query vector, we only keep its k largest similarity scores with respect to the L keys, where $k \ll L$. We show that top-k attention can be implemented in a memory-efficient manner by (a) chunking the query vectors when

---

[*] majority of work done while author was part of IBM AI Residency program.

Proceedings of the 2nd Workshop on Simple and Efficient Natural Language Processing, pages 39–52
November 10, 2021.

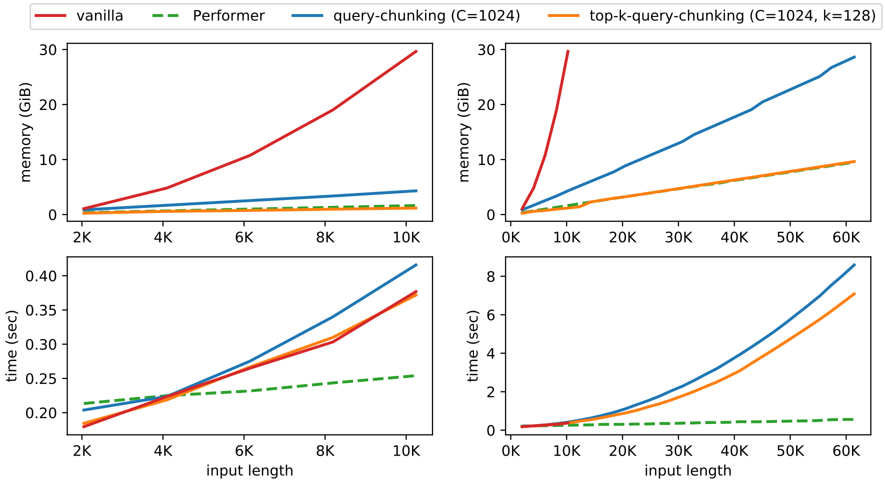


Figure 1: Memory and time required for a forward and backward pass on a single BERT-base multi-head self-attention layer with causal masking on short (left) and long (right) inputs. Details are in §3.

computing the output one chunk at a time, when computing $\mathrm{softmax}(QK^{\top})V$, and (b) a custom implementation of the forward and backward pass that does not require caching activations while processing chunks in the forward pass.

Compared to prior methods, top-k attention has multiple attractive properties:

- Top-k attention has the same memory footprint as Performer (Choromanski et al., 2021), a state-of-the-art attention variant with linear time and memory complexity, on very long inputs (orange curve, Fig. 1, top-right), while being as fast as vanilla attention, and even faster than linear variants on inputs of length up to $4K$ (Figure 1, bottom-left). This allows us, e.g., to train a typical 12-layer Transformer decoder over $32K$-long inputs on a 30GiB GPU (Figure 3a).
- Top-k attention also reduces memory consumption in Transformer feed-forward layers, by casting this layer into the familiar query-key-value framework using ReLU instead of the row-wise $\mathrm{softmax}$ (Sukhbaatar et al., 2019). This is specifically appealing in models such as T5 (Raffel et al., 2020) and GPT-3 (Brown et al., 2020), where for short inputs, the memory consumption is dominated by the feed-forward layers, as the number of keys, corresponding to the feed-forward hidden dimension size, is as large as $65K$. Conversely, methods that rely on random feature approximations of attention, such as Performer (Choromanski et al., 2021) and RFA (Peng et al., 2021) do not admit an efficient approximation for the ReLU activation (Yehudai and Shamir, 2019).
- Top-k attention is a highly accurate approximation to vanilla attention and is a *plug-and-play* replacement at both multi-head attention and feed-forward layers of a Transformer. This is unlike past attention variants (Katharopoulos et al., 2020; Choromanski et al., 2021; Peng et al., 2021) that require an expensive corrective pre-training stage to adjust model weights to the new variant, which can be prohibitive for large models. We show top-k attention can replace vanilla attention in a zero-shot inference setup and at fine-tuning time without any corrective pre-training.

We extensively evaluate top-k attention on a wide range of tasks and demonstrate its mentioned advantages. Training from scratch, we show top-k attention performs as well as vanilla self-attention on Long Range Arena, a benchmark dedicated to evaluating the ability of transformers to handle long sequences, and in a language modeling task (WikiText-103). Second, we show top-k attention can be used as a drop-in replacement for vanilla attention at inference time without any additional training at the feed-forward layer of the UnifiedQA model (Khashabi et al., 2020) on 12 different question answering (QA) datasets, reducing the number of keys used per query by more than 99%. Last, we show top-k attention obtains similar performance to vanilla attention on a wide range of QA tasks when fine-tuning T5 (Raffel et al., 2020), without the need for any corrective pre-training.

Overall, our results demonstrate that top-k attention is a simple and effective method for dramatically reducing the memory footprint of Transformers without loss of accuracy that can allow resource-constrained researchers enjoy the benefits of large pre-trained Transformer-based models. Our code is available at `https://github.com/ag1988/top_k_attention`.

2 Efficient Transformer through Top-k Attention

In this section, we briefly review the Transformer architecture, its sparse approximations, and show how to cast the feed-forward layer into the query-key-value framework (§2.1). We then describe top-k attention and our memory-efficient implementation for it (§2.2).

2.1 Attention in Transformers

A Transformer (Vaswani et al., 2017) is a stack of layers each consisting of multi-head attention and feed-forward sub-layers. Its contextualizing component is the multi-head attention defined as follows.

Multi-head Attention Given a query $Q \in \mathbb{R}^{L_Q \times d}$, key $K \in \mathbb{R}^{L_K \times d}$ and value $V \in \mathbb{R}^{L_K \times d}$, the output $\in \mathbb{R}^{L_Q \times d}$ of dot-product attention is defined as:

$$\mathrm{Attention}(Q, K, V) = \text{row-softmax}\left(\frac{QK^\top}{\lambda}\right) V, \tag{1}$$

where λ is an optional temperature typically fixed as $\sqrt{d}$.[1] In multi-head attention, for a given number of heads h, instead of computing a single attention output with d_{model} dimensional queries, keys and values, these are linearly projected down in parallel h times to $d = d_{\text{model}}/h$ dimensions, using different learned projection matrices. Attention is applied to each of the h new queries, keys and values, yielding d dimensional outputs which are concatenated and again projected to obtain a d_{model}-dimensional output.

