RWKV: Reinventing RNNs for the Transformer Era

Bo Peng[1]* **Eric Alcaide**[2,3,4]* **Quentin Anthony**[2,5]*
Alon Albalak[2,6] **Samuel Arcadinho**[2,7] **Huanqi Cao**[8] **Xin Cheng**[9] **Michael Chung**[10]
Matteo Grella[11] **Kranthi Kiran GV**[12] **Xuzheng He**[2] **Haowen Hou**[13] **Przemysław Kazienko**[14]
Jan Kocoń[14] **Jiaming Kong**[15] **Bartłomiej Koptyra**[14] **Hayden Lau**[2] **Krishna Sri Ipsit Mantri**[16]
Ferdinand Mom[17,18] **Atsushi Saito**[2,19] **Xiangru Tang**[20] **Bolun Wang**[27] **Johan S. Wind**[21] **Stanisław Woźniak**[14]
Ruichong Zhang[8] **Zhenyuan Zhang**[2] **Qihang Zhao**[22,23] **Peng Zhou**[27] **Jian Zhu**[24] **Rui-Jie Zhu**[25,26]

[1]RWKV Foundation [2]EleutherAI [3]University of Barcelona [4]Charm Therapeutics [5]Ohio State University
[6]University of California, Santa Barbara [7]Zendesk [8]Tsinghua University [9]Peking University
[10]Storyteller.io [11]Crisis24 [12]New York University [13]National University of Singapore
[14]Wroclaw University of Science and Technology [15]Databaker Technology Co. Ltd [16]Purdue University
[17]Criteo AI Lab [18]Epita [19]Nextremer Co. Ltd. [20]Yale University [21]University of Oslo
[22]University of Science and Technology of China [23]Kuaishou Technology Co. Ltd
[24]University of British Columbia [25]University of California, Santa Cruz
[26]University of Electronic Science and Technology of China [27]RuoxinTech

Abstract

Transformers have revolutionized almost all natural language processing (NLP) tasks but suffer from memory and computational complexity that scales quadratically with sequence length. In contrast, recurrent neural networks (RNNs) exhibit linear scaling in memory and computational requirements but struggle to match the same performance as Transformers due to limitations in parallelization and scalability. We propose a novel model architecture, Receptance Weighted Key Value (RWKV), that combines the efficient parallelizable training of Transformers with the efficient inference of RNNs. Our approach leverages a linear attention mechanism and allows us to formulate the model as either a Transformer or an RNN, which parallelizes computations during training and maintains constant computational and memory complexity during inference, leading to the first non-transformer architecture to be scaled to tens of billions of parameters. Our experiments reveal that RWKV performs on par with similarly sized Transformers, suggesting that future work can leverage this architecture to create more efficient models. This work presents a significant step towards reconciling the trade-offs between computational efficiency and model performance in sequence processing tasks.[1]

1 Introduction

Deep learning techniques have made significant strides in artificial intelligence, playing a pivotal role in various scientific and industrial applications. These applications often involve complex sequential data processing tasks that include natural language understanding, conversational AI, time-series analysis, and even indirect modalities that can be reframed as sequences, such as images and graphs (Brown et al., 2020; Ismail Fawaz et al., 2019; Wu et al., 2020; Albalak et al., 2022). Predominant among these techniques are RNNs, convolutional neural networks (CNNs), and the Transformer models (Vaswani et al., 2017).

Each of these has distinct drawbacks that restrict their efficiency in certain scenarios. RNNs suffer from the vanishing gradient problem, making them difficult to train for long sequences. Additionally, they cannot be parallelized in the time dimension during training, which restricts their scalability (Hochreiter, 1998; Le and Zuidema, 2016). CNNs, on the other hand, are only adept at capturing local patterns, which limits their capacity to deal with long-range dependencies, crucial to many sequence processing tasks (Bai et al., 2018).

Transformer models emerged as a powerful alternative due to their ability to handle both local and long-range dependencies and their capability for parallelized training (Tay et al., 2022). Recent models such as GPT-3 (Brown et al., 2020), ChatGPT (OpenAI, 2022; Kocoń et al., 2023), GPT-4 (OpenAI, 2023), LLaMA (Touvron et al., 2023), and Chinchilla (Hoffmann et al., 2022) exemplify the capability of this architecture, pushing the frontiers of what's possible in NLP. Despite these significant advancements, the self-attention mechanism inherent to Transformers poses unique challenges,

* Equal first authorship. Others listed alphabetically.
[1]Code at: https://github.com/BlinkDL/RWKV-LM

Model	Time	Space
Transformer	$O(T^2 d)$	$O(T^2 + Td)$
Reformer	$O(T \log Td)$	$O(T \log T + Td)$
Linear Transformers	$O(Td^2)$	$O(Td + d^2)$
Performer	$O(Td^2 \log d)$	$O(Td \log d + d^2 \log d)$
AFT-full	$O(T^2 d)$	$O(Td)$
MEGA	$O(cTd)$	$O(cTd)$
RWKV (ours)	$O(\mathbf{Td})$	$O(\mathbf{d})$

Table 1: Complexity comparison with different Transformers: Reformer (Kitaev et al., 2020), Linear Transformer (Katharopoulos et al., 2020), Performer (Choromanski et al., 2020), AFT (Zhai et al., 2021), MEGA (Ma et al., 2023). Here T denotes the sequence length, d the feature dimension, and c is MEGA's chunk size of quadratic attention.

primarily due to its quadratic complexity. This complexity renders the architecture computationally expensive and memory-intensive for tasks involving long input sequences or in resource-constrained situations. These limitations have spurred a wealth of research aiming to improve the scaling properties of Transformers, often at the expense of some of the properties that make it so effective (Wang et al., 2020; Zaheer et al., 2020; Dao et al., 2022a).

To tackle these challenges, we introduce the Receptance Weighted Key Value (**RWKV**) model, a novel architecture that effectively combines the strengths of RNNs and Transformers while circumventing key drawbacks. RWKV is carefully designed to alleviate the memory bottleneck and quadratic scaling associated with Transformers (Katharopoulos et al., 2020) with a more efficient linear scaling, while still preserving the rich, expressive properties that make the Transformer a dominant architecture in the field.

One of the defining characteristics of RWKV is its ability to offer parallelized training and robust scalability, similar to Transformers. Moreover, we have reformulated the attention mechanism in RWKV to introduce a variant of linear attention, eschewing the traditional dot-product token interaction in favor of more effective channel-directed attention. This approach contrasts significantly with the traditional Transformer architecture, where specific token interactions predominantly drive attention. The implementation of linear attention in RWKV is carried out *without approximation*, which offers a considerable improvement in efficiency and enhances the scalability, see Table 1.

The overarching motivation behind developing RWKV is to bridge the gap between computational efficiency and expressive capacity in neural network architectures. It offers a promising and viable solution for handling tasks involving large-scale models with billions of parameters, exhibiting competitive performance at a fraction of the computational cost. Our experimental results suggest that RWKV could be a valuable tool for addressing the ongoing challenges in scaling and deploying AI models across various domains, particularly those involving sequential data processing. Thus, RWKV paves the way for the next generation of more sustainable and computationally efficient AI models for sequence processing tasks.

Our contributions in this paper are as follows:

- We introduce the RWKV network architecture, which combines the advantages of RNNs and Transformers while mitigating their known limitations.
- We propose a new attention mechanism reformulation that results in linear attention, eschewing the quadratic complexity associated with standard Transformer models.
- We conduct a comprehensive series of experiments on benchmark datasets to showcase the performance, efficiency and scaling of RWKV in managing tasks involving large-scale models and long-range dependencies.
- We release pretrained model ranging in size from 169 million to 14 billion parameters trained on the Pile (Gao et al., 2020).[2]

2 Related Work

Recently, a number of techniques have been proposed to address the limitations of transformers.

Optimizing Attention Mechanism Many transformer variants ("x-formers") have been introduced to reduce the complexity of transformers (Tay et al., 2022), including sparse attention (Beltagy et al., 2020; Kitaev et al., 2020; Guo et al., 2022), approximating the full attention matrix (Wang et al., 2020; Ma et al., 2021; Choromanski et al., 2020), combining chunked attention with gating (Ma et al., 2023) and other efficient methods (Katharopoulos et al., 2020; Jaegle et al., 2021).

Some recent works like FlashAttention (Dao et al., 2022a) and others (Rabe and Staats, 2022; Jang et al., 2019) share similarities with RWKV's chunked computation scheme. Despite being memory-efficient, their time complexity remains quadratic or contains chunk size as a hidden factor. In contrast, RWKV achieves better space and

[2] https://huggingface.co/RWKV

time complexity during inference by formulating a linear attention as an RNN.

Attention Free Models Another line of research replaces the attention mechanism with other modules to scale to long sequences. MLP-Mixer and others (Tolstikhin et al., 2021; Liu et al., 2021) proposed the replacement of attention by Multi-Layer Perceptrons (MLPs) in computer vision tasks. The Attention Free Transformer (AFT) (Zhai et al., 2021) replaces dot-product self-attention with a computationally efficient alternative which can be seen as a multi-head attention where each feature dimension corresponds to a head. Inspired by AFT, RWKV takes a similar approach but modifies the interaction weights for simplicity such that it can be transformed into an RNN. In parallel, RNN-style (Hochreiter and Schmidhuber, 1997; Chung et al., 2014) recursive components have also been modified to increase context length, such as the Recurrent Memory Transformer (Bulatov et al., 2022, 2023) and Linear Recurrent Units (Orvieto et al., 2023). State space models (SSM) like S4 (Gu et al., 2022) and its variants (Dao et al., 2022b; Poli et al., 2023) are also proposed.

Notably, Quasi-Recurrent neural network (QRNN) (Bradbury et al., 2017) uses both convolutional layers and recurrent pooling functions across timesteps and channels. While QRNN utilizes convolutional filters with fixed sizes, RWKV employs a time-mixing module as an attention mechanism with time-decaying factors. Different from the element-wise pooling in QRNN, RWKV includes a parametrized channel-mixing module (see the green blocks in Fig.1c) that is parallelizable.

3 Background

Here we briefly review the fundamentals of RNNs and Transformers.

3.1 Recurrent Neural Networks (RNNs)

Popular RNN architectures such as LSTM (Hochreiter and Schmidhuber, 1997) and GRU (Chung et al., 2014) are characterized by the following formulation (shown for LSTM, others can be reasoned similarly):

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f), \quad (1)$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i), \quad (2)$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o), \quad (3)$$
$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c), \quad (4)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \quad (5)$$
$$h_t = o_t \odot \sigma_h(c_t). \quad (6)$$

The data flow of RNNs is shown in Fig. 1a. Although RNNs can be factored into two linear blocks (W and U) and an RNN-specific block (1)–(6), as noted by Bradbury et al. (2017), the data dependency relying on previous time steps prohibits parallelizing these typical RNNs.

