Linearizing Transformer with Key-Value Memory

Yizhe Zhang[*]
Meta AI [†]
yizhe.zhang@hotmail.com

Deng Cai[*]
The Chinese University of Hong Kong
thisisjcykcd@gmail.com

Abstract

Efficient transformer variants with linear time complexity have been developed to mitigate the quadratic computational overhead of the vanilla transformer. Among them are low-rank projection methods such as Linformer and kernel-based Transformers. Despite their unique merits, they usually suffer from a performance drop comparing with the vanilla transformer on many sequence generation tasks, and often fail to obtain computation gain when the generation is short. We propose MemSizer, an approach towards closing the performance gap while improving the efficiency even with short generation. It projects the source sequences into lower dimension representations like Linformer, while enjoying efficient recurrent-style incremental computation similar to kernel-based transformers. This yields linear computation time and constant memory complexity at inference time. MemSizer also employs a lightweight multi-head mechanism which renders the computation as light as a single-head model. We demonstrate that MemSizer provides an improved balance between efficiency and accuracy over the vanilla transformer and other efficient transformer variants in three typical sequence generation tasks, including machine translation, abstractive text summarization, and language modeling. Our code is released at https://github.com/jcyk/memsizer

1 Introduction

Transformer (Vaswani et al., 2017) has become the *de facto* standard for almost all NLP tasks across the board. At the core of the vanilla transformer is the attention mechanism that captures the interactions between feature vectors at different positions in a sequence. Despite its great success, the vanilla transformer models are typically computationally expensive as the computation of the attention mechanism scales quadratically with the sequence length. This bottleneck limits the efficient deployment of large-scale pre-trained models, such as GPT-3 (Brown et al., 2020), Image Transformer (Parmar et al., 2018), Codex (Chen et al., 2021) and DALL-E (Ramesh et al., 2021). Training and deploying such gigantic transformer models can be prohibitively difficult for scenarios with limited resource budgets and may result in huge energy consumption and greenhouse gas emission (Strubell et al., 2019; Schwartz et al., 2020).

A number of transformer variants have been proposed to reduce the computational overhead (Tay et al., 2020c). One family of methods leverages low-rank projections to reduce the number of pair-wise interactions (*i.e.*, the size of attention matrices) (Wang et al., 2020; Xiong et al., 2021; Tay et al., 2020a). These methods first project the input sequence into a low-resolution representation. For example, Wang et al. (2020) project the length dimension to a fixed feature dimension. Nevertheless, these methods have difficulties modeling variable-length sequences and autoregressive (causal) attention, impeding their applications in sequence generation tasks. Recent works propose to approximate the softmax attention through kernelization (Katharopoulos et al., 2020; Peng et al., 2021; Choromanski et al., 2021; Kasai et al., 2021). For sequence generation tasks, these works can cache computation in a *recurrent* manner, leading to constant memory complexity in sequence length during inference. Despite the improved efficiency in long-form generation, the computation gain of these kernel-based approaches vanishes when the generation is as short as a typical sentence length. Additionally, they usually suffer from a performance loss when training from scratch (Kasai et al., 2021).

In this work, we propose an approach called MemSizer, an efficient transformer variant which follows the paradigm of low-rank projections while

[*] Equal contribution.
[†] Currently at Apple

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 346–359
December 7-11, 2022

enjoying memory-efficient recurrent-style generation as the kernel-based transformers. Concretely, we develop a key-value memory layer (Sukhbaatar et al., 2015) to substitute the multi-head attention layer in the vanilla transformer. We pack the information in the source sequence into a fixed-sized set of memory values in a length-dynamic manner, and use input-independent parametric matrices as the memory keys. In this way, we emphasize more on modeling the values and significantly simplify the design of keys. This unbalanced design of keys and values further enables us to suppress the multi-head computation to be as fast as with single head.

MemSizer is conceptually simple yet can handle variable-length sequences and causal attention for generation thanks to the length-dynamic projection. With the unbalanced memory layer and dynamic projection, MemSizer enjoys linear time complexity and constant memory complexity. Our experiments in three typical sequence generation tasks (machine translation, abstractive text summarization, and language modeling) show that the proposed method achieves comparable or better performance to state-of-the-art linear recurrent transformers, with more substantial reductions in inference latency, memory consumption, and model size. The advantages are more prominent with longer input lengths. In some tasks, the proposed MemSizer can maintain or even improve the performance of the vanilla transformer, offering an appealing alternative for sequence generation tasks.

2 Preliminaries

2.1 Key-Value Memory Networks

We first review the general ideas of memory networks (Graves et al., 2014; Sukhbaatar et al., 2015). In a nutshell, given a set of *source* vectors $\mathbf{X}^s = \{\mathbf{x}_i^s\}_{i=1}^M$, a basic key-value memory network first projects the entire set into memory key vectors $\mathbf{K} \in \mathbb{R}^{M\times h}$ and value vectors $\mathbf{V} \in \mathbb{R}^{M\times h}$ respectively. A *target* vector $\mathbf{x}^t$ for querying the key-value memories will also be embedded as $\mathbf{q} \in \mathbb{R}^h$ which shares the same embedding space of $\mathbf{K}$. This is followed by computing a probability vector over the key vectors according to the inner product similarity:

$$\alpha = f(\mathbf{q}\mathbf{K}^T), \tag{1}$$

where f denotes an activation function. A typical choice for f is the softmax function. The output vector $\mathbf{x}^{\text{out}}$, which can be used for final prediction or next layer's input, is simply summarizing over the value vectors according to their probabilities:

$$\mathbf{x}^{\text{out}} = \alpha\mathbf{V}. \tag{2}$$

2.2 Transformer

Architecture The vanilla transformer architecture consists of multi-head attention, feedforward layers, and layer normalization modules (Vaswani et al., 2017). The multi-head attention module (referred to as standard attention, or SA, throughout this paper) plays a core role in a vanilla transformer. SA takes input as sequences of *source* and *target* vectors. The source vectors are used to produce *key* and *value* features, while the target vectors are mapped to *query* vectors. We denote the source and target vectors by $\mathbf{X}^s \in \mathbb{R}^{M\times d}$ and $\mathbf{X}^t \in \mathbb{R}^{N\times d}$, where d is the model dimensionality. The input vectors for each head are first mapped to h-dimensional *query*, *key*, and *value* features by learned affine transformations with $\mathbf{W}_* \in \mathbb{R}^{d\times h}$ and $\mathbf{b}_* \in \mathbb{R}^h$:

