SpikeGPT: Generative Pre-trained Language Model with Spiking Neural Networks

Anonymous authors
Paper under double-blind review

Abstract

As the size of large language models continue to scale, so does the computational resources required to run them. Spiking Neural Networks (SNNs) have emerged as an energy-efficient approach to deep learning that leverage sparse and event-driven activations to reduce the computational overhead associated with model inference. While they have become competitive with non-spiking models on many computer vision tasks, SNNs have proven to be more challenging to train. As a result, their performance lags behind modern deep learning, and until now, SNNs have yet to succeed at language generation on large-scale datasets. In this paper, inspired by the Receptance Weighted Key Value (RWKV) language model, we successfully implement 'SpikeGPT', a generative language model with binary, event-driven spiking activation units. We train the proposed model on two model variants: 46M and 216M parameters. To the best of our knowledge, SpikeGPT is the largest backpropagation-trained SNN model when released, rendering it suitable for both the generation and comprehension of natural language. We achieve this by modifying the transformer block to replace multi-head self-attention to reduce quadratic computational complexity $\mathcal{O}(T^2)$ to linear complexity $\mathcal{O}(T)$ with increasing sequence length. Input tokens are instead streamed in sequentially to our attention mechanism (as with typical SNNs). Our experiments show that SpikeGPT remains competitive with non-spiking models on tested benchmarks, while maintaining 32.2× fewer operations when processed on neuromorphic hardware that can leverage sparse, event-driven activations.

1 Introduction

Artificial Neural Networks (ANNs) have recently achieved widespread, public-facing impact in Natural Language Processing (NLP), but with a significant computational and energy consumption burden across training and deployment. As an example, training GPT-3 was projected to use 190,000 kWh of energy (Brown et al., 2020; Dhar, 2020; Anthony et al., 2020). Spiking neural networks (SNNs), inspired by neuroscientific models of neuronal firing, offer a more energy-efficient alternative by using discrete spikes to compute and transmit information (Maass, 1997). Spike-based computing combined with neuromorphic hardware holds great potential for low-energy AI (Davies et al., 2018; Merolla et al., 2014; Sun et al., 2022), and its effectiveness in integration with deep learning has been demonstrated through numerous studies (Roy et al., 2019; Pfeiffer & Pfeil, 2018; Fang et al., 2021; Eshraghian et al., 2021; Wu et al., 2018; Zhang et al., 2020).

While SNNs have shown competitive performance in computer vision tasks such as classification and object detection (Barchid et al., 2023; Kim et al., 2020; Cordone et al., 2022), they have yet to attain similar success in generative models. With respect to language generation, this can be attributed to several reasons: i) the absence of an effective language encoding technique for SNNs, ii) the difficulty of training large-scale SNNs due to the extreme constraint on layer-to-layer bandwidth (i.e., binarized spike activations) (Eshraghian et al., 2022), and the lack of informative gradient signals in excessively sparsified models (Eshraghian & Lu, 2022), and iii) the inherent recurrence of SNNs is incompatible with self-attention, where input token are passed to the model in parallel (Vaswani et al., 2017). These issues mean that training large-scale SNNs via error backpropagation is extremely challenging, leading to a lack of performant SNNs in language generation.

Despite the difficulties faced by recurrent networks in NLP, the sequential structure of linguistic data presents a unique advantage for SNNs. To address these problems, we propose three techniques. First, rather than using an encoder to project information into an additional temporal dimension, SpikeGPT aligns the sequence dimension of language with the temporal dimension of SNNs. This eliminates the need for an encoder as with most prior attention-based SNNs (Zhou et al., 2023; Li et al., 2022). Second, we apply autoregressive training rather than accumulating spikes at the output to calculate the loss, as is common in modern SNNs. Third, although binary activations constrain layer-to-layer bandwidth, we show that using stateful neurons can greatly reduce the adverse impact of binarization. By leveraging the capabilities of the RWKV model and integrating these three techniques, we have developed the SpikeGPT language model. This was the first SNN to achieve language generation at the time of writing and code release of SpikeGPT. It is also the largest SNN trained to date in terms of parameter count when it is released; the maximum version has 216M parameters, which is 3× more than the previous largest SNN (Zhou et al., 2023).

The implementation of SpikeGPT is based on integrating recurrence into the Transformer block such that it is compatible with SNNs and eliminates quadratic computational complexity, allowing for the representation of words as event-driven spikes. Combining recurrent dynamics with linear attention enables our network to stream incoming data word-by-word, and commence computation before a sentence has been completed, while still retaining long-range dependencies present in complex syntactic structures. Our experiments show that SpikeGPT achieves competitive performance on all tested datasets while consuming significantly less energy compared to traditional ANN models. Our contributions in the field of NLP and language generation can be succinctly described as follows:

1. we provide the first demonstration of language-generation using direct-SNN training;
2. we achieve performance comparable to that of ANNs, while preserving the energy efficiency of spike-based computations and reducing quadratic computational complexity $\mathcal{O}(T^2)$ to linear complexity $\mathcal{O}(T)$;
3. our results demonstrate that a small-scale variant of the SpikeGPT model with 46 million parameters performs competitively against similar transformer models. Furthermore, it achieves this performance with an estimated 33.2× less energy consumption on asynchronous hardware.

2 Related Works

Although language generation has not previously been achieved with SNNs, this section provides an overview of how SNNs have been used in basic NLP tasks, and the ways in which transformers have been adopted for SNNs.

Spiking Neural Networks for Natural Language Processing. Xiao et al. (2022) proposes a bi-directional SNN for sentiment classification and machine translation tasks. Their approach uses spiking encoders, which replace costly multiplication operations with much cheaper additive operations to significantly reduce computational energy consumption. Similarly, Lv et al. (2023b) presents a two-step method to train SNNs for text classification with a simple way to encode pre-trained word embeddings as spike trains. Their results indicate that converted SNNs achieve comparable results to their ANN counterparts and are more robust against adversarial attacks. Furthermore, Diehl et al. (2016) demonstrate the train-and-constrain methodology that enables the mapping of machine-learned recurrent neural networks (RNNs) on a substrate of spiking neurons. The authors achieve 74% accuracy on a question classification task using less than 0.025% of the cores on one TrueNorth chip (Merolla et al., 2014), showcasing the potential for SNNs in classification tasks in NLP. For the pre-train language model side, two works (Bal & Sengupta (2023) and Lv et al. (2023a)) offer a direct-trained spiking Bidirectional Encoder Representations from Transformers (BERT) model via knowledge distillation.

Transformer in Spiking Neural Networks. The Transformer model, first introduced in Vaswani et al. (2017), has shown significant success in various NLP tasks. However, the application of the Transformer model to SNNs has been relatively limited. The first Spiking Transformer model was proposed in Zhou et al. (2023), which proposes spiking self-attention to model visual features using sparse Query, Key and Value

matrices. Li et al. (2022) proposes another variant on Transformer-based SNNs, adopting spatial-temporal attention instead of spatial or temporal-wise attention to better incorporate the attention mechanism within the Transformer.

While Transformers were initially proposed to solve NLP tasks, SNN-based Transformers have only succeeded with vision tasks. An additional temporal dimension must be added which increases the computational complexity from quadratic to the cubic order ($\mathcal{O}(T^3)$), which makes training more expensive. The additional challenges of extreme sparsity, non-differential operators, approximate gradients, and single-bit activations that are characteristic of SNNs make training convergence more challenging. The demonstrated image classification tasks have a far smaller number of output classes, which shrinks the scale of demonstrated networks. Image classification also does not exploit the inherent long-range learning capacity of self-attention. Therefore, there is under-explored potential in the application of Transformer models in other SNN-based applications beyond vision tasks. In the following sections, we demonstrate how this computational complexity is reduced to enable scaled-up models that are capable of language generation.

3 Methods

3.1 Leaky Integrate-and-Fire Neuron

We employ the Leaky Integrate-and-Fire (LIF) neuron as the default spiking neuron of our model, a widely used model for SNNs often trained via error backpropagation (Maass, 1997). The LIF dynamics are represented as follows:

$$\begin{cases} U[t] = H[t] + \beta(Y[t] - (H[t-1] - U_{\text{reset}})) \\ S[t] = \Theta(U[t] - U_{\text{threshold}}) \\ H[t] = U[t] \cdot (1 - S[t]) \end{cases} \tag{1}$$

where β is a decay factor, U is the membrane potential (or hidden state) of the neuron, S is the spiking tensor with binarized elements, Y denotes the output of the previous series RWKV block (see Eq. 7), $\Theta(\cdot)$ denotes the Heaviside function, and H represents the reset process after spike emission. We set $U_{\text{threshold}} = 1$, $U_{\text{reset}} = 0$ and $\beta = 0.5$ as done in Zhu et al. (2022) and Yao et al. (2021; 2023).