3.2 Transformers and AFT

Introduced by Vaswani et al. (2017), Transformers are a class of neural networks that have become the dominant architecture for several NLP tasks. Instead of operating on sequences step-by-step like RNNs, Transformers rely on attention mechanisms to capture relationships between all input and all output tokens:

$$\text{Attn}(Q, K, V) = \text{softmax}(QK^\top)V, \quad (7)$$

where the multi-headness and scaling factor $\frac{1}{\sqrt{d_k}}$ is omitted for convenience. The core $QK^\top$ multiplication is an ensemble of pairwise attention scores between each token in a sequence, which can be decomposed as vector operations:

$$\text{Attn}(Q, K, V)_t = \frac{\sum_{i=1}^{T} e^{q_t^\top k_i} v_i}{\sum_{i=1}^{T} e^{q_t^\top k_i}}. \quad (8)$$

In AFT (Zhai et al., 2021), this is alternately formulated as

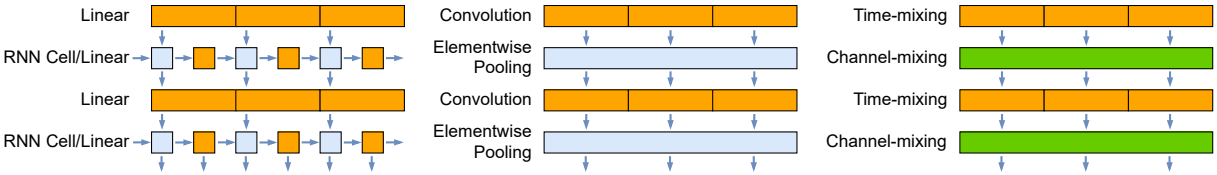
$$\text{Attn}^+(W, K, V)_t = \frac{\sum_{i=1}^{t} e^{w_{t,i}+k_i} v_i}{\sum_{i=1}^{t} e^{w_{t,i}+k_i}}, \quad (9)$$

where $\{w_{t,i}\} \in R^{T\times T}$ is the learned pair-wise position biases, and each $w_{t,i}$ is a scalar.

Inspired by AFT, we let each $w_{t,i}$ in RWKV be a channel-wise time decay vector multiplied by the relative position, traced backwards from current time as it decays:

$$w_{t,i} = -(t-i)w, \quad (10)$$

where $w \in (R_{\geq 0})^d$, with d the number of channels. We require w to be non-negative to ensure that $e^{w_{t,i}} \leq 1$ and the per-channel weights decay backwards in time.



(a) RNN (b) QuasiRNN (Bradbury et al., 2017) (c) RWKV

Figure 1: Computation structure of the RWKV in comparison to QRNN and RNN (Vanilla, LSTM, GRU, etc) architectures. Color codes: orange indicates time-mixing, convolutions or matrix multiplications, and the continuous block indicates that these computations can proceed simultaneously; blue signifies parameterless functions that operate concurrently along the channel or feature dimension (element-wise). Green indicates channel-mixing.

4 The Receptance Weighted Key Value (RWKV) Model

The RWKV architecture derives its name from the four primary model elements used in the time-mixing and channel-mixing blocks:

- R: **Receptance** vector acting as the acceptance of past information.
- W: **Weight** is the positional weight decay vector. A trainable model parameter.
- K: **Key** is a vector analogous to K in traditional attention.
- V: **Value** is a vector analogous to V in traditional attention.

Interactions between the main elements for every timestep are multiplicative, as illustrated in Fig. 2

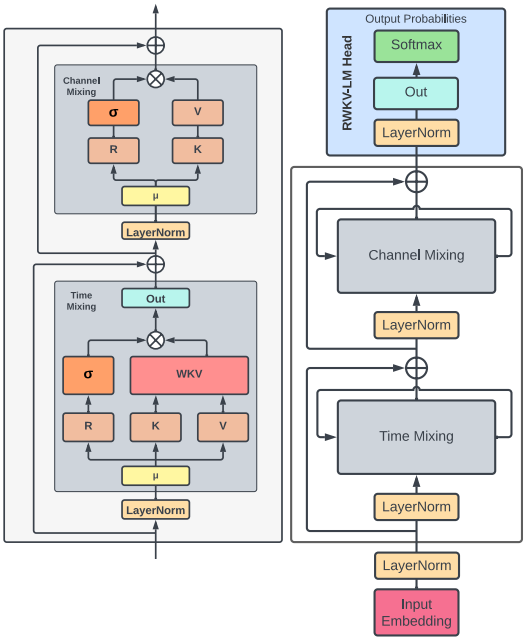


Figure 2: RWKV block elements (left) and RWKV residual block with a final head for language modeling (right) architectures.

4.1 High-Level Summary

The RWKV architecture is comprised of a series of stacked residual blocks, each formed by a time-mixing and a channel-mixing sub-blocks with recurrent structures.

The recurrence is formulated both as a linear interpolation between the current input and the input at the previous time step (a technique we refer to as time-shift mixing or token shift, indicated by the diagonal lines in Fig. 3), which can be adjusted independently for every linear projection of the input embedding (e.g., R, K, V in time-mixing, and R, K in channel-mixing), and as the time-dependent update of the WKV which is formalized in equation 14. The WKV computation is similar to AFT (Zhai et al., 2021), but W is now a channel-wise vector multiplied by relative position rather than a pairwise matrix in AFT. We also introduce a vector U for separately attending to the current token in order to compensate for potential degeneration of W (see Appendix G for more details).

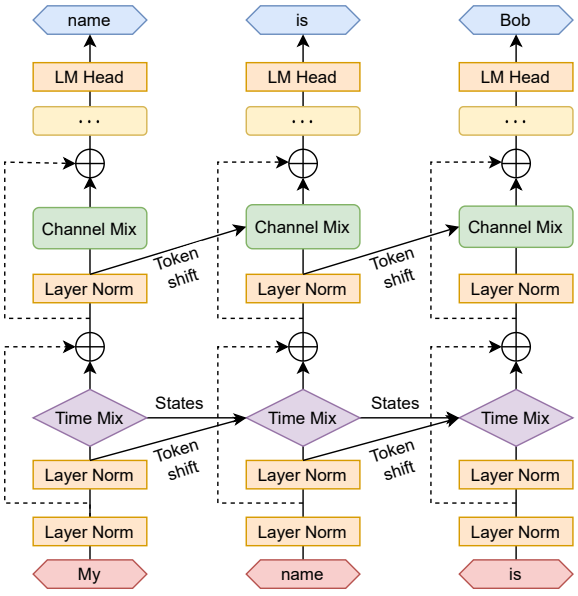


Figure 3: RWKV architecture for language modelling.

The time-mixing block is given by:

$$r_t = W_r \cdot (\mu_r x_t + (1 - \mu_r) x_{t-1}), \quad (11)$$

$$k_t = W_k \cdot (\mu_k x_t + (1 - \mu_k) x_{t-1}), \quad (12)$$

$$v_t = W_v \cdot (\mu_v x_t + (1 - \mu_v) x_{t-1}), \quad (13)$$

$$wkv_t = \frac{\sum_{i=1}^{t-1} e^{-(t-1-i)w+k_i} v_i + e^{u+k_t} v_t}{\sum_{i=1}^{t-1} e^{-(t-1-i)w+k_i} + e^{u+k_t}}, \quad (14)$$

$$o_t = W_o \cdot (\sigma(r_t) \odot wkv_t), \quad (15)$$

where the WKV computation, wkv_t, plays the role of $\mathrm{Attn}(Q, K, V)$ in Transformers without incurring a quadratic cost as interactions are between scalars. Intuitively, as time t increases, the vector o_t is dependent on a long history, represented by the summation of an increasing number of terms. For the target position t, RWKV performs a weighted summation in the positional interval of $[1, t]$, and then multiplies with the receptance $\sigma(r)$. Therefore, interactions are multiplicative inside a given timestep and summed over different timesteps.

Further, the channel-mixing block is given by:

$$r_t = W_r \cdot (\mu_r x_t + (1 - \mu_r) x_{t-1}), \quad (16)$$

$$k_t = W_k \cdot (\mu_k x_t + (1 - \mu_k) x_{t-1}), \quad (17)$$

$$o_t = \sigma(r_t) \odot (W_v \cdot \max(k_t, 0)^2), \quad (18)$$

where we adopt squared ReLU activation (So et al., 2021). Note that in both time-mixing and channel-mixing, by taking the sigmoid of the receptance, we're intuitively using it as a "forget gate" to eliminate unnecessary historical information.

4.2 Transformer-like Parallelization

RWKV can be efficiently parallelized in what we call a *time-parallel mode*, reminiscent of Transformers. The time complexity of processing a batch of sequences in a single layer is $O(BTd^2)$, which mainly consists of matrix multiplications $W_\square$, $\square \in \{r, k, v, o\}$ (assuming B sequences, T maximum tokens and d channels). Meanwhile, updating attention scores wkv_t requires a serial scan (see Appendix B for more detail) and has complexity $O(BTd)$.

The matrix multiplications can be parallelized akin to $W^\square$, $\square \in \{Q, K, V, O\}$ in typical Transformers. The element-wise WKV computation is time-dependent, but can be readily parallelized along the other two dimensions (Lei et al., 2018)[3].

[3]If the sequence is very long, more sophisticated methods such as Martin and Cundy (2017) that parallelize over sequence length could be used.

Additionally, token shift is implemented as a simple offset in the temporal dimension at each block using PyTorch (Paszke et al., 2019) library as `nn.ZeroPad2d((0,0,1,-1))`.

4.3 RNN-like Sequential Decoding

It is common in recurrent networks to use output at state t as input at state $t + 1$. This is especially evident in the autoregressive decoding inference of a language model, requiring each token to be computed before fed into the next step, making it possible for RWKV to take advantage of its RNN-like structure, referred to as *time-sequential mode*. In such circumstances, RWKV can be conveniently formulated recursively for decoding during inference, as shown in Appendix B, which leverages the advantage that each output token is dependent only on the latest state, which is of constant size, irrespective of the sequence length.

It then behaves as an RNN decoder, yielding constant speed and memory footprint with respect to the sequence length, enabling the processing of longer sequences more efficiently. In contrast, self-attention typically requires a KV cache growing linearly with respect to the sequence length, resulting in degraded efficiency and increasing memory footprint and time as the sequence grows longer.

