SpikingBrain Technical Report: Spiking Brain-inspired Large Models

Yuqi Pan[1,2,3], **Yupeng Feng**[1], **Jinghao Zhuang**[1], **Siyu Ding**[1], **Zehao Liu**[1,4],
Bohan Sun[1], **Yuhong Chou**[1,4], **Han Xu**[1,5], **Xuerui Qiu**[1,6], **Anlin Deng**[1], **Anjie Hu**[1,7],
Peng Zhou[8], **Man Yao**[1,2,3], **Jibin Wu**[4], **Jian Yang**[9], **Guoliang Sun**[9], **Bo Xu**[1,2,*], **Guoqi Li**[1,2,3*]

[1]Institute of Automation, Chinese Academy of Sciences
[2]Beijing Key Laboratory of Brain-Inspired General Intelligence Large Model
[3]Key Laboratory of Brain Cognition and Brain-inspired Intelligence Technology
[4]The Hong Kong Polytechnic University [5]Beijing Academy of Artificial Intelligence
[6]Zhongguancun Academy [7]Beihang University [8]LuxiTech [9]MetaX Integrated Circuit Co., Ltd.

Abstract

Mainstream Transformer-based large language models (LLMs) face significant efficiency bottlenecks: training computation scales quadratically with sequence length, and inference memory grows linearly. These constraints limit their ability to process long sequences effectively. In addition, building large models on non-NVIDIA computing platforms poses major challenges in achieving stable and efficient training and deployment. To address these issues, we introduce SpikingBrain, a new family of brain-inspired models designed for efficient long-context training and inference. SpikingBrain leverages the MetaX[1] GPU cluster and focuses on three core aspects: i) **Model Architecture:** linear and hybrid-linear attention architectures with adaptive spiking neurons; ii) **Algorithmic Optimizations:** an efficient, conversion-based training pipeline compatible with existing LLMs, along with a dedicated spike coding framework; iii) **System Engineering:** customized training frameworks, operator libraries, and parallelism strategies tailored to the MetaX hardware.

Using these techniques, we develop two models: **SpikingBrain-7B**, a linear LLM, and **SpikingBrain-76B**, a hybrid-linear MoE LLM. These models demonstrate the feasibility of large-scale LLM development on non-NVIDIA platforms. SpikingBrain achieves performance comparable to open-source Transformer baselines while using exceptionally low data resources (continual pre-training of $\sim$150B tokens). Our models also significantly improve long-sequence training efficiency and deliver inference with (partially) constant memory and event-driven spiking behavior. For example, SpikingBrain-7B achieves more than 100× speedup in Time to First Token (TTFT) for 4M-token sequences. Our training framework supports weeks of stable large-scale training on hundreds of MetaX C550 GPUs, with the 7B model reaching a Model FLOPs Utilization (MFU) of 23.4%. In addition, the proposed spiking scheme achieves 69.15% sparsity, enabling low-power operation. Overall, this work demonstrates the potential of brain-inspired mechanisms to drive the next generation of efficient and scalable large model design.[2]

[*] Corresponding authors: xubo@ia.ac.cn and guoqi.li@ia.ac.cn

[1] https://www.metax-tech.com/en

[2] The code for this project is publicly available at `https://github.com/BICLab/SpikingBrain-7B`.

1 Introduction

Recent advances in large language models (LLMs) built on the Transformer architecture [1] have been driven by the scaling law [2], which suggests that performance improves with larger model sizes and more data [3, 4, 5, 6]. However, this scale-driven approach comes with significant challenges: extremely high training costs, high energy consumption, and complex deployment pipelines. Therefore, achieving high performance and energy efficiency under limited resources has become a critical research goal. To address this, our work draws inspiration from brain mechanisms. We explore novel architectures, training paradigms, and spike coding schemes to develop efficient, brain-inspired LLMs that move beyond the traditional Transformer framework.

A key focus of this study is to validate the training and deployment of such models on non-NVIDIA computing clusters. We use an open-source Transformer checkpoint (Qwen2.5-7B-base [7] as an example) together with our efficient development framework to train and evaluate two models on the MetaX GPU cluster. The models, SpikingBrain-7B and SpikingBrain-76B-A12B, undergo end-to-end validation, including continual pre-training (CPT), long-context extension (up to 128k tokens), and supervised fine-tuning (SFT). We also adapt the vLLM inference framework to demonstrate deployment feasibility on MetaX hardware.

Our main technical contributions are as follows:

- **Hybrid Linear Architectures:** Moving away from quadratic self-attention, we design hybrid models combining linear, local, and standard attention modules. We explore two hybrid strategies: inter-layer sequential (SpikingBrain-7B) and intra-layer parallel (SpikingBrain-76B-A12B). The former achieves linear complexity and excels at long-context efficiency; the latter provides stronger language modeling capability through a more sophisticated architectural design. Notably, the modeling of linear attention [8] also resonates with neuronal dynamics in biological systems [9].
- **Efficient Model Conversion:** i) **Attention modules:** Using a unified attention map analysis, we convert quadratic attention modules into sparse sliding-window and low-rank linear attention, by remapping the weights of existing Transformer models [10]. This reduces training and inference costs, enabling efficient long-context handling with less than 2% of the compute needed for training from scratch. ii) **FFN modules:** For SpikingBrain-76B, we utilize an MoE upcycling technique [11] that replicates dense FFN weights to create sparse experts, increasing capacity with minimal compute and memory overhead.
- **Adaptive Threshold Spiking:** Inspired by event-driven biological neurons, we propose a spiking scheme that converts activations into integer spike counts and expands them into sparse spike trains. This enables addition-based, event-driven computation. Several encoding formats are supported, including binary {0,1}, ternary {-1,0,1}, and bitwise spike coding. These sparse, event-driven representations provide the basis for our low-power inference design and may also inspire the development of next-generation neuromorphic hardware [12, 13, 14].
- **Large-Scale Training and Inference on MetaX:** Both models are trained on hundreds of MetaX C550 GPUs, covering the entire pipeline, from data preprocessing to distributed training and inference. We adapt frameworks, operators, and communication primitives to ensure stability. This work represents, to our knowledge, the first large-scale training of brain-inspired LLMs on a non-NVIDIA platform, achieving stable training at 76B parameters.

Our design philosophy holds that linear attention modules exhibit modeling characteristics highly analogous to human memory mechanisms, particularly through their use of compressed, continuously updated memory states [15]. From a biological perspective, linear attention can also be interpreted as a simplified form of dendritic dynamics. In addition, the Mixture-of-Experts (MoE) component reflects a principle of modular specialization, akin to the distributed and specialized processing observed in biological neural networks [16]. By combining network-level sparsity (via MoE) with neuron-level spiking sparsity, our approach offers a robust multi-scale efficiency strategy. Taken together, these results point toward a promising direction for developing more efficient and biologically plausible large-model architectures that extend beyond standard Transformer-based LLMs through brain-inspired mechanisms.

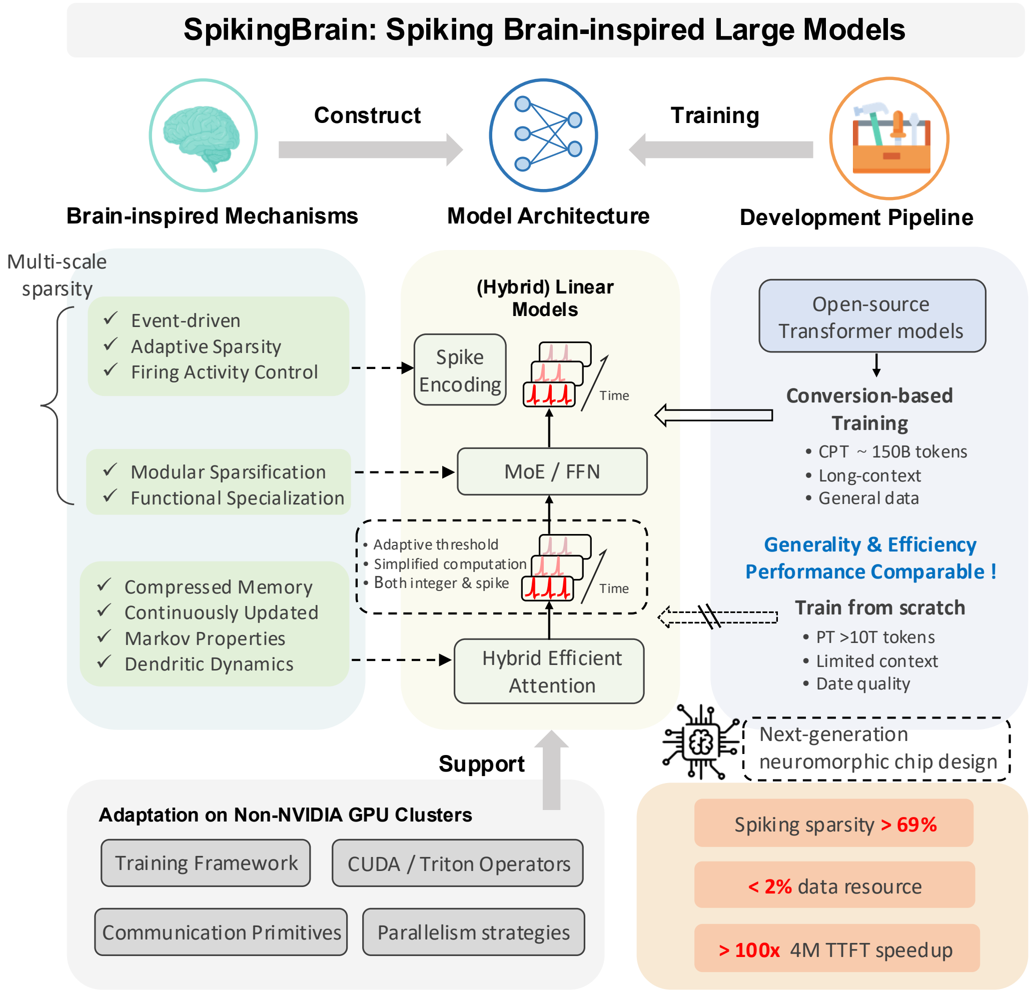


Figure 1: **Overview of SpikingBrain.** Inspired by brain mechanisms, SpikingBrain integrates hybrid efficient attention, MoE modules, and spike encoding into its architecture, supported by a universal conversion pipeline compatible with the open-source model ecosystem. This enables continual pre-training with less than 2% of the data while achieving performance comparable to mainstream open-source models. We further adapt frameworks, operators, parallel strategies, and communication primitives for non-NVIDIA (MetaX) clusters, ensuring stable large-scale training and inference. SpikingBrain achieves over 100× speedup in TTFT for 4M-token sequences, while spiking delivers over 69% sparsity at the micro level. Combined with macro-level MoE sparsity, these advances provide valuable guidance for the design of next-generation neuromorphic chips.

In summary, this work presents efficient brain-inspired LLM training on MetaX GPUs, producing two models: SpikingBrain-7B and SpikingBrain-76B MoE (see Figure 1 for an overview). They achieve (near-)linear complexity, high training stability, and competitive performance with only $\sim 2\%$ of the pre-training data typical for mainstream LLMs. Inference shows substantial speedups, with SpikingBrain-7B reaching over 100× speedup in TTFT (Time to First Token) for 4M-token inputs.We further deploy a compressed 1B SpikingBrain model on a CPU-based mobile inference framework, achieving a 15.39× speedup at a sequence length of 256k. The proposed spiking scheme also yields $\sim 69\%$ sparsity, offering strong support for reducing power consumption. These findings highlight the potential of brain-inspired design for future neuromorphic hardware and efficient model scaling on diverse GPU platforms.

2 Model Architecture

2.1 Core Components

2.1.1 Attention Mechanisms

The core temporal interaction module of an LLM is the attention mechanism, which projects input tokens into QKV vectors. At each time step, the current query $\mathbf{q}_t$ interacts with past keys and values $\mathbf{k}_s, \mathbf{v}_s (s \leq t)$ to produce the output $\mathbf{o}_t$.

Softmax Attention Standard softmax attention [1] captures global, token-wise interactions across the entire sequence:

$$\mathbf{O} = \mathrm{softmax}(\mathbf{Q}\mathbf{K}^\top \odot \mathbf{M})\mathbf{V}; \quad \mathbf{o}_t = \frac{\sum_{s=1}^{t} \exp(\mathbf{q}_t \mathbf{k}_s^\top)\mathbf{v}_s}{\sum_{s=1}^{t} \exp(\mathbf{q}_t \mathbf{k}_s^\top)}. \tag{1}$$

Here, $\mathbf{M}$ is a causal mask where $\mathbf{M}_{ij} = 1$ if $i \geq j$ and $\mathbf{M}_{ij} = -\infty$ if $i < j$. For a sequence of length n, softmax attention provides high arithmetic intensity and parallelism during training but incurs $O(n^2)$ computation. During inference, maintaining the key-value cache adds an $O(n)$ memory footprint, becoming a bottleneck for long-context tasks.

