SCHTs: A Semi-Structured Dynamic Sparse Training Framework for Hardware-Efficient Deep Learning

Yuan Hua[1, 2, †] **Zhaoxu Ding**[1, †] **Jilin Zhang**[1] **Ziyi Cheng**[1] **Zhijie Jian**[1]

Wenqi Gu[3, 5] **Carlo Vittorio Cannistraci**[3, 4, 5] **Hong Chen**[1, *]

[1] School of Integrated Circuits, Tsinghua University, Beijing, China
[2] Peng Cheng Laboratory, Shenzhen, China
[3] Center for Complex Network Intelligence at the Tsinghua Laboratory of Brain and Intelligence, Department of Psychological and Cognitive Sciences
[4] Department of Computer Science and Technology, Tsinghua University, Beijing, China
[5] School of Biomedical Engineering, Tsinghua Medicine, Tsinghua University, Beijing, China

Abstract

Although Dynamic Sparse Training (DST) has emerged as a promising solution for deploying deep neural networks on resource-constrained edge devices, unstructured DST methods suffer from irregular weight distributions that incur extra index overhead and severely degrade computational efficiency. In contrast, semi-structured sparsity offers computational efficiency but typically suffers from performance degradation at high sparsity levels with strict semi-structured constraints. To address these challenges, we propose Semi-structured Cannistraci-Hebb Training soft rule (SCHTs), a novel hardware-efficient dynamic sparse training framework, which includes four key stages: semi-structured sparse topological initialization, weight initialization, global network pruning, and local network regrowth. Unlike unstructured DST methods, SCHTs supports fine-grained and coarse-grained N:M semi-structured sparsity patterns with constant fan-in or fan-out, ensuring computational efficiency. Extensive experiments demonstrate that the proposed SCHTs framework enables highly sparse networks (linear-layer sparsity > 90%) to achieve performance comparable to, or even surpassing, their dense counterparts. **(1) For spiking neural networks**, models trained with SCHTs outperform their dense counterparts at 90% hidden linear-layer sparsity, yielding accuracy improvements of +0.15% and +2.67% on CIFAR-10 and CIFAR-100. **(2) For artificial neural networks**, sparse networks trained with SCHTs outperform their dense counterparts with accuracy improvements of +0.21% and +0.49% using GoogLeNet on CIFAR-100 and TinyImageNet, as well as a +0.30% improvement using ResNet-152 on CIFAR-100. **(3) For large language models**, SCHTs enables a 90% sparse LLaMA-130M to yield a highly competitive validation perplexity of 25.49 on OpenWebText. Furthermore, on a representative LLaMA-130M MLP projection measured on an NVIDIA A100 GPU, SCHTs achieves a $1.24\times$ speedup over dense cuBLAS and a $19.85\times$ speedup over equal-sparsity CSR using NVIDIA's native FP16 2:4 Tensor-Core operator. With optimized FP32 kernels, SCHTs achieves speedups of up to $3.54\times$ and $3.91\times$ over dense execution under fine-grained and coarse-grained sparsity, respectively. Our code is available at https://github.com/ddddazhaxie/SCHTs.

[†]These authors contributed equally to this work.
[*]Corresponding author, hongchen@tsinghua.edu.cn

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

As the scale of Deep Neural Networks (DNNs) continues to expand, their computational and memory demands surge [1]. In resource-constrained edge computing scenarios, such as edge devices or Application-Specific Integrated Circuits (ASICs), deploying massive AI models poses significant challenges [2]. To balance model performance and lightweight design, sparse training has gained widespread attention. It effectively reduces network parameters and computational overhead [3].

Two common approaches to obtaining sparse neural networks are pruning-based methods and Dynamic Sparse Training (DST). In a conventional post-training pruning pipeline, a dense network is first trained and then pruned, optionally followed by fine-tuning of the resulting sparse network. In contrast, DST adopts a "*sparse-to-sparse*" training scheme. It maintains sparsity throughout the entire training and inference cycle [4]. By utilizing a dynamic "prune-and-regrow" topological evolution cycle, DST achieves performance comparable to dense networks with minimal training overhead [5–8].

However, despite achieving high theoretical sparsity and reducing floating point operations (FLOPs), existing DST methods (e.g., RigL [5], CHT [6], and CHTs [7]) primarily generate unstructured sparse topologies. This unstructured nature creates a gap between algorithmic gains and actual hardware performance [9]. From a hardware architecture perspective, the irregular distribution of non-zero weights necessitates complex indexing mechanisms, which impose extra memory overhead [10]. Furthermore, when mapped onto parallel processing arrays, this irregularity leads to severe load imbalance across Processing Elements (PEs) [10, 11]. Since the execution time of a parallel workload is bottlenecked by the most heavily loaded PE, such imbalance results in low global utilization and diminished computational efficiency (e.g., FLOPs/cycle).

Compared to unstructured sparsity, semi-structured sparsity patterns (such as N:M sparsity [12]) demonstrate significant computational efficiency. Semi-structured sparsity imposes regular connectivity constraints on the network topology (e.g., restricting a constant fan-in or fan-out count), which effectively reduces hardware index storage requirements, simplifies internal data routing logic [13, 14], and ensures strict load balancing in parallel computing arrays. However, semi-structured sparse training faces a challenge during the dynamic evolution of DST. Directly imposing strict semi-structured topological constraints during training severely restricts the network's capacity for adaptive structural optimization. Consequently, this leads to significant model performance degradation under high sparsity levels. To address the challenge, this paper proposes a semi-structured dynamic sparse training framework, which maintains performance comparable to dense networks even under high semi-structured sparsity levels. Our contributions are as follows:

- We propose a semi-structured dynamic sparse training method for both Artificial Neural Networks (ANNs) and Spiking Neural Networks (SNNs), named Semi-structured Cannistraci-Hebb Training soft rule (SCHTs). This framework integrates semi-structured sparse topological initialization, weight initialization, global network pruning, and local network regrowth. SCHTs enforces either a constant fan-in or a constant fan-out constraint. It also implements N:M semi-structured sparsity configurations ranging from coarse-grained to fine-grained.
- We introduce a Semi-structured Sparse Topological Initialization (SSTI) method. SSTI calculates the correlation coefficients between input features and directly constructs an initial sparse topology satisfying semi-structured constraints before training. This method improves the final performance of SCHTs on MLPs.
- We design a novel topological update mechanism combining "global pruning and local regrowth". The global pruning step realizes node and block removal during each topology iteration to remove redundant structures and enhance the model's representation capacity in fine-grained N:M sparsity mode. Meanwhile, the local regrowth strategy guarantees that the newly generated weights strictly adhere to the semi-structured sparsity pattern after every "prune-and-regrow" cycle.
- We conduct extensive experiments across diverse paradigms, including SNNs, ANNs, and large language models (LLMs), on benchmark datasets such as CIFAR-10, CIFAR-100, TinyImageNet, and OpenWebText. SCHTs demonstrates superior scalability for various architectures. Crucially, we benchmark a representative LLaMA-130M linear operator on an NVIDIA A100 GPU. NVIDIA's native FP16 2:4 path achieves a $1.24\times$ speedup over dense execution and a $19.85\times$ speedup over CSR. Optimized FP32 kernels achieve up to $3.54\times$

and $3.91\times$ speedups over dense execution for fine- and coarse-grained sparsity, respectively. We further evaluate end-to-end FP32 LLaMA inference, where SCHTs-Hybrid consistently accelerates inference from 1:4 sparsity onward and achieves up to a $2.68\times$ speedup over dense execution.

2 Related Works

2.1 Unstructured Dynamic Sparse Training

Unlike network pruning, which typically removes weights after training a dense network, Dynamic Sparse Training (DST) adopts a "*sparse-to-sparse*" paradigm. It maintains sparsity in both the forward and backward propagation. DST dynamically optimizes the network topology alongside the weights via continuous "prune-and-regrow" cycles. Currently, most DST methods [5–7] generate unstructured sparse topologies, achieving performance comparable to dense networks through extensive exploration of the topological space.

Unstructured DST in artificial neural networks (ANNs). Numerous studies have investigated unstructured DST applicable for ANNs, focusing on the impact of various pruning and regrowth strategies on model performance. For example, Sparse Evolutionary Training (SET) [4] utilizes random regrowth after removing weights with the smallest magnitudes. This proves that stochastic topology search could approximate dense model performance. Similarly, RigL [5] removes weights with the smallest magnitudes but incorporates gradient information during regrowth to identify high-saliency missing weights. Deep Rewiring (Deep R) [15] follows the random regrowth strategy of SET. It prunes weights whose signs change during the update process.

Unlike the aforementioned methods relying on randomness or gradients, Epitopological Learning (EL) [6] offers a novel perspective. EL posits that existing topological information is sufficient to predict potential optimal connections in complex networks. Consequently, it completely eliminates the dependence on gradient information. Based on this, Cannistraci-Hebb Training (CHT) [6] utilizes Relative Importance (RI) scores and Weight Magnitude (WM) as pruning criteria. During the regrowth phase, it employs the CH3-L3 path-based predictor from the Cannistraci-Hebb theory [16–20] to regrow weights. CHT achieves performance surpassing fully connected networks even at extreme sparsity levels up to 99%. Furthermore, CHT soft rule (CHTs) [7] builds upon CHT by introducing soft link removal and regrowth strategies. This mechanism effectively prevents the model from falling into epitopological local minima.

Unstructured DST in spiking neural networks (SNNs). Unlike ANNs, SNNs use sparse discrete spikes to transmit information and employ specific neurons like Leaky Integrate-and-Fire (LIF) [21] models. The spike signals enable SNNs to avoid multiply-accumulate (MAC) operations. However, the non-differentiable nature of the spiking activation function prevents the direct application of gradient-dependent DST algorithms from ANNs. Therefore, researchers have proposed dedicated dynamic DST methods for SNNs. Shen et al. [22] propose a two-stage method. The first stage evaluates the network using the PQ index to determine pruning and regrowth ratios. The second stage then executes topological updates based on these ratios. Spiking Neural Networks with Connection Gates (SNN-CG) [23] alternately optimize weights and network topology. This process continues until a predefined convergence condition is met. Building upon Deep Rewiring, Gradient Rewiring (Grad R) [24] introduces constraints on weight gradients. This mechanism enables the regrowth of previously pruned weights. Recently, CH-SNN [8] extended the CHTs framework with the topology and weight initialization methods specifically for SNNs. It also introduced a novel network pruning score. Even under extreme sparsity levels exceeding 90%, CH-SNN achieves performance surpassing fully connected networks.

2.2 N:M Semi-Structured Sparsity

The N:M semi-structured sparsity pattern imposes strict constraints on the network topology. It requires exactly N non-zero elements within every block of M consecutive weights [12]. This regular sparsity pattern demonstrates hardware friendliness. Therefore, extensive research has focused on exploring semi-structured sparse training methods. These studies aim to directly acquire network topologies that satisfy the N:M constraint.

One of the most representative methods is the Sparse-Refined Straight-Through Estimator (SR-STE) [25], which employs a magnitude-based pruning strategy during forward propagation and introduces Straight-Through Estimator (STE) [26] for backward propagation. Specifically, SR-STE maintains a dense weight matrix W in memory throughout the training process. During the forward pass, it projects W onto the target N:M sparsity pattern for computation. During the backward pass, it ignores the non-differentiability of the projection function and directly updates the dense W using gradients from the sparse network. Additionally, SR-STE incorporates an extra regularization term to penalize pruned weights and maintain performance. However, SR-STE essentially follows a "*dense-to-sparse*" training paradigm. Since it must store the dense weights and calculate dense gradients throughout the training cycle, it incurs massive storage and computational overhead.

To alleviate the aforementioned computational overhead, Lasby et al. proposed Structured RigL (SRigL) [13]. This is a novel "*sparse-to-sparse*" dynamic sparse training method based on RigL [5]. Through dynamic evolution, SRigL learns a sparse network topology with a constant fan-in constraint. It maintains performance comparable to fully connected networks even at high sparsity levels (80%). However, the topology generated by SRigL does not strictly adhere to fine-grained N:M semi-structured constraints. Standard N:M sparsity strictly requires exactly N non-zero elements within M consecutive physical locations. However, the constant fan-in constraint of SRigL only restricts the total number of non-zero elements. It does not enforce positional continuity. Therefore, it can only be considered a coarse-grained semi-structured constraint. It fails to support fine-grained N:M semi-structured sparsity patterns, such as $2:4$. Recently, Lyu et al. proposed CHTs24 [27], which integrates Cannistraci-Hebb Training with N:M semi-structured sparsity and demonstrates the feasibility of hardware-efficient dynamic sparse training. However, CHTs24 mainly focuses on transposable 2:4 sparsity, while a more general framework supporting diverse N:M sparsity patterns and flexible structural constraints remains unexplored.