4.4 Software Implementation

RWKV is originally implemented using the Pytorch Deep Learning Library (Paszke et al., 2019) and a custom CUDA kernel for the WKV computation explained in 4.7. Although RWKV is a general recurrent network, its current implementation focuses in the task of language modeling (RWKV-LM). The model architecture is comprised of an embedding layer, for which we follow the setup described in Section 4.7 and several identical residual blocks applied sequentially as seen in Fig. 2 and 3 following the principles outlined in Section 4.6. After the last block, a simple output projection head composed by a LayerNorm (Ba et al., 2016) and a linear projection is used to obtain the logits to be used in the next-token prediction task and calculate the cross entropy loss during training. Both the embeddings generated after the last residual block and the logits could also be used later for downstream NLP tasks. Training is performed in time-parallel mode (Section 4.2) while autoregressive inference and a potential chat interface[4]

[4]https://github.com/BlinkDL/ChatRWKV

leverage the time-sequential mode (Section 4.3).

4.5 Gradient Stability and Layer Stacking

The RWKV architecture has been designed as a fusion of both Transformers and RNNs, offering the advantage of stable gradients and deeper architectures of Transformers compared to traditional RNNs while being efficient in inference.

Previous work has sought to tackle the problem of gradient stability in RNNs with a variety of techniques including using non-saturated activation functions (Chandar et al., 2019), gating mechanism (Gu et al., 2019), gradient clipping (Pascanu et al., 2012), and adding constraints (Kanai et al., 2017; Miller and Hardt, 2018). While these techniques have seen little success, RWKV avoids the problem inherently by utilizing softmax in conjunction with RNN-style updates.

The RWKV model features a single-step process for updating attention-like scores, which includes a time-dependent softmax operation that helps numerical stability and guards against vanishing gradients (for rigorous proof, see Appendix F). Intuitively, this operation ensures the gradient is propagated along the most relevant path. Layer normalization (Ba et al., 2016) is another key aspect of the architecture which enhances the training dynamics of deep neural networks by stabilizing gradients, addressing both vanishing and exploding gradient issues.

These design elements not only contribute to the RWKV architecture's stability and learning capabilities but enable the stacking of multiple layers in a manner that surpasses the capabilities of any existing RNN. In doing so, the model is able to capture more complex patterns across various levels of abstraction (see also Appendix G).

4.6 Harnessing Temporal Structure for Sequential Data Processing

RWKV captures and propagates sequential information through the combination of three mechanisms: recurrence, time decay and token shift.

The **recurrence** in the time-mixing block of RWKV is the basis for the model's capacity to capture intricate relationships between sequence elements and to propagate locality information through time.

The **time decay** mechanism (e^{-w} and e^{u} in equation 14), maintains sensitivity to the positional relationship between sequence elements. By gradually diminishing the influence of past information over time, the model preserves a sense of temporal locality and progression, which is essential for sequential processing. This treatment of positional information in sequential data exhibits similarities to the Attention with Linear Biases (ALiBi) model (Press et al., 2022), where the linear biases facilitate input length extrapolation. In this context, the RWKV architecture can be perceived as a trainable version of ALiBi, seamlessly incorporating positional information without the necessity for explicit encoding. It can also be seen as an extension of the gated convolution introduced in Zhai et al. (2021) to the full sequence length until a given step.

The **token shift** or **time-shift mixing**, or (diagonal arrows in Figure 3), also contributes to the model's adaptation to sequential data. By linearly interpolating between the current input and the previous time step input, the model naturally aggregates and gates information in the input channels. The overall structure of time-shift mixing bears resemblance to the causal convolution with no dilations in WaveNet (van den Oord et al., 2016), which is a classical architecture used for forecasting time series data.

4.7 Additional Optimizations

Custom Kernels To address inefficiencies in the WKV computation due to the sequential nature of the task when using standard deep learning frameworks, we implement a custom CUDA kernel so as to launch a single compute kernel in training accelerators. All other parts of the model are matrix multiplications and point-wise operations that can already be efficiently parallelized.

FFN with R gate Prior research (Tolstikhin et al., 2021; Liu et al., 2021; Yu et al., 2022) suggests that self-attention may not be as essential in Transformer-based vision tasks as previously thought. Although it provided us with some insights, replacing self-attention entirely in natural language tasks could be too drastic. In our study, we partially dismantle the attention mechanism by replacing the fixed QKV formula with KV and introducing a new time-decaying factor W. This approach enables us to incorporate token and channel-mixing components akin to MLP-mixer (Tolstikhin et al., 2021) and a gating unit R similar to gMLP (Liu et al., 2021), which enhance the performance of our RWKV model.

Small Init Embedding During the initial stage of training a transformer model (Vaswani et al.,

2017), we observe that the embedding matrix undergoes slow changes, which pose a challenge for the model to deviate from its initial noisy embedding state. To mitigate this issue, we propose an approach that involves initializing the embedding matrix with small values and subsequently applying an additional LayerNorm operation. By implementing this technique, we accelerate and stabilize the training process, enabling the training of deep architectures with post-LN components. The effectiveness of this approach is demonstrated in Figure 8, where it is shown to facilitate improved convergence by allowing the model to quickly transition away from the initially small embedding. This is achieved through small changes following a single step, which in turn lead to substantial alterations in directions and subsequently significant changes after the LayerNorm operation.

Custom Initialization Building on principles from previous works (He et al., 2016; Jumper et al., 2021), we initialize parameters to values as similar as possible to an identity mapping while breaking symmetry so there is a clean information path. Most weights are initialized to zero. No biases are used for linear layers. Specific formulas are given in Appendix D. We find the choice of initialization to be significant in convergence speed and quality (see Appendix E).

5 Evaluations

In this section, we focus on evaluating to answer the following questions:

- **RQ1:** Is RWKV competitive against quadratic transformer architectures with equal number of parameters and training tokens?
- **RQ2:** When increasing the number of parameters, does RWKV remain competitive against quadratic transformer architectures?
- **RQ3:** Does increasing parameters of RWKV yield better language modeling loss, when RWKV models are trained for context lengths that most open-sourced quadratic transformers *cannot* efficiently process?

Addressing **RQ1** and **RQ2**, from Fig. 4, we can see that RWKV is very competitive on six benchmarks (Winogrande, PIQA, ARC-C, ARC-E, LAMBADA, and SciQ) against major open source quadratic complexity transformer models: Pythia (Biderman et al., 2023), OPT (Zhang et al., 2022) and BLOOM (Scao et al., 2022). RWKV even outperforms Pythia and GPT-Neo (Black et al., 2022) in four tasks: PIQA, OBQA, ARC-E, and COPA (See details in Appendix H). For **RQ3**, Fig. 5 shows that increasing context length leads to lower test loss on the Pile, an indication that RWKV can make effective use of long contextual information.

6 Inference Experiments

We benchmark inference requirements according to size and family. Specifically, we evaluate text generation speed and memory requirements on a typical compute platforms including CPU (x86) and GPU (NVIDIA A100 80GB). For all our experiments we use float32 precision. We include all model parameters in parameter count, including both embedding and non-embedding layers. Performance under different quantization setups is left to further work. See Appendix I for more results.

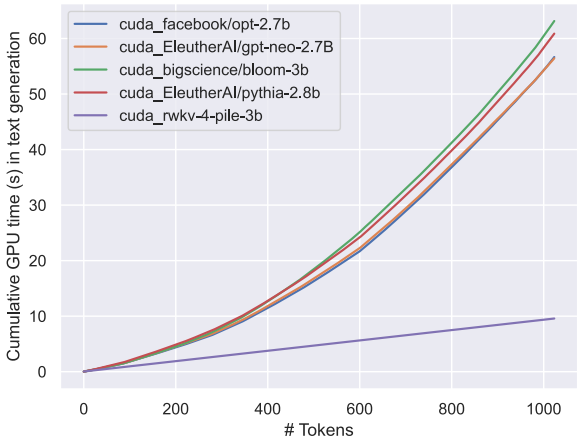


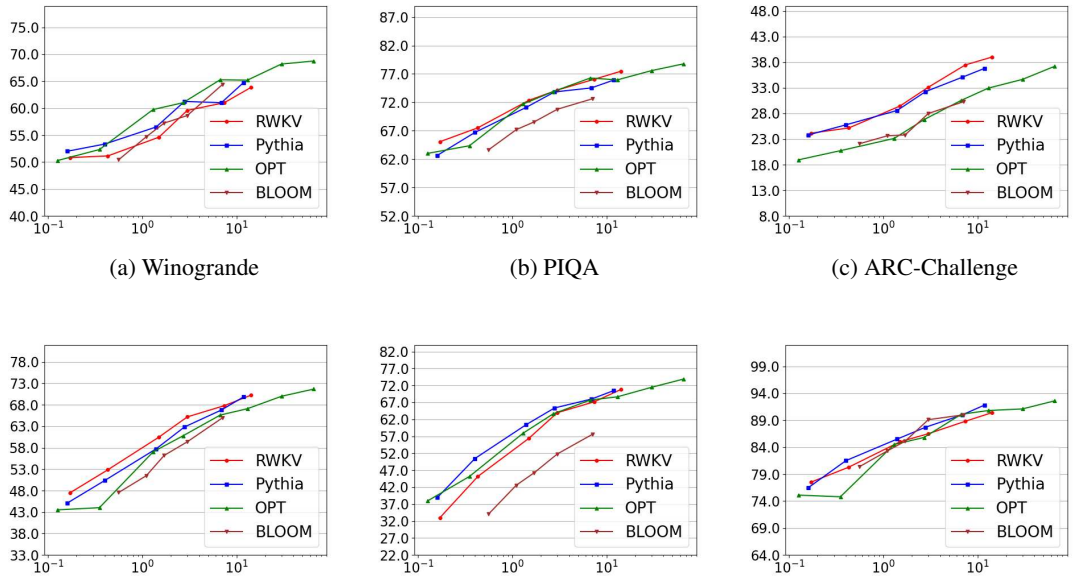
Figure 6: Cumulative time during text generation for different LLMs.

Additionally, we carried out comparative studies on RWKV-4 and ChatGPT / GPT-4, see Appendix J. They revealed that RWKV-4 is very sensitive to prompt engineering. When the prompts were adjusted from the ones used for GPT to more suitable for RWKV, the F1-measure performance increased even from 44.2% to 74.8%.

7 Future Work

There are several promising directions for future work on the RWKV architecture:

- Increasing model expressivity with enhanced time-decay formulations and exploring initial model states while maintaining efficiency.



(a) Winogrande (b) PIQA (c) ARC-Challenge

(d) ARC-Easy (e) LAMBADA (f) SciQ

Figure 4: Zero-Shot Performance: The horizontal axis is a number of parameters and the vertical axis is accuracy.

