Recurrent Attention Networks for Long-text Modeling

Xianming Li[1†], **Zongxi Li**[2†*], **Xiaotian Luo**[1], **Haoran Xie**[3], **Xing Lee**[1], **Yingbin Zhao**[1], **Fu Lee Wang**[2], **Qing Li**[4]
[1] Ant Group, Shanghai, China
[2] School of Science and Technology, Hong Kong Metropolitan University, Hong Kong SAR
[3] Department of Computing and Decision Sciences, Lingnan University, Hong Kong SAR
[4] Department of Computing, Hong Kong Polytechnic University, Hong Kong SAR
{niming.lxm,lxt267638,lx250976,zyb166123}@antgroup.com
{zoli, pwang}@hkmu.edu.hk, hrxie@ln.edu.hk, qing-prof.li@polyu.edu.hk

Abstract

Self-attention-based models have achieved remarkable progress in short-text mining. However, the quadratic computational complexities restrict their application in long text processing. Prior works have adopted the chunking strategy to divide long documents into chunks and stack a self-attention backbone with the recurrent structure to extract semantic representation. Such an approach disables parallelization of the attention mechanism, significantly increasing the training cost and raising hardware requirements. Revisiting the self-attention mechanism and the recurrent structure, this paper proposes a novel long-document encoding model, Recurrent Attention Network (RAN), to enable the recurrent operation of self-attention. Combining the advantages from both sides, the well-designed RAN is capable of extracting global semantics in both token-level and document-level representations, making it inherently compatible with both sequential and classification tasks, respectively. Furthermore, RAN is computationally scalable as it supports parallelization on long document processing. Extensive experiments demonstrate the long-text encoding ability of the proposed RAN model on both classification and sequential tasks, showing its potential for a wide range of applications.

1 Introduction

Recently, self-attention-based neural networks, such as Transformer (Vaswani et al., 2017), GPT (Radford et al., 2018, 2019; Brown et al., 2020), and BERT family (Devlin et al., 2019; Liu et al., 2019; Lan et al., 2020), have demonstrated superior text encoding ability in many natural language processing (NLP) tasks with the help of large-scale pretraining. These models have set state-of-the-art benchmarks in classification tasks like text categorization (Li et al., 2021a) and sentiment analysis (Naseem et al., 2020; Li et al., 2021c, 2023), and sequential tasks like question answering (Lee et al., 2019; Karpukhin et al., 2020) and information extraction (Li et al., 2021b; Wu et al., 2022). The time and space complexities of self-attention computation are $O(n^2)$ with respect to the sequence length, making it computationally expensive to encode long texts. Therefore, BERT models adopt an absolute positional encoding strategy to manage computational overhead. However, such a setting makes the BERT models unable to handle texts longer than 512 tokens, restricting their application in realistic scenarios like processing user comments, news articles, scientific reports, and legal documents with arbitrary lengths.

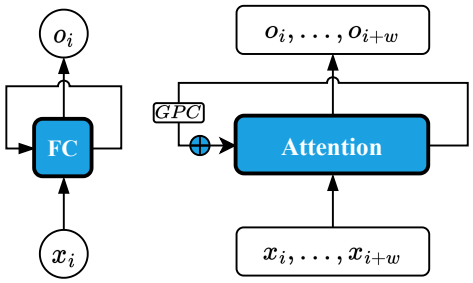
Current works focus on two solutions to enable self-attention-based models for handling longer texts. The first solution reduces the computing complexity of self-attention from quadratic to linear by approximating its softmax operation (Beltagy et al., 2020; Choromanski et al., 2021; Hua et al., 2022). These models can handle relatively long texts within the hardware capacity but also suffer from a performance drop (Schlag et al., 2021; Hutchins et al., 2022). Another solution is to divide the long document into chunks shorter than 512 tokens so that pretrained BERT models can be applied (Pappagari et al., 2019; Hutchins et al., 2022). However, as the chunks are individually encoded, the resulted representations do not contain the crucial contextual information for sequential tasks. While a special recurrent mechanism can handle sequential tasks (Hutchins et al., 2022), it cannot produce a document-level representation for classification, limiting their generality as none works for both classification and sequential tasks. Additionally, introducing recurrent modules disables the parallel computing feature, leading to unscalable implementation.

To address the aforementioned issues, this paper proposes the **R**ecurrent **A**ttention **N**etwork (RAN)[1],

*Corresponding author; † Equal contribution.

[1]The code is available at https://github.com/4AI/RAN.

Findings of the Association for Computational Linguistics: ACL 2023, pages 3006–3019
July 9-14, 2023



(a) RNN (b) RAN

Figure 1: The architectures of (a) RNN and (b) RAN. The basic encoding unit of RNN is the fully-connected (FC) layer, and that of RAN is the self-attention layer. RNN is a token-level recurrent architecture, while the proposed RAN is a window-level recurrent model.

a novel model architecture supporting recurrent self-attention operation over long sequences, enabling global dependency extraction and long-term memory. RAN iterates through the sequence by non-overlapping windows. Unlike token-level recurrent architectures such as LSTM (Hochreiter and Schmidhuber, 1997) and GRU (Chung et al., 2014), RAN applies positional multi-head self-attention (pMHSA) on a window area to extract local dependency. To propagate the information forward, the RAN model extracts the global perception cell (GPC) vector from the self-attention representation of the current window. The GPC vector is then concatenated with tokens in the next window as the input of the self-attention layer. The new GPC vector will be passed to the subsequent windows with residual connection to alleviate the gradient vanishing (He et al., 2016) and updated in the same manner. Figure 1 depicts the difference between the recurrent neural network (RNN) and our proposed RAN.

