Cannistraci-Hebb Training with N:M Semi-Structured Sparsity for Pre-Training and Re-Training

Jiaqing Lyu[1], Ruijie Wang[2], Kangyou Bao[1], Yingtao Zhang[1], Carlo Vittorio Cannistraci[1*]
[1]Tsinghua University, [2]University of Oxford
{lyujiaqing, baoky302, zhangyingtao1024, kalokagathos.agon}@gmail.com,
ruijie.wang@wadham.ox.ac.uk

Sparse training offers a pivotal pathway for scaling deep learning efficiency, replacing dense networks with sparse counterparts that maintain competitive performance using significantly fewer parameters. While brain-inspired sparse training methods like Cannistraci-Hebb Training (CHT) have shown great promise, they typically rely on unstructured sparsity, failing to exploit the acceleration capabilities of modern GPU architectures. Conversely, NVIDIA's N:M semi-structured sparsity has emerged as a standard for hardware-efficient acceleration. However, the existing N:M training methods always rely on straight-through estimators (STE) and need to maintain dense weights, which do not constitute true sparse training. In this work, we bridge the gap between dynamic sparse training and hardware efficiency. We make three primary contributions: (1) We introduce CHTs24, the first framework to integrate Cannistraci-Hebb Training with 2:4 semi-structured sparsity. This approach outperforms strong baselines (e.g., SR-STE) in training linear layers within Large Language Models (LLMs). (2) We propose the epitopology Dynamic Sparse re-Training (eDSrT) pipeline, a novel methodology for transitioning dense models to semi-structured sparsity. (3) We demonstrate the efficacy of this pipeline by adapting CHTs24 to prune and retrain a Vision Transformer (ViT) into 2:4 sparsity in just 100 epochs with negligible performance loss. Collectively, our research presents a synergistic, hardware-friendly approach to advancing sparse training for large-scale neural networks.

1. Introduction

The exponential growth in the scale of Large Language Models (LLMs) has precipitated a significant challenges in computational efficiency, intensifying the demand for training paradigms that are both accurate and resource-efficient [1]. While modern hardware accelerators, particularly NVIDIA GPUs, deliver substantial throughput for dense matrix operations, the resulting models are often severely over-parameterized[2], incurring prohibitive costs for training and deployment. Sparse training has emerged as a pivotal solution to this tension. By replacing dense Artificial Neural Networks (ANNs) with sparse counterparts, sparse training aims to maintain competitive performance while significantly reducing parameter counts and computational requirements[3]. The primary challenge in this field is to design strategies that simultaneously satisfy three criteria: (i) preserving model performance, (ii) exploiting hardware-specific acceleration capabilities, and (iii) scaling effectively to modern deep learning architectures.

Dynamic Sparse Training (DST) addresses this challenge by initializing a sparse topology and evolving it during training via prune-and-regrow regimes. Recent research demonstrates that, under optimal topological evolution, ultra-sparse networks can rival or even surpass their fully connected

[*]This work was completed at the Complex Network Intelligence Center, Tsinghua Laboratory of Brain and Intelligence, Department of Psychological and Cognitive Sciences, Tsinghua University. Jiaqing and Yingtao are affiliated with the Department of Computer Science and Technology; kangyou is affiliated with the Department of Physics; Carlo Vittorio Cannistraci is affiliated with the Department of Psychological and Cognitive Sciences, Department of Computer Science and Technology, and School of Biomedical Engineering, Tsinghua University.

counterparts[4–8]. A leading approach in this domain is Cannistraci–Hebb Training (CHT) [8] and its soft rule variant (CHTs)[9], which introduces a brain-inspired, network-science perspective to DST. CHT and CHTs evolve network topology via link prediction—a process termed epitopological learning[10]—where the network learns by actively reshaping its connectivity rather than merely updating weight values.

However, a critical disconnect exists between the theoretical efficiency of DST and practical hardware realization. Most DST methods, including CHT and CHTs, rely on unstructured sparsity. The patterns prevent current GPUs from utilizing optimized dense tensor cores, often leading to no practical speedup or even slower training compared to dense baselines[11]. To address this, NVIDIA introduced N:M semi-structured sparsity (specifically the 2:4 pattern), where at least N weights are zeros in every contiguous block of M[12, 13]. Previous research on 2:4 sparsity have demonstrated that, for a fixed 50% sparsity level, 2:4 patterns can accelerate training and inference with modest accuracy loss[12–20]. While this pattern offers tangible hardware acceleration, existing methods for training 2:4 networks predominantly rely on Straight-Through Estimators (STE)[19, 20].

Crucially, these STE-based approaches are not *true* sparse training. They require maintaining a dense weight matrix during the backward pass and estimate gradients for the weights which are not in the forward process. This dependence on dense weights negates the memory reduction benefits of sparsity during training and reduces the process to a dense training routine with a sparse forward pass. Consequently, the field lacks a framework that combines the parameter efficiency of true dynamic sparse training with the acceleration of semi-structured hardware constraints.

In this work, we bridge this gap between dynamic sparse training and hardware efficiency. We propose a synergistic approach that eliminates the need for dense weight maintenance and STE by integrating brain-inspired topological evolution with hardware-friendly constraints. Our contributions are threefold:

- **CHTs24: The First CHT Framework with 2:4 Semi-Structured Sparsity.** We introduce CHTs24, a novel framework that integrates Cannistraci–Hebb Training with 2:4 semi-structured sparsity. Unlike existing methods that rely on STE and dense weight buffers, CHTs24 implements true sparse training by evolving the topology strictly within 2:4 constraints. This allows for genuine parameter reduction during training. Our experiments demonstrate that CHTs24 outperforms strong baselines, such as SR-STE[21], in training linear layers within LLMs.
- **The epitopology Dynamic Sparse re-Training (eDSrT) Pipeline.** We propose eDSrT, a novel methodology designed to transition dense models into semi-structured sparsity. This pipeline resolves the difficulty of imposing 2:4 constraints on pre-trained dense weights without severe accuracy degradation. By leveraging epitopological principles to guide the redistribution of connections, eDSrT provides a principled pathway for converting legacy dense models into efficient, hardware-compliant sparse networks.
- **Efficient Dense-to-Sparse Transition for Vision Transformers.** We demonstrate the efficacy of the eDSrT pipeline by adapting CHTs24 to prune and retrain a Vision Transformer (ViT) into a 2:4 sparse architecture. This process achieves the target sparsity in just 100 epochs with negligible performance loss. This result validates that our brain-inspired approach is not limited to LLMs but serves as a generalizable, hardware-friendly strategy for advancing sparse training across large-scale neural architectures.

Collectively, our research presents a distinct methodology for efficient deep learning. By successfully decoupling N:M sparse training from the computational burden of Straight-Through Estimators and dense weight retention, we move beyond gradient approximation toward strictly constrained, hardware-accelerated sparsity. This work proves that Cannistraci–Hebb dynamics can be effectively harmonized with the rigid constraints of modern GPU architectures, thereby resolving the long-standing gap between dynamic sparse training and hardware acceleration. By demonstrating that complex architectures—from LLM linear layers to Vision Transformers—can evolve into hardware-efficient states without relying on dense proxies, we provide a scalable method for training large-scale sparse neural networks.

2. Related Work

2.1. Dynamic Sparse Training: From Gradients to Topology

Gradient-Driven DST. Dynamic Sparse Training (DST) challenges the traditional dense training paradigm by maintaining a fixed sparsity budget throughout the optimization process. This is typically achieved via a cycle of pruning low-magnitude weights and regrowing connections to explore the parameter space. Early foundational works, such as SET[4], utilized random regrowth, proving that stochastic topology search could approximate dense model performance. Subsequent iterations sought to accelerate convergence by incorporating gradient information into the regrowth criteria. RigL[6] utilizes the magnitude of gradients to identify high-saliency missing links, while methods like MEST[7] leverage momentum-based heuristics for pruning.

Epitopological Learning and Cannistraci-Hebb Training. A recent paradigm shift in DST is the transition from gradient-dependent evolution to intrinsic topological evolution, formalized as Epitopological Learning (EL)[8]. Grounded in network science, EL posits that the connectivity structure of a neural network encodes latent information sufficient to predict future optimal links. Cannistraci–Hebb Training (CHT) operationalizes this principle through Cannistraci–Hebb (CH) automata theory, leveraging the CH3–L3 path-based (CH3-L3p) link predictor[8, 9, 22]. In contrast to RigL—which incurs the computational cost of computing gradients for non-existent connections—CHT functions as a topological predictor, using local community organization to guide synaptic rewiring. While CHT achieves state-of-the-art efficiency among brain-inspired DST methods, it also exhibits certain limitations: its deterministic link predictions can cause the epitopological dynamics to become trapped in epitopological local minima, and the path-based link predictor introduces substantial computational overhead. CHTs (the CHT soft rule) mitigate these issues by incorporating soft link regrowth and soft link removal and by adopting the lighter CH2–L3 node-based (CH2-L3n) link predictor. Nevertheless, despite their efficiency gains, both CHT and CHTs operate on unstructured sparsity, which remains incompatible with high-throughput GPU execution—motivating the integration of semi-structured sparsity constraints proposed in this work.

2.2. N:M Semi-Structured Sparsity

Hardware-Friendly Sparsity Patterns. To reconcile the tension between algorithmic sparsity and hardware utilization, NVIDIA introduced the N:M semi-structured sparsity format (typically 2:4) for Ampere architectures[12, 13]. This format enforces a rigid constraint where every contiguous block of M weights must contain at least N zeros. For acceleration, the granularity of this constraint is critical. *Row-wise* N:M sparsity accelerates the forward pass ($C = A \times B$) but fails to accelerate the backward pass, as the transposition of A disrupts the row-aligned sparse structure. Conversely, *Block-wise* (or Transposable) N:M sparsity enforces constraints on both rows and columns, enabling acceleration of both forward and backward passes via Sparse Tensor Cores[15, 19]. Our work focuses on the latter to achieve end-to-end training acceleration.

The Limitations of Straight-Through Estimators (STE). Current methodologies for training N:M networks from scratch predominantly rely on STE to approximate gradients[16, 19, 20]. In these frameworks (e.g. SR-STE[21]), the system maintains a *dense* master weight matrix. During the forward pass, a binary mask is applied to enforce N:M sparsity; during the backward pass, gradients are calculated for the projected sparse weights but applied to the dense master weights. While effective for accuracy, this approach is fundamentally *not* sparse training. It requires memory to store dense parameters and incurs the computational cost of dense gradient updates. Recent variants like SR-STE[21] add regularization to prevent weights that do not participate in the forward propagation from being updated significantly, but do not resolve the dense storage requirement. In contrast, our proposed CHTs24 framework eliminates the dense master weight entirely, performing updates only on valid N:M connections.