Sliding Window Attention (SWA) To reduce this quadratic computational cost, many linear-complexity variants have been proposed. One example is SWA [17, 18, 19], which limits the attention scope to a fixed local window of size w, capturing fine-grained local interactions:

$$\mathbf{O} = \mathrm{softmax}(\mathbf{Q}\mathbf{K}^\top \odot \mathbf{M}')\mathbf{V}; \quad \mathbf{o}_t = \frac{\sum_{s=t-w+1}^{t} \exp(\mathbf{q}_t \mathbf{k}_s^\top)\mathbf{v}_s}{\sum_{s=t-w+1}^{t} \exp(\mathbf{q}_t \mathbf{k}_s^\top)}. \tag{2}$$

Here, $\mathbf{M}'$ is a windowed causal mask where $\mathbf{M}'_{ij} = 1$ if $i - w + 1 \leq j \leq i$, otherwise $\mathbf{M}'_{ij} = -\infty$. Thus, training complexity and inference memory reduce to $O(n)$ and $O(1)$ respectively, as w is a fixed constant independent of n.

Linear Attention Another common variant is linear attention [8, 20, 21], which removes the softmax function to achieve linear complexity and can be reformulated as a state-based linear recurrence:

$$\mathbf{O} = (\mathbf{Q}\mathbf{K}^\top \odot \mathbf{M})\mathbf{V}; \tag{3}$$

$$\mathbf{o}_t = \sum_{s=1}^{t} (\mathbf{q}_t \mathbf{k}_s^\top)\mathbf{v}_s = \mathbf{q}_t \sum_{s=1}^{t} (\mathbf{k}_s^\top \mathbf{v}_s) = \mathbf{q}_t \mathbf{S}_t, \text{ where } \mathbf{S}_t = \mathbf{S}_{t-1} + \mathbf{k}_t^\top \mathbf{v}_t. \tag{4}$$

Here, $\mathbf{M}$ is a causal mask where $\mathbf{M}_{ij} = 1$ if $i \geq j$ and $\mathbf{M}_{ij} = 0$ if $i < j$. Linear attention supports chunk-wise parallelism during training and maintains a fixed-size state in recurrent form [22, 20], reducing computational complexity and inference memory to $O(n)$ and $O(1)$.

Hybrid Attention Each attention mechanism has distinct strengths: softmax attention excels at global retrieval, SWA is efficient for local context, and linear attention effectively compresses long-range information. Hybrid Attention aims to combine these methods to balance efficiency and accuracy. Two common paradigms are: i) **Inter-layer sequential hybridization** [23, 24, 25], where different attention types are stacked across layers:

$$\mathbf{o}_t^1 = \mathrm{attention1}(\mathbf{x}_t), \quad \mathbf{o}_t = \mathrm{attention2}(\mathbf{o}_t^1). \tag{5}$$

ii) **Intra-layer parallel hybridization** [26, 27], where different attention modules process the same input in parallel and outputs are merged:

$$\mathbf{o}_t^1 = \mathrm{attention1}(\mathbf{x}_t), \quad \mathbf{o}_t^2 = \mathrm{attention2}(\mathbf{x}_t), \quad \mathbf{o}_t = w_1 \cdot \mathbf{o}_t^1 + w_2 \cdot \mathbf{o}_t^2. \tag{6}$$

Both approaches are widely adopted, enabling flexible trade-offs between modeling accuracy and efficiency by adjusting the proportions of each attention type. Our SpikingBrain-7B employs inter-layer hybridization of linear attention and SWA, whereas SpikingBrain-76B uses intra-layer hybridization that combines linear attention, SWA, and full softmax attention.

2.1.2 Mixture-of-Experts (MoE)

The core idea of sparse MoE [28, 29] is to enhance model capacity by introducing N parallel expert networks into the original feed-forward network (FFN) layer, while dynamically selecting the most relevant k experts for each input token $\mathbf{x}$ through a router $\mathbf{W}_r$:

$$\mathbf{p} = \sigma(\mathbf{W}_r \mathbf{x}), \quad \mathcal{I} = \{i \mid p_i \in \text{top-}\,k(\mathbf{p})\}, \tag{7}$$

$$\text{MoE\_activation}(\mathbf{x}) = \sum_{i \in \mathcal{I}} p_i \times E_i(\mathbf{x}). \tag{8}$$

Here, each expert $E_i(\cdot)$ is an FFN. The router produces a probability vector $\mathbf{p}$, where σ is typically a softmax or sigmoid function. For each token, only the top-k experts with the highest probabilities are activated (with $\mathcal{I}$ denoting the set of activated experts). The final MoE output is then computed as the weighted sum of these selected experts.

Unlike traditional dense layers, MoE does not activate all experts simultaneously. Instead, it leverages sparse activation to significantly increase parameter capacity and expressiveness while keeping computational cost nearly unchanged. This property enables MoE to combine efficiency with high performance in both training and inference. Prior studies [30, 31, 6] further suggest that introducing shared experts that are always active, and retaining a few dense FFN layers in shallow stages of the model can improve training stability and overall performance.

In practice, dense models can be efficiently expanded into MoE models using the **upcycling** technique [32, 11], which scales parameter capacity without sacrificing the original performance. The procedure involves: i) replicating the dense FFN weights across all experts so that the upcycled model initially matches the dense baseline; ii) rescaling expert outputs appropriately to maintain consistency with the output scale of the original model.

2.1.3 Spiking Neuron Modeling

Spiking neurons are the core components of Spiking Neural Networks (SNNs) [33], and their modeling generally falls into two paradigms: simplified approximations and detailed simulations. The **Hodgkin–Huxley (HH)** model [34] represents the latter, accurately describing the membrane potential dynamics of biological neurons through a set of nonlinear differential equations. It focuses on simulating the synergistic effects of transmembrane ion fluxes. Although such models can faithfully replicate fine-scale bioelectrical phenomena of neurons (e.g., complex firing patterns and refractory periods), they entail high computational complexity due to the involvement of multiple high-order differential operations. Thus, HH-based models are currently mainly suitable for physiological studies on small neuron populations and cannot meet the efficiency and latency requirements of LLMs.

By contrast, simplified models such as the **Leaky Integrate-and-Fire (LIF)** model, are more commonly used in practice [35, 36]. The LIF neuron is a first-order approximation of soma dynamics incorporating the temporal dimension. For an input token $\mathbf{x}$ (extended over time, where $\mathbf{x}_t$ is the input at the t-th time step), the membrane potential $\mathbf{v}_t$ and spike output $\mathbf{s}_t$ of the neuron can be formulated as below:

$$\mathbf{v}_{t+1} = \begin{cases} \lambda(1-\mathbf{s}_t)\mathbf{v}_t + \mathbf{v}_{\text{reset}} \cdot \mathbf{s}_t + \mathbf{x}_{t+1}, & \text{hard reset} \\ \lambda \mathbf{v}_t - V_{\text{th}} \cdot \mathbf{s}_t + \mathbf{x}_{t+1}, & \text{soft reset} \end{cases}; \quad \mathbf{s}_t = 1 \text{ if } \mathbf{v}_t \geq V_{\text{th}} \text{ else } 0. \tag{9}$$

Here, the membrane potential accumulates charge in the soma, λ represents a decay factor (mimicking the natural leakage of ion potential), where V_{th} represents a fixed firing threshold, and $\mathbf{v}_{\text{reset}}$ is the reset potential, which is usually set to 0. While this model simplifies the ion-based mechanisms and retains the core logic of natural neurons, it still exhibits several limitations for large-scale models [37, 38, 39]: i) The temporal dynamics (mainly introduced by the decay factor and reset mechanism) lead to increased training complexity and instability; ii) Even when integrated into pre-trained models, redundant complexity still persists, making it complicated to build models and simulate the original values; iii) The fixed threshold can cause suboptimal spike generation, leading to many neurons becoming silent or over-activated, posing challenges for optimization and impeding the balance between accuracy and energy efficiency.

To address these limitations, we propose **Adaptive-threshold Spiking Neurons**, which simplify LIF neurons for enhanced computational efficiency and modeling accuracy while preserving biological plausibility:

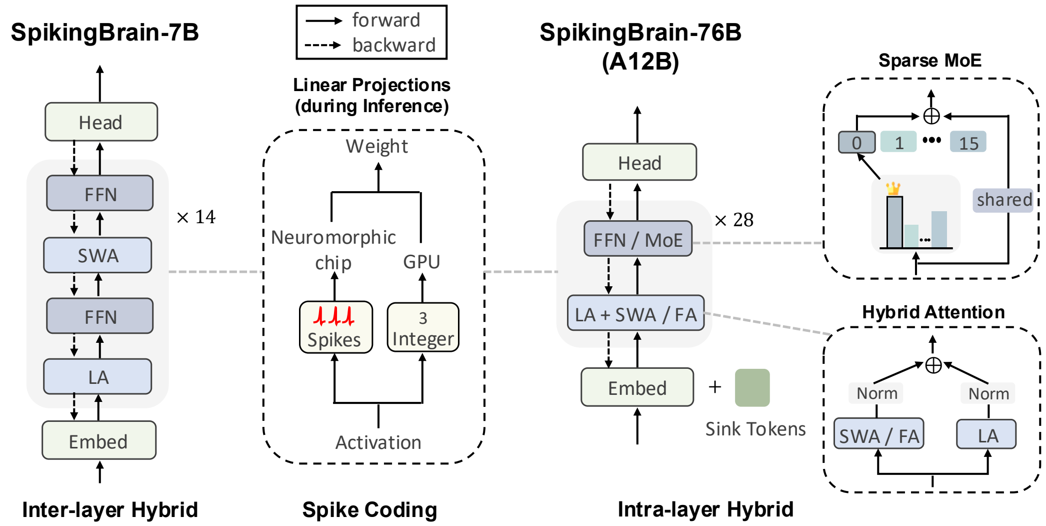


Figure 2: **Integrated architectures of SpikingBrain models.** FA: Full Softmax Attention; SWA: Sliding Window Attention; LA: Linear Attention. *(Left)* SpikingBrain-7B is a linear model with inter-layer hybridization. *(Middle)* Spike coding converts activations into integer counts for GPU execution or into spike trains for event-driven neuromorphic hardware. *(Right)* SpikingBrain-76B is a hybrid-linear MoE model with intra-layer hybridization, configured with 128 sink tokens, 16 routed experts, and 1 shared expert. Seven dense FFNs are located at layers $[1, 2, 3, 5, 7, 9, 11]$, with all other FFNs implemented as MoE layers. Attention modules are arranged as "LA + FA" at layers $[7, 14, 21, 28]$, and "LA + SWA" at all other layers.

- **Adaptive Threshold**: Inspired by adaptive dynamic properties of biological neurons, the firing threshold V_{th} is designed as a dynamic value correlated with the membrane potential. This prevents neurons from being over-excited or over-quiescent, maintaining a moderately active state from a statistical perspective.
- **Simplified Temporal Computation**: The decay factor is eliminated, and the soft-reset mechanism is adopted, enabling the conversion from continuous values to integer spike counts in a single step. During optimization, the temporal dimension is merged for stability and computational efficiency on GPUs. For inference, the temporal dimension can be re-expanded, and energy-efficient computation can be achieved using sparse event-driven asynchronous hardware [40].

Our modeling preserves essential characteristics of biological neurons and is appropriately simplified to eliminate redundant computations. It not only leverages the energy efficiency advantages of biological neural systems but also holds promise for effective engineering implementation in pre-trained LLMs when combined with specific asynchronous hardware.

2.2 Integrated Model Architectures

Through lightweight training, a base Transformer can be converted into efficient attention variants of different forms (see Section 3). This enables flexible trade-offs between performance and efficiency by adjusting factors such as the proportion of full-attention layers or the window size of local attention. As a case study, we develop two models from the Qwen2.5-7B-base checkpoint: **SpikingBrain-7B**, a pure linear model optimized for long-context efficiency, and **SpikingBrain-76B**, a hybrid-linear MoE model designed to balance efficiency and performance. Their architectures are illustrated in Figure 2. Both models are integrated with HuggingFace and vLLM inference frameworks, supporting deployment on single- and multi-GPU settings.

SpikingBrain-7B The 7B model achieves purely linear complexity by interleaving linear attention and sliding window attention (SWA) layers with a fixed 4K window in a 1:1 ratio. The FFN modules adopt the same SwiGLU design as the base model. In this architecture, SWA captures precise local patterns, while linear attention efficiently compresses long-range information. The model thus provides linear-time training complexity and constant memory usage during inference, regardless of sequence length, offering substantial efficiency gains for long-context processing. In practice, we implement linear attention using a Gated Linear Attention Module [20] (GLA), where the gating vector $\mathbf{g}_t$, derived from low-rank projection parameters, enhances expressivity and recurrent modeling capability:

$$\mathbf{S}_t = \mathrm{diag}(\mathbf{g}_t) \odot \mathbf{S}_{t-1} + \mathbf{k}_t^\top \mathbf{v}_t, \tag{10}$$

$$\mathbf{o}_t = \mathbf{q}_t \mathbf{S}_t. \tag{11}$$

SpikingBrain-76B The 76B model integrates linear attention and SWA in a 1:1 intra-layer parallel hybrid configuration, while standard full-attention layers are interleaved at a 1:6 ratio across layers. During parallel hybridization, outputs of both attention branches are normalized to ensure consistent scale and to prevent instability in the early stages of training. In addition, 128 learnable sink tokens [41, 26] are introduced to mitigate the attention-sink phenomenon in softmax attention and to enhance the flexibility of SWA for local modeling. Specifically, 128 trainable embeddings are prepended to the input; these tokens are attended by all other tokens and also attend to each other without causal masking. This functionality is implemented by customizing FlashAttention [42] kernels. The model also employs Gated Linear Attention Modules, but in this case, the gating vector is tied directly to the key vector [43, 44], i.e., $\mathbf{g}_t = 1 - \mathbf{k}_t$, eliminating the need for extra gating parameters. The FFN modules adopt a sparse Mixture-of-Experts (MoE) design: each MoE layer consists of 16 routed experts (top-1 activated) and one shared expert, so that only about 15% of parameters (12B) are activated per token. To stabilize training and control parameter growth, 7 dense FFN layers are preserved out of 28 total layers (specifically at layers 1, 2, 3, 5, 7, 9, and 11).