The function of the GPC vector is twofold. First, like the [CLS] token in BERT, the GPC vector is a window-level contextual representation. But unlike the [CLS] token, the GPC vector is only applied to the self-attention layer, and no special token is inserted during text preprocessing. Second, the GPC vector, resembling the state cell in a recurrent architecture, maintains a long-distance memory over the sequence. For each window, the attended GPC vector encodes an aggregated representation of all the previous windows, which enables the window-level self-attention to perceive global semantics. With the help of a well-designed memory review mechanism, the GPC vector from the last window can be used as a document-level representation and serve the classification tasks. Meanwhile, the memory review mechanism enhances the token representations of RAN in the sequence, encoding both contextual and global information, which can be leveraged for sequential tasks such as language modeling (LM) and named entity recognition (NER).

We pretrain the RAN model using a masked language modeling (MLM) objective from scratch, which outperforms other pretrained baselines in long document classification. The RAN framework also supports auto-regressive LM and achieves the lowest perplexity score compared with state-of-the-art language models on the WikiText-103 dataset. Furthermore, we apply RAN to different downstream tasks via finetuning and observe consistent improvements compared to baseline models.

RAN solely relies on self-attention, and no LSTM-style gate is involved when propagating information via GPC vectors. Therefore, RAN is computationally efficient as it supports parallelized GPU computing. Although the memory complexity is still quadratic, it is regarding the window size W rather than the whole text length L, where $W \ll L$. Nevertheless, the window size can be adjusted based on hardware availability to achieve a relatively larger batch size for better training.

In summary, our contribution is to devise the RAN model for long document processing. RAN allows for parallelization on GPU and provides the interfaces for serving both classification and sequential tasks. With pretraining, RAN can outperform the BERT-based models in various tasks.

2 Related Work

This section reviews the relevant works focusing on sequence modeling in NLP, especially long document processing. RNNs are widely used for sequential modeling by recursively updating a state cell to maintain a long-distance memory. Traditional recurrent networks, such as LSTM (Hochreiter and Schmidhuber, 1997) and GRU (Chung et al., 2014), use the fully-connected layer as the basic encoding unit and apply the gate mechanism to update state memory. The recurrent operation is conducted on the token level, which is inefficient as such a framework cannot compute parallelly on GPU. Besides, it might suffer from gradient vanishing for long sequences during the backpropagation phase (Hutchins et al., 2022).

Self-attention models are powerful in global

representation learning. However, applying self-attention in long document processing is intractable due to the quadratic time and memory complexities. To address this issue, some works (Beltagy et al., 2020; Choromanski et al., 2021; Hua et al., 2022) attempt to reduce the computing complexity of self-attention from quadratic to approximately linear complexity. Beltagy et al. (2020) propose a drop-in replacement of the softmax operation in self-attention with a sparse attention mechanism. Similarly, Choromanski et al. (2021) rely on prior knowledge like sparsity and low-rankness to efficiently estimate the full-rank attention. However, these approaches face a trade-off between efficiency and accuracy, as approximations may lead to a performance drop (Schlag et al., 2021; Hutchins et al., 2022).

Other works leverage the power of full-rank self-attention as backbones, such as pretrained BERT and RoBERTa. These works cope with the token-length limitation with different strategies. Ding et al. (2020) propose CogLTX framework to generate a brief summary of the document. The short summary is used for the classification task employing BERT. However, it is inevitable to lose information in the length compression. Pappagari et al. (2019) segment the long text into smaller chunks so that BERT can be then used. A recurrent layer is employed to obtain the document-level representation upon chunk-level representations. These models can be applied for the classification task but cannot handle sequential tasks because of losing crucial contextual and sequential information. Hutchins et al. (2022) adopt the chunking strategy and devise a specifically-designed gate mechanism to obtain token-level representations for sequential tasks. Similarly, Didolkar et al. (2022) propose a Transformer-based temporal latent bottleneck for image classification, reinforcement learning, and text classification, in which temporal states are updated using a recurrent function across chunks. In each Transformer block, temporal states update the chunk-level representation by cross-attention layers interleaved with self-attention layers. In general, these models with BERT backbones cannot simultaneously handle classification and sequential tasks for long documents. Meanwhile, the RNN-style gate architecture does not support parallel computing, so the computing efficiency is also impaired.

Our proposed RAN achieves recurrent operation of the self-attention model and hence supports parallelization. Moreover, like the traditional RNN architecture, RAN can produce both token-level and document-level representations, which can be leveraged for both sequential and classification tasks.

3 Methodology

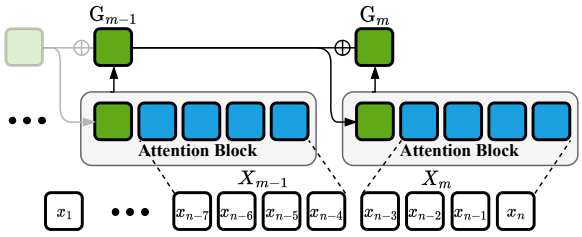


Figure 2: The generic framework of RAN.

This section introduces the proposed RAN framework in terms of its components. Figure 2 depicts the structure of the basic RAN module. In RAN, the primary encoder is the pMHSA, encoding the GPC vector and the current input with the rotary positional information carried (Su et al., 2021). The GPC vector is employed to propagate information through the sequence.