3 Methods

To achieve a high level of semi-structured sparsity, we incorporate semi-structured sparse constraints into the Cannistraci-Hebb Training soft rule (CHTs) [7] and propose a four-step dynamic sparse training method, named the Semi-structured Cannistraci-Hebb Training soft rule (SCHTs), including semi-structured sparse topological initialization, weight initialization, global network pruning, and local network regrowth. SCHTs is a semi-structured dynamic sparse training framework for linear layers in both SNNs and ANNs, featuring capabilities for N:M sparsity and supporting either constant fan-in or constant fan-out modes. Before detailing our proposed method, we first establish several key definitions. To facilitate a better understanding, an illustrative example is provided in Figure 1.

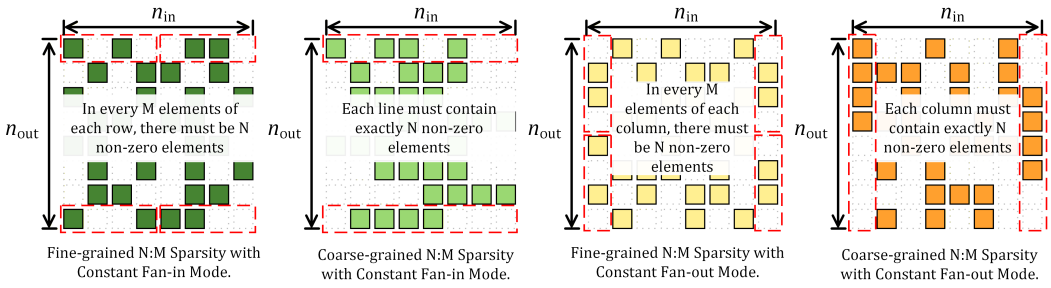

Figure 1: Comparison of four semi-structured sparsity configurations supported by SCHTs.

Definition 1. Consider a linear layer with a weight matrix $\mathbf{W} \in \mathbb{R}^{n_{\text{out}} \times n_{\text{in}}}$. We define a binary adjacency matrix $\mathbf{A} \in \mathbb{R}^{n_{\text{out}} \times n_{\text{in}}}$ to represent the connectivity, where $A_{i,j} = 1$ indicates an active connection. We define two distinct modes of semi-structured sparsity constraints:

Constant Fan-in Mode. The N:M semi-structured constraint is applied row-wise to $\mathbf{A}$. Specifically, for each row i, every block of M consecutive elements must contain exactly N non-zero entries.

Constant Fan-out Mode. The N:M semi-structured constraint is applied column-wise to $\mathbf{A}$. Specifically, for each column j, every block of M consecutive elements must contain exactly N non-zero entries.

Definition 2. We categorize the N:M semi-structured sparsity into two categories:

Fine-grained N:M Sparsity. This mode refers to configurations where M is a small value (e.g., 2:4, 1:8, 1:16, 1:32). The sparsity constraint is enforced locally within each disjoint block of M elements.

Coarse-grained N:M Sparsity. In the constant fan-in mode, it is defined by setting $M = n_{\text{in}}$ and $N = \lfloor \rho \cdot n_{\text{in}} \rfloor$, where $\rho \in (0, 1)$ denotes the target connectivity density. This ensures a global count of N non-zero elements per row. In the constant fan-out mode, it is defined by setting $M = n_{\text{out}}$ and $N = \lfloor \rho \cdot n_{\text{out}} \rfloor$, ensuring a global count of N non-zero elements per column.

3.1 Semi-structured Sparse Topological Initialization

To improve the performance of semi-structured dynamic sparse training, we propose Semi-structured Sparse Topological Initialization (SSTI) to initialize the topology of linear layers that interact directly with input features. As illustrated in Figure 2, SSTI primarily consists of four steps.

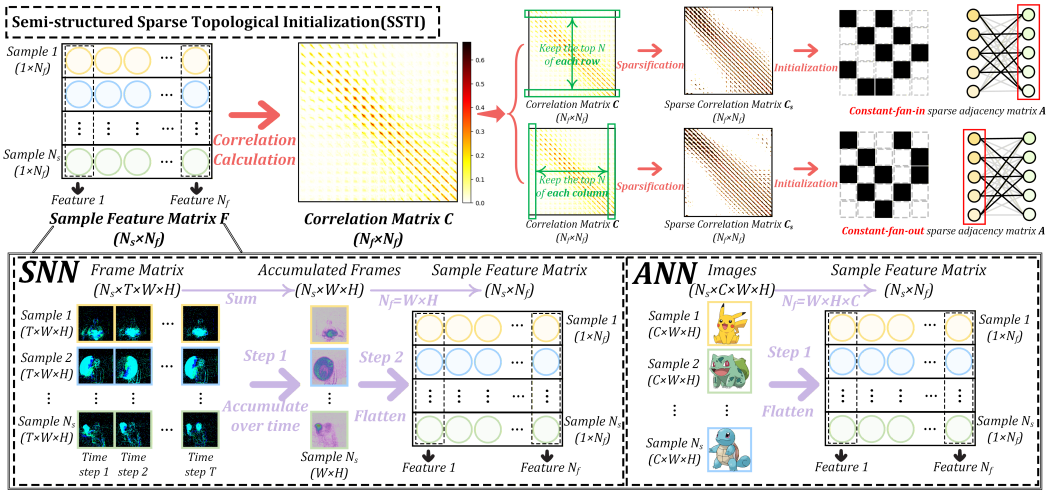

Figure 2: Illustration of SSTI. The boxes show the extraction methods of sample feature matrices for SNN and ANN datasets. The top pipeline demonstrates how the semi-structured network topology is initialized using correlation coefficients calculated from these matrices.

1) Obtaining the sample feature matrix. For ANNs, we employ the vectorization method from CSTI [6] to obtain the sample feature matrix, which flattens each sample $x_i \in \mathbb{R}^{C \times W \times H}$ to get the sample feature matrix $\mathbf{F} \in \mathbb{R}^{N_s \times N_f}$, where $N_f = C \times W \times H$ is the number of features and N_s is the total number of training samples. For SNNs, we first accumulate each sample $x_i \in \mathbb{R}^{T \times W \times H}$ along the time dimension to obtain the accumulated frames and then flatten these accumulated frames to acquire the sample feature matrix $\mathbf{F}$.

2) Calculating the correlation coefficient matrix. After obtaining the sample feature matrix $\mathbf{F}$, we calculate the Pearson correlation for each feature and take the magnitude to form the correlation matrix $\mathbf{C}$. We assume that both the input dimension n_{in} and the output dimension n_{out} of the linear layer to be initialized are equal to N_f. For more general cases, please refer to the Appendix A.

3) N:M Semi-Structured Sparsification. As shown in Figure 2, under coarse-grained N:M sparsity, for the constant fan-in mode, $M = n_{\text{in}}$ and $N = \lfloor n_{\text{in}} \times (1 - \text{sparsity}) \rfloor$. We process the matrix row by row, retaining N elements in each row and setting the remaining elements to 0. For the constant fan-out mode, $M = n_{\text{out}}$ and $N = \lfloor n_{\text{out}} \times (1 - \text{sparsity}) \rfloor$. We process the matrix column by column, retaining N elements in each column. For fine-grained N:M sparsity, the top-N largest elements are strictly selected from every M-element block along either the row (constant fan-in) or column (constant fan-out) dimension. This process yields the sparse correlation matrix $\mathbf{C_s}$. Because the correlation matrix $\mathbf{C}$ is symmetric, the sparse correlation matrices obtained from the constant fan-in and constant fan-out modes are transposes of each other.

4) Generating the adjacency matrix. We set the non-zero elements in the sparse correlation matrix to 1 and keep the remaining elements unchanged. This generates the adjacency matrix $\mathbf{A}$ and completes the topological initialization.

3.2 Weight Initialization

Numerous well-established methods exist for initializing weights in sparse networks. For ANNs, we employ sparse weight initialization (SWI) [6], while for SNNs, we utilize sparse spike weight initialization (SSWI) [8]. For the detailed mathematical formulations, please refer to Appendix B.

Although SWI and SSWI are primarily applied to unstructured dynamic sparse training, they remain applicable to semi-structured dynamic sparse training. For fixed weight matrix dimensions, the standard deviation of SWI depends only on the network sparsity S_s, as shown in Equation 1. Similarly, the variance of SSWI incorporates network sparsity S_s alongside dataset-dependent temporal sparsity S_t, as shown in Equation 2. Since these factors remain consistent between unstructured and semi-structured sparsity, we can directly adopt these initialization methods.

3.3 Global Network Pruning

Global links pruning. We calculate the importance scores and perform pruning globally, specifically adopting the soft link removal method from CHTs [7]. The importance score s_{ij} is calculated by Relative Importance (RI). Subsequently, we perform probabilistic pruning by sampling the links to retain rather than directly sampling the links to remove. This ensures that the pruning process is not rigidly dictated solely by s_{ij}. Instead, it allows weights with high importance scores to be retained with a high probability, which helps the network escape epitopological local minima. For a detailed description of the global network pruning process and the corresponding formulations, please refer to Appendix C.

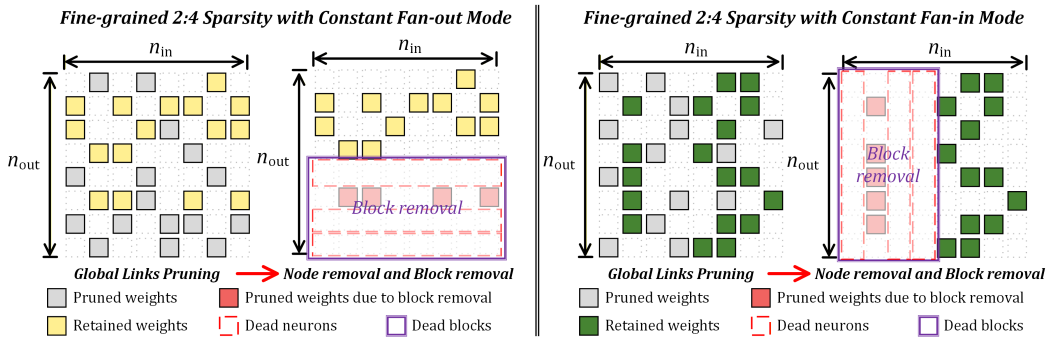

Figure 3: Schematic diagram detailing the processes of global links pruning, node removal, and block removal in fine-grained 2:4 sparsity. The left panel shows the constant fan-out mode, while the right panel displays the constant fan-in mode.

Node removal and block removal. After the global links pruning, we proceed with node removal. We define neurons that lack connections on one or both sides as "dead neurons." Because these dead neurons cannot regrow new weights, we remove them directly. Furthermore, for semi-structured sparsity, we execute block removal. Under fine-grained $N : M$ sparsity, if the number of "dead neurons" within an $N : M$ block exceeds $M - N$, the block can no longer support the required N links. We define such blocks as "dead blocks." In this scenario, we remove all M neurons in "dead blocks", to strictly enforce the semi-structured sparsity constraints. By implementing both node removal and block removal, we guarantee effective information transmission throughout the network while successfully eliminating redundant nodes. For clarity, Figure 3 provides examples of constant fan-in and constant fan-out modes for fine-grained 2:4 sparsity, illustrating the processes of global links pruning, node removal, and block removal.

3.4 Local Network Regrowth

The global network pruning method makes the network temporarily exhibit unstructured sparsity. However, our local network regrowth strategy regrows new weights to satisfy the semi-structured sparsity constraints. Our method adopts a global computation and local regrowth paradigm. We globally compute regrowth scores using a CH3-L3 path-based predictor (please refer to Appendix D for details). This strategy allows the model to comprehensively learn the global direction for topological optimization before identifying the optimal weights for local regrowth. Specifically, after computing the regrowth scores, our local regrowth process proceeds as follows.