Connections with Brain Mechanisms Our architectural choices are closely aligned with principles observed in biological brains. i) Linear attention modules exhibit modeling properties analogous to human memory, relying on compressed and continuously updated states [15]. At each step, they retrieve information only from the current memory, showing Markov-like behavior. From a biological perspective, their stateful temporal recurrence can be viewed as a simplified abstraction of dendritic dynamics with multi-branch morphology. ii) The Mixture-of-Experts (MoE) component embodies the principle of modular sparse activation and functional specialization, reminiscent of the distributed and specialized processing found in neural circuits [16]. iii) Our spike coding scheme (see Section 3.3) draws inspiration from event-driven and adaptively sparse neuronal activation in biological systems [33]. By combining network-level sparsity (MoE) with neuron-level spiking sparsity, our approach enables on-demand allocation of computation and provides a robust two-scale efficiency strategy. Collectively, these results suggest a promising pathway for designing large-model architectures that are both efficient and biologically plausible [45].

3 Training Paradigm

3.1 Generality: Attention Map Correspondence

Here, we provide a brief analysis of the relationships among different attention maps to illustrate the feasibility of transferring parameters across attention types. First, the attention map in pre-trained Transformers is defined as:

$$\mathbf{A} = \mathrm{softmax}(\mathbf{Q}\mathbf{K}^\top \odot \mathbf{M}) \in \mathcal{R}^{n \times n}. \tag{12}$$

On this basis, SWA can be interpreted as a sparsified version of this map with a strong recency bias [17, 18, 46]:

$$\mathbf{A} = \mathrm{softmax}(\mathbf{Q}\mathbf{K}^\top \odot \mathbf{M}'). \tag{13}$$

Moreover, linear attention can be regarded as a low-rank approximation of the attention map, where the maximum rank of $\mathbf{A}$ is d (the dimensionality of the query/key vectors) [47, 48]:

$$\mathbf{A} = \mathbf{Q}\mathbf{K}^\top \odot \mathbf{M}. \tag{14}$$

Leveraging this attention-map correspondence, we can initialize the QKV projection parameters directly from a pre-trained Transformer checkpoint [10]. By reusing the learned QK similarity and performing lightweight training on a small amount of data, the attention can be adapted to these local or low-rank cases. Since SWA maintains local modeling precision and linear attention retains global interaction, combining them in a hybrid paradigm provides a closer approximation to the original attention map [49, 47, 50]. This enables a smoother transition during conversion and accelerates loss convergence.

Finally, due to the generality of this attention-map formulation, we can construct diverse attention patterns—sparse, local, linear, or hybrid—and transfer them from any pre-trained softmax attention model.

To ensure stable convergence during the conversion stage, we follow several key practices:

- **Apply non-negative activation to the QK vectors in linear attention** [10, 51], such as ReLU or Sigmoid. Because softmax attention maps are inherently non-negative, the converted attention must preserve this property to enable smooth transfer. Normalization, as in softmax attention, is applied after the linear attention outputs.
- **Keep newly introduced parameters low-rank**, such as normalization layers, gating components and sink tokens. In conversion training, the learning rate is typically lower and the dataset smaller than in pre-training, making it difficult to optimize a large number of randomly initialized parameters. We also want the pre-trained weights to guide optimization; therefore, we reuse all projection weights in the attention and FFN modules and minimize new parameters.
- **Perform long-context extension during conversion**, increasing context length while maintaining training efficiency. Since efficient attention mechanisms scale sub-quadratically, a practical approach is to restrict the training length and resource usage of the original quadratic attention model, then integrate long-context extension with continual pre-training during conversion.
- **Fully train the model during conversion to ensure performance**, using learning rate warm-up and either cross-architecture distillation [52, 53] or full-parameter training [54]. For simplicity and to fit the memory capacity of MetaX GPUs, we adopt full-parameter training without freezing the backbone.

3.2 Efficiency: Conversion-based Training

Multi-stage Conversion Pipeline The multi-stage conversion pipeline consists of continual pre-training (CPT) with long-context extension, followed by supervised fine-tuning (SFT). For demonstration purposes, all training data of SpikingBrain are sampled from high-quality open-source datasets; in practice, domain-specific data can be incorporated for vertical adaptation.

The continual pre-training process comprises three stages which progressively extend the context window while adapting the model. In the first stage, our models are trained on 100B tokens with a sequence length of 8k, aiming to transfer attention patterns toward local or low-rank variants and ensure loss convergence. Subsequently, the second and third stages extend the sequence length to 32k and 128k, respectively, each trained with 20B to 30B tokens. The entire conversion process consumes about 150B tokens in total. Compared to the $\sim$10T tokens typically required for training from scratch, the CPT approach requires only about 2% of the data, enabling efficient adaptation under resource and budget constraints. All three stages use the Matrix [55] dataset, with long-context data generated through a simple per-domain packing strategy. The RoPE base remains consistent with the base model at 1M.

SFT is conducted in three stages, each employing domain-specific data to progressively enhance the capabilities of the model in general knowledge, dialogues, and reasoning. The first stage focuses primarily on improving fundamental language understanding and domain knowledge, using the Infinity Instruct foundational [56] dataset covering various foundational topics such as scientific knowledge, code interpretation, and mathematical problem solving. This stage is trained with 500k samples under an 8k sequence length. The second stage specializes in dialogue ability and instruction following, utilizing the Infinity Instruct chat [56] dataset that includes multi-turn conversations, task-oriented dialogues, and knowledge-based question answering. The data volume and sequence length remain consistent with the first stage. The third stage targets reasoning tasks, using a high-quality reasoning dataset [57, 58] distilled via the DeepSeek-R1 [59] method, containing examples with detailed chain-of-thought annotations for mathematical proofs, logical reasoning, case analyses, and other multi-step inference problems. To ensure cross-linguistic reasoning, the dataset maintains a 1:1 Chinese-English ratio. A total of 150k samples are used under a sequence length of 8k.

MoE Upcycling To efficiently expand a dense model into an MoE architecture, we adopt the upcycling technique [32], which increases model capacity while reusing the knowledge encoded in the original parameters. At initialization, the FFN in the base dense model is replicated into N experts, and a randomized router is introduced. The router selects the top-k experts for each token with probability $\mathbf{p}$ and outputs their weighted sum, ensuring functional equivalence to the original dense FFN at the start:

$$E_1(\mathbf{x}) := E_2(\mathbf{x}) := \cdots := E_N(\mathbf{x}) := \mathrm{FFN}(\mathbf{x}), \quad \text{at initialization.} \tag{15}$$

$$\mathbf{p} = \sigma(\mathbf{W}_r\mathbf{x}), \quad \mathcal{I} = \{i \mid p_i \in \text{top-}k(\mathbf{p})\}, \tag{16}$$

$$\text{MoE\_activation}(\mathbf{x}) = \sum_{i \in \mathcal{I}} p_i \times E_i(\mathbf{x}). \tag{17}$$

During training, stochastic routing and data noise gradually break the symmetry among experts, leading to differentiated gradients and specialized expert weights. In addition to the N routed experts, our 76B model includes S shared experts ($S = 1$ in our setting) that are always active for all tokens, helping stabilize the conversion process.

Directly replicating and activating multiple experts, however, amplifies the output scale. To maintain consistency between MoE and dense FFN outputs, we rescale the expert weights [11]. The relationship between MoE activation and dense activation is:

$$\begin{aligned} \text{MoE\_activation} &= S \times \text{dense\_activation} + \frac{k}{N}\text{dense\_activation} \qquad (18)\\ &= (S + \frac{k}{N})\text{dense\_activation}. \qquad (19) \end{aligned}$$

Accordingly, the scaling factor is defined as:

$$\text{scaling\_factor} = \sqrt[3]{\frac{1}{S + \frac{k}{N}}}. \tag{20}$$

We apply this factor to both shared and routed experts at initialization:

$$E_i(\mathbf{x}) := \text{scaling\_factor} \times \mathrm{FFN}(\mathbf{x}), \quad \text{at initialization.} \tag{21}$$

3.3 Spiking Driven LLMs

Inspired by biological computation mechanisms (event-driven and sparse activation) and aiming to balance performance with efficiency, we propose a dedicated spiking strategy that encodes activations as equivalent integer values and spike sequences. This method can be applied both during and after training, converting the activations of large models into spikes. To further improve energy efficiency, we quantize both model weights and the KV cache to INT8 precision in conjunction with the spiking process. Integrated with SpikingBrain's lightweight conversion pipeline, this approach requires no full fine-tuning; a small calibration set is sufficient to optimize quantization parameters. For SpikingBrain-7B, the entire optimization process takes about 1.5 hours on a single GPU with 15 GB memory, significantly reducing deployment cost while preserving accuracy.

Our activation spiking scheme follows a decoupled two-step approach: i) **Adaptive-threshold spiking during optimization**: single-step generation of integer spike counts while maintaining appropriate neuronal firing activity. ii) **Spike coding during inference**: expansion of spike counts into sparse spike trains over virtual time steps. This approach enables the integer-based formulation to support computationally efficient optimization on GPUs, while the expanded spiking formulation provides event-driven, energy-efficient inference when combined with specialized hardware.

3.3.1 Step 1: Adaptive Threshold Spiking

The first step, adaptive-threshold spiking, focuses on the single-step generation of integer spike counts. At this stage, activations are converted using a simplified adaptive-threshold neuron model. The core idea is to design a dynamic threshold that ensures neurons remain statistically balanced—neither over-excited or over-quiescent—thereby avoiding redundant spikes or information loss often caused by fixed thresholds. Specifically, we define:

$$\mathbf{x} = \sum_{t=1}^{T} \mathbf{x}_t, \quad \mathbf{s}_{\text{INT}} = \sum_{t=1}^{T} \mathbf{s}_t, \quad V_{\text{th}}(\mathbf{x}) = \frac{1}{k}\,\text{mean}(\text{abs}(\mathbf{x})), \tag{22}$$

where $\mathbf{x}$ denotes the continuous floating-point activations at the projection layer of the large model, equivalent to the accumulated multi-step inputs $\mathbf{x}_t$ in conventional SNNs. $V\text{th}(\mathbf{x})$ is the adaptive threshold determined by the membrane potential, k is a hyperparameter that controls the firing rate, and $\mathbf{s}_{\text{INT}}$ is the aggregated integer spike count.

We adopt simplified adaptive-threshold spiking neurons to convert continuous activations into integer spike counts. By eliminating the decay factor (i.e., using an IF neuron model) and utilizing the soft-reset mechanism, the inference dynamics can be expressed as:

$$\mathbf{v}_{t+1} = \mathbf{v}_t - V_{\text{th}} \cdot \mathbf{s}_t + \mathbf{x}_{t+1}, \tag{23}$$

$$\mathbf{s}_t = 1 \;\text{ if }\; \mathbf{v}_t \geq V_{\text{th}}(\mathbf{x}) \;\text{ else }\; 0. \tag{24}$$

During optimization, the temporal dimension can be collapsed into a single-step computation:

$$\mathbf{v}_T = \mathbf{x}, \quad \mathbf{s}_{\text{INT}} = \text{round}\left(\frac{\mathbf{v}_T}{V_{\text{th}}(\mathbf{x})}\right). \tag{25}$$

This single-step integer formulation enables computationally efficient and stable optimization on GPUs, while the adaptive threshold maintains appropriate firing rates. As a result, our spiking scheme preserves the essential characteristics of biological neurons, yet is sufficiently simplified to be practical for large-scale models.

Furthermore, controlling firing activity is the primary motivation for adopting adaptive-threshold neurons rather than LIF neurons in our SpikingBrain architecture. The effects of the adaptive threshold and hyperparameter k on firing activity can be summarized as follows:

- **Threshold dynamics**: $V_{th}(\mathbf{x})$ is determined by the mean absolute membrane potential. When the input potential is high, $V_{th}(\mathbf{x})$ increases, preventing excessive spiking and thus controlling sparsity to reduce redundant computation. When the input potential is low, $V_{th}(\mathbf{x})$ decreases, allowing neurons to emit a small number of spikes to retain key information and avoid accuracy loss from inactivity. Statistically, membrane potentials often follow a long-tailed Gaussian-like distribution with occasional outliers. In this setting, the mean absolute value approximates 0.8 times the standard deviation, providing a stable metric for regulating spike activity.
- **Effect of the hyperparameter** k: k directly scales the threshold and thereby determines the distribution of spike counts. A larger k lowers $V_{th}(\mathbf{x})$, producing higher spike counts $\mathbf{s}_{INT}$ and broader firing ranges. This is suited for accuracy-critical scenarios where additional computation is acceptable. A smaller k raises $V_{th}(\mathbf{x})$, reducing spike counts and producing sparser encodings, which is advantageous for low-power edge deployments. Tuning k thus provides a flexible trade-off between accuracy and efficiency.
- **Outlier handling**: Large models often contain rare but high-magnitude activations (outliers) that significantly affect accuracy. The adaptive threshold, due to its statistical basis, is robust to these values. Neurons maintain stable activity for the majority of inputs while producing higher spike counts for rare outliers. This behavior resembles the burst response of biological neurons, ensuring that critical information carried by outliers is preserved. On specialized asynchronous hardware, these rare bursts have minimal impact on overall energy efficiency.