From Pruning to Dynamic Retraining. Beyond training from scratch, transforming pre-trained dense models (e.g., LLMs, ViTs) into N:M sparse counterparts is a critical optimization challenge. Standard approaches involve one-shot magnitude pruning followed by fine-tuning[12], which may lead to accuracy degradation due to the rigid N:M constraint. While iterative retraining strategies exist, they are typically static—pruning weights once and retraining fixed connections. To the best of our knowledge, no existing framework applies *dynamic* sparse training principles to the retraining phase. By allowing the topology to evolve dynamically during retraining, our proposed eDSrT pipeline offers a novel mechanism to recover accuracy while strictly adhering to hardware-efficient constraints.

3. Method

While the Cannistraci-Hebb Training soft rule (CHTs) successfully introduces brain-inspired epitopological learning to unstructured sparsity, the translation of these principles to hardware-accelerated formats remains an open challenge. Current hardware accelerators, particularly NVIDIA's Ampere architectures, utilize N:M semi-structured sparsity (typically 2:4) to double throughput. However, standard methods for training 2:4 networks rely heavily on straight-through estimators (STE).

Here, we present CHTs24 (Cannistraci-Hebb Training structured 2:4), a novel framework that enforces strict row-wise and column-wise 2:4 constraints while leveraging network shape intelligence to evolve topology. CHTs24 is not merely an extension of CHTs; it represents a transition from unstructured sparsity to semi-structured sparsity of brain-inspired dynamic sparsity training, enabling efficient exploration of the sparsity acceleration with constrained topological.

3.1. Theory of Transposable 2:4 Sparsity

Standard 2:4 sparsity requires that every block of 4 contiguous elements in a row contains 2 non-zero values. To maximize theoretical acceleration, we use a stricter transposable 2:4 sparsity mask, where the 2:4 constraint applies simultaneously to both rows and columns within 4×4 tensor blocks.

Let $\mathbf{W} \in \mathbb{R}^{N \times M}$ be a weight matrix partitioned into 4×4 sub-blocks $\mathbf{B}^{(k)}$. A binary mask $\mathbf{M}^{(k)} \in \{0,1\}^{4\times4}$ is valid if and only if:

$$\sum_{j=1}^{4} M_{ij}^{(k)} = 2 \quad \forall i, \quad \text{and} \quad \sum_{i=1}^{4} M_{ij}^{(k)} = 2 \quad \forall j.$$

Combinatorial analysis reveals that there are exactly $|\mathcal{P}| = 90$ unique 4×4 binary patterns that satisfy this transposable 2:4 sparsity (excluding permutations that break the row/col sums). Let $\mathcal{P} = \{\mathbf{P}_1, \ldots, \mathbf{P}_{90}\}$ be the set of these valid distinct patterns (the "atomic topologies").

The challenge of CHTs24 is to dynamically select the optimal assignment $\mathbf{M}^{(k)} \in \mathcal{P}$ for every block k across the dynamic sparsity training process, maximizing the epitopological fitness defined by the Cannistraci-Hebb theory, without violating the hardware constraints.

3.2. Soft Epitopological Learning under Semi-structured Sparsity

3.2.1. Topological Initialization

To avoid inducing inductive bias before the data is seen, we assume a uniform prior over the valid atomic topologies. For every 4×4 block in the network weight tensors, we sample an initial mask index $z \sim \mathcal{U}\{1, 90\}$ and assign $\mathbf{M}^{(k)} \leftarrow \mathbf{P}_z$. This ensures that the network begins with a valid transposable 2:4 structure, serving as an unbiased starting point for the evolutionary process.

3.2.2. The Soft Semi-structured Pruning via Fréchet Perturbation

To escape local minima, CHTs24 utilizes *soft* probabilistic removal, where weights are dropped based on a probability distribution derived from their magnitude. Directly sampling of the whole matrix like Zhang et al.[9] may generate a mask which violates 2:4 sparsity. This occurs because the magnitudes of some weights are very close to zero in floating point. The time cost of sampling every 4×4 block one by one is unacceptable. As a result, we propose an equivalent probabilistic outcome via a deterministic maximization step by introducing specific noise into the weight magnitudes, analogous to the Gumbel-Max trick used in discrete optimization.

We propose injecting Fréchet (Inverse Weibull) noise into the weight tensor. This specific distribution is chosen because maximizing over Fréchet-perturbed magnitudes is equivalent in distribution to sampling from the distribution of weight importance $|W|^T$. The detailed proof is shown in Appendix A.

Let $\mathbf{W}$ be the weights. We compute the **Stochastic Magnitude Matrix $\tilde{\mathbf{W}}$** by modulating the absolute weight values with a noise factor derived from a Uniform distribution $U_{ij} \sim \mathcal{U}(0,1)$:

$$\tilde{W}_{ij} = |W_{ij}| \cdot (-\ln(U_{ij}))^{-1/T}$$

Here, T acts as the topological temperature. This multiplicative perturbation transforms the discrete sampling problem into a maximum value selection problem. We then sort all the weights based on $\tilde{W}_{ij}$, and remove small ones.

3.2.3. The Soft-CH2-L3n Score

We adopt the node-based predictor as our base metric for structural likelihood to guide epitopology learning. To introduce the necessary exploration noise (softness) without the overhead of multinomial sampling, we define the soft-CH2-L3n score $\tilde{S}_{ij}$ as:

$$\tilde{S}_{ij} = \sum_{z \in L3} \frac{di^*_z}{de^*_z} \times \xi_{ij}, \quad \text{where } \xi_{ij} \sim \mathcal{U}[0,1]$$

In this formulation, i and j represent the seed nodes, and z denotes an intermediate node located on an L3 path—a three-hop walk connecting i to j [23]. Nodes which are located on all L3 paths between i and j form a local community. The variables di^*_z and de^*_z refer to the count of internal local community links (iLCLs) and external local community links (eLCLs) associated with node z, respectively. Both terms include a default increment of 1 to avoid zero. By definition, iLCLs connect nodes within the same local community, whereas eLCLs connect nodes across different communities. The detailed information of CH2-L3n can be found in Appendix B.

Here, $\mathcal{U}[0,1]$ is a uniform random variable sampled independently for each potential link. This multiplicative noise perturbs the rank order of connections. By adjusting the score directly, we can employ deterministic maximization in the selection step while still maintaining the probabilistic exploration properties required.

Crucially, soft-CH2-L3n extends beyond merely injecting stochasticity into channel-wise link predictors; it is rigorously designed to satisfy the hardware constraints and computational requisites of 2:4 semi-structured sparsity. From the perspective of computational efficiency, standard soft sampling within 2:4 blocks necessitates distinct sampling operations for each block, significantly increasing the algorithm's computational overhead. In contrast, soft-CH2-L3n enables the replacement of sampling with Top-k selection of the whole matrix, a strategy that aligns effectively with the hardware acceleration principles of 2:4 sparsity.

From the perspective of training dynamics, Zhou et al.[21] found that frequent 2:4 mask oscillation during the late stages of training deteriorates model performance. soft-CH2-L3n mitigates this issue by establishing an inverse relationship between the variance of the CH2-L3n metric and the injected stochasticity. The mathematical proof is in Appendix C. During the early training phase, the network topology is nascent and random, resulting in lower metric variance; consequently, soft-CH2-L3n exhibits higher stochasticity, encouraging exploration during link regrowth. Conversely, as the topology evolves into a community structure towards the end of training, the metric variance increases. In this regime, soft-CH2-L3n inherently attenuates randomness, thereby suppressing mask oscillation and stabilizing the final network structure. The empirical results are presented in Appendix D.

3.2.4. The Link Regrowth under Semi-Structured Sparsity

We implement a Survivor Boosting mechanism. Let $\mathcal{A}$ be the set of active links after the previous pruning. We augment the scores:

$$\hat{S}_{ij} = \begin{cases} \tilde{S}_{ij} + \Gamma & \text{if } (i,j) \in \mathcal{A} \\ \tilde{S}_{ij} & \text{otherwise} \end{cases}$$

where Γ is a large scalar constant ensuring that currently active links are preferentially retained.

The regrowth step then becomes a Block-wise Pattern Optimization. For each 4×4 block k, we extract the local 4×4 score sub-matrix $\hat{\mathbf{S}}^{(k)}$. We select the new topology $\mathbf{M}_{new}^{(k)}$ by solving:

$$\mathbf{M}_{new}^{(k)} = \underset{\mathbf{P} \in \mathcal{P}}{\arg\max} \left(\sum_{x=1}^{4} \sum_{y=1}^{4} \hat{S}_{xy}^{(k)} \cdot P_{xy} \right)$$

This operation selects the valid 2:4 pattern that maximizes the total Cannistraci-Hebb score within the block. Since $|\mathcal{P}| = 90$ is small, this can be implemented efficiently on GPUs via a pre-computed lookup table of patterns, computed as a batched dot product between the score blocks and the pattern tensor $\mathbf{T}_{\mathcal{P}} \in \{0,1\}^{90 \times 4 \times 4}$. Appendix E provides the detailed pseudocode and complexity analysis of regrowth.

3.2.5. The Whole Pipeline of CHTs24 Pretraining

This pipeline differs fundamentally from the original CHT/CHTs. While the original method performs global ranking and allows arbitrary unstructured sparsity, CHTs24 constrains the solution space to $\mathcal{P}$. By replacing global sorting with localized block-wise pattern selection, CHTs24 ensures strict alignment with N:M Tensor Core acceleration.

Integrating the components described above, the CHTs24 pretraining pipeline operates as a discrete optimization process over the manifold of hardware-compliant topologies. Unlike standard DST, which treats connections as independent variables, CHTs24 treats the 4×4 block as the atomic unit of topological evolution. The training process follows a specific cadence to balance weight convergence and topological exploration:

1. **Initialization:** The network is initialized with random weights and a random valid 2:4 topology $\mathbf{M}_0$ sampled uniformly from $\mathcal{P}$.
2. **Sparse Forward/Backward Pass:** For each step, the network performs standard backpropagation. Crucially, gradients are computed *only* for the non-zero elements defined by $\mathbf{M}_t$. No dense gradients are calculated, and no dense master weights are stored, ensuring true sparse training efficiency.
3. **Weight Remove:** At the end of the interval of Δt steps, we compute $\tilde{W}_{ij}$ and remove a portion of weights.
4. **Regrow and Block-wise Pattern Optimization:** We calculate the Regrowth Score derived from the Soft-CH2-L3n metric (Sec. 3.2.3), which predicts the latent potential of connections based on the current local community structure. We combine the regrow scores into a unified matrix $\hat{\mathbf{S}}$. For every disjoint 4×4 block in the tensor, we identify the pattern $\mathbf{P} \in \mathcal{P}$ that maximizes the dot product with the local score block.