3.1 Input Layer

We first employ the padding operation for the input documents to keep a uniform length L. Then we map each word into a D-dimensional continuous space and obtain the word embedding $\mathbf{x}_i \in \mathbb{R}^D$. The word vectors are concatenated to form the model input: $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L] \in \mathbb{R}^{L \times D}$. To feed the text into the RAN, we chunk the input document into $m = \text{ceil}(\frac{L}{W})$ windows, where W is the window size. We use $\mathbf{X}_i \in \mathbb{R}^{W \times D}$ to denote the i-th window input. In RAN, the GPC vector $\mathcal{G}_0 \in \mathbb{R}^D$ is initialized to $\mathbf{0}$ by default, following the common operation in RNN. We parameterize the GPC vector with the layer normalization (Ba et al., 2016) as follows:

$$\mathbf{G}_0 = \text{LayerNorm}(\mathbf{W}_g \mathcal{G}_0) \in \mathbb{R}^D. \quad (1)$$

3.2 Positional Multi-Head Self-Attention

As the positional space of long documents is prohibitively large, it is not feasible to use absolute positional embedding following Transformer families (Vaswani et al., 2017) in long document processing. Hence, we follow Su et al. (2021); Chowdhery et al. (2022); Black et al. (2022) to incorporate the local positional information of the current window and leverage rotary position information as follows:

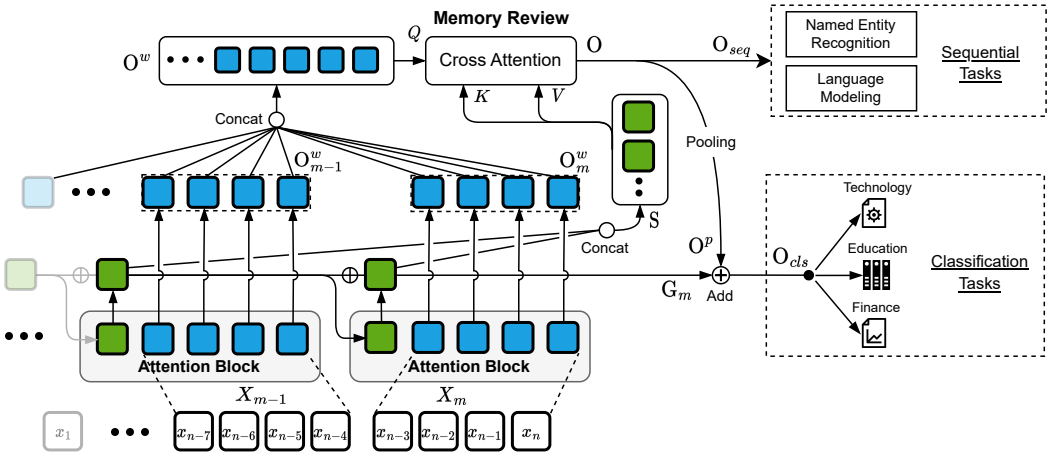


Figure 3: The frameworks of RAN with memory review on different tasks.

$$\text{pMHSA}(\mathbf{X}_i) = \mathbf{W}[\text{Att}_1(\mathbf{X}_i); ...; \text{Att}_h(\mathbf{X}_i)] + b, \tag{2}$$

and

$$\begin{aligned} \text{Att}_j(\mathbf{X}_i) &= \text{SoftMax}(\frac{\text{RP}(\mathbf{Q}_j) \cdot \text{RP}(\mathbf{K}_j^{\text{T}})}{\sqrt{d^k}} \\ &\quad + \mathbf{M})\mathbf{V}_j \\ \mathbf{Q}_j &= \mathbf{W}_j^q \mathbf{X}_i + b_j^q \\ \mathbf{K}_j &= \mathbf{W}_j^k \mathbf{X}_i + b_j^k \\ \mathbf{V}_j &= \mathbf{W}_j^v \mathbf{X}_i + b_j^v, \end{aligned} \tag{3}$$

where $\text{Att}_j(\cdot)$ is the j-th head of pMHSA, h denotes the head size, $[;]$ means the concatenation operation, $\mathbf{M}$ is the attention mask to adapt to different sequential tasks, and $\text{RP}(\cdot)$ stands for the rotary position function (Su et al., 2021).

3.3 Encoding and Updating Layer

To encode the i-th window, we concatenate the GPC vector from the previous window $\mathbf{G}_{i-1}$ and the current window input $\mathbf{X}_i \in \mathbb{R}^{W \times D}$ to form the model input $\mathbf{X}_i^{in} = [\mathbf{G}_{i-1}; \mathbf{X}_i] \in \mathbb{R}^{(1+W) \times D}$. A layer normalization layer is applied to normalize the input.

We then apply the pMHSA to encode the concatenated input to obtain the outputs of the current window:

$$\mathbf{O}_i = \text{pMHSA}(\mathbf{X}_i^{in}). \tag{4}$$

After encoding, we extract the updated GPC vector $\mathbf{G}_i'$ and the output corresponding to the tokens in the window:

$$\begin{aligned} \mathbf{G}_i' &= \text{SN}(\mathbf{O}_i^{[1:2]}) \in \mathbb{R}^D \\ \mathbf{O}_i^w &= \text{SN}(\mathbf{O}_i^{[2:1+W]}) \in \mathbb{R}^{W \times D}, \end{aligned} \tag{5}$$

where $^{[start:end]}$ is the tensor slice operation, and $\text{SN}(X) = \frac{X - X_{mean}}{\sigma}$ stands for the standard normalization. To alleviate the gradient vanishing issue in modeling long sequences, we employ residual connection to connect the current GPC vector with the previous one, then pass it to a layer normalization layer to normalize the updated GPC vector,

$$\mathbf{G}_i = \text{LayerNorm}(\mathbf{G}_i' + \mathbf{G}_{i-1}). \tag{6}$$

The updated GPC vector $\mathbf{G}_i \in \mathbb{R}^D$ will be propagated to the next window.