To overcome the non-differentiable problem during back-propagation caused by the Heaviside step function $\Theta(\cdot)$, we employ the arctangent surrogate function during the backward pass. The arctangent function $\sigma'(x) = \frac{\alpha}{2(1+(\frac{\pi}{2}\alpha x)^2)}$ (a Sigmoid-like shape) is applied as a 'surrogate gradient' for backward propagation to provide a biased gradient estimator (Fang et al., 2021; Neftci et al., 2019).

3.2 Model Architecture

The high-level architecture of SpikeGPT is shown in Fig. 1. Given an embedded input $I \in \mathbb{R}^{T \times d}$, we first use a Binary Embedding (BE) layer to embed the input to a binarized representation, X_0 (Eq. 2). Using Spiking RWKV, X_0 is passed to the L-*th* SpikeGPT layer. Similar to the standard Transformer block, a SpikeGPT block consists of Spiking RWKV (SRWKV) unit and a Spiking Receptance Feed-Forward Networks (SRFFN) unit. Residual connections are used in both the SRWKV and SRFFN blocks using the SEW-ResNet (Fang et al., 2021) configuration, which is a well-established standard form within the Spiking ResNet framework with sparse integer spikes. Once the data has traversed through all layers, the model is directed towards the generation head (GH) for the purpose of generating the next token. For natural language understanding (NLU) tasks, the model uses a classification head instead. Sec. 3.6 provides further information.

$$X_0 = \text{BE}\,(I)\,, \qquad I \in \mathbb{R}^{T \times d}, X_0 \in \mathbb{R}^{T \times d}, \tag{2}$$

$$X'_l = \text{SRWKV}(X_{l-1}) + X_{l-1}, \qquad X'_l \in \mathbb{R}^{T \times d}, l = 1...L \tag{3}$$

$$X_l = \text{SRFFN}(X'_l) + X'_l, \qquad X_l \in \mathbb{R}^{T \times d}, l = 1...L \tag{4}$$

$$Y = GH(X_L). \tag{5}$$

In Section 3.3, detailed information about Binary Embedding (BE) will be provided. Sec. 3.4 will primarily focus on the SRWKV block, while Sec. 3.5 will delve into the Spiking Receptance Feed-Forward Network

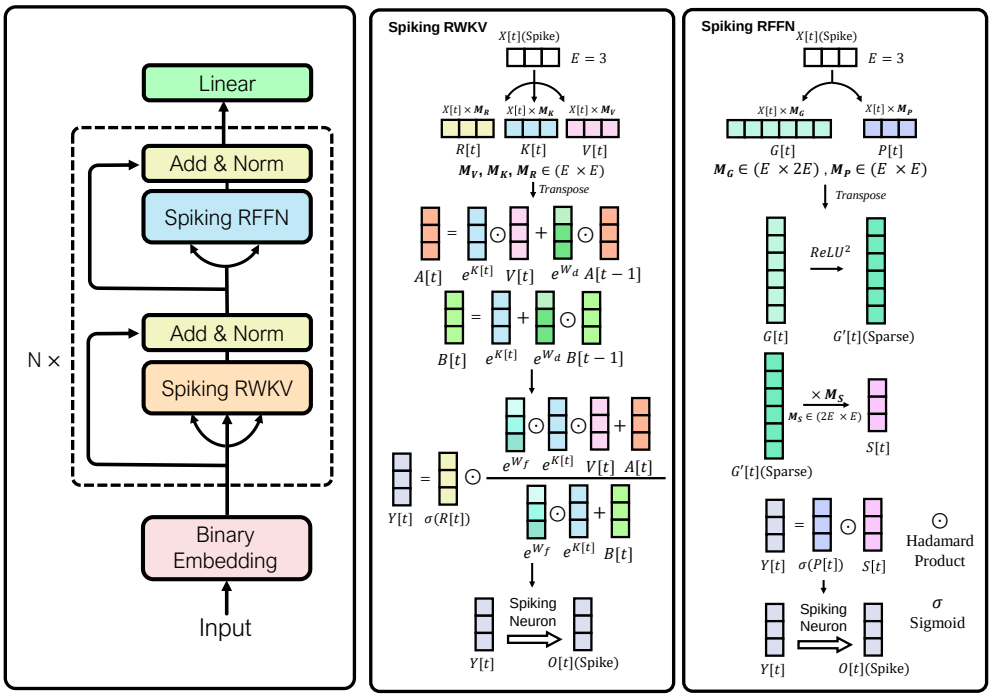

Figure 1: Model Architecture. The left portion displays the block-level structure. The middle and right illustrations demonstrate the Spiking RWKV and Spiking RFFN architectures, respectively. Spiking RWKV serves as a token mixer and Spiking RFFN functions as a channel mixer. These components are arranged in a loop with residual connections in a manner akin to a Transformer architecture.

(SRFFN) block. Finally, in Sec .3.6, we will describe the training and inference details employed for natural language generation (NLG) and natural language understanding (NLU). At the commencement of each block, a token-shift is applied which is detailed in Appendix B.1.

3.3 Binary Embedding

To maintain consistency with the binary activations of SNNs, we propose a binary embedding step to convert the continuous outputs of the embedding layer into binary spikes. The conversion is performed using a Heaviside function for feed-forward propagation which maps the continuous values to binary spikes, This allows us to convert continuous embedding values into spikes using non-differentiable functions, while still being able to perform backpropagation and update the weights of the embedding layer (Neftci et al., 2019). During backpropagation, the arctangent surrogate function is applied as is the case with LIF neurons.

3.4 Efficient Processing of Variable-Length Sequences Using Spiking RWKV

In this section, we revisit the self-attention mechanism, which endows Transformers with the ability to process variable-length sequences, enabling advanced language modeling capabilities. Subsequently, we highlight a fundamental challenge: the incompatibility of self-attention with SNN language modeling. Self-attention relies on access to the entire sequence for effective language modeling and is not inherently recurrent, making it incompatible with SNNs. Furthermore, it involves dynamic matrix-matrix multiplication operations. The associated computational overhead compromises the objective of reducing computational costs via SNNs. To address these limitations, we introduce the Spiking RWKV (SRWKV) module as a replacement for the self-attention module. This module draws inspiration from the RWKV model (Peng et al., 2023), an RNN model renowned for achieving Transformer-level proficiency across a spectrum of model sizes. This module serves the same essential function of facilitating information exchange across token dimensions but achieves

this through element-wise products rather than matrix-matrix multiplication, while also accommodating the recurrent nature of SNNs.

Recall Self-Attention. In the context of Transformers, the self-attention mechanism operates on an input sequence denoted as $X \in \mathbb{R}^{T \times d}$ and employs a scaled dot product attention technique. Mathematically, self-attention is formally expressed as follows:

$$f(X) = \text{softmax}\left(\frac{Q(K)^T}{\sqrt{d_k}}\right) V, Q = XM_Q, K = XM_K, V = XM_V \tag{6}$$

Here, $M_Q \in \mathbb{R}^{d \times d_k}$, $M_K \in \mathbb{R}^{d \times d_k}$, and $M_V \in \mathbb{R}^{d \times d_v}$ represent linear transformations, while d_k and d_v signify the dimensions of the key and value vectors, respectively. This mechanism enables dynamic information mixing across token dimensions and accommodates sequences of variable lengths by leveraging dynamic matrix-matrix multiplication ($Q(K)^T$) and the softmax function (which necessitates access to the entire sequence).

Nevertheless, in the case of a typical SNN characterized by its event-driven nature, the recurrent structure of the network poses a challenge. SNNs can only generate spikes based on information from previous time-steps, making it impossible to access the entire sequence at once, as required by self-attention. One potential solution involves introducing an additional temporal dimension, denoted as $T_{\text{additional}}$, to allow spiking neurons to feed forward. However, this approach would substantially increase the model's size and necessitate additional computations, scaling proportionally with $T_{\text{additional}}$. This is not practical, especially for large-scale language models reaching the scale of hundreds of billions of parameters. Therefore, the optimal approach is not to introduce a new dimension, but to leverage the existing sequence dimension for forward propagation. This is why a spiking transformer would benefit from a recurrent alternative to self-attention.