Sparse approximations The attention function (Eq. 1) requires the computation of $QK^\top$ containing $L_Q \cdot L_K$ entries and can be expensive for long sequences (L_Q and L_K are typically the sequence length). To alleviate this issue, *sparse attention* variants (Child et al., 2019; Qiu et al., 2020; Kitaev et al., 2020; Beltagy et al., 2020; Gupta and Berant, 2020) relax this requirement and compute only a few entries of $QK^\top$, masking out the rest. For a binary mask $B \in \{0, -\infty\}^{L_Q \times L_K}$,

$$\begin{aligned}&\mathrm{SparseAttention}(Q, K, V, B) = \\ &\text{row-softmax}\left(QK^\top + B\right) V.\end{aligned} \tag{2}$$

The sparsity of B can be leveraged via customized implementations of matrix product (Child et al., 2019; Beltagy et al., 2020) and, thus Eq. 2 can be significantly cheaper to compute compared to Eq. 1.

Feed-forward as attention In the feed-forward layer, a 1-hidden layer fully-connected network is applied identically to every input token. As observed in past work (Sukhbaatar et al., 2019; Shazeer, 2020; Geva et al., 2020), a feed-forward layer can be cast into the query-key-value framework as:

$$\mathrm{FF}_{K,V}(Q) = \mathrm{ReLU}\left(QK^\top\right) V. \tag{3}$$

In this case, the queries $Q \in \mathbb{R}^{L_Q \times d_{\text{model}}}$ are the inputs to the layer with L_Q tokens, similar to self-attention. However, the keys $K = W_K \in \mathbb{R}^{L_K \times d_{\text{model}}}$ and values $V = W_V \in \mathbb{R}^{L_K \times d_{\text{model}}}$ are learned parameters that are independent of the input. The number of keys L_K here is known as the *feed-forward dimension* and can be as large as $65K$ for wide models such as T5 (Raffel et al., 2020) and GPT-3 (Brown et al., 2020). In the common case where the input sequences are relatively short, memory consumption is dominated by the feed-forward sub-layer and not the self-attention sub-layer.

Unlike top-k attention, past approaches for approximating attention are incompatible with feed-forward layers. Most approximate attention variants, such as Sparse Transformer (Child et al., 2019), LongFormer (Beltagy et al., 2020), BigBird (Zaheer et al., 2020), Sinkhorn attention (Tay et al., 2020a), rely on a *locality bias* in sequences, where the key vectors indexed close to each other in K are assumed to have similar representations. This is irrelevant for keys in a feed-forward layer, which are permutation-equivariant and do not have any local structure. Dense attention variants relying on random fourier features for approximating the $\mathrm{softmax}$ function are also not applicable, since it is known that ReLU cannot be approximated using such features (Yehudai and Shamir, 2019).

2.2 Top-k Attention

In this work we propose top-k attention, where for each query, we mask out all but its k largest dot products with the keys, that is, in each row of $QK^\top$ we only keep its k largest elements and mask out the rest:

$$\begin{aligned}&\text{top-}k\text{-Attention}(Q, K, V) = \\ &\mathrm{activation}\left(\text{top-}k(QK^\top)\right) V,\end{aligned} \tag{4}$$

---

[1] we omit this in rest of our presentation as Q can be scaled by $1/\lambda$ beforehand.

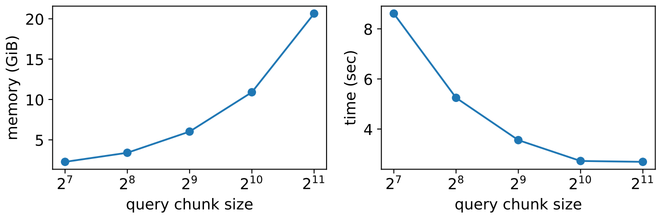


Figure 2: Memory and time required for a forward pass on a single BERT-base multi-head self-attention layer on inputs of length $65,536$.

where $\mathrm{activation}$ can be $\mathrm{softmax}$, ReLU, or any other activation, and $\text{top-}k(QK^\top)$ denotes a sparse matrix consisting only of the row-wise top-k elements of $QK^\top$. A naïve approach for computing $\text{top-}k(QK^\top)$ would be to first compute $QK^\top$ and applying a row-wise top-k operation. Unfortunately, computing $QK^\top \in \mathbb{R}^{L_Q \times L_K}$ explicitly would require $\Omega\left(L_Q \cdot L_K\right)$ memory. We now describe our approach, which avoids this high cost.

Query chunking A simple way to implement attention and reduce its peak memory consumption is to chunk queries: instead of processing all the queries at once, we partition the queries into chunks and process them sequentially, one chunk at a time. For a chunk size C, the rows of Q are grouped into L_Q/C contiguous chunks of size C and the attention function (Eq. 1, 3, 4) is computed using $Q_\mathcal{C}, K, V$ as inputs where $Q_\mathcal{C}$ denotes the subset of Q corresponding to chunk $\mathcal{C}$.

During *inference*, once a query chunk is fully processed, the intermediate activations produced during its processing can be discarded and, hence, the peak memory required to process all L_Q queries is bounded by the memory required to process a single chunk. Therefore, modulo the storage required for Q, K, V and the outputs themselves, the peak memory usage reduces from $\Omega\left(L_Q \cdot L_K\right)$ to $O(C \cdot L_K)$ which is linear with respect to L_K for a fixed chunk size C.

Chunk size provides a simple way to trade-off between the maximum memory usage and the slowdown due to the sequential processing of chunks. Fig. 2 shows memory and time for different chunk sizes for a single BERT-base self-attention layer over a sequence of length $65,536$. We observe that chunk sizes $2^9, 2^{10}$ yield a good trade-off between time and memory.