3.4 Memory Review and Output Layer

After encoding all windows, we can obtain the sequence output by concatenating all window outputs, as follows:

$$\mathbf{O}^w = [\mathbf{O}_1^w; \mathbf{O}_2^w; ...; \mathbf{O}_m^w] \in \mathbb{R}^{L \times D}, \tag{7}$$

where m is the number of windows. $\mathbf{O}^w$ has the same shape as the input $\mathbf{X}$. To prevent history forgetting in handling long sequences, this paper proposes a novel memory review mechanism. Specifically, we first concatenate all updated GPC vectors to produce the history states vector:

$$\mathbf{S} = [\mathbf{G}_1; \mathbf{G}_2; ...; \mathbf{G}_m] \in \mathbb{R}^{m \times D}. \tag{8}$$

We compute the cross attention of the concatenated output and the historical memory states to obtain the final output:

$$\begin{aligned} \mathbf{O} &= \text{SoftMax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d^k}})\mathbf{V} \\ \mathbf{Q} &= \mathbf{W}^q \mathbf{O}^w + b^q \\ \mathbf{K} &= \mathbf{W}^k \mathbf{S} + b^k \\ \mathbf{V} &= \mathbf{W}^v \mathbf{S} + b^v. \end{aligned} \tag{9}$$

This procedure mimics the human behavior of reviewing key points after reading an article, the way that humans naturally consolidate information and reinforce memory.

The sequence output $\mathbf{O} \equiv \mathbf{O}_{seq} \in \mathbb{R}^{L \times D}$ can be used for sequential tasks like NER. Although the GPC vector of the last window, $\mathbf{G}_m$, can serve as the document representation, it may lose crucial semantics and long-term memory during the propagation. Therefore, we also add the memory review mechanism to RAN for classification tasks by generating $\mathbf{O}_{clf}$:

$$\mathbf{O}_{clf} = \mathbf{W}^g \mathbf{G}_m + \mathbf{W}^o \mathbf{O}^p + b^o, \quad (10)$$

where $\mathbf{O}^p$ is the pooling of the output $\mathbf{O}$ over time sequence. Our empirical results show that the max pooling works better than the average pooling in classification tasks. Therefore, we adopt max pooling to obtain $\mathbf{O}^p$:

$$\mathbf{O}^p = \text{MaxPooling}(\mathbf{O}). \quad (11)$$

Figure 3 provides a visual illustration of the implementations for both classification and sequential tasks. It is noticeable that the model parameters of RAN are shared across all windows, allowing for efficient computation and reduced memory usage. Particularly, RAN supports multiple sequential tasks with different attention masks. For instance, it employs a causal attention mask (Vaswani et al., 2017) for LM tasks and a prefix causal attention mask (Dong et al., 2019) for the seq2seq tasks to prevent forward information exposure.

4 Experiment

4.1 Datasets and Evaluation Metrics

To comprehensively evaluate the model performance, we conduct experiments on three major tasks: text classification (TC), NER, and LM.

For the *TC task*, we attempt to test the model performance on datasets with various document lengths. Specifically, we extend the benchmarks from Park et al. (2022) by adding the long-text dataset Arxiv and the short-text dataset AGNews. The extended benchmarks include (1) **AGNews**[2], (2) **20NewsGroups** (Lang, 1995), and (3) **Arxiv** (He et al., 2019) for multi-class classification; (4) **Book Summary** (Park et al., 2022; Bamman and Smith, 2013) (*abbr.* **B.S.**) and (5) **EURLEX-57K** (Chalkidis et al., 2019) (*abbr.* **EUR.57K**) for multi-label classification; and **Hyperpartisan** (Kiesel et al., 2019) (*abbr.* **Hyper.**) for binary classification. Figure 4 depicts the text length distribution and the long-text ratio of the benchmark datasets. For a fair comparison, following Park et al. (2022), we report micro-F1 for multi-label classification and accuracy for binary and multi-class classification.

For the *LM task*, we adopt the commonly-used dataset **WikiText-103**[3] (Merity et al., 2017) and report the perplexity score following the baselines.

For the *NER task*, we experiment on two widely-adopted English datasets: **OntoNotesV5.0**[4] (*abbr.* **OntoV5**, average length is 77.5) and **CoNLL2003** (Tjong Kim Sang and De Meulder, 2003) (average length is 63.4). Noted that both datasets consist of short texts with an average length shorter than 100, as there are no available NER datasets of long documents. Accordingly, we adopt a small window size for the NER task to test the effectiveness of the recurrent architecture. We use *conlleval*[5] to measure the model performance and report the F1 score following the baselines.

4.2 Implementation Details

The primary experiments in Section 4.3 were conducted using the NVIDIA A100 GPU, while the remaining experiments were conducted using the NVIDIA Titan X GPU (12G memory). The code was implemented using TensorFlow and Keras. By default, we used two layers of RAN, with a window size of 256 for the TC and LM tasks and 64 for the NER task. The head number of pMHSA is set to 12, and the head size is 768. We trained the models for different tasks using the Adam optimizer (Kingma and Ba, 2015) by optimizing the corresponding objective function. For pretrained and non-pretrained RAN models, we set the learning rate to $2e-5$ and $3e-4$, respectively.