Spiking RWKV. To address the challenges posed by self-attention in the context of language modeling, we introduce the Spiking RWKV, a novel approach that incorporates both recurrent and block-level spiking features. The foundation of the vanilla RWKV is inspired by the Attention Free Transformer (Zhai et al., 2021), serving as a replacement for the traditional self-attention mechanism. For comprehensive information on vanilla RWKV, including its overall structure and parallelization techniques, please refer to Appendix B.1. In contrast to self-attention, which operates on the entire sequence dimension, the spiking input of Spiking RWKV is unrolled as $X[t] \in \mathbb{R}^{1 \times d}$, where t represents the time step index, rather than the entire sequence $X \in \mathbb{R}^{T \times d}$. Similar to self-attention, Spiking RWKV initiates by applying linear transformations: $R = X[t]M_R$, $K = X[t]M_K$, and $V = X[t]M_V$, where $M_R, M_K, M_V \in \mathbb{R}^{d \times H}$, with H indicating the hidden size. Notably, all inputs to the three MLP layers are in the form of spiking activations. The subsequent step involves generating the output using the following equation:

$$Y[t+1] = \mathcal{SN}\left(\sigma(R[t]) \odot \frac{\exp(W_f) \odot \exp(K[t]) \odot (V[t]) + A[t]}{\exp(W_f) \odot \exp(K[t]) + B[t]}\right) \tag{7}$$

The hidden states A and B are defined as:

$$A[t] = \exp(K[t]) \odot (V[t) + \exp(W_d) \odot A[t-1] \tag{8}$$

and

$$B[t] = \exp(K[t-1]) + \exp(W_d) \odot B[t-1] \tag{9}$$

Here, $\mathcal{SN}$ refers to spiking neuron layers, the $\odot$ symbol represents element-wise multiplication, $W_d \in \mathbb{R}^{1 \times d}$ and $W_f \in \mathbb{R}^{1 \times d}$ are decay vectors, where W_f is responsible for weighting the current time-step's information, and W_d plays a role in decaying the influence of information from previous time-steps. Details of W_d and W_f can be found in Appendix B.4. Notably, Spiking RWKV diverges from self-attention by not employing matrix-matrix multiplication to dynamically adjust the attention map according to the input. Instead, it utilizes the learnable vectors W_d and W_f to recurrently blend the token dimensions of the input. While self-attention dynamically reweights token dimensions based on the input, Spiking RWKV adopts a continuous decay strategy rather than a dynamic weighted attention strategy. Because the hidden states $A[t]$ and $B[t]$ encapsulate information from the previous time-step, this module can effectively blend information across

token dimensions, much like self-attention. Additionally, this approach is particularly well-suited for language modeling. When using self-attention in casual language models, it's necessary to mask out half of the attention map to prevent information leakage. In contrast, Spiking RWKV combines the inherent recurrent properties of SNN with RWKV, and both naturally operate in a unidirectional manner.

3.5 Spiking Receptance Feed-Forward Networks (SRFFN)

Each block in our model contains a fully connected feed-forward network with a gating mechanism (SRFFN), which is applied to normalized and token-shifted outputs of each spiking-RWKV module. This SRFFN module consists of three linear transformations as follows:

$$Y'[t] = \mathcal{SN}(\sigma(M_P X[t]) \odot M_S(\text{Activation}(M_G X[t]))) \tag{10}$$

where $\mathcal{SN}$ means spiking neuron layers, $Y'[t]$ denotes the output of SRFFN at time-step t which is then passed to the spiking neuron (Eq. 1). $\{M_P, M_G, M_S\} \in \mathbb{R}^{d \times H}$ are learnable parameters of the linear transformations, and Activation represents the activation function. Depending on the specific SpikeGPT variant being used, we employ either the $ReLU^2$ or $\mathcal{SN}$ activation function. SRFFN is a variant of the Gated Linear Unit (GLU) (Dauphin et al., 2017), which can control the degree of information flowing into the model by $\sigma(M_P X[t])$. In order to maintain consistency between SRFFN and GEGLU parameters (Shazeer, 2020), we set the size of H from the SRFFN to $4d$. After the feedforward process in the SRFFN layer, the resulting output $Y'[t]$ will be subsequently fed into a LIF neuron, as depicted in Eq. 1. In this context, $Y[t]$ is replaced with the updated value $Y'[t]$ to preserve the block-level spiking and binary characteristics, thereby ensuring the maintenance of the desired sparsity.

3.6 Training & Inference

Our training procedure consists of two stages. The first stage is pre-training on a large-scale corpus to build a high-capacity language model, and the next stage is specific fine-tuning to perform downstream tasks, such as natural language generation (NLG) and natural language understanding (NLU).

NLG Tasks. We adopt a decoder-only pre-training paradigm similar to GPT to train the model. Specifically, our model utilizes SRWKV and SRFFN modules to process the input token sequence and generate an output distribution for each target token. Formally, given a token sequence $\mathcal{C} = \{c_1, c_2, \cdots, c_n\}$, we use the standard language modeling objective to maximize the following likelihood:

$$\mathcal{P}(c_t) = \text{softmax}(Y'[t] W_e^T) \tag{11}$$

$$\mathcal{L}_p = \sum_{i=1}^{T} log\mathcal{P}(c_i | c_1, c_2, \cdots, c_{i-1}; \Theta) \tag{12}$$

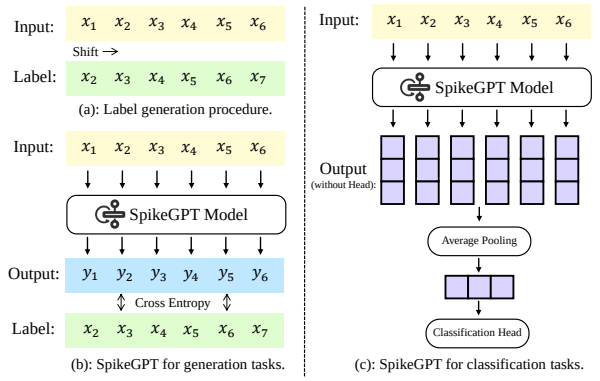

Figure 2: Training SpikeGPT for NLG and NLU tasks.

where W_e^T is the token embedding matrix, and Θ is the set of all model parameters.

After pre-training the model using the loss in Eq. 12, model parameters are fine-tuned to adapt to different downstream tasks in NLG and NLU. For natural language generation tasks, we define a new dataset $\mathcal{D}_G$, where each sample of data consists of a sequence of input tokens. The consistency between the pre-training process and the NLG task allows for the fine-tuning procedure to reuse the method and objectives adopted in the pre-training pipeline (Eq. 11 and Eq. 12), which maximizes the likelihood of the target token based on the previous information of the target position.

NLU Tasks. As for NLU tasks, such as sentiment classification, the fine-tuning process requires several modifications to the top-level of the pre-trained SpikeGPT model to adapt to NLU tasks, as shown in Fig. 2. Specifically, given a new dataset for the downstream task $\mathcal{D}_U$, where each instance consists of a token sequence $\mathcal{C}_i$ and label l_i, the following objective is maximized:

$$\mathcal{L}_{NLU} = \sum_{(\mathcal{C}_i, l_i)} l_i * logP(\mathcal{C}_i) \tag{13}$$

where $\mathcal{P}(\mathcal{C}_i)$ is defined as:

$$\mathcal{P}(\mathcal{C}_i) = \text{softmax}(YW_m^T) \tag{14}$$

W_m^T includes the learnable parameters of the MLP module in the top layer, and Y is generated by passing the input token sequence $\mathcal{C}_i$ through the model and then average-pooling the embedding of each target, which is formalized as:

$$Y = \text{AvgPooling}(X_L[1], X_L[2], \cdots, X_L[T]) \tag{15}$$

In the inference phase, we directly give a prompt for the NLG task, and let the model continue and calculate bits-per-character (BPC) and perplexity (PPL) as an evaluation metric. For the NLU task, we pass the inputs through our model to obtain the embedding over target positions, and then apply a mean-pooling operation to all embedded tokens to predict the label of each instance.

3.7 Theoretical Energy Consumption Analysis

The key attribute of SNNs here is their remarkable energy efficiency due to dynamical sparsity. We provide a theoretical energy consumption analysis of SpikeGPT's constituent components, while evaluating its feasibility for deployment on asynchronous hardware. The block-level theoretical energy consumption analysis is shown in Tab. 1.

Spiking RWKV. The SRWKV block comprises two integral components: the Spiking MLP layer, which receives sparse spike inputs, and the full-precision element-wise product. Within the Spiking RWKV, akin to self-attention mechanisms, the spike input denoted as X undergoes projection through MLP layers, resulting in three distinct variables, namely, R, K, and V. Notably, all these MLP layers operate on spike inputs, facilitating the transformation of matrix multiplication into sparse addition. It is worth highlighting that the non-binary element-wise product, as depicted in Tab. 1, constitutes a relatively minor proportion compared to the MLP layers. Importantly, it is essential to note that all these operations can be efficiently supported by neuromorphic chip technology.

SRFFN. The SRFFN block primarily consists of MLP layers and element-wise product operations. However, an alternative operation is introduced when employing the $ReLU^2$ activation function instead of spiking neurons. While $ReLU^2$ activations are not binary, they induce dynamical sparsity. We can further employ quantization techniques to convert them into integers, a common practice in LLMs (Liu et al., 2023; Dettmers et al., 2022b;a), where 8-bit activations and weights can be used with no noticeable loss in performance (Xiao et al., 2023). Moreover, we investigated the possibility of using LIF to replace $ReLU^2$ in Sec. 4.4, which demonstrates that the impact of this substitution on performance is negligible. With regard to integer sparse spikes, modern neuromorphic chips (Orchard et al., 2021; Parpart et al., 2023) accommodate graded spikes that support up to 32-bit integer, rendering the entire SRFFN block compatible with neuromorphic hardware, requiring solely addition operations for its execution.