Coarse-grained N:M semi-structured sparsity. For the constant fan-in mode, we perform regrowth strictly along each row. If a row contains fewer than N non-zero weights, we regrow weights until exactly N weights are present. Rows already containing N weights are simply skipped. Similarly, for the constant fan-out mode, this process is exclusively applied along the column dimension.

Fine-grained N:M semi-structured sparsity. For the constant fan-in mode, we process every block of M consecutive parameters within each row, ensuring that exactly N non-zero weights exist per block. Blocks with fewer than N weights are replenished through regrowth, while blocks already possessing N weights are bypassed. Similarly, for the constant fan-out mode, this block-wise processing is exclusively applied along the column dimension.

During the local network regrowth, we also apply the soft rule mechanism. This procedure uses an analogous stochastic sampling procedure, except that importance scores are replaced with regrowth scores. The complete regrowth process is summarized in Appendix E.

4 Experiments

To evaluate the performance of SCHTs at the algorithmic level, we conduct extensive experiments across both ANNs and SNNs. For SNNs, we benchmark the Spiking Convolutional Neural Network (SCNN), Multi-Layer Perceptrons (MLPs), and Spikformer [28] architectures on the CIFAR-10 [29], CIFAR-100 [29], N-MNIST [30], and MNIST [31] datasets. For ANNs, we utilize MLPs on the Fashion-MNIST [32], MNIST and EMNIST [33] datasets. Furthermore, we evaluate deeper architectures by applying SCHTs to the linear layers (except the output layer) of ResNet-152 [34] and GoogLeNet [35] on TinyImageNet [36] and CIFAR-100. For GoogLeNet and ResNet-152, we replace the original classification layer with three hidden fully connected layers followed by a dense output layer. All compared methods use these modified architectures, and DST is applied only to the three hidden linear layers. Additionally, we extend our empirical studies to LLMs, experimenting with LLaMA-60M and LLaMA-130M [37] on the OpenWebText [38] dataset. Detailed model settings and ablations are provided in Appendix F and H. For hardware evaluation, we benchmark SCHTs-compatible sparse operators on an NVIDIA A100 GPU against matched dense and unstructured CSR implementations. Appendix I reports end-to-end LLaMA inference results.

4.1 SCHTs on Spiking Neural Networks

Baseline. We benchmark SCHTs against existing SNN sparse training methods (SD-SNN [39], DPAP [40], and CH-SNN [8]) under identical network configurations. While Table 1 reports results for coarse-grained N:M sparsity, the fine-grained counterparts and detailed setups are provided in Appendix G.1.2 and F.1. Furthermore, SCHTs is extended to SNN-based MLPs and Spikformer architectures, with comprehensive evaluations detailed in Appendix G.1.1 and G.1.3.

Main results. As shown in Table 1, SCHTs demonstrates remarkable resilience under strict semi-structured constraints. While unstructured baselines like SD-SNN and DPAP suffer from significant link sparsity degradation to maintain accuracy (e.g., dropping to 35.57% on CIFAR-10), SCHTs sustains high link sparsity (over 70% on CIFAR datasets and 89.75% on N-MNIST). Remarkably, under the rigid constant fan-in constraint, SCHTs achieves an accuracy of 94.89% on CIFAR-10, outperforming all unstructured counterparts. On the more challenging CIFAR-100 dataset, SCHTs trades a negligible accuracy margin for rigorous hardware compliance, presenting a superior algorithm-hardware co-design solution.

Table 1: Performance comparison of different methods on CIFAR-10, CIFAR-100, and N-MNIST. Bold indicates SCHTs, and the best results among DST methods are underlined. * DPAP results and their corresponding dense reference accuracies are taken directly from the original paper. All other methods are reproduced within a unified framework. SCHTs adopts a coarse-grained N:M sparsity pattern with a target hidden linear-layer sparsity of 90%. SCHTs accuracies are reported as mean ± standard deviation (SD) across three random seeds. Uncertainty estimates are not reported for DPAP, SD-SNN, or CH-SNN.

Dataset	Method	Sparse Pattern	Link sparsity	Accuracy	Accuracy improvement
CIFAR-10	DPAP*	Unstructured	50.80%	93.83%	–0.71%
	SD-SNN	Unstructured	35.57%	94.59%	–0.15%
	CH-SNN	Unstructured	74.62%	94.60%	–0.14%
	SCHTs	**Constant Fan-In**	**70.73%**	**94.89±0.09%**	**+0.15%**
	SCHTs	**Constant Fan-Out**	**70.73%**	**94.72±0.16%**	**–0.02%**
CIFAR-100	SD-SNN	Unstructured	36.94%	75.33%	+3.27%
	CH-SNN	Unstructured	74.45%	75.22%	+3.16%
	SCHTs	**Constant Fan-In**	**70.58%**	**74.67±0.55%**	**+2.61%**
	SCHTs	**Constant Fan-Out**	**70.58%**	**74.73±0.31%**	**+2.67%**
N-MNIST	DPAP*	Unstructured	63.95%	99.59%	+0.06%
	SD-SNN	Unstructured	58.62%	98.78%	–0.29%
	CH-SNN	Unstructured	94.73%	99.15%	+0.08%
	SCHTs	**Constant Fan-In**	**89.75%**	**99.08±0.12%**	**+0.01%**
	SCHTs	**Constant Fan-Out**	**89.75%**	**99.04±0.09%**	**–0.03%**

4.2 SCHTs on Artificial Neural Networks

Baseline. We benchmark SCHTs against DST methods (SET [4], CHT [6], and CHTs [7]) using GoogLeNet and ResNet-152 on CIFAR-100 and TinyImageNet. While Table 2 reports coarse-grained N:M sparsity results, the fine-grained counterparts, MLP evaluations, and detailed configurations are deferred to Appendix G.2.2, G.2.1, and F.2, respectively.

Table 2: Performance comparison of different methods on CIFAR-100 and TinyImageNet. Bold indicates our method, and the best results of DST methods are underlined. FC denotes the dense network. * indicates unstructured dynamic sparse training methods. For this experiment, SCHTs adopts a coarse-grained N:M sparsity pattern, and the sparsity level for all evaluated dynamic sparse training methods is fixed at 90%. SCHTs_{in} and $\text{SCHTs}_{\text{out}}$ represent the results under constant fan-in and fan-out, respectively. SCHTs accuracies are reported as mean ± standard deviation (SD) across three random seeds. Uncertainty estimates are not reported for FC or the other DST baselines.

Method	GoogLeNet CIFAR-100	GoogLeNet TinyImageNet	ResNet-152 CIFAR-100	ResNet-152 TinyImageNet
FC	76.64%	63.24%	78.30%	68.05%
SET*	76.93%	62.58%	78.59%	66.57%
CHT*	77.52%	63.53%	79.82%	67.71%
CHTs*	77.41%	62.64%	79.72%	67.76%
SCHTs_{in}	**76.82±0.64%**	**63.58±0.31%**	**78.49±0.36%**	**67.25±0.77%**
$\text{SCHTs}_{\text{out}}$	**76.85±0.49%**	**63.73±0.15%**	**78.60±0.60%**	**67.23±0.37%**

Main Results. As shown in Table 2, SCHTs surpasses the dense counterparts, yielding accuracy improvements of +0.21% (to 76.85%) and +0.49% (to 63.73%) using GoogLeNet on CIFAR-100 and TinyImageNet, and a +0.30% improvement (to 78.60%) using ResNet-152 on CIFAR-100. Furthermore, SCHTs exhibits highly competitive performance against unstructured methods (SET, CHT, CHTs), achieving the highest accuracies on TinyImageNet using GoogLeNet and maintaining robust convergence on the complex ResNet-152/TinyImageNet task (67.25%, closely approaching the dense baseline of 68.05%). Finally, the performance of constant fan-in and fan-out modes provides critical topological flexibility for hardware-algorithm co-design, allowing optimal sparsity patterns to be tailored to the specific data-reuse preferences of parallel accelerators without sacrificing accuracy.

4.3 SCHTs on Large Language Models

To demonstrate the scalability and broad applicability of SCHTs, we extend our evaluation to LLMs, specifically training LLaMA-60M and LLaMA-130M architectures on the OpenWebText corpus. For specific experimental settings, please refer to Appendix F.3.

Table 3: OpenWebText validation perplexity for LLaMA variants (60M and 130M) trained with different DST methods. FC denotes the dense baseline. * indicates unstructured dynamic sparse training methods. SCHTs adopts a coarse-grained N:M sparsity pattern. SCHTs results are highlighted in bold. For each model and sparsity level, the lowest perplexity among the evaluated DST methods is underlined. SCHTs_{in} and $\text{SCHTs}_{\text{out}}$ denote constant fan-in and constant fan-out, respectively. SCHTs perplexities are reported as mean $\pm$ standard deviation (SD) across three random seeds. Uncertainty estimates are not reported for FC or the other DST baselines.

Method	LLaMA-60M				LLaMA-130M			
	70%	80%	90%	95%	70%	80%	90%	95%
FC	26.56				19.27			
SET*	31.77	30.69	35.26	39.70	20.82	22.02	24.73	28.37
RigL*	39.96	41.33	45.34	51.49	25.85	66.35	37.18	49.39
CHT*	31.02	32.99	35.01	41.87	21.02	22.82	26.27	30.01
CHTs*	<u>28.12</u>	<u>29.84</u>	<u>33.03</u>	<u>36.47</u>	<u>20.10</u>	<u>21.33</u>	<u>23.71</u>	<u>26.45</u>
SCHTs_{in}	**29.76±0.45**	**32.05±0.31**	**35.77±0.35**	**41.55±0.96**	**21.52±0.61**	**24.09±1.22**	**25.49±0.30**	**29.61±0.21**
$\text{SCHTs}_{\text{out}}$	**29.85±0.43**	**32.08±1.26**	**35.99±0.18**	**41.37±0.59**	**21.32±0.22**	**24.21±1.17**	**25.61±0.35**	**29.48±0.40**

As shown in Table 3, SCHTs demonstrates remarkable efficiency in LLM pre-training, achieving an optimal balance between algorithmic performance and hardware-friendly execution. While the unstructured CHTs method attains the lowest validation perplexity, SCHTs delivers highly competitive results despite operating under strict hardware constraints (constant fan-in/fan-out). For instance, on LLaMA-130M at 70% sparsity, SCHTs yields a perplexity of 21.32, narrowly trailing the unstructured CHTs (20.10) while ensuring hardware deployability. Notably, SCHTs entirely avoids the severe instability observed in RigL, whose perplexity spikes catastrophically at 80% sparsity. Furthermore, SCHTs maintains robust performance as the model scales from 60M to 130M parameters and under 95% sparsity, demonstrating the framework's adaptability for LLMs.

4.4 Hardware Efficiency Evaluation

To assess the hardware efficiency of SCHTs, we benchmarked a representative LLaMA-130M operator on a single NVIDIA A100 80GB PCIe GPU. Specifically, we extracted the final Transformer block's MLP down-projection ($W \in \mathbb{R}^{768\times2048}$) and measured $Y = XW^T$ at batch size 512, without bias or activation. SCHTs and the dense baseline use the same five independently trained SCHTs weight matrices. For the dense baseline, each SCHTs weight matrix is stored in dense format and executed using the corresponding dense operator, without changing its values. The unstructured CSR baseline uses five trained sparse weight matrices obtained using CHTs. Inputs are randomly generated. For each matrix, we perform 50 warm-up calls, followed by 200 CUDA-event measurements, each averaging ten consecutive calls. Effective throughput is calculated as $(2\times512\times2048\times768)/Latency$ and reported in dense-equivalent TFLOP/s.

Table 4: Operator-level evaluation of NVIDIA's native FP16 2:4 Tensor-Core path on an A100 80 GB PCIe GPU. Values are mean $\pm$ standard error of the mean (SEM) across five trained matrices. Throughput is dense-equivalent.

Metric	SCHTs 2:4	Dense cuBLAS	Unstructured CSR
Latency (μs)	<u>16.794 ± 0.020</u>	20.872 ± 0.021	333.281 ± 0.031
Throughput (TFLOP/s)	<u>95.906 ± 0.114</u>	77.169 ± 0.078	4.833 ± 0.0004

We evaluate NVIDIA's native cuSPARSELt 2:4 Tensor-Core operator with FP16 inputs and outputs and FP32 accumulation. As shown in Table 4, SCHTs reduces latency from 20.872 ± 0.021 μs to 16.794 ± 0.020 μs relative to dense cuBLAS. This corresponds to a 19.5% latency reduction and a $1.24\times$ speedup. Compared with equal-sparsity cuSPARSE CSR, SCHTs is $19.85\times$ faster.