Input checkpointing While query chunking provides a straightforward approach for bounding the peak memory usage of attention during *inference*, it is not so straightforward to employ it during *training*. Let $d\left(A\right)$ denote the gradient of the loss with respect to a tensor A. For a given query chunk $Q_\mathcal{C}$, the intermediate activations produced during the computation of the output $o_\mathcal{C} = \mathrm{Attention}(Q_\mathcal{C}, K, V)$ are required for computing $d\left(Q_\mathcal{C}\right)$ from $d\left(o_\mathcal{C}\right)$ via backpropagation. Unfortunately, for the above bound on the peak memory usage to hold, we cannot afford to cache these activations for all the chunks, as done by standard automatic differentiation packages.

Taking inspiration from *gradient checkpointing* (Chen et al., 2016), we observe that if the inputs $Q_\mathcal{C}, K, V$ are available during the backward pass, we can re-compute $o_\mathcal{C}$ and then use the produced intermediate activations to compute $d\left(Q_\mathcal{C}\right)$ from $d\left(o_\mathcal{C}\right)$. Once $d\left(Q_\mathcal{C}\right)$ is computed, we can again discard the intermediate activations and gradients produced during this step and move on to the next chunk. This ensures that the peak memory usage during the backward pass through the attention layer is bounded by the memory required to backpropagate through a single chunk.

To summarize, a customized backward pass allows us to utilize query chunking, both during forward and backward passes, and only requires us to cache the inputs to the attention function. For a stack of N attention layers and fixed d, this reduces the peak memory usage from $\Omega\left(L_Q \cdot L_K \cdot N\right)$ to $O\left((L_Q + L_K) \cdot N + C \cdot L_K\right)$.

As described above, the combination of query chunking and input checkpointing provides a simple method for reducing the memory-footprint of vanilla attention, independent of top-k attention. Indeed, our benchmarking experiments in §3 demonstrate this. However, a drawback of this approach is that, during the backward pass, an implicit second forward pass is performed to re-compute the intermediate activations as described above. This can potentially increase the compute (FLOPs) required for a combined forward and backward pass by 50%. We now describe how to further improve both compute and memory by combining query chunking and input checkpointing with top-k attention.

Improving efficiency through top-k attention We now show that one can avoid re-computing activations in the case of $\text{top-}k\text{-}\mathrm{Attention}$ (Eq. 4). At a high level, $\text{top-}k(QK^\top)$ provides a highly compressed but accurate representation of $QK^\top$ and requires only $O(L_Q \cdot k)$ storage, compared to $\Omega(L_Q \cdot L_K)$ for $QK^\top$, where we assume $k \ll L_K$. Hence, we can cache it in addition to Q, K, V with-

```
 1  def forward (Q, K, V, k, activation):
 2      # Q: query chunk, K: keys, V: values            # [C, d], [L_K, d], [L_K, d]
 3      dots = matrix_prod(Q, transpose(K))             # [C, L_K]
 4      top_dots, top_indices = row_wise_topk(dots, k)  # [C, k], [C, k]
 5      del dots
 6      top_actv = activation(top_dots)                 # [C, k]
 7      actv = matrix(top_actv, top_indices)
 8      out = matrix_prod(actv, V)                      # [C, d]
 9      to_cache(Q, K, V, top_dots, top_indices, activation)
10      return out
11
12  def backward(d_out):
13      # d_out: grad of loss w.r.t. out                # [C, d]
14      Q, K, V, top_dots, top_indices, activation = from_cache()
15      d_top_actv = matrix_prod(d_out, transpose(V), out_indices=top_indices) # [C, k]
16      # did not cache top_actv so re-compute it to backpropagate
17      with compute_grads():
18          top_actv = activation(top_dots)             # [C, k]
19      d_top_dots = top_actv.backpropagate(d_top_actv)  # [C, k]
20      actv = matrix(top_actv, top_indices)
21      d_V = transpose(matrix_prod(transpose(d_out), actv))# [L_K, d]
22      del actv
23      d_dots = matrix(d_top_dots, top_indices)
24      d_Q = matrix_prod(d_dots, K)                    # [C, d]
25      d_K = transpose(matrix_prod(transpose(Q), d_dots))  # [L_K, d]
26      return d_Q, d_K, d_V
```

out incurring a significant increase in memory usage.

In the pseudo-code above, we show the forward and backward pass for top-k attention over a query chunk with input checkpointing. The steps of the forward pass that we re-compute during the backward pass is the application of activation on the output of top-$k(QK^\top)$ and forming its matrix representation for subsequent operations (compare Lines 6, 7 and Lines 18, 20). Therefore, the number of FLOPs spent on re-computation in our implementation of top-k attention is at most $O(L_Q \cdot (k + L_K) \cdot N)$, typically much lower than $\Omega(L_Q \cdot L_K \cdot d \cdot N)$, as there is no need to re-compute the dot-products.

Moreover, our benchmarking experiments (§3) show that top-k attention leads to improved memory usage compared to query chunking. This is because in vanilla attention (Eq. 1), while performing the re-computation in the backward pass of a query chunk $Q_\mathcal{C}$, we first re-compute $Q_\mathcal{C}K^\top$, then apply activation and backpropagate through this operation to compute $d(Q_\mathcal{C}K^\top)$. This implies that at this point there are three $C \times L_K$ matrices in memory. In top-k attention, activation is applied only on a $C \times k$ matrix and at any given time during the backward pass shown below there is at most one $C \times L_K$ matrix in memory: either `actv` (Line 20) or `d_dots` (Line 23). As our experiments show, this can lead to a much smaller memory footprint for small values of k, N.

3 Benchmarking

In this section, we benchmark top-k attention in terms of time and memory, and compare it to vanilla attention, query-chunking without the top-k operation, and to Performer (Choromanski et al., 2021), as a representative of state-of-the-art linear attention variants. We separately benchmark (a) a single self-attention layer over long sequences, (b) a single feed-forward layer with a large feed-forward dimension, and (c) a 12-layer Transformer decoder with same architecture as BERT-base (Devlin et al., 2019).

