Sparse-IFT: Sparse Iso-FLOP Transformations for Maximizing Training Efficiency

Vithursan Thangarasa [*1] **Shreyas Saxena** [*†] **Abhay Gupta** [†] **Sean Lie** [1]

Abstract

Recent research has focused on weight sparsity in deep neural network training to reduce FLOPs, aiming for improved efficiency (test accuracy w.r.t training FLOPs). However, sparse weight training often compromises accuracy, requiring extended training schedules to attain the accuracy of dense models. In contrast, our approach, Sparse Iso-FLOP Transformations (Sparse-IFT), uses sparsity to *improve accuracy* while maintaining dense model FLOPs. Using a single hyperparameter (i.e., the sparsity level), Sparse-IFTs efficiently replace dense layers, expanding the search space for optimal sparse masks. In addition, dynamic sparse training (DST) with Sparse-IFT models effectively navigate this larger sparse mask-weight space, which is evidenced by a spectral analysis using Ramanujan graph properties. Our study reveals a robust correlation among mask topology, weights, and final performance. Notably, without adjusting any training hyperparameters, replacing dense layers with Sparse-IFT yields significant improvements, such as a +3.5% boost for ResNet-18 on ImageNet and +0.9% for GPT-3 Small on the Open LLM leaderboard. To the best of our knowledge, this is the first work to demonstrate the use of sparsity for improving the accuracy of dense models through a set of simple-to-use sparse transformations. Code is available at: https://github.com/CerebrasResearch/Sparse-IFT.

1. Introduction

Increases in model size and training data have led to many breakthroughs in deep learning (e.g., AlexNet (Krizhevsky et al., 2012), ResNet (He et al., 2016), Transformers (Vaswani et al., 2017), GPT (Radford et al., 2018; 2019), AlphaGo (Silver et al., 2017), etc.). Consequently, computational and memory demands for training and deploying deep neural networks (DNNs) have surged dramatically. To enable the deployment of large models, multiple techniques (e.g., distillation (Hinton et al., 2015), quantization (Han et al., 2015a), pruning (Han et al., 2015b)) have been introduced to reduce inference FLOPs and memory requirements. While these techniques improve inference efficiency (test accuracy w.r.t inference FLOPs), the associated training costs are still prohibitive. Our work focuses on improving the training efficiency (test-accuracy w.r.t training FLOPs) of DNNs through weight sparsity. In recent years, we have witnessed progress in using weight sparsity to reduce training FLOPs (Evci et al., 2020; Liu et al., 2021a; Jayakumar et al., 2020). Frankle & Carbin (2018) show that sparse subnetworks ("lottery tickets") exist at initialization and can be trained to match dense network accuracy. Dynamic sparse training (DST) methods (Ma et al., 2022; Evci et al., 2020; Liu et al., 2021b; Jayakumar et al., 2020) iteratively adjust sparsity patterns to facilitate the discovery of optimal sparse subnetworks within a single training run. However, they often lag behind dense baselines or require longer training schedules (e.g., 2-5x training steps) to close the gap (Yuan et al., 2021; Tai et al., 2022; Liu et al., 2021a). Our unique contribution focuses on using sparsity to improve a given dense model's accuracy. We introduce the Sparse Iso-FLOP

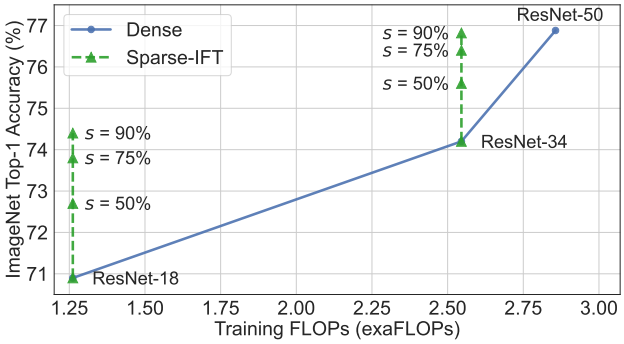

Figure 1: Top-1 Accuracy vs. Training FLOPs for variants of ResNet on ImageNet. Sparse-IFT provides significant accuracy gains across different models and sparsity levels, $s \in \{50\%, 75\%, 90\%\}$, while using the same training FLOPs as its dense counterpart.