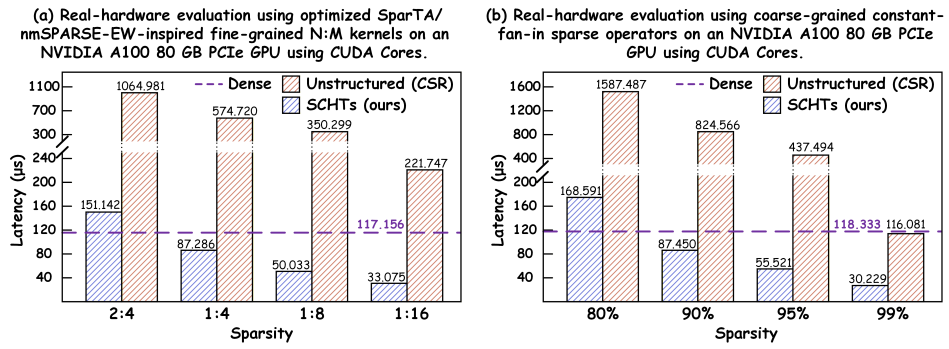

Figure 4: Operator-level latency on an NVIDIA A100 80 GB PCIe GPU. Lower is better. (a) FP32 fine-grained N:M kernels inspired by SparTA/nmSPARSE-EW [41, 42]. (b) FP32 coarse-grained constant-fan-in kernels following the execution approach of SRigL [13]. Dense baselines use cuBLAS with TF32 disabled, and unstructured baselines use equal-sparsity CSR. Values are means over five trained matrices; all relative SEMs are below 1%.

Figure 4 evaluates the custom FP32 CUDA-Core paths and shows that the realized speedup depends on both sparsity and kernel support. For fine-grained sparsity, the 1:4, 1:8, and 1:16 kernels achieve $1.34\times$, $2.34\times$, and $3.54\times$ speedups over dense cuBLAS, respectively. They outperform equal-sparsity CSR by $6.58\times$–$7.05\times$. Unlike the native FP16 path in Table 4, the custom FP32 2:4 kernel is 29.0% slower than dense execution. This contrast shows that numerical precision and specialized Tensor-Core support can determine whether moderate sparsity is beneficial. For coarse-grained constant fan-in, SCHTs achieves $1.35\times$, $2.13\times$, and $3.91\times$ speedups over dense execution at 90%, 95%, and 99% sparsity, respectively. The 80% configuration is 42.5% slower than dense, while all four settings outperform CSR by $3.84\times$–$9.43\times$.

The timed region includes on-device arithmetic, memory access, metadata decoding, indexing, and scheduling, but excludes data transfer, allocation, format conversion, and setup. These results establish operator-level inference acceleration on the evaluated A100 kernels, not end-to-end training acceleration. To further determine whether the operator-level gains translate into whole-model acceleration, we conducted end-to-end FP32 LLaMA inference experiments. At 1:16 sparsity, SCHTs-Hybrid achieves $1.71\times$, $1.97\times$, and $2.68\times$ speedups for LLaMA-60M, LLaMA-130M, and LLaMA-1B, respectively. Detailed results are provided in Appendix I.

5 Conclusion

In this paper, we propose SCHTs, a semi-structured DST framework for linear layers. SCHTs combines SSTI with global pruning and local regrowth to maintain either constant fan-in or constant fan-out N:M topology. Across SNNs, ANNs, and LLMs, the resulting sparse models retain competitive predictive performance under high semi-structured sparsity (linear-layer sparsity >90%). On an NVIDIA A100 GPU, SCHTs achieves a $1.24\times$ speedup over dense cuBLAS and a $19.85\times$ speedup over equal-sparsity CSR using NVIDIA's native FP16 2:4 Tensor-Core operator, as well as speedups of up to $3.54\times$ and $3.91\times$ with optimized FP32 fine- and coarse-grained kernels. Furthermore, end-to-end FP32 LLaMA inference achieves up to a $2.68\times$ speedup. SCHTs therefore offers a controllable accuracy–efficiency trade-off for linear-layer workloads, rather than universal acceleration for every model or sparsity configuration. Finally, we describe the limitations of this paper and future work in Appendix J.

Acknowledgments and Disclosure of Funding

This work is supported by National Natural Science Foundation of China (No. 62574117 and 62334014). The National Natural Science Foundation of China grant number W2531064 to C.V.C. The Zhou Yahui Chair Professorship award of Tsinghua University to C.V.C. The National High-Level Talent Program of the Ministry of Science and Technology of China grant number 20241710001 to C.V.C. The authors declare no competing interests.

References

[1] Ben Cottier. Trends in the dollar training cost of machine learning systems, 2023. URL https://epoch.ai/blog/trends-in-the-dollar-training-cost-of-machine-learning-systems. Accessed: 2026-05-01.

[2] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1): 82–99, 2018. doi: 10.1109/MM.2018.112130359.

[3] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021. URL http://jmlr.org/papers/v22/21-0366.html.

[4] Decebal Mocanu, Elena Mocanu, Peter Stone, Phuong Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 9, 06 2018. doi: 10.1038/s41467-018-04316-3.

[5] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2943–2952. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/evci20a.html.

[6] Yingtao Zhang, Jialin Zhao, Wenjing Wu, Alessandro Muscoloni, and Carlo Vittorio Cannistraci. Epitopological learning and cannistraci-hebb network shape intelligence brain-inspired theory for ultra-sparse advantage in deep learning. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Learning Representations*, volume 2024, pages 46211–46239, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/c9ef471a579197c4ed99df2aa542ce97-Paper-Conference.pdf.

[7] Yingtao Zhang, Diego Cerretti, Jialin Zhao, Wenjing Wu, Ziheng Liao, Umberto Michieli, and Carlo Vittorio Cannistraci. Brain network science modelling of sparse neural networks enables transformers and llms to perform as fully connected. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 48241–48277. Curran Associates, Inc., 2025. doi: 10.52202/085713-1610. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/44f9b7b8a39e67eca18600ec6d2c303c-Paper-Conference.pdf.

[8] Yuan Hua, Jilin Zhang, Yingtao Zhang, Leyi You, Baobo Xiong, Carlo Vittorio Cannistraci, and Hong Chen. Cannistraci-hebb training on ultra-sparse spiking neural networks. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust, editors, *International Conference on Learning Representations*, volume 2026, pages 114755–114779, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/baa7fc022f35b6ea7b8b2a2fe60babe0-Paper-Conference.pdf.

[9] Shail Dave, Riyadh Baghdadi, Tony Nowatzki, Sasikanth Avancha, Aviral Shrivastava, and Baoxin Li. Hardware acceleration of sparse and irregular tensor computations of ml models: A survey and insights. *Proceedings of the IEEE*, 109(10):1706–1752, 2021. doi: 10.1109/JPROC.2021.3098483.

[10] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. Eie: Efficient inference engine on compressed deep neural network, 2016. URL https://arxiv.org/abs/1602.01528.

[11] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. Scnn: An

accelerator for compressed-sparse convolutional neural networks. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 27–40, 2017. doi: 10.1145/3079856.3080254.

[12] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks, 2021. URL https://arxiv.org/abs/2104.08378.

[13] Mike Lasby, Anna Golubeva, Utku Evci, Mihai Nica, and Yani Ioannou. Dynamic sparse training with structured sparsity. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Learning Representations*, volume 2024, pages 32982–33010, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/8c5f30296296d2ae402ebbd09aaa9c12-Paper-Conference.pdf.

[14] Chao Fang, Aojun Zhou, and Zhongfeng Wang. An algorithm–hardware co-optimized framework for accelerating n:m sparse transformers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 30(11):1573–1586, 2022. doi: 10.1109/TVLSI.2022.3197282.

[15] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks, 2018. URL https://arxiv.org/abs/1711.05136.

[16] Carlo Cannistraci, G. Alanis-Lobato, and Timothy Ravasi. From link-prediction in brain connectomes and protein interactomes to the local-community-paradigm in complex networks. *Scientific reports*, 3(1), April 2013. ISSN 2045-2322. doi: 10.1038/srep01613. KAUST Repository Item: Exported on 2020-10-01.

[17] Simone Daminelli, Josephine Maria Thomas, Claudio Durán, and Carlo Vittorio Cannistraci. Common neighbours and the local-community-paradigm for topological link prediction in bipartite networks. *New Journal of Physics*, 17(11):113037, November 2015. ISSN 1367-2630. doi: 10.1088/1367-2630/17/11/113037. URL http://dx.doi.org/10.1088/1367-2630/17/11/113037.

[18] Carlo Cannistraci. Modelling self-organization in complex networks via a brain-inspired network automata theory improves link reliability in protein interactomes. *Scientific Reports*, 8, 10 2018. doi: 10.1038/s41598-018-33576-8.

[19] Alessandro Muscoloni, Ilyes Abdelhamid, and Carlo Vittorio Cannistraci. Local-community network automata modelling based on length-three-paths for prediction of complex network structures in protein interactomes, food webs and more. *bioRxiv*, 2018. doi: 10.1101/346916. URL https://doi.org/10.1101/346916.

[20] Jialin Zhao, Alessandro Muscoloni, Umberto Michieli, Yingtao Zhang, and Carlo Vittorio Cannistraci. Adaptive cannistraci-hebb network automata modelling of complex networks for path-based link prediction. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 40253–40288. Curran Associates, Inc., 2025. doi: 10.52202/085713-1344. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/396c99af13c9a2e867172bea3e6b6d42-Paper-Conference.pdf.

[21] L.F Abbott. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain Research Bulletin*, 50(5):303–304, 1999. ISSN 0361-9230. doi: https://doi.org/10.1016/S0361-9230(99)00161-6. URL https://www.sciencedirect.com/science/article/pii/S0361923099001616.

[22] Jiangrong Shen, Qi Xu, Gang Pan, and Badong Chen. Improving the sparse structure learning of spiking neural networks from the view of compression efficiency. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, editors, *International Conference on Learning Representations*, volume 2025, pages 92834–92846, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/e7fd2c0a1a6f956c94024e955b34cc43-Paper-Conference.pdf.

[23] Yu Qi, Jiangrong Shen, Yueming Wang, Huajin Tang, Hang Yu, Zhaohui Wu, and Gang Pan. Jointly learning network connections and link weights in spiking neural networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, page 1597–1603. AAAI Press, 2018. ISBN 9780999241127.

[24] Yanqi Chen, Zhaofei Yu, Wei Fang, Tiejun Huang, and Yonghong Tian. Pruning of deep spiking neural networks through gradient rewiring. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, IJCAI-2021, page 1713–1721. International Joint Conferences on Artificial Intelligence Organization, August 2021. doi: 10.24963/ijcai.2021/236. URL http://dx.doi.org/10.24963/ijcai.2021/236.

[25] Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning n:m fine-grained structured sparse neural networks from scratch, 2021. URL https://arxiv.org/abs/2102.04010.

[26] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. URL https://arxiv.org/abs/1308.3432.

[27] Jiaqing Lyu, Ruijie Wang, Kangyou Bao, Yingtao Zhang, and Carlo Vittorio Cannistraci. Cannistraci-hebb training with n:m semi-structured sparsity for pre-training and re-training. In Rebekka Burkholz, Shiwei Liu, Saiprasad Ravishankar, William Redman, Wei Huang, Weijie Su, and Zhihui Zhu, editors, *Conference on Parsimony and Learning*, volume 328 of *Proceedings of Machine Learning Research*, pages 192–217. PMLR, 23–26 Mar 2026. URL https://proceedings.mlr.press/v328/lyu26a.html.

[28] Zhaokun Zhou, Yuesheng Zhu, Chao He, Yaowei Wang, Shuicheng Yan, Yonghong Tian, and Li Yuan. Spikformer: When spiking neural network meets transformer, 2022. URL https://arxiv.org/abs/2209.15425.

[29] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[30] Garrick Orchard, Ajinkya Jayawant, Gregory Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades, 2015. URL https://arxiv.org/abs/1507.07629.

[31] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[32] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. URL https://arxiv.org/abs/1708.07747.