Experimental details For all models, we benchmark by running a forward and backward pass over random inputs. Each measurement is an average over 3 runs on an Nvidia A100 GPU and is discarded if memory usage exceeds 30GiB. We use causal masking for self-attention layers to highlight the simplicity of our approach that can seamlessly handle arbitrary attention masks, unlike other methods (Wang et al., 2020; Katharopoulos et al., 2020;

Choromanski et al., 2021), where implementing causal masking requires customized CUDA implementations. For Performer, we use 256 random features, and the CUDA implementation from (Katharopoulos et al., 2020).

Multi-head attention layer We benchmark a single multi-head attention layer over long sequences in a configuration similar to BERT-base: d_{model} is 768, 12 heads of size 64, and feed-forward dimension 3072. Fig. 1 shows the results when setting k to 128 and the query chunk size to 1024, which was shown to provide a good time-memory trade-off in §2.2.

We observe that top-k attention has the same device-memory usage as the Performer (top) for sequences as long as $65K$ tokens, while being as fast as vanilla attention, and even faster than Performer on inputs of length up to $4K$. With vanilla attention, we cannot fit even a single multi-head attention layer over a sequence of more than $10K$ tokens, while top-k uses less than 10GiB of memory over sequences of length $65K$. Lastly, we observe improvement in both time and memory when comparing top-k attention to query chunking over vanilla attention, where using top-k leads to a $3\times$ memory reduction for sequences of length $65K$.

Feed-forward layer While considerable effort has been dedicated to devising efficient models for long contexts, a large feed-forward dimension is useful for knowledge-intensive tasks such as open-domain QA (Roberts et al., 2020; Brown et al., 2020), and efforts have been made to reduce its complexity (Fedus et al., 2021). We benchmark the resource usage of top-k attention at a single feed-forward layer for different feed-forward dimensions using batch size 512 and input length 512, which results in 2^{18} queries per batch.

Top-k attention (Figure 3b), for $k = 512$ and query chunk size 2^{14}, dramatically improves device-memory usage compared to vanilla attention: it allowed us to use a feed-forward dimension $65K$ with 11GiB, while vanilla attention uses the same amount of memory with a feed-forward dimension $2K$. Fitting a linear curve to the memory usage of vanilla attention and top-k attention, we estimate that top-k attention can handle feed-forward dimension $205K$ compared to $7K$ for vanilla attention on a 30GiB machine. Moreover, comparing top-k attention to query chunking, we again observe a $3\times$ improvement in memory usage when the number of keys is $65K$. Lastly, we observe only a minor slowdown in top-k attention compared to vanilla attention.

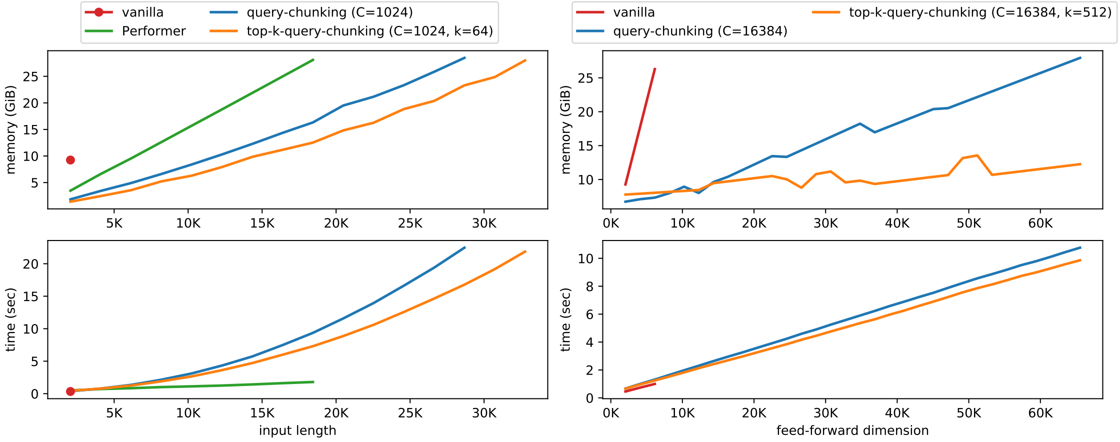
12-layer model We benchmark a 12-layer model to examine the cumulative utility of not caching $QK^\top$ in all N layers compared to the Performer. We use the same architecture as BERT-base with batch size 1 and vary the input length. We use a Transformer decoder with top-64 attention and chunk size $1,024$ at the self-attention layers, and simple query chunking with chunk size $4,096$ at the feed-forward layers.

We easily fit a $32K$-long input on a 30GiB GPU, improving memory consumption by more than $8\times$ compared to vanilla Transformer and $2\times$ compared to Performer. Moreover, top-k attention outperforms query chunking in terms of both memory and runtime. As top-k attention targets memory consumption but not runtime, a current limitation is that runtime, unlike Performer, is still quadratic. Thus, running multi-layer models on long sequences is reasonable in a fine-tuning or zero-shot inference setup, but further work is required for training from scratch 12-layer models over large datasets that contain long sequences.

Overall, our benchmarking results over multi-head attention, feed-forward, and multi-layer Transformer establish top-k attention as a strong baseline for future work on efficient Transformers that dramatically improves memory consumption. Next, we evaluate top-k attention on downstream tasks and show that top-k attention can be used as a drop-in replacement for vanilla attention without additional pre-training, which can allow resource-constrained research groups experiment with Transformers over long sequences or models with a large feed-forward dimension.