4.3 Main Results

4.3.1 Long Text Classification

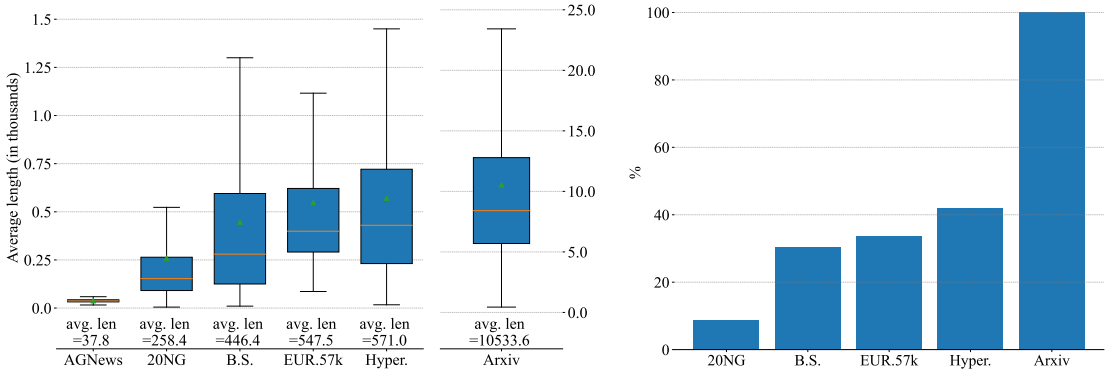
We compare RAN with baselines on the long-text classification task, including **BiLSTM** (Hochreiter and Schmidhuber, 1997) and pretrained language

[2] http://groups.di.unipi.it/~gulli/AG_corpus_of_news_articles

[3] https://blog.salesforceairesearch.com/the-wikitext-long-term-dependency-language-modeling-dataset

[4] https://catalog.ldc.upenn.edu/LDC2013T19

[5] https://www.clips.uantwerpen.be/conll2002/ner/bin/conlleval.txt



(a) Text length distribution boxplot.

(b) Ratio of the long texts (length > 512).

Figure 4: Statistics of long document benchmark datasets. In (a), the right y-axis is for the Arxiv dataset, and the left y-axis is for the rest datasets.

Model	AGNews	20NG	B.S.	Hyper.	EUR.57K	Arxiv	Avg.
	Acc.	Acc.	F1(micro)	Acc.	F1(micro)	Acc.	
BiLSTM+GloVe	93.34	77.97	49.90	90.77	65.00	81.28	76.38
BERT	93.80	84.79†	58.18†	92.00†	73.09†	82.00	80.64
Longformer	93.22	83.39†	56.53†	95.69†	54.53†	84.24	77.93
ToBERT	93.80	**85.52**†	58.16†	89.54†	67.57†	83.75	79.72
CogLTX	93.68	84.63†	58.27†	94.77†	70.13†	83.56	80.84
RAN+Random	91.70	78.88	50.52	93.85	66.59	80.08	76.94
RAN+GloVe	93.46	79.16	51.58	95.38	67.21	83.36	78.36
RAN+Pretrain	**93.83**	85.41	**58.43**	**96.92**	**73.94**	**85.92**	**82.41**

Table 1: Results on long document benchmarks for the classification task. † indicates results retrieved from Park et al. (2022). The rest results are from our implementation based on the official code. For the pre-trained word embedding GloVe, we use the embedding model glove.6B.300d[a]. *Acc.* is accuracy score. *Avg.* stands for the average numerical results. All the reported results are in percentage (%).

[a]https://nlp.stanford.edu/data/glove.6B.zip

models such as **BERT** (Devlin et al., 2019), **Longformer** (Beltagy et al., 2020), **ToBERT** (Pappagari et al., 2019), and **CogLTX** (Ding et al., 2020). For a comprehensive review, we adopt different initialization methods for RAN parameters. **RAN+Random** indicates the weights of RAN are randomly initialized. **RAN+GloVe** stands for using the GloVe embedding (Pennington et al., 2014) as word representation. **RAN+Pretrain** is the RAN pretrained on the MLM task, following settings in Devlin et al. (2019); Liu et al. (2019). We pretrained RAN on the BookCorpus (Zhu et al., 2015) (5GB) and C4 (Raffel et al., 2020) (RealNews-like subset, 15GB).

We present the results of long document benchmarks in Table 1. In general, the pretrained RAN achieves the five best results among the six benchmarks except for the 20NG dataset and outperforms all the baselines regarding the average score. Note that the pretrained RAN has only 96M parameters which are fewer than other pretrained baselines, suggesting that RAN is more efficient and scalable than the baselines. Particularly, the pretrained RAN achieves a 2.2% improvement compared with ToBERT on the super-long text dataset Arxiv, demonstrating the superiority of RAN in handling long documents.

It is worth noticing that the average performance of RAN is higher than that of the chunking-based ToBERT and the document summarization model CogLTX. These two models drop essential information in the chunking and summarizing processes, while RAN can preserve the sequence information

with the help of the well-designed recurrent and memory review mechanisms. Moreover, the pretrained RAN achieves the best result on the short-text dataset AGNews, indicating that RAN also performs well in short-text tasks.

Remarkably, even without pretraining, RAN can still yield competitive performance. For example, the randomly initialized RAN achieved better results than BiLSTM with pretrained GloVe word embedding. RAN with GloVe embedding outperforms pretrained BERT and ToBERT on the accuracy of the Hyper dataset and Longformer on average score. Such observations illustrate that RAN is effective for text encoding and flexible in adopting different initialization methods for various scenarios. It also suggests that the recurrent attention-based architecture of RAN is more powerful than the recurrent architecture of LSTM in modeling texts.

4.3.2 Language Modeling

The self-attention-based RAN can be employed for LM. Extensive experiments are conducted to evaluate RAN on language modeling. To avoid information exposure, we apply the causal attention mask to ensure the prediction for i-th position only depends on the known outputs before i, following Vaswani et al. (2017). We compare RAN with widely-adopted baselines that are shown in Table 2. The compared models have the same vocabulary and parameter sizes, and the parameters are randomly initialized. The experiment settings follow Zhong et al. (2022). Observing the results, we notice that RAN achieves the state-of-the-art result on the WikiText-103 dataset with 22.76 perplexity. It suggests that RAN is efficient in handling the sequence generation task.