[33] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926, 2017. doi: 10.1109/IJCNN.2017.7966217.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

[35] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015. doi: 10.1109/CVPR.2015.7298594.

[36] Ya Le and Xuan Yang. Tiny ImageNet visual recognition challenge. Cs231n course project report, Stanford University, 2015. URL https://cs231n.stanford.edu/reports/2015/pdfs/yle_project.pdf.

[37] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

[38] Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

[39] Bing Han, Feifei Zhao, Wenxuan Pan, and Yi Zeng. Adaptive sparse structure development with pruning and regeneration for spiking neural networks. *Information Sciences*, 689:121481, 2025. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2024.121481. URL https://www.sciencedirect.com/science/article/pii/S0020025524013951.

[40] Bing Han, Feifei Zhao, Yi Zeng, and Guobin Shen. Developmental plasticity-inspired adaptive pruning for deep spiking and artificial neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(1):240–251, 2025. doi: 10.1109/TPAMI.2024.3467268.

[41] Ningxin Zheng, Bin Lin, Quanlu Zhang, Lingxiao Ma, Yuqing Yang, Fan Yang, Yang Wang, Mao Yang, and Lidong Zhou. Sparta: Deep-learning model sparsity via tensor-with-sparsity-attribute. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 213–232, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-28-1. URL https://www.usenix.org/conference/osdi22/presentation/zheng-ningxin.

[42] Bin Lin, Ningxin Zheng, Lei Wang, Shijie Cao, Lingxiao Ma, Quanlu Zhang, Yi Zhu, Ting Cao, Jilong Xue, Yuqing Yang, and Fan Yang. Efficient gpu kernels for n:m-sparse weights in deep learning. In D. Song, M. Carbin, and T. Chen, editors, *Proceedings of Machine Learning and Systems*, volume 5, pages 513–525. Curran, 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/file/a10deb4d5227a8ea307ea8ff3cb712f4-Paper-mlsys2023.pdf.

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015. URL https://arxiv.org/abs/1502.01852.

[44] Tomaso Poggio. On efficiently computable functions, deep networks and sparse compositionality, 2025. URL https://arxiv.org/abs/2510.11942.

[45] Levi Rauchwerger, Stefanie Jegelka, and Ron Levie. Dense neural networks are not universal approximators, 2026. URL https://arxiv.org/abs/2602.07618.

A Semi-Structured Sparse Topological Initialization (SSTI)

A.1 General Cases for SSTI Initialization

In the main text, we introduced the workflow of SSTI. After computing the correlation matrix in Step 2, we assumed a special case where $n_{\text{in}} = n_{\text{out}} = N_f$. Here, we discuss more general scenarios:

(1) $n_{\text{in}} < N_f$ and $n_{\text{out}} < N_f$: To perform N:M sparsification in this case, we simply truncate the correlation matrix C. We extract the first n_{out} rows and the first n_{in} columns of C to serve as the actual correlation matrix. Subsequently, executing Steps 3 and 4 of SSTI completes the network topology initialization.

(2) $n_{\text{in}} > N_f$ and $n_{\text{out}} > N_f$: In this scenario, we calculate two integers: $\alpha = \lceil n_{\text{in}}/N_f \rceil$ and $\beta = \lceil n_{\text{out}}/N_f \rceil$. Next, we duplicate and expand matrix C into $C_{\text{extended}} \in \mathbb{R}^{\beta N_f \times \alpha N_f}$. We then extract the first n_{out} rows and the first n_{in} columns from C_{extended} to act as the actual correlation matrix. Finally, proceeding with Steps 3 and 4 of SSTI completes the initialization.

(3) For other cases: We simply duplicate and expand C along a single dimension to form C_{extended}. The initialization is then completed using the identical truncation and execution process described above.

A.2 SSTI for Non-MLP Architectures

SSTI can be utilized to initialize linear layers that interact directly with raw input features. However, for non-MLP architectures, such as Convolutional Neural Networks (CNNs) and Transformers, the input features of linear layers are heavily influenced by preceding convolutional or attention layers. As training progresses, the correlations between these input features change dynamically. Consequently, computing a static correlation matrix to initialize the network topology of these linear layers becomes infeasible.

To address this issue in such architectures, our strategy is to randomly generate a matrix to serve as the proxy correlation matrix. We then proceed with Steps 3 and 4 of SSTI to complete the topological initialization. We adopt this specific setting for all non-MLP experiments presented in this paper.

B Sparse Weight Initialization and Sparse Spike Weight Initialization

For ANNs, various weight initialization methods exist, with Kaiming initialization [43] being a prominent example. Its core principle is to maintain the variance of input features constant across layers. While Kaiming initialization is primarily designed for dense networks, Sparse Weight Initialization (SWI) incorporates network sparsity into the Kaiming framework, making it suitable for the weight initialization of sparse ANNs. The formulation of SWI is given in Equation 1.

$$\text{SWI}(w) \sim \mathcal{N}(0, \sigma^2), \quad \sigma = \sqrt{\text{Var}[w]} = \frac{\sqrt{2}}{\sqrt{(1 - S_s) \cdot n_{\text{in}}}} \tag{1}$$

where S_s is the network sparsity and n_{in} is the input dimension of the linear layer.

For SNNs, the information transmission mechanism (discrete spike trains) and activation functions are fundamentally distinct. To address these unique characteristics, Sparse Spike Weight Initialization (SSWI) builds upon the SWI method. SSWI incorporates the temporal sparsity of discrete spike trains and the firing threshold of Leaky Integrate-and-Fire (LIF) neurons, tailoring it specifically for the weight initialization of sparse SNNs. The detailed formulation of SSWI is presented in Equation 2.

$$SSWI(w^{(l)}) \sim \mathcal{N}(0, \sigma^2), \quad \sigma^2 = \begin{cases} \dfrac{S_t}{n_{\text{in}}(1 - S_s)}, & (l = 1) \\ \dfrac{\theta^2 \sqrt{\pi}}{\sqrt{2} e^{-1/2} n_{\text{in}}(1 - S_s)}, & (1 < l < L) \\ \dfrac{\theta^2 \sqrt{\pi}}{\sqrt{2} e^{-1/2} n_{\text{in}}}, & (l = L) \end{cases} \tag{2}$$

where S_t is the temporal sparsity of the spiking dataset, l is the index of the layer (with a total of L layers), n_{in} is the input dimension of the linear layer, and θ denotes the spike threshold in the LIF neuron.

C Global Network Pruning

We perform probabilistic pruning by sampling the links to retain rather than directly sampling the links to remove. Let $W \in \mathbb{R}^{m \times n}$ denote the weight matrix of a layer with n input neurons and m output neurons, and let $\mathbf{A} \in \{0, 1\}^{m \times n}$ denote its binary connectivity mask. Specifically, $A_{ij} = 1$ indicates that the connection from input neuron j to output neuron i is present, whereas $A_{ij} = 0$ indicates that it is absent. The effective weight matrix is $\mathbf{A} \odot W$, where $\odot$ denotes element-wise multiplication. Accordingly, $\mathcal{E} = \{(i, j) : A_{ij} = 1\}$ is the set of existing links before pruning. For each existing link, we compute an importance score based on its weight magnitude relative to the weights in the same row and column:

$$s_{ij} = \frac{|W_{ij}|}{\sum_a (|W_{aj}|) + \epsilon} + \frac{|W_{ij}|}{\sum_b (|W_{ib}|) + \epsilon}, \tag{3}$$

where $\epsilon = 10^{-5}$ ensures nonzero denominators. We then define the normalized sampling weights as

$$q_{ij} = \frac{s_{ij}^{\tau_e}}{\sum_{(a,b) \in \mathcal{E}} s_{ab}^{\tau_e}}, \qquad \tau_e = 1 + \frac{2e}{E_{\text{total}}}, \tag{4}$$

where e is the zero-based epoch index and E_{total} is the total number of training epochs. Increasing τ_e strengthens the preference for retaining high-scoring links. Given a pruning ratio ζ, we select $K = |\mathcal{E}| - \lfloor \zeta |\mathcal{E}| \rfloor$ links from $\mathcal{E}$ using weighted sampling without replacement, with sampling weights q_{ij}. The selected links form the retained set $\mathcal{E}_{\text{keep}}$, and all remaining links are pruned. The connectivity mask after link removal is therefore

$$A'_{ij} = \begin{cases} 1, & (i, j) \in \mathcal{E}_{\text{keep}}, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Here, q_{ij} specifies a sampling weight, not the marginal probability that a link is retained or removed. Consequently, pruning is not implemented as an independent Bernoulli decision for each link.

D CH3-L3 Path-Based Predictor

The CH3-L3 path-based predictor is derived from the Cannistraci-Hebb theory of local-community organization [16–20]. The CH3-L3 path-based predictor is designed to evaluate the potential of regrowing a connection between two currently disconnected nodes, u and v.

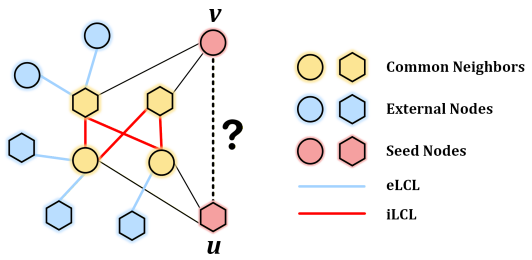

Figure 5: Schematic diagram of the CH3-L3 path-based predictor.

As illustrated in Figure 5, the red nodes represent the disconnected pair u and v for which we calculate the regrowth potential. The yellow nodes, which share direct connections with either u or v, are defined as Common Neighbors. Furthermore, nodes that are connected to the Common Neighbors but remain entirely disconnected from u and v are referred to as External Nodes (depicted in blue). Regarding the topological edges, the red lines indicating connections strictly between Common Neighbors are termed internal Local Community Links (iLCL), whereas the blue lines connecting Common Neighbors with External Nodes are defined as external Local Community Links (eLCL). With these topological concepts established, the precise computation of CH3-L3 is formulated as Equation 6:

$$\textbf{CH3-L3}(u,v) = \sum_{z_1,z_2 \in l3(u,v)} \frac{1}{\sqrt{(1+de_{z_1}) \times (1+de_{z_2})}} \tag{6}$$

In this formulation, u and v serve as the target seed nodes, while z_1 and z_2 identify the intermediate common neighbors that form a length-3 path connecting them. The variables de_{z_1} and de_{z_2} quantify the number of eLCLs incident to z_1 and z_2, respectively. To derive the final CH3-L3 score, we aggregate the evaluations across all possible length-3 paths bridging u and v. Furthermore, a smoothing factor of 1 is added to de_{z_1} and de_{z_2} to avert division-by-zero errors, thereby guaranteeing numerical stability even in the absence of external links.

E Local Network Regrowth

The detailed procedure for local network regrowth is outlined in Algorithm 1. The input mask is obtained after link pruning and neuron removal. For fine-grained N:M sparsity, block removal is additionally performed when the number of dead neurons within a block exceeds $M - N$. An eligibility mask $\mathcal{L}$ indicates positions whose input and output neurons survive and which belong to retained blocks. Each retained block must contain at least N eligible positions and at most N existing connections.

Regrowth prioritizes eligible missing connections with positive CH3-L3 scores. If these candidates are sufficient, the required connections are sampled without replacement, with probabilities proportional to their scores. Otherwise, all positive-score candidates are activated, and the remaining required connections are sampled uniformly without replacement from the other eligible missing positions. Removed neurons are excluded from both stages. Under the stated feasibility condition, this procedure restores exactly N active connections in each retained block.

Algorithm 1 Local network regrowth based on CH3-L3.