3.3.2 Step 2: Spike Coding

During inference, the integer spike count $\mathbf{s}_{\text{INT}}$ generated in Step 1 is expanded into a sparse spike sequence with values $\{0, 1, -1\}$ along the time dimension to support event-driven computation. The process is formulated as:

$$\mathbf{s}_{\text{INT}} = \sum_{t=1}^{T} \mathbf{s}_t, \quad \mathbf{y} = V_{\text{th}}(\mathbf{x}) \cdot \sum_{t=1}^{T} (\mathbf{W}\mathbf{s}_t), \tag{26}$$

where $\mathbf{W}$ is the weight matrix and $\mathbf{y}$ is the output of the linear projection layer. Because each $\mathbf{s}_t$ takes values from $\{0, 1, -1\}$, matrix multiplications are replaced by event-driven accumulations, thereby improving computational efficiency.

To expand $\mathbf{s}_{\text{INT}}$ into multi-step spikes $\mathbf{s}_t$, we design three encoding schemes tailored to different application needs. The primary goal is to replace dense matrix multiplications with sparse, event-driven additions when supported by appropriate hardware, while optimizing the spike firing rate and number of time steps without sacrificing representational capacity. This reduces power consumption and increases computational sparsity.

- **Binary Spike Coding {0,1}**: This is the most basic event-driven spike coding method. Each time a unit spike is fired (with a value of 1), it represents an activation of the neuron state, and the spike count is accumulated over continuous time steps. This coding scheme is intuitive and simple, with low computational overhead, making it suitable for scenarios with very low spike counts and effectively reducing system complexity. However, when representing large counts, this coding often requires many time steps to complete the representation. Additionally, the lack of directional optimization for spike firing results in a higher firing rate, further limiting system energy efficiency.
- **Ternary Spike Coding {-1,0,1}**: To improve neuronal expressivity and sparsity in event-driven computations, we introduce ternary spike coding. The core idea is to extend the traditional binary spike coding by adding inhibitory spikes (-1), resulting in three firing states: {-1,0,1}. Compared to binary coding, which can only represent positive values, ternary coding offers bidirectional expression capabilities. Here, "1" represents excitatory spikes, "-1" inhibitory spikes, and "0" the silent state, making it more aligned with the "excitation/inhibition" regulatory mechanism in biological neural systems. As shown in Figure 3 (b), ternary spike coding not only provides symmetric expression capabilities of ± 1 but also reconstructs the mapping from membrane potential to spike counts through a symmetric quantization strategy. This strategy maps activations to a symmetric distribution $[-k, \ldots, 0, \ldots, +k]$ rather than $[0, 1, 2, \ldots]$, so that low-amplitude counts (e.g., ± 1) statistically occupy a larger probability mass, effectively absorbing high-frequency large counts from the tail of the original distribution. As a result, this scheme halves the number of time steps and reduces the spike firing rate by more than 2×, significantly improving sparsity and energy efficiency without sacrificing expressivity.
- **Bitwise Spike Coding**: Bitwise coding can be viewed as an event sequence encoding method, where integer count values are expanded bit by bit into spike events over time steps. Each time step corresponds to one binary bit of the count value, significantly compressing the time dimension overhead. As shown in Figure 3 (c), this mechanism supports three implementation forms to accommodate different symbolic representations and precision requirements: i) **Pure bitwise encoding**, suitable for positive integers, provides extremely high temporal compression in high-count scenarios. For instance, a count of 256 requires 256 consecutive time steps in binary coding, 128 in ternary coding, but only 8 steps in 8-bit bitwise encoding. ii) **Bidirectional bitwise encoding** uses ± 1 to represent each bit, replacing negative part with -1. This halves the time step emission rate and reduces the required number of time steps by one (e.g., 7 steps for a count of 256), while maintaining equivalence in representation. iii) **Two's complement encoding** incorporates sign information into the highest bit, supporting both positive and negative counts while retaining binary simplicity and biological plausibility. Compared to ternary coding, which still requires many time steps for higher bit counts, the bitwise encoding scheme significantly compresses time steps, reducing total spike emissions by up to 8× (for an 8-bit count). This effectively reduces overall spike communication overhead and computational load while maintaining precision, making it particularly suitable for high-precision, low-power, and time-sensitive neuromorphic computing tasks.

3.3.3 Hardware Adaptation and Deployment Potential

Our spiking scheme can be executed on GPUs. By collapsing the temporal dimension into a single step, it avoids the incompatibility between event-driven computation and the synchronous architecture of GPUs, enabling direct simulation and inference validation on general-purpose hardware.

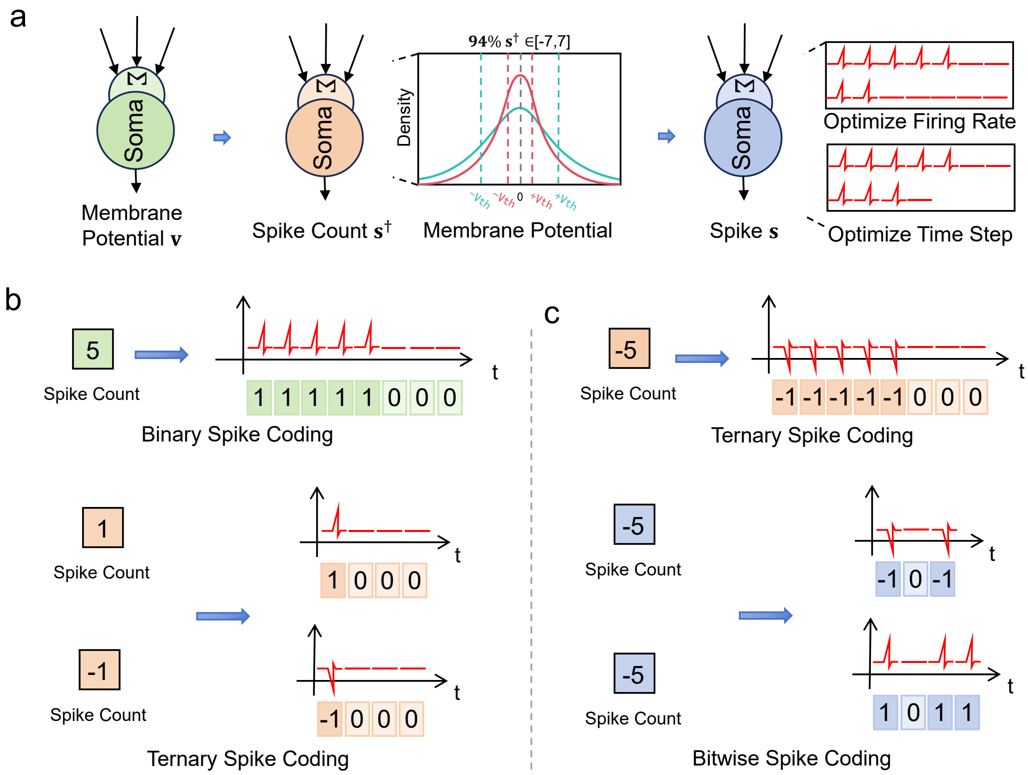


Figure 3: **Schematic of three spike coding schemes.** (a) An adaptive threshold maps membrane potential to spike counts, which are expanded over virtual timesteps into sparse spike trains, enabling the conversion from continuous activations to discrete spikes. (b) Ternary vs. Binary: binary uses $\{0, 1\}$ to represent "spike/no-spike", while ternary uses $\{-1, 0, 1\}$ to encode both excitatory and inhibitory events. Compared with binary, ternary reduces both timesteps and firing rate by half. (c) Bitwise vs. Ternary: spike counts are unfolded into binary bits across timesteps to achieve temporal compression. In high-count scenarios, the required timesteps are far fewer than ternary, leading to significantly higher efficiency.

However, the synchronous design of GPUs cannot fully exploit the event-driven and sparse-asynchronous advantages of spiking signals [13, 14]. GPUs operate under fixed high-frequency clock cycles, unlike biological neural systems that remain idle without spikes and trigger computation only upon spiking. To fully unlock the low-power potential of our scheme, deployment on specialized asynchronous hardware architectures—such as neuromorphic chips or spiking processors based on asynchronous circuit design—for matrix operations is required. These platforms can natively respond to sparse spike events without requiring clock synchronization [12, 40]: circuits remain in a quiescent, low-power state in the absence of spikes and perform addition operations only when spikes occur.

This approach maximizes energy efficiency and offers a practical pathway for deploying low-power brain-inspired LLMs in edge scenarios such as industrial control and mobile devices. It also outlines a reference technology roadmap for developing next-generation energy-efficient neuromorphic hardware, supporting the shift of large-scale models from compute-driven to energy-optimized paradigm.

4 Implementation on the MetaX Cluster

Distributed training of brain-inspired large models on the MetaX non-NVIDIA cluster poses several challenges. These include: ensuring stability for large-scale parallel training, sustaining intensive communication under long-sequence parallel topologies, and adapting CUDA and Triton operators for hybrid attention. In this work, we introduce targeted optimizations to address each of these challenges, which enable the successful training of both the SpikingBrain-7B and -76B models. This section details the adaptations for distributed training, operator customization, and the parallel topologies used in our conversion pipeline.

4.1 Distributed Training Adaptation

To enable efficient and stable training on non-NVIDIA GPU clusters, the MetaX software platform has implemented a series of hardware-aware adaptations across multiple dimensions, including MoE optimization, computation-communication overlap, memory optimization, auto-tuning, distributed cluster training, and kernel fusion. Some of these optimizations are designed as plug-in modules, enhancing the flexibility of the training framework.

For MoE training, four strategies are introduced to mitigate memory and computational pressure during the early training stages:

- **Hot-Cold Expert Optimization**: To address communication hotspots caused by uneven token routing in the early phases, frequently accessed experts are replicated locally. This reduces communication overhead until expert load stabilizes, after which the local copies are removed.

- **Adaptive Recomputation**: When a heavily utilized expert processes tokens beyond a set threshold, activation recomputation is triggered to save memory. This technique is automatically disabled in later stages when the token distribution is balanced.

- **Multi-Granularity Recomputation** [60]: To balance computation and memory under high memory pressure, experts support three recomputation levels: lightweight (activations and router), moderate (including FFN and shared experts), and full (entire MoE layer).

- **Length Alignment**: Variations in token counts per expert can affect the efficiency of GEMM. Token dropping and padding are applied to unify input sequence lengths, improving overall computational efficiency.

To address high communication overhead in distributed training, MetaX utilizes SDMA engines for intra-node high-speed data transfer. For tensor parallelism and expert parallelism, communication kernels are fused with compute kernels to reduce scheduling conflicts [61]. Memory optimizations include fine-grained offloading of transformer layer weights or optimizer states to the CPU, as well as selective recomputation of normalization layers, activation functions, or individual transformer layers based on memory demand.

For long-sequence parallelism, the MetaX GPU cluster supports efficient multi-GPU and multi-node communication with stable training throughput. This alleviates synchronization challenges and improves both GPU memory utilization and compute efficiency. Inter-GPU connectivity is enhanced through MetaLink and PCIe 5.0, eliminating intra-node bottlenecks. RDMA over InfiniBand 200/400G or RoCE ensures low-latency, high-bandwidth inter-node communication, meeting the requirements of large-scale distributed training.

The software stack also incorporates auto-tuning and fast recovery mechanisms for large-scale clusters:

- **Auto-Tuning**: An automated tuning engine covers operators, memory, and communication. It benchmarks common operators, models communication performance across network topologies, and explores parallel configuration spaces to recommend top-k strategies, minimizing manual effort.

- **Fast Checkpointing**: The DLRover [62] Flash Checkpoint technique first writes training state (model weights, optimizer, learning rate scheduler, etc.) to CPU memory before asynchronously persisting to a distributed file system. This reduces the I/O time by 85% and shortens recovery time after failures. The built-in profiling tools automatically instrument training jobs, monitor performance per layer, detect slow nodes, and trigger alerts, helping maintain high cluster utilization.

4.2 Operator Adaptation

The efficient adaptation of SpikingBrain large models to MetaX GPUs relies on the comprehensive MetaX software ecosystem. The overall process can be divided into two major components: Triton adaptation and CUDA migration to the MetaX self-developed MACA framework. These two pathways are designed for different subsets of operators within SpikingBrain models. While distinct in implementation, they complement each other and together constitute a complete hardware adaptation framework tailored for MetaX GPUs (see Figure 4).

In the Triton adaptation workflow, we designed four progressive stages based on the MetaX technology stack and Triton's compilation pipeline. The objective is to fully exploit the MetaX GPU's strengths in compilation optimization and instruction scheduling, while keeping the adaptation process transparent to higher-level applications:

- **JIT Compilation Optimization**: During Triton's just-in-time compilation stage, the Gated Linear Attention (GLA) [20] operators used in SpikingBrain are reorganized at the code level. By refining instruction pipelining and register allocation, the operators achieve a balance between memory access latency and computational density. This stage directly leverages Triton's hardware-agnostic general optimization strategies, enabling dynamic and efficient execution of compute kernels.
- **Grid Search and Architecture Matching**: Systematic exploration of Block/Grid configurations is carried out and mapped to the scale of MetaX GPU streaming multiprocessors and thread concurrency features. By extending Triton's architectural support to the MetaX GPU family, we significantly improve the throughput of matrix multiplication in linear attention operators.
- **Cache Structure Specification**: To minimize redundant computation and memory traffic, we introduce fixed, structured cache designs aligned with MetaX's on-chip memory hierarchy. For instance, by optimizing reuse strategies for weight matrices and Key-Value sequences in line with L2 cache capacity and bandwidth properties, inference efficiency for long-sequence processing is markedly improved.
- **Target Code Generation via MetaX Compiler**: In the final stage, Triton kernels are transformed by the MetaX compiler backend into executable target machine code for MetaX GPUs. Beyond standard code generation, the compiler introduces deep optimizations for tensor cores, SIMD instruction sets, and memory alignment constraints, ensuring operators run at near-peak hardware performance.