Model	#Params	PPL↓
LSTM (Grave et al., 2017)	150M	48.70
TransformerXL (Dai et al., 2019)	150M	24.00
B.R. Trans. (Hutchins et al., 2022)	150M	39.48†
Transformer (Zhong et al., 2022)	150M	29.14
Com. Trans.(Zhong et al., 2022)	150M	24.56
∞-former (Zhong et al., 2022)	150M	24.22
TRIMELM (Zhong et al., 2022)	150M	25.60
RAN	150M	**22.76**

Table 2: Results of the LM task on the WikiText-103 dataset. Note that the parameter size for language modeling is much larger than that for classification tasks (96M) as we used the same vocabulary for all baselines for a fair comparison. ↓ means the result is the lower the better. † denotes that the result is from our implementation of the official code.

4.3.3 Named Entity Recognition

The NER task is a common information extraction task, and we conduct experiments on the NER task to test RAN for information extraction. As the available NER datasets contain mostly short texts, we set the window size to 64 to test the effectiveness of RAN's recurrent structure. We compare with the following widely-used baselines: **ID-CNN** (Strubell et al., 2017), **LSTM** (Langlais et al., 2018), **LSTM-CNN** (Li et al., 2020), **ELMo** (Peters et al., 2018), and **BERT** (Devlin et al., 2019). As shown in Table 3, we notice that RAN consistently outperforms LSTM-based baselines. Specifically, RAN without pretraining achieves 0.5% and 0.3% improvement compared with BERT on both datasets, indicating that the well-designed GPC vector is effective in handling information extraction of long sequences. Both the NER and LM tasks are sequential tasks, and the results demonstrate that RAN is effective in sequence modeling.

Model	OntoV5	CoNLL2003
ID-CNN (Strubell et al., 2017)	86.84	90.54
LSTM (Langlais et al., 2018)	87.95	91.73
LSTM-CNN (Li et al., 2020)	88.40	–
ELMo (Peters et al., 2018)	–	92.22
BERT (Devlin et al., 2019)	88.88†	92.40
RAN ($W = 64$)	**89.38**	**92.68**

Table 3: F1 score of the NER task. The results of the baselines are retrieved from the original paper. † denotes results from our implementation by the official code.

4.4 Ablation Study

We conducted an ablation study to investigate the significance of each component of our proposed model on the Arxiv dataset. The results are presented in Table 4.

In the first ablation model, we substituted the max pooling layer depicted in Eq. 11 with an average pooling layer, resulting in a 1.12% drop in Accuracy. Moreover, our findings show that the residual connection between two windows is essential to alleviate gradient vanishing. When we removed it from RAN, the performance drops approximately 1.6%. We also ablated the rotary positional encoding from RAN, which leads to a 1.23% performance drop.

When the memory review mechanism of RAN was removed in the last ablation model, the result shows the most significant drop compared with other ablation models. RAN without the mem-

ory review mechanism suffers a 2.5% performance drop. Such an observation indicates that mitigating information forgetting in processing long documents is crucial, and our proposed memory review mechanism is effective in preserving long-distance memory over the sequence.

In general, the ablation study demonstrates the significance of each component in our proposed RAN model and highlights the importance of the memory review mechanism in processing long documents. Particularly, our observation accentuates the importance of maintaining long-distance memory in long document processing.

Model	Accuracy (%)	Δ (%)
RAN+GloVe	**83.36**	
w/ avg pool	82.24	−1.12
w/o residual connection	81.76	−1.60
w/o memory review	80.85	−2.51
w/o rotary position	82.13	−1.23

Table 4: Results of ablation models on the Arxiv dataset.

4.5 Discussion

4.5.1 Scalability Analysis of RAN

This section discusses the scalability of RAN in actual implementation. The window size W determines the number of tokens that are encoded by the attention block. In theory, RAN with a larger window size can yield better performance, as we have fewer windows and less information loss when iterating over the windows. However, given the quadratic memory complexity, the hardware capacity limits the maximum batch size that can be used in training and hence imposes a ceiling to the performance improvement. We have conducted additional experiments with RAN+Glove on the Arxiv dataset. Figure 5 depicts the accuracy of the test set and the training time per epoch of RAN with different window sizes. Results of each configuration are obtained with the maximum batch size runnable on the GPU. As expected, when window size increases, the accuracy also gains continuous improvements, albeit minor. The accuracy curve begins to flatten out when the size exceeds 256, partially due to the decreasing maximum batch size. Such an observation indicates the performance is approaching the bottleneck caused by the hardware capacities.

On the other hand, the V-shape curve observed in the training time is an intriguing sign, and we

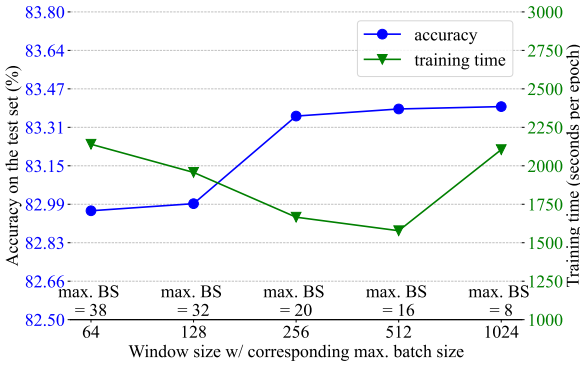


Figure 5: The accuracy of the test set (the blue line with •) and the training time (in seconds per epoch, the green line with ▾) of RAN + Glove with different RAN window sizes on the Arxiv dataset. For each window, the results are obtained with the maximum batch size (max. BS) allowed on a Titan X GPU.

attribute it to the different time complexities of recurrent and self-attention operations of RAN. Although computing self-attention is of quadratic time, it is significantly accelerated owing to the tensor computation on GPU. In contrast, the recurrent component involves tensor manipulations, such as splitting and concatenation, and thus takes more time. Therefore, a smaller window size will lead to a longer training time as more recurrent operations need to perform. When the window size is large enough, the quadratic time of self-attention becomes significant and dominates the overall spent time. Hence, the green curve bounces back when the window size is 1024. Moreover, when the window size is even larger, such as 2048, the model becomes too large to be loaded on the GPU, and training becomes infeasible.