Input: Post-pruning mask $\mathcal{M}$, eligibility mask $\mathcal{L}$, sparsity pattern $N:M$, and target dimension $d \in \{\text{in}, \text{out}\}$

Output: Updated mask $\mathcal{M}_{\text{new}}$

$\mathcal{M}_{\text{new}} \leftarrow \mathcal{M}$
$Z \leftarrow \{(i,j) \mid \mathcal{M}_{ij} = 0,\ \mathcal{L}_{ij} = 1\}$
For each $(i,j) \in Z$, compute $S_{ij} \leftarrow$ CH3-L3(i,j)
for each retained non-overlapping block b of M entries along dimension d **do**
  $Z_b \leftarrow Z \cap b$
  $r_b \leftarrow N - \|\mathcal{M}_{\text{new}}[b]\|_0$
  assert $0 \le r_b \le |Z_b|$
  $Z_b^+ \leftarrow \{(i,j) \in Z_b \mid S_{ij} > 0\}$
  while $r_b > 0$ **and** $Z_b^+ \neq \emptyset$ **do**
    For each $(u,v) \in Z_b^+$, set $p_{uv} \leftarrow S_{uv} / \sum_{(a,c)\in Z_b^+} S_{ac}$
    Sample one position $(x,y) \in Z_b^+$ according to p_{xy}
    $\mathcal{M}_{\text{new},x,y} \leftarrow 1$
    $r_b \leftarrow r_b - 1$
    $Z_b \leftarrow Z_b \setminus \{(x,y)\}$
    $Z_b^+ \leftarrow Z_b^+ \setminus \{(x,y)\}$
  end while
  while $r_b > 0$ **do**
    Sample one position (x,y) uniformly from Z_b
    $\mathcal{M}_{\text{new},x,y} \leftarrow 1$
    $Z_b \leftarrow Z_b \setminus \{(x,y)\}$
    $r_b \leftarrow r_b - 1$
  end while
  assert $\|\mathcal{M}_{\text{new}}[b]\|_0 = N$
end for
return $\mathcal{M}_{\text{new}}$

F Experimental Setup

F.1 Experimental Setup for SNNs

In Section 4.1, to evaluate the performance of SCHTs on SNNs, experiments are conducted on Spiking MLPs, Spiking CNNs, and Spikformer architectures. Model training uniformly employs the AdamW optimizer with a base learning rate set to 5×10^{-3}, which is linearly scaled according to $\text{lr} = 5 \times 10^{-3} \times \text{batch size}/1024$. For Spiking MLPs and Spiking CNNs, the total number of training epochs is set to 800, utilizing UnilateralMse(1.0) as the loss function, a weight decay of 0.01, and a cosine learning rate schedule featuring 5 warmup epochs and 10 cooldown epochs. For the non-event datasets (CIFAR-10, CIFAR-100, and MNIST), direct encoding is applied to the inputs.

For Spiking MLPs, a two-layer fully connected network is adopted, structured as $784 \to 1568 \to 10$. The time step of the model is set to 8, and with a batch size of 50, the actual learning rate corresponds to 2.4414×10^{-4}.

For Spiking CNNs, identical convolutional backbone architectures are employed on CIFAR-10 and CIFAR-100: (Input $\to$ [128 Conv 3×3] $\times 2 \to$ MaxPool $2 \times 2 \to$ [256 Conv 3×3] $\times 2 \to$ MaxPool $2 \times 2 \to$ [512 Conv 3×3] $\times 2 \to$ Flatten $\to 512 \times 8 \times 8 \to 512 \to n_{\text{classes}}$). The time step for these datasets is set to 16 with a batch size of 100, resulting in an actual learning rate of 4.8828×10^{-4}. A more lightweight convolutional structure is adopted for MNIST and N-MNIST: (Input $\to$ 15 Conv $3 \times 3 \to$ MaxPool $2 \times 2 \to$ 40 Conv $3 \times 3 \to$ MaxPool $2 \times 2 \to$ Flatten $\to$ $1960/10240 \to 304 \to n_{\text{classes}}$), where the number of input channels is 1 for MNIST and 2 for N-MNIST. The time step for these two sets of experiments is uniformly set to 8 with a batch size of 50, corresponding to an actual learning rate of 2.4414×10^{-4}. For settings employing fine-grained N:M sparsity, the hidden dimension preceding the classification layer must be configured to a value divisible by the block size (e.g., 304).

For the Spikformer model, the total number of training epochs is set to 500, with cross-entropy chosen as the loss function. A weight decay of 1×10^{-4} is applied, and the learning rate follows a cosine schedule featuring 5 warmup epochs. On the CIFAR-10 and CIFAR-100 datasets, the standard

Spikformer is adopted as the backbone network. The input channel count is set to 3, and the time step is set to 4. The network consists of 8 Transformer blocks, with an embedding dimension of 512, 8 attention heads, and an MLP expansion ratio of 4. On the N-MNIST dataset, Spikformer-DVS is employed as the backbone for event data. This configuration includes 2 input channels, 10 classification categories, and a time step of 10, with the input event frame size uniformly resized to 64×64. The network incorporates 2 Transformer blocks, an embedding dimension of 256, 16 attention heads, and an MLP expansion ratio of 4. For the raw event stream, the ToFrame operation is first utilized to convert the event sequence into a frame sequence of T time bins before feeding it into the network for temporal processing.

F.2 Experimental Setup for ANNs

In Section 4.2, experiments are conducted on Multi-Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs). Unless otherwise specified, the training hyperparameters for the different models are configured as follows.

For the MLP models trained on the MNIST, Fashion-MNIST, and EMNIST datasets, the parameters are uniformly set to a learning rate of 0.01, a batch size of 32, 250 training epochs, and a weight decay of 1×10^{-5}. To ensure a fair comparison with other methods, no additional learning rate decay strategy is applied. A four-layer fully connected architecture is employed, consisting of three hidden layers and one output layer. The input images are flattened into 784-dimensional vectors, and the default network structure is $(784 \to 784 \to 784 \to 784 \to n_{\text{classes}})$. All sparse hidden linear layers in the MLP use the same SSTI initialization procedure as the first layer. The correlation matrix computed from the original training inputs serves as an initialization prior for every such layer. This matrix is adapted to each layer's dimensions as described in Appendix A, while the final output layer remains dense.

For the CNN models, including GoogLeNet and ResNet, datasets CIFAR-100 and TinyImageNet are used to evaluate models' performance. The overall training parameters are typically set to a learning rate of 0.1, a batch size of 256, 200 training epochs, and a weight decay of 5×10^{-4}. A MultiStep learning rate scheduling strategy is employed, with milestones set at epochs $[61, 121, 161]$ and a decay factor (γ) of 0.2. It should be noted that, in the ResNet-152 experiments on the TinyImageNet dataset, due to GPU memory constraints, the batch size is reduced from 256 to 128, and the learning rate is correspondingly scaled down from 0.1 to 0.05. Furthermore, the original GoogLeNet and ResNet-152 architectures contain only a single fully connected output layer. This single layer is replaced with four layers in the experiments so that SCHTs can be consistently applied across models. Specifically, the fully connected block is expanded from $(1024 \to n_{\text{classes}})$ to $(1024 \to 1024 \to 1024 \to 1024 \to n_{\text{classes}})$ for GoogLeNet, and from $(2048 \to n_{\text{classes}})$ to $(2048 \to 2048 \to 2048 \to 2048 \to n_{\text{classes}})$ for ResNet-152. Except for the aforementioned adjustments, consistent training parameter settings are maintained across methods and models throughout the remaining experiments.

F.3 Experimental Setup for LLMs

In Section 4.3, two model configurations, LLaMA-60M and LLaMA-130M, are investigated. Both models are optimized using the Adam optimizer with a global batch size of 512, a sequence length of 256, and a base learning rate of 3×10^{-3} (which is adjusted to 1×10^{-3} for fully connected baselines). For the LLaMA-60M variant, the architecture comprises 8 layers and 8 attention heads, featuring an embedding dimension of 512 and a feed-forward dimension of 1376. The training process for this model spans 10,000 steps, incorporating 1,000 warmup steps. In contrast, the LLaMA-130M variant is scaled up to 12 layers and 12 attention heads, with an embedding dimension of 768 and a feed-forward dimension of 2048. Accordingly, the total number of training steps for this larger model is increased to 30,000, with the warmup period extended to 10,000 steps.

F.4 Experimental Setup for DST

The SCHTs framework supports both coarse-grained and fine-grained N:M semi-structured sparsity, with optional constant-fan-in and constant-fan-out modes. Unless otherwise specified, dynamic sparse training is applied exclusively to the hidden linear layers, while the final output classification layer remains dense. To construct the initial network topology, SSTI is utilized, followed by the application

of SWI or SSWI for weight assignment. During the training process, SCHTs simultaneously evaluates Relative Importance (RI) to guide link removal, while a path-based CH3-L3 predictor is deployed to strategically formulate new connections. Building upon this foundation, the soft rule ensures a more robust and comprehensive exploration of the topological search space. For newly grown connections, small random values are assigned to satisfy structural verification requirements and prevent these new weights from vanishing during subsequent gradient updates.

For the ANN experiments, the pruning ratio is uniformly set to $\zeta = 0.1$. Additionally, node removal is enabled in both coarse-grained and fine-grained N:M settings, whereas block removal is applied only in fine-grained settings when the number of dead neurons within a block exceeds $M - N$. The topological structure is updated once per epoch. To ensure a fair comparison with other methods, an early stopping mechanism is employed with a threshold set to 0.9.

For the SNN experiments, no additional early stopping mechanism is utilized. The pruning ratio ζ follows a cosine schedule with an initial value of 0.35. The structural topology is updated every 5 epochs. Specifically, for the Spikformer architecture, sparsity is imposed on the linear attention projections (q, k, v, proj) and the two linear layers (fc1, fc2) within the MLP of each Transformer block, whereas the final classification head remains dense. Since the MLP module in Spikformer consists of two adjacent cascaded hidden linear layers, node removal and block removal are enabled during topological evolution. It should be noted that this mechanism is exclusively applied to the fc1 and fc2 layers within the block and does not affect the q, k, v, and proj linear layers in the attention module.

For LLM experiments, the removal ratio is set to $\zeta = 0.1$, and the sparse topological structure is updated every 100 optimization steps. The learning rate follows a cosine-restart schedule, with the iterative warmup steps configured to 20.

G Additional Experimental Results

G.1 Additional Experiments on SNNs

G.1.1 Experiments on Spiking MLPs

We use a spiking MLP on MNIST and compare SCHTs with DPAP, SD-SNN, and CH-SNN. Since the sparse linear layers interact directly with the input feature vectors, SSTI is used for topology initialization. The experiment focuses on coarse-grained N:M sparsity with constant fan-in and constant fan-out constraints at 90% linear-layer sparsity.

Table 5: Performance comparison of sparse SNN training methods on MNIST using a spiking MLP. Bold indicates SCHTs, and the best results of DST methods are underlined. * The DPAP result is cited from the original paper, whereas SD-SNN, CH-SNN, and SCHTs are reproduced within a unified framework. SCHTs adopts coarse-grained N:M semi-structured sparsity with constant fan-in or constant fan-out.

Dataset	Method	Sparse Pattern	Link sparsity	Accuracy	Accuracy improvement
MNIST	DPAP*	Unstructured	77.36%	98.74%	–0.07%
	SD-SNN	Unstructured	45.86%	98.90%	+0.09%
	CH-SNN	Unstructured	<u>97.75%</u>	<u>98.97%</u>	<u>+0.16%</u>
	SCHTs	**Constant Fan-In**	**88.92%**	**98.93%**	**+0.12%**
	SCHTs	**Constant Fan-Out**	**88.92%**	**98.84%**	**+0.03%**

As shown in Table 5, under the constant fan-in constraint, SCHTs reaches 98.93% accuracy with 88.92% link sparsity, outperforming DPAP and SD-SNN in accuracy while maintaining substantially higher sparsity than SD-SNN. CH-SNN achieves the best accuracy and sparsity among all methods, but it relies on an unstructured topology. SCHTs therefore provides a more deployable alternative: it sacrifices only 0.04% accuracy relative to CH-SNN in the constant fan-in mode while satisfying semi-structured constraints.

We further evaluate fine-grained N:M patterns on the same spiking MLP and MNIST dataset. This experiment tests whether the accuracy advantage of SCHTs remains stable when non-zero weights

must be selected within small local blocks rather than only satisfying a global per-row or per-column count.

Table 6: Accuracy of SCHTs on MNIST using a spiking MLP under fine-grained N:M semi-structured sparsity with **constant fan-in**. The dense FC model is reported as the dense network. The best result of SCHTs is underlined. SCHTs accuracies are reported as mean ± standard deviation (SD) across three random seeds. Uncertainty estimates are not reported for the dense baseline.

Method	2:4	1:4	1:8	1:16
FC		98.81%		
SCHTs	<u>98.99±0.03%</u>	98.93±0.06%	98.65±0.08%	98.38±0.05%

Table 7: Accuracy of SCHTs on MNIST using a spiking MLP under fine-grained N:M semi-structured sparsity with **constant fan-out**. The dense FC model is reported as the dense network. The best result of SCHTs is underlined. SCHTs accuracies are reported as mean ± standard deviation (SD) across three random seeds. Uncertainty estimates are not reported for the dense baseline.