In the CUDA-to-MACA migration workflow, four tightly integrated stages are involved:

- **Call Layer Adaptation**: Original CUDA APIs and runtime interfaces are encapsulated and redirected to the MACA framework, allowing seamless mapping without altering user-facing code. This significantly reduces migration overhead for developers.
- **Optimization Analysis**: Using performance profiling tools, we identify performance bottlenecks in our SpikingBrain models, such as long-sequence attention operators involving softmax, exp/sum, dot-product accumulation, and certain normalization operations. These operators are reimplemented using native MACA optimizations to fully leverage tensor acceleration units.
- **Cache and Architecture Matching**: Similar to the Triton adaptation process, cache-awareness is embedded into the MACA execution engine. Key data structures (e.g., intermediate accumulation matrices, positional encoding caches) are retained in high-speed cache, reducing global memory traffic and improving efficiency through hardware-specific optimizations.
- **Replacement with MetaX Acceleration Libraries**: Finally, core baseline operators are substituted with MetaX's high-performance libraries. The Hotspot operator acceleration library is used to replace foundational calls, while advanced libraries such as mcFlashInfer2, specifically optimized for MetaX hardware, are employed to deliver sustained performance improvements at the hardware level.

It is worth emphasizing that the entire adaptation and migration process for brain-inspired large models on MetaX GPUs adheres to a user-friendly design philosophy. Whether through Triton kernel

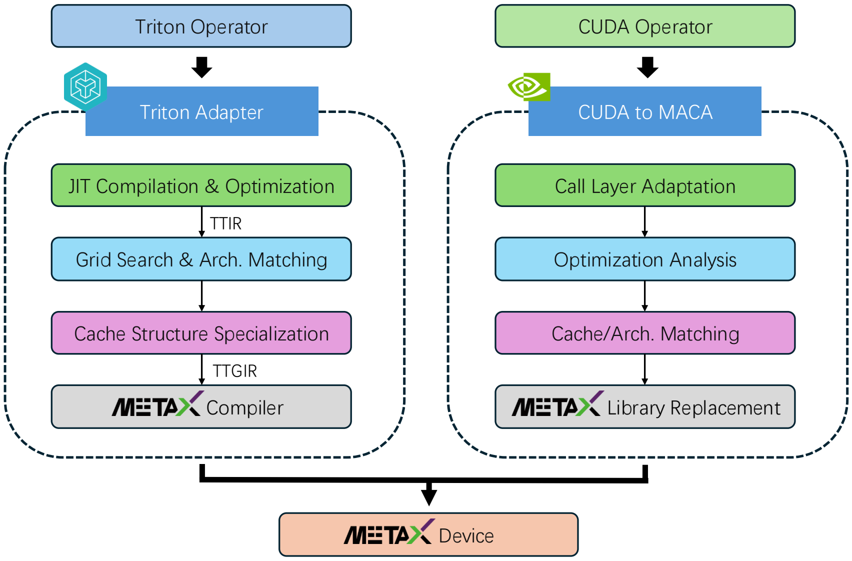


Figure 4: **Operator adaptation of SpikingBrain on MetaX GPUs.** The adaptation involves two complementary pathways: Triton adaptation and CUDA migration to the MACA framework, covering different operator subsets. Together, they form a unified hardware adaptation framework tailored for MetaX GPUs.

optimization or CUDA-to-MACA migration, end users can preserve their existing programming practices and interface calls. In most cases, no extensive code modifications are required for the models to run efficiently on MetaX GPUs. Meanwhile, the MetaX ecosystem provides unified debugging and performance analysis tools, enabling developers to transparently monitor execution characteristics on hardware and further fine-tune performance as needed.

4.3 Parallel Topology

The memory demands of training large language models often exceed the capacity of a single GPU. To make training such models feasible, it is essential to employ efficient and scalable distributed training techniques alongside other memory reduction methods. These approaches distribute computational and storage loads while maintaining training efficiency. In this section, we present the parallelization strategies and training topologies employed to train large brain-inspired models on the MetaX cluster.

Data Parallelism (DP) Data parallelism [63] involves partitioning the training data into batches, each processed by a separate GPU. Each GPU maintains a complete replica of the model and performs forward and backward passes independently, while gradient synchronization occurs only at the beginning of the backward propagation. Data parallelism offers low communication overhead and excellent scalability. As computing resources increase, the scale of data parallelism can grow accordingly, increasing the total batch size to enhance overall throughput and reduce training time. During training, we employ ZeRO [64] to eliminate redundancy of the optimizer states, effectively distributing the GPU memory pressure.

Pipeline Parallelism (PP) Pipeline parallelism divides the model into different stages by layers. Each GPU is assigned a subset of layers, and intermediate activations are communicated between GPUs during the forward and backward passes using peer-to-peer (p2p) communication. We employ the 1F1B [65, 66] scheduling algorithm for efficient pipeline execution. Furthermore, incorporating Mixture-of-Experts (MoE) layers interspersed with dense FFN layers helps mitigate the memory imbalance issue, where the first pipeline stage typically exhibits higher memory usage compared to other stages.

Expert Parallelism (EP) In Mixture-of-Experts models, most parameters reside within the expert networks. Expert parallelism [67] addresses this by partitioning experts across multiple GPUs. This requires all-to-all communication to dispatch and combine tokens. We employ Grouped GEMM kernels to accelerate computation across experts, and an auxiliary loss is used to promote balanced token routing across experts.

Sequence Parallelism (SP) Training with long sequences incurs substantial memory costs for storing intermediate activations. Sequence parallelism tackles this by splitting the input sequence along its length dimension, with each GPU processes a contiguous chunk. Communication primitives like all-to-all (a2a) or peer-to-peer (p2p) are then used to exchange necessary information between devices, enabling the computation of attention over the entire sequence. For our experiments, we employ DeepSpeed Ulysses [68] for efficient sequence parallelism and incorporate specialized algorithms for linear attention modules to further improve performance. For linear attention, AllGather communication is used to collect intermediate local states [69] when the SP size is small, while for larger SP sizes (e.g., across multiple nodes), the All-Scan primitive from ZeCO [70] sequence parallelism is adopted to effectively eliminate communication overhead.

Parallel Topology of SpikingBrain The SpikingBrain-7B model is trained using the Colossal-AI [71] framework. For 128k sequence length training, a parallel strategy of 8-way data parallelism and 32-way sequence parallelism is adopted. To reduce GPU memory consumption, techniques such as ZeRO-2 [64] and activation recomputation [72, 60] are applied. Sequence parallelism is implemented with DeepSpeed Ulysses [68], which performs all-to-all communication within nodes. In addition, the number of attention heads is padded to satisfy the partition requirements.

The SpikingBrain-76B model is trained using the Megatron [73] framework. For 8k sequence length training, the parallel configuration consists of 128-way data parallelism, 8-way expert parallelism, and 4-way pipeline parallelism. ZeRO [64] and selective activation recomputation [72, 60] are applied to reduce memory consumption. For extended contexts of 32k and 128k, 4-way and 8-way sequence parallelism are added on top of the base 8k parallel strategy, respectively. The 76B hybrid model integrates linear attention and (local) softmax attention. The softmax attention branch uses DeepSpeed Ulysses [68] with all-to-all communication and head padding to enable partitioning, while the linear attention branch employs the AllGather-based SP algorithm (LASP-2 [69]) to reduce communication overhead.

5 Results

5.1 Downstream Performance

We conduct a comprehensive evaluation of the two converted efficient models, SpikingBrain-7B and -76B, across a broad set of downstream tasks[3]. They are compared against open-source baselines of similar scale, including Hybrid, Transformer, and MoE architectures, as well as the strong base model Qwen2.5-7B. To ensure fairness, all evaluations are performed under identical settings using the OpenCompass [74] framework. In selecting evaluation metrics, we place greater emphasis on pretraining-oriented general-purpose benchmarks (e.g., MMLU, CMMLU, C-Eval, ARC-C, HS), as these better indicate whether our models—trained with fewer than 200B tokens during efficient conversion—can inherit the knowledge and modeling capacity of the base model. Moreover, these benchmarks reveal whether the converted models preserve strong generalization while demonstrating significant long-sequence efficiency advantages, without being confounded by data quality factors such as alignment performance after SFT.

As shown in Table 1, our SpikingBrain-7B linear model recovers nearly 90% of the base model's performance across benchmarks, reaching a level comparable to advanced Transformer models such as Mistral-7B and Llama3-8B. This indicates that, with appropriate training strategies, linear attention architectures can preserve strong general-purpose modeling ability while substantially reducing inference complexity. However, due to the significant structural modifications of the attention mechanism, there remains a certain performance gap between the 7B pure linear model and base

[3] This section evaluates the effectiveness of the architectural design and conversion pipeline. All models remain in floating-point format; results for spiking models are presented in Section 5.5.

Qwen2.5-7B. This suggests that the extent of recovery is strongly influenced by architectural changes when pursuing extreme efficiency.

Table 1: **Performance evaluation of the SpikingBrain-7B pre-trained model.** All models are tested with the HuggingFace framework and evaluated using a perplexity-based method. Except for Qwen2.5, the other baselines are trained on limited Chinese data, resulting in clear disadvantages on CMMLU and C-Eval.

	SpikingBrain-7B	Falcon-Mamba [75]	Mistral [76]	Zamba-v1 [77]	Llama3.1 [78]	Qwen2.5 [7] (base)
Params	7B	7B	7B	7B	8B	7B
Tokens	+150B	5.8T	–	1T	15T	7T
Complexity Type	Linear	Linear	Linear	Hybrid	Quadratic	Quadratic
Benchmarks						
MMLU [79]	**65.84**	63.24	62.56	58.19	65.74	74.21
CMMLU [80]	**71.58**	42.50	44.58	38.42	52.44	81.73
ARC-C [81]	**43.32**	47.53	45.13	37.18	51.96	44.04
HS [82]	**70.89**	71.50	75.81	52.02	71.60	72.81
Ceval [83]	**69.80**	41.93	47.04	36.40	51.46	81.60

As shown in Table 2, our SpikingBrain-76B hybrid-linear MoE model nearly closes the performance gap with its base model by scaling parameter size and adopting more fine-grained attention designs. Despite using fewer activated parameters, it achieves performance comparable to, and in some cases surpassing, representative Transformer models such as Llama2-70B, Mixtral-8×7B, and Gemma2-27B. Compared with SpikingBrain-7B, the 76B model delivers consistent improvements across all benchmarks, providing strong evidence that within the conversion framework, flexible choices in attention and FFN design allow a better balance between performance and efficiency, while enabling architectures tailored to specific application needs.

Table 2: **Performance evaluation of the SpikingBrain-76B pre-trained model.** All models are tested with the vLLM [84] framework and evaluated using a perplexity-based method. Except for Qwen2.5, the other baselines are trained on limited Chinese data, resulting in clear disadvantages on CMMLU and C-Eval.

	SpikingBrain-76B (76B-A12B)	Jamba [23] (52B-A12B)	Mixtral-8*7B [85] (47B-A13B)	LLama2-70b [86]	Gemma2-27B [87]	Qwen2.5 [31] (base)
Params	12B/76B	12B/52B	13B/47B	70B	27B	7B
Tokens	+160B	–	–	2T	13T	7T
Complexity Type	Hybrid	Hybrid	Quadratic	Quadratic	Quadratic	Quadratic
Benchmarks						
MMLU [79]	**73.58**	67.17	71.23	69.57	75.94	75.31
CMMLU [80]	**78.83**	51.11	52.70	52.94	61.80	81.50
ARC-C [81]	**42.00**	49.10	48.98	46.21	57.64	43.56
HS [82]	**73.31**	79.39	79.42	79.15	79.34	73.37
Ceval [83]	**78.89**	48.94	54.39	49.26	61.02	81.68

We conduct a three-stage SFT process on the converted pre-trained models to endow them with instruction-following and chain-of-thought reasoning abilities. As shown in Table 3, our aligned models achieve performance on par with open-source chat baselines of similar scale across general knowledge, long-sequence modeling, and instruction following. Importantly, the SFT process does not cause overfitting or degrade the general capabilities acquired during pretraining, indicating that the (hybrid) linear architectures maintain both stability and scalability under alignment training. Given current data limitations, we primarily aim to demonstrate effective alignment results obtained on MetaX computing clusters, leaving large-scale and higher-quality post-training alignment as future work.

5.2 Long-context Efficiency

Beyond task performance, we also place particular emphasis on the efficiency of our models in long-sequence inference. Benefiting from the integration of (hybrid) linear attention with the MoE mechanism, both SpikingBrain models exhibit substantial inference speedups. Since the 7B model is more compact and shows stronger efficiency gains, we adapt it to the HuggingFace framework with

Table 3: **Performance Evaluation of SpikingBrain Chat Models.** All models are tested with the vLLM framework and evaluated using a generation-based method. Except for Qwen2.5, the other baselines are trained on limited Chinese data, resulting in clear disadvantages on CMMLU and C-Eval. For QuALITY and IFEval, we report results from the non-CoT model (after SFT stage 2) to avoid chain-of-thought interference.