4 Experimental Evaluation of Top-k Attention

Having established top-k attention as a memory efficient alternative to vanilla attention, we now show that, even for small values of k, top-k attention provides a high-quality approximation of vanilla attention, both at the multi-head attention and feed-forward layers. We empirically show this in a wide range of setups including (a) training from scratch on tasks that require handling long-range dependencies (§4.1) and on language modeling (§4.2), (b) fine-tuning pre-trained language models (T5) on multiple QA datasets (§4.5), and (c) perform-



(a) 12-layer "base" Transformer decoder

(b) single feed-forward layer

Figure 3: Memory and time required for a combined forward and backward pass (details in §3).

ing zero-shot inference using pre-trained language models (UNIFIEDQA) without any training (§4.3).

4.1 Long Range Arena

Long Range Arena (Tay et al., 2021) is a recently established benchmark for evaluating the ability of Transformer variants to handle long sequences. It comprises of multiple text classification tasks with inputs containing thousands of tokens (Table 1). In ListOps (Nangia and Bowman, 2018), given a sequence of operations on single-digit integers, the model predicts a single-digit solution modeled as 10-way classification. IMDb movie reviews (Maas et al., 2011) is a character-level binary sentiment classification task. Lastly, in the ACL Anthology Network (AAN) (Radev et al., 2013) task, a character-level model classifies if there is a citation between a pair of papers.

For each task, we downloaded and directly used the vanilla Transformer code offered by the authors (Tay et al., 2021) and compared the performance before and after replacing the multi-head attention layers with top-128 attention, using identical hyperparameters for both cases (details in §A.1).[2]

Test accuracy measured at the training checkpoint with the highest accuracy on the development set is reported in Table 1 and the learning curves on the development and test sets are shown in Fig. 5. On IMDb and AAN, the performance of top-128 is comparable or better than vanilla attention. For ListOps, there is a minor drop in performance (1.5 points), but learning curves (Figure 5a) exhibit similar behaviour.

Thus, top-k attention, even for k as small as 3% of the number of keys, results in a performance very similar to that of vanilla attention. This shows that an exact and sparse top-k solution is a high-quality approximation for vanilla attention at multi-head attention layers.

	ListOps	IMDb	AAN	mean
input length	$2K$	$1K$	$4K$	
vanilla (reported in (Tay et al., 2021))	36.4	64.3	57.5	52.7
vanilla (our run)	**38.12**	63.66	57.93	53.2
top-128	36.56	**63.72**	**59.14**	53.1

Table 1: Test accuracy for vanilla and top-128 attention on Long Range Arena.

4.2 Language Modeling

We further ascertain the findings of §4.1 via language modeling on WikiText-103 (Merity et al., 2017) using a 6-layer Transformer decoder with 156M parameters. Using an input length of 1024, we trained two models with vanilla and top-64 attentions at the self-attention layers, obtaining test perplexity scores of 30.96 and 30.51 respectively, slightly better in case of top-64 (details in §A.3).

4.3 Zero-shot Inference with UNIFIEDQA

We have established that the performance of top-k attention is comparable to vanilla attention when training the model from scratch. In this set-up, several recently-proposed approaches have also reported competitive performances (Tay et al., 2021). Now, we consider a different and more practical setup, where the starting point is using an already pre-trained language model (Devlin et al., 2019; Raffel et al., 2020). As such models were trained using vanilla attention, replacing it with a new attention variant typically requires a corrective pre-training stage to allow the model weights to adjust

---

[2] `https://github.com/google-research/long-range-arena`

k	AI2 elem.	AI2 mid.	ARC easy	ARC chal.	BoolQ	CSQA	MCTest	NarQA	OBQA	RACE	SQuAD v1	SQuAD 2.0
64	82.9	77.6	79.3	71.6	88.7	72.4	92.8	30.5	77.4	83.5	86.1	82.0
128	87.8	81.6	83.5	72.2	89.7	74.8	**93.1**	31.6	80.0	85.8	86.8	84.3
256	91.1	**84.0**	85.3	75.6	90.0	75.9	92.8	32.1	**85.4**	86.8	86.8	85.7
512	91.1	84.0	85.4	75.9	90.4	77.0	93.1	**32.3**	83.6	87.2	86.9	86.2
1024	**91.9**	83.2	**86.3**	**76.3**	90.7	77.2	93.1	32.3	85.0	87.3	**87.0**	**86.3**
2048	91.9	82.4	85.8	75.9	**90.8**	77.2	93.1	32.3	85.0	87.3	87.0	86.3
4096	91.9	82.4	86.0	75.6	90.8	**77.6**	93.1	32.3	85.2	**87.4**	87.0	86.3
65536 (vanilla)	91.9	82.4	86.0	75.6	90.7	77.5	93.1	32.3	85.2	87.4	87.0	86.3

Table 2: Exact match scores of UNIFIEDQA on development sets with top-k attention at feed-forward layers. *Notation*: AI2 science elementary (AI2 elem.), AI2 science middle (AI2 mid.), ARC challenging (ARC chal.), CommonsenseQA (CSQA), NarrativeQA (NarQA), OpenbookQA (OBQA).

to the new variant, which can be expensive for large models. For example, (Gupta and Berant, 2021; Peng et al., 2021) have shown that using random features *without* corrective pre-training leads to high error rates in a language modeling task. Moreover, as explained in §2.1, most past methods are incompatible with feed-forward layers. In the subsequent experiments we show that it is possible to replace vanilla with top-k attention, at multi-head attention and feed-forward layers, and perform inference and fine-tuning without any need for such correction.

First, we compare the performance of UNIFIEDQA (Khashabi et al., 2020) before and after replacing its feed-forward layers with our implementation of top-k attention and directly performing inference on 12 different question answering (QA) datasets without any training. UNIFIEDQA is a T5-based (Raffel et al., 2020) model with 11B parameters (Raffel et al., 2020), fine-tuned on a weighted mixture of QA datasets. The 12 datasets include diverse domains, such as science questions, factoid questions over Wikipedia, commonsense questions, etc. Details regarding the datasets and metrics can be found in §A.2.