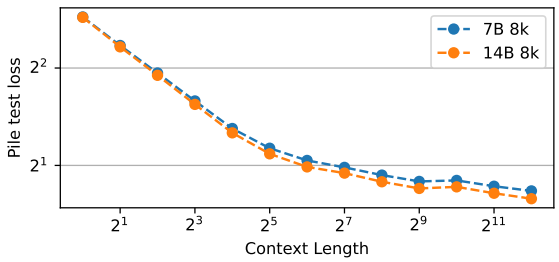


Figure 5: Increasing context length contributes to lower test loss on the Pile (Gao et al., 2020).

- Further improving RWKV computational efficiency by applying parallel scan in the wkv_t step to reduce the computational cost to $O(B\log(T)d)$.
- Investigating the application of RWKV to encoder-decoder architectures and potential replacement of cross-attention mechanism. This could have applicability seq2seq or multimodal settings, enhancing efficiency both in training and inference.
- Leveraging RWKV's state (or *context*) for interpretability, predictability in sequence data and safety. Manipulating the hidden state could also guide behavior and allow greater customizability through prompt tuning.
- Exploring fine-tuned models in specific settings for enhanced interaction with humans (Ouyang et al., 2022). Particularly interesting would be the performance under different datasets and specific use cases.
- Adapting parameter-efficient fine-tuning methods such as LoRA (Hu et al., 2022) and characterizing behavior under different quantization schemes for the proposed architecture

8 Conclusions

We introduced RWKV, a new approach to RNN models exploiting the potential of time-based mixing components. RWKV introduces several key strategies which allow it to capture locality and long-range dependencies, while addressing limitations of current architectures by: (1) replacing the quadratic QK attention by a scalar formulation with linear cost, (2) reformulating recurrence and sequential inductive biases to unlock efficient training parallelization and efficient inference, and (3) enhancing training dynamics using custom initializations.

We benchmark the proposed architecture in a wide variety of NLP tasks and show comparable performance to SoTA with reduced cost. Further experiments on expressivity, interpretability, and scaling showcase the model capabilities and draw parallels in behavior between RWKV and other LLMs.

RWKV opens a new door to scalable and efficient architectures to model complex relation-

ships in sequential data. While many alternatives to Transformers have been proposed with similar claims, ours is the first to back up those claims with pretrained models with tens of billions of parameters.

9 Limitations

While our proposed RWKV model has demonstrated promising results regarding training and memory efficiency during inference, some limitations should be acknowledged and addressed in future work. First, the linear attention of RWKV leads to significant efficiency gains but still, it may also limit the model's performance on tasks that require recalling minutiae information over very long contexts. This is due to the funneling of information through a single vector representation over many time steps, compared with the full information maintained by the quadratic attention of standard Transformers. In other words, the model's recurrent architecture inherently limits its ability to "look back" at previous tokens, as opposed to traditional self-attention mechanisms. While learned time decay helps prevent the loss of information, it is mechanistically limited compared to full self-attention.

Another limitation of this work is the increased importance of prompt engineering in comparison to standard Transformer models. The linear attention mechanism used in RWKV limits the information from the prompt that will be carried over to the model's continuation. As a result, carefully designed prompts may be even more crucial for the model to perform well on tasks.

Acknowledgements

We acknowledge EleutherAI and StabilityAI for compute access and technical support in development of RWKV. We also acknowledge the members of the RWKV Discord server for their help and work on further extending the applicability of RWKV to different domains. Finally, we thank Stella Biderman for feedback on the paper.

References

Alon Albalak, Yi-Lin Tuan, Pegah Jandaghi, Connor Pryor, Luke Yoffe, Deepak Ramachandran, Lise Getoor, Jay Pujara, and William Yang Wang. 2022. FETA: A benchmark for few-sample task transfer in open-domain dialogue. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 10936–10953, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling.

Francesco Barbieri, Jose Camacho-Collados, Luis Espinosa Anke, and Leonardo Neves. 2020. TweetEval: Unified benchmark and comparative evaluation for tweet classification. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1644–1650, Online. Association for Computational Linguistics.

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

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. *arXiv preprint arXiv:2304.01373*.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Sid Black, Leo Gao, Phil Wang, Connor Leahy, and Stella Biderman. 2022. Gpt-neo: Large scale autoregressive language modeling with mesh-tensorflow, 2021. *URL: https://doi. org/10.5281/zenodo*, 5297715.

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. 2017. Quasi-recurrent neural networks. In *ICLR*.

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. *Advances in neural information processing systems*, 33:1877–1901.

Aydar Bulatov, Yuri Kuratov, and Mikhail S. Burtsev. 2023. Scaling transformer to 1m tokens and beyond with rmt.

Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. 2022. Recurrent memory transformer. *Advances in Neural Information Processing Systems*, 35:11079–11091.

A. P. Sarath Chandar, Chinnadhurai Sankar, Eugene Vorontsov, Samira Ebrahimi Kahou, and Yoshua Bengio. 2019. Towards non-saturating recurrent units for modelling long-term dependencies. In *AAAI Conference on Artificial Intelligence*.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. 2020. Rethinking attention with performers.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Deep Learning and Representation Learning Workshop*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. In *arXiv:1803.05457*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. In *arXiv*, volume abs/2110.14168.

Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Re. 2022a. Flashattention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*.

Tri Dao, Daniel Y Fu, Khaled K Saab, Armin W Thomas, Atri Rudra, and Christopher Ré. 2022b. Hungry hungry hippos: Towards language modeling with state space models. *arXiv preprint arXiv:2212.14052*.

Dorottya Demszky, Dana Movshovitz-Attias, Jeongwoo Ko, Alan S. Cowen, Gaurav Nemade, and Sujith Ravi. 2020. Goemotions: A dataset of fine-grained emotions. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 4040–4054. Association for Computational Linguistics.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. 2020. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Albert Gu, Karan Goel, and Christopher Ré. 2022. Efficiently modeling long sequences with structured state spaces. In *The International Conference on Learning Representations (ICLR)*.

Albert Gu, Çaglar Gülçehre, Tom Le Paine, Matthew W. Hoffman, and Razvan Pascanu. 2019. Improving the gating mechanism of recurrent neural networks. *ArXiv*, abs/1910.09890.

Mandy Guo, Joshua Ainslie, David C Uthus, Santiago Ontanon, Jianmo Ni, Yun-Hsuan Sung, and Yinfei Yang. 2022. Longt5: Efficient text-to-text transformer for long sequences. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 724–736.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Sepp Hochreiter. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116.

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

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. 2022. Training compute-optimal large language models.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. 2019. Deep learning for time series classification: a review. *Data mining and knowledge discovery*, 33(4):917–963.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. 2021. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pages 4651–4664. PMLR.

Hanhwi Jang, Joonsung Kim, Jae-Eon Jo, Jaewon Lee, and Jangwoo Kim. 2019. Mnnfast: A fast and scalable system architecture for memory-augmented neural networks. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 250–263.

Matt Gardner Johannes Welbl Nelson F. Liu. 2017. Crowdsourcing multiple choice science questions. In *DOI:10.18653/v1/W17-4413*.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *ACL*.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, and et al. 2021. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589.

Sekitoshi Kanai, Yasuhiro Fujiwara, and Sotetsu Iwamura. 2017. Preventing gradient explosions in gated recurrent units. In *NIPS*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR.

Nikita Kitaev, L. Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *ArXiv*, abs/2001.04451.

Jan Kocoń, Igor Cichecki, Oliwier Kaszyca, Mateusz Kochanek, Dominika Szydło, Joanna Baran, Julita Bielaniewicz, Marcin Gruza, Arkadiusz Janz, Kamil Kanclerz, Anna Kocoń, Bartłomiej Koptyra, Wiktoria Mieleszczenko-Kowszewicz, Piotr Miłkowski, Marcin Oleksy, Maciej Piasecki, Łukasz Radliński, Konrad Wojtasik, Stanisław Woźniak, and Przemysław Kazienko. 2023. Chatgpt: Jack of all trades, master of none.

Jan Kocoń, Piotr Miłkowski, and Monika Zaśko-Zielińska. 2019. Multi-level sentiment analysis of polemo 2.0: Extended corpus of multi-domain consumer reviews. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 980–991.

Phong Le and Willem Zuidema. 2016. Quantifying the vanishing gradient and long distance dependency problem in recursive neural networks and recursive lstms. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 87–93.

Tao Lei, Yu Zhang, Sida I. Wang, Hui Dai, and Yoav Artzi. 2018. Simple recurrent units for highly parallelizable recurrence. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4470–4481, Brussels, Belgium. Association for Computational Linguistics.

Hanxiao Liu, Zihang Dai, David R. So, and Quoc V. Le. 2021. Pay attention to mlps.

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*, 34:2441–2453.

Xuezhe Ma, Chunting Zhou, Xiang Kong, Junxian He, Liangke Gui, Graham Neubig, Jonathan May, and Luke Zettlemoyer. 2023. Mega: Moving average equipped gated attention. In *ICLR*.

Eric Martin and Chris Cundy. 2017. Parallelizing linear recurrent neural nets over sequence length. *ArXiv*, abs/1709.04057.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in GPT. *Advances in Neural Information Processing Systems*, 36.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*.

John Miller and Moritz Hardt. 2018. Stable recurrent models. *arXiv: Learning*.

Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. 2016. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 839–849.

OpenAI. 2022. Introducing chatgpt. `https://openai.com/blog/chatgpt`.

OpenAI. 2023. Gpt-4 technical report.

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. 2023. Resurrecting recurrent neural networks for long sequences. *arXiv preprint arXiv:2303.06349*.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernandez. 2016. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534, Berlin, Germany. Association for Computational Linguistics.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2012. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library.

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. 2023. Hyena hierarchy: Towards larger convolutional language models. *arXiv preprint arXiv:2302.10866*.

Ofir Press, Noah A. Smith, and Mike Lewis. 2022. Train short, test long: Attention with linear biases enables input length extrapolation. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*.

Ilan Price, Jordan Gifford-Moore, Jory Flemming, Saul Musker, Maayan Roichman, Guillaume Sylvain, Nithum Thain, Lucas Dixon, and Jeffrey Sorensen. 2020. Six attributes of unhealthy conversations. In *Proceedings of the Fourth Workshop on Online Abuse and Harms*, pages 114–124, Online. Association for Computational Linguistics.

Markus N. Rabe and Charles Staats. 2022. Self-attention does not need $o(n^2)$ memory.

Melissa Roemmele, Cosmin Adrian Bejan, , and Andrew S. Gordon. 2018. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *AAAI*.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2022. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*.

Ramsha Siddiqui. 2019. SARCASMANIA: Sarcasm Exposed! `http://www.kaggle.com/rmsharks4/sarcasmania-dataset`. [Online; accessed 02-February-2023].

David R. So, Wojciech Manke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V. Le. 2021. Primer: Searching for efficient transformers for language modeling. *CoRR*, abs/2109.08668.

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

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2022. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28.