Method	2:4	1:4	1:8	1:16
FC		98.81%		
SCHTs	<u>99.13±0.02%</u>	98.97±0.17%	98.92±0.07%	98.63±0.10%

Tables 6 and 7 show that SCHTs remains accurate even under fine-grained constraints. The 2:4 setting is particularly strong: constant fan-in reaches 98.99%, and constant fan-out further improves to 99.13%, both exceeding the dense baseline of 98.81%. Accuracy gradually decreases as the pattern becomes more sparse, but even at 1:16 SCHTs remains close to the dense model. These results indicate that the local regrowth rule can identify useful connections within strict N:M blocks instead of relying on unconstrained topology search.

G.1.2 Fine-Grained N:M Sparsity Experiments on Spiking CNNs

This experiment complements the coarse-grained SCNN results in the main text by evaluating fine-grained N:M sparsity. We use SCNNs on CIFAR-10, MNIST, and N-MNIST and apply SCHTs to the sparse linear layers under constant fan-in and constant fan-out constraints. Since the linear features are generated by preceding convolutional layers, SSTI is replaced by the proxy-correlation initialization described in Appendix A.2.

Table 8: Accuracy of SCHTs on CIFAR-10, MNIST, and N-MNIST using SCNNs under fine-grained N:M semi-structured sparsity. The dense FC row reports the corresponding dense network for each dataset. SCHTs is evaluated under constant fan-in and constant fan-out constraints with 2:4, 1:4, 1:8, and 1:16 sparsity patterns. "– –" indicates unavailable data due to the post-convolution dimension not being divisible by "M". The best results of SCHTs are underlined. SCHTs accuracies are reported as mean ± standard deviation (SD) across three random seeds. Uncertainty estimates are not reported for the dense baseline.

Method	SCHTs - Constant Fan-In			SCHTs - Constant Fan-Out		
	CIFAR-10	MNIST	N-MNIST	CIFAR-10	MNIST	N-MNIST
FC	94.74%	99.37%	99.07%	94.74%	99.37%	99.07%
2 : 4	<u>94.94±0.24%</u>	99.52±0.04%	<u>99.24±0.08%</u>	<u>94.98±0.16%</u>	<u>99.58±0.02%</u>	<u>99.28±0.07%</u>
1 : 4	94.92±0.21%	<u>99.54±0.05%</u>	99.17±0.06%	94.80±0.09%	99.51±0.02%	99.21±0.01%
1 : 8	94.84±0.08%	99.53±0.02%	99.18±0.07%	94.82 ±0.10%	99.52±0.02%	99.19±0.02%
1 : 16	94.76±0.17%	– –	98.93±0.18%	94.87±0.12%	99.47±0.05%	99.04±0.12%

As shown in Table 8, the fine-grained SCNN results are consistent with the main-text conclusion: SCHTs can satisfy strict semi-structured sparsity while preserving dense-level accuracy. On CIFAR-10, SCHTs produces accuracies above the dense network, with constant fan-out at 2:4 reaching 94.98%. On MNIST, all completed fine-grained settings remain above 99.45%, exceeding the dense

baseline of 99.37%. On N-MNIST, SCHTs also remains robust, and the 2:4 constant fan-out setting reaches 99.28%.

G.1.3 Experiments on Spikformer

This experiment studies whether SCHTs can be extended beyond MLP and SCNNs to a transformer-based spiking architecture. We evaluate Spikformer on CIFAR-10, CIFAR-100, and N-MNIST and compare SCHTs with CH-SNN. The goal is to verify whether semi-structured sparse topology evolution remains effective when the sparse linear layers are embedded in attention-based SNN blocks.

Table 9: Performance comparison between CH-SNN and SCHTs on Spikformer using CIFAR-10, CIFAR-100, and N-MNIST. For this experiment, SCHTs adopts a coarse-grained N:M sparsity pattern, and the sparsity of the linear layer has been set to 90%. For each dataset, the highest link sparsity, accuracy and accuracy improvement are underlined, and SCHTs results are highlighted in bold.

Dataset	Method	Sparse Pattern	Link sparsity	Acc.	Accuracy improvement
CIFAR-10	CH-SNN	unstructured	82.21%	94.26%	–0.10%
	SCHTs	**Semi-structured**	**77.02%**	**94.37%**	**+0.01%**
CIFAR-100	CH-SNN	unstructured	82.11%	76.23%	+0.75%
	SCHTs	**Semi-structured**	**77.79%**	**75.87%**	**+0.39%**
N-MNIST	CH-SNN	unstructured	60.93%	99.65%	+0.07%
	SCHTs	**Semi-structured**	**55.39%**	**99.67%**	**+0.09%**

Table 9 demonstrates that SCHTs can also support Spikformer. On CIFAR-10, SCHTs improves the accuracy from 94.26% to 94.37% and changes the accuracy improvement from -0.10% to +0.01%, although CH-SNN retains a higher unstructured link sparsity. On N-MNIST, SCHTs reaches 99.67%, improving over CH-SNN by 0.02% absolute accuracy and achieving a +0.09% improvement over the dense network. On CIFAR-100, SCHTs maintains a positive accuracy improvement of +0.39% over the dense baseline. Overall, the results indicate that SCHTs can be integrated into Spikformer without losing the core performance benefits observed in simpler SNN architectures.

G.2 Additional Experiments on ANNs

The ANN appendix evaluates SCHTs on both MLPs and CNN backbones. The MLP experiments test the setting where SSTI can directly exploit input-feature correlations. For the CNNs, we conducted evaluations under fine-grained N:M sparsity using both constant fan-in and constant fan-out configurations.

G.2.1 Experiments on MLPs

This experiment evaluates SCHTs on MLPs using MNIST, Fashion-MNIST, and EMNIST. We compare against SET, CHT, and CHTs to determine whether SCHTs preserves the benefits in MLPs. All sparse hidden linear layers use SSTI initialization based on the original training-input correlations, following the procedure described in Appendix F.2.

Table 10 shows that SCHTs is competitive with unstructured DST methods on MLPs. SCHTs achieves the highest reported accuracies among the evaluated DST methods on all three datasets. Its best accuracies are 98.41% on MNIST, 90.58% on Fashion-MNIST, and 84.84% on EMNIST. On Fashion-MNIST, SCHTs also exceeds the dense baseline by 0.15 percentage points. The small gap between constant fan-in and constant fan-out further indicates that SCHTs is not overly sensitive to the chosen structural direction.

We then evaluate the same MLP setting under fine-grained N:M sparsity. This experiment isolates the effect of local N:M constraints on simple ANN classifiers and tests whether performance degrades smoothly as the sparse blocks become more restrictive.

Table 10: Performance comparison of ANN sparse training methods on MNIST, Fashion-MNIST, and EMNIST using MLPs. Bold indicates SCHTs, and the best results among DST methods are underlined. FC denotes the dense baseline. * indicates unstructured dynamic sparse training methods. SCHTs adopts a coarse-grained N:M sparsity pattern. The target hidden linear-layer sparsity is 90% for all evaluated DST methods. SCHTs_{in} and $\text{SCHTs}_{\text{out}}$ denote constant fan-in and constant fan-out, respectively. Accuracies for FC and SCHTs are reported as mean ± standard deviation (SD) across three random seeds. Uncertainty estimates are not reported for SET, CHT, or CHTs.

Method	MNIST	Fashion-MNIST	EMNIST
FC	98.69±0.02%	90.43±0.09%	85.58±0.06%
SET*	98.19%	89.91%	84.80%
CHT*	98.27%	90.27%	84.07%
CHTs*	98.24%	90.38%	84.48%
SCHTs_{in}	**98.39±0.07%**	**90.58±0.27%**	**84.84±0.28%**
$\text{SCHTs}_{\text{out}}$	**98.41±0.11%**	**90.58±0.07%**	**84.58±0.05%**

Table 11: Accuracy of SCHTs on MNIST, Fashion-MNIST, and EMNIST using ANN MLPs under fine-grained N:M semi-structured sparsity. The dense FC row is included as the dense network. SCHTs is evaluated with 2:4, 1:4, 1:8, and 1:16 sparsity patterns under both constant fan-in and constant fan-out constraints. The best results of SCHTs are underlined. Accuracies are reported as mean ± standard deviation (SD) across three random seeds.

Method	SCHTs - Constant Fan-In			SCHTs - Constant Fan-Out		
	MNIST	Fashion-MNIST	EMNIST	MNIST	Fashion-MNIST	EMNIST
FC	98.69±0.02%	90.43±0.09%	85.58±0.06%	98.69±0.02%	90.43±0.09%	85.58±0.06%
2 : 4	98.39±0.07%	90.73±0.10%	84.20±0.06%	98.48±0.02%	90.80±0.20%	84.31±0.16%
1 : 4	98.14±0.04%	89.86±0.08%	83.45±0.13%	98.25±0.04%	90.50±0.23%	83.97±0.21%
1 : 8	97.97±0.12%	89.55±0.11%	82.96±0.14%	98.18±0.12%	90.01±0.23%	83.55±0.17%
1 : 16	97.86±0.08%	89.31±0.23%	82.98±0.32%	98.04±0.05%	89.89±0.13%	83.14±0.07%

As shown in Table 11, the 2:4 setting provides the best trade-off among the fine-grained configurations. On Fashion-MNIST, constant fan-in reaches 90.73%, outperforming the dense baseline of 90.43%. On MNIST, both fan-in and fan-out 2:4 settings remain within 0.30% of the dense network. Accuracy decreases under more aggressive 1:8 and 1:16 patterns, especially on EMNIST, which is the most challenging of the three datasets. Even so, the degradation is gradual rather than catastrophic, supporting the claim that SCHTs can maintain usable performance under strict local sparsity.

G.2.2 Fine-Grained N:M Sparsity Experiments on CNNs

This experiment extends the main-text CNN evaluation from coarse-grained to fine-grained N:M sparsity. We apply SCHTs to the linear layers of GoogLeNet and ResNet-152 on CIFAR-100 and TinyImageNet. Because the sparse linear layers receive high-level features from convolutional blocks, the experiment uses proxy-correlation initialization described in Appendix A.2 rather than feature-correlation SSTI. The goal is to test whether fine-grained semi-structured sparsity can still preserve or improve accuracy in deeper ANN backbones.

Table 12 shows that fine-grained SCHTs scales well to deeper CNN backbones. On GoogLeNet, the 1:16 setting achieves the strongest available results, reaching 76.94% and 77.12% on CIFAR-100 under constant fan-in and fan-out, respectively, both above the dense baseline of 76.64%. On TinyImageNet, 1:16 also improves over the dense baseline, reaching 64.26% and 64.43%. For ResNet-152 on CIFAR-100, 1:16 exceeds the dense baseline under constant fan-in, with the best value of 78.73%. These results suggest that, for CNN classifiers, stricter fine-grained sparsity does not necessarily reduce accuracy and can sometimes improve generalization.

The phenomenon where extreme sparsity (e.g., 1:16) outperforms both the dense baseline and lower sparsity levels (e.g., 1:4, 1:8) corroborates the principle of "compositional sparsity" inherent in real-world computable functions [44]. Unlike dense networks, which are highly susceptible to expressive saturation [45], strict fine-grained sparsity acts as a powerful structural regularizer that mitigates overfitting. By continuously rewiring connections through dynamic topological evolution,

Table 12: Accuracy of SCHTs on CIFAR-100 and TinyImageNet using GoogLeNet and ResNet-152 under fine-grained N:M semi-structured sparsity. SCHTs is applied to linear layers except the output layer and is evaluated under constant fan-in and constant fan-out constraints. The FC row reports the corresponding dense-network baselines. The best results of SCHTs are underlined. SCHTs accuracies are reported as mean ± standard deviation (SD) across three random seeds. Uncertainty estimates are not reported for the dense baseline.

