\documentclass[12pt]{article}

\title{Neuromorphic Computing Based Transformer Model}

\author{Kaseri Abdul Rehman}

\usepackage{amsmath}

\usepackage{graphicx}

\usepackage{amsfonts}

\usepackage{graphicx}

\usepackage{cite}

\usepackage{longtable}

\usepackage{array}

\usepackage{enumitem}

\usepackage{geometry}

\usepackage{caption}

\geometry{a4paper,margin=1in}

\begin{document}

\maketitle

\clearpage

\tableofcontents

\clearpage

\section{Abstract}

Deep learning models pose increasing energy and computational challenges. One intriguing answer to these challenges is neuromorphic computing, which draws inspiration from the design and dynamics of the human brain \cite{indiveri2015memory}. Transformers are very resource-intensive and are known for their outstanding performance in tasks like computer vision and natural language processing \cite{vaswani2017attention}\cite{devlin2019bert}. Therefore, creative ways to improve their efficiency are needed. This work explores the use of neuromorphic transformers for event-driven computation and spiking neural networks \cite{roy2019towards}.

We start by going over the foundations of transformers and neuromorphic computing, including event-driven binary spike communication and their self-attention mechanism and computational complexity\cite{davies2018loihi}. The theoretical and practical elements of neuromorphic transformers are then covered in detail \cite{chen2020neuromorphic}.

We investigate spike-driven transformers through in-depth study, which are applied to text and picture categorization \cite{chen2020neuromorphic}. According to our research, neuromorphic transformers provide notable benefits in terms of latency and energy efficiency over traditional transformers because of their spike-based operations and linear complexity self-attention\cite{litvak2017learning}.

The findings highlight the potential of neuromorphic transformers in real-world applications, where efficiency and rapid processing are essential.

By offering insights into the computational benefits of transformers and opening the door for future advancements in energy-efficient AI systems, this research contributes to our understanding of how to integrate transformers with neuromorphic computing.

\section{Introduction}

Artificial intelligence (AI) and machine learning (ML) have advanced at a rapid pace, transforming many industries. Deep learning models—in particular, transformers—have proven essential in computer vision, natural language processing, and other fields. Through their self-attention mechanisms, transformers have shown remarkable ability in managing sequential data and capturing long-range relationships \cite{vaswani2017attention}\cite{devlin2019bert}. However, energy efficiency and real-time processing face major problems due to transformers' growing complexity and computing needs \cite{brown2020language}.

Inspired by the structure and operations of the human brain, neuromorphic computing presents a viable way to address these issues. Neuromorphic systems seek to achieve greater efficiency and adaptability by using event-driven computation and emulating neural structures \cite{indiveri2015memory}\cite{roy2019towards}. Spiking neural networks (SNNs) are used by neuromorphic hardware, like IBM's TrueNorth and Intel's Loihi, to carry out computations similarly to biological neurons, potentially providing significant increases in processing speed and energy efficiency \cite{davies2018loihi}\cite{chen2020neuromorphic}.

Despite the advantages of both transformers and neuromorphic computing, integrating these technologies poses a significant research challenge. The primary objective of this paper is to explore how spike-driven transformers can improve traditional transformers in terms of accuracy and efficiency .Specifically, we aim to investigate the computational advantages and trade-offs associated with spike-driven transformer, focusing on event-driven binary spike communication, achieving self-attention with linear complexity, and understanding the operations between spike-form queries, keys, and values with masking \cite{litvak2017learning}\cite{patel2015neuromorphic}.

\end{enumerate}

\section{Literature Review}

\subsection{Neuromorphic Computing}

The goal of the discipline of neuromorphic computing is to develop extremely effective and adaptable computing systems by simulating the neural architecture and functionality of the human brain. Carver Mead first presented the idea in the 1980s, focusing on how to duplicate neuro-biological architectures using very-large-scale integration (VLSI) \cite{mead1990neuromorphic}. This method focuses on building artificial neuronal networks with synapses that exchange information through action potentials or spikes, just like real neurons do.

The event-driven nature of neuromorphic computing is one of its main benefits. Clock-driven processes, which are the foundation of traditional digital systems, can result in inefficiencies and excessive power consumption, particularly in large-scale systems. Neuromorphic systems, on the other hand, function asynchronously, meaning that calculations take place only in response to noteworthy alterations or occurrences. Because the system is idle until it detects an event that needs to be processed, this event-driven method can dramatically lower power consumption and improve processing efficiency \cite{schuman2017survey}.

Spiking neural networks (SNNs) and specific neuromorphic chips like IBM's TrueNorth and Intel's Loihi are examples of recent developments in neuromorphic hardware. These systems have the potential to be used in fields that need real-time processing and low power consumption because of their ability to process information more like biological brains\cite{furber2016large}.

\subsection{Von Neumann Architecture v/s Neuromorphic Architecture}

\subsubsection{Von Neumann Architecture}

Von Neumann architecture, sometimes referred to as the Princeton architecture, is the computer architecture designed by mathematician and physicist John von Neumann in 1945. It basically outlines a design framework of a stored-program digital computer running a single processor designed for instruction-pattern execution in sequence. This architecture typically consists of a central processing unit, memory unit, and input/output mechanisms. The data and instructions are stored in the same memory and transported among the CPU and memory through “buses.” This architecture remains the backbone of all commercial computers and is applicable to a broad spectrum of computer use, from general-purpose computing to highly specialized and complex ones such as scientific computation. However, the CPU’s execution rate is determined by the mixture of the three primary architecture components and their interaction and not the most substantial could be limited by the so-called “Von Neumann bottleneck,” i.e., the transfer rate between memory and CPU becomes a limiting factor.

\subsubsection{Neuromorphic Architecture}

The Neuromorphic Architecture is a computing model with structure and function inspired by the human brain. The key in this type of architecture is that neurons and synapses are the primary computational elements, thus replicating both how information processing in brain happens (in an event-driven manner) but also doing it parallelly. Neuromorphic systems, in contrast to the sequential execution model of Von Neumann systems, implement an asynchronous mode of operation whereby neurons fire spikes or events that propogate through a network composed primarily (or entirely) of interconnected neurons. By integrating memory and processing, the power consumption is minimized so as to reduce latency. Neuromorphic computers are built to meet those requirements and can handle tasks like pattern recognition, robotics or sensory processing that needs real-time response. Well known examples of these so called neuromorphic chips are Intel Loihi and IBM TrueNorth, designed to do low-power computations in an energy efficient way mimicking the principles of spiking neural networks.

\subsubsection{Comparison between Von Neumann Architecture and Neuromorphic Architecture}

\begin{longtable}{|>{\raggedright\arraybackslash}p{5cm}|>{\raggedright\arraybackslash}p{5cm}|>{\raggedright\arraybackslash}p{5cm}|}

\hline

\textbf{Feature/Aspect} & \textbf{Von Neumann Architecture} & \textbf{Neuromorphic Architecture} \\

\hline

\endfirsthead

\multicolumn{3}{c}%

{{\bfseries \tablename\ \thetable{} -- continued from previous page}} \\

\hline

\textbf{Feature/Aspect} & \textbf{Von Neumann Architecture} & \textbf{Neuromorphic Architecture} \\

\hline

\endhead

\hline \multicolumn{3}{r}{{\bfseries Continued on next page}} \\

\endfoot

\hline

\endlastfoot

Basic Concept & Sequential processing & Parallel, event-driven processing \\

\hline

Inspiration & Traditional computing models & Brain-inspired computing models \\

\hline

Processing Units & Central Processing Unit (CPU) & Neurons (processing elements) \\

\hline

Memory Structure & Separate memory and processor units & Integrated memory and processing elements (synapses) \\

\hline

Data Transfer & Data and instructions are transferred between memory and CPU & Data is transmitted via spikes or events \\

\hline

Data Handling & Data handled in a step-by-step manner & Data handled in parallel, event-based manner \\

\hline

Power Consumption & Generally higher, especially with increased processing power & Lower power consumption due to event-driven nature \\

\hline

Latency & Can be high due to sequential processing and data transfer & Low latency due to parallel and local processing \\