	SpikingBrain-7B	**SpikingBrain-76B**	Llama3 [78]	Qwen2.5 [31]	Mixtral [85]
Params	7B	12B/76B	8B	7B	13B/47B
Complexity Type	Linear	Hybrid	Quadratic	Quadratic	Quadratic
Benchmarks					
MMLU [79]	**65.57**	**73.71**	68.69	75.17	71.03
CMMLU [80]	**68.76**	**77.41**	55.17	79.14	51.03
HS [82]	**68.95**	**86.63**	76.80	85.39	75.63
Ceval [83]	**69.07**	**76.32**	55.01	77.93	50.88
NQ [88]	**21.47**	**21.55**	30.97	17.67	28.48
TrQ [89]	**57.03**	**55.13**	65.78	55.72	71.00
QuALITY [90]	**60.12**	**69.56**	66.25	73.63	51.34
IFEval [91]	**42.70**	**49.72**	73.01	73.20	48.06

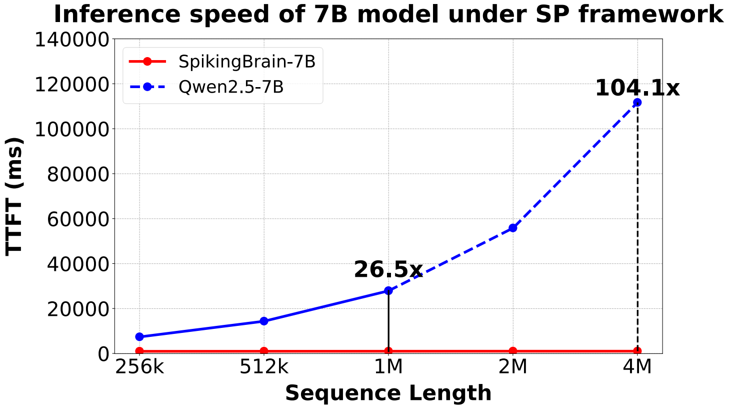


Figure 5: **TTFT comparison under sequence parallelism.** Time to First Token (TTFT) latency of SpikingBrain-7B compared with the Qwen2.5-7B baseline across different input lengths. For inputs beyond 2M tokens, direct evaluation of Qwen2.5-7B is constrained by resource limits and attention head count; results are therefore extrapolated using a fitted scaling curve.

multi-device sequence-parallel inference (ZeCO + p2p), successfully supporting efficient prefill up to 4M tokens. As shown in Table 4 and Figure 5, at an input length of 1M tokens, the SpikingBrain-7B model achieves a 26.5× speedup in TTFT (Time to First Token) compared with the Qwen2.5 baseline using full attention and all-to-all communication. Due to resource limitations and the number of attention heads, Qwen2.5 is difficult to evaluate at 4M tokens. Based on the fitted scaling curve, we conservatively estimate a speedup of over 100×. Moreover, as sequence length and GPU count increase proportionally, the time overhead of our model remains nearly constant. These results clearly demonstrate the scalability and superiority of our approach in ultra-long-sequence scenarios, providing strong support for the deployment of more efficient large language models in practical applications.

In addition, we perform a systematic evaluation of inference speed for both the 7B and 76B models without sequence parallelism enabled. The test environments include HuggingFace single-GPU deployment on MetaX C550 GPUs as well as both single- and multi-GPU deployments using vLLM. As shown in Table 5, SpikingBrain-7B consistently achieves multiplicative speedups over Qwen2.5 at a sequence length of 128k (1.41× on HuggingFace and 2.75× on vLLM), while delivering performance comparable to—or even exceeding—that of the Mistral model, which also employs SWA to achieve linear complexity. For the larger 76B model, inference speed is equally compelling: it not only significantly outperforms Llama2-70B, but even surpasses the MoE baseline Mixtral-8×7B in settings

Table 4: **Inference speed comparison of SpikingBrain-7B under sequence parallelism (SP).** The metric is TTFT (ms), defined as the latency to complete prefill and generate the first token for a given prompt length. In SP configuration, our 7B model uses ZeCO for the linear attention module and P2P communication for the SWA module, while the Qwen2.5 baseline employs A2A communication. All measurements are conducted on NVIDIA H100 GPUs and averaged over 10 runs. "–" indicates infeasible measurement due to resource constraints.

Sequence length	GPU count	**SpikingBrain-7B**	Qwen2.5-7B
256k	8	1015	7419
512k	16	1037	14398
1M	32	1054	27929
2M	64	1070	–
4M	128	1073	–

with Expert Parallel (EP) enabled. These results highlight the broad applicability and performance advantages of our approach across model scales and inference frameworks.

For long-sequence training, we further measure Throughput per GPU Second (TGS) on SpikingBrain-7B and the Qwen2.5 baseline under the sequence parallelism framework. The experiments show a 5.36× throughput advantage for our 7B model at a sequence length of 128k, consistent with the TTFT improvements observed during inference.

5.3 CPU-side Inference

In this section, we demonstrate the deployment of compressed 1B-scale models, including the SpikingBrain linear model with SWA and GLA, as well as Llama3.2 [78], on a CPU-based mobile inference framework. The deployment leverages llama.cpp as the inference backend. As illustrated in Figure 6, the workflow consists of four main steps:

- **Weight conversion and quantization**. The trained weights are first converted into the GGUF format with an appropriate quantization strategy. Unlike generic training formats, GGUF emphasizes compact encoding and cross-platform consistency, yielding more predictable load latency and memory footprint during CPU inference. Floating-point weights are compressed into low-bit integers (e.g., Q4, Q5). Combined with efficient decoding and optimized matrix-multiplication kernels, this greatly improves throughput while reducing memory usage.
- **Model registration and tensor mapping**. A dedicated computation graph is constructed by registering architecture components and mapping tensors within the inference engine. Each tensor in the weight file is directly linked to its corresponding graph structure, with associated metadata such as shape and precision format (e.g., float32, float64).
- **Graph and operator optimization**. Following the forward propagation flow, we implement key operators including attention, rotary position embedding (RoPE [92]), and residual connections. Two major optimizations are applied: i) Operator fusion: combining matrix multiplication, bias addition, and nonlinear activation into a single kernel to reduce intermediate storage and memory access. ii) Graph-level optimization: decomposing and reorganizing operations such as RMSNorm, Softmax, and RoPE to enable fusion with matrix multiplication or low-level kernels, minimizing redundant computation.
- **Quantized inference**. Finally, the quantized weights are loaded, and forward computation is performed in low precision. By default, Q4_0 quantization is applied to the weights to balance computational throughput with memory usage. In addition, we use compact single-step integer activations rather than expanded spike trains, ensuring compatibility with current CPU hardware.

A key innovation of the SpikingBrain-1B linear model lies in its inter-layer hybridization of linear attention and SWA, which avoids full computation over all historical Keys and Values. This design not only reduces KV-cache memory consumption but also significantly lowers inference latency. Compared with conventional Transformer-based models, our computation graph explicitly integrates these mechanisms to further optimize KV-cache management. We evaluate decoding speed at different

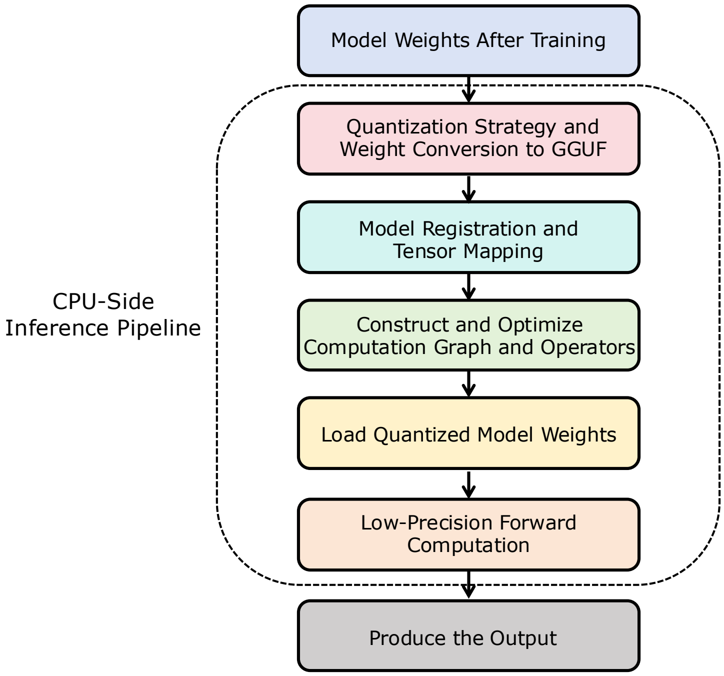

Model Weights After Training
CPU-Side Inference Pipeline
Quantization Strategy and Weight Conversion to GGUF
Model Registration and Tensor Mapping
Construct and Optimize Computation Graph and Operators
Load Quantized Model Weights
Low-Precision Forward Computation
Produce the Output

Figure 6: **Overview of the CPU-side inference pipeline.** The workflow includes four main steps: weight conversion and quantization, model registration and tensor mapping, graph and operator optimization, and quantized inference.

output lengths using a 1k input. The hardware setup includes an Intel Core i5-12600KF CPU, 64 GB of memory, and Ubuntu 22.04.4 LTS, with the project compiled using CMake 3.28.3. As shown in Figure 7, the SpikingBrain-1B model maintains constant computation and memory overhead during decoding, resulting in stable throughput as the output sequence length increases. In contrast, the Llama3.2-1B baseline shows a sharp decline in decoding speed due to full KV-cache computation. Overall, SpikingBrain-1B achieves speedups of 4.04×, 7.52×, and 15.39× at sequence lengths of 64k, 128k, and 256k, respectively.

This workflow allows the inference engine to seamlessly adapt brain-inspired models for CPU deployment, achieving substantial gains in computational efficiency, memory utilization, and hardware adaptability. It provides a practical solution for running large models efficiently in resource-constrained environments.

Table 5: **Inference latency (ms) comparison across models, frameworks, and sequence lengths.** Tests are conducted on HuggingFace single-GPU deployment with MetaX C550 GPUs and on vLLM with both single- and multi-GPU settings. Latency is measured as the time to process the input sequence and generate 128 output tokens.

	SpikingBrain -7B	Mistral -7B-v0.1	Qwen2.5 -7B	SpikingBrain-7B	Mistral -7B-v0.1	Qwen2.5 -7B	**SpikingBrain -76B**	Mixtral -8x7B-v0.1	Llama -2-70b
GPUs	1			1			4		
Frameworks	HuggingFace			vllm			vllm		
Expert Parallel	Non-MoE						Enable / Disable		Non-MoE
32k	8452	10202	7969	4369	4570	6421	9451 / 5305	10477 / 3209	15881
64k	11685	16467	16257	7399	7084	15093	14077 / 8142	19062 / 4840	32953
128k	15974	28382	38487	12048	11867	45141	27106 / 14109	34983 / 8046	86120

5.4 Performance on the MetaX Cluster

The performance of the MetaX C550 GPU cluster is evaluated from two perspectives: training efficiency and system stability. For efficiency, we report model FLOPs utilization (MFU) and tokens

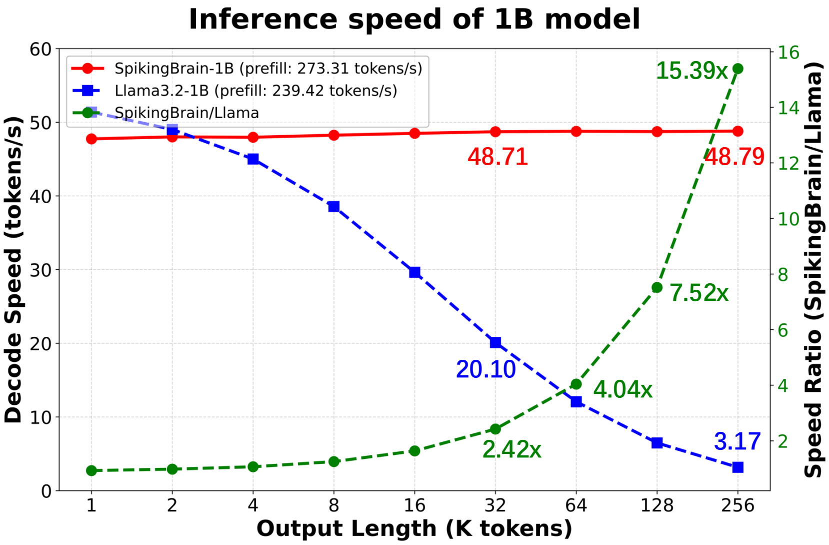


Figure 7: **Decoding speed comparison at different output lengths on a CPU-based mobile inference framework.** Results are reported for the 1B-parameter SpikingBrain linear model and the Llama3.2-1B baseline, both evaluated on an Intel Core i5-12600KF CPU. By default, Q4_0 weight quantization is applied to balance throughput and memory efficiency.

per second per GPU (TGS). During training, the cluster demonstrates consistently competitive performance. For the SpikingBrain-7B model, we achieve a TGS of 1558 and an MFU of 23.4% (8-way DP, 4-way PP, PP micro-batch size 2, global batch size 512), reflecting high computational efficiency and effective resource utilization.