$$\mathbf{Q} = \mathbf{X}^t\mathbf{W}_q + \mathbf{b}_q, \quad \mathbf{K} = \mathbf{X}^s\mathbf{W}_k + \mathbf{b}_k, \tag{3}$$

$$\mathbf{V} = \mathbf{X}^s\mathbf{W}_v + \mathbf{b}_v. \tag{4}$$

The attention is achieved by computing the normalized similarities of query vector and key vectors:

$$\alpha = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{h}}). \tag{5}$$

The attention weights α are then used to calculate a weighted average of the value vectors as in eq (2). It is generally assumed there are r attention heads of h-dimensional such that $d = hr$. SA performs above procedure for each of the r heads in parallel and concatenates r output vectors to get the final d-dimensional vector:[1]

$$\mathbf{X}^{\text{out}} = [\mathbf{X}^{\text{out}}_{(1)}, \ldots, \mathbf{X}^{\text{out}}_{(r)}]W_o + b_o, \tag{6}$$

where $W_o \in \mathbb{R}^{d\times d}$ and $b_o \in \mathbb{R}^d$ are the output projection weights.

Time Complexity The computation in a transformer can be divided into three stages:

(i) FEATURE MAPPING: The time complexity of the computation of $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ for all r heads (Eq. (3-4)) is $\mathcal{O}(Nd^2)$, $\mathcal{O}(Md^2)$, and $\mathcal{O}(Md^2)$, respectively.

[1]The layer normalization (Ba et al., 2016) and residual connection (He et al., 2016) steps are suppressed for brevity.

(ii) ATTENTION: The time complexity of the computation of attention matrices for all r heads (Eq. (5)) is $\mathcal{O}(MNd)$, which scales quadratically in sequence length (M, N).

(iii) PROJECTION: The time complexity of projecting the concatenated $\mathbf{x}^{\text{out}}$ from r heads back to d-dimensional vector is $\mathcal{O}(Nd^2)$.

Taking all three parts together, a SA module scales at $\mathcal{O}(MNd + Md^2 + Nd^2)$. When sequence length is long ($M, N \gg d$), $\mathcal{O}(MNd)$ will dominate the computation.

Memory Complexity At every generation step, query, key, and value vectors consume space complexity of $\mathcal{O}(d)$, $\mathcal{O}(Md)$, and $\mathcal{O}(Md)$, respectively. Every step's attention weight (Eq. (5)) attends across M source positions, consuming $\mathcal{O}(Mr)$ space.

3 MemSizer: A Different Perspective of Attention Mechanism

As discussed in Section 2.2, the SA in the vanilla transformer can be perceived as an instantiation of the key-value memory network in Section 2.1, where the memory key $\mathbf{K}$ and value $\mathbf{V}$ are pointwise projections of the source $\mathbf{X}^s$. In this work, we replace the SA module with a different memory mechanism which achieves recurrent inference computation thus linear complexity. Our memory mechanism comes with a different specification of query, key and value in SA. Specifically, following Eq. (1-2), we specify the key-value memory layer as

$$\mathbf{Q} = \mathbf{X}^t, \quad \mathbf{K} = \mathbf{\Phi}, \tag{7}$$

$$\mathbf{V} = \text{LN}(\mathbf{W}_l(\mathbf{X}^s)^T)\text{LN}(\mathbf{X}^s\mathbf{W}_r). \tag{8}$$

Unbalanced Key-value Memory Mechanism The key-value memory layer in MemSizer contains k memory slots. Inspired by Tay et al. (2020a), which demonstrates that query-key attention can be significantly simplified in the vanilla transformer, the key matrix $\mathbf{\Phi} \in \mathbb{R}^{k\times d}$ in Eq. (7) is a learnable parametric matrix, which is input-independent and shared across different instances. The value matrix in MemSizer, likewise, contains k memory value vectors of d dimension. It summarizes the source information into a fixed-sized space $\mathbb{R}^{k\times d}$ regardless of the source length M. Compared with the vanilla Transformer which treats keys and values equally, this unbalanced key-value mechanism emphasizes learning better input-dependent values to match with input-independent keys.

Values Matrix via Dynamic-length Projection To pack the source information into the $\mathbb{R}^{k\times d}$ value matrix, Linformer (Wang et al., 2020) uses a low-rank projection. However, performing the low-rank projection would require the input sequence length M to be preset before training, making Linformer difficult to be applied to scenarios with dynamic input length and generation tasks. To solve this issue, we apply a linear kernel ($\mathbf{X}^T\mathbf{X}$) to the source input $\mathbf{X}^s$ to cancel out the length dimension M, so that M is not required to be preset. The value matrix essentially captures the second moment (covariance) information from the source $\mathbf{X}^s$ (El-Nouby et al., 2021; Zhu et al., 2021). We use two adaptor projection matrices $\mathbf{W}_l \in \mathbb{R}^{k\times d}$ and $\mathbf{W}_r \in \mathbb{R}^{d\times d}$ to project source information into k global token-independent memory value vectors[2]. The value matrix $\mathbf{V}$ is formulated in Eq. (8), where $\text{LN}(\cdot)$ denotes the layer normalization (Ba et al., 2016), which makes the training robust in our experiments. To control the magnitude of V across variable-length input sequences, we multiply the $\mathbf{V}$ by a scaling factor of $1/\sqrt{M}$, which resembles the rescaling rationale from SA in Eq. (5).

Lightweight Multi-head Computation The model can be made more expressive with multi-head specification, where we share $\mathbf{V}$ across r different heads but use a distinct $\mathbf{K}$ for each head. Following Lample et al. (2019), the outputs from each head are simply aggregated through mean-pooling. Specifically,

$$\mathbf{X}^{\text{out}} = 1/r \cdot \sum_{i=1}^{r} \mathbf{X}^{\text{out}}_{(i)}, \tag{9}$$

where $\mathbf{X}^{\text{out}}_{(i)}$ is the output from i-th head. The final output $\mathbf{X}$ has dimension d, therefore the output projection layer in the vanilla transformer is *no longer* needed.