[*]Equal contribution [1]Cerebras Systems Inc, California, USA, [†]Work done while at Cerebras. Correspondence to: Vithursan Thangarasa <vithu@cerebras.net>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

Transformations (Sparse-IFT), a family of techniques serving as drop-in replacements for dense layers in DNNs.

Sparse-IFTs increase layer representational capacity, facilitating the discovery of optimal sparse subnetworks while maintaining constant FLOPs (i.e., Iso-FLOP). For example, widening a layer with maintained sparsity increases dimensionality without impacting FLOPs; expanding the sparse mask-weight space for more diverse configurations. This enables DST methods to navigate the search space effectively, potentially finding improved sparse subnetworks for higher accuracy. Drawing inspiration from prior works (Hoang et al., 2023b;a), we analyze the connectivity of Sparse-IFT models as Ramanujan graphs and their impact on performance when trained with DST. All Sparse-IFTs are parameterized by a single hyperparameter, the sparsity level. Figure 1 summarizes ImageNet performance, showing significant accuracy gains with Sparse Wide IFT ResNet variants. Sparse Wide ResNet-18 achieves +3.5% top-1 accuracy at 90% sparsity, surpassing a dense ResNet-34 (74.2%) with 2x fewer FLOPs. These gains result from replacing dense layers with Sparse-IFTs, requiring no changes to training hyperparameters. The main contributions of our work are:

1. We introduce Sparse Iso-FLOP Transformations (Sparse-IFTs), a family of techniques aimed at enhancing DNN training efficiency. These transformations boost accuracy while maintaining a constant FLOP count. Sparse-IFTs are parameterized by a *single hyperparameter, sparsity level*, and can be seamlessly used as drop-in replacements for dense layers.

2. We empirically validate the consistent advantage of DST over static sparse training for Sparse-IFT networks. Our investigation into the dynamic evolution of sparse topologies in DST via Ramanujan graph spectral analysis highlights optimized connectivity patterns and improved spectral characteristics.

3. We show consistent benefits of Sparse-IFT across computer vision and natural language processing domains. Sparse-IFT enhances ResNet-18 and ResNet-34 top-1 accuracy on ImageNet by 3.5% and 2.6%, respectively. Fine-tuning for object detection (MS COCO) and segmentation (CityScapes) yields improvements of 5.2% mAP and 2.4% mIoU. Sparse-IFT with GPT-3 results in a 0.9% improvement on the Open LLM leaderboard.

4. We showcase the practical value of Sparse-IFT with real-world timings for training on the Cerebras CS-2 (Lie, 2023) and inference with Neural Magic DeepSparse (NeuralMagic, 2021) using unstructured sparsity. Despite being 2x wider at 75% sparsity with Sparse Wide IFT, we observe minimal compute overhead on both platforms compared to GPUs.

2. Method

In this section, we first explain our intuition and hypotheses, followed by our methodology to improve training efficiency.

Training with Dense Matrices is FLOP Inefficient Modern DNNs are often overparameterized, showing sparsity in features and weights across layers. The Lottery Ticket Hypothesis (Frankle & Carbin, 2018; Chen et al., 2020) suggests sparse DNNs, initialized with an effective sparsity mask ("lottery ticket"), can achieve the same accuracy as dense counterparts. Sparse training methods theoretically enhance efficiency but often yield lower accuracy than dense baselines. This discrepancy may stem from challenges in identifying optimal masks within a single training run. Existing sparse training methods (Jayakumar et al., 2020; Evci et al., 2020; Yuan et al., 2021; Tai et al., 2022; Liu et al., 2021a) invest these FLOP savings into longer training schedules to bridge accuracy gaps, inefficiently requiring more FLOPs than dense baselines for the same target accuracy.