4 Experiments

4.1 Experiment Settings

We test two variants of the 46 million parameter model; one where $T = 1,024$ and another where $T = 3,072$. We used the Enwik8 dataset to conduct both training and testing in 46M scale, and our most extensive model with 216 million parameters was trained using the OpenWebText2 (Gao et al., 2020) corpus for pre-training. In our 46M model, we employ a character-level tokenizer, as has been done in previous works Zhai et al.

Table 1: Energy Evaluation: FL_{MLP} represents the Floating-Point Operations (FLOPs) of the MLP layers in the ANNs, while $\hat{R}$ denotes the spike firing rates (indicating the proportion of non-zero elements in the spike matrix) within the spike matrices. In computing operation counts, models are parameterized with $T = 3072$, $d = 512$, $\hat{R} = 0.15$, all derived from real test data of SpikeGPT-46M. Additionally, energy consumption is characterized by assuming $E_{MAC} = 4.5$pJ and $E_{AC} = 0.9$pJ based on Horowitz (2014) and Rathi & Roy (2021). Other than element-wise operation involving dot products in SpikeGPT, all other module inputs are in spike form, leading to a significant reduction in operations by a factor of up to 32.2× overall. This is reflected in the Vanilla GPT-to-SpikeGPT (V/S) ratio.

		Vanilla GPT (Radford et al., 2018) (with GLU (Dauphin et al., 2017))	SpikeGPT (**This work**)	Energy Consumption (pJ) Vanilla GPT	SpikeGPT	Ratio V/S
Attention	$Q/R, K, V$	$E_{MAC} \cdot 3Td^2$	$E_{AC} \cdot \hat{R} \cdot 3Td^2$	1.09×10^{10}	3.25×10^8	33.3×
	$f(Q/R, K, V)$	$E_{MAC} \cdot 2T^2d$	$E_{MAC} \cdot 7Td$	8.50×10^7	4.95×10^7	1.71×
	Scale	$E_{MAC} \cdot T^2$	-	4.25×10^7	-	-
	Softmax	$E_{MAC} \cdot 2T^2$	-	8.50×10^7	-	-
FFN	Layer 1	$E_{MAC} \cdot FL_{MLP1}$	$E_{AC} \cdot \hat{R} \cdot FL_{MLP1}$	3.62×10^9	1.09×10^8	33.3×
	Layer 2	$E_{MAC} \cdot FL_{MLP2}$	$E_{AC} \cdot \hat{R} \cdot FL_{MLP2}$	1.45×10^{10}	4.35×10^8	33.3×
	Layer 3	$E_{MAC} \cdot FL_{MLP3}$	$E_{AC} \cdot \hat{R} \cdot FL_{MLP3}$	3.62×10^9	1.09×10^8	33.3×
Overall	-	-	-	3.29×10^{10}	1.02×10^9	32.2×

(2021). To evaluate the performance of our model, we calculate its BPC metrics. To mitigate the issue of overfitting, we incorporate dropout after the output of each SRFFN block and set the dropout ratio to 0.03. In our 216M model with pre-training, we employ the Byte Pair Encoding (BPE) tokenizer and share the same hyper-parameters as GPT-NeoX (Black et al., 2022). Due to the availability of sufficient data for pre-training, we do not incorporate dropout as we did in our 46M model and remove the binary embedding but use the first layer neurons for encoding. To facilitate better convergence, we utilize a warmup technique during the first 500 training steps. For both the 46M and 216M models, we use the Adam optimizer, and set the learning rate to 6×10^{-4} and 4×10^{-4}, respectively. All experiments were conducted on four NVIDIA V100 graphic cards. For the models of 46M and 216M, we trained them for 12 and 48 hours respectively.

We evaluated SpikeGPT on two major language-related tasks: Natural Language Generation (NLG) and Natural Language Understanding (NLU). For NLG, we evaluated the text generation performance of SpikeGPT on three classic text generation datasets: Enwik8 (Mahoney, 2011), WikiText-2 (Merity et al., 2017), and WikiText-103 (Merity et al., 2017). These datasets are widely used to measure a model's ability to perform compression and for benchmarking. For NLU, we evaluated the performance of SpikeGPT on four classic text classification datasets: MR (Pang & Lee, 2005), SST-5 (Socher et al., 2013), SST-2 (Socher et al., 2013), and Subj (Pang & Lee, 2004). These datasets cover sentiment analysis and subjective/objective classification. Our implementation is based on PyTorch (Paszke et al., 2019) and SpikingJelly (Fang et al., 2020). For detailed information regarding the datasets and baseline methods, please refer to Appendix A. Our zero-shot text samples of this experiment can be found in Appendix C.

4.2 Results on Natural Language Generating Tasks

A summary of results are provided in Tabs. 2 and 3. This includes the BPC and PPL achieved on NLG tasks using SpikeGPT trained on Enwik8, WikiText-103, and WikiText-2 compared to several baselines, including 46M and 216M parameter variants.

As shown in Tab. 2, the generative performance of SpikeGPT has far surpassed that of models with an LSTM backbone, and can approach or even surpass some simplified variants of the Transformer, such as Linear Transformer and Synthesizer. However, it should be pointed out that there is still a certain gap between SpikeGPT and the vanilla Transformer. While increasing the size of T significantly reduces the training BPC of SpikeGPT, its test BPC has not changed significantly, indicating that SpikeGPT is potentially suffering from over-fitting.

Table 2: 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), Linear Transformer (Katharopoulos et al., 2020a), Performer (Choromanski et al., 2020), Stacked LSTM (Graves, 2013) and SHA-LSTM (Merity, 2019). 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.

Method	Spiking	L	d	T	Train BPC	Test BPC	Complexity	Params.
Transformer	✗	12	512	1024	0.977	1.137	$\mathcal{O}(T^2 \cdot d)$	43.0M
Reformer	✗	12	512	1024	1.040	1.195	$\mathcal{O}(TlogT \cdot d)$	40.1M
Synthesizer	✗	12	512	1024	0.994	1.298	$\mathcal{O}(T \cdot d^2)$	42.8M
Linear Transformer	✗	12	512	1024	0.981	1.207	$\mathcal{O}(T \cdot d^2)$	43.0M
Performer	✗	12	512	1024	1.002	1.199	$\mathcal{O}(T \cdot d^2 logd)$	43.0M
Stacked LSTM	✗	7	-	-	1.420	1.670	$\mathcal{O}(T \cdot d^2)$	-
SHA-LSTM (no attention)	✗	4	1024	1024	-	1.330	$\mathcal{O}(T \cdot d^2)$	-
SpikeGPT 46M	✓	**12**	**512**	**1024**	**1.113**	**1.283**	$\mathcal{O}(T \cdot d)$	46.1M
SpikeGPT 46M	✓	**12**	**512**	**3072**	**0.903**	**1.262**	$\mathcal{O}(T \cdot d)$	46.1M

We also compared the Perplexity of SpikeGPT and GPT-2 based on WikiText-2 and WikiText-103 datasets on text generation tasks. The results are shown in Tab. 3. In the interest of fairly comparing models of similar scales, we selected GPT-2 small and GPT-2 medium (Radford et al., 2019) with parameter sizes similar to those of the fine-tuned 216M SpikeGPT. We found that after fine-tuning, the performance of SpikeGPT on WikiText-2 has surpassed that of GPT-2 series. Unfortunately, the performance of SpikeGPT on the larger WikiText-103 dataset has fallen behind the GPT-2 series models, which suggests a potential need for refined, and perhaps more sophisticated, training methodologies for SpikeGPT when dealing with larger scale corpora, e.g., knowledge distillation (Bal & Sengupta, 2023).

Table 3: Results on WikiText-2 and WikiText-103 measured in token-level perplexity. Lower values indicate better performance. We report the perplexity when the lowest value was achieved on the validation datasets. Note that for WikiText-2, we use both the WikiText-103 and WikiText-2 training sets to extend the training corpus.

Method	Parameters	WikiText-103		WikiText-2	
		Val.	Test.	Val.	Test.
GPT-2 Small (Radford et al., 2019)	124M	-	29.41	-	37.50
GPT-2 Medium (Radford et al., 2019)	346M	-	26.37	-	22.76
SpikeGPT With Pre-training	**216M**	**39.92**	**39.75**	**19.17**	**18.01**

4.3 Results on Natural Language Understanding Tasks

For NLU tasks, we utilize SpikeGPT as a dynamic embedding generator that constructs embeddings based on context. We compare SpikeGPT with text classification algorithms of similar scales, including LSTM (Hochreiter & Schmidhuber, 1997), TextCNN (Kim, 2014), BERT (Kenton & Toutanova, 2019), and the latest SNN-based text classification algorithm TextSCNN (Lv et al., 2023b). The accuracy on four datasets is shown in Tab. 4. The fine-tuned 216M SpikeGPT achieves the second-highest performance among the models, only surpassed by BERT. BERT is a bidirectional Transformer encoder that uses masked training to obtain a high-quality text embedding. However, unlike SpikeGPT, BERT does not have the capability to generate text directly. Our 46M model without any fine-tuning also achieves competitive results compared to the baseline models, indicating the potential of SpikeGPT in NLU tasks. We also analyze the complexity of each method and show that SpikeGPT can achieve linear complexity by using spiking neurons and a recurrent

structure. Unlike TextSCNN, our model does not require an additional temporal dimension for processing, as it uses the sequence dimension iteratively during the forward-pass through the spiking neurons.