In MemSizer, the above multi-head computation is *negligible*, as it can be done by first averaging the attention weights α in Eq. (1) from different heads into $\bar{\alpha}$, followed by as if performing single-head attention using $\bar{\alpha}$. The overall computation is as lightweight as a single-head model.

Recurrent Computation for Memory-efficient Generation Similar to previous kernel-based transformers (Kasai et al., 2021; Peng et al., 2021),

[2]Note that the inclusion of $\mathbf{W}_r$ does not affect the dimensionality of $\mathbf{V}$. However, in our experiments removing the $\mathbf{W}_r$ will harm the performance.

generation computation in MemSizer can be rolled out as a recurrent procedure. At each generation step i, define $\mathbf{V}_i$ as the recurrent states (Katharopoulos et al., 2020):

$$\mathbf{V}_i = \sum_{j=1}^{i} \mathrm{LN}(\mathbf{W}_l(\mathbf{x}_j^s)^T)\mathrm{LN}(\mathbf{x}_j^s\mathbf{W}_r), \quad (10)$$

where $\mathbf{x}_j^s$ is the j-th row of $\mathbf{X}^s$, $\mathbf{V}_i$ can be perceived as a rolling-sum matrix:

$$\mathbf{V}_i = \mathbf{V}_{i-1} + \mathrm{LN}(\mathbf{W}_l(\mathbf{x}_i^s)^T)\mathrm{LN}(\mathbf{x}_i^s\mathbf{W}_r). \quad (11)$$

Consequently, the output $\mathbf{x}_i^{\text{out}}$ can be computed in an incremental manner from cached recurrent matrix V_{i-1}. This avoids quadratic computation overhead in input sequence length.

Time Complexity We break down the time complexity of each step in MemSizer. MemSizer proceeds over two stages which correspond to the first two stages of SA. The last output projection stage in SA does not exist in MemSizer.

(i) MEMORY PROJECTION: To obtain the value matrix $\mathbf{V}$ (Eq. (8), shared over heads), we first compute $\mathbf{W}_l(\mathbf{X}^s)^T$ and $\mathbf{X}^s\mathbf{W}_r$, of which the time complexity is $\mathcal{O}(Mdk)$ and $\mathcal{O}(Md^2)$, respectively. The product of $\mathbf{W}_l(\mathbf{X}^s)^T$ and $\mathbf{X}^s\mathbf{W}_r$ further takes $\mathcal{O}(Mdk)$. In total, $\mathcal{O}(Md^2 + Mdk)$.

(ii) ATTENTION: The attention computation (Eq. (1)) is computed with $\mathcal{O}(Ndk)$.

Taking both parts together, the attention mechanism in MemSizer scales with $\mathcal{O}(Mdk + Md^2 + Ndk)$. Compared to $\mathcal{O}(MNd + Md^2 + Nd^2)$ of SA, we see that if the number of memory slots k is much smaller than sequence lengths ($k << M, N$), the change of time complexity from $\mathcal{O}(MNd)$ to $\mathcal{O}(Mdk) + \mathcal{O}(Ndk)$ brings a substantial speedup.

Memory Complexity MemSizer only needs to store the value matrix $\mathbf{V}$, and thus its space complexity is $\mathcal{O}(dk)$, constant in sequence length. This implies a reduction in memory footprints when $k << M$, compared to SA's $\mathcal{O}(Md)$.

Comparison with Other Transformers Compared with the vanilla transformer, each memory slot value $\mathbf{v}_{j\in\{1,\cdots,k\}}$ summarizes a *global position-agnostic* feature of the source context $\mathbf{X}^s$. MemSizer enjoys linear time complexity as Linformer and additionally possesses the advantage of recurrent-style sequence generation as kernel-based transformers. A detailed comparison among MemSizer, the vanilla transformer and other efficient transformers is in the Appendix B (Table 4).

4 Experiments

We present extensive experiments on three typical sequence generation tasks in NLP, including machine translation, abstractive text summarization, and language modeling.

4.1 Baselines

We compare MemSizer with previous transformer variants with *linear* time complexity and *constant* memory complexity in input sequence length, which limits the comparison to kernelization approaches (Katharopoulos et al., 2020; Peng et al., 2021; Choromanski et al., 2021; Kasai et al., 2021). Linformer assumes a fixed sequence length. This makes Linformer suit well with understanding tasks but difficult to be applied to generation tasks, as generation tasks typically assume variable generation length and autoregressive (causal) attention Likewise, Synthesizer needs to specify the maximum input length and thus does not suit well tasks with variable generation lengths. Thus Linformer and Synthesizer are excluded from the comparison. The compared methods correspond to three different feature maps ϕ: **ELU** ($\phi(\mathbf{x}) = \mathrm{elu}(\mathbf{x}) + 1$, Katharopoulos et al., 2020); **RFA** (random feature approximation with softmax temperature reparameterization, Peng et al., 2021; Katharopoulos et al., 2020); **T2R** (trainable random feature). **Performer** (Choromanski et al., 2021) employs a similar random approximation to RFA. We omitted it from the comparison as it diverges during training in our experiments. All models are randomly initialized via Xavier initialization (Glorot and Bengio, 2010).