Furthermore, we compare the training time of pretrained RAN with other pretrained and non-pretrained baselines on the Arxiv dataset. The results in Table 5 indicate that the proposed RAN is highly scalable and efficient. Notably, RAN has six times the parameter size of LSTM but has a shorter training time, which indicates our devised recurrent attention structure is more efficient than the LSTM-style gate mechanism.

4.5.2 The number of RAN Layers

Similar to RNNs, RAN layers can be stacked to build a deep architecture. We adopt a serial manner to pass the previous layer's GPC output as the input to the subsequent hidden RAN layers. The GPC output at the last RAN layer will be con-

Models	#Params	Time (s/epoch)
LSTM (w/o pretrain)	15M	7,947
LongFormer (w/ pretrain)	148M	26,536
ToBERT (w/ pretrain)	110M	6,568
CogLTX (w/ pretrain)	110M	21,032
RAN (w/ pretrain)	96M	5,393

Table 5: Training time comparison (in seconds per epoch) on Titan X GPU of different models on Arxiv dataset.

catenated with the following window input. Intuitively, with more RAN layers, the model will contain more parameters and is promising to produce higher performance. We compare the RANs with different depths and list the results in Table 6. As expected, the accuracy improves as the number of layers increases. However, the average training time will significantly increase due to the serial connection between layers, and the improvements become marginal. Therefore, to balance the performance and the time consumption, we adopt the two-layer RAN in this paper by default. This also implies that the results presented in this paper could be further enhanced by increasing the depth of the RAN.

Layers	#Params	Time (s/epoch)	Accuracy (%)
1	15M	678	83.14
2	17M	1,311	83.36
3	19M	2,186	83.76
4	21M	2,967	83.93

Table 6: Results of RAN+GloVe with different layers on Arxiv dataset. *Time* is the average training time (in seconds per epoch).

5 Conclusion & Future Work

This paper has presented a novel RAN architecture for long-text modeling that combines the advantages of both recurrent and self-attention networks. The use of a positional multi-head attention mechanism and GPC vector enhances the model's performance by capturing both local and global dependencies in the input sequence. Our ablation study also highlights the critical role of residual connection and memory review mechanisms in preserving long-distance memory.

With the well-designed recurrent self-attention mechanism, RAN's training can be accelerated by parallel computing on a GPU, making it highly efficient and scalable. We have conducted extensive experiments on TC, NER, and LM tasks. The extensive experiments demonstrate the effectiveness of the proposed RAN model on both classification and sequential tasks.

The flexibility and scalability of our proposed RAN make it a promising choice for future research, with broad potential applications in translation, summarization, conversation generation, and large language models. Additionally, we plan to extend the RAN to tasks involving multi-modality input and output like audio and video, to exploit RAN's long sequence handling capacity in different fields.

6 Limitations

The proposed model, Recurrent Attention Network (RAN), effectively models long sequential data by propagating information window-by-window through the sequence via its well-designed recurrent architecture. However, the multi-head self-attention applied to each window is still limited to local attention, which prevents it from providing a global dependency relationship for the entire sequence. This limitation restricts RAN's application in scenarios where a global dependency relationship is necessary, such as visualizing attention weights for the entire document via a heatmap. This limitation potentially reduces the interpretability of the model, although it does not affect the model's performance. Hence, exploring ways to incorporate global attention mechanisms into the RAN architecture is a promising research direction to improve its interpretability and expand its range of applications.

Acknowledgments

Xianming Li, Xiaotian Luo, Xing Lee, and Yingbin Zhao's work has been supported by Ant Group. Zongxi Li's work has been supported by a grant from Hong Kong Metropolitan University (Project Reference No. CP/2022/02). Haoran Xie's work has been supported by the Direct Grant (DR23B2) and the Faculty Research Grant (DB23A3) of Lingnan University, Hong Kong. Qing Li's work has been supported by the Hong Kong Research Grants Council through the Collaborative Research Fund (Project No. C1031-18G). We thank the anonymous reviewers for their careful reading of our manuscript. Their insightful comments and suggestions helped us improve the quality of our manuscript.

References

Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *CoRR*, abs/1607.06450.

David Bamman and Noah A. Smith. 2013. New alignment methods for discriminative book summarization. *CoRR*, abs/1305.1319.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *CoRR*, abs/2004.05150.

Sid Black, Stella Biderman, Eric Hallahan, et al. 2022. Gpt-neox-20b: An open-source autoregressive language model. *CoRR*, abs/2204.06745.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. volume 33, pages 1877–1901.

Ilias Chalkidis, Emmanouil Fergadiotis, Prodromos Malakasiotis, and Ion Androutsopoulos. 2019. Large-scale multi-label text classification on EU legislation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6314–6322, Florence, Italy. Association for Computational Linguistics.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J. Colwell, and Adrian Weller. 2021. Rethinking attention with performers. In *9th International Conference on Learning Representations, ICLR 2021*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Jeff Dean, et al. 2022. Palm: Scaling language modeling with pathways. *CoRR*, abs/2204.02311.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc Viet Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019*, pages 2978–2988. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186.