Ilya O. Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. 2021. Mlp-mixer: An all-mlp architecture for vision. *CoRR*, abs/2105.01601.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. *ArXiv*, abs/1609.03499.

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*, volume 30. Curran Associates, Inc.

David Vilares and Carlos Gómez-Rodríguez. 2019. Head-qa: A healthcare dataset for complex reasoning. In *ACL*.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

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

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24.

Ellery Wulczyn, Nithum Thain, and Lucas Dixon. 2017. Ex machina: Personal attacks seen at scale. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, pages 1391–1399. ACM.

Weihao Yu, Mi Luo, Pan Zhou, Chenyang Si, Yichen Zhou, Xinchao Wang, Jiashi Feng, and Shuicheng Yan. 2022. Metaformer is actually what you need for vision.

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

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *ACL*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2020. Winogrande: An adversarial winograd schema challenge at scale. In *ACL*.

Shuangfei Zhai, Walter Talbott, Nitish Srivastava, Chen Huang, Hanlin Goh, Ruixiang Zhang, and Josh Susskind. 2021. An attention free transformer.

Sheng Zhang, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, Kevin Duh, and Benjamin Van Durme. 2018. Record: Bridging the gap between human and machine commonsense reading comprehension. In *arXiv:1810.12885*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

A Author Contributions

Bo Peng Original RWKV idea, original code, performance optimizations, original experiments, and trained RWKV models from 0.1B to 14B.

Eric Alcaide Manuscript (initial draft sections 1, 2; sections 4, 7 and 8; revision and proofreading; final version). Figures (2, 3, 4, 7). Experiments section 6. Appendices D, I. Contributions to Appendix K.

Quentin Anthony Led writing the paper. Manuscript (initial draft sections 1, 2, 3; revision and proofreading; final version).

Zhenyuan Zhang Manuscript (revision and proofreading) Figure 3. Experiments Appendix G. Contributions to Appendices B and K.

Kranthi Kiran GV Manuscript (sections 2 and 5; contributions to section 3; revision and proofreading). Tables 3 and 4. Appendix C.

Xiangru Tang Manuscript (sections 2 and 3; contributions to abstract; revision and proofreading). Contributions to Appendix K.

Matteo Grella Manuscript (sections 4.5, 4.6, 8; contributions to sections 1, 7 and 9; proofreading and revision). Contributions to Appendix B.

Ferdinand Mom Manuscript (contributions to section 1, 2, 4.3, 4.6; proofreading and revision). Contributions to Appendix B.

Atsushi Saito Manuscript (sections 3 and 5; contributions to section 2). Figures 1a , 1b, 1c. Contributions to Appendix H

Krishna Sri Ipsit Mantri Figure 4

Rui-Jie Zhu Tables 1 and 5. Experiments for table 5.

Peng Zhou Contributions to Table 5.

Qihang Zhao Manuscript (proofreading and revision). Contributions to Table 5.

Xuzheng He Manuscript (contributions to section 3; proofreading and revision). Contributions to Figures 1, 7. Appendix G. Contributions to appendix F.

Hayden Lau Manuscript (contributions to section 1; proofreading and revision). Contributions to Appendix K.

Michael Chung Manuscript (contributions to section 4.6; proofreading and revision).

Haowen Hou Figure 8. Appendix E

Jiaming Kong Manuscript (revision and proofreading). Appendix F.

Johan S. Wind RWKV performance optimizations (CUDA), Contributions to Appendix C.

Jian Zhu Manuscript (section 2; proofreading and revision). Figures 3 and 5.

Huanqi Cao Manuscript (contributions to 4.2 and 4.3; proofreading and revision). Experiments for Appendix G.

Samuel Arcadinho Contributions to Figures 6, 10, and 11. Contributions to Appendix I.

Xin Cheng Manuscript (proofreading and revision). Contributions to Appendix K, H.

Alon Albalak Manuscript (abstract and sections 1, 9; proofreading and revision).

Jan Kocon Manuscript (sections 1; proofreading and revision). Contributions to Appendix J.

Przemysław Kazienko Manuscript (section 6; proofreading and revision). Contributions Appendix J.

Ruichong Zhang Manuscript (proofreading and revision); Contributions to Figure 5 and Appendix K.

Stanisław Woźniak Appendix J.

Bartłomiej Koptyra Contributions to Appendix J.

B Time-Mixing Block as an RNN Cell

As stated in 4.3, the RWKV time-mixing block can be formulated as an RNN, as the WKV computation can be written in such a recursive form:

$$a_0, b_0 = 0, \tag{19}$$

$$wkv_t = \frac{a_{t-1} + e^{u+k_t} v_t}{b_{t-1} + e^{u+k_t}}, \tag{20}$$

$$a_t = e^{-w} a_{t-1} + e^{k_t} v_t, \tag{21}$$

$$b_t = e^{-w} b_{t-1} + e^{k_t}. \tag{22}$$

The dataflow of the RNN-like time-mixing is shown in Fig. 7, where the hidden states h is the numerator-denominator tuple (a, b).

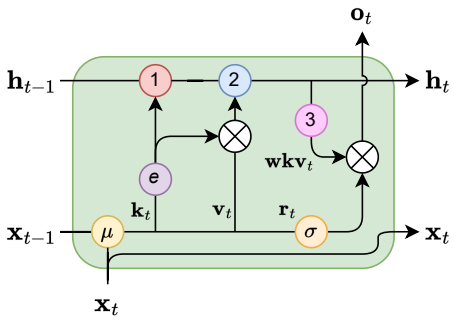


Figure 7: RWKV time-mixing block formulated as an RNN cell. Color codes: yellow (μ) denotes the token shift, red (1) denotes the denominator, blue (2) denotes the numerator, pink (3) denotes the fraction computations in 14. h denotes the numerator-denominator tuple (a, b).

To avoid overflow in calculating e^{k_t}, a numerical trick is used in the official implementation. Note that

$$a_1 = e^{-w} a_0 + e^{k_0} v_0 = e^{k_0} v_0, \tag{23}$$

$$b_1 = e^{-w} b_0 + e^{k_0} = e^{k_0}, \tag{24}$$

and we set $a'_1 = v_0, b'_1 = 1, p_0 = k_0$, where p_{t-1} stores the shared exponents of a_t and b_t. Now the above recursion can be converted into a numerical safe version, for each time step $t > 1$:

$$q := \max(p_{t-1}, u + k_t), \tag{25}$$

$$a^*_t = e^{p_{t-1}-q} a'_{t-1} + e^{u+k_t-q} v_t, \tag{26}$$

$$b^*_t = e^{p_{t-1}-q} b'_{t-1} + e^{u+k_t-q}, \tag{27}$$

$$wkv_t = \frac{a^*_t}{b^*_t}. \tag{28}$$

The update to a'_t, b'_t and their shared exponent are also carried out in similar fashion:

$$q := \max(p_{t-1} - w, k_t), \tag{29}$$

$$a'_t = e^{p_{t-1}-w-q} a'_{t-1} + e^{k_t-q} v_t, \tag{30}$$

$$b'_t = e^{p_{t-1}-w-q} b'_{t-1} + e^{k_t-q}, \tag{31}$$

$$p_t = q. \tag{32}$$

C Parameter and FLOP Count for the RWKV Models

The following section provides an overview of the different RWKV model architectures along with their respective parameter and FLOP counts in Table 2.

Name	Layers	Model Dimension	Parameters	FLOPs per token
169 M	12	768	1.693×10^8	2.613×10^8
430 M	24	1024	4.304×10^8	7.573×10^8
1.5 B	24	2048	1.515×10^9	2.823×10^9
3 B	32	2560	2.985×10^9	5.710×10^9
7 B	32	4096	7.393×10^9	1.437×10^{10}
14 B	40	5120	1.415×10^{10}	2.778×10^{10}

Table 2: RWKV model architectures and associated FLOP counts

The number of parameters for each model is computed using the formula: $\#parameters = 2VD + 13D^2L + D(11L + 4)$ where V = 50277 is the vocabulary size, D represents the Model Dimension and L corresponds to the number of layers.

FLOPs is for a forward pass for one token. It was calculated as $6(VD + 13D^2L)$, which is the twice (add and multiply) the number of parameters in linear layers. The backwards pass FLOPs can be approximated as twice that of the forward pass. So

the total is $6(VD + 13D^2L)$ per token for training (3x fw FLOPs). It is noteworthy that FLOPs are independent of the context length, unlike regular transformers. The FLOP approximations in this paper are in line with the methodology used by Kaplan et al. (2020).

Alternative approximations for FLOPs include doubling the parameters which yields similar results within 2% for 14B and a 30% discrepancy for 169M variant. Another approximation is based on the number of non-embedding parameters multiplied by 2. This gives $2(VD + 13D^2L + D(11L + 4))$ resulting in 1.6% more FLOPs for 14B model and 8% more FLOPs for 169M model.

D Parameter initializations

We describe the specific parameter initializations below and motivate the design choices. Parameters belonging to residual blocks are often adjusted by layer depth and total number of layers. Let $\#$ denote the vocabulary size, s denote the embedding dimension, d denote the hidden size (we use $d = 4s$), L the number of layers, l the layer index (from 0 to L-1), we use the following initializations:

- Embeddings are initialized to $\mathcal{U}(\pm 1e\text{-}4)$ as explained in 4.7
- For the channel-mixing blocks (11), μ_{k_i} and μ_{r_i} are initialized to $(\frac{i}{s})^{1-\frac{l}{L}}$
- For the time-mixing blocks (16), initializations are $\mu_{k_i} = (\frac{i}{s})^{1-\frac{l}{L}}$, $\mu_{v_i} = (\frac{i}{s})^{1-\frac{l}{L}} + \frac{0.3l}{L-1}$ and $\mu_{r_i} = 0.5(\frac{i}{s})^{1-\frac{l}{L}}$
- w_i (14), also known as "time decay", is initialized to $-5 + 8 \cdot (\frac{i}{d-1})^{0.7+\frac{1.3l}{L-1}}$. Intuitively, it is the discount factor applied to previous tokens over time.
- u_i (14), also known as "bonus", is set to $0.5(((i+1) \mod 3) - 1) + \log 0.3$. It is the special weighting applied to the current token in equation 14. The alternating zigzag pattern initially creates subtle variations in the tensor elements, which are intended to help the model treat different dimensions of the embedding distinctively.
- W_o (15) (time-mixing) and W_v (channel-mixing) are initialized to $\mathcal{N}(0, \sqrt{\frac{d}{s} = 2})$
- All W_r, W_k, W_v weights are initialized to 0 so the model can start learning from the beginning without noisy signals.
- All LayerNorm weights start from 1 and biases from 0.