In our work, we take an orthogonal approach and invest these FLOP savings to (1) enhance a layer's representational capacity and (2) expand its search space, aiming to discover an optimal sparse mask (Ramanujan et al., 2020; Stosic & Stosic, 2021). Larger sparse models show potential for improved accuracy, but designing an appropriate architecture is challenging. For instance, achieving performance surpassing ResNet-18 on ImageNet requires careful balance of sparsity and network size. Existing studies explore diverse combinations but often lack FLOP efficiency, requiring multiple iterations for optimal settings and hyperparameter tuning. To address this, we propose the Sparse Iso-FLOP Transformation (Sparse-IFT) family, replacing dense layers with FLOP-equivalent sparse transformations. Notably, Sparse-IFT is parameterized by a single hyperparameter—the sparsity level, simplifying the tuning process.

2.1. Sparse Iso-FLOP Transformations

Setup For clarity, we explain our method in the context of a fully connected network. Let $\mathcal{N}$ denote a L layered DNN parameterized by $\Theta_{\mathcal{N}}$. Let $\Theta_{\mathcal{N}} \in \{\theta_1, \ldots, \theta_L\}$ denote the parameters of the DNN. The output of the l-th layer is defined as: $z_l = \sigma(f_{\theta_l}(z_{l-1}))$ for some activation function σ (e.g., ReLU (Nair & Hinton, 2010)) and feedforward function f_{θ_l}. Specifically, let $f_{\theta_l}(z_{l-1}) = \theta_l^T z_{l-1}$, where $\theta_l \in \mathbb{R}^{D_{in} \times D_{out}}$, $z_{l-1} \in \mathbb{R}^{D_{in} \times B}$ and B, D_{in}, D_{out} denote the batch-size, input, and output dimensionality of features respectively. The total FLOPs needed for f_{θ_l} are given by $B \cdot D_{in} \cdot D_{out}$. In Appendix A.1, we detail a straightforward extension to convolutional layers. In the standard setup, the feedforward function f_{θ_l} computes output features through a linear transformation of input features. While theoretically, arbitrary non-linear transformations can be applied, practical implementations often resort to expressing transfor-

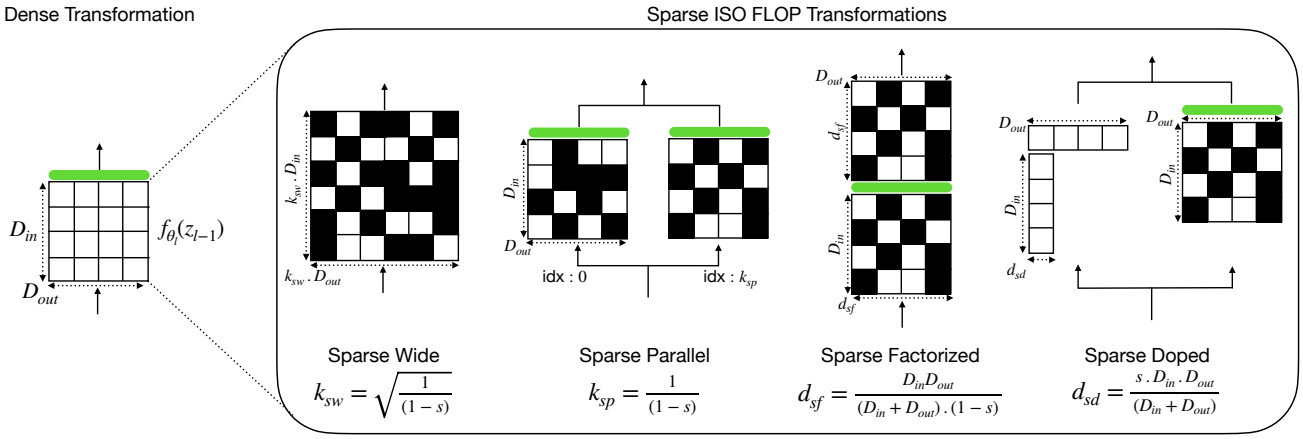

Figure 2: Different members of the Sparse-IFT family, each parameterized by a single hyperparameter (i.e., sparsity level, s). Black and white squares denote non-active and active weights, respectively. Green block indicates a non-linear activation function (e.g., ReLU). Derived with sparsity set at 50% as an example, all transformations are Iso-FLOP to the dense feedforward function f_{θ_l}, making them suitable drop-in replacements for f_{θ_l}. Details about each member are in Section 2.2.

mations as dense matrix multiplications for efficient GPU support (Nvidia, 2023). We aim to boost DNN training efficiency by enhancing the representational capacity of the feedforward function. Unlike conventional methods that increase capacity by stacking more layers (Lin et al., 2014a), widening (Zagoruyko & Komodakis, 2016), or ensembling (Littwin et al., 2020), our approach introduces unstructured sparsity in weight matrices, achieving the same FLOPs as a dense feedforward function.