Aniket Didolkar, Kshitij Gupta, Anirudh Goyal, Nitesh Bharadwaj Gundavarapu, Alex M Lamb, Nan Rosemary Ke, and Yoshua Bengio. 2022. Temporal latent bottleneck: Synthesis of fast and slow processing mechanisms in sequence learning. *Advances in Neural Information Processing Systems*, 35:10505–10520.

Ming Ding, Chang Zhou, Hongxia Yang, and Jie Tang. 2020. Cogltx: Applying BERT to long texts. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*.

Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pre-training for natural language understanding and generation. In *Annual Conference on Neural Information Processing Systems 2019*, pages 13042–13054.

Edouard Grave, Armand Joulin, and Nicolas Usunier. 2017. Improving neural language models with a continuous cache. In *5th International Conference on Learning Representations, ICLR 2017*.

Jun He, Liqun Wang, Liu Liu, Jiao Feng, and Hao Wu. 2019. Long document classification from local word glimpses via recurrent attention learning. *IEEE Access*, 7:40707–40718.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*, pages 770–778.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc Le. 2022. Transformer quality in linear time. In *International Conference on Machine Learning*, pages 9099–9117.

DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. 2022. Block-recurrent transformers. *CoRR*, abs/2203.07852.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. 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*, pages 6769–6781. Association for Computational Linguistics.

Johannes Kiesel, Maria Mestre, Rishabh Shukla, Emmanuel Vincent, Payam Adineh, David Corney, Benno Stein, and Martin Potthast. 2019. SemEval-2019 task 4: Hyperpartisan news detection. In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 829–839, Minneapolis, Minnesota, USA. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*.

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 *8th International Conference on Learning Representations, ICLR 2020*.

Ken Lang. 1995. Newsweeder: Learning to filter netnews. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 331–339. Morgan Kaufmann.

Abbas Ghaddar Philippe Langlais et al. 2018. Robust lexical features for improved neural network named-entity recognition. In *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018*, pages 1896–1907. Association for Computational Linguistics.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. In *Proceedings of the 57th Conference of the Association for Computational Linguistics*, pages 6086–6096.

Peng-Hsuan Li, Tsu-Jui Fu, and Wei-Yun Ma. 2020. Why attention? Analyze bilstm deficiency and its remedies in the case of NER. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*, pages 8236–8244. AAAI Press.

Xianming Li, Zongxi Li, Haoran Xie, and Qing Li. 2021a. Merging statistical feature via adaptive gate for improved text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13288–13296.

Xianming Li, Xiaotian Luo, Chenghao Dong, Daichuan Yang, Beidi Luan, and Zhen He. 2021b. TDEER: an efficient translating decoding schema for joint extraction of entities and relations. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8055–8064.

Zongxi Li, Xianming Li, Haoran Xie, Fu Lee Wang, Mingming Leng, Qing Li, and Xiaohui Tao. 2023. A novel dropout mechanism with label extension schema toward text emotion classification. *Information Processing & Management*, 60(2):103173.

Zongxi Li, Haoran Xie, Gary Cheng, and Qing Li. 2021c. Word-level emotion distribution with two schemas for short text emotion classification. *Knowledge-Based Systems*, page 107163.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017*.

Usman Naseem, Imran Razzak, Katarzyna Musial, and Muhammad Imran. 2020. Transformer based deep intelligent contextual embedding for twitter sentiment analysis. *Future Generation Computer Systems*, 113:58–69.

Raghavendra Pappagari, Piotr Zelasko, Jesús Villalba, Yishay Carmiel, and Najim Dehak. 2019. Hierarchical transformers for long document classification. In *IEEE Automatic Speech Recognition and Understanding Workshop, ASRU 2019*, pages 838–844. IEEE.

Hyunji Hayley Park, Yogarshi Vyas, and Kashif Shah. 2022. Efficient classification of long documents using transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, ACL 2022*, pages 702–709. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Matthew E. 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, NAACL-HLT 2018*, pages 2227–2237. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

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.*, pages 140:1–140:67.

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

Emma Strubell, Patrick Verga, David Belanger, and Andrew McCallum. 2017. Fast and accurate entity recognition with iterated dilated convolutions. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017*, pages 2670–2680. Association for Computational Linguistics.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. 2021. Roformer: Enhanced transformer with rotary position embedding. *CoRR*, abs/2104.09864.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

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 *Advances in Neural Information Processing Systems*, pages 5998–6008.

Xueqing Wu, Jiacheng Zhang, and Hang Li. 2022. Text-to-table: A new way of information extraction. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, pages 2518–2533. Association for Computational Linguistics.

Zexuan Zhong, Tao Lei, and Danqi Chen. 2022. Training language models with memory augmentation.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
6

☒ A2. Did you discuss any potential risks of your work?
This paper focuses on fundamental research of long sequence modeling in natural language processing. The proposed model does not contain any component that is potentially malicious or harmful to human beings. The model can only be applied for classification and information retrieval tasks, and in its current form, the model cannot be used for conducting adversarial attacks. The model was tested on language modeling with sentence generation, in which the pretraining corpus are commonly adopted ones for language model pretraining.
The work described does not aim to present any new dataset and hence is free of disseminating any false/biased/unfair/discriminative information. The model and its variants are trained with widely adopted and recognized data for different tasks, thus, to the best of our knowledge, it will not contribute to any fairness issue.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
1

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

4.1, 4.3

☑ B1. Did you cite the creators of artifacts you used?
4.1

☒ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Most of the datasets we used are open for research purposes. Rest datasets such as OntoNotesV5.0, we have applied to use in this paper on the corresponding official sites, and we have been granted to use in this paper.

☑ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
4.1

☒ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
The datasets we used have been widely adopted in the literature and are considered safe to use.

☑ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
4.1

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
4.1

C ☑ Did you run computational experiments?

4

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
4.2,4.3

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
4.2

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
4.3

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
4.2

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.