\hline

Efficiency & Efficiency decreases with complex, large-scale computations & High efficiency for tasks requiring parallel processing and adaptability \\

\hline

Programming Model & Traditional programming languages and paradigms & New paradigms, often event-based or spike-driven \\

\hline

Key Components & ALU (Arithmetic Logic Unit), control unit, memory, input/output & Neurons, synapses, axons, dendrites \\

\hline

Example Technologies & General-purpose CPUs, GPUs & Intel Loihi, IBM TrueNorth, BrainChip Akida \\

\hline

Scalability & Limited by the separation of memory and processing units & Highly scalable with many neurons and synapses \\

\hline

Adaptability & Limited adaptability; requires explicit programming & High adaptability through learning mechanisms \\

\hline

Typical Applications & General computing, scientific computing, traditional AI & Real-time pattern recognition, robotics, sensory processing \\

\hline

Advantage & Well-established, mature ecosystem & Energy efficiency, real-time processing, parallelism \\

\hline

Challenge & Power consumption, memory bottlenecks & New programming models, specialized hardware \\

\hline

Interconnects & Buses, interconnects between CPU and memory & Synaptic connections, event-driven communication \\

\hline

Learning Capability & No inherent learning; relies on software algorithms & Inherent learning capability through synaptic plasticity \\

\hline

Fault Tolerance & Requires redundancy and error-checking mechanisms & Inherent fault tolerance through network plasticity \\

\hline

Development Tools & Mature tools and frameworks & Emerging tools and frameworks \\

\hline

Data Storage & Stored in external memory units & Distributed within the network (synaptic weights) \\

\hline

\end{longtable}

\subsection{Intel Loihi and IBM TrueNorth v/s GPUs}

\subsubsection{Intel Loihi}

Intel Loihi is a neuromorphic chip that seeks to replicate the way our brains work. This neuromorphic chip introduced by Intel in 2017 is designed with a spiking neural network (SNN) event-driven architecture which has of the order 128000 neurons and 128 million synapses to provide efficient, real-time processing. The use of spikes, as opposed to the constant data flow in traditional processors reduces power consumption significantly for Loihi. It provides on-chip learning and adaptation, which are ideal for applications needing real-time pattern recognition, robotics and edge AI. With its high parallelism, and low energy consumption ,Loihi is best suited for brain-inspired computing tasks.

\\

\begin{itemize}

\item \textbf{Key Features:}

\begin{itemize}

\item Implements SNNs with on-chip learning.

\item Uses spikes (discrete events) to process information.

\item Consumes very low power compared to traditional processors.

\item On-chip plasticity rules for learning and adaptation.

\item Supports hierarchical temporal memory and other biologically inspired models.

\end{itemize}

\item \textbf{Applications:}

\begin{itemize}

\item Real-time pattern recognition

\item Robotics

\item Edge AI applications

\item Adaptive control systems

\end{itemize}

\item \textbf{Advantages:}

\begin{itemize}

\item Energy-efficient due to event-driven processing.

\item High parallelism.

\item Suitable for tasks that benefit from temporal dynamics and sparse event-based data.

\end{itemize}

\item \textbf{Challenges:}

\begin{itemize}

\item Programming complexity due to new paradigms.

\item Specialized hardware not as versatile as GPUs for general-purpose tasks.

\end{itemize}

\end{itemize}

\subsubsection{IBM TrueNorth}

Another major milestone in the field of neuromorphic computing was reached with IBM TrueNorth, which emerged from DARPA SyNAPSE program’s efforts to develop a novel brain-inspired many-core form factor ans was released in 2014. TrueNorth Is an event-driven digital architecture with 1 million neurons and 256 million synapses. This chip is designed to operate with ultra-low power, making it highly efficient for sensory processing and cognitive computing. TrueNorth’s architecture is fixed, which means it is optimized for specific neuromorphic tasks but lacks the flexibility of more general-purpose processors. Yet its efficiency and also scalability make it fit for big range neuromorphic applications. \\

\begin{itemize}

\item \textbf{Key Features:}

\begin{itemize}

\item 1 million neurons and 256 million synapses.

\item Extremely low power consumption.

\item Implements a core array of neurosynaptic cores, each simulating neural processes..

\item Fixed architecture designed for specific neuromorphic tasks.

\end{itemize}

\item \textbf{Applications:}

\begin{itemize}

\item Sensory processing (vision, audition)

\item Cognitive computing

\item Real-time data analytics

\end{itemize}

\item \textbf{Advantages:}

\begin{itemize}

\item Ultra-low power consumption.

\item High efficiency for SNN-based tasks.

\item Scalable architecture suitable for large-scale neuromorphic applications.

\end{itemize}

\item \textbf{Challenges:}

\begin{itemize}

\item Programming complexity.

\item Fixed architecture limits flexibility for other types of neural networks.

\item Limited support for on-chip learning (compared to Loihi).

\end{itemize}

\end{itemize}

\subsubsection{GPUs}

Although GPUs were originally for rendering graphics, today they are central to anything which is related to Artificial Intelligence and Machine Learning. GPUs are general-purpose parallel processors, unlike neuromorphic chips and they thrive on computationally intensive tasks across a broad spectrum. The highlights of these cards are the massively parallel, SIMD technology (Single Instruction Multiple Data) that lets them execute hundreds or thousands of calculations simultaneously. GPUs can be used across a range of frameworks from CUDA, OpenCL, and TensorFlow making them essential for deep learning algorithms as well as image and video processing to highly complex scientific computing problems. However, this is at the cost of higher power consumption compared to dedicated neuromorphic chips. \\

\begin{itemize}

\item \textbf{Key Features:}

\begin{itemize}

\item High throughput for parallel tasks.

\item Flexible and programmable for various types of computations (not just graphics).

\item Support for frameworks like CUDA, OpenCL for general-purpose computing.

\item Capable of running traditional deep learning models (CNNs, RNNs, Transformers).

\end{itemize}

\item \textbf{Applications:}

\begin{itemize}

\item Deep learning (training and inference)

\item Scientific computing

\item Image and video processing

\item Cryptography and financial modeling

\end{itemize}

\item \textbf{Advantages:}

\begin{itemize}

\item Versatile and widely used across industries.

\item High performance for both deep learning and general-purpose parallel tasks.

\item Mature software ecosystem and broad developer support.

\end{itemize}

\item \textbf{Challenges:}

\begin{itemize}

\item Higher power consumption compared to neuromorphic chips.

\item Inefficient for tasks requiring low power and real-time event processing.

\item Not optimized for SNNs and other neuromorphic models without significant software adaptation.

\end{itemize}

\end{itemize}

\subsubsection{Comparison of Intel Loihi, IBM TrueNorth v/s GPUs}

\begin{longtable}{|p{4cm}|p{3.5cm}|p{3.5cm}|p{3.5cm}|}

\hline

\textbf{Feature/Aspect} & \textbf{Intel Loihi} & \textbf{IBM TrueNorth} & \textbf{GPUs} \\

\hline

\endfirsthead

\multicolumn{4}{c}%

{{\bfseries \tablename\ \thetable{} -- continued from previous page}} \\

\hline

\textbf{Feature/Aspect} & \textbf{Intel Loihi} & \textbf{IBM TrueNorth} & \textbf{GPUs} \\

\hline

\endhead

\hline \multicolumn{4}{r}{{\bfseries Continued on next page}} \\

\endfoot

\hline

\endlastfoot

Type & Neuromorphic Chip & Neuromorphic Chip & Parallel Processor \\

\hline

Architecture & Event-driven, SNN & Digital, event-driven, SNN & Massively parallel, SIMD \\

\hline

Inspiration & Brain-inspired computing & Brain-inspired computing & Originally graphics rendering \\

\hline

Neurons and Synapses & Configurable & 1 million neurons, 256 million synapses & Varies (not neuron/synapse based) \\

\hline

Power Consumption & Very low & Ultra-low & Higher than neuromorphic chips \\

\hline

Parallelism & High & High & Very high \\

\hline

Processing Type & Spiking neural networks (SNNs) & Spiking neural networks (SNNs) & General-purpose parallel processing \\