E Small Init Embedding

This section presents experimental validation of small initialization embedding. The experimental setup is as follows. In the baseline configuration, the parameters are initialized using a normal distribution with a mean of 0.0 and a standard deviation of 0.02, which is a commonly used initialization method in models like BERT and GPT. On the other hand, in the small initialization of the embedding (small init emb) experiment, the parameters are initialized using a uniform distribution with a range of 1e-4, which is slightly different from RWKV where a normal distribution with a standard deviation of 1e-4 is used. However, this difference is negligible and does not affect our conclusions. The experiments were conducted with a batch size of 400. As depicted in the figure 8, the loss curve for the small init emb exhibits a faster rate of decrease and convergence compared to the traditional initialization using a normal distribution.

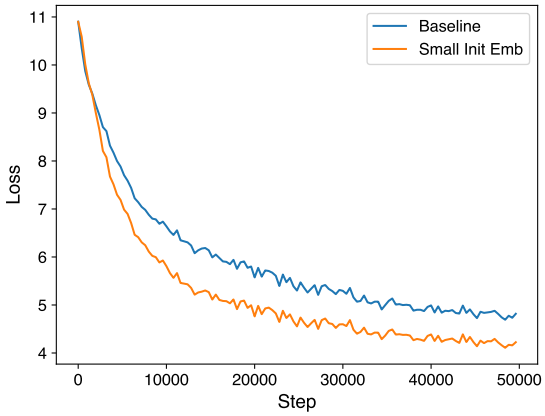


Figure 8: Effect of small initialization embedding.

F Gradient Stability in RWKV

In this section, we present a mathematical description of the gradient stability property in RWKV, focusing specifically on the time-mixing block. By gradient stability we mean that if the inputs x_t are bounded and the model parameters are fixed, then the gradients with respect to W_k and W_v are uniformly bounded for all T (thus not exploding). Consequently, we can control the amount each x_t contributes to the gradient at T in a naturally decaying fashion by the weight decay mechanism w

(thus not vanishing unless desired).

First, we make the simplification that there are no token shifts, this will not affect the final conclusion. In this scenario, wkv_T can be written as

$$wkv_T = \frac{\sum_{t=1}^{T} K_t^e v_t}{\sum_{t=1}^{T} K_t^e} = \mathrm{E}(v_t) = \frac{\mathrm{S}(v_t)}{\mathrm{S}(1)}, \quad (33)$$

where

$$v_t = W_v x_t, \quad \frac{\partial (v_t)_i}{\partial (W_v)_{i,j}} = (x_t)_j,$$

$$K_t^e = e^{W_k x_t + w_{T,t}}, \quad \frac{\partial (K_t^e)_i}{\partial (W_k)_{i,j}} = (x_t)_j (K_t^e)_i,$$

and $\mathrm{S}(\cdot)$ and $\mathrm{E}(\cdot)$ are shorthand for denoting sums and averages over weights K_t^e.

The loss function at position T can be written as

$$L_T = l(f(wkv_T), y_T). \quad (34)$$

Because wkv_T relates to $(W_k)_{i,j}$ and $(W_v)_{i,j}$ only through the i-th channel $(wkv_T)_i$, we have

$$\frac{\partial L_T}{\partial (W_v)_{i,j}} = \frac{\partial L_T}{\partial (wkv_T)_i} \frac{\partial (wkv_T)_i}{\partial (W_v)_{i,j}}. \quad (35)$$

The first part of above equation contains trivial operations like output layers, and other layers of time-mixing, which can be proven inductively. The second part of above equation can be bounded as

$$\begin{aligned} \left| \frac{\partial (wkv_T)_i}{\partial (W_v)_{i,j}} \right| &= \left| \frac{\partial \mathrm{E}_i[(v_t)_i]}{\partial (W_v)_{i,j}} \right| \\ &= |\mathrm{E}_i[(x_t)_j]| \leq \max_t |(x_t)_j|, \end{aligned} \quad (36)$$

which is irrelevant to T. Similarly,

$$\begin{aligned} \frac{\partial (wkv_T)_i}{\partial (W_k)_{i,j}} &= \partial \frac{\mathrm{S}_i[(v_t)_i]}{\mathrm{S}_i(1)} / \partial (W_k)_{i,j} \\ &= \frac{\mathrm{S}_i[(x_t)_j (v_t)_i]}{\mathrm{S}_i(1)} - \frac{\mathrm{S}_i[(x_t)_j]\mathrm{S}_i[(v_t)_i]}{\mathrm{S}_i(1)^2} \\ &= \mathrm{E}_i[(x_t)_j (v_t)_i] - \mathrm{E}_i[(x_t)_j]\mathrm{E}_i[(v_t)_i] \\ &= \mathrm{cov}_i((x_t)_j, (v_t)_i) \end{aligned} \quad (37)$$

can also be bounded. Note that wkv's softmax operation contains at least two non-zero terms (u and w), so the above "covariance" will not degenerate into 0.

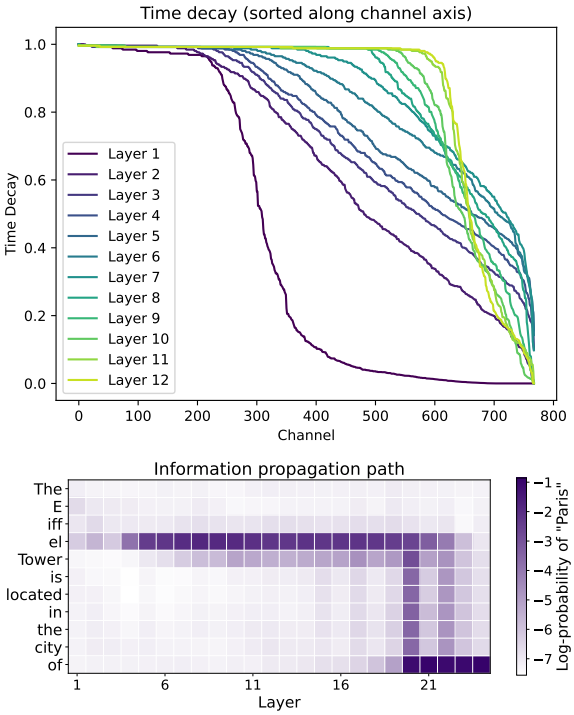


Figure 9: Model behavior visualizations of the RWKV model.

G Model Behavior Visualization

In Figure 9, we present visualizations of some behavior of the RWKV model.

The top plot illustrates the time decays (e^{-w}) in each layer of the RWKV-169M model, sorted along the channel axis. Notably, several decays in the last layers are very close or equal to one, implying that certain information is preserved and propagated throughout the model's temporal context. Meanwhile, many decays in the initial layer are close to zero, which corresponds to local operations in wkv (14), likely to be associated with tasks such as text parsing or lexical analysis. (Note that the local operations in wkv is due to the extra parameter u, when e^{-w} is degenerated into 0.) These patterns of time decays are partly learned, but also come from parameter initialization as it speeds up training.

The bottom plot shows the information retrieval and propagation path in the RWKV-430M model. The experiment follows the *causal trace* method introduced by Meng et al. (2022), where we

1. Run the model once, and record all states and activation of each layer during the computation;
2. Corrupt the input embeddings of the subject using noise ("The Eiffel Tower" in this example);

3. Restore the states and activation of a certain layer at a certain token during the computation, and record the log-probability of the model outputting the correct answer ("Paris").

Unlike transformers, RWKV relies on recursive propagation of information in the time dimension. In this case, the fact that "the Eiffel Tower is located in Paris" is retrieved in layer 4. It is then passed down to the subsequent layers. In layer 20, mostly, the information is propagated through time until reaching where it is needed. Finally, it is passed down to the last layer for outputting the answer.

H Evaluation Details

The results for following tasks are in Table 3 and 4.

Tasks:

- LAMBADA (Paperno et al., 2016). A benchmark dataset that evaluates the model's contextual reasoning and language comprehension abilities by presenting context-target pairs, where the objective is to predict the most probable target token.
- PIQA (Bisk et al., 2020). A benchmark for the task of physical common sense reasoning, which consists of a binary choice task that can be better understood as a set of two pairs, namely (Goal, Solution).
- HellaSwag (Zellers et al., 2019) A novel benchmark for commonsense Natural Language Inference (NLI) which is build by adversarial filtering against transformer models.
- Winogrande (Zellers et al., 2020) A dataset designed to evaluate the acquisition of common sense reasoning by neural language models, aiming to determine whether we are accurately assessing the true capabilities of machine common sense.
- StoryCloze (Mostafazadeh et al., 2016) A benchmark to present a novel approach to assess comprehension of narratives, narrative generation, and script acquisition, focusing on commonsense reasoning.
- ARC Challenge (Clark et al., 2018) A dataset designed for multiple-choice question answering, encompassing science exam questions ranging from third grade to ninth grade.
- ARC Easy An easy subset of ARC.
- HeadQA (Vilares and Gómez-Rodríguez, 2019) A benchmark consisting of graduate-level questions encompassing various fields such as medicine, nursing, biology, chemistry, psychology, and pharmacology.
- OpenBookQA (Mihaylov et al., 2018) A QA dataset to evaluate human comprehension of a subject by incorporating open book facts, scientific knowledge, and perceptual common sense, drawing inspiration from open book exams.
- SciQ (Johannes Welbl Nelson F. Liu, 2017) A multiple-choice QA dataset which was created using an innovative approach to gather well-crafted multiple-choice questions that are focused on a specific domain.
- TriviaQA (Joshi et al., 2017) A QA-IR dataset which is constituted of triples of questions, answers, supporting evidence, and independently collected evidence documents, with an average of six documents per question for reliable sources.
- ReCoRD (Zhang et al., 2018) A benchmark for evaluating commonsense reasoning in reading comprehension by generating queries from CNN/Daily Mail news articles and requiring text span answers from corresponding summarizing passages.
- COPA (Roemmele et al., 2018) A dataset to evaluate achievement in open-domain commonsense causal reasoning.
- MMMLU (Hendrycks et al., 2021) A multitask dataset for 57 tasks containing elementary mathematics, US history, computer science, law, etc.