Table 4: Results of NLU tasks on four text classification datasets using both SNN and ANN methods. Measured in classification accuracy: the higher the better. In the 'complexity per layer' column, we compute the complexity of each method using the following notation: T is sequence length, d is the embedding dimension, K is the convolution kernel size, and $T_{additional}$ is the additional temporal dimension for feed-forward processing. Our model combines recurrent and spiking features and does not need an extra temporal dimension for feed-forward processing, as it exploits the inherent temporal dimension in the language model.

Method	Spiking	Recurrent	Complexity per layer	SST-2	SST-5	MR	Subj.
TextCNN (Kim, 2014)^{EMNLP}	✗	✗	$\mathcal{O}(K \cdot T \cdot d^2)$	81.93	44.29	75.02	92.20
TextSCNN-Direct Training (Lv et al., 2023b)$^{ICLR\text{-}2023}$	✓	✗	$\mathcal{O}(T_{additional} \cdot K \cdot T \cdot d^2)$	75.73	23.08	51.55	53.30
TextSCNN-ANN2SNN+Fine-tune (Lv et al., 2023b)$^{ICLR\text{-}2023}$	✓	✗	$\mathcal{O}(T_{additional} \cdot K \cdot T \cdot d^2)$	80.91	41.63	75.45	90.60
LSTM (Tai et al., 2015)	✗	✓	$\mathcal{O}(T \cdot d^2)$	84.92	46.43	81.60	-
BERT (Kenton & Toutanova, 2019)	✗	✗	$\mathcal{O}(T^2 \cdot d)$	91.73	53.21	86.72	-
SpikeGPT 46M	✓	✓	$\mathcal{O}(T \cdot d)$	80.39	37.69	69.23	88.45
SpikeGPT 216M	✓	✓	$\mathcal{O}(T \cdot d)$	82.45	38.91	68.11	89.10
SpikeGPT 216M With Pre-training	✓	✓	$\boldsymbol{\mathcal{O}(T \cdot d)}$	**88.76**	**51.27**	**85.63**	**95.30**

4.4 Ablation Study

We conducted an extensive ablation study on SpikeGPT, including the vanilla RWKV configuration. The study encompassed two primary modifications: the utilization of the Heaviside function for binarization in place of spiking neurons, and the replacement of the second-MLP activation function in the SRFFN layer from $ReLU^2$ to the spiking neuron LIF. The results are presented in Tab. 5.

For the Heaviside RWKV variant, we applied the Heaviside function directly to the RWKV layer for binarization, in lieu of the LIF neuron. Specifically, we employed the Heaviside function for the feed-forward process, and use the arctangent surrogate function to address the non-differentiable nature of the Heaviside function during the backward pass, similar to the original SpikeGPT. For SpikeGPT-S, we substituted the middle activation function in the SRFFN layer (as illustrated in Eq. 10) with the spiking neuron LIF to introduce layer-level binary spiking.

The results presented in Tab. 5 reveal a stark performance disparity between using Heaviside for RWKV binarization versus employing LIF. Furthermore, SpikeGPT-S and SpikeGPT-R represent a pair of models with a slight variation to the activation function in the hidden dense layer in the SRFFN module: SpikeGPT-S uses a spiking neuron thresholded activation function, whereas SpikeGPT-R uses a $ReLU^2$ activation function. SpikeGPT-S achieves competitive performance when compared to SpikeGPT-R, the original SpikeGPT configuration. Notably, SpikeGPT-S exhibits superior training loss compared to SpikeGPT-R, indicating that spiking neurons can enhance model performance in certain cases.

Table 5: An ablation study showcases the impact of different architectural modifications on the performance of SpikeGPT models, with all experiments conducted under the consistent settings of $L = 12$, $T = 1024$, and $d = 512$. The evaluation includes Vanilla RWKV as the baseline model without any modifications, Heaviside RWKV, which replaces all spiking neurons with Heaviside functions, SpikeGPT-S featuring a variation where the middle activation in the SRFFN block is also a spiking neuron rather than $ReLU^2$, and SpikeGPT-R, representing the default configuration of SpikeGPT.

Model	Vanilla RWKV	Heaviside RWKV	SpikeGPT-S	SpikeGPT-R
Train BPC	1.014	1.318	1.109	1.113
Test BPC	1.201	1.403	1.306	1.283

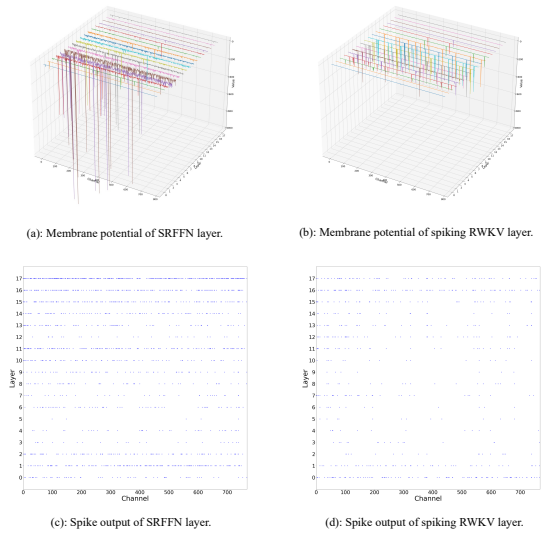

Figure 3: Visualization of spike and membrane potential of neurons. Figure (a) and (b) depict the membrane potential of the SRFFN layer, while figure (c) and (d) display the spike patterns observed in the SRFFN layer, where each dot represents a spike event.

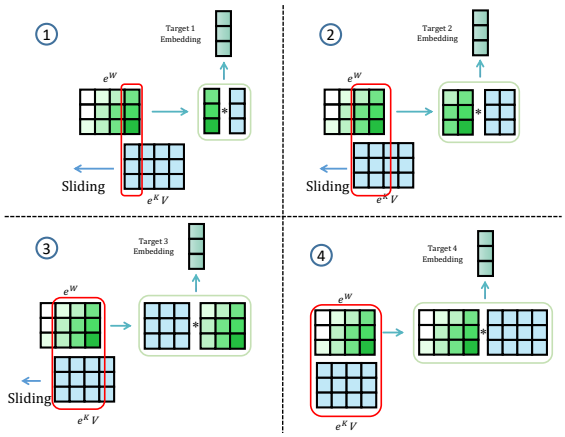

Figure 4: A demonstration of the Parallelized RWKV model adeptly handling computations involving e^W and $e^K V$ through the application of a large-kernel convolution operation. Notably, during the convolution's sliding window process, the model implements a decay mechanism to effectively manage temporal dependencies. For this demonstration, the sequence length is set to $N = 4$, and the embedding size is configured to $E = 3$.

5 Discussions

5.1 Visualization of Spike and Membrane Potential

To gain a more comprehensive understanding of SpikeGPT, we conducted visualizations of the spike and membrane potential patterns in the Spiking RWKV layer and Spiking Receptance Feed-Forward Networks (SRFFN) layers (Fig. 3). These visualizations clearly reveal distinct differences between the Spiking RWKV and SRFFN layers, indicating diverse information representation patterns. Notably, the SRFFN layer exhibits a higher firing rate, suggesting that it may retain more information similar to Transformer FFN layers (Geva et al., 2021). It is worth noting that in various studies, outliers in Transformer-based language models have been shown to significantly impact performance, making them a crucial concern in the quantification of large language models (Wei et al., 2022). Therefore, understanding and addressing the implications of outliers in SpikeGPT are of utmost importance, particularly in the context of optimizing its overall performance and reliability. However, due to the binary nature of SNNs, these outliers cannot be expressed in activation values as in ANNs. Nevertheless, a notable observation is the presence of prominent outliers in the membrane potential of individual neurons, many of which are negative. This finding suggests that SpikeGPT employs a different approach to accommodate and preserve outliers.

5.2 Parallelize RWKV using 1-D Convolution

Eq. 18 only calculates the weighted summation across target positions t. On the basis of Eq. 18, the values of all target positions can be represented as a 1-D convolution:

$$Y = \sigma(R) \odot \frac{\exp(W) \otimes (\text{LeftPad}(\exp(K) \odot V))}{\exp(W) \otimes \text{LeftPad}(\exp(K))} \tag{16}$$

where $\otimes$ denotes the 1-D convolution operation, LeftPad applies zero-padding to all columns preceding the $T - 1^{th}$ position.