\hline

Learning & On-chip learning (plasticity) & Limited on-chip learning & Depends on software, e.g., deep learning \\

\hline

Flexibility & Specialized for neuromorphic tasks & Fixed architecture, limited flexibility & Highly versatile \\

\hline

Key Features & On-chip learning, real-time processing, low power & Low power, large scale SNNs & High throughput, general-purpose, mature software ecosystem \\

\hline

Programming Complexity & High (new paradigms) & High & Moderate to high \\

\hline

Software Ecosystem & Emerging & Emerging & Mature (CUDA, OpenCL, TensorFlow, etc.) \\

\hline

Main Applications & Real-time pattern recognition, robotics, edge AI, adaptive control & Sensory processing, cognitive computing, real-time analytics & Deep learning, scientific computing, image/video processing, cryptography \\

\hline

Advantages & Energy-efficient, real-time, high parallelism & Ultra-low power, efficient SNNs & High performance, versatile, wide adoption \\

\hline

Challenges & Specialized hardware, programming complexity & Fixed architecture, programming complexity & Higher power consumption, not optimized for SNNs \\

\hline

Companies & Intel & IBM & NVIDIA, AMD, etc. \\

\hline

Release Year & 2017 (Loihi) & 2014 (TrueNorth) & 1999 (First GPU by NVIDIA) \\

\hline

Current Generation & Loihi 2 & TrueNorth (no new versions reported) & Latest GPUs (e.g., NVIDIA RTX series) \\

\hline

Notable Projects/Usage & Intel's neuromorphic research & DARPA SyNAPSE program & DeepMind's AlphaGo, autonomous vehicles, gaming \\

\hline

\end{longtable}

\subsection{Spiking Neural Networks(SNNs)}

Spiking Neural Networks (SNNs) represent a class of artificial neural networks that attempt to emulate the biological processes of the brain more closely than traditional neural networks. Unlike traditional neural networks that process information in continuous time using real-valued activations, SNNs use discrete events called spikes.

\begin{figure}[h]

\centering

\includegraphics[width=0.5\textwidth]{snn.png}

\caption{Architecture of SNN}

\label{fig:SNN architecture}

\end{figure}

SNNs have several key features and concepts:

\begin{itemize}

\item \textbf{Event-Driven Processing}:

\begin{itemize}

\item Compared to typical neural networks that need constant computation, SNNs may be more energy-efficient because they only analyze information when spikes occur.

\item SNNs exhibit sparse and asynchronous activity because neurons in these networks do not fire until they sense a spike.

\end{itemize}

\item \textbf{Temporal Dynamics}:

\begin{itemize}

\item Spike timing is a means of carrying information, which enables SNNs to efficiently encode and process temporal patterns.

\item Rate coding and temporal coding are two examples of temporal coding schemes (where the precise timing of spikes is critical) and where the spike rate reflects information.

\end{itemize}

\item \textbf{Biological Plausibility}:

\begin{itemize}

\item SNNs use neuron models, like the Integrate-and-Fire or Hodgkin-Huxley models, that simulate the action of biological neurons.

\ Spike-Timing Dependent Plasticity (STDP), one of the biological learning processes, serves as a model for many SNN learning mechanisms.

\end{itemize}

\end{itemize}

SNNs offer several advantages:

\begin{itemize}

\item \textbf{Energy Efficiency}:

\begin{itemize}

\item SNNs can be much more energy-efficient because they are event-driven, which is advantageous for real-time applications and hardware implementations.

\end{itemize}

\item \textbf{Temporal Information Processing}:

\begin{itemize}

\item SNNs are well-suited for jobs involving sequential information because they are excellent at processing temporal data and can capture complex time-dependent patterns.

\end{itemize}

\item \textbf{Robustness and Adaptability}:

\begin{itemize}

\item Because of their biologically inspired principles, SNNs can be more resilient to noise and flexible in changing situations.

\end{itemize}

\end{itemize}

However, SNNs also face several challenges:

\begin{itemize}

\item \textbf{Training Complexity}:

\begin{itemize}

\item The non-differentiability of spike events and the discrete and sparse character of spikes make training SNNs more complicated than training standard neural networks.

\end{itemize}

\item \textbf{Scalability}:

\begin{itemize}

\item Research is now being conducted to find effective training techniques, as scaling SNNs to big datasets and designs remains a difficulty.

\end{itemize}

\item \textbf{Hardware Implementation}:

\begin{itemize}

\item SNNs are ideally suited for neuromorphic hardware, however creating such hardware is difficult and necessitates technological developments.

\end{itemize}

\end{itemize}

\subsection{Transformers}

Transformers have revolutionized field of natural language processing (NLP) and are even being applied to other realms such as vision or reinforcement learning. They were introduced by Vaswani et al(2017). The Transformer model leverages a self-attention mechanism that allows it to dynamically assign importance weights for different tokens in a sequence \cite{vaswani2017attention}.It provided a way to better catch long-range dependencies as compared with previous models like recurrent neural networks (RNNs) and convolutional neural networks (CNNs).

Despite their success, Transformers have significant computational challenges, primarily due to the self-attention mechanism's quadratic complexity. The self-attention computation takes \( O(n^2) \) time for a sequence of length \n\, resulting in high memory and processing demands for long sequences. However, this computational burden helps control the scalability of Transformers and makes them less suitable for applications requiring real-time processing or operating under constrained computational resources\cite{brown2020language}.

Solutions to alleviate these constraints have involved the construction of models such as the Reformer which attempts to reduce self-attention's computationally prohibitive quadratic complexity via techniques like Locality-Sensitive Hashing (LSH)\cite{kitaev2020reformer}. But these come with compromises in accuracy or involve substantial modifications to the original Transformer design.\\

Transformers have several key features and concepts:

\begin{figure}[h]

\centering

\includegraphics[width=0.5\textwidth]{transformerarchitecture.png}

\caption{The Transformer-Model Architecture}

\label{fig:tranformer architecture}

\end{figure}\\

\begin{itemize}

\item \textbf{Attention Mechanism}:

\begin{itemize}

\item Central to the Transformer architecture is the self-attention mechanism, which allows the model to weigh the importance of different words in a sentence, irrespective of their position.

\item This mechanism enables the model to capture long-range dependencies more effectively than traditional recurrent neural networks (RNNs) and convolutional neural networks (CNNs).

\end{itemize}

\item \textbf{Encoder-Decoder Architecture}:

\begin{itemize}

\item Transformers are composed of an encoder and a decoder. The encoder processes the input sequence, and the decoder generates the output sequence.

\item Each encoder and decoder consists of a stack of identical layers, each containing a multi-head self-attention mechanism and a feed-forward neural network.

\end{itemize}

\item \textbf{Parallelization}:

\begin{itemize}

\item Unlike RNNs, which process sequences sequentially, Transformers allow for parallelization. This significantly speeds up training and makes them scalable to larger datasets.

\end{itemize}

\item \textbf{Positional Encoding}:

\begin{itemize}

\item Since Transformers do not inherently capture the order of words (due to the lack of recurrence), they use positional encodings to inject information about the position of each token in the sequence.

\end{itemize}

\end{itemize}

The Transformer architecture consists of two main components:

\begin{itemize}

\item \textbf{Encoder}:

\begin{itemize}

\item The encoder stack consists of multiple identical layers. Each layer has two main components:

\begin{itemize}

\item \textbf{Multi-Head Self-Attention}: Computes attention scores for each token relative to every other token in the sequence.

\item \textbf{Feed-Forward Neural Network}: A position-wise fully connected feed-forward network applied to each position separately and identically.

\end{itemize}

\item Each sub-layer in the encoder (self-attention and feed-forward network) is followed by layer normalization and residual connections to aid in training deeper networks.

\end{itemize}

\item \textbf{Decoder}:

\begin{itemize}

\item The decoder stack is similar to the encoder but with an additional cross-attention layer. Each decoder layer has three main components:

\begin{itemize}

\item \textbf{Masked Multi-Head Self-Attention}: Ensures that predictions for a certain position can depend only on known outputs up to that position.

\item \textbf{Multi-Head Attention over Encoder Outputs}: Attends to the encoder's output.