You can also find the relationship of CHTs24 to prior work in Appendix F.

3.3. Epitopology Dynamic Sparse Re-Training

A significant industrial need lies in converting pre-trained, high-performance dense models (e.g., LLaMA, ViT) into hardware-efficient 2:4 sparse models. Existing solutions typically employ Static Sparse Retraining — pruning the model once and fine-tuning the remaining weights. This static approach often locks the model into a suboptimal local minimum defined by the initial pruning decision. To address this, we introduce the **epitopology Dynamic Sparse Re-Training (eDSrT)** pipeline. To our knowledge, no previous work applies dynamic sparse training principles under hard N:M constraints during the pretraining and retraining phases.

Here, we define epitopology as the discrete topological space spanned by the permutation set $\mathcal{P}$ of the 90 valid transposable 2:4 patterns. The eDSrT pipeline enables a model to traverse this space dynamically during the retraining phase to find an optimal sparse configuration. The pipeline consists of two distinct phases.

3.3.1. Phase 1: Post Training Pruning

Given a dense weight matrix $\mathbf{W}_{dense}$, we first prune it onto the 2:4 manifold. Unlike standard unstructured pruning, we must respect the block constraints. For each block, we identify the pattern $\mathbf{P}_{best}$ that captures the maximum weight magnitude:

$$\mathbf{M}_{init}^{(k)} = \arg\max_{\mathbf{P}\in\mathcal{P}} \left(\sum_{x,y} |W_{xy}^{(k)}| \cdot P_{xy} \right)$$

3.3.2. Phase 2: Epitopology Dynamic Sparse Training

Unlike standard methods that fix $\mathbf{P}_{init}$ for the remainder of training, eDSrT treats $\mathbf{P}_{init}$ merely as a starting point. eDSrT uses DST method to retrain the model and keeps block 2:4 sparsity. We engage the CHTs24 described in Section 3.2 as the First eDSrT Method. During retraining, the soft-CH2-L3n helps find the good patterns for every 4×4 blocks. The eDSrT process dynamically changes the 2:4 patterns for specific blocks, effectively performing a search on the epitopology landscape.

4. Experiments

In this section, we empirically validate CHTs24 across two distinct regimes: (1) Pre-training of Large Language Models (LLMs) from scratch, and (2) Post-training pruning and retraining of Vision Transformers (ViTs). Our goal is to demonstrate that CHTs24 not only bridges the performance gap between sparse and dense models but does so while strictly adhering to the transposable 2:4 sparsity constraints required for bidirectional (training and inference) hardware acceleration. Furthermore, the success on LLM and ViT also demonstrates that CHTs24 is applicable to tasks of different modalities, such as text and images.

4.1. Pre-Training

Pre-Training Result of 2:4 Sparsity: To evaluate the efficacy of CHTs24 in the generative language modeling domain, we conduct pre-training experiments on a scaled-down version of the Llama-60M architecture. The detailed information of experimental setup can be found in Appendix G.

Table 1 presents the comparative results. CHTs24 achieves a perplexity of 36.33, outperforming the STE baseline and SR-STE baseline. Compared with STE and SR-STE, CHTs24 achieves better performance without maintaining dense weights and gradients. Consequently, the future of sparse training may lie in brain-inspired paradigms like CHTs24. By leveraging epitopology learning, CHTs24 aligns more naturally with the true sparsity.

Pre-Training Result of 1:4 Sparsity: Many efficient AI technologies were first proposed by the academic community and then implemented by chip manufacturers such as NVIDIA, such as sparse cores[24]. Although NVIDIA GPUs currently do not support 1:4 semi-structured sparse computing, 1:4 has already attracted widespread attention in the academic community[25–27]. In the future, NVIDIA may launch GPUs that support 1:4 semi-structured sparse computing. So we adapt CHTs24 into CHTs14 and compare it with two baselines under transposable 1:4 sparsity.

Table 1 presents that CHTs14 is much better than STE and SR-STE method. When the sparse mode becomes 1:4, 75% of the parameters in the STE-based algorithm does not participate in the forward process but need to be updated by the estimated gradient, which obviously further reduces the performance of the STE-based algorithm. This suggests that CHTs24 and CHTs14 with epitopological evolution are a far more promising avenue for maximizing the efficiency of hardware-aware, semi-structured sparse LLMs.

Table 1: Pre-training results on Llama. Ours achieves the best perplexity.

		Llama-60M		Llama-130M	
Method	Sparsity Type	Avg PPL ↓	Std PPL	Avg PPL ↓	Std PPL
STE	Transposable 2:4	39.59	0.24	-	-
SR-STE	Transposable 2:4	36.44	0.09	29.34	0.09
Ours	Transposable 2:4	**36.33**	**0.21**	**29.32**	**0.21**
STE	Transposable 1:4	52.21	0.11	-	-
SR-STE	Transposable 1:4	45.64	0.17	35.07	0.20
Ours	Transposable 1:4	**42.10**	**0.17**	**33.94**	**0.19**

Scaling to Larger Networks: In addition to the previous test on Llama-60M, we have expanded our experiments on Llama-130M. Considering the results of Llama-60M, the experiments of Llama-130M did not include STE to save computing resource. In Table 1, CHTs24 maintains its promising performance over SR-STE at this scale. It indicates that our method has the potential to be extended to larger models.

Convergence Analysis: To ensure that the dynamic topological evolution introduced in our method does not impose an overhead on the convergence of training duration, we analyze the convergence behavior of our method compared to the strong baseline, SR-STE. And the result shows that our method does not incur a convergence penalty. You can find the details in Appendix H.

Pipeline Analysis: To empirically check for potential pipeline stalls from topological evolution steps, we conducted a rigorous pipeline analysis. The result demonstrates that the CHTs24 regrowth mechanism is performed on-the-fly and is highly parallelizable. The block-wise pattern selection aligns well with GPU architecture, causing negligible pipeline stalls and no effective throughput degradation. You can find the details in Appendix I.

FLOPs Analysis: : As noted in recent literature [19], current PyTorch does not yet support acceleration for N:M training. It shows that training can be accelerated with semi-structured sparsity and achieving actual training acceleration requires a rather complex engineering implementation. So we provide FLOPs as an instead. By eliminating the dense gradient calculations required by SR-STE, CHTs24 reduces total training FLOPs to 87% of the SR-STE baseline and 70% of a fully dense baseline. You can find the details in Appendix J.

4.2. Ablation Study: The Role of Softness

We perform an ablation study on the Llama-60M pre-training task to isolate the contributions of the Soft Regrowth and Soft Removal components. As shown in Table 2, all experiments were conducted over three independent runs (seeds 0, 1, 2).

Effect of Soft Regrow: Introducing the soft CH-rule for regrowth improves PPL to 36.36. This suggests that probabilistic exploration of the topological space prevents the network from getting stuck in suboptimal connectivity patterns early in training.

Effect of Soft Removal: Combining Soft Regrow with Soft Removal yields the lowest mean perplexity (36.33). Theoretically, the Soft Removal mechanism mitigates the risk of prematurely pruning newly grown ('young') connections that have not yet accumulated sufficient magnitude—a known limitation of greedy Top-K selection. While the performance variance indicates the stochastic nature of this approach, the improved mean score suggests that this probabilistic protection facilitates better topological exploration and convergence.

Table 2: Ablation study of Soft Regrow and Soft Removal components in CHTs24.

Soft Regrow	Soft Removal	Avg PPL $\downarrow$	Std PPL
$\times$	$\times$	36.41	0.09
✓	$\times$	36.36	0.11
✓	✓	**36.33**	0.21

4.3. CHTs24 for Epitopology Dynamic Sparse Re-Training

In this set of experiments, we apply CHTs24 to a Vision Transformer (ViT) model for eDSrT pipeline. We begin with a fully trained Dense model, prune it onto the 2:4 manifold, and perform a short retraining phase (100 epochs) to recover accuracy. Note that CHTs24 and the Nvidia baseline enforce transposable 2:4 sparsity. Consequently, both of them support acceleration during the forward and backward process of retraining. The detail of experiment can be found in Appendix K.

Table 3 summarizes the retraining performance. CHTs24 achieves a Top-1 accuracy of 81.76%, effectively recovering 99.95% of the dense model's performance within 100 epochs. The accuracy of Nvidia baseline is 80.80%. We hypothesize the reason may be that Nvidia baseline (80.80%) fails to recover latent functional pathways destroyed during the initial "hard" pruning. CHTs24, by allowing the topology to evolve via the soft-ch2-l3n during retraining, successfully rediscovers the good 2:4 sparse model.

Table 3: Retraining results on ViT. CHTs24 recovers nearly the full accuracy of the dense baseline.

Method	Avg Accuracy $\uparrow$	Std Acc
Dense Baseline	81.80%	-
Nvidia	80.80%	0.00
CHTs24 (Ours)	**81.76%**	**0.05**

5. Conclusion

This work addresses a gap between the algorithmic advances in dynamic sparse training and the practical constraints of modern hardware accelerators. While prior brain-inspired methods such as Cannistraci–Hebb Training demonstrated that epitopological learning can endow ultra-sparse networks with competitive performance, they remained confined to unstructured sparsity and thus could not exploit NVIDIA's semi-structured sparse tensor cores. Conversely, existing N:M sparsity approaches for 2:4 or 1:4 patterns have relied on Straight-Through Estimators (STE) with dense master weights, forfeiting the memory and compute benefits of true sparse training.

We bridge this gap by proposing CHTs24, the first framework that couples Cannistraci–Hebb epitopological dynamics with strict, transposable 2:4 semi-structured sparsity. At the core of CHTs24 is a reformulation of dynamic sparse training where the atomic unit of evolution is not the individual weight, but the 4×4 block constrained to one of 90 valid 2:4 patterns. Within this constrained manifold, we introduce:

- Soft semi-structured pruning via Fréchet perturbation, which replaces expensive block-wise sampling with a theoretically grounded Gumbel/Fréchet-style reparameterization, enabling global Top-k selection that is compatible with hardware-efficient 2:4 patterns.
- Soft-CH2-L3n regrowth, which injects controlled stochasticity into the Cannistraci–Hebb node-based predictor. This mechanism performs probabilistic—but deterministically implementable—exploration early in training and automatically anneals randomness as the epitopological structure matures, mitigating harmful late-stage mask oscillations.
- Block-wise pattern optimization under transposable 2:4 constraints, implemented as a small, GPU-friendly lookup over 90 atomic topologies, ensuring that both forward and backward passes can be accelerated on Sparse Tensor Cores without ever materializing dense weights or gradients.