4.2 Machine Translation

Setup We experiment with WMT16 En-De (4.5M train pairs, average target length 29.5 tokens), WMT14 En-Fr (36M, 31.7) and WMT17 Zh-En (20M, 28.5) translation benchmarks (Bojar et al., 2016). We follow the experiment setup, preprocessing and data splits by previous work (Kasai et al., 2021). Following Vaswani et al. (2017), we use the large-sized transformer with 6 layers, 16 attention heads, 1024 model dimensions, and 4096 hidden dimensions for both the encoder and decoder. We apply dropout with 0.3, weight decay with 0.01, and label smoothing with $\varepsilon = 0.1$. Following Ott et al. (2018), we use an increased batch size of approximately 460K tokens by accumulating gradients without updating parameters. Each

Model	k (cross, causal)		En-De	En-Fr	Zh-En	Speed	Memory	Model size
ELU	64	64	28.4	*	23.4	4605.6	9.842G	209M
RFA	32	4	28.1	41.7	23.4	3771.6	4.058G	210M
T2R	32	4	27.5	39.8	23.1	5408.4	4.057G	210M
MemSizer	32	4	**28.4**	**42.4**	**24.5**	**7476.3**	**3.896G**	**176M**
Transformer	–	–	28.9	42.2	24.2	5506.5	5.537G	209M

Table 1: Machine translation test results on MT datasets. The results for baselines are from Kasai et al. (2021). The vanilla transformer is implemented following Vaswani et al. (2017). (Vaswani et al. (2017) reports BLEU= 28.4 for En-De and 41.8 for En-Fr, which is worse than this implementation). "*" indicates divergence during training. The inference speed (Speed) measured in the number of tokens per second, peak memory usage (Memory), and model size are benchmarked on En-De translation task.

model is trained from random initialization for 30K (60K for the large En-Fr dataset) steps using Adam with a learning rate of $5 \cdot 10^{-4}$ and $\beta = (0.9, 0.98)$ (Kingma and Ba, 2015). We employ beam search decoding with beam size 5 and length penalty 1.0 (Wu et al., 2016). The checkpoints from the last five epochs are averaged to obtain the final model (Vaswani et al., 2017). Following previous works, we use tokenized BLEU (Papineni et al., 2002) for evaluation. Our method is applied to both cross and causal attention. Following Kasai et al. (2021), we use memory sizes $k = (32, 4)$ for cross and causal attention.

Results Table 1 presents machine translation results. In general, the kernel-based transformers suffer from additional overhead when the generated sequence is relatively short ($\sim$ 30 tokens in this task), leading to an incremental speedup compared with the vanilla transformer. ELU has a much larger feature size k, leading to increased memory overhead. With $\sim$ 17% smaller model size, MemSizer outperforms RFA and T2R while being comparable to ELU, in terms of test BLEU score in En-De. In En-Fr and Zh-En, MemSizer outperforms all baseline methods including the vanilla transformer.

As a result of significantly reduced model size, MemSizer achieves faster generation time and more efficient GPU memory utilization compared to other linear recurrent transformer variants.

4.3 Abstractive Text Summarization

Setup We evaluate on two popular datasets, namely CNN/DailyMail (Hermann et al., 2015) and XSUM (Narayan et al., 2018). We used the standard splits of (Nallapati et al., 2016) for training, validation, and testing (287,113/13,368/11,490 documents). The average lengths of articles and highlights are 766 and 53 respectively. The XSUM dataset (Narayan et al., 2018) consists of 227K (204,045/11,332/11,334 for training/validation/testing) BBC articles covering a wide variety of subjects. The average lengths of articles and summaries are 431 and 23 respectively.

We follow Lewis et al. (2020) for data preprocessing and model configuration. We use the BART-large configuration with 12 layers, 16 attention heads, 1024 model dimensions, and 4096 hidden dimensions for both the encoder and decoder. We apply dropout with 0.1, weight decay with 0.01, and label smoothing with $\varepsilon = 0.1$. Each model is trained from random initialization for 50K steps using Adam (Kingma and Ba, 2015). We employ beam search decoding with length penalty as in Lewis et al. (2020). We use the standard ROUGE metrics (F1 scores ROUGE-1/2/L) (Lin, 2004) for evaluation. Following the settings in machine translation, we use memory sizes $k = (32, 4)$ for cross and causal attention.

Results Table 2 presents abstractive text summarization results in ROUGE scores. MemSizer outperforms RFA and T2R on both datasets in terms of ROUGE scores.[3] On the XSUM dataset, MemSizer even achieves better results than the vanilla transformer while being much faster and memory-efficient. On the CNN/DailyMail dataset, however, there are still considerable performance gaps between MemSizer and the vanilla transformer. We attribute it to the distinct characteristics of the two datasets. XSUM contains highly abstractive summaries while the summaries in the CNN/DailyMail tend to be more extractive. In fact, the Lead-3 baseline (Zhang et al., 2020) outperforms all presented models. We hypothesize that MemSizer may suffer from the limited capacity of the reduced memory bank for memorizing the exact wordings in the source documents.

[3] We omitted the results of ELU because it diverged during training in our experiments.

Model	k		XSUM			CNN/DailyMail			Speed	Memory
	cross	casual	R1	R2	RL	R1	R2	RL		
Lead-3			16.3	1.6	12.0	40.4	17.6	36.7		
RFA	32	4	28.0	9.0	22.4	35.0	10.7	31.9	323.4	8.6G
T2R	32	4	28.6	9.3	22.8	35.8	11.2	32.7	358.3	6.2G
MemSizer	32	4	**32.3**	**11.6**	**25.8**	**36.3**	**12.1**	**33.1**	**412.3**	**5.9G**
Transformer	–	–	31.8	11.3	25.3	39.1	15.3	35.8	338.6	23.4G
Zhang et al. (2020)	–	–	30.8	10.8	24.4	38.3	15.0	35.5	-	-

Table 2: Summarization test results on XSUM and CNN/DailyMail datasets. The inference speed (Speed) measured in the number of tokens per second and peak memory usage (Memory) are benchmarked on XSUM dataset. The last row is from Zhang et al. (2020) with the same transformer architecture in our Transformer baseline.

Similar to machine translation, the kernel-based transformers suffer from additional overhead when the generated sequence is relatively short (~ 30 tokens for summaries), leading to an incremental speedup compared with the vanilla transformer. However, the reduction in peak memory consumption is substantial. This is because the lengthy input documents are packed into a fixed-sized key-value memory bank. Overall, MemSizer achieves the largest speed-up (22% speed-up compared to the vanilla transformer) and the smallest memory consumption (75% reduction compared to the vanilla transformer).