\item \textbf{Feed-Forward Neural Network}: Similar to the encoder's feed-forward network.

\end{itemize}

\item Like the encoder, each sub-layer in the decoder is followed by layer normalization and residual connections.

\end{itemize}

\end{itemize}

The attention mechanism in Transformers includes:

\begin{itemize}

\item \textbf{Scaled Dot-Product Attention}:

\begin{itemize}

\item Computes attention scores using the dot product of query (Q) and key (K) vectors, scales them, and applies a softmax function to obtain attention weights.

\item The attention output is a weighted sum of values (V) based on these attention weights.

\end{itemize}

\item \textbf{Multi-Head Attention}:

\begin{itemize}

\item Instead of performing a single attention function, multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions.

\item This is achieved by linearly projecting the queries, keys, and values multiple times with different learned linear projections, performing attention in parallel, and then concatenating and projecting the results.

\end{itemize}

\end{itemize}

Transformers use positional encoding to give the model information about the relative or absolute position of the tokens in the sequence. These encodings can be learned or fixed and typically use sinusoidal functions to generate different frequencies for different positions.

Transformers have several advantages:

\begin{itemize}

\item \textbf{Handling Long-Range Dependencies}:

\begin{itemize}

\item Self-attention mechanisms allow Transformers to capture relationships between distant tokens more effectively than RNNs.

\end{itemize}

\item \textbf{Parallelization}:

\begin{itemize}

\item Enables faster training and inference due to the ability to process sequences in parallel.

\end{itemize}

\item \textbf{Scalability}:

\begin{itemize}

\item Transformers can be scaled to handle large datasets and model sizes, leading to state-of-the-art performance on various tasks.

\end{itemize}

\end{itemize}

Despite their advantages, Transformers also face challenges:

\begin{itemize}

\item \textbf{Computationally Intensive}:

\begin{itemize}

\item Training large Transformer models requires significant computational resources and memory.

\end{itemize}

\item \textbf{Data Intensive}:

\begin{itemize}

\item Large amounts of training data are necessary to achieve high performance, particularly for tasks involving nuanced language understanding.

\end{itemize}

\item \textbf{Overfitting}:

\begin{itemize}

\item Transformers can overfit on small datasets, requiring regularization techniques and careful hyperparameter tuning.

\end{itemize}

\end{itemize}

\subsection{Comparison of Transformer, Spiking Neural Networks (SNN), and Spike-Driven Transformer}

\begin{longtable}{|>{\raggedright}p{3cm}|>{\raggedright}p{3.5cm}|>{\raggedright}p{3.5cm}|>{\raggedright\arraybackslash}p{3.5cm}|}

\hline

\textbf{Feature} & \textbf{Transformer} & \textbf{Spiking Neural Networks (SNN)} & \textbf{Spike-Driven Transformer} \\

\hline

\endfirsthead

\hline

\textbf{Feature} & \textbf{Transformer} & \textbf{Spiking Neural Networks (SNN)} & \textbf{Spike-Driven Transformer} \\

\hline

\endhead

\hline

\endfoot

\textbf{Computation Mechanism} & Continuous valued activations & Discrete spikes & Combines continuous activations with spikes \\

\hline

\textbf{Core Operation} & Self-Attention & Spike-based computation & Self-Attention with spike-based processing \\

\hline

\textbf{Temporal Dynamics} & Limited, primarily through positional encoding & Integral part of processing & Leverages spike timing and self-attention \\

\hline

\textbf{Energy Efficiency} & Moderate to high (depends on hardware) & High (event-driven processing) & Improved efficiency due to spiking nature \\

\hline

\textbf{Biological Plausibility} & Low & High & Moderate (combines aspects of both) \\

\hline

\textbf{Training Paradigm} & Gradient Descent & Spike-Timing Dependent Plasticity (STDP), Hebbian Learning & Hybrid training methods \\

\hline

\textbf{Scalability} & Highly scalable with large datasets & Challenging to scale & Scalable with improved efficiency \\

\hline

\textbf{Latency} & Moderate to high & Low (due to event-driven nature) & Low \\

\hline

\textbf{Data Type} & Primarily text and sequential data & Temporal data, real-time processing & Sequential data with temporal features \\

\hline

\textbf{Applications} & NLP, text generation, translation & Robotics, neuromorphic computing, real-time processing & Efficient NLP, energy-constrained applications \\

\hline

\textbf{Handling Long-Range Dependencies} & Excellent due to self-attention & Limited, depends on network architecture & Excellent due to combined self-attention and temporal processing \\

\hline

\textbf{Hardware Compatibility} & General-purpose GPUs and TPUs & Specialized neuromorphic hardware & General-purpose and neuromorphic hardware \\

\hline

\textbf{Real-Time Processing} & Less suitable & Highly suitable & Highly suitable \\

\end{longtable}

\subsection{Integrating Neuromorphic Computing with Transformers}

Combining neuromophic computing principles with Transformer models offers an interesting new avenue to tackle the computational inefficiencies of typical Transformers. Taking advantage of the event-driven nature demonstrated in neural systems, Neuromorphic Transformers introduce new forms of self-attention mechanisms to reduce power consumption.

In such neuromorphic systems, event-driven binary spike communication is adopted - so that neurons send out spikes (binary signals) only when the input signal surpasses a specific threshold. This type of communication is asynchronous as it enables neurons to function separately and initiate events depending on local circumstances. We could swap the continuous-valued vectors (for queries, keys and values) in Transformers by spike trains to lead a more energy efficient spiking self-attention mechanism\cite{liu2018spiking}.

\subsection{Spike-based Self-Attention Mechanism}

In a neuromorphic adaptation of the Transformer, rather than using continuous-valued query,key and value vectors we replace them by sequences called spike trains that consist of binary events which indicate whether or not a Spike occurred at different times. The spike-based representation permits information to be encoded as the time and frequency of spikes, thus paralleling processing in biological neural networks \cite{indiveri2015memory}.

\begin{enumerate}

\item Encoding: Information in the input sequence is encoded into spike trains, where the timing and frequency of spikes convey the data.

\item Processing: Neurons representing queries, keys, and values interact through their spike timings. The alignment or coincidence of spikes determines the attention weights.

\item Event-driven Processing: Neurons fire spikes only when they detect significant inputs, leading to selective computation and reducing unnecessary operations compared to traditional Transformers \cite{boahen2005neuromorphic}.

\end{enumerate}

\subsection{Linear Complexity Self-Attention}

One of the most significant advantages of neuromorphic Transformers is their potential to achieve linear complexity in self-attention. By leveraging the sparsity in spike trains, neuromorphic Transformers ensure that only relevant token interactions are computed, thus scaling linearly with sequence length.

\begin{enumerate}

\item Linear Scaling: Sparse interactions and efficient computation ensure that the number of operations scales with the number of spikes, not the sequence length, leading to linear complexity \( O(n) \).

\item Sparse Communication: The sparsity inherent in spike-based communication reduces redundant operations, focusing computational resources on significant interactions \cite{devlin2019bert}.

\end{enumerate}

\subsection{Spike-Driven Transformer}

Based on our extensive experience with machine learning, the Spike-driven Transformer \cite{Yao2024SpikeDrivenTransformer} is a major breakthrough in this area by Man Yao and Jiakui Hu. This unifies the spike-driven paradigm across not just spiking neural networks but throughout an application-level Transformer architecture, such that all operations are event-driven and binary-spike based. This approach instead uses sparse additions, eliminating the energy-intensive matrix multiplications.