Consider W to be a large convolutional kernel, performing a convolution with the matrix $\exp(K)$ (or $\exp(K) \odot V$). The computational complexity of the complete convolution is $\mathcal{O}(ET^2)$ (assuming the number of filters matches the sequence length, and E is the embedding size). This can be further optimized by adopting the Fast Fourier Transform (FFT) to reduce the time complexity of the whole convolution operation to $\mathcal{O}(ET\log T)$.

As shown in Fig. 4, given a text sequence N with length of 4, set the embedding size E to 3. The RWKV parallel process is a 1-d convolution process with e^W as the large convolution kernel and $e^K V$ as the feature.

6 Conclusion

Our results demonstrate that event-driven spiking activations are not only capable of language generation, but they can do so with fewer high-cost operations. We develop techniques that promote lightweight models for the NLP community, and make large-scale models for the neuromorphic and SNN community more effective. We demonstrate how large SNNs can be trained in a way that harnesses advances in transformers and our own serialized version of the attention mechanisms. We expect this research can open new directions for large-scale SNNs.

References

Lasse F Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. *arXiv preprint arXiv:2007.03051*, 2020.

Malyaban Bal and Abhronil Sengupta. Spikingbert: Distilling bert to train spiking language models using implicit differentiation. *arXiv preprint arXiv:2308.10873*, 2023.

Sami Barchid, José Mennesson, Jason Eshraghian, Chaabane Djéraba, and Mohammed Bennamoun. Spiking neural networks for frame-based and event-based single object localization. *Neurocomputing*, pp. 126805, 2023.

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, et al. Gpt-neox-20b: An open-source autoregressive language model. *arXiv preprint arXiv:2204.06745*, 2022.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:1877–1901, 2020.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

Loïc Cordone, Benoît Miramond, and Philippe Thierion. Object detection with spiking neural networks on automotive event data. In *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2022.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pp. 933–941, 2017. URL `http://proceedings.mlr.press/v70/dauphin17a.html`.

Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:30318–30332, 2022a.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022b.

Payal Dhar. The carbon impact of artificial intelligence. *Nature Machine Intelligence*, 2:423–5, 2020.

Peter U Diehl, Guido Zarrella, Andrew Cassidy, Bruno U Pedroni, and Emre Neftci. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In *IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1–8, 2016.

Jason K Eshraghian and Wei D Lu. The fine line between dead neurons and sparsity in binarized spiking neural networks. *arXiv preprint arXiv:2201.11915*, 2022.

Jason K Eshraghian, Max Ward, Emre Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D Lu. Training spiking neural networks using lessons from deep learning. *arXiv preprint arXiv:2109.12894*, 2021.

Jason K Eshraghian, Xinxin Wang, and Wei D Lu. Memristor-based binarized spiking neural networks: Challenges and applications. *IEEE Nanotechnology Magazine*, 16(2):14–23, 2022.

Wei Fang, Yanqi Chen, Jianhao Ding, Ding Chen, Zhaofei Yu, Huihui Zhou, Timothée Masquelier, Yonghong Tian, and other contributors. Spikingjelly. `https://github.com/fangwei123456/spikingjelly`, 2020. Accessed: 2022-05-21.

Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, Timothée Masquelier, and Yonghong Tian. Deep residual learning in spiking neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 34: 21056–21069, 2021.

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

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 5484–5495, 2021.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

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

Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)*, pp. 10–14. IEEE, 2014.

A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning (ICML)*, 2020a. URL `https://fleuret.org/papers/katharopoulos-et-al-icml2020.pdf`.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165, 2020b. URL `http://proceedings.mlr.press/v119/katharopoulos20a.html`.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT (NAACL)*, pp. 4171–4186, 2019.

Seijoon Kim, Seongsik Park, Byunggook Na, and Sungroh Yoon. Spiking-yolo: spiking neural network for energy-efficient object detection. In *Proceedings of the AAAI conference on artificial intelligence (AAAI)*, volume 34, pp. 11270–11277, 2020.

Yoon Kim. Convolutional neural networks for sentence classification. In Alessandro Moschitti, Bo Pang, and Walter Daelemans (eds.), *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1746–1751, 2014. URL https://doi.org/10.3115/v1/d14-1181.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *8th International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=rkgNKkHtvB.

Yudong Li, Yunlin Lei, and Xu Yang. Spikeformer: A novel architecture for training high-performance low-latency spiking neural network. *arXiv preprint arXiv:2211.10686*, 2022.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. Llm-qat: Data-free quantization aware training for large language models. *arXiv preprint arXiv:2305.17888*, 2023.

Changze Lv, Tianlong Li, Jianhan Xu, Chenxi Gu, Zixuan Ling, Cenyuan Zhang, Xiaoqing Zheng, and Xuanjing Huang. Spikebert: A language spikformer learned from bert with knowledge distillation. 2023a.

Changze Lv, Jianhan Xu, and Xiaoqing Zheng. Spiking convolutional neural networks for text classification. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023b.

Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.

Matt Mahoney. Large text compression benchmark, 2011.

Stephen Merity. Single headed attention RNN: stop thinking with your head. *CoRR*, abs/1911.11423, 2019. URL http://arxiv.org/abs/1911.11423.

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

Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019.

Garrick Orchard, E Paxon Frady, Daniel Ben Dayan Rubin, Sophia Sanborn, Sumit Bam Shrestha, Friedrich T Sommer, and Mike Davies. Efficient neuromorphic signal processing with loihi 2. In *2021 IEEE Workshop on Signal Processing Systems (SiPS)*, pp. 254–259. IEEE, 2021.

Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. *arXiv preprint cs/0409058*, 2004.

Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. *arXiv preprint cs/0506075*, 2005.

Gavin Parpart, Sumedh Risbud, Garrett Kenyon, and Yijing Watkins. Implementing and benchmarking the locally competitive algorithm on the loihi 2 neuromorphic processor. In *Proceedings of the 2023 International Conference on Neuromorphic Systems*, pp. 1–6, 2023.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: Opportunities and challenges. *Frontiers in Neuroscience*, 12:774, 2018.

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

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

Nitin Rathi and Kaushik Roy. Diet-snn: A low-latency spiking neural network with direct input encoding and leakage and threshold optimization. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019.

Noam Shazeer. GLU variants improve transformer. *CoRR*, abs/2002.05202, 2020. URL `https://arxiv.org/abs/2002.05202`.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1631–1642, 2013.

Pao-Sheng Vincent Sun, Alexander Titterton, Anjlee Gopiani, Tim Santos, Arindam Basu, Wei D Lu, and Jason K Eshraghian. Intelligence processing units accelerate neuromorphic learning. *arXiv preprint arXiv:2211.10725*, 2022.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

Y Tay, D Bahri, D Metzler, D Juan, Z Zhao, and C Zheng. Synthesizer: rethinking self-attention in transformer models. *arXiv preprint arXiv:2005.00743*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems (NeurIPS)*, pp. 5998–6008, 2017.

Xiuying Wei, Yunchen Zhang, Xiangguo Zhang, Ruihao Gong, Shanghang Zhang, Qi Zhang, Fengwei Yu, and Xianglong Liu. Outlier suppression: Pushing the limit of low-bit transformer language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in neuroscience*, 12:331, 2018.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning (ICML)*, pp. 38087–38099. PMLR, 2023.

Rong Xiao, Yu Wan, Baosong Yang, Haibo Zhang, Huajin Tang, Derek F Wong, and Boxing Chen. Towards energy-preserving natural language understanding with spiking neural networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 31:439–447, 2022.

Man Yao, Huanhuan Gao, Guangshe Zhao, Dingheng Wang, Yihan Lin, Zhaoxu Yang, and Guoqi Li. Temporal-wise attention spiking neural networks for event streams classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 10221–10230, 2021.

Man Yao, Guangshe Zhao, Hengyu Zhang, Yifan Hu, Lei Deng, Yonghong Tian, Bo Xu, and Guoqi Li. Attention spiking neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP: 1–18, 01 2023.

Shuangfei Zhai, Walter Talbott, Nitish Srivastava, Chen Huang, Hanlin Goh, Ruixiang Zhang, and Josh Susskind. An attention free transformer. *arXiv preprint arXiv:2105.14103*, 2021.

Youhui Zhang, Peng Qu, Yu Ji, Weihao Zhang, Guangrong Gao, Guanrui Wang, Sen Song, Guoqi Li, Wenguang Chen, Weimin Zheng, et al. A system hierarchy for brain-inspired computing. *Nature*, 586 (7829):378–384, 2020.

Zhaokun Zhou, Yuesheng Zhu, Chao He, Yaowei Wang, Shuicheng YAN, Yonghong Tian, and Li Yuan. Spikformer: When spiking neural network meets transformer. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023.