Regarding stability, continuous monitoring of the cluster operation confirms exceptional reliability and robustness. The training session remains uninterrupted for more than two weeks, underscoring the stability and maturity of the MetaX hardware and software ecosystem.

Overall, these results highlight the capability of the MetaX GPU cluster to efficiently and reliably support large-scale long-duration training tasks, including the stable training of models with tens of billions of parameters.

5.5 Analysis of Spiking Scheme

We implement the joint optimization of spiking-based activation encoding and 8-bit fixed-point weight quantization (INT8 Quantization) on our pre-trained models. This combination achieves an effective balance between accuracy and energy efficiency, leveraging event-driven computation paradigm to reduce cost while preserving model performance.

Scheme deployment and accuracy maintenance Specifically, we apply spike coding to the activation inputs of all linear projection layers in our models, while the corresponding weights are quantized using symmetric INT8. Quantization parameters are calibrated with a small set of 128 text samples. As shown in Table 6, evaluations on commonsense reasoning, MMLU, and CMMLU benchmarks indicate that the average performance drop under this scheme is limited to approximately 2% for both SpikingBrain-7B and SpikingBrain-76B, confirming its effectiveness in preserving accuracy.

Spike distribution characteristics To evaluate the sparsity and efficiency of the proposed spiking scheme, we analyze the statistical firing patterns under the bitwise–ternary coding method. We collect the absolute values of input spike integers across all linear projection layers. The detailed distribution of spike counts is illustrated in Figure 8. Take the SpikingBrain-7B model for example, results show that 94.1% of values fall within [0,7], and only about 1% exceed 16. After expansion into bitwise–ternary spikes, the average number of spikes fired per channel is only 1.13, while 18.4%

Table 6: **Performance comparison of SpikingBrain before and after applying the spiking scheme.** Evaluations are performed on commonsense reasoning, MMLU, and CMMLU benchmarks. The integration of adaptive-threshold spike coding with 8-bit weight quantization results in negligible performance degradation (approximately 1–3%).

	Winogrande	Arc-e	Arc-c (norm)	HellaSwag (norm)	PIQA	MMLU	CMMLU	Avg.
SpikingBrain-7B	0.6992	0.8047	0.5566	0.6777	0.7949	0.6751	0.6904	0.6998
SpikingBrain-7B -W8ASpike	0.6895	0.7861	0.5410	0.6758	0.7979	0.6546	0.6677	0.6875
SpikingBrain -76B	0.7275	0.8125	0.5615	0.7000	0.8125	0.7247	0.7740	0.7304
SpikingBrain -76B-W8ASpike	0.7148	0.7949	0.5371	0.6863	0.8004	0.7081	0.7512	0.7133

of channels do not fire any spikes at all. Since 94.1% of values are within 7 (i.e., within the INT3 range), we further analyze the firing rate under a step size of 3. By filling inactive bits within the 3-step window with 0, the overall sparsity reaches approximately 69.15%.

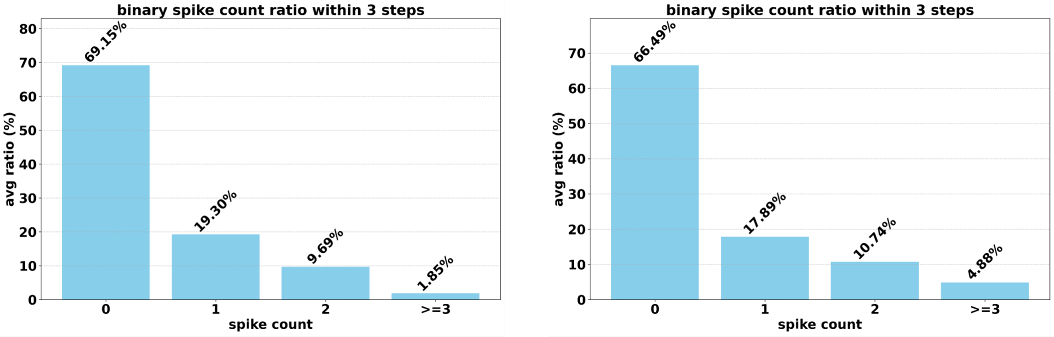


Figure 8: **Spike counts distribution of the bitwise spike coding scheme.** Results are shown for SpikingBrain-7B (left) and SpikingBrain-76B (right).

Spike visualization interface We provide a unified visualization interface to examine the spiking activity of neurons under different coding strategies. As shown in Figure 9, we compare Binary, Ternary, and Bitwise Spike Coding using two-dimensional time–neuron firing maps. This tool provides an intuitive means for analyzing spiking behaviors across different spike coding strategies:

- **Binary Spike Coding** typically requires longer time steps and higher firing frequencies, resulting in a relatively dense overall spike distribution.
- **Ternary Spike Coding** introduces positive and negative spikes, yielding a significantly sparser representation that reduces both the total spike count and the required time steps.
- **Bitwise Spike Coding** further enhances compression through bit-expansion strategies, substantially reducing time overhead while preserving representational precision.

Energy Efficiency with Asynchronous Hardware Statistical analysis shows that during inference, 18.4% of channels remain completely inactive (i.e., emit no spikes). Under the event-driven computing paradigm of asynchronous hardware, all weight-fetching operations for these inactive channels are skipped (including data transfer from off-chip DRAM to on-chip SRAM and from SRAM to compute units), thereby proportionally reducing memory access overhead.

For computation, we estimate and compare the energy consumption of MAC (multiply–accumulate) operations under different paradigms, based on published hardware energy consumption data at 45nm technology [40]:

- **Conventional FP16 MAC**: Each operation consumes 1.5 pJ, requiring full floating-point multiplication and addition regardless of whether the activation is effective.
- **INT8 MAC**: Each operation consumes 0.23 pJ, but still depends on synchronous clock-driven execution, incurring energy costs for idle cycles.

- **Proposed scheme (event-driven spiking computing + INT8 weight quantization)**: The MAC energy consumption E is estimated based on "spike-triggered weight additions", i.e.,

  $$E = \text{Average Spikes} \times E_{\text{INT8 Add}}. \tag{27}$$

  With an average spike count of 1.13 and INT8 (weight) addition energy consumption of approximately 0.03 pJ per operation, the average MAC energy cost is about 0.034 pJ (1.13 × 0.03 pJ).

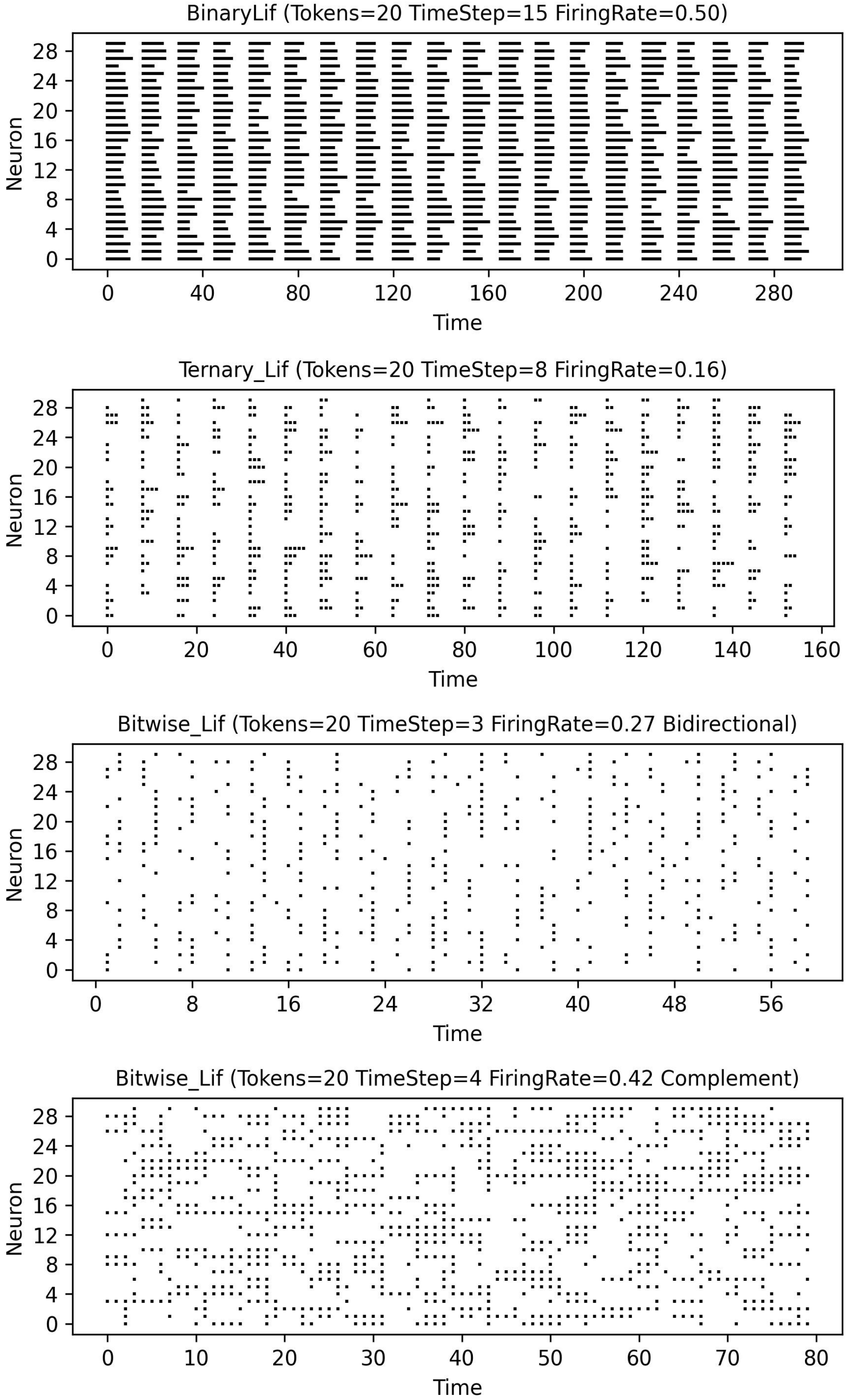
The results show that our scheme reduces energy consumption by 97.7% compared with FP16 MACs and by 85.2% compared with INT8 MACs, yielding energy-efficiency improvements of up to 43.48× and 6.76×, respectively. This demonstrates the effectiveness of combining spiking-driven computation with quantization to significantly reduce energy overhead, while maintaining precision degradation within a controllable range.

6 Conclusion

This work provides a comprehensive demonstration of efficient brain-inspired large model training on the MetaX GPU cluster. By integrating several key techniques—including novel (hybrid) linear architectures beyond Transformers, sparse MoE design, lightweight conversion training, and adaptive-threshold sparse spiking activation—we validate and implement a practical development pipeline for spiking-based large models on a non-NVIDIA cluster with hundreds of GPUs.

We release two models as outcomes of this effort: the linear model SpikingBrain-7B and the MoE hybrid-linear model SpikingBrain-76B-A12B. These models offer two main advantages: i) Training efficiency: With linear or near-linear complexity, they substantially accelerate long-sequence training and match the performance of many open-source Transformer models while using less than 2% of the training data. ii) Inference performance: During inference, the models exhibit event-driven spiking behavior and maintain constant (SpikingBrain-7B) or layer-wise partially constant (SpikingBrain-76B) memory footprint. This yields order-of-magnitude improvements in long-sequence efficiency and speed (e.g., over 100× TTFT acceleration at 4M tokens).

These results not only demonstrate the feasibility of efficient large-model training on non-NVIDIA platforms, but also outline new directions for the scalable deployment and application of brain-inspired models in future computing systems.



BinaryLif (Tokens=20 TimeStep=15 FiringRate=0.50)

Ternary_Lif (Tokens=20 TimeStep=8 FiringRate=0.16)

Bitwise_Lif (Tokens=20 TimeStep=3 FiringRate=0.27 Bidirectional)

Bitwise_Lif (Tokens=20 TimeStep=4 FiringRate=0.42 Complement)

Figure 9: **Time–neuron firing maps under different spike coding schemes.** The figure shows the spike firing distributions for the same input under different coding strategies, including Binary, Ternary, and two variants of Bitwise spike coding. The horizontal axis represents time (Time), defined as token timesteps × expanded timesteps; the vertical axis represents neuron index (Neuron). Black dots indicate spike events of the corresponding neuron at that time.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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

[3] OpenAI. GPT-5. `https://openai.com/gpt-5/`, 2025.

[4] Google DeepMind. Gemini 2.5 Pro. `https://deepmind.google/models/gemini/pro/`, 2025.

[5] Anthropic. Introducing Claude Opus 4.1. `https://www.anthropic.com/news/claude-opus-4-1`, 2025.

[6] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[7] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[8] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

[9] Linxuan He, Yunhui Xu, Weihua He, Yihan Lin, Yang Tian, Yujie Wu, Wenhui Wang, Ziyang Zhang, Junwei Han, Yonghong Tian, et al. Network model with internal complexity bridges artificial intelligence and neuroscience. *Nature Computational Science*, 4(8):584–599, 2024.

[10] Jungo Kasai, Hao Peng, Yizhe Zhang, Dani Yogatama, Gabriel Ilharco, Nikolaos Pappas, Yi Mao, Weizhu Chen, and Noah A Smith. Finetuning pretrained transformers into rnns. In *2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021*, pages 10630–10643. Association for Computational Linguistics (ACL), 2021.