\subsubsection{Overall Architecture}

\begin{figure}[h]

\centering

\includegraphics[width=1\textwidth]{spiketransformer.png}

\caption{An overview of Spike driven Transformer}

\label{fig:spike driven transformer}

\end{figure}

Figure 3 illustrates the Spike-driven Transformer, comprising four components: Spiking Patch Splitting (SPS), SDSA, MLP, and a linear classification head. SPS follows the design in \cite{zhou2023spikformer}. A 2D image sequence \( I \in \mathbb{R}^{T \times C \times H \times W} \) is split into \( N \) flattened spike patches \( s \) with \( D \) dimensional channels using the Patch Splitting Module (PSM), where \( T \) (images are repeated \( T \) times in the static dataset as input), \( C \), \( H \), and \( W \) denote timestep, channel, height and width of the 2D image sequence. Another Conv layer is then used to generate Relative Position Embedding (RPE).:

\[

\begin{aligned}

u & = \text{PSM}(I), \quad I \in \mathbb{R}^{T \times C \times H \times W}, \quad u \in \mathbb{R}^{T \times N \times D} \\

s & = \text{SN}(u), \quad s \in \mathbb{R}^{T \times N \times D} \\

\text{RPE} & = \text{BN}(\text{Conv2d}(s)), \quad \text{RPE} \in \mathbb{R}^{T \times N \times D} \\

U\_0 & = u + \text{RPE}, \quad U\_0 \in \mathbb{R}^{T \times N \times D}

\end{aligned}

\]

where \( u \) and \( U\_0 \) are the output membrane potentials of PSM and SPS, respectively, and \( \text{SN}(\cdot) \) represents the spike neuron layer.\( U\_0 \) is then processed by the Spike-driven Transformer encoder comprising SDSA and MLP blocks with residual connections applied to membrane potentials:

\[

\begin{aligned}

S\_0 & = \text{SN}(U\_0), \quad S\_0 \in \mathbb{R}^{T \times N \times D} \\

U'\_l & = \text{SDSA}(S\_{l-1}) + U\_{l-1}, \quad U'\_l \in \mathbb{R}^{T \times N \times D}, \quad l = 1...L \\

S'\_l & = \text{SN}(U'\_l), \quad S'\_l \in \mathbb{R}^{T \times N \times D}, \quad l = 1...L \\

S\_l & = \text{SN}(\text{MLP}(S'\_l) + U'\_l), \quad S\_l \in \mathbb{R}^{T \times N \times D}, \quad l = 1...L \\

Y & = \text{CH}(\text{GAP}(S\_L))

\end{aligned}

\]

\subsubsection{Membrane Shortcut in Spike-driven Transformer}

Residual connections are fundamental in Transformer architecture. There are three shortcut techniques in existing Conv-based SNNs : Vanilla Res-SNN , Spike-Element-Wise (SEW) Res-SNN , and Membrane Shortcut (MS) Res-SNN . The current SNN community lacks a uniform standard shortcut, with SEW shortcuts being used in spiking Transformers . Our approach employs membrane shortcuts for the following reasons:

\begin{enumerate}

\item Spike-driven Functionality: Only binary spikes can transform matrix multiplications into sparse additions. MS shortcuts ensure binary spike signals in spike tensors, unlike SEW shortcuts, which output multi-bit spikes.

\item High Performance: MS-Res-SNN shows higher task accuracy compared to SEW-Res-SNN.

\item Bio-plausibility: MS shortcuts optimize membrane potential distribution, aligning with neuroscience-inspired SNN optimization methods.

\item Dynamical Isometry:MS-Res-SNN complies with dynamical isometry theory , which explains well-behaved deep neural networks.

\end{enumerate}

\subsection{Spike-driven Self Attention}

\textbf{Spike-Driven Self-Attention (SDSA)}:Figure 4(b) left illustrates SDSA-V1. Given a spike input sequence \( S \in \mathbb{R}^{T \times N \times D} \),where \( T \) (images are repeated \( T \) times in the static dataset as input), \( N \) is the number of Flattened Spike Patches the 2-D image sequence is split into with \( D \) dimensional channel, float-point Q, K, and V are computed by linear matrices, followed by a spike neuron layer converting them into spike tensors \( QS, KS, \) and \( VS \). SDSA-V1 is defined as:

\[

\text{SDSA}(Q, K, V) = g(QS, KS) \otimes VS = \text{SN}(\text{SUMc}(QS \otimes KS)) \otimes VS

\]

where \( \otimes \) denotes the Hadamard product, \( g(·) \) computes the attention map, and \( \text{SUMc}(·) \) sums each column. The Hadamard product between spike tensors acts as a mask.\\

\textbf{Discussion on SDSA}:Above Equation can be rewritten, highlighting SDSA as a form of linear attention with computational complexity linear in token number \( N \) since \( KS \) and \( VS \) can be precomputed. This is possible because SDSA omits the softmax operation, using the spike neuron layer \( \text{SN}(·) \) as the kernel function. For the special case where \( H = D \), SDSA operates on \( H \) heads, concatenating their outputs.

\begin{figure}[h]

\centering

\includegraphics[width=1\textwidth]{sdsa.png}

\caption{SDSA and VSA}

\label{fig:SDSA}

\end{figure}

\subsubsection{Event-Driven Binary Spike Communication}

Computations in the spike-driven Transformer are only initiated when spikes, which are binary events (0 or 1), are present. This event-driven method significantly lowers energy usage and pointless computations. Further energy savings are achieved by the model by utilizing the sparsity of spike events to convert matrix multiplications into sparse additions.

\subsubsection{Self-Attention with Linear Complexity}

Spike-Driven Self-Attention (SDSA) is a major breakthrough in the spike-driven Transformer. SDSA uses column-wise summation and element-wise masking (Hadamard product), as opposed to classical self-attention, which requires expensive matrix multiplications. With respect to both the amount of tokens and channels, this method keeps the advantages of self-attention while lowering the computational complexity to linear.

\subsubsection{Operations Between Spike-Form Query, Key, and Value}

The spike-form Query, Key, and Value matrices' interactions are reduced to mask and addition operations in SDSA. This not only saves energy but also prevents the creation of big integers that need further scaling, which is a major problem with spiking Transformers that are now in use.

\subsubsection{Comparison with Traditional Transformers}

With up to 87.2× less computing energy required for self-attention activities, the spike-driven Transformer provides a significant energy savings over conventional Transformers. Because of this, it is ideal for use with neuromorphic devices, where energy efficiency is crucial.

\subsubsection{Future Directions}

Spike-driven paradigms' use into Transformer designs creates a number of intriguing study directions. Further developments might concentrate on improving the spike-driven Transformer training algorithms, investigating other neuromorphic hardware implementations, and broadening the application domains to encompass more intricate and varied activities.

\subsection{Spiking Neurons}

Spiking neurons, also known as spiking neural networks (SNNs), are a third generation of artificial neural networks that, compared to even more advanced deep learning models and classic ANNs, more closely resemble the functioning of real neurons. Spiking neurons are characterized by their ability to communicate through discrete spikes, also known as action potentials, which only happen when a membrane potential threshold is crossed. Unlike the continuous-valued activations of neurons in conventional ANNs, this process is event-driven.

\subsubsection{Biological Inspiration}

Spiking neurons are modeled after the information-processing mechanisms of real neurons. Short-lived, discrete impulses called spikes or action potentials are the means by which biological neurons exchange information. This kind of communication makes information processing effective and energy-efficient, making it especially suitable for activities involving real-time sensory processing.   
  
Spiking neurons target several important biological neuronal characteristics, such as:

\begin{itemize}

\item \textbf{Temporal Dynamics:} Spiking neurons model the temporal aspect of neural activity, which is essential for understanding the timing of spikes.

\item \textbf{Event-Driven Processing:} Unlike traditional neural networks that process information in discrete time steps, SNNs process information only when spikes occur, making them inherently event-driven.

\item \textbf{Synaptic Plasticity:} Learning in spiking neurons is often modeled using mechanisms like Spike-Timing Dependent Plasticity (STDP), which adjusts synaptic weights based on the relative timing of pre- and post-synaptic spikes.

\end{itemize}

\subsubsection{Mathematical Models of Spiking Neurons}

Several mathematical models have been developed to simulate the behavior of spiking neurons. The most commonly used models include:

\begin{enumerate}

\item \textbf{Leaky Integrate-and-Fire (LIF) Model}

The Leaky Integrate-and-Fire (LIF) model is one of the simplest and most widely used spiking neuron models. It captures the essential features of neuronal dynamics while being computationally efficient.