Rui-Jie Zhu, Qihang Zhao, Tianjing Zhang, Haoyu Deng, Yule Duan, Malu Zhang, and Liang-Jian Deng. Tcja-snn: Temporal-channel joint attention for spiking neural networks. *arXiv preprint arXiv:2206.10177*, 2022.

APPENDIX

A Datasets and Baselines

A.1 Datasets

We conducted experiments on two major types of tasks, Natural Language Generation (NLG) and Natural Language Understanding (NLU).
For NLG tasks, we chose the following 3 classic text classification datasets to evaluate the text generation performance of SpikeGPT: Enwik8 (Mahoney, 2011), WikiText-2 (Merity et al., 2017) and WikiText-103 (Merity et al., 2017).

- Enwik8. The Enwik8 dataset is a subset of the English Wikipedia XML dump from March 2006. It contains the first 100 million bytes of the dump and is typically used to measure a model's ability to compress data. The dataset is based on the Hutter Prize, a competition for lossless compression of human knowledge. We split the tokens into three subsets: 90% for training, 5% for validation, and 5% for testing.
- WikiText-2. WikiText-2 is a natural language dataset comprising a collection of 2 million tokens derived from Wikipedia articles. This dataset is commonly utilized for benchmarking various natural language processing models.
- WikiText-103. The Wikitext-103 dataset is a large collection of text extracted from Wikipedia articles that are verified as Good or Featured. It contains over 100 million tokens and covers a wide range of topics and domains. The dataset is suitable for language modeling tasks that require long-term dependencies and rich vocabulary. The Wikitext-103 dataset is a larger and more diverse version of the Wikitext-2 dataset.

For NLU tasks, we chose the following 4 classic text classification datasets to evaluate the performance of our proposed SpikeGPT: MR (Pang & Lee, 2005), SST-5 (Socher et al., 2013), SST-2 (Socher et al., 2013), Subj. (Pang & Lee, 2004)

- MR (Pang & Lee, 2005). It consists of movie review files, labeled based on their overall sentiment polarity (positive or negative) or subjective rating.
- SST-5. The Stanford Sentiment Tree Library 5 includes 11855 sentences extracted from movie reviews for sentiment classification (Socher et al., 2013). There are 5 different categories (very negative, negative, neutral, positive, and very positive)
- SST-2 (Socher et al., 2013). It is a binary version of SST-5, with only two classes (positive and negative).
- Subj (Pang & Lee, 2004). Classify sentences in the dataset as subjective or objective.

The sample sizes and text lengths of these datasets vary. If there is no standard training test segmentation, we will follow Lv et al. (2023b) and randomly select 10% of the samples from the entire dataset as the test set.

A.2 Baselines

To verify the effectiveness on NLG and NLU tasks of our proposed SpikeGPT, we compare it with the following representative baselines:

For NLG, we list the baselines that we have selected as follows:

- **Stacked LSTM**. A model architecture that stacks multiple LSTM modules together.

- **SHA-LSTM (Merity, 2019)**. An LSTM model that follows by a single attention head layer.
- **Transformer (Vaswani et al., 2017)**. Transformer is a state-of-the-art neural network architecture, leveraging self-attention mechanisms to capture global dependencies of sequential data.
- **Reformer (Kitaev et al., 2020)**. Reformer is an extensible variant of the Transformer model. By introducing the invertible sheaf and using the local sensitive hash mechanism, it solves the problem of low memory and computing efficiency of the traditional Transformer, and realizes efficient processing of long sequences.
- **Synthesizer (Tay et al., 2020)**. Synthesizer is also a variant of Transformer, which is a model that learns to synthesize attention weights without token-token interaction.
- **Linear Transformer (Katharopoulos et al., 2020b)**. Linear Transformer is a lightweight variant of Transformer that uses linear transformation layers to construct a self attention mechanism.
- **Performer (Choromanski et al., 2020)**. A variant of Transformer that does not depend on sparsity or low-rankness assumptions and could use linear complexity to accurately estimate attention weights.
- **GPT-2 (Radford et al., 2019)**. GPT-2 is a transformer-based language model that specifically functions as a decoder. It is an extensively trained, large-scale generative model using the autoregressive paradigm. To ensure compatibility with the parameter sizes of SpikeGPT, we selected GPT-2 medium and GPT-2 small as suitable alternatives.

For NLU, the baselines we have selected are as follows:

- **LSTM (Hochreiter & Schmidhuber, 1997)**. LSTM model is a type of recurrent neural network with the ability to capture and utilize long-term dependencies in input sequences.
- **TextCNN (Kim, 2014)**. TextCNN is a convolutional neural network architecture specifically designed for text classification tasks, leveraging convolutional layers to capture local patterns and features in textual data.
- **TextSCNN (Lv et al., 2023b)**. A variant of TextCNN model that combines spiking neural networks.
- **BERT (Kenton & Toutanova, 2019)**. BERT is a bidirectional language model based on the Transformer Encoder-only architecture and an auto-encoding training paradigm.

B Details of RWKV

B.1 Token Shift

Given an input X, we perform a *token shift* operation on it as follows:

$$
\begin{aligned}
X_s &= \text{ZeroPad}_{[0,0,-1,1]}(X) \\
W_{\text{shift}} &= \left[(\frac{i}{E})^{n/N} \right], i = 1, \cdots, E \\
\mathcal{X} &= W_{\text{shift}} \odot X + (1 - W_{\text{shift}}) \odot X_s
\end{aligned}
\tag{17}
$$

where ZeroPad[1] denotes the zero padding operation, W_{shift} represents a learnable shift mask, E is the embedding size of each token, t is the current block, and T is the total number of blocks.

[1]The subscript $[0, 0, -1, 1]$ is written with PyTorch syntax in mind, where -1 clips the top row and 1 zero-pads the bottom row.

B.2 General RWKV

Inspired by the Attention Free Transformer (Zhai et al., 2021), RWKV acts as a replacement for self-attention. It reduces computational complexity by swapping matrix-matrix multiplication with a convolution that sweeps along the time dimension. We subsequently modify this step to instead operate recurrently on input data. This modification enables compatibility with recurrent SNNs, thus making it more manageable to run on limited resources.

Given an input token-shifted embedding vector $\mathcal{X}$, similar to self-attention, RWKV first applies a linear transform $R = \mathcal{X}M_R$, $K = \mathcal{X}M_K$, $V = \mathcal{X}M_V$[2]. $\mathcal{X}$ is a time-varying embedding (varying over the sequence), and so R, K, V are also time-varying. Fig. 1 depicts the sequence unrolled into a set of 2-D matrices.

M_R, M_K and M_V consist of learnable parameters, where K and V can be likened to the key and value matrices of self-attention. R is referred to as the receptance matrix, where each element indicates the acceptance of past information.

Next, the following operation is applied:

$$Y_t = \sigma(R_t) \odot \frac{\sum_{i=1}^{t} \exp(W_{(T+i-t)}) \odot \exp(K_i) \odot V_i}{\sum_{i=1}^{t} \exp(W_{(T+i-t)}) \odot \exp(K_i)} \tag{18}$$

where $\odot$ is the element-wise product, T is the sequence length, σ is the non-linearity applied to R with the default being Sigmoid; $W \in \mathbb{R}^{T \times E}$ is the positional weight decay matrix (represented as a vector unrolled over time in Fig. 1). W encodes the sequential importance of a given word on subsequent words. It is not directly learnable, but it varies over time with learnable dynamics. Long-range dependence can be captured when the dynamics are learnt to decay slowly.

Intuitively, as time t increases, the vector Y_t is dependent on a longer 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 takes the Hadamard product of the weighted result with the receptance $\sigma(R_t)$. By taking the Sigmoid of R_t, the receptance acts as a 'forget gate' by eliminating unnecessary historical information.

B.3 Self-Attention and RWKV

Distinct from the method of calculating the matching degree[3] between tokens by the self-attention mechanism, RWKV decomposes the calculation of matching degree into: $\alpha_{ij} = \sigma(R_i) \odot \exp(W_{T-i+1}) \odot \exp(K_j)$, where $\alpha_{ij} \in \mathbb{R}^E$ is a vector. Each element in α_{ij}, that is α_{ijk}, represents the matching degree at the k-th position of the embedding of the i-th and j-th tokens. In other words, it can be seen as a multi-headed RWKV with E heads, each of which has a hidden size=1, which is similar to the multi-headed self-attention (MHA) mechanism.

To gain a more comprehensive understanding of SpikeGPT, we conducted visualizations of the spike and membrane potential patterns in the Spiking RWKV layer and Spiking Receptance Feed-Forward Networks (SRFFN) layers (Fig. 3). These visualizations clearly reveal distinct differences between the Spiking RWKV and SRFFN layers, indicating diverse information representation patterns. Notably, the SRFFN layer exhibits a higher firing rate, suggesting that it may retain more information similar to Transformer FFN layers (Geva et al., 2021). It is worth noting that in various studies, outliers in Transformer-based language models have been shown to significantly impact performance, making them a crucial concern in the quantification of large language models (Wei et al., 2022). Therefore, understanding and addressing the implications of outliers in SpikeGPT are of utmost importance, particularly in the context of optimizing its overall performance and reliability. However, due to the binary nature of SNNs, these outliers cannot be expressed in activation values as in ANNs. Nevertheless, a notable observation is the presence of prominent outliers in the membrane

[2] $\{M_R, M_K, M_V\} \in \mathbb{R}^{E \times H}$, where H denotes hidden size. In RWKV, we set $E = H$.