In summary, this work demonstrates that hardware-aware semi-structured sparsity and brain-inspired epitopological learning are not competing paradigms but complementary ingredients of a unified sparse training framework. By eliminating the reliance on straight-through estimators and dense master weights while strictly adhering to transposable 2:4 constraints, CHTs24 and the eDSrT pipeline provide a concrete, scalable path toward authentically sparse, hardware-accelerated training for large-scale neural networks. We hope these results catalyze further research at the intersection of network science, sparse optimization, and hardware–algorithm co-design.

Acknowledgements

This work was supported by: 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 thank Mo Yang for her administrative support.

References

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

[2] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

[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.

[4] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):2383, 2018.

[5] Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. Top-kast: Top-k always sparse training. *Advances in Neural Information Processing Systems*, 33:20744–20754, 2020.

[6] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International conference on machine learning*, pages 2943–2952. PMLR, 2020.

[7] Geng Yuan, Xiaolong Ma, Wei Niu, Zhengang Li, Zhenglun Kong, Ning Liu, Yifan Gong, Zheng Zhan, Chaoyang He, Qing Jin, et al. Mest: Accurate and fast memory-economic sparse training framework on the edge. *Advances in Neural Information Processing Systems*, 34:20838–20850, 2021.

[8] Yingtao Zhang, Jialin Zhao, Wenjing Wu, and Alessandro Muscoloni. Epitopological learning and cannistraci-hebb network shape intelligence brain-inspired theory for ultra-sparse advantage in deep learning. In *The Twelfth International Conference on Learning Representations*, 2024.

[9] 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 *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL `https://openreview.net/forum?id=0M0Qkq9xtY`.

[10] Carlo Vittorio Cannistraci. Modelling self-organization in complex networks via a brain-inspired network automata theory improves link reliability in protein interactomes. *Scientific reports*, 8(1):15760, 2018.

[11] Geonhwa Jeong, Po-An Tsai, Abhimanyu R Bambhaniya, Stephen W Keckler, and Tushar Krishna. Enabling unstructured sparse acceleration on structured sparse accelerators. *arXiv preprint arXiv:2403.07953*, 2024.

[12] Nvidia. A100 tensor core gpu architecture. `https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf`, 2020. Accessed: 2025-01-01.

[13] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

[14] Mohit Mahajan, Wen-Mei Hwu, and Rakesh Nagi. Determining optimal channel partition for 2: 4 fine grained structured sparsity. *Optimization Letters*, 18(9):2079–2090, 2024.

[15] Itay Hubara, Brian Chmiel, Moshe Island, Ron Banner, Joseph Naor, and Daniel Soudry. Accelerated sparse neural training: A provable and efficient method to find n: m transposable masks. *Advances in neural information processing systems*, 34:21099–21111, 2021.

[16] Yucheng Lu, Shivani Agrawal, Suvinay Subramanian, Oleg Rybakov, Christopher De Sa, and Amir Yazdanbakhsh. Step: learning n: M structured sparsity masks from scratch with precondition. In *International Conference on Machine Learning*, pages 22812–22824. PMLR, 2023.

[17] Brian Chmiel, Itay Hubara, Ron Banner, and Daniel Soudry. Minimum variance unbiased n: m sparsity for the neural gradients. *arXiv preprint arXiv:2203.10991*, 2022.

[18] Bradley McDanel, Helia Dinh, and John Magallanes. Accelerating dnn training with structured data gradient pruning. In *2022 26th International Conference on Pattern Recognition (ICPR)*, pages 2293–2299. IEEE, 2022.

[19] Yuezhou Hu, Kang Zhao, Weiyu Huang, Jianfei Chen, and Jun Zhu. Accelerating transformer pre-training with 2: 4 sparsity. *arXiv preprint arXiv:2404.01847*, 2024.

[20] Yuezhou Hu, Jun Zhu, and Jianfei Chen. S-ste: Continuous pruning function for efficient 2: 4 sparse pre-training. *Advances in Neural Information Processing Systems*, 37:33756–33778, 2024.

[21] 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.

[22] Jialin Zhao, Alessandro Muscoloni, Umberto Michieli, Yingtao Zhang, and Carlo Vittorio Cannistraci. Adaptive network automata modelling of complex networks for link prediction. 2025.

[23] Alessandro Muscoloni, Umberto Michieli, Yingtao Zhang, and Carlo Vittorio Cannistraci. Adaptive network automata modelling of complex networks. *Preprints*, May 2022. doi: 10.20944/preprints202012.0808.v3. URL https://doi.org/10.20944/preprints202012.0808.v3.

[24] 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. *ACM SIGARCH Computer Architecture News*, 44(3):243–254, 2016.

[25] Gongfan Fang, Hongxu Yin, Saurav Muralidharan, Greg Heinrich, Jeff Pool, Jan Kautz, Pavlo Molchanov, and Xinchao Wang. Maskllm: Learnable semi-structured sparsity for large language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 7736–7758. Curran Associates, Inc., 2024. doi: 10.52202/079017-0248. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/0e9a05f5ce62284c91e4a33498899124-Paper-Conference.pdf.

[26] Mike Lasby, Max Zimmer, Sebastian Pokutta, and Erik Schultheis. Compressed sparse tiles for memory-efficient unstructured and semi-structured sparsity. In *Sparsity in LLMs (SLLM): Deep Dive into Mixture of Experts, Quantization, Hardware, and Inference*, 2025.

[27] Hongyi Liu, Rajarshi Saha, Zhen Jia, Youngsuk Park, Jiaji Huang, Shoham Sabach, Yu-Xiang Wang, and George Karypis. PROXSPARSE: REGULARIZED LEARNING OF SEMI-STRUCTURED SPARSITY MASKS FOR PRETRAINED LLMS. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=zkxe5vASi8.

A. Proof of the Soft Semi-structured Pruning via Fréchet Perturbation

Problem Statement: We define two methods for selecting a subset of elements from a matrix W.

- **Method A:** Probabilistically select elements based on a categorical distribution derived from the weights W and temperature T. Specifically, the probability of selecting element (i, j) is proportional to $|W_{ij}|^T$.
- **Method B:** Construct a perturbed matrix $\tilde{W}$ where $\tilde{W}_{ij} = |W_{ij}| \cdot (-\ln(U_{ij}))^{-1/T}$, with $U_{ij} \sim$ Uniform$(0, 1)$, and deterministically select the elements corresponding to the largest values of $\tilde{W}$.

Theorem: Selecting the element with the maximum value in $\tilde{W}$ (Method B) is equivalent in distribution to sampling an element index from the categorical distribution defined by probabilities $P(i, j) \propto |W_{ij}|^T$ (Method A). Furthermore, selecting the top-k elements of $\tilde{W}$ is equivalent to Weighted Random Sampling (WRS) without replacement respecting these probabilities.

Proof: We show that Method B generates samples from the distribution defined in Method A by invoking the Gumbel-Max structural property.

First, we transform the perturbed weight $\tilde{W}_i$ into the log-domain. Taking the natural logarithm of the definition provided in Method B:

$$\begin{aligned}\ln(\tilde{W}_i) &= \ln\left(|W_i| \cdot (-\ln(U_i))^{-1/T}\right)\\ &= \ln|W_i| + \ln\left((-\ln(U_i))^{-1/T}\right)\\ &= \ln|W_i| - \frac{1}{T}\ln(-\ln(U_i))\end{aligned}$$

We introduce the random variable G_i, defined as:

$$G_i = -\ln(-\ln(U_i))$$

Since $U_i \sim \mathcal{U}(0,1)$, the variable G_i follows a standard Gumbel distribution, $G_i \sim \text{Gumbel}(0,1)$.

Substituting G_i back into the log-equation:

$$\ln(\tilde{W}_i) = \ln|W_i| + \frac{1}{T}G_i$$

Since the ranking of elements is invariant under strictly monotonic transformations, selecting the indices with the largest $\tilde{W}_i$ is equivalent to selecting the indices with the largest $T \cdot \ln(\tilde{W}_i)$ (assuming $T > 0$). Multiplying by T:

$$\begin{aligned}Z_i &\triangleq T \cdot \ln(\tilde{W}_i)\\ &= T\ln|W_i| + G_i\\ &= \ln(|W_i|^T) + G_i\end{aligned}$$

Let $\phi_i = |W_i|^T$. The score Z_i can be written as:

$$Z_i = \ln(\phi_i) + G_i$$

According to the Gumbel-Max trick (and its extension to the Gumbel-Top-k trick for sampling without replacement), selecting the top k indices that maximize $Z_i = \ln(\phi_i) + G_i$ is equivalent to sampling k indices without replacement from the categorical distribution defined by probabilities:

$$P(i) = \frac{\phi_i}{\sum_{j=1}^{N}\phi_j} = \frac{|W_i|^T}{\sum_{j=1}^{N}|W_j|^T}$$

Conclusion: Method A and Method B are mathematically similar. Method B is the reparameterized algorithmic implementation of the probabilistic sampling described in Method A.

B. CH2-L3n Score

There is a detailed definition of CH2-L3n, using the following equation:

$$\textbf{CH2-L3n}(u,v) = \sum_{z \in L3} \frac{di_z^*}{de_z^*} \tag{1}$$

By definition, iLCLs connect nodes within the same local community, whereas eLCLs connect nodes across different communities. The local community structure, along with the L2 and L3 paths between seeds u and v, is visualized in Figure 1.

C. Proof of the Soft-Ch2-L3n Stochasticity

C.1. Problem Statement:

We define three methods for selecting a subset of elements from a matrix S.