[11] Ethan He, Abhinav Khattar, Ryan Prenger, Vijay Korthikanti, Zijie Yan, Tong Liu, Shiqing Fan, Ashwath Aithal, Mohammad Shoeybi, and Bryan Catanzaro. Upcycling large language models into mixture of experts. *arXiv preprint arXiv:2410.07524*, 2024.

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

[13] Catherine D Schuman, Shruti R Kulkarni, Maryam Parsa, J Parker Mitchell, Prasanna Date, and Bill Kay. Opportunities for neuromorphic computing algorithms and applications. *Nature Computational Science*, 2(1):10–19, 2022.

[14] Charlotte Frenkel, David Bol, and Giacomo Indiveri. Bottom-up and top-down approaches for the design of neuromorphic processing systems: Tradeoffs and synergies between natural and artificial intelligence. *Proceedings of the IEEE*, 111(6):623–652, 2023.

[15] Albert Gu. On the tradeoffs of state space models and transformers, 2025.

[16] John P O'Doherty, Sang Wan Lee, Reza Tadayonnejad, Jeff Cockburn, Kyo Iigaya, and Caroline J Charpentier. Why and how the brain weights contributions from a mixture of experts. *Neuroscience & Biobehavioral Reviews*, 123:14–23, 2021.

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

[18] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[19] Albert Qiaochu Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *ArXiv*, abs/2310.06825, 2023.

[20] Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *International Conference on Machine Learning*, pages 56501–56523. PMLR, 2024.

[21] Tri Dao and Albert Gu. Transformers are ssms: generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning*, pages 10041–10071, 2024.

[22] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

[23] Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, et al. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.

[24] Liliang Ren, Yang Liu, Yadong Lu, Chen Liang, Weizhu Chen, et al. Samba: Simple hybrid state space models for efficient unlimited context language modeling. In *The Thirteenth International Conference on Learning Representations*, 2024.

[25] Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024.

[26] Xin Dong, Yonggan Fu, Shizhe Diao, Wonmin Byeon, Zijia Chen, Ameya Sunil Mahabaleshwarkar, Shih-Yang Liu, Matthijs Van Keirsbilck, Min-Hung Chen, Yoshi Suhara, et al. Hymba: A hybrid-head architecture for small language models. *arXiv preprint arXiv:2411.13676*, 2024.

[27] Jingwei Zuo, Maksim Velikanov, Ilyas Chahed, Younes Belkada, Dhia Eddine Rhayem, Guillaume Kunsch, Hakim Hacid, Hamza Yous, Brahim Farhat, Ibrahim Khadraoui, et al. Falcon-h1: A family of hybrid-head language models redefining efficiency and performance. *arXiv preprint arXiv:2507.22448*, 2025.

[28] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

[29] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

[30] Damai Dai, Chengqi Deng, Chenggang Zhao, Rx Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1280–1297, 2024.

[31] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Ke-Yang Chen, Kexin Yang, Mei Li, Min Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yunyang Wan, Yunfei Chu, Zeyu Cui, Zhenru Zhang, and Zhi-Wei Fan. Qwen2 technical report. *ArXiv*, abs/2407.10671, 2024.

[32] Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. Sparse upcycling: Training mixture-of-experts from dense checkpoints. *arXiv preprint arXiv:2212.05055*, 2022.

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

[34] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500, 1952.

[35] Man Yao, JiaKui Hu, Zhaokun Zhou, Li Yuan, Yonghong Tian, Bo Xu, and Guoqi Li. Spike-driven transformer. In *Advances in Neural Information Processing Systems*, volume 36, pages 64043–64058, 2023.

[36] Man Yao, JiaKui Hu, Tianxiang Hu, Yifan Xu, Zhaokun Zhou, Yonghong Tian, Bo XU, and Guoqi Li. Spike-driven transformer v2: Meta spiking neural network architecture inspiring the design of next-generation neuromorphic chips. In *The Twelfth International Conference on Learning Representations*, 2024.

[37] Xinhao Luo, Man Yao, Yuhong Chou, Bo Xu, and Guoqi Li. Integer-valued training and spike-driven inference spiking neural network for high-performance and energy-efficient object detection. In *European Conference on Computer Vision*, pages 253–272. Springer, 2025.

[38] Man Yao, Xuerui Qiu, Tianxiang Hu, Jiakui Hu, Yuhong Chou, Keyu Tian, Jianxing Liao, Luziwei Leng, Bo Xu, and Guoqi Li. Scaling spike-driven transformer with efficient spike firing approximation training. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(4):2973–2990, 2025.

[39] Liangwei Fan, Hui Shen, Xiangkai Lian, Yulin Li, Man Yao, Guoqi Li, and Dewen Hu. A multisynaptic spiking neuron for simultaneously encoding spatiotemporal dynamics. *Nature Communications*, 16(1):7155, 2025.

[40] Man Yao, Ole Richter, Guangshe Zhao, Ning Qiao, Yannan Xing, Dingheng Wang, Tianxiang Hu, Wei Fang, Tugba Demirci, Michele De Marchi, Lei Deng, Tianyi Yan, Carsten Nielsen, Sadique Sheik, Chenxi Wu, Yonghong Tian, Bo Xu, and Guoqi Li. Spike-based dynamic computing with asynchronous sensing-computing neuromorphic chip. *Nature Communications*, 15(1):4464, 2024.

[41] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

[42] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*, 2024.

[43] Yuhong Chou, Man Yao, Kexin Wang, Yuqi Pan, Ruijie Zhu, Yiran Zhong, Qiao Yu, Jibin Wu, Bo Xu, and Guoqi Li. Metala: Unified optimal linear approximation to softmax attention map. In *The Thirty-Eighth Annual Conference on Neural Information Processing Systems*. Neural information processing systems foundation, 2024.

[44] Zhen Qin, Songlin Yang, Weixuan Sun, Xuyang Shen, Dong Li, Weigao Sun, and Yiran Zhong. Hgrn2: Gated linear rnns with state expansion. In *First Conference on Language Modeling*, 2024.

[45] Guoqi Li, Lei Deng, Huajin Tang, Gang Pan, Yonghong Tian, Kaushik Roy, and Wolfgang Maass. Brain-inspired computing: A systematic survey and future trends. *Proceedings of the IEEE*, 112(6):544–584, 2024.

[46] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

[47] Beidi Chen, Tri Dao, Eric Winsor, Zhao Song, Atri Rudra, and Christopher Ré. Scatterbrain: Unifying sparse and low-rank attention. *Advances in Neural Information Processing Systems*, 34:17413–17426, 2021.

[48] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[49] Jyotikrishna Dass, Shang Wu, Huihong Shi, Chaojian Li, Zhifan Ye, Zhongfeng Wang, and Yingyan Lin. Vitality: Unifying low-rank and sparse approximation for vision transformer acceleration with a linear taylor attention. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 415–428. IEEE, 2023.

[50] Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, James Zou, Atri Rudra, and Christopher Re. Simple linear attention language models balance the recall-throughput tradeoff. In *International Conference on Machine Learning*, pages 1763–1840. PMLR, 2024.

[51] Jean Mercat, Igor Vasiljevic, Sedrick Keh, Kushal Arora, Achal Dave, Adrien Gaidon, and Thomas Kollar. Linearizing large language models. *arXiv preprint arXiv:2405.06640*, 2024.

[52] Junxiong Wang, Daniele Paliotta, Avner May, Alexander Rush, and Tri Dao. The mamba in the llama: Distilling and accelerating hybrid models. *Advances in Neural Information Processing Systems*, 37:62432–62457, 2024.

[53] Michael Zhang, Simran Arora, Rahul Chalamala, Alan Wu, Benjamin Spector, Aaryan Singhal, Krithik Ramesh, and Christopher Ré. Lolcats: On low-rank linearizing of large language models. *arXiv preprint arXiv:2410.10254*, 2024.

[54] Yu Zhang, Songlin Yang, Rui-Jie Zhu, Yue Zhang, Leyang Cui, Yiqiao Wang, Bolun Wang, Freda Shi, Bailin Wang, Wei Bi, et al. Gated slot attention for efficient linear-time sequence modeling. *Advances in Neural Information Processing Systems*, 37:116870–116898, 2024.

[55] Ge Zhang, Scott Qu, Jiaheng Liu, Chenchen Zhang, Chenghua Lin, Chou Leuang Yu, Danny Pan, Esther Cheng, Jie Liu, Qunshu Lin, Raven Yuan, Tuney Zheng, Wei Pang, Xinrun Du, Yiming Liang, Yinghao Ma, Yizhi Li, Ziyang Ma, Bill Lin, Emmanouil Benetos, Huan Yang, Junting Zhou, Kaijing Ma, Minghao Liu, Morry Niu, Noah Wang, Quehry Que, Ruibo Liu, Sine Liu, Shawn Guo, Soren Gao, Wangchunshu Zhou, Xinyue Zhang, Yizhi Zhou, Yubo Wang, Yuelin Bai, Yuhan Zhang, Yuxiang Zhang, Zenith Wang, Zhenzhu Yang, Zijian Zhao, Jiajun Zhang, Wanli Ouyang, Wenhao Huang, and Wenhu Chen. Map-neo: Highly capable and transparent bilingual large language model series. *arXiv preprint arXiv: 2405.19327*, 2024.

[56] Jijie Li, Li Du, Hanyu Zhao, Bo-wen Zhang, Liangdong Wang, Boyan Gao, Guang Liu, and Yonghua Lin. Infinity instruct: Scaling instruction selection and synthesis to enhance language models. *arXiv preprint arXiv:2506.11116*, 2025.

[57] Cong Liu, Zhong Wang, ShengYu Shen, Jialiang Peng, Xiaoli Zhang, ZhenDong Du, and YaFang Wang. The chinese dataset distilled from deepseek-r1-671b. `https://huggingface.co/datasets/Congliu/Chinese-DeepSeek-R1-Distill-data-110k`, 2025.

[58] Sathwik Tejaswi Madhusudhan, Shruthan Radhakrishna, Jash Mehta, and Toby Liang. Millions scale dataset distilled from r1-32b. https://huggingface.co/datasets/ServiceNow-AI/R1-Distill-SFT, 2025.

[59] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[60] Vijay Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. *arXiv preprint arXiv:2205.05198*, 2022.

[61] NVIDIA. Communication overlap. `https://docs.nvidia.com/nemo-framework/user-guide/latest/nemotoolkit/features/optimizations/communication_overlap.html`, 2025.

[62] Qinlong Wang, Bo Sang, Haitao Zhang, Mingjie Tang, and Ke Zhang. Dlrover: An elastic deep training extension with auto job resource recommendation. *CoRR*, 2023.

[63] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, August 1990.

[64] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[65] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Anand Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm. *arXiv preprint arXiv:2104.04473*, 2021.

[66] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377*, 2018.

[67] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International conference on machine learning*, pages 18332–18346. PMLR, 2022.

[68] Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models. *arXiv preprint arXiv:2309.14509*, 2023.

[69] Weigao Sun, Disen Lan, Yiran Zhong, Xiaoye Qu, and Yu Cheng. Lasp-2: Rethinking sequence parallelism for linear attention and its hybrid. *ArXiv*, abs/2502.07563, 2025.

[70] Yuhong Chou, Zehao Liu, Ruijie Zhu, Xinyi Wan, Tianjian Li, Congying Chu, Qian Liu, Jibin Wu, and Zejun Ma. Zeco: Zero communication overhead sequence parallelism for linear attention. *arXiv preprint arXiv:2507.01004*, 2025.

[71] Shenggui Li, Hongxin Liu, Zhengda Bian, Jiarui Fang, Haichen Huang, Yuliang Liu, Boxiang Wang, and Yang You. Colossal-ai: A unified deep learning system for large-scale parallel training. In *Proceedings of the 52nd International Conference on Parallel Processing*, pages 766–775, 2023.

[72] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

[73] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[74] Paola A Buitrago and Nicholas A Nystrom. Open compass: accelerating the adoption of ai in open research. In *Practice and Experience in Advanced Research Computing 2019: Rise of the Machines (learning)*, pages 1–9. 2019.

[75] Jingwei Zuo, Maksim Velikanov, Dhia Eddine Rhaiem, Ilyas Chahed, Younes Belkada, Guillaume Kunsch, and Hakim Hacid. Falcon mamba: The first competitive attention-free 7b language model. *arXiv preprint arXiv:2410.05355*, 2024.

[76] Mistral AI team. Mistral 7b. https://mistral.ai/news/announcing-mistral-7b, 2023.

[77] Paolo Glorioso, Quentin Anthony, Yury Tokpanov, James Whittington, Jonathan Pilault, Adam Ibrahim, and Beren Millidge. Zamba: A compact 7b ssm hybrid model. *arXiv preprint arXiv:2405.16712*, 2024.

[78] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407, 2024.

[79] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[80] Haonan Li, Yixuan Zhang, Fajri Koto, Yifei Yang, Hai Zhao, Yeyun Gong, Nan Duan, and Timothy Baldwin. Cmmlu: Measuring massive multitask language understanding in chinese. *arXiv preprint arXiv:2306.09212*, 2023.

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

[82] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

[83] Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Yao Fu, et al. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *Advances in Neural Information Processing Systems*, 36:62991–63010, 2023.

[84] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[85] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[86] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[87] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

[88] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

[89] Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

[90] Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, and Samuel Bowman. QuALITY: Question answering with long input texts, yes! In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5336–5358, Seattle, United States, July 2022. Association for Computational Linguistics.

[91] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

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