\textbf{Equations:} The LIF neuron is described by a differential equation that models the membrane potential $V(t)$:

\[

\tau\_m \frac{dV(t)}{dt} = -V(t) + R\_m I(t)

\]

where:

\begin{itemize}

\item $\tau\_m$ is the membrane time constant.

\item $V(t)$ is the membrane potential at time $t$.

\item $R\_m$ is the membrane resistance.

\item $I(t)$ is the input current at time $t$.

\end{itemize}

\textbf{Spiking Mechanism:}

\begin{itemize}

\item When $V(t)$ reaches a threshold $V\_{\text{th}}$, the neuron emits a spike, and the membrane potential is reset to a reset value $V\_{\text{reset}}$.

\item After emitting a spike, the neuron may enter a refractory period during which it cannot fire another spike.

\end{itemize}

\textbf{Key Features:}

\begin{itemize}

\item \textbf{Leakage:} The term $-V(t)$ represents the passive leakage of the membrane potential over time.

\item \textbf{Integration:} The neuron integrates incoming input currents, increasing the membrane potential.

\item \textbf{Firing:} The neuron fires when the membrane potential exceeds the threshold.

\end{itemize}

\item \textbf{Hodgkin-Huxley Model}

The Hodgkin-Huxley model is a more detailed and biophysically accurate model of neuronal dynamics. It describes the ionic mechanisms underlying the generation and propagation of action potentials in neurons.

\textbf{Equations:} The membrane potential $V(t)$ is governed by:

\[

C\_m \frac{dV(t)}{dt} = I(t) - I\_{\text{Na}}(t) - I\_{\text{K}}(t) - I\_{\text{L}}(t)

\]

where:

\begin{itemize}

\item $C\_m$ is the membrane capacitance.

\item $I(t)$ is the total input current.

\item $I\_{\text{Na}}(t)$, $I\_{\text{K}}(t)$, and $I\_{\text{L}}(t)$ are the sodium, potassium, and leakage currents, respectively.

\end{itemize}

The ionic currents are given by:

\[

I\_{\text{Na}}(t) = g\_{\text{Na}}(m^3 h)(V(t) - E\_{\text{Na}})

\]

\[

I\_{\text{K}}(t) = g\_{\text{K}}(n^4)(V(t) - E\_{\text{K}})

\]

\[

I\_{\text{L}}(t) = g\_{\text{L}}(V(t) - E\_{\text{L}})

\]

where:

\begin{itemize}

\item $g\_{\text{Na}}$, $g\_{\text{K}}$, and $g\_{\text{L}}$ are the maximum conductances for sodium, potassium, and leakage currents.

\item $E\_{\text{Na}}$, $E\_{\text{K}}$, and $E\_{\text{L}}$ are the reversal potentials for sodium, potassium, and leakage currents.

\item $m$, $h$, and $n$ are gating variables that follow their own differential equations.

\end{itemize}

\textbf{Gating Variables:} The gating variables $m$, $h$, and $n$ follow first-order kinetics:

\[

\frac{dm}{dt} = \alpha\_m(V)(1 - m) - \beta\_m(V)m

\]

\[

\frac{dh}{dt} = \alpha\_h(V)(1 - h) - \beta\_h(V)h

\]

\[

\frac{dn}{dt} = \alpha\_n(V)(1 - n) - \beta\_n(V)n

\]

where $\alpha$ and $\beta$ are voltage-dependent rate constants.

\textbf{Key Features:}

\begin{itemize}

\item \textbf{Ion Channels:} The model incorporates detailed dynamics of ion channels (sodium and potassium) that contribute to the action potential.

\item \textbf{Biophysical Realism:} It provides a close match to the actual behavior of biological neurons.

\end{itemize}

\item \textbf{Izhikevich Model}

The Izhikevich model is designed to be both biologically plausible and computationally efficient. It can reproduce a wide range of neuronal behaviors with minimal computational cost.

\textbf{Equations:} The Izhikevich model is defined by:

\[

\frac{dv(t)}{dt} = 0.04v(t)^2 + 5v(t) + 140 - u(t) + I(t)

\]

\[

\frac{du(t)}{dt} = a(bv(t) - u(t))

\]

where:

\begin{itemize}

\item $v(t)$ is the membrane potential.

\item $u(t)$ is a recovery variable.

\item $I(t)$ is the input current.

\item $a$, $b$, $c$, and $d$ are parameters that control the neuron's dynamics.

\end{itemize}

\textbf{Spiking Mechanism:}

\begin{itemize}

\item When $v(t)$ reaches a threshold (typically 30 mV), the neuron emits a spike, and the membrane potential and recovery variable are reset:

\[

v(t) \leftarrow c

\]

\[

u(t) \leftarrow u(t) + d

\]

\end{itemize}

\textbf{Key Features:}

\begin{itemize}

\item \textbf{Diverse Dynamics:} The model can produce various spiking patterns, such as regular spiking, bursting, and fast spiking, by adjusting the parameters.

\item \textbf{Efficiency:} It is computationally less intensive than the Hodgkin-Huxley model while still capturing essential neuronal behaviors.

\end{itemize}

\item \textbf{Spike Response Model (SRM)}

The Spike Response Model (SRM) is another simplified spiking neuron model that focuses on the neuron's response to incoming spikes.

\textbf{Equations:} The membrane potential $V(t)$ is given by:

\[

V(t) = \eta(t - t\_{\text{last}}) + \sum\_{t\_i < t} \epsilon(t - t\_i) + \sum\_j w\_j \epsilon\_j(t - t\_j)

\]

where:

\begin{itemize}

\item $\eta$ is the refractory response function.

\item $\epsilon$ is the postsynaptic potential function.

\item $t\_{\text{last}}$ is the time of the last spike.

\item $t\_i$ are the times of incoming spikes.

\item $w\_j$ are the synaptic weights.

\end{itemize}

\textbf{Key Features:}

\begin{itemize}

\item \textbf{Refractory Period:} The function $\eta$ models the refractory period after a spike.

\item \textbf{Postsynaptic Potentials:} The function $\epsilon$ models the effect of incoming spikes on the membrane potential.

\end{itemize}

\paragraph{Comparison of Models}

\begin{itemize}

\item \textbf{LIF Model:} Simple and computationally efficient but lacks biological detail.

\item \textbf{Hodgkin-Huxley Model:} Biophysically accurate but computationally intensive.

\item \textbf{Izhikevich Model:} Balances biological realism with computational efficiency, capable of diverse spiking behaviors.

\item \textbf{SRM Model:} Focuses on spike response, useful for understanding synaptic integration and refractory effects.

\end{itemize}

\end{enumerate}

\subsection{Spike Encoding in Spiking Neural Networks (SNNs)}

Spike encoding is a crucial process in Spiking Neural Networks (SNNs), converting continuous-valued inputs into discrete spike trains that can be processed by the network. This conversion is essential for SNNs to mimic the way biological neurons communicate through spikes.

\subsubsection{Types of Spike Encoding}

\begin{enumerate}

\item \textbf{Rate Coding}

\begin{itemize}

\item \textbf{Description}: The most straightforward method where the intensity of a stimulus is encoded by the firing rate of neurons. Higher stimulus intensity corresponds to a higher firing rate.

\item \textbf{Advantages}: Simple and widely used due to its biological plausibility.

\item \textbf{Disadvantages}: Loses temporal information and can be inefficient for representing dynamic inputs.

\item \textbf{Example}: In visual processing, the brightness of a pixel could be encoded by the rate at which a corresponding neuron fires.

\end{itemize}

\item \textbf{Temporal Coding}

\begin{itemize}

\item \textbf{Description}: The information is encoded in the precise timing of individual spikes. This can be more efficient and informative than rate coding.

\item \textbf{Advantages}: Can represent temporal patterns and dynamic changes in the input signal.

\item \textbf{Disadvantages}: More complex to implement and decode.

\item \textbf{Types of Temporal Coding}:

\begin{itemize}

\item \textbf{Time-to-First-Spike (Latency Coding)}: The timing of the first spike after stimulus onset encodes the information. Shorter latencies correspond to stronger stimuli.

\item \textbf{Phase Coding}: Information is encoded in the phase of spikes relative to a reference oscillatory cycle.