Method	SCHTs - Constant Fan-In			SCHTs - Constant Fan-Out		
	GoogLeNet CIFAR-100	GoogLeNet TinyImageNet	ResNet-152 CIFAR-100	GoogLeNet CIFAR-100	GoogLeNet TinyImageNet	ResNet-152 CIFAR-100
FC	76.64%	63.24%	78.30%	76.64%	63.24%	78.30%
2 : 4	76.23±0.19%	63.07±0.08%	78.57±0.28%	76.38±0.18%	62.68±0.30%	77.59±0.28%
1 : 4	76.70±0.20%	62.99±0.44%	78.13±0.65%	76.75±0.50%	62.74±0.43%	78.25±0.23%
1 : 8	76.72±0.48%	63.55±0.17%	78.17±0.14%	77.06±0.56%	63.68±0.02%	77.70±0.78%
1 : 16	<u>76.94±0.51%</u>	<u>64.26±0.24%</u>	<u>78.73±0.51%</u>	<u>77.12±0.75%</u>	<u>64.43±0.21%</u>	<u>78.29±0.89%</u>

SCHTs effectively bypasses the capacity bottlenecks of dense architectures. It discovers excellent sparse computational paths that match the underlying data structure to ultimately achieve strong generalization performance.

H Ablation Experiment

H.1 Ablation Study on the SSTI Module

This ablation studies the contribution of SSTI. We evaluate ANN MLPs on MNIST, Fashion-MNIST, and EMNIST under coarse-grained N:M sparsity at 90% linear-layer sparsity. The comparison between SCHTs with and without SSTI directly measures whether correlation-based topology initialization improves the subsequent dynamic sparse training process.

Table 13: Ablation study of SSTI on MNIST, Fashion-MNIST, and EMNIST using ANN MLPs. SCHTs adopts coarse-grained N:M semi-structured sparsity under constant fan-in or constant fan-out, and the sparsity of the linear layer is set to 90%. The FC row reports the dense baseline, while "w/ SSTI" and "w/o SSTI" compare SCHTs with and without SSTI. Accuracies are reported as mean ± standard deviation (SD) across three random seeds.

Method	SCHTs - Constant Fan-In			SCHTs - Constant Fan-Out		
	MNIST	Fashion-MNIST	EMNIST	MNIST	Fashion-MNIST	EMNIST
FC	98.69±0.02%	90.43±0.09%	85.58±0.06%	98.69±0.02%	90.43±0.09%	85.58±0.06%
w/ SSTI	98.39±0.07%	90.58±0.27%	84.84±0.28%	98.41±0.11%	90.58±0.07%	84.58±0.05%
w/o SSTI	98.13±0.10%	89.65±0.21%	83.71±0.23%	98.12±0.05%	89.80±0.17%	83.61±0.03%

Table 13 confirms that SSTI consistently improves SCHTs. Under constant fan-in, SSTI increases accuracy by 0.26%, 0.93%, and 1.13% on MNIST, Fashion-MNIST, and EMNIST, respectively. Under constant fan-out, the corresponding gains are 0.29%, 0.78%, and 0.97%. The gains are especially large on Fashion-MNIST and EMNIST, where the input-feature correlations are more informative and the classification task is more difficult. This supports the role of SSTI as a better starting point for the subsequent prune-and-regrow topology evolution.

H.2 Ablation Study on the Block Removal Module

This ablation evaluates the block removal module used in fine-grained N:M sparsity. The experiment is conducted on LLaMA-60M trained on OpenWebText and reports validation perplexity, where lower values are better. Fine-grained blocks can become invalid when too many dead neurons appear inside the same local group. The block removal module removes such invalid groups before regrowth, allowing SCHTs to restore a valid N:M topology more effectively.

Table 14: Ablation study of block removal on LLaMA-60M trained on OpenWebText under fine-grained N:M semi-structured sparsity. Validation perplexity is reported, so lower values are better. SCHTs is evaluated under constant fan-in and constant fan-out constraints, and "w/o" and "w/ block removal" denote whether the block removal module is disabled or enabled. The dense FC model is included as the reference baseline.

Sparsity	SCHTs-Constant fan-in		SCHTs-Constant fan-out	
	w/o	w/ block removal	w/o	w/ block removal
FC	26.56			
2 : 4	31.07	27.89	28.34	26.03
1 : 4	31.07	31.06	37.04	28.93
1 : 8	34.85	33.48	37.99	32.97
1 : 16	40.18	38.69	40.64	37.80

As shown in Table 14, block removal improves perplexity for every fine-grained sparsity pattern and for both constant fan-in and constant fan-out. The improvement is most pronounced under constant fan-out, where 1:4 sparsity improves from 37.04 to 28.93 after enabling block removal. At 2:4 sparsity, block removal reduces perplexity from 31.07 to 27.89 under constant fan-in and from 28.34 to 26.03 under constant fan-out; the latter even outperforms the dense baseline of 26.56. The consistent gains indicate that invalid or weak local blocks are a major optimization bottleneck in fine-grained N:M DST, and removing them before regrowth makes the sparse topology easier to repair.

I Hardware Evaluation

I.1 End-to-End LLaMA Inference on A100

We benchmark shape-faithful LLaMA-60M, LLaMA-130M, and LLaMA-1B inference on one NVIDIA A100 80 GB PCIe GPU. All experiments use FP32, batch size 32, prompt length 512, and eight generated tokens. The complete path includes embedding, attention, KV-cache updates, MLPs, the output head, prefill, and autoregressive decoding. We sparsify the same projections targeted by SCHTs, corresponding to 56, 84, and 112 logical matrices in the three models. The benchmark uses randomly generated weights because kernel latency is value-independent, while layer dimensions, constant-fan-in layouts, and operator placement match the SCHTs setting. For each configuration, five independently generated masks are measured five times using CUDA Events. Table 15 reports the mean and SEM across masks.

Table 15: End-to-end LLaMA inference latency on an NVIDIA A100 80 GB PCIe GPU. SCHTs-Hybrid uses the sparse kernel for prefill and dense cuBLAS for autoregressive decoding. Values are mean $\pm$ SEM across five sparse masks.

Model	N:M	Dense (ms)	SCHTs-Hybrid (ms)	Speedup
LLaMA-60M	2:4	85.047 ± 0.210	91.311 ± 0.142	$0.931\times$
	1:4	85.021 ± 0.099	68.053 ± 0.074	$1.249\times$
	1:8	85.033 ± 0.078	54.953 ± 0.125	$1.547\times$
	1:16	84.783 ± 0.099	49.666 ± 0.049	$1.707\times$
LLaMA-130M	2:4	236.745 ± 0.446	260.175 ± 0.301	$0.910\times$
	1:4	235.615 ± 0.078	181.934 ± 0.078	$1.295\times$
	1:8	234.951 ± 0.120	137.862 ± 0.129	$1.704\times$
	1:16	234.706 ± 0.277	119.231 ± 0.085	$1.969\times$
LLaMA-1B	2:4	1731.583 ± 2.858	1999.233 ± 3.969	$0.866\times$
	1:4	1726.670 ± 1.432	1286.762 ± 15.239	$1.342\times$
	1:8	1720.491 ± 1.234	834.325 ± 0.541	$2.062\times$
	1:16	1717.617 ± 1.659	640.627 ± 0.286	$2.681\times$

The 2:4 implementation is slower than dense execution for all three models. From 1:4 onward, SCHTs-Hybrid consistently accelerates inference. At 1:16, speedups reach $1.71\times$, $1.97\times$, and $2.68\times$ for LLaMA-60M, LLaMA-130M, and LLaMA-1B, respectively. These measurements establish

end-to-end inference acceleration, not end-to-end training acceleration, because the kernels do not cover the full backward and topology-update pipeline.

I.2 Prefill and Decode Breakdown

SCHTs-Hybrid dispatches the sparse kernel to the large-matrix prefill phase and retains dense cuBLAS for small-M autoregressive decoding. Table 16 localizes this design choice on LLaMA-1B. Prefill has $M = B \times L = 16{,}384$, which provides sufficient parallelism and weight reuse. Each decode step has $M = B = 32$ and is more sensitive to metadata decoding, kernel launches, weight movement, and low occupancy.

Table 16: Prefill/decode latency breakdown for LLaMA-1B. The hybrid path uses sparse prefill and dense decode.

N:M	Dense total	Hybrid	Sparse prefill+decode	Sparse-decode penalty
2:4	1731.6 ms	1999.2 ms	2228.8 ms	$+229.5$ ms
1:4	1726.7 ms	1286.8 ms	1442.8 ms	$+156.0$ ms
1:8	1720.5 ms	834.3 ms	912.8 ms	$+78.5$ ms
1:16	1717.6 ms	640.6 ms	693.6 ms	$+53.0$ ms

The end-to-end improvement originates primarily from sparse prefill. The current sparse decode path launches 64 sparse kernels per LLaMA-1B forward pass and cannot be captured by `torch.compile` through the PyBind CUDA extension. Dense decoding therefore avoids a regime in which the current sparse kernel is inefficient.

J Limitations

The current SCHTs formulation sparsifies linear layers only. It therefore does not reduce the cost of convolutional operators, and its whole-model benefit for convolution-dominated networks depends on the fraction of computation assigned to linear layers. Applying the present topology evolution directly to an expanded convolution matrix would either count inherent structural zeros or break convolutional weight sharing. Extending SCHTs to convolution therefore requires kernel-parameter-level topology evolution rather than a direct reuse of the linear-layer formulation. A practical intermediate direction is to combine SCHTs for linear layers with existing kernel-level N:M pruning for convolutional layers. This limitation is less restrictive for Transformer models, whose attention projections and feed-forward blocks are dominated by linear transformations.

Hardware speedup is also conditional on the execution backend. Our real-hardware results are limited to an NVIDIA A100 GPU, and they do not establish equivalent performance on H100 GPUs, CPUs, or edge accelerators. The current FP32 fine-grained 2:4 and coarse-grained 80% kernels are slower than dense cuBLAS. The end-to-end results establish inference acceleration only; they do not cover backward propagation or topology updates. Finally, constant fan-in and fan-out are alternative modes in the current local-regrowth logic and cannot be enforced simultaneously.

K Experiments compute resources

All experiments were conducted on NVIDIA A100 80GB GPUs. Specifically, the SNN and ANN evaluations were executed on a single GPU, while the LLM training experiments were parallelized across four GPUs.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction describe the scope and contributions of SCHTs. Predictive performance is evaluated on SNNs, ANNs, and LLMs in Section 4 and the additional experiments in the appendix. Hardware efficiency is evaluated using operator-level measurements on an NVIDIA A100 GPU and end-to-end inference benchmarks in Appendix I. These measurements support inference acceleration in the evaluated configurations; they do not establish end-to-end training acceleration.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Appendix J discusses the limitations of SCHTs, including its restriction to linear layers and the inability of the current method to enforce constant fan-in and constant fan-out simultaneously. The hardware evaluation also distinguishes operator-level and end-to-end inference acceleration from training acceleration.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: This paper focuses on algorithmic design, hardware co-optimization, and empirical evaluations. It does not propose formal mathematical theorems or theoretical proofs.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Detailed experimental settings, including datasets, network architectures, and hyperparameters, are thoroughly documented in Appendix F. Furthermore, to fully facilitate reproducibility, the complete source code is provided in our GitHub repo.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We release code and data in our GitHub repo.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The core experimental configurations are introduced in Section 4 of the main text. Comprehensive training and test details necessary to understand and reproduce the results—including hyperparameters, datasets, and optimizer choices for SNNs, ANNs, and LLMs—are extensively documented in Appendix F. We release code and data in our GitHub repo.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: Where uncertainty estimates are provided, predictive-performance results are reported as the mean plus or minus the standard deviation across three random seeds, as specified in the corresponding table captions. Operator-level hardware measurements report the mean and SEM across five trained weight matrices, while end-to-end inference measurements report the mean and SEM across five independently generated sparse masks. Entries reported without uncertainty estimates should not be interpreted as having zero variance. We report these uncertainty estimates rather than formal hypothesis-test results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Information regarding the compute resources, including the hardware infrastructure (e.g., GPU specifications and memory) utilized to execute the experiments, is documented in Appendix K.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This research adheres to the NeurIPS Code of Ethics.

   Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This does not have a direct path to specific negative societal applications, and therefore a dedicated broader impact discussion is not included.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: No risk for misuse.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All existing assets used in the paper are properly credited.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The primary new asset introduced in this work is the source code for the proposed SCHTs framework.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: This paper does not involve any form of crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This paper does not conduct any research involving human subjects. Therefore, Institutional Review Board (IRB) approval is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development in this research, specifically the SCHTs framework and sparsity algorithms, does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.