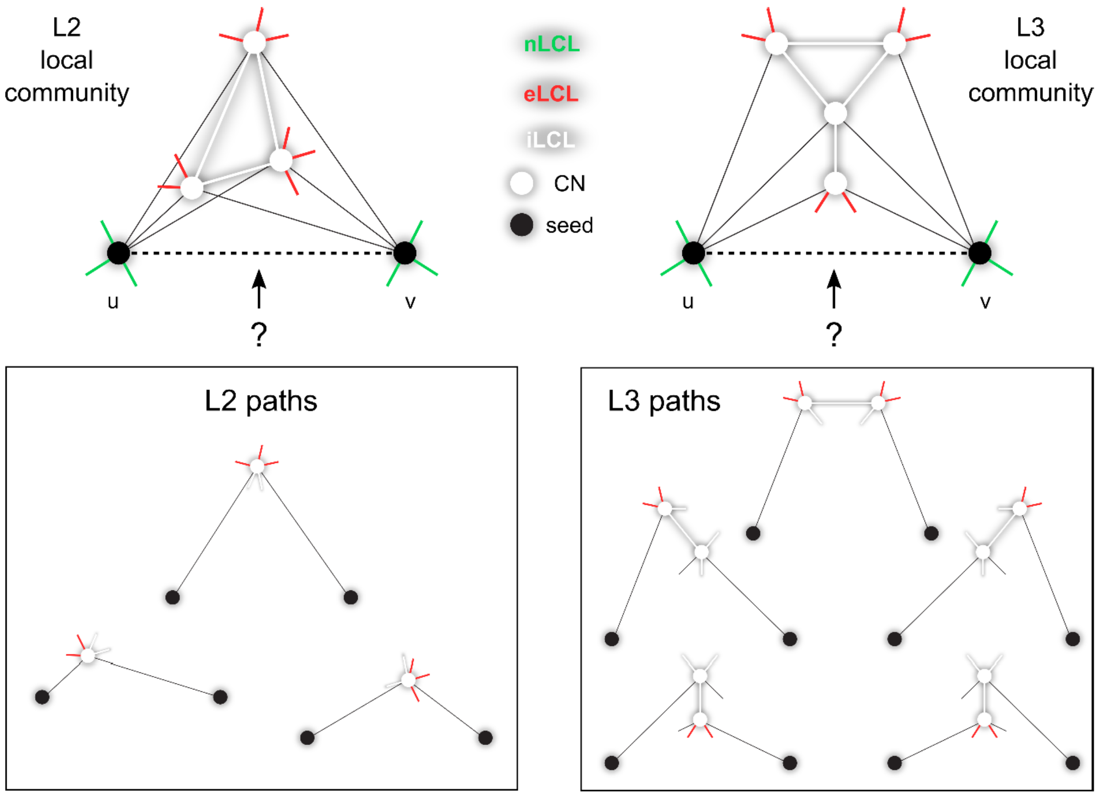


Figure 1: **Cannistraci-Hebb epitopological rationale.** [9, 23] This diagram provides a demonstrative instance of topological link prediction utilizing the Cannistraci-Hebb epitopological framework via L2 or L3 pathways. The pair of black nodes identifies the seed nodes for which the probability of a latent interaction is being calculated. Nodes shown in white signify the common neighbours (CNs) located at an L2 or L3 distance from the seeds. The local community is defined by the combination of these CNs and the internal local community links (iLCL). Link categories are distinguished by specific colors: green represents nLCLs, red indicates external local community links (eLCLs), and white denotes iLCLs. Paths corresponding to path lengths of 2 (L2) and 3 (L3) within the depicted communities are emphasized. Significantly, within artificial neural networks (ANNs), linear layers function as bipartite networks, a structure that inherently accommodates only L3 path predictions.

Method A: Typically refers to **Weighted Random Sampling** (where the probability of picking element i is $P_i = \frac{s_i}{\sum s_j}$).

Method B: A specific form of stochastic ranking where the score is perturbed by multiplicative uniform noise: $s'_i = s_i \cdot u_i$, where $u_i \sim U(0, 1)$.

Method C: The deterministic selection of the largest weights (Top-K).

C.2. Theorem:

As the variance of S increases (specifically, as the gap between the weights of the top n items and the remaining items widens), the probability that the random multiplicative noise $U(0, 1)$ causes a "swap" in the ranking decreases.

C.3. A simple counter-example:

With two items having weights $S = [2, 1]$.

Method A : The probability of selecting the first element ($s_1 = 2$) is:

$$P(A = 1) = \frac{2}{2+1} = \frac{2}{3} \approx 0.667$$

Method B : We select index 1 if $s_1 u_1 > s_2 u_2 \implies 2u_1 > 1u_2 \implies u_2 < 2u_1$. Geometrically, this is the area within the unit square $(u_1, u_2) \in [0,1]^2$ satisfying the inequality.

$$P(B = 1) = 1 - \frac{1}{2} \cdot \frac{s_2}{s_1} = 1 - \frac{1}{4} = 0.75$$

(Derivation provided in the proof section).

C.4. Proof:

Let $S = \{s_1, s_2, \ldots, s_N\}$ be a set of positive real-valued scores. Without loss of generality, assume the indices are sorted such that $s_1 > s_2 > \cdots > s_N > 0$. We aim to select a subset of size n, denoted $\mathcal{I}_n$.

Lemma 1 (Pairwise Error Probability). Consider two fixed indices i and j such that $s_i > s_j$. Let $r_{ij} = \frac{s_j}{s_i} < 1$. The probability that the smaller weight element "beats" the larger weight element in Method B is:

$$P(s_j U_j > s_i U_i) = \frac{s_j}{2s_i}$$

Proof. We seek $P(U_i < \frac{s_j}{s_i} U_j)$. Let $k = \frac{s_j}{s_i} \in (0,1)$. Since U_i, U_j are independent uniform variables on $[0,1]$, the joint PDF is 1 over the unit square. The condition $U_i < kU_j$ describes a triangular region in the square defined by vertices $(0,0)$, $(k,1)$, and $(0,1)$ (assuming U_i is x-axis, U_j is y-axis, we want $x < ky$). However, simpler integration:

$$P(U_i < kU_j) = \int_0^1 \int_0^{ku_j} 1 \, du_i \, du_j = \int_0^1 k u_j \, du_j = k \left[\frac{u_j^2}{2} \right]_0^1 = \frac{k}{2}$$

Substituting $k = s_j/s_i$, we get $P(\text{Error}) = \frac{s_j}{2s_i}$.

Definition 1 (Success Event). Let E be the event that Method B selects exactly the set $\mathcal{I}^*$. This occurs if and only if:

$$\min_{i \in \{1,\ldots,n\}} (s_i U_i) > \max_{j \in \{n+1,\ldots,N\}} (s_j U_j)$$

However, a sufficient condition for E is that for every pair (i,j) where $i \leq n$ and $j > n$, we have $s_i U_i > s_j U_j$.

Theorem 1 (Convergence in High Variance). Let the "variance" or separation of the score matrix increase such that for all $i \leq n$ and $j > n$, the ratio $\frac{s_i}{s_j} \to \infty$ (or equivalently $\frac{s_j}{s_i} \to 0$). Then, the probability that Method B is equivalent to Method C approaches 1.

Proof. Let $P(\text{Error})$ be the probability that Method B selects at least one element $j > n$ or fails to select at least one element $i \leq n$. This implies a swap occurred between the set of top n and the set of bottom $N - n$.

$$P(\text{Error}) = P\left(\exists i \leq n, \exists j > n : s_j U_j > s_i U_i\right)$$

By the Union Bound (Boole's inequality):

$$P(\text{Error}) \leq \sum_{i=1}^{n} \sum_{j=n+1}^{N} P(s_j U_j > s_i U_i)$$

Using Lemma 1:

$$P(\text{Error}) \leq \sum_{i=1}^{n} \sum_{j=n+1}^{N} \frac{s_j}{2s_i}$$

$$P(\text{Error}) \leq \frac{1}{2} \sum_{i=1}^{n} \sum_{j=n+1}^{N} \frac{s_j}{s_i}$$

Let $R_{\min} = \min_{i \leq n, j > n} \frac{s_i}{s_j}$. This represents the minimum separation ratio between the selected and non-selected groups. Then $\frac{s_j}{s_i} \leq \frac{1}{R_{\min}}$.

$$P(\text{Error}) \leq \frac{n(N-n)}{2R_{\min}}$$

As the variance of S increases such that the separation between high and low scores grows, $R_{\min} \to \infty$. Consequently:

$$\lim_{R_{\min} \to \infty} P(\text{Error}) = 0$$

Thus, $P(\text{Method B} = \text{Method C}) = 1 - P(\text{Error}) \to 1$.

Comparison with Method A It is worth noting why Method B shows "less randomness" than Method A. For a pairwise comparison $s_i > s_j$:

- Error rate Method A: $\frac{s_j}{s_i + s_j} \approx \frac{s_j}{s_i}$ (for large s_i).
- Error rate Method B: $\frac{s_j}{2s_i}$.

Method B suppresses the probability of selecting lower-weighted items by a factor of 2 compared to Method A in the limit. Thus, Method B is closer to the deterministic Method C.

D. Empirical Results of the Soft-CH2-L3n Stochasticity

To empirically validate the stochastic behavior of soft-CH2-L3n, we monitor the evolution of the Soft-CH2-L3n scores during the pre-training of Llama-60M. Because CHTs24 only updates the topology in the first 75% of the steps, the x-axis has a range of 7500.

Figure 2 demonstrates that the variance of the CH2-L3n scores increases monotonically as training proceeds. This aligns with the formation of local communities within the network, distinguishing high-potential links from redundant ones.

Crucially, this increase in variance triggers an automatic annealing of the exploration rate. As shown in Figure 3, the proportion of blocks affected by stochastic perturbation—where the noisy selection differs from a greedy selection—starts at approximately 10% and decays naturally.

This confirms that CHTs24 successfully transitions from a high-exploration phase in the early stages (preventing premature convergence to suboptimal topologies) to a stable exploitation phase in the late stages (ensuring convergence of weights), all without requiring a manual hyperparameters schedule for stochasticity.

E. Parallel Implementation and Efficiency of CHTs24 Regrowth

In this section, we detail the implementation of the link regrowth phase in CHTs24. We demonstrate how the epitopological scoring and the block-wise pattern optimization are formulated as high-throughput tensor operations, completely avoiding sequential processing of blocks. This formulation ensures that the regrowth process scales efficiently on modern GPU architectures.

E.1. Algorithm Formulation

The regrowth process involves two primary stages: (1) **Global Score Calculation**, where the Cannistraci-Hebb (CH) scores are computed for all potential links simultaneously using matrix algebra, and (2) **Parallel Block-wise Optimization**, where the optimal 2:4 pattern is selected for every block in a single batched operation.