Table 2 shows the results for increasing values of k, where the feed-forward dimension of the model is $65,536$. We observe that already when $k = 256$ and $k = 512$, i.e., less than 1% of the number of keys, performance is comparable to vanilla Transformer. When $k = 4,096$ (6% of the number of keys), performance is equal or better than vanilla Transformer on all tasks. This highlights the plug-and-play property of top-k attention, which can be used without *any* additional training.

4.4 Zero-shot Inference with BERT

To verify that the plug-and-play property of top-k attention also holds at self-attention layers, we downloaded a BERT-large-uncased-whole-word-masking checkpoint (Devlin et al., 2019) already fine-tuned on SQuAD v1 (Rajpurkar et al., 2016) and evaluated its performance on the development set before and after replacing its self-attention layers with top-k attention. For k as low as 16 (4% of input length), we only saw a minor decrease in the exact match scores ($86.9 \rightarrow 86.2$). Moreover, to empirically verify that dense approximations of vanilla attention (Performer, RFA, etc) indeed require corrective pre-training, we repeated the measurement using Performer attention with 256 features, obtaining a score of 0.38.

4.5 T5 Finetuning

Having established the plug-and-play property of top-k attention in zero-shot inference (§4.3, §4.4), we now show the effectiveness of top-k attention when fine-tuning a model, and that there are no unforeseen issues stemming from training under high sprasity. Here, we use T5-base rather than T5-11B and evaluate on five QA datasets (and not 12) due to computational constraints.

Similar to §4.3, we replace the feed-forward layers of T5-base, which has feed-forward dimension 3072, with our implementation of top-256 attention and fine-tuned on multiple QA datasets. As summarized in Table 3, we found that the performance of top-256 attention was again comparable to vanilla attention on BoolQ, CommonsenseQA and ROPES with a minor loss in performance on MCTest ($81.2 \rightarrow 79.4$) and OpenbookQA ($58.8 \rightarrow 58.0$).

	BoolQ	CSQA	MCTest	OBQA	ROPES
T5-base	0.5	35.7	36.6	17.0	21.7
UNIFIEDQA-base	82.0	45.0	85.3	59.6	26.3
T5-base + finetuning	83.3	61.9	81.2	58.8	54.0
T5-base, top-256 + finetuning	83.1	62.0	79.4	58.0	53.8

Table 3: Exact match scores on development sets. "Finetuning" denotes model was finetuned on the dataset, else it was evaluated directly without any training. All models use vanilla feed-forward layers except the ones that say top-256 (§4.5).

To summarize, our experiments in §4.1-§4.5 demonstrated that the performance of vanilla attention and top-k attention is comparable at both multi-head attention (§4.1, §4.4) and feed-forward layers in multiple set-ups including training from scratch (§4.1, §4.2), fine-tuning (§4.5) and zero-shot inference (§4.3, §4.4), while dramatically improving memory usage, as shown in §3.

5 Discussion

Related work Our work follows a long line of works on efficient Transformers (see §1). Our method employs three main ideas: (a) computing the top-k attention scores for each query (b) grouping the queries into chunks and processing these sequentially (c) caching only a part of the activations for the backward pass. Top-k operation was used at self-attention layers by (Zhao et al., 2019) to show improved model performance, attributed to the removal of irrelevant information in the context. We use it to reduce the resource usage of multi-head attention and feed-forward layers. Processing query chunks sequentially was also used in Reformer (Kitaev et al., 2020) as activations are not cached. But in that case, by replacing vanilla residual connections in the Transformer with reversible connections (Gomez et al., 2017). Similar to the explanation provided in §2.2, these require an extra implicit forward pass during the backward pass and do not provide the compute and memory savings we get from our top-k specific backward pass (§2.2). Secondly, replacing residual connections with reversible ones changes the function computed by the model and would require corrective pre-training to be used with BERT, T5, etc (§4.3-§4.5).

Limitations and future work As our method requires computing inner products of all queries and keys, it has a quadratic compute requirement. As seen in our pseudo-code (§2.2), there are *four* matrix products (Lines 8, 15, 21, 25) involving a large sparse matrix and a small dense one. The results presented in the benchmarking section (§3) are based on our implementation that does not leverage this sparsity and hence is as slow as vanilla attention.

Leveraging sparsity of matrices We considered the option of performing matrix products involving large sparse matrices (Lines 8, 15, 21, 25 in our pseudo-code (§2.2)) by representing them in PyTorch's `torch.sparse_coo_tensor` format and using the `torch.sparse` framework to explicitly leverage their sparsity for saving compute. Unfortunately, the results were not encouraging even for $k = 1\%$ of number of keys (Figure 4). While future devices might allow faster sparse-dense products, in the immediate future, one can leverage block-sparse kernels (Child et al., 2019; Tillet et al., 2019) which have been successfully used for such products (Rasley et al., 2020).

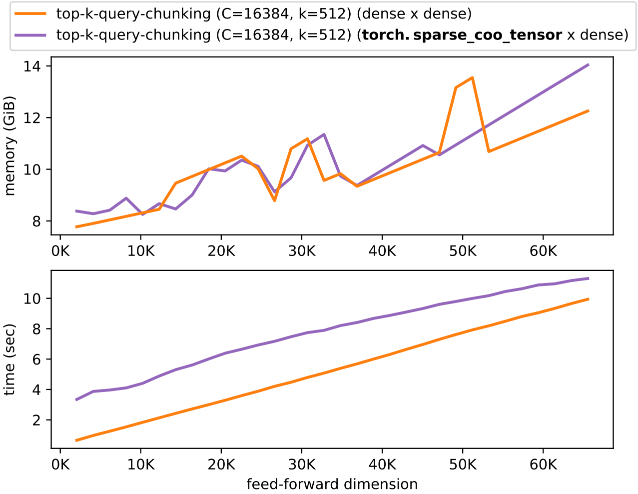


Figure 4: Memory and time required for a combined forward and backward pass on a single feed-forward layer for ***random*** inputs.

Conclusion In this work, we proposed a memory-efficient and accurate sparse approximation of the primary sub-layers of a Transformer, benchmarked the resulting resource savings, and verified its quality and unique advantages, on a wide range of downstream tasks and evaluation set-ups.