I Inference results

Figures 10 and 11 illustrate, respectively, the results on time (s) and memory (RAM, VRAM) requirements for LLM inference in *float32* precision. We benchmark the following model families and sizes:

- **RWKV**: 169m, 430m, 1.4b, 3b, 7b, 14b
- **Bloom** (Scao et al., 2022): 560m, 1b, 3b
- **OPT** (Zhang et al., 2022): 125m, 350m, 1.3b, 2.7b, 6.7b, 13b
- **GPT-Neo** (Black et al., 2022): 125m, 1.3b, 2.7b
- **Pythia** (Biderman et al., 2023): 160m, 410m, 1.4b, 2.8b, 6.7b, 12b

Missing models in are due to Out Of Memory (OOM) errors. A comparison at 512 tokens is shown in Figure 11 as some large transformer models produced an OOM when inferencing longer se-

Model	Params B	PIQA acc	StoryCloze acc	HellaSwag acc_norm	WinoGrande acc	ARC-e acc	ARC-c acc_norm	OBQA acc_norm
RWKV-4	0.17	**65.07**	**58.79**	**32.26**	50.83	**47.47**	**24.15**	**29.60**
Pythia	0.16	62.68	58.47	31.63	**52.01**	45.12	23.81	29.20
GPT-Neo	0.16	63.06	58.26	30.42	50.43	43.73	23.12	26.20
RWKV-4	0.43	**67.52**	**63.87**	**40.90**	51.14	**52.86**	25.17	**32.40**
Pythia	0.40	66.70	62.64	39.10	**53.35**	50.38	**25.77**	30.00
GPT-Neo	0.40	65.07	61.04	37.64	51.14	48.91	25.34	30.60
RWKV-4	1.5	**72.36**	**68.73**	**52.48**	54.62	**60.48**	**29.44**	**34.00**
Pythia	1.4	71.11	67.66	50.82	**56.51**	57.74	28.58	30.80
GPT-Neo	1.4	71.16	67.72	48.94	54.93	56.19	25.85	33.60
RWKV-4	3.0	**74.16**	**70.71**	**59.89**	59.59	**65.19**	**33.11**	**37.00**
Pythia	2.8	73.83	**70.71**	59.46	**61.25**	62.84	32.25	35.20
GPT-Neo	2.8	72.14	69.54	55.82	57.62	61.07	30.20	33.20
RWKV-4	7.4	**76.06**	73.44	65.51	61.01	**67.80**	**37.46**	**40.20**
Pythia	6.9	74.54	72.96	63.92	61.01	66.79	35.07	38.00
GPT-J	6.1	75.41	**74.02**	**66.25**	**64.09**	66.92	36.60	38.20
RWKV-4	14.2	**77.48**	**76.06**	**70.65**	63.85	70.24	**38.99**	**41.80**
GPT-level*	14.2	76.49	74.97	68.72	**65.14**	**70.77**	37.99	39.27
Pythia (c.f.)	11.8	75.90	74.40	67.38	64.72	69.82	36.77	38.80
GPT-NeoX (c.f.)	20.6	77.69	76.11	71.42	65.98	72.69	40.44	40.20

Table 3: Zero-Shot Performance of the model on Common Sense Reasoning Tasks. * Interpolation of Pythia and GPT-Neo models

Model	Params B	LAMBADA ppl	LAMBADA acc	headQA acc_norm	sciq acc	triviaQA acc	ReCoRD em	COPA acc
RWKV-4	0.17	**29.33**	32.99	25.78	**77.50**	1.26	62.03	**66.00**
Pythia	0.16	24.38	**38.97**	**25.82**	76.50	**1.31**	**66.32**	62.00
GPT-Neo	0.16	30.27	37.36	25.16	76.60	1.18	64.92	64.00
RWKV-4	0.43	13.04	45.16	**27.32**	80.30	**2.35**	70.48	65.00
Pythia	0.40	**11.58**	**50.44**	25.09	**81.50**	2.03	**75.05**	**67.00**
GPT-Neo	0.40	13.88	47.29	26.00	81.10	1.38	73.79	65.00
RWKV-4	1.5	7.04	56.43	**27.64**	85.00	**5.65**	76.97	**77.00**
Pythia	1.4	**6.58**	**60.43**	27.02	**85.50**	5.52	**81.43**	73.00
GPT-Neo	1.4	7.5	57.25	27.86	86.00	5.24	80.62	69.00
RWKV-4	3.0	5.25	63.96	28.45	86.50	**11.68**	80.87	**82.00**
Pythia	2.8	**4.93**	**65.36**	**28.96**	**87.70**	9.63	85.10	77.00
GPT-Neo	2.8	5.63	62.22	27.17	89.30	4.82	**83.80**	80.00
RWKV-4	7.4	4.38	67.18	**31.22**	88.80	**18.30**	83.68	**85.00**
Pythia	6.9	4.3	**67.98**	28.59	90.00	15.42	86.44	**85.00**
GPT-J	6.1	**4.1**	68.31	28.67	**91.50**	16.74	**87.71**	83.00
RWKV-4	14.2	**3.86**	70.83	**32.64**	90.40	**24.58**	85.67	**85.00**
GPT-level*	14.2	3.81	**70.94**	31.03	**92.20**	22.37	87.89	82.66
Pythia (c.f.)	11.8	3.89	70.44	30.74	91.80	20.57	87.58	82.00
GPT-NeoX (c.f.)	20.6	3.64	71.94	31.62	93.00	25.99	88.52	84.00

Table 4: Zero-Shot Performance of various models on different tasks. * Interpolation of Pythia and GPT-Neo models

Method	L	d	T	Train bpc	Test bpc	Time Complexity	Space Complexity
Transformer	12	512	1024	0.977	1.137	$O(T^2 d)$	$O(T^2 + Td)$
Transformer	24	256	1024	1.039	1.130	$O(T^2 d)$	$O(T^2 + Td)$
Reformer	12	512	1024	1.040	1.195	$O(T \log T d)$	$O(T \log T + Td)$
Synthesizer	12	512	1024	0.994	1.298	$O(T^2 d)$	$O(T^2 + Td)$
Linear Transformer	12	512	1024	0.981	1.207	$O(Td^2)$	$O(Td + d^2)$
Performer	12	512	1024	1.002	1.199	$O(Td^2 \log d)$	$O(Td \log d + d^2 \log d)$
AFT-simple	12	512	1024	0.854	1.180	$O(Td)$	$O(Td)$
RWKV-RNN	6	512	1024	0.720	-	$O(\mathbf{Td})$	$O(\mathbf{d})$

Table 5: Enwik8 results, measured in bits per character (bpc): the lower the better. Baseline comparisons are made with Reformer (Kitaev et al., 2020), Synthesizer (Tay et al., 2020) (the best performing dense version), Linear Transformer (Katharopoulos et al., 2020), Performer (Choromanski et al., 2020). L, d, and T denote the number of blocks (network depth), dimension of features, and sequence length, respectively. Both Linear Transformer and Performer are implemented with customized CUDA kernels (github.com/idiap/fast-transformers), and all other models are implemented in native Pytorch.

quences. For GPU experiments, we use an NVIDIA A100 with 80GB of VRAM. For CPU experiments, we use an AMD EPYC processor with 30 CPU cores and 200 GiB RAM.

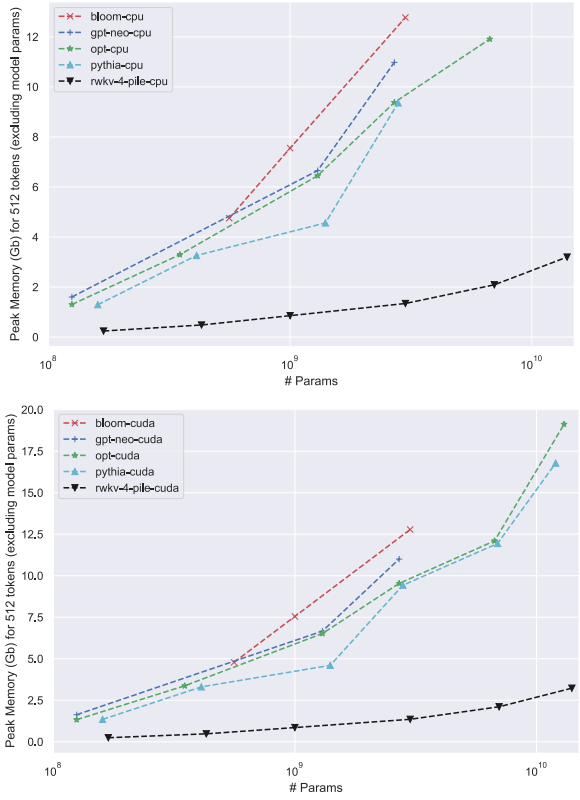


Figure 10: Text generation inference memory (CPU RAM, GPU VRAM) for LLMs. Model parameters are not accounted.

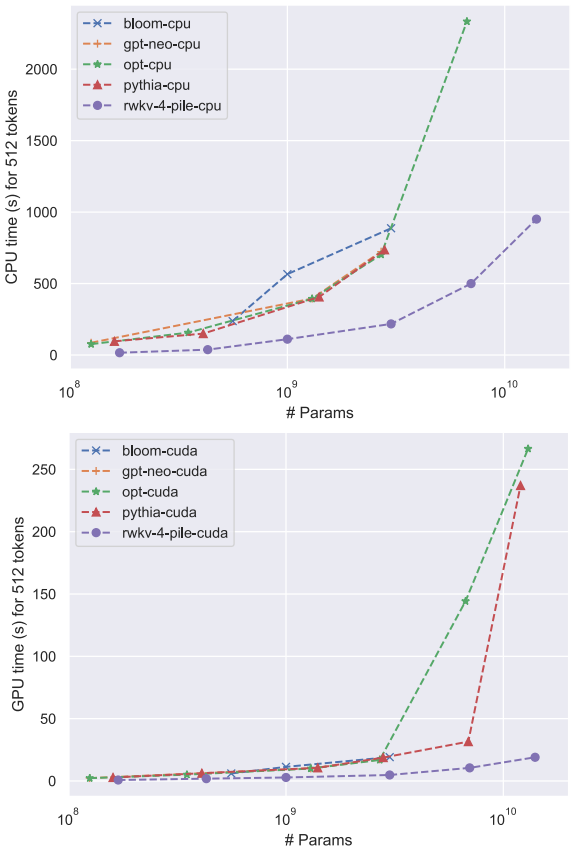


Figure 11: Text generation inference time for LLMs.

Task Name	Measure type	ChatGPT [%]	GPT-4 [%]	RWKV-4 GPT [%]	RWKV-4 changed [%]	SOTA [%]
RTE	F1 Macro	88.1	**91.3**	44.2	74.8	92.1
WNLI	Accuracy	81.7	**91.6**	47.9	49.3	97.9
GoEmotions	F1 Macro	**25.6**	23.1	7.9	7.9	52.8
PolEmo2	F1 Macro	**44.1**	41.0	38.2	40.9	76.4

Table 6: ChatGPT, GPT-4 and RWKV-4-Raven-14B reasoning performance comparison in RTE (Wang et al., 2019), WNLI (Wang et al., 2018), GoEmotions (Demszky et al., 2020), and PolEmo2 (Kocoń et al., 2019) benchmarks. SOTA is provided as a supplementary reference.