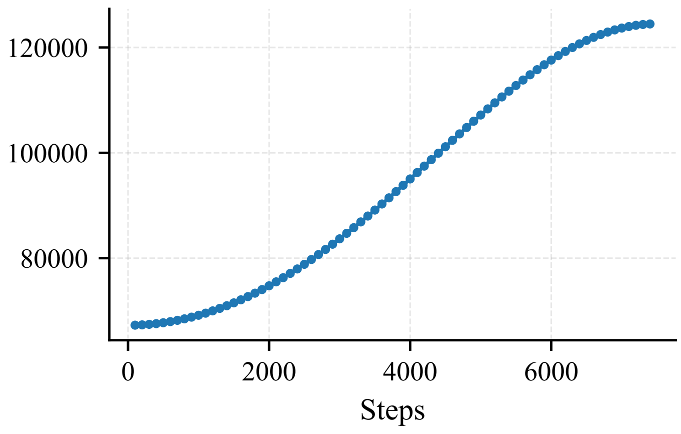


Figure 2: **The variance of the CH score.**

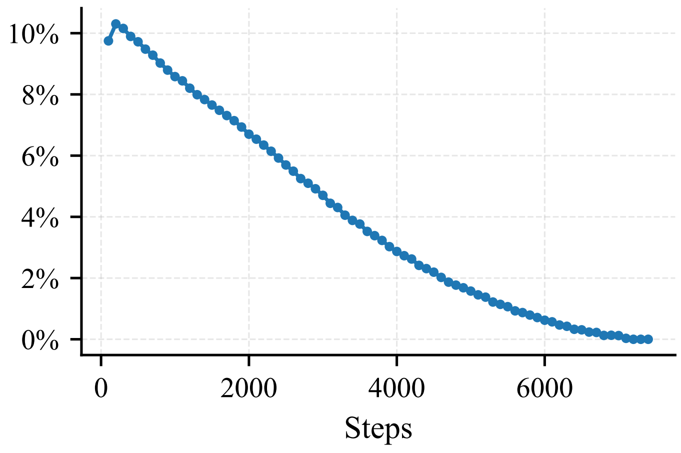


Figure 3: **The proportion of blocks affected by stochasticity.**

We define the binary adjacency matrix (mask) as $\mathbf{M} \in \{0,1\}^{H \times W}$. The library of valid transposable 2:4 patterns is denoted as a tensor $\mathcal{L} \in \{0,1\}^{90 \times 4 \times 4}$. The implementation logic is formalized in Algorithm 1.

E.2. Efficiency Analysis

Standard implementations of block-constrained sparsity often rely on greedy heuristics or iterative CPU-bound loops to enforce constraints on every block. In contrast, CHTs24 leverages the massive parallelism of GPUs. We analyze the computational complexity of the two main stages.

Parallel Score Computation. The calculation of Cannistraci-Hebb scores requires analyzing the connectivity of the network to find Local Community Links (L3 paths). As formalized in Stage 1 of Algorithm 1, this is achieved entirely via dense matrix multiplications (GEMM) and element-wise tensor operations. For example, computing common neighbors $\mathbf{M}\mathbf{M}^T$ and $\mathbf{M}^T\mathbf{M}$ captures the number of paths of length 2. This is a standard $O(N^3)$ matrix multiplication, which is highly optimized on NVIDIA Tensor Cores. Crucially, **no graph traversal algorithms (e.g., BFS/DFS) are used**. The graph topology is treated as a linear algebra problem, allowing the score generation to utilize the full memory bandwidth of the GPU.

Parallel Block-wise Pattern Selection. Stage 3 represents a novel contribution to efficient constrained optimization. Instead of iterating through each 4×4 block (k) sequentially to solve the

Algorithm 1 Parallel Block-wise Pattern Optimization via Epitopological Dynamics

```
Input: Current Mask $\mathbf{M}$, Pattern Library $\mathcal{L} \in \{0,1\}^{90\times4\times4}$.
Hyperparameters: Softness $\xi$ (Boolean flag), Survivor Bias $\Gamma \gg 1$.
Output: Updated Mask $\mathbf{M}_{new}$.

1. Dual-Projection Link Prediction (Row & Column Streams)
Compute Path Matrices (Common Neighbors)
$\mathbf{P}_{row} \leftarrow \mathbf{M} \times \mathbf{M}^T$   and   $\mathbf{P}_{col} \leftarrow \mathbf{M}^T \times \mathbf{M}$
$\mathbf{B}_{row} \leftarrow (\mathbf{P}_{row} \neq 0)$   and   $\mathbf{B}_{col} \leftarrow (\mathbf{P}_{col} \neq 0)$
Calculate External Local Links (Degree Penalties)
$\mathbf{d}_{row} \leftarrow \sum \mathbf{M}$ (row sums),   $\mathbf{d}_{col} \leftarrow \sum \mathbf{M}$ (col sums)
$\mathbf{E}_{row} \leftarrow (\mathbf{d}_{row} - \mathbf{P}_{row}) \odot \mathbf{B}_{row}$
$\mathbf{E}_{col} \leftarrow (\mathbf{d}_{col} - \mathbf{P}_{col}) \odot \mathbf{B}_{col}$
Compute Weighted Projection Matrices (Inverse Link weighting)
$\mathbf{W}_{row} \leftarrow \frac{1}{\mathbf{E}_{row}+1} \odot (\mathbf{P}_{row} + \mathbf{B}_{row})$
$\mathbf{W}_{col} \leftarrow \frac{1}{\mathbf{E}_{col}+1} \odot (\mathbf{P}_{col} + \mathbf{B}_{col})$
Project back to Link Space (L3 Path Generation)
$\mathbf{S}_{row} \leftarrow \mathbf{W}_{row} \times \mathbf{M}$
$\mathbf{S}_{col} \leftarrow \mathbf{W}_{col} \times \mathbf{M}^T$
$\mathbf{S}_{total} \leftarrow \mathbf{S}_{row} + \mathbf{S}_{col}^T$

2. Stochasticity and Survivor Boosting
Uniform random matrix
$\mathbf{S}_{total} \leftarrow \mathbf{S}_{total} \odot \mathcal{U}(0,1)$
Preserve existing links
$\mathbf{S}_{total}[\mathbf{M}=1] \leftarrow \mathbf{S}_{total}[\mathbf{M}=1] + \Gamma$

3. Parallel Block-wise Pattern Selection
Reshape to Block-Grid Tensor $\mathcal{T}$
$H, W \leftarrow$ shape($\mathbf{S}_{total}$)
$\mathcal{T} \leftarrow$ Permute(Reshape($\mathbf{S}_{total}, (H/4, 4, W/4, 4)$), $(0,2,1,3)$)
Broadcasting Dot-Product against Library $\mathcal{L}$
$\mathcal{F} \leftarrow \sum_{x,y}(\mathcal{T}$.unsqueeze$(2) \odot \mathcal{L}$.unsqueeze$(0)$.unsqueeze$(0))$
Select Top Pattern per Block
$\mathbf{I}^* \leftarrow$ argmax($\mathcal{F}$, dim $= 2$)
$\mathbf{M}_{blocks} \leftarrow \mathcal{L}[\mathbf{I}^*]$
$\mathbf{M}_{new} \leftarrow$ Reshape($\mathbf{M}_{blocks}, (H, W)$)
```

optimization problem:

$$\mathbf{M}_{new}^{(k)} = \underset{\mathbf{P}\in\mathcal{P}}{\arg\max} \left(\sum_{x,y} \hat{S}_{xy}^{(k)} \cdot P_{xy} \right) \tag{2}$$

We utilize **Tensor Broadcasting** to solve this for all blocks simultaneously.

- **Reshape and Permute:** The global score matrix $\mathbf{S}$ is reshaped into a tensor of shape $(B_h, B_w, 4, 4)$.
- **Batched Dot Product:** We compute the dot product between *every* block in the network and *every* pattern in the library $\mathcal{P}$ simultaneously using tensor broadcasting. The operation corresponds to expanding the dimensions of the block tensor and the library tensor to perform element-wise multiplication followed by summation.
- **Complexity:** Let N_{params} be the total number of weights in the layer. The complexity of the selection step is $O(N_{params} \cdot |\mathcal{P}|)$. Since the library size $|\mathcal{P}|$ is a small constant (90), the complexity is effectively linear $O(N_{params})$.

In our implementation, this avoids Python-level loops entirely. For a standard linear layer (e.g., 4096×4096), processing 10^6 blocks serially would be prohibitive. By vectorizing this into a tensor reduction, CHTs24 achieves the theoretical speedup required for training large-scale LLMs, ensuring the regrowth overhead remains negligible compared to the forward/backward pass.

F. Clarification of Terminology and Relationship to Prior Work

To ensure a straightforward presentation and bridge the gap between our contributions and previous literature, this section simplifies the specific technical terminology used in this paper (e.g., epitopology, manifolds, Fréchet perturbation) and delineates the specific evolutionary relationship between the proposed **CHTs24** and its predecessors, **CHT** and **CHTs**.

F.1. De-mystifying the Terminology

While grounded in network science and stochastic optimization, the core concepts of our framework map directly to standard deep learning and dynamic sparse training (DST) principles:

- **Epitopological Learning (EL):** In simple terms, this refers to *Topology-Driven Learning*. Unlike standard training which updates weights (synaptic strength), EL updates the binary connectivity mask (network shape). In the context of this paper, it is synonymous with the "Regrow" phase of Dynamic Sparse Training, but guided by a specific link prediction rule (Cannistraci-Hebb) rather than gradient magnitudes (as in RigL).
- **Cannistraci-Hebb (CH) Automata:** This is the specific *Link Predictor* used to decide which connections to grow. It is a heuristic that says: "If two neurons share many neighbors that interact with each other, they should be connected." It replaces the gradient-based regrowth used in methods like RigL.
- **Manifold of Transposable 2:4 Patterns:** This simply refers to the *Constraint Space*. In standard pruning, any weight can be zero. In N:M sparsity, we are restricted to a specific set of valid patterns (e.g., in a 4×4 block, rows and columns must sum to 2). We term this valid set of patterns a "manifold" because the optimization process is constrained to move only within this specific discrete set of valid configurations, rather than the entire space of possible sparse masks.
- **Fréchet Perturbation (Modulating Weights):** This is a *Soft Sampling Trick*. Instead of greedily keeping the largest weights (which gets stuck in local minima) or using slow Python loops to sample probabilistically, we add a specific type of noise (Fréchet/Inverse Weibull) to the weights. Mathematically, selecting the maximum value after adding this noise is equivalent to sampling from the distribution of weights. It is the continuous analog of the "Gumbel-Max trick" used in discrete classifications.

F.2. Relationship to Prior Work: From CHT to CHTs to CHTs24

The proposed **CHTs24** is the third stage in the evolution of Cannistraci-Hebb Training. The relationship and specific technical jumps between these works are summarized below and in Table 4.