Acknowledgments

We thank Achille Fokoue, Maria Chang and Avi Sil for helpful discussions. Most of our experiments were conducted on IBM's Cognitive Computing Cluster with additional resources from Hybrid Cloud Infrastructure Research. This research was partially supported by (1) The Yandex Initiative for Machine Learning, and the European Research Council (ERC) under the European Union Horizons 2020 research and innovation programme (grant ERC DELPHI 802800), and (2) the IBM AI Residency program.

References

Irwan Bello. 2021. Lambdanetworks: Modeling long-range interactions without attention. In *International Conference on Learning Representations*.

Iz Beltagy, Matthew E Peters, and Arman Cohan.

2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

T. Chen, B. Xu, C. Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *ArXiv*, abs/1604.06174.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller. 2021. Rethinking attention with performers. In *International Conference on Learning Representations*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. 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 2924–2936. Association for Computational Linguistics.

Peter Clark. 2015. Elementary school science and math tests as a driver for AI: take the aristo challenge! In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 4019–4021. AAAI Press.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *ArXiv*, abs/1803.05457.

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. Association for Computational Linguistics.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 489–500. Association for Computational Linguistics.

W. Fedus, Barret Zoph, and Noam M. Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *ArXiv*, abs/2101.03961.

Mor Geva, R. Schuster, Jonathan Berant, and Omer Levy. 2020. Transformer feed-forward layers are key-value memories. *ArXiv*, abs/2012.14913.

Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. 2017. The reversible residual network: Backpropagation without storing activations. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 2214–2224.

Ankit Gupta and Jonathan Berant. 2020. GMAT: Global memory augmentation for transformers. *arXiv preprint arXiv:2006.03274*.

Ankit Gupta and Jonathan Berant. 2021. Value-aware approximate attention. *ArXiv*, abs/2103.09857.

John Hewitt and Christopher D. Manning. 2019. A structural probe for finding syntax in word representations. 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 4129–4138. Association for Computational Linguistics.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781. Association for Computational Linguistics.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 5156–5165. PMLR.

Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hannaneh Hajishirzi. 2020. UNIFIEDQA: Crossing format boundaries with a single QA system. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1896–1907. Association for Computational Linguistics.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. 2018. The NarrativeQA reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. RACE: Large-scale ReAding comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 785–794. Association for Computational Linguistics.

Kevin Lin, Oyvind Tafjord, Peter Clark, and Matt Gardner. 2019. Reasoning over paragraph effects in situations. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pages 58–62. Association for Computational Linguistics.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150. Association for Computational Linguistics.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391. Association for Computational Linguistics.

Nikita Nangia and Samuel Bowman. 2018. ListOps: A diagnostic dataset for latent tree learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 92–99. Association for Computational Linguistics.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah Smith, and Lingpeng Kong. 2021. Random feature attention. In *International Conference on Learning Representations*.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473. Association for Computational Linguistics.

Jiezhong Qiu, Hao Ma, Omer Levy, Wen-tau Yih, Sinong Wang, and Jie Tang. 2020. Blockwise self-attention for long document understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2555–2565. Association for Computational Linguistics.

Dragomir R. Radev, Pradeep Muthukrishnan, Vahed Qazvinian, and Amjad Abu-Jbara. 2013. The ACL anthology network corpus. *Language Resources and Evaluation*, 47:919–944.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392. Association for Computational Linguistics.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 3505–3506. ACM.

Matthew Richardson, Christopher J.C. Burges, and Erin Renshaw. 2013. MCTest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 193–203. Association for Computational Linguistics.

Adam Roberts, Colin Raffel, and Noam Shazeer. 2020. How much knowledge can you pack into the parameters of a language model? In *Proceedings of the*

2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 5418–5426. Association for Computational Linguistics.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. 2021. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68.

Noam M. Shazeer. 2020. GLU variants improve transformer. *ArXiv*, abs/2002.05202.

Sainbayar Sukhbaatar, Edouard Grave, Guillaume Lample, H. Jégou, and Armand Joulin. 2019. Augmenting self-attention with persistent memory. *ArXiv*, abs/1907.01470.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. 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 4149–4158. Association for Computational Linguistics.

Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. 2020a. Sparse sinkhorn attention. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 9438–9447. PMLR.

Yi Tay, M. Dehghani, Dara Bahri, and Donald Metzler. 2020b. Efficient transformers: A survey. *ArXiv*, abs/2009.06732.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2021. Long Range Arena : A benchmark for efficient transformers. In *International Conference on Learning Representations*.

Philippe Tillet, H. Kung, and D. Cox. 2019. Triton: an intermediate language and compiler for tiled neural network computations. *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

A. Vyas, A. Katharopoulos, and F. Fleuret. 2020. Fast transformers with clustered attention. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *ArXiv*, abs/2006.04768.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Zihao Ye, Qipeng Guo, Quan Gan, Xipeng Qiu, and Zheng Zhang. 2019. BP-Transformer: Modelling long-range context via binary partitioning. *arXiv preprint arXiv:1911.04070*.

Gilad Yehudai and Ohad Shamir. 2019. On the power and limitations of random features for understanding neural networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 6594–6604.

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. *Advances in Neural Information Processing Systems*, 33.

Guangxiang Zhao, Junyang Lin, Zhiyuan Zhang, Xuancheng Ren, Qi Su, and X. Sun. 2019. Explicit sparse transformer: Concentrated attention through explicit selection. *ArXiv*, abs/1912.11637.

A Supplemental Material

A.1 Details of Long Range Arena

ListOps (Nangia and Bowman, 2018) aims to diagnose the capability of modelling hierarchically structured data. Given a sequence of operations on single-digit integers, the model predicts the solution, also a single-digit integer modeled as a 10-way classification. Character-level text classification with the IMDb movie review dataset (Maas et al., 2011) is a binary sentiment classification task. In the character-level document retrieval with the ACL Anthology Network (AAN) (Radev et al., 2013), the model classifies if there is a citation between a pair of papers.