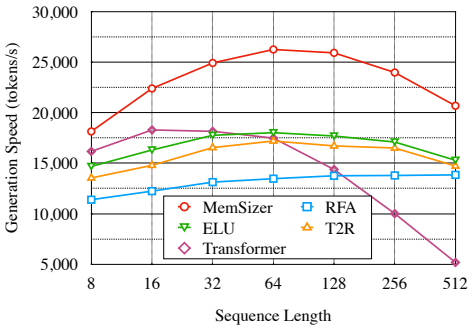
4.4 Language Modeling

Setup For the first task, we use the WikiText-103 language model (LM) benchmark, which consists of 103M tokens sampled from English Wikipedia (Merity et al., 2017). Following Kasai et al. (2021), we choose similar hyperparameters to prior work (Baevski and Auli, 2019; Fan et al., 2020): 32 layers, 8 heads, 128 head dimensions, 1024 model dimensions, 4096 fully connected dimensions and dropout (Srivastava et al., 2014) and layer dropout rates of 0.2. We set the memory size k to be 32. The word embedding and softmax matrices are tied (Press and Wolf, 2017; Inan et al., 2017). We partition the training data into non-overlapping blocks of 512 contiguous tokens and train the model to autoregressively predict each token (Baevski and Auli, 2019). Validation and test perplexities are measured by predicting the last 256 words out of the input of 512 consecutive words to avoid evaluating tokens in the beginning with limited context (*early token curse*, Press et al., 2021). We generally follow the optimization method from Baevski and Auli (2019), with a slight modification for some hyperparameters including learning rate (we use 10^{-4}), which shows better convergence. To evaluate the time and memory efficiency of MemSizer in sequence generation, we generate 256 tokens for each method. The batch size is set to be 256.

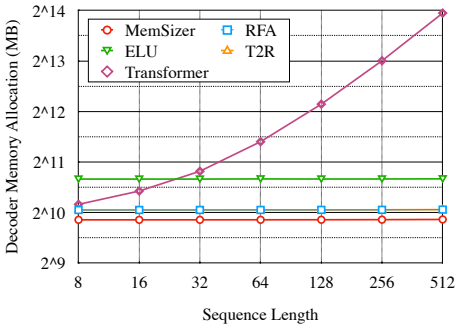
Model	k	PPL		Speed	Memory	Model Size
		dev.	test			
ELU	128	22.0	22.8	2491	6.825G	449M
RFA	32	20.4	21.3	2311	3.731G	449M
T2R	32	**20.1**	20.8	2692	3.733G	450M
MemSizer	32	20.2	**20.8**	**3165**	**3.373G**	**357M**
Transformer	–	17.9	18.5	1932	19.21G	448M

Table 3: WikiText-103 language modeling results in perplexity. The speed is measured for free text generation in the number of tokens per second. The top three rows are implementations from Kasai et al. (2021). The vanilla transformer is implemented according to Baevski and Auli (2019), which reports the test perplexity to be 18.7 (worse than our 18.5 result).

Results Table 3 presents the language modeling results in perplexity and computation cost. We observe that MemSizer outperforms ELU and RFA, and achieves comparable performance to T2R, suggesting that a similar level of performance to the state-of-the-art kernel-based transformer can be obtained without approximating the softmax attention in the vanilla transformer. The generation time, memory usage, and model size are significantly reduced in MemSizer. We attribute this reduction to the fact that MemSizer: i) uses fewer parameters in feature mapping as it projects the input into a much lower dimension k; ii) does not have the output projection layer; iii) suppresses the computation of intermediate state for feature mapping required in kernel-based transformers. There remains a gap of 2.3 perplexity points between the MemSizer and transformer models, which might be reduced by leveraging a swap-then-finetune approach similar to Kasai et al. (2021). Further improvement of the MemSizer is left for future work. Compared with the results from machine translation and abstractive text summarization, we hypothesize that Mem-

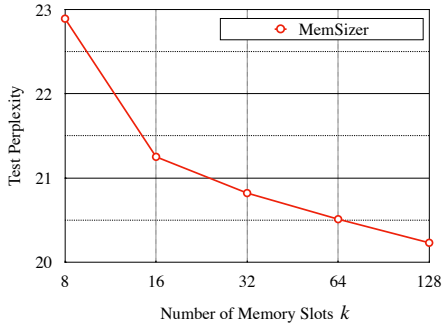


(a) Generation speed.

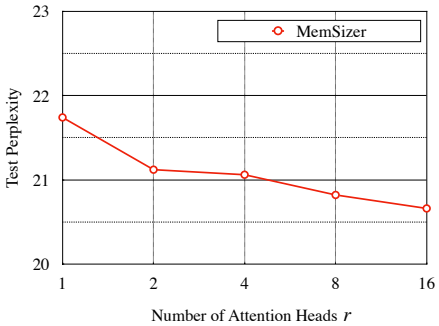


(b) Peak memory consumption.

Figure 1: Computational overhead of machine translation (En-Dn) of different sequence lengths.



(a) Effect of different numbers of memory slots k.



(b) Effect of different numbers of attention heads r.

Figure 2: Language model (Wikitext-103) perplexities of different model configurations.

Sizer is more advantageous with cross-attention in encoder-decoder architectures.

4.5 Analysis of MemSizer

Computational Overhead vs. Sequence Length As discussed, MemSizer is a linear and recurrent model for sequence generation tasks. To evaluate the time and memory efficiency against length, we run a set of experiments with different sequence lengths. For simplicity, we assume the source length is equal to the target length in our experiments (Kasai et al., 2021). Figure 1a and 1b show the time and memory cost results of MT (En-De) models in Table 1. All models are tested using greedy decoding with the same batch size of 256 on the same NVIDIA A100 GPU. As shown in figure 1a, we observe that MemSizer can generate a nearly-constant number of tokens per second regardless of the sequence length, dramatically outpacing the vanilla transformer model in longer sequence generation (300% speedup when the length becomes 512). MemSizer also outperforms other linear recurrent variants by large margins (35% faster than ELU for 512-length sequences). The maximum speedup compared with other linear recurrent variants is achieved at length=64. Figure 1b plots decoder memory consumption when running the generation with different lengths. The curves show that the peak memory consumption is almost a constant over varying sequence lengths and is consistently lower than other baselines. This reveals the potential of MemSizer to achieve even more significant speed gains by allowing for a larger batch size thanks to its lower memory consumption.