1. **CHT (Cannistraci-Hebb Training):** The original framework introduced the idea of using the CH-score for regrowth.
    - *Limitation:* It relied on "Hard" selection (greedy top-k), causing it to get stuck in local optima. It used unstructured sparsity, offering no wall-clock speedup on GPUs.
2. **CHTs (CHT Soft Rule):** Addressed the "Hard" selection problem.
    - *Innovation:* Introduced "Soft" probabilistic sampling to allow exploration of connections.

- *Limitation:* Still relied on unstructured sparsity. While theoretically efficient, it could not utilize NVIDIA Sparse Tensor Cores, meaning it offered theoretical FLOPs reduction but no real-world training acceleration.

3. **CHTs24 (Ours):** Bridges the gap between CHT logic and Hardware constraints.

   - *Innovation 1 (Structured Constraints):* We replace global unstructured sorting with **Block-wise Pattern Optimization**. Instead of ranking individual weights, we rank valid 2:4 patterns (the "atomic topologies").
   - *Innovation 2 (Hardware Efficiency):* By enforcing the 2:4 (or 1:4) pattern strictly, CHTs24 allows the use of Sparse Tensor Cores for both forward and backward passes, achieving actual wall-clock speedup.
   - *Innovation 3 (Fast Sampling):* We replace the computationally expensive multinomial sampling of CHTs with the deterministic Fréchet perturbation, allowing for parallelized GPU execution of the "Soft" rule.

Table 4: Comparison of Cannistraci-Hebb Training variants. CHTs24 is the first to achieve hardware-acceleration compatibility.

Method	Sparsity Type	Regrowth Strategy	Weight Maintenance	Hardware Acceleration
CHT (Prior Work)	Unstructured	Deterministic (Greedy Top-K)	Sparse	No (Theoretical FLOPs only)
CHTs (Prior Work)	Unstructured	Probabilistic (Multinomial)	Sparse	No (Theoretical FLOPs only)
SR-STE (Baseline)	N:M Structured	Gradient-based (STE)	**Dense**	Yes (Forward/Backward)
CHTs24 (Ours)	**N:M Structured**	**Probabilistic (Fréchet Noise)**	**Sparse**	**Yes (Forward/Backward)**

G. Pre-Training Experiment Settings

We compare our method against two state-of-the-art 2:4 training methods. To ensure a rigorous and fair comparison, we adapt all baselines to strictly enforce transposable 2:4 sparsity, as standard unstructured implementations are incompatible with the hardware constraints we target. The information of baselinse is listed here:

STE (Transposable)[19]: STE maintains a dense weight w. Before each forward, STE selects the mask with the largest absolute value of weight for each 4×4 block, and then performs the process with this mask. During backward process, STE updates the dense weights according to the following formula.

$$\nabla_w \mathcal{L}(\tilde{w}) \leftarrow \nabla_{\tilde{w}} \mathcal{L}(\tilde{w})$$

SR-STE (Transposable)[21]: SR-STE also maintains a dense weight w. Before each forward, SR-STE selects the mask with the largest absolute value of weight for each 4×4 block, and then performs the process with this mask. During backward process, SR-STE updates the dense weights according to the following formula. This is because only a subset of the weights in a layer are involved in the forward computation, while all weights are updated with gradients during backward process. This implies that the gradients assigned to the masked weights may be imprecise. To address this issue, SR-STE applies regularization to the weights that do not participate in the forward pass.

$$\mathbf{w}_t \leftarrow \mathbf{w}_{t-1} - \gamma \left(\nabla_{\mathbf{w}} \mathcal{L}_t(\tilde{\mathbf{w}}_{t-1}) + \lambda_W \left[\mathbf{m}(\mathbf{w}_{t-1}) \right] \odot \mathbf{w}_{t-1} \right)$$

We report the perplexity (PPL) on the validation set. Lower PPL indicates better performance.

Table 5 shows the experiment settings of pre-training.

Table 5: **The experiment settings of pre-training.**

Parameter	Value
Model	Llama-60m
Dataset	OpenWebText
Dataset Seed	42
Max Length	256
Batch Size	64
Total Batch Size	256
Optimizer	Adam
Learning Rate	0.001
Steps	10000
Warmup Steps	1000
Seeds	0, 1, 2
Zeta	0.1
Update Interval	100
Iterative Warmup Steps	10
Start T	1
End T	9

H. Convergence Analysis

To ensure that the dynamic topological evolution introduced in CHTs24 does not impose an overhead on the convergence of training duration, we analyze the convergence behavior of our method compared to the strong baseline, SR-STE. A primary concern in dynamic sparse training (DST) is whether the continuous rewiring of connections delays the model's ability to settle into a loss minimum.

We compare the validation Perplexity (PPL) curves over the course of pre-training on the Llama-60M model. Figure 4 visualizes the training dynamics for both the transposable 2:4 sparsity pattern and the transposable 1:4 sparsity pattern.

2:4 Sparsity (Figure 4a): Under the standard transposable 2:4 constraint, CHTs24 exhibits a convergence trajectory that is highly competitive with SR-STE. The training curves overlap significantly, demonstrating that CHTs24 does not require additional training steps to reach convergence. The brain-inspired topology evolution operates efficiently alongside weight updates, maintaining stability throughout the optimization process.

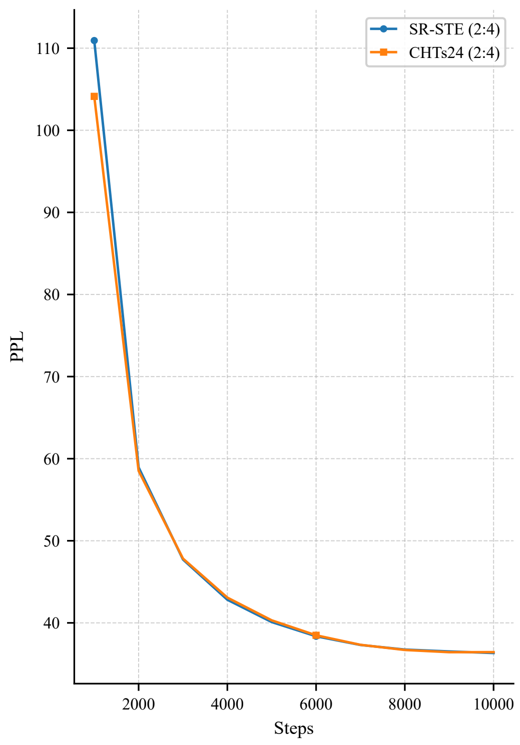
1:4 Sparsity (Figure 4b): The efficiency of our method becomes even more pronounced under the 1:4 sparsity constraint. As shown in Figure 4b, CHTs14 converges significantly faster than SR-STE. A clear gap emerges, with CHTs14 achieving lower perplexity scores much earlier in the training regime. This indicates that the epitopological guidance provided by the Cannistraci-Hebb framework is still effective when valid connections are more scarce. In contrast, gradient-estimation methods (STE-based method) struggle with high sparsity.

In conclusion, CHTs24 and CHTs14 do not incur a convergence penalty. Conversely, in high-sparsity scenarios (1:4), our method makes a faster convergence than STE-based method, offering both training efficiency and superior final performance.

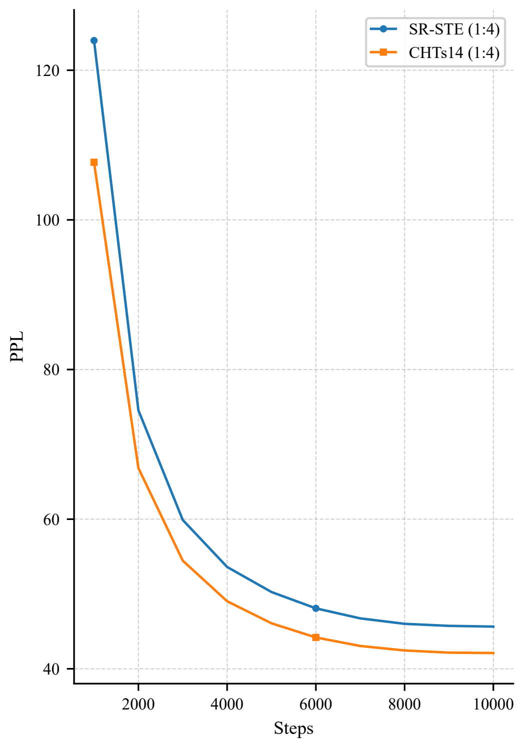
I. Pipeline Analysis

A critical consideration for dynamic sparse training methods is whether the topological evolution steps introduce computational bottlenecks. To empirically check for potential pipeline stalls, we conducted a rigorous performance analysis using the PyTorch Profiler on an NVIDIA A100 GPU.

Time Overhead Analysis: We captured a profiling window of 5 training steps that included a full regrowth phase. In order to objectively reflect the situation of the time cost of regrowth, we make



(a) **2:4 sparsity**



(b) **1:4 sparsity**

Figure 4: The line graph comparing the PPL convergence of our method and SR-STE under different semi-structured sparsity.

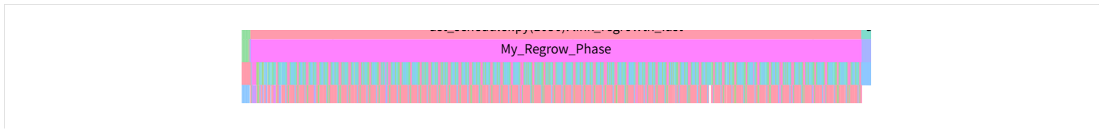
reasonable adjustments to the code. The adjustments involve removing the if branch, eliminating the indicator statistics, and removing logging operations, etc. The total duration of this window was 4794 ms. Within this period, the specific *My Regrow Phase* (which includes the score computation and pattern matching) consumed only 254 ms. When amortized over the entire training process (where regrowth occurs only at specific intervals t), the regrowth operation accounts for approximately 0.3% of the total training time. This confirms that the batched dot product optimization is computationally inexpensive relative to the forward and backward passes of the model.