\end{itemize}

\end{itemize}

\item \textbf{Population Coding}

\begin{itemize}

\item \textbf{Description}: Involves a group of neurons encoding a piece of information collectively. Each neuron in the population may have a different preferred stimulus, and the combined activity pattern represents the input.

\item \textbf{Advantages}: Robust to noise and can represent complex stimuli.

\item \textbf{Disadvantages}: Requires a larger number of neurons and more complex decoding mechanisms.

\item \textbf{Example}: In motor control, a population of neurons might encode the direction of movement.

\end{itemize}

\item \textbf{Rank Order Coding}

\begin{itemize}

\item \textbf{Description}: The order in which neurons fire in response to a stimulus encodes the information. The sequence of firing is the crucial element.

\item \textbf{Advantages}: Utilizes the precise spike timing and is efficient in representing information.

\item \textbf{Disadvantages}: Decoding can be complex and requires accurate spike timing.

\item \textbf{Example}: For sensory processing, neurons might fire in a specific order corresponding to the features of the stimulus.

\end{itemize}

\item \textbf{Burst Coding}

\begin{itemize}

\item \textbf{Description}: Information is encoded in bursts of spikes rather than single spikes. The number of spikes in a burst or the inter-spike intervals within a burst carry the information.

\item \textbf{Advantages}: Can be more reliable and less affected by noise compared to single spike events.

\item \textbf{Disadvantages}: Requires more complex decoding and can be energetically costly.

\item \textbf{Example}: In auditory processing, bursts of spikes might encode the onset of a sound.

\end{itemize}

\end{enumerate}

\subsection{SpikeBERT}

\subsubsection{Introduction}

\paragraph{Spiking Neural Networks and Their Importance\\}

Inspired by the brain's natural information-processing capabilities, spiking neural networks (SNNs) represent a paradigm change in the field of neural network models. SNNs, which Maas first proposed in 1997, are frequently referred to as the third generation of neural network models. In contrast to conventional artificial neural networks (ANNs), which compute and transmit information using continuous floating-point values, SNNs use discrete spike trains, which are made up of binary values (0s and 1s). Because of this feature, they are especially well-suited for implementation on neuromorphic hardware, such the 28nm TrueNorth chips from IBM and the 14nm Loihi 2 chips from Intel.

Spike trains enable SNNs to emulate the biological neural networks' event-driven architecture, offering a more effective and perhaps more potent way to handle massive amounts of data. SNNs in particular have a lot to offer in terms of energy efficiency. SNNs can accomplish comparable or even better performance with substantially lower energy requirements than classic ANNs running on GPUs, especially during inference workloads. SNNs are a potential option for implementing complex models in contexts with limited resources due to their efficiency.

\paragraph{Challenges in Natural Language Processing\\}

SNNs have not been used extensively for natural language processing (NLP) tasks, despite their potential. The absence of neuromorphic language datasets is a significant barrier. ANNs are meant to tackle traditional NLP datasets because they can handle continuous values. It is still very difficult to convert language data into a format that SNNs can use without losing important information. Since SNNs are frequently evaluated using datasets tuned for artificial neural networks (ANNs), this constraint makes it challenging to objectively assess their performance in language tasks.

When comparing SNNs and ANNs' performance on tasks meant for continuous data processing, it becomes clear how different they are from one another in NLP. SNNs have demonstrated performance comparable to vision transformers (ViTs) in image classification on event-based datasets such as DVS-128 Gesture and CIFAR-10-DVS. But comparable progress in NLP has proven elusive, mostly because of the previously noted data format problems. The advancement of SNNs in NLP depends on the creation of novel techniques for turning language data into spike trains.

\paragraph{GPU Memory Constraints\\}

Another significant challenge in deploying SNNs is the increased demand for GPU memory. SNNs introduce an additional dimension, denoted as \(T\) (timestep), which is not present in traditional ANNs. Increasing the number of timesteps allows SNNs to capture more information, enhancing their performance. However, this also exponentially increases the memory requirements during training, posing a substantial constraint. In our experiments, we found that maintaining a consistent number of timesteps necessitated reducing the sentence length of input data, which adversely affected the models' performance. Overcoming these memory limitations is essential for the practical application of SNNs in NLP.

\paragraph{Motivations for Developing SpikeBERT\\}

Large language models (LLMs), like BERT and ChatGPT, are dominating the NLP field nowadays because of their remarkable performance on a variety of tasks. Nevertheless, a high energy consumption of these models is a downside, especially when doing inference on GPUs. These models' significant energy consumption makes them difficult to implement in settings with limited resources.   
We developed SpikeBERT because we needed to make more energy-efficient versions of these traditional LLMs. By utilizing SNNs' energy efficiency and keeping BERT-like models' sophisticated capabilities, SpikeBERT is an inventive technique. Our goal is to achieve GPU-free inference by creating a spiking version of BERT, which will result in a large reduction in energy consumption.

\paragraph{Knowledge Distillation and SpikeBERT\\}

The fundamental distinctions between SNNs and ANNs present one of the main obstacles in the development of SpikeBERT. SpikeBERT must use discrete spikes in order to function, whereas BERT uses continuous data. We present a novel two-stage "pre-training + task-specific" knowledge distillation (KD) strategy to close this gap. In order to guarantee that the student model (SpikeBERT) can efficiently learn from the teacher model (BERT), this method is intended to condense knowledge from BERT into SpikeBERT.

Our KD method includes appropriate normalization throughout training instances and timesteps in a batch. For a meaningful comparison and alignment of feature representations between the instructor and student models, this normalization is essential. By this technique, we hope to enable SpikeBERT to execute difficult language tasks at high performance and low energy consumption, opening the door for further advancements in neuromorphic hardware-based energy-efficient LLMs.

To sum up, SpikeBERT is an important step in fusing sophisticated NLP models with the ideas of neuromorphic computing. We want to set a new benchmark for energy-efficient language processing by tackling the problems of data conversion, GPU memory limitations, and efficient knowledge distillation.

\subsubsection{Method}

In this part, we present our two-stage distillation method for training SpikeBERT and explain how we enhanced the Spikformer architecture. First, we describe how surrogate gradients in spiking neural networks and spiking neurons work. We then go over the changes made to Spikformer so that it can handle text data. Our "pre-training + task-specific" distillation technique is finally explained in detail.

\begin{enumerate}

\item \textbf{Spiking Neurons and Surrogate Gradients}

One popular spiking neuron is the Leaky Integrate-and-Fire (LIF) neuron \cite{wu2017spiking}. LIF neurons depend on a weighted sum of inputs, just like conventional activation functions like ReLU, and contribute to the membrane potential $U\_t$ of the neuron at time step $t$. A spike $S\_t$ is produced if the membrane potential crosses a threshold $U\_{thr}$:

\begin{equation}

S\_t = \begin{cases}

1 & \text{if } U\_t \ge U\_{thr} \\

0 & \text{if } U\_t < U\_{thr}

\end{cases}

\end{equation}

A resistor-capacitor circuit can be used to simulate the dynamics of a neuron's membrane potential \cite{maas1997networks}. The differential equation can be roughly solved by:

\begin{equation}

U\_t = I\_t + \beta U\_{t-1} - S\_{t-1} U\_{thr}, \quad I\_t = W X\_t

\end{equation}

where $X\_t$ are the inputs at time step $t$, $W$ is a set of learnable weights, $I\_t$ is the weighted sum of inputs, $\beta$ is the decay rate of the membrane potential, and $U\_{t-1}$ is the membrane potential at time $t-1$. The term $S\_{t-1} U\_{thr}$ models the spiking and membrane potential reset mechanism.

We follow \cite{fang2020surrogate} and use the arctangent-like surrogate gradient function, which regards the Heaviside step function as:

\begin{equation}

S \approx \frac{1}{\pi} \arctan\left(\frac{\pi}{2\alpha} U\right) + \frac{1}{2}

\end{equation}

Thus, the gradients of $S$ are:

\begin{equation}

\frac{\partial S}{\partial U} = \frac{\alpha}{2 (1 + (\frac{\pi}{2} \alpha U)^2)}

\end{equation}

where $\alpha$ defaults to 2.