Number of Memory Slots Next, we study the effect of the number of memory slots k. Figure 2a compares the test perplexities using different values of k on the WikiText-103 language model task. We observe that the performance gets better as k goes larger. Among the values of k in Figure 2a, we do not observe that the number of memory slots k has a considerable impact on inference time and memory cost. Presumably, as shown in Section 3, as k is generally much smaller than the model dimension d, a larger k does not slow down the inference. However, during training time, processing time per token is roughly linear to k, presumably because more intermediate states need to be stored for back-propagation.

Number of Attention Heads We also investigate the impact of the number of attention heads

on model performance. Figure 2b shows the results with varying values of r on the WikiText-103 language model task. As can be seen, the number of attention heads slightly affects the test perplexity, resulting in slightly better performance with more attention heads. No significant difference in training and inference overhead is observed, as the multi-head computation is lightweight in MemSizer (*e.g.*, setting $r = 16$ only introduces 4.5 % more parameters and GPU memory than $r = 1$).

MemSizer with alternative design of Keys K We further experiment with freezing the keys $\mathbf{K}$ with random standard Xavier initialization and let the input $\mathbf{q}$ adapt to these keys. In both language model and machine translation tasks, the performance dropped by a relatively small margin (See Appendix C, Table 5), indicating learning $\mathbf{K}$ is less essential comparing to learning $\mathbf{V}$. Another evidence of this is that we also performed experiments to model $\mathbf{K}$ in the same input-dependent manner as $\mathbf{V}$, which failed to yield performance gains.

5 Related Work

Transformers with Memory Mechanism Previous work investigated injecting a memory mechanism into transformers. Burtsev et al. (2020) augmented Transformer by adding memory tokens to store non-local representation. Lample et al. (2019) used a product key memory layer to substitute the feed-forward layer in Transformer. Fan et al. (2021) used a KNN-based information fetching module to enable Transformer access to external knowledge. Our approach is fundamentally different from them as we replace the standard attention (SA) with a key-value memory layer, which leads to linear complexity and recurrent computation.

Recurrent Transformers Previous work proposed several recurrent transformers focusing on approximating the softmax attention kernel between $\mathbf{q}$ and $\mathbf{k}$ by projecting them via feature map function $\phi(\cdot)$. These recurrent variants scale at the linear time and constant space complexity in sequence length. Katharopoulos et al. (2020) proposed $\phi(\mathbf{x}) = \mathrm{elu}(\mathbf{x}) + 1$ and applied it to image generation. In language modeling and machine translation tasks, RFA (Peng et al., 2021) and Performer (Choromanski et al., 2021) used random features that approximate the softmax attention via Monte Carlo sampling (Rahimi and Recht, 2007; Yu et al., 2016). T2R (Kasai et al., 2021) used trainable feature mapping which allows smaller feature size thus further improving the efficiency. Schlag et al. (2021) connects kernel-based transformers with previous Fast Weight Programmers. However, approximating softmax typically needs additional steps to obtain intermediate feature mapping results. Instead of approximating the self-attention softmax kernel, MemSizer employs a key-value memory module, which suppresses these intermediate steps. The output projection step in SA is also omitted in this key-value memory module, yielding further computation and memory savings.

Other Efficient Transformers One family of efficient transformers limited the receptive fields that are attended to by sparsifying the attention patterns. Some works introduced fixed patterns of blockwise attention (Qiu et al., 2020) and strided attention (Child et al., 2019; Beltagy et al., 2020; Zaheer et al., 2020). (Sukhbaatar et al., 2019) learned sparse attention patterns in a data-driven manner. These sparse local attention approaches reduced the computation at a cost of potentially harming the modeling capacity. Another family of efficient transformers compresses the context via low-rank projections to reduce memory overhead (Wang et al., 2020; Tay et al., 2020a). Other methods add "global tokens" as surrogates for global information exchange (Rae et al., 2020; Ma et al., 2021) or employ clustering-based attention (Kitaev et al., 2020; Roy et al., 2020; Tay et al., 2020b). We compared MemSizer in detail with some of these efficient Transformers in the Appendix B.

Prior work also suggested many other strategies to improve efficiency in transformers, such as factorization (Dehghani et al., 2019; Lan et al., 2020), pruning (Michel et al., 2019; Fan et al., 2020), and quantization (Zafrir et al., 2019; Shen et al., 2020). Some of these methods present orthogonal design choices and can be integrated into our MemSizer model to gain further efficiency.

6 Conclusion

We present MemSizer, a method that leverages a novel key-value memory network specification to accelerate the original self-attention module. MemSizer compresses source information to a set of global memory entries and uses an unbalanced key-value mechanism which further leads to lightweight multi-head computation. MemSizer advances recent recurrent transformers with kernel approximation with lower time, memory, and storage cost

during generation. Our experiments in three standard generation tasks demonstrate that our model achieves an improved balance between efficiency and accuracy. The proposed method can be stacked with other computation reduction techniques to further advance the efficiency of transformers.

Limitations

This work has several limitations. First, there is still a performance gap between our method and the vanilla transformer in the language modeling task and CNN/Daily summarization task. We expect this can be closed by leveraging a swap-then-finetune procedure similar to (Kasai et al., 2021). We left it for future work as we focus on closing the gap by training from scratch in this paper. It would also be interesting to make the attention sparse so that fewer memory slots are attended to further reduce the training and generation computation. We also note that the feedforward layer still takes a lot of computation, which can be further reduced by unifying the self-attention layer with the feedforward layer with memory network framework.

Broader Impact

This work focuses on improving the natural language processing (NLP) and general artificial intelligence (AI) research community. Our work can be leveraged to improve natural language generation (NLG) models, including but not limited to text editing, conversational agents and question answering systems. The **broader impact** the **risks** of this work are summarized as following:

• This work can facilitate research in the NLG tasks in a generic manner, to potentially accelerate generations in applications like machine translation, text summarization, and virtual assistants.

• This work is a fundamental research work that focuses on the technical improvement, thus we have NOT imposed additional aggressive filtering techniques to the text data we used, beyond what has been performed to the original dataset from their sources. The text data we used may have offensiveness/toxicity/fairness/bias issues that we have not been able to identify, as those are not the focus of this work.