[3] A scalar in self-attention, $\alpha_{ij} = Q_i K_j^T$

potential of individual neurons, many of which are negative. This finding suggests that SpikeGPT employs a different approach to accommodate and preserve outliers.

B.4 Positional Weight Decay

The positional weight bias matrix W is determined by three matrices, W_d, W_c and W_f, parameterized as follows:

$$W_d = \ln(W_s), W_s \in \mathbb{R}^{E\times 1} \tag{19}$$

$$W_c = \begin{bmatrix}(-T+2) & (-T+3) & (-T+4) & \cdots & -1 & 0\end{bmatrix} \in \mathbb{R}^{1\times(T-1)} \tag{20}$$

$$W_f = \begin{bmatrix}\ln(p_k) & \ln(p_k) & \cdots & \ln(p_k)\end{bmatrix} \in \mathbb{R}^{E\times 1} \tag{21}$$

where W_s is a pre-calculated matrix dependent on the layer and size of E, the vector W_d contains the decay factors for each time-step, W_c is an indicator of time-step, and W_f is the initial decay factor to avoid the constant decay phenomenon of RNNs. W_d and W_f are both learnable, and W_c is a static, pre-calculated matrix based on training time-step. p_k is a hyperparameter, which is set to 0.3 in this paper.

C Generation Examples

```
Context →                                          Who are you?
                                                   I am the AI.
                                                   Who are you?
                                                   I am Jack.
                                                   Who are you?
                                                   I am a woman.
                                                   Who are you?

---- Generated Answer 1 ----                       ---- Generated Answer 2 ----

Amy.                                               I am the woman who is,
The amazing thing is,                              in my book.
you can tell the stories about what happened to me.
```

Figure 5: Example of text generated by SpikeGPT. The model is trained on OpenWebText2.

```
Context →                                                                <start>

---- Generated Answer ----
"so what do you think of me?  "
I asked.
he was standing behind me, still looking at the posters,
and I suddenly knew what he was asking.
" what do you think of me?  "
I looked up at him , the corners of his lips pressed together.
"I don't know."
he looked at me and I was staring at him, not even realizing what he was doing.
all I could do was watch him,
look at him, listen to his voice,
and then I would fall back to the stage and look at him.
```

Figure 6: Example of text generated by SpikeGPT. The model is trained on BookCorpus.

```
Context →
In a shocking finding, scientist discovered a herd of dragons living in a remote, previously unexplored valley,
in Tibet.  Even more surprising to the researchers was the fact that the dragons spoke perfect Chinese.
```

```
---- Generated Answer ----

In a separate case, they found that their earliest ancestor were extinct.  They did not have the physical
condition of the tree.  The discovery of a Chinese zoo was the latest study by archaeologist Peter Horowitz,
who led the researchers at the University of Texas, who made the discovery a great example.  The researchers
also examined the significance of the artifacts of the bones, the humanoid skeleton, the skull, the cell parts
and the outer jaws.  The species is believed to be located in the mountains and to provide information about the
artifacts of the island.  The discovery shows that this is a large area in the surrounding area of a local island,
which has a huge influx of the populations.
```

Figure 7: Example of text generated by SpikeGPT. The model is trained on OpenWebText2.

```
Context →
Pinky
The pink ghost's AI is designed to "feel" opposite of the red ghost's behavior.  Pinky actually attempts to get
out in front of Pac-Man.  This is accomplished by setting the target 4 tiles ahead of Pac-Man's current location
in the direction that Pac-Man is travelling.  One exception to this is when Pac-Man is traveling up.  Due to an
overflow bug in the code, the calculation includes a left offset equal to the expected up offset.
```

```
---- Generated Answer ----
This is where the AI is able to see how much the AI is still in motion.  This is because the player can control
the location of the player in the real world.  It can also predict the movement of the player by moving the
screen forward towards the player.  This is why the game is so powerful when it comes to setting the target up.
This means that the player can be able to see what is happening in the world, but not exactly where the player is.
This allows the player to see where they are at and how far the player is heading.  The game can also be used
to track the location of the player, and even track the location of the player.  In addition to this, the game
can also track the location of enemies in the sky and detect targets.  It can also also monitor the location of
enemies in the area, such as the locations of buildings, schools, schools, or even buildings where a player is in
a situation where the player can see the world around them.
One of the most interesting features of the game is the ability to control the direction of the player.  This
allows players to choose between the location of the player they want to fly, or the distance of the enemy
location.  This allows players to see the direction of the player, as well as where they want to go.  The game
also supports navigation, and can also track the location of vehicles in the sky.
```

Figure 8: Example of text generated by SpikeGPT. The model is trained on OpenWebText2.

```
Context →
Aaron loves mint chocolate cake, but he requires that it be paired with mini chocolate chips, so I threw some
of those in between the layers.  I also had a few Peppermint Jo Jos on hand so I crushed them up and threw
some of those in along with some crushed meringue cookies because, why not?  It's a total smorgasbord of minty
chocolate chippy cookie crunchy goodness.  I didn't measure how much of each topping I used, but after I tasted
the finished product, I wish I had added more.  You can add anything you want- crushed candy canes, peppermint
bark, etc.  And don't be afraid to use a heavy hand.  Texture = good.
```

```
---- Generated Answer ----
Pour out of the cupcake, cut out some vanilla butter, then add sugar and vanilla.  Remove from heat until thick
and smooth, then place into the pan for 10 minutes until golden brown.  Turn on and off, stirring until the
cake's browned, as if it were black chocolate and orange syrup.
```

Figure 9: Example of text generated by SpikeGPT. The model is trained on OpenWebText2.

```
Context →
Corporal Michael P. Goeldin was an unskilled laborer from Ireland when he enlisted in Company A in November 1860.
Goldein survived the war.  Corporal Patrick O'Neal, also from Ireland, first enlisted in 1854 and served with
Company L, 3d U.S. Artillery, in Oregon.  He returned to the East Coast and enlisted in the company in 1860.
O'Neal served until 1874, when he was named superintendent of the National Cemetery at Willets Point, New York.
Corporal Benjamin Browne was a shoemaker from Orange County, New York.  In August 1862, he enlisted in the
newly formed 124th New York Volunteers, and was one of sixty-one men who transferred into Company A that October.
Browne reenlisted in the company in February 1864 while it was camped at Brandy Station.  He returned to civilian
life after completing his enlistment in 1867.  On 10 June, Artificer William Collins was promoted to corporal,
probably to fill a combat leadership void for the crossing of the James River.  Collins's service record does not
reflect the qualities he demonstrated to earn this promotion, but he had obviously overcome some serious problems.
Born in Sacketts Harbor, New York, Collins enlisted in the company in December 1853 at the age of twenty-two, and
reenlisted in December 1858.  Just a month before the war began in April 1861, Collins went "over the hill" and
was not caught until three years later.  Returned to the company on 22 March 1864,
```

```
---- Generated Answer ----

he was tried for possession of marijuana and attempted first degree murder.
For more information on Collins on Twitter visit www.facebook.com/JoeCollins
In April 1879, Collins was sentenced to life in prison for two counts of first degree murder:  first degree first
degree first degree murder, second degree second degree second degree murder,
```

Figure 10: Example of text generated by SpikeGPT. The model is trained on OpenWebText2.

```
Context →
Prehistoric man sketched an incredible array of prehistoric beasts on the rough limestone walls of a cave in
modern day France 36,000 years ago.  Now, with the help of cutting-edge technology, those works of art in the
Chauvet-Pont-d'Arc Cave have been reproduced to create the biggest replica cave in the world.  The manmade
cavern named the Caverne du Pont-d'Arc has been built a few miles from the original site in Vallon-Pont-D'arc
in Southern France and contains 1,000 painstakingly-reproduced drawings as well as around 450 bones and other
features...  Cavemen and women sketched an incredible array of prehistoric beasts on the rough limestone walls of
a cave 36,000 years ago and now a replica has been created (pictured) that show how beautiful the cave was in the
70s and 80s
```

```
---- Generated Answer ----

Scroll down for video
Growth of a rare cave on a cliff near the river level near the Great Wall of Barcelona (pictured right):  The
world's oldest stone cave in Italy has been discovered in part because of ancient cave erosion.
It has become a key part of modern archaeology to determine how the pyramids lived before the Stone Age.
In an open letter dated 19 June, archaeologists from the University of Barcelona say they found the excavation
and excavation on the site 'significant'.
```

Figure 11: Example of text generated by SpikeGPT. The model is trained on OpenWebText2.