\item \textbf{The Architecture of SpikeBERT}

\begin{figure}[h]

\centering

\includegraphics[width=1\textwidth]{spikbert.png}

\captionof{figure}{(a) Architecture of Spikformer with L encoder blocks. Spikformer is specially designed for image classification tasks, where the spiking patch splitting (SPS) module and the “convolution layer + batch normalization” module can process vision signals well. The spiking self-attention (SSA) module in Spikformer aims to model the attention between every two dimensions, so we denote it as “D-SSA”. (b) Architecture of SpikeBERT with L' encoder blocks. To improve the model’s ability to process texts, we adopt a “linear layer + layer normalization”, and also replace the SPS module with a word embedding layer. Furthermore, we modify the SSA module to enhance SpikeBERT’s ability to concentrate on the interrelation between all pairs of words (or tokens), instead of dimensions.}

\label{fig:Spikbert}

\end{figure}

One of the first sizable spiking neural networks created specifically for language applications is called SpikeBERT. Our architecture is based on the Transformer-based spiking neural network Spikformer \cite{zhou2023spikformer}, which is hardware-friendly and shown in Figure 1(a). The spiking self-attention (SSA) module, at the heart of Spikformer, uses discrete spikes to imitate the vanilla self-attention mechanism:

\begin{equation}

\text{SSA}(Q\_s, K\_s, V\_s) = \text{SN}(\text{BN}(\text{Linear}(Q\_s K\_s^T V\_s \cdot \tau)))

\end{equation}

where $Q\_s = \text{SN}\_Q(\text{BN}(X\_s W\_Q))$, $K\_s = \text{SN}\_K(\text{BN}(X\_s W\_K))$, $V\_s = \text{SN}\_V(\text{BN}(X\_s W\_V))$, and $X\_s \in \mathbb{R}^{T \times L \times D}$ is the input to SSA. $T$ is the number of time steps, $\text{BN}$ is batch normalization, and $\tau$ is a scaling factor. Outputs of SSA and $Q\_s, K\_s, V\_s$ are spike matrices containing only 0s and 1s. $W\_Q, W\_K, W\_V$, and Linear are learnable decimal parameters. The shape of the attention map in Spikformer, $Q\_s K\_s^T$, is $D \times D$, where $D$ is the dimensionality of the hidden layers.

We adapt Spikformer so that it can handle textual data efficiently. In order to translate tokens to tensors, we first replace the spiking patch splitting (SPS) module with a word embedding layer and a spiking neuron layer. Significantly, we rearrange the SSA module's attention map from $D \times D$ to $N \times N$, where $N$ is the length of the input sentences. Lastly, since convolution layers collect pixel features in images, which makes them inappropriate for language tasks, we substitute "convolution layers + batch normalization" with "linear layers + layer normalization." Figure 1(b) displays the SpikeBERT architecture.

\item \textbf{Two-Stage Distillation}

The model failed to converge when next sentence prediction (NSP) techniques like BERT and masked language modeling (MLM) were applied directly. This was because of "self-accumulating dynamics" \cite{fang2020deep}, an unresolved problem in large-scale spiking neural networks. In order to prevent the quick accumulation of surrogate gradient variation, we employ knowledge distillation during the SpikeBERT training process. The different data formats of hidden features—floating-point in ANNs and time-varying spike trains in SNNs—make it difficult to extract knowledge from ANNs to SNNs. This issue is resolved by adding external modules for feature alignment during training.

\begin{figure}[h]

\centering

\includegraphics[width=1\textwidth]{kdspikbert.png}

\caption{Overview of our two-stage distillation method (pre-training + task-specific distillation) for training SpikeBERT. $T$ is the number of time steps of features in every layer. Notice that the logits loss and cross-entropy loss are only considered in stage 2. The varying shades of color represent the magnitude of the floating-point values. The dotted line under $L\_i^{\text{fea}}$ indicates that features of some hidden layers can be ignored when calculating feature alignment loss. If the student model contains different numbers of layers from the teacher model, we will align features every few layers.}

\label{fig:SpikeBERT\_Distillation}

\end{figure}

Our suggestion is a two-phase distillation procedure that adheres to the "pre-training + fine-tuning" formula. Using a large-scale corpus, the first stage aligns hidden features and embeddings between BERT and SpikeBERT. In the second stage, the model from stage 1 is refined using logits and cross-entropy data from a fine-tuned BERT. Figure 2 gives a summary of our approach.

\begin{enumerate}

\item \textbf{The First Stage: Pre-Training Distillation}

Our objective is to align the embeddings and hidden features of the teacher model $TM$ and student model $SM$, SpikeBERT, and a pre-trained BERT \cite{devlin2019bert}, using a set of unlabelled texts. We present two new alignment loss functions: feature and embedding.

\textbf{Feature Alignment Loss:} The similarity of features between $TM$ and $SM$ at each hidden layer is measured by this loss $L\_{fea}$. The shape of the feature $F\_{sm}$ in the student model at each layer is $T \times N \times D$, whereas the feature $F\_{tm}$ in the BERT model is $N \times D$. While $F\_{tm}$ is a decimal matrix, $F\_{sm}$ is a binary matrix. We convert the characteristics of $TM$ and $SM$ to the same content space in order to address these differences:

\begin{equation}

F\_{tm}' = F\_{tm}, \quad F\_{sm}' = \text{LayerNorm}(\text{MLP}(F\_{sm}))

\end{equation}

If the student model has fewer layers than BERT, we align features every few layers. The feature alignment loss for layer $i$ in the student model is:

\begin{equation}

L\_{fea}^i = ||F\_{tm}' - F\_{sm}'||^2

\end{equation}

\textbf{Embedding Alignment Loss:} The embeddings of input sentences are not spikes until processed by the Heaviside step function. Define $E\_{tm}$ and $E\_{sm}$ as the embeddings of the teacher and student models, respectively. The embedding alignment loss is:

\begin{equation}

L\_{emb}^i = ||E\_{tm} - \text{MLP}(E\_{sm})||^2

\end{equation}

The total loss $L\_1$ in stage 1 is the sum of the feature and embedding alignment losses:

\begin{equation}

L\_1 = \sigma\_1 \sum\_i L\_{fea}^i + \sigma\_2 L\_{emb}

\end{equation}

where $\sigma\_1$ and $\sigma\_2$ are hyperparameters balancing the learning of embeddings and features.

\item \textbf{Second Stage: Task-Specific Distillation}

In step 2, the teacher model is a BERT that has been fine-tuned on a task-specific dataset, and the student model is the stage 1 model. For language tasks, a task-specific head is superimposed on top of the core language model, such an MLP layer for text categorization. The effectiveness of knowledge distillation is increased through data augmentation.

\textbf{Data Augmentation:} We add to the training set by randomly adding [MASK] tokens to words, changing words to other words with the same POS tag, and selecting n-grams at random from training instances.

\textbf{Logits Loss:} We quantify the difference between the teacher and student models' prediction distributions using KL-divergence, as per \cite{hinton2015distilling}:

\begin{equation}

L\_{logits} = \sum\_{i} p\_i \log \frac{p\_i}{q\_i}

\end{equation}

where $p\_i$ and $q\_i$ denote the prediction distributions of the teacher and student models.

\textbf{Cross-Entropy Loss:} This loss helps the student model learn from task-specific dataset samples:

\begin{equation}

L\_{ce} = - \sum\_{i} \hat{q\_i} \log(q\_i)

\end{equation}

The total loss $L\_2$ in stage 2 is a weighted sum of the losses:

\begin{equation}

L\_2 = \lambda\_1 \sum\_i L\_{fea}^i + \lambda\_2 L\_{emb} + \lambda\_3 L\_{logits} + \lambda\_4 L\_{ce}

\end{equation}

where $\lambda\_1, \lambda\_2, \lambda\_3$, and $\lambda\_4$ are hyperparameters controlling the weight of each loss.

For both stages, we adopt backpropagation through time (BPTT), suitable for training spiking.

\end{enumerate}

\end{enumerate}

\bibliographystyle{unsrt}

\bibliography{references}

\end{document}