• Given the above potential risks, due to the nature of natural language generative models, we note that the generations or outputs of this work, though not likely, may reflect gender and other historical biases in the data. Under rare circumstances, the generations may exhibit a mild extent of unethical, biased, or offensive attitudes. These are known issues with current state-of-the-art text generation models. We would hope that a faster generation system like what we present can enable more iterations of further mitigation strategies for inappropriate and hallucinated generations.

• This work aims to advance AI technology in an environmental-friendly manner. Our proposed method can potentially reduce the carbon footprints produced by AI models.

References

Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization.

Alexei Baevski and Michael Auli. 2019. Adaptive input representations for neural language modeling. In *Proc. of ICLR*.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurélie Névéol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. Findings of the 2016 conference on machine translation. In *Proc. of WMT*.

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 *Proc. of NeurIPS*.

Mikhail S Burtsev, Yuri Kuratov, Anton Peganov, and Grigory V Sapunov. 2020. Memory transformer. *arXiv preprint arXiv:2006.11527*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. 2021. Rethinking attention with performers. In *Proc. of ICLR*.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *Proc. of ICLR*.

Alaaeldin El-Nouby, Hugo Touvron, Mathilde Caron, Piotr Bojanowski, Matthijs Douze, Armand Joulin, Ivan Laptev, Natalia Neverova, Gabriel Synnaeve, Jakob Verbeek, et al. 2021. Xcit: Cross-covariance image transformers. In *Advances in Neural Information Processing Systems*.

Angela Fan, Claire Gardent, Chloé Braud, and Antoine Bordes. 2021. Augmenting transformers with knn-based composite memory for dialog. *Transactions of the Association for Computational Linguistics*, 9:82–99.

Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. In *Proc. of ICLR*.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proc. of CVPR*.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28.

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2017. Tying word vectors and word classifiers: A loss framework for language modeling. In *Proc. of ICLR*.

Jungo Kasai, Hao Peng, Yizhe Zhang, Dani Yogatama, Gabriel Ilharco, Nikolaos Pappas, Yi Mao, Weizhu Chen, and Noah A Smith. 2021. Finetuning pretrained transformers into rnns. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10630–10643.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proc. of ICML*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proc. of ICLR*.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. In *Proc. of ICLR*.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. 2019. Large memory layers with product keys. In *NeurIPS*, volume 32.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *Proc. of ICLR*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proc. of ACL*.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Xuezhe Ma, Xiang Kong, Sinong Wang, Chunting Zhou, Jonathan May, Hao Ma, and Luke Zettlemoyer. 2021. Luna: Linear unified nested attention. *Advances in Neural Information Processing Systems*, 34.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *Proc. of ICLR*.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Proc. of NeurIPS*.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Gulçehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany. Association for Computational Linguistics.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807, Brussels, Belgium. Association for Computational Linguistics.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. In *Proc. of WMT*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proc. of ACL*.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. 2018. Image transformer. In *Proc. of ICML*.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A. Smith, and Lingpeng Kong. 2021. Random feature attention. In *Proc. of ICLR*.

Ofir Press, Noah A. Smith, and Mike Lewis. 2021. Shortformer: Better language modeling using shorter inputs.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proc. of EACL*.

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, Online. Association for Computational Linguistics.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. 2020. Compressive transformers for long-range sequence modelling. In *Proc. of ICLR*.

Ali Rahimi and Benjamin Recht. 2007. Random features for large-scale kernel machines. In *Proc. of NeurIPS*.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. 2020. Efficient content-based sparse attention with routing transformers. *TACL*.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. 2021. Linear transformers are secretly fast weight programmers. In *International Conference on Machine Learning*, pages 9355–9366. PMLR.

Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. 2020. Green ai. *Communications of the ACM*, 63(12):54–63.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. Q-BERT: hessian based ultra low precision quantization of BERT. In *Proc. of AAAI*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*.

Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. 2019. Adaptive attention span in transformers. In *Proc. of ACL*.

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. 2015. End-to-end memory networks. In *NeurIPS*.

Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. 2020a. Synthesizer: Rethinking self-attention in transformer models.

Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. 2021. Synthesizer: Rethinking self-attention for transformer models. In *International Conference on Machine Learning*, pages 10183–10192. PMLR.

Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. 2020b. Sparse sinkhorn attention. In *Proc of ICML*.

Yi Tay, M. Dehghani, Dara Bahri, and Donald Metzler. 2020c. Efficient Transformers: A survey.

Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. 2021. Mlp-mixer: An all-mlp architecture for vision. *Advances in Neural Information Processing Systems*, 34.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proc. of NeurIPS*.

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

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. 2021. Nyströmformer: A nyström-based algorithm for approximating self-attention. In *Proc. of AAAI*.

Felix Xinnan X Yu, Ananda Theertha Suresh, Krzysztof M Choromanski, Daniel N Holtmann-Rice, and Sanjiv Kumar. 2016. Orthogonal random features. In *Proc. of NeurIPS*.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8BERT: quantized 8bit BERT. In *Proc. of EMC*^{2}.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. 2020. Big Bird: Transformers for longer sequences. In *Proc. of NeurIPS*.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. 2020. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International Conference on Machine Learning*, pages 11328–11339. PMLR.

Chen Zhu, Wei Ping, Chaowei Xiao, Mohammad Shoeybi, Tom Goldstein, Anima Anandkumar, and Bryan Catanzaro. 2021. Long-short transformer: Efficient transformers for language and vision. In *NeurIPS*, volume 34.

Appendix for Linearizing Transformer with Key-Value Memory

A Illustration of MemSizer

We provide an illustrative figure of Memsizer in Figure 3. Details are provided in the main text.