J Importance of prompt construction and comparison to GPT models

Inspired by article (Kocoń et al., 2023), we compared the zero-shot performance of the RWKV-4-Raven-14B with ChatGPT (access in February 2023) and GPT-4 using several known NLP tasks, i.e., recognizing textual entailment (RTE), Winograd Natural Language Inference (WNLI), and recognizing emotions elicited in readers (GoEmotions and PolEmo2). Each model got the same prompts manually chosen to receive proper responses from the ChatGPT model. As shown in Tab. 6, RWKV performs significantly worse than ChatGPT and GPT-4 in specific task performance. We suspect that this disparity is likely caused by the choice of prompts used to generate the answers. Given that prompts are in natural language and do not consider that RWKV is an RNN, so it can not look back inside an instruction.

When the instruction style was adapted to respect that RNNs is not capable for *retrospective processing*, quality on some datasets increased significantly (ex. for RTE (Wang et al., 2019) F1 Macro increased from 44.2% to 74.8%). We hypothesize that RWKV models are more sensitive to the position of the components in the context, as RNN-based architectures cannot look back and readjust the weight of previous information. For better performance, desired information should be after the question. Example of ChatGPT prompt to RTE:

Having premise <here is a premise> judge if the following hypothesis <here is a hypothesis> are logically connected with the premise? Answer "entailment" if yes, or "not_entailment" if no.

RWKV prompt taking into account the characteristics of the RNN:

Can you tell me if the hypothesis is entailment or is not entailment to the premise?

Task Name	Measure type	ChatGPT [%]	RWKV-4 adapted [%]	SOTA [%]
Aggression	F1 Macro	**69.10**	56.66	74.45
MathQA	Accuracy	71.40	**80.69**	83.20
Sarcasm	F1 Macro	49.88	**50.96**	53.57
TweetSent	F1 Macro	**63.32**	52.50	72.07
Unhealthy	F1 Macro	**45.21**	43.30	50.96

Table 7: ChatGPT and RWKV-4-Raven-14B performance comparison in Aggresion (Wulczyn et al., 2017), Sarcasm (Siddiqui, 2019), Unhealthy (Price et al., 2020), MathQA (Cobbe et al., 2021), and TweetSent (Barbieri et al., 2020) benchmarks. SOTA is provided as a supplementary reference.

premise: <here is a premise>
hypothesis: <here is a hypothesis>

While separating the instruction from the input is relatively easy to do, other aspects of prompt engineering are harder to quantify. Testing the approach of stating the input after the question on multiple other tasks, shown in tab. 7, suggests that better prompts might reduce the disparity between models. Raven achieves comparable result to ChatGPT on unhealthy conversation detection and even surpasses it on the sarcasm detection dataset. While this approach in prompting looks necessary, it alone is not enough to replace the capability of having free access to the whole context. Therefore, prompt engineering seems to be of significantly more importance to the RNN models compared to standard transformers. It is entirely possible that good prompts to RNN models do not mean additional restrictions, but should simply be constructed using completely different guidelines. While authors of a forementioned paper (Kocoń et al., 2023) perform chain-of-thought to improve results on MathQA dataset, trying the same approach on the Raven model achieved a very low accuracy of 48.11%. Despite that, using a very similar prompt without the chain-of-thought outperformed ChatGPT on this dataset. It would not be surprising if, thanks to the hidden state, RNN models do not need additional steps during inference, and adding those only makes the model forget the information it analyzed and deemed important. This further emphasizes the importance of the order of information the model receives.

The template used to prompt the Raven model in MathQA with chain-of-thought prompting:

Write the reasoning and highlight the answer to the question at the end in format: 'Answer: '. The question is: <here is question>

The template used to prompt the Raven model in MathQA without chain-of-thought prompting:
Write the answer to the math question in the format: 'Answer: '.
The question is: <here is question>

K Cases

In this part, we present a few instances of outputs produced by RWKV model using a Chat interface[5][6].

RWKV-4-Raven-14B

Tell me about ravens.

Ravens are large, black birds with a distinctive white head and neck. They are found in most parts of the world, including North America, Europe, Asia, and Australia. Ravens are known for their intelligence and problem-solving abilities. They are also considered to be symbols of death and transformation in many cultures. Ravens are often associated with the afterlife or death because they have been known to eat carrion or even other birds. In some cultures, ravens are also believed to be messengers of the gods or guardians of treasure.

RWKV-4-Raven-14B

Explain the concept of machine learning to a 10-year-old.

Machine learning is like a super-smart computer that can learn on its own. It can recognize patterns and make decisions based on what it has learned. Just like how you might remember something from the past, the computer can remember what it's learned and use it to make predictions or decisions in the future. It's like having a smart friend who always knows what to do!

RWKV-4-Raven-14B

Write a short poem describing nature's beauty in springtime.

Nature's beauty in springtime is rare,
A symphony of colors, a gentle surprise.
The flowers bloom, the birds sing and sing,
As the sun shines down, warming our skin.
The trees sway in the gentle breeze,
As we take in their beauty with ease.
The sky is blue, the clouds are white,
A sight to behold, it's truly divine.

[5]https://github.com/BlinkDL/ChatRWKV
[6]https://huggingface.co/spaces/BlinkDL/ChatRWKV-gradio

RWKV-4-Raven-14B

Below is the numerical stable formula for updating the state variable in RWKV:

$$a_t^* = e^{p-q} a_{t-1} + e^{u+k_t-q} v_t$$
$$b_t^* = e^{p-q} b_{t-1} + e^{u+k_t-q}$$
$$wkv_t = \frac{a_t^*}{b_t^*}$$

Please write out the corresponding pytorch code to do so, where both a_t and b_t are torch.Tensor([n])

Here is the PyTorch code to update the state variable in RWKV using the given formula:

```
import torch
def update_state(a, b, u, v, p, q):
    a_t = torch.exp(p - q) * a + torch.exp(u + k - q) * v
    b_t = torch.exp(p - q) * b + torch.exp(u + k - q)
    return a_t, b_t
a, b = torch.randn(2, 3)
u = torch.randn(2)
v = torch.randn(2)
wkv = torch.randn(2)
a_t, b_t = update_state(a, b, u, v, wkv, torch.randn(2))
print("a_t:", a_t)
print("b_t:", b_t)
```

This code defines a function update_state that takes in the state variables a and b, the control variables u and v, and the parameters wkv and torch.randn. It then uses the PyTorch torch.exp function to compute the exponential of the control variables and the log of the normalization factor. Finally, it uses these values to update the state.

RWKV-4-Raven-14B

Explain the following metaphor: Life is like cats.

The metaphor "Life is like cats" suggests that life can be unpredictable and full of unexpected twists and turns, just like a cat's behavior. It implies that we cannot fully control or predict the outcome of our lives, just as we cannot control the movements of a cat. The metaphor also implies that life is complex and multifaceted, with many different variables and factors at play. Just as a cat can have multiple personalities or moods, life can have many twists and turns that are difficult to predict or understand. Overall, the metaphor suggests that life is complex and unpredictable, much like a cat's behavior.

RWKV-4-Raven-14B

Test the convergence of $\sum_{i=0}^{\infty} 2^{-i}$.

The given series is a geometric series with the first term $a_0 = 2$ and common ratio $r = \frac{1}{2}$. We can find the sum of this series using the formula for a geometric series:

$$\sum_{i=0}^{\infty} a_i = \frac{a_0}{1-r} = \frac{\frac{1}{2}}{1-\frac{1}{2}} = \frac{1}{1-\frac{1}{2}} = \frac{2}{1} = 2$$

So, the sum of the given series is $\boxed{2}$.

RWKV-4-Raven-14B

I would like to cook some Chinese food at home. Do you have any suggestions on the meal I could choose?

Yes, of course! If you are not very experienced in cooking Chinese food, you could start with making noodles or rice, then make a dish using one of those. I also think making stir-fry with some meat is an excellent idea.

I want to make something more special. What about red braised pork belly?

Yes, red braised pork belly is also very popular and tasty.

Tell me the steps to cook.

You will need a few ingredients such as pork belly, star anise, sugar, ginger, scallions and wine. Put all of these in a pot, cook on medium heat and simmer until the meat is tender. After that, remove the meat from the pot and reserve the sauce. Slice the meat and add some ginger slices into it, then boil until it's almost done.

What do I need to be careful with the most during the cooking process?

To cook pork belly properly, it is very important to monitor the heat of the stove and be patient with the cooking process. Overcooking it can cause the pork to be tough, dry and difficult to chew. You can add a few ingredients like soy sauce or black beans to give the pork more flavor.

Tell me more about the proper heat and timing.

If the temperature of the water is too high, it can damage the collagen and connective tissues, resulting in tough pork belly. It is essential to control the temperature carefully, especially when braising the pork. Additionally, keep the water simmering but not boiling to maintain a low simmer, as high heat will dry out the pork belly and result in tough and unpalatable meat. Lastly, monitor the heat during cooking and make sure the heat is medium, as this is what is most appropriate for slow cooking pork belly.

I see. Thank you for your advice.

You're welcome! It was a pleasure chatting with you. Let me know if you need any further assistance in the future.

RWKV-4-Raven-14B

If all apples are fruits and all fruits are sweet, are all apples sweet? Please explain your answer in detail.

Yes, all apples are fruits, and all fruits are sweet. Therefore, it is safe to assume that all apples are sweet.

If birds can fly. Penguins are birds. Can penguins fly? Please explain your answer in detail.

No, penguins cannot fly. Penguins are birds, but they are not capable of flight like other birds. They have wings and can glide for short distances, but they cannot take off or fly long distances like other birds.

The day before two days after the day before tomorrow is Saturday. What day is it today? Please explain your answer in detail.

Today is Friday. Two days after tomorrow is Sunday. Tomorrow is Saturday.

If five cats can catch five mice in five minutes, how long will it take one cat to catch one mouse? Please explain your answer in detail.

If five cats can catch five mice in five minutes, it means that each cat can catch one mouse in one minute. To find out how long it will take one cat to catch one mouse, we can use the concept of proportionality.
Let's assume that it takes x minutes for a cat to catch a mouse. Then, the number of mice caught by a cat in one minute is equal to the number of cats multiplied by the number of mice caught by each cat: x = 5 * 1 = 5
So, it will take 5 minutes for one cat to catch one mouse.

RWKV-4-Raven-14B

Summarize the following paragraph.

<The Abstract of this paper goes here>

The paper presents a novel architecture, Receptance Weighted Key Value (RWKV), that combines the efficient parallelizable training of Transformers with the efficient inference of RNNs. The model is able to scale to tens of billions of parameters and exhibits linear computational complexity during training and inference, making it a promising alternative to Transformers for sequence processing tasks.

Please further summarize it in one sentence.

The researchers have developed a new architecture that combines the efficient parallelizable training of Transformers with the efficient inference of RNNs, which allows for linear scaling in memory and computational requirements.