We used the code and pre-processed data provided by the authors of Long Range Arena (Tay et al., 2021) and default model configurations. For each task, we used identical hyperparameters for vanilla and top-k attentions (Table 4) and used at most two Nvidia A100 for each run.

Hyperparameter Notation BSZ: effective batch size, SQL: input sequence length, LR: learning rate, WRM: linear LR warm-up steps, STEP: number of gradient updates, EFQ: evaluated every these many steps, NL: number of layers in encoder/decoder, HS: hidden size, FF: feed-forward dimension, NH: number of heads, VOC: vocabulary size, DRP: dropout rate, CLIP: maximum gradient norm.

A.2 Details of UNIFIEDQA inference & T5 finetuning

We used Hugging Face's Transformers library (Wolf et al., 2019) for these experiments. Authors of UNIFIEDQA collected and pre-processed several QA datasets into a common format: "`QUESTION \n CHOICES \n CONTEXT`". We downloaded this data by following the instructions provided by the authors[3] and used it for the UNIFIEDQA inference experiments (§4.3). Some statistics are shown in Table 5. Longer inputs were truncated to 512 tokens.

Given an instance from the pre-processed data, we computed the exact match score of a prediction with respect to the list of provided answers via the SQuAD v1 evaluation script (Rajpurkar et al., 2016).

For the T5 experiments (§4.5), we used a slightly different input format. Given an instance in the UNIFIEDQA format, we formed the modified instance as "`question: QUESTION context: <yes> <no> <No Answer> CHOICES \n CONTEXT`".

A.3 Language Modeling on WikiText-103

WikiText-103 is a language modeling task based on English Wikipedia. We used the language modeling framework provided by Faiseq[4] and hyperparameters in Table 7. The details of Adam optimizer are β_1=0.9, β_2=0.98, weight-decay: 0.01, CLIP: none, LR schedule: inverse square root. During evaluation on test set, dataset is chunked into segments of length 1024 and perplexity is computed over each segment normally without access to other segments.

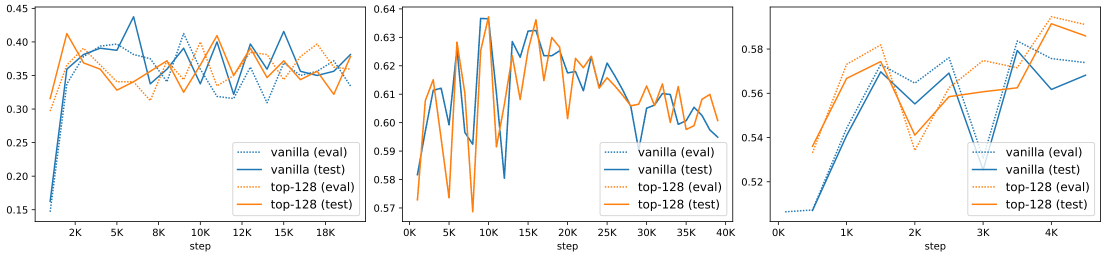
A.4 Benchmarking details

Benchmarking (§3) was done in PyTorch 1.8.1. For each run, we sampled a batch of random 32-bit input vectors and a backward pass was performed using the mean of the output elements as the loss. The part of code that was timed was enclosed within `torch.cuda.synchronize()` to ensure all CUDA threads finished. Memory usage was measured using `torch.cuda.max_memory_reserved()`. On Nvidia A100, any internal casting to TF32 was explicitly disabled.

---

[3] `https://github.com/allenai/unifiedqa`

[4] `https://github.com/pytorch/fairseq`



(a) ListOps (b) IMDb (c) AAN

Figure 5: Learning curves of vanilla and top-128 attention on Long Range Arena (§4.1).

task	BSZ	SQL	LR	WRM	STEP	EFQ	NL	HS	FF	NH
ListOps	16	2K	0.02	1K	20K	1K	6	512	2048	8
IMDb	32	1K	0.03	8K	40K	1K	4	256	1024	4
AAN	8	4K	0.05	8K	5K	500	4	128	512	4

Table 4: Hyperparameters for LRA tasks (§4.1). Other hyperparameters were used as provided at `https://github.com/google-research/long-range-arena`.

		training samples	eval samples	T5 tokens per sample (90th percentile)
CSQA	(Talmor et al., 2019)	9741	1221	54
OBQA	(Mihaylov et al., 2018)	4957	500	60
ARC easy	(Clark et al., 2018)	2251	570	90
AI2 elem.	(Clark, 2015)	623	123	96
ARC chal.	(Clark et al., 2018)	1119	299	101
BoolQ	(Clark et al., 2019)	9427	3270	253
SQuAD 2.0	(Rajpurkar et al., 2018)	130124	11873	294
SQuAD v1	(Rajpurkar et al., 2016)	87489	10570	295
ROPES	(Lin et al., 2019)	10924	1688	345
MCTest	(Richardson et al., 2013)	1480	320	435
RACE	(Lai et al., 2017)	87860	4887	568
NarQA	(Kočiský et al., 2018)	65494	6922	1278

Table 5: Statistics of the pre-processed datasets used in §4.3 and §4.5.

dataset	BSZ	LR	STEP	SQL	CLIP
CSQA	576	5e-5	3K	512	1
OBQA	512	5e-5	3K	512	1
BoolQ	80	5e-5	3K	512	1
MCTest	80	5e-5	3K	512	1
ROPES	80	5e-5	3K	512	1

Table 6: Hyperparameters for T5-base finetuning (§4.5). Trainings were performed on a single Nvidia V100 using Adam optimizer with a constant LR and evaluation was done only at end of training.

BSZ	SQL	LR	WRM	STEP	NL	HS	FF	NH	VOC	DRP
64	1024	5e-4	4K	50K	6	512	2048	8	267744	0.1

Table 7: Hyperparameters for WikiText-103 task (§4.2).