Pipeline Stall Analysis : We analyzed the execution traces for both the CPU and GPU to identify synchronization barriers:

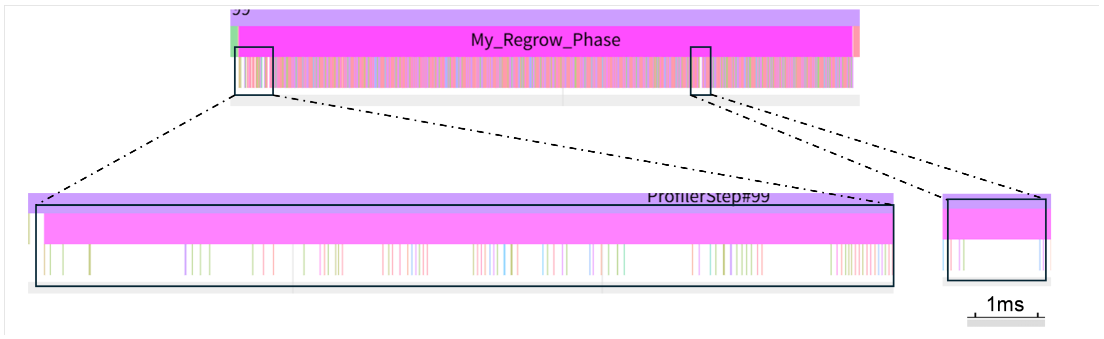
- CPU Pipeline (Figure 5a): The CPU trace shows a continuous stream of instruction dispatching throughout the regrowth phase. There are no significant gaps, indicating that the CPU is not blocking while waiting for GPU synchronization, allowing for efficient asynchronous execution.

- GPU Pipeline (Figure 5b): The GPU trace reveals a dense packing of kernels. We detected only 1 stall of 2 milliseconds and 6 stalls of 1 millisecond during the regrowth process. These stalls cumulatively account for approximately 8 ms, representing less than 3% of the regrowth operation's duration.

These results, visualized in Figure 5, demonstrate that the CHTs24 regrowth mechanism is performed on-the-fly and is highly parallelizable. The block-wise pattern selection aligns well with GPU architecture, causing negligible pipeline stalls and no effective throughput degradation.



(a) **CPU Pipeline**



(b) **GPU Pipeline**

Figure 5: Pipeline analysis for CHTs24 regrowth process.

J. Computational Complexity Analysis: FLOPs Decomposition

In this section, we provide a detailed decomposition of the floating-point operations (FLOPs) required for training the Llama-60M architecture. We compare three training paradigms: (1) Standard Dense Training, (2) SR-STE with transposable 2:4 sparsity, and (3) CHTs24 with transposable 2:4 sparsity.

J.1. Model Architecture and Parameters

The Llama-60M model consists of $L = 8$ decoder layers, a hidden dimension $h = 512$, an intermediate dimension $i = 1280$, and a vocabulary size $V = 32,000$. The sparsity masks are applied exclusively to the Linear layers within the Transformer blocks (Attention and MLP), excluding the embedding layer and the language model head.

We categorize the parameters into two groups: *Sparse-able Parameters* (θ_{sparse}) and *Fixed Dense Parameters* (θ_{dense}).

$$\theta_{sparse}^{(layer)} = \underbrace{4h^2}_{\text{Q,K,V,O}} + \underbrace{3hi}_{\text{Gate, Up, Down}} \tag{3}$$

Substituting the dimensions for Llama-60M:

$$\begin{aligned}\theta_{sparse}^{(layer)} &= 4(512)^2 + 3(512)(1280) \\ &= 1,048,576 + 1,966,080 = 3,014,656\end{aligned}$$

Total sparse-able parameters: $\Theta_{sparse} = 8 \times 3,014,656 \approx 24.12$ M.

The fixed dense parameters (dominated by language model head) are:

$$\Theta_{dense} = V \times h = 32,000 \times 512 \approx 16.38 \text{ M} \tag{4}$$

Note: We exclude the Embedding layer ($V \times h$) from the FLOPs calculation as it involves lookup operations rather than matrix multiplications, and its gradient update cost is negligible compared to dense layers.

J.2. FLOPs Formulation per Training Step

Let D be the total number of tokens in a training batch ($B \times T$). The standard FLOPs for a Matrix Multiplication (MatMul) of inputs $X \in \mathbb{R}^{D \times C_{in}}$ and weights $W \in \mathbb{R}^{C_{in} \times C_{out}}$ are $2 \cdot D \cdot C_{in} \cdot C_{out}$.

Training involves three primary passes for each layer:

- **Forward Pass (F_{fwd}):** Computing $Y = XW$.
- **Backward Input (F_{din}):** Computing $\nabla X = \nabla Y W^T$.
- **Backward Weight (F_{dw}):** Computing $\nabla W = X^T \nabla Y$.

J.2.1. Dense Training

For dense training, all three passes utilize dense matrices.

$$\mathcal{C}_{dense} = \underbrace{1}_{\text{Fwd}} + \underbrace{1}_{\text{Bwd-In}} + \underbrace{1}_{\text{Bwd-W}} = 3 \times (2D\Theta) \tag{5}$$

J.2.2. SR-STE (Transposable 2:4)

SR-STE utilizes N:M sparse tensor cores for the forward pass and the backward input pass (using the sparse weight matrix $\tilde{W}$). However, to maintain the Straight-Through Estimator logic, it calculates gradients for the *full dense* weight matrix ∇W to update the latent dense weights.

- Forward: Sparse ($0.5\times$ cost due to 2:4 sparsity).
- Backward Input: Sparse ($0.5\times$ cost due to 2:4 sparsity).
- Backward Weight: Dense ($1.0\times$ cost, as gradients are computed for all elements).

$$\mathcal{C}_{SR-STE} = (0.5 + 0.5 + 1.0) \times (2D\Theta_{sparse}) + \mathcal{C}_{dense}(\Theta_{dense}) \tag{6}$$

J.2.3. CHTs24 (Transposable 2:4)

CHTs24 performs "True Sparse Training". It does not maintain dense weights. Gradients are only computed for the active 2:4 connections.

- Forward: Sparse ($0.5\times$ cost).
- Backward Input: Sparse ($0.5\times$ cost).
- Backward Weight: Sparse ($0.5\times$ cost, gradients only for active connections).

Additionally, CHTs24 incurs a topological update cost $\mathcal{C}_{topo}$ every $\Delta t = 100$ steps.

Based on the algorithm code (ch2-l3n), the topology update involves four matrix multiplications per layer (computing WW^T, W^TW, and two projections). For a layer with weight shape $M \times N$, the $\mathcal{C}_{topo}$ is no larger than $\mathcal{C}_{fwd\_sparse}$:

$$k = \frac{\mathcal{C}_{layer\_topo}}{\mathcal{C}_{fwd\_sparse}} = \frac{2MN(M+N)}{DMN} = \frac{2(M+N)}{D} \tag{7}$$

We estimate $\mathcal{C}_{topo} \approx \mathcal{C}_{fwd\_sparse}$ (conservative upper bound).

$$\mathcal{C}_{CHTs24} = (0.5 + 0.5 + 0.5) \times (2D\Theta_{sparse}) + \frac{1}{\Delta t}\mathcal{C}_{topo} + \mathcal{C}_{dense}(\Theta_{dense}) \tag{8}$$

J.3. Numerical Comparison on Llama-60M

Table 6 presents the FLOPs consumption normalized by the number of tokens D.

Analysis:

1. **Gradient Bottleneck in SR-STE:** While SR-STE utilizes sparse tensor cores for acceleration during the forward pass and backward propagation, the requirement to update dense weights forces the calculation of dense gradients. This results in the sparse-able layers consuming $2/3$ of the dense FLOPs, limiting maximum theoretical speedup.

Table 6: FLOPs Breakdown for Llama-60M (Units: MegaFLOPs per token). The comparison highlights the efficiency of CHTs24 in the sparse-able layers compared to SR-STE and Dense baselines. Note that fixed dense layers (Head) dilute the total theoretical acceleration for this specific model size.

Component	**Metric**	**Dense**	**SR-STE**	**CHTs24**
Sparse-able Layers (Attention & MLP)				
Params (Θ_{sparse})	Count	24.12 M	24.12 M	12.06 M (Active)
Fwd Pass	Ratio	$1.0\times$	$0.5\times$	$0.5\times$
Bwd Pass (∇X)	Ratio	$1.0\times$	$0.5\times$	$0.5\times$
Bwd Pass (∇W)	Ratio	$1.0\times$	**$1.0\times$**	**$0.5\times$**
Subtotal FLOPs	MFLOPs/tok	**144.72**	**96.48**	**72.36**
Fixed Dense Layers (lm_head)				
Params (Θ_{dense})	Count	16.38 M	16.38 M	16.38 M
Subtotal FLOPs	MFLOPs/tok	**98.28**	**98.28**	**98.28**
Topology Overhead (amortized $\Delta t = 100$)				
ch2-l3n Update	MFLOPs/tok	0	0	$\approx \mathbf{0.24}$
Total FLOPs	**MFLOPs/tok**	**243.00**	**194.76**	**170.88**
Overall Savings	vs Dense	-	**19.85%**	**29.68%**
Sparse-Layer Savings	vs Dense	-	**33.33%**	**50.00%**

2. **True Sparsity in CHTs24:** CHTs24 eliminates the dense gradient calculation, reducing the operation count for sparse-able layers to exactly 50% of the dense baseline. Even with the periodic overhead of epitopological updates (which adds $< 0.2\%$ overhead amortized over 100 steps), CHTs24 is significantly more efficient.
3. **Impact of Model Scale:** For the Llama-60M model, the language model head constitutes a disproportionately large fraction of total computations ($\sim 40\%$). Consequently, the global FLOPs reduction is dampened. In larger models (e.g., Llama-7B) where $\Theta_{sparse} \gg \Theta_{dense}$, the total savings for CHTs24 would asymptotically approach 50%.

K. Re-Training Experiment Settings

We compare our CHTs24 for eDSrT against two state-of-the-art retraining methods. To ensure a rigorous and fair comparison, we adapt all baselines to strictly enforce transposable 2:4 sparsity, as standard unstructured implementations are incompatible with the hardware constraints we target. The information of baselinse are list here:

Nvidia (Transposable)[13]. : A streamlined three-step workflow converts a dense model into a 2:4 sparse model, enabling inference acceleration while retaining near-identical accuracy:

- Train: Optimize a standard dense model to convergence.
- Prune: Apply amplitude-based pruning to pre-trained weights, enforcing the 2:4 structured sparsity pattern (2 zeros per 4 consecutive elements).
- Retrain (Fine-tune): Fix the sparse mask and retrain non-zero weights with original hyperparameters to recover pruning-induced accuracy loss.

Table 7 shows the experiment settings of re-training.

Table 7: **The experiment settings of re-training.**

Parameter	**Value**
Model	ViT_deit_base_patch16_224
Dataset	ImageNet
Batch Size	64
Total Batch Size	256
Optimizer	AdamW
Learning Rate	5.00E-04
Weight Decay	5.00E-02
Drop Path	1.00E-01
Epochs	100
Seeds	0, 1
Zeta	0.1
Iterative Warmup Steps	10
Start T	1
End T	3