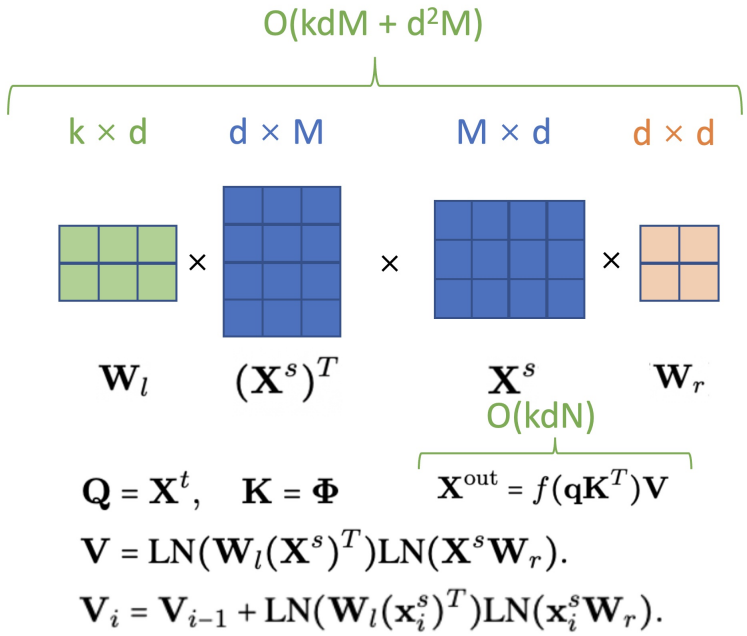


Figure 3: Illustration of the recurrent computation of Memsizer. "LN" represents the Layer Normalization.

B Detailed Comparison with Other Transformers

	$\mathbf{Q}$	$\mathbf{K}$	$\mathbf{V}$	Linear	Recurrent	M-Agnostic
Transformer	$\mathbf{X}^t\mathbf{W}_q$	$\mathbf{X}^s\mathbf{W}_k$	$\mathbf{X}^s\mathbf{W}_v$			
Dim.	$N \times h$	$M \times h$	$M \times h$	×	×	✓
Synthesizer (R)	$\mathbf{I}$	$\mathbf{\Phi}$	$\mathbf{X}^s\mathbf{W}_v$			
Dim.	$N \times N$	$M \times N$	$M \times h$	×	×	×
Synthesizer (D)	$\mathbf{X}^t$	$\mathbf{\Phi}$	$\mathbf{X}^s\mathbf{W}_v$			
Dim.	$N \times d$	$M \times d$	$M \times h$	×	×	×
Linformer	$\mathbf{X}^t\mathbf{W}_q$	$\mathbf{W}_e\mathbf{X}^s\mathbf{W}_k$	$\mathbf{W}_f\mathbf{X}^s\mathbf{W}_v$			
Dim.	$N \times h$	$k \times h$	$k \times h$	✓	×	×
RFA/Performer	$\phi(\mathbf{X}^t\mathbf{W}_q)$	$\phi(\mathbf{X}^s\mathbf{W}_k)$	$\mathbf{X}^s\mathbf{W}_v$			
Dim.	$N \times k$	$M \times k$	$M \times h$	✓	✓	✓
Ours	$\mathbf{X}^t$	$\mathbf{\Phi}$	$\mathbf{W}_l(\mathbf{X}^s)^T\mathbf{X}^s\mathbf{W}_r$			
Dim.	$N \times d$	$k \times d$	$k \times d$	✓	✓	✓

Table 4: A high-level comparison of attention mechanism perspectives in different transformer variants, including Synthesizer (Tay et al., 2021) random/dense (R/D), Linformer (Wang et al., 2020) and Performer (Choromanski et al., 2021). Details are removed for brevity. "M-Agnostic" indicates the maximum source length M is *not* required to be preset.

Comparison with Vanilla Transformer Compared with SA in the vanilla transformer, the number of memory slots k in MemSizer is independent of the source sequence length M and can be arbitrarily configured to balance between performance and efficiency. Also, we only pack the source information

$\mathbf{X}^s$ into $\mathbf{V}$. Note that each row of $\mathbf{V}$ ($\mathbf{v}_{j\in\{1,\cdots,M\}}$) in the vanilla transformer corresponds to one input dimension out of the total length M, in a point-wise manner. However, in MemSizer, each memory slot value $\mathbf{v}_{j\in\{1,\cdots,k\}}$ summarizes a *global position-agnostic* feature of the source context $\mathbf{X}^s$. Vanilla transformer is not linear and not recurrent.

Comparison with Linformer MemSizer operates with the original $\mathbf{X}^t$ rather than the projection of $\mathbf{X}^t$ in Linformer. The key $\mathbf{K}$ in MemSizer does not contain source information. The projection matrices $\mathbf{W}_l$ and $\mathbf{W}_r$ do not depend on source dimension M, which allows dynamic input length thus facilitating generation. In contrast, the projection matrices W_e and W_f is a $k \times M$ matrix. Linformer is linear but not recurrent.

Comparison with Synthesizer/MLP-Mixer MemSizer also share similarities with Synthesizer (Tay et al., 2021). MLP-Mixer (Tolstikhin et al., 2021) is computationally comparable to Synthesizer (random) except that in MLP-mixer the f is an identity function. As show in Table 4, MemSizer becomes akin to Synthesizer (dense) if the $\mathbf{V}$ is computed by an MLP $\mathbf{X}^s$ ($\mathbf{V} = \mathbf{X}^s\mathbf{W}_v + \mathbf{b}_v \in \mathbb{R}^{M\times h}$). However, Synthesizer attends to M different token and MemSizer attends on k different memory slots in memory. Consequently, Synthesizer scales quadratically with input length while MemSizer scales linearly. As the maximum sequence length needs to be preset when initializing the weights, it is not straightforward to apply Synthesizer to generation tasks with various input lengths. Synthesizer is not linear and not recurrent.

C MemSizer with fixed Keys K

Inspired by the "random" version of Synthesizer (Tay et al., 2020a), we further experiment with fixing the keys $\mathbf{K}$ and let the input $\mathbf{q}$ adapt to these keys. Specifically, we initialize $\mathbf{K}$ for each layer and each head with standard Xavier initialization and freeze them during the training process. In both language model and machine translation tasks, the performance dropped by a relatively small margin (Table 5). Presumably, as $k \ll d$, the keys in $\mathbf{K}$ are almost orthogonal with Xavier initialization, thus less likely to "collide" with each other (Schlag et al., 2021). Therefore, updating $\mathbf{K}$ becomes less essential comparing to other parts of the model.

	LM (PPL) ↓	MT (BLEU) ↑
$\mathbf{K}$ Trainable	**20.8**	**28.4**
$\mathbf{K}$ fixed	21.3	27.8

Table 5: Fixing $\mathbf{K}$ results in performance decrease.