Let Ψ_l denote the set of Sparse Iso-FLOP Transformations (Sparse-IFT) for a particular layer l:

$$\Psi_l : \{\psi_l(s), 0 \leq s < 1, g(\psi_l) \approx g(f_{\theta_l})\},$$

where ψ_l is a transformation, s represents the sparsity level, and $g(\cdot)$ returns the computational FLOPs. Each transformation in this set satisfies the following properties: (1) the computational FLOPs of the transformation ψ_l are same as that of dense transformation f_{θ_l}, and (2) the transformation is parameterized by a single hyperparameter - the sparsity level. These Iso-FLOP transformations serve as drop-in replacements for dense feedforward functions, preserving layer FLOPs. While there may be other FLOP-invariant transformations, in this work, we explore: Sparse Wide, Sparse Parallel, Sparse Factorized, and Sparse Doped.

2.2. Members of Sparse-IFT

Sparse Wide This transformation augments the representational capacity of a layer by increasing the number of output features while keeping s fraction of weights sparse. Hence, it widens the input and output features for all L layers of the network with the same widening factor, k_{sw}, to avoid mismatch in feature dimensionality across layers. Let $\theta_l^{sw} \in \mathbb{R}^{k_{sw} \cdot D_{in} \times k_{sw} \cdot D_{out}}$ denote the transformation matrix, with s fraction of weights being sparse. Since the fraction of non-sparse weights is given by $1 - s$, the FLOPs required by this transformation are $B \cdot (k_{sw} \cdot D_{in}) \cdot (k_{sw} \cdot D_{out}) \cdot (1 - s)$. Setting these equal to the FLOPs of the original dense f_{θ_l}, we obtain the widening factor $k_{sw} = \sqrt{1/(1-s)}$. If we set the sparsity s to 0, we obtain k_{sw} as 1 and recover the dense feedforward function.

Sparse Parallel The sparse parallel transformation replaces the feedforward function with a sum of k_{sp} non-linear functions. Let $\theta_l^{sp} \in \{\theta_l^{sp,1}, \ldots, \theta_l^{sp,k_{sp}}\}$ denote the parameters of this transformation, where $\theta_l^{sp,j} \in \mathbb{R}^{D_{in} \times D_{out}}$ denotes the transformation matrix of jth function, where s fraction of weights are sparse. The sparse parallel transformation in this case is $\psi_l^{sp} = \sum_{j=1}^{k_{sp}} \sigma((\theta_l^{sp,j})^T z_l)$, where σ is a non linear function. In practice, ψ_l^{sp} is implemented as a layer with k_{sp} parallel branches. The computational FLOPs of this transformation is $k_{sp} \cdot B \cdot D_{in} \cdot D_{out} \cdot (1 - s)$. Setting these FLOPs equal to FLOPs of f_θ, we obtain $k_{sp} = 1/(1 - s)$. Note, at $s = 0$, the number of parallel branches k_{sp} is 1. If we replace σ with Identity, we can recover the original dense feedforward function.

Sparse Factorized The transformation matrix of the feedforward function f_{θ_l} is denoted by $\theta_l \in \mathbb{R}^{D_{in} \times D_{out}}$. Multiple works have explored matrix factorization techniques to express the transformation matrix θ_l as a product of two matrices $\theta_l = UV^T$, where $U \in \mathbb{R}^{D_{in} \times d}$, $V \in \mathbb{R}^{D_{out} \times d}$. Khodak et al. (2020); Tai et al. (2016) and Chen et al. (2021b) have explored low-rank factorization ($d << D_{out}$) as a form of structured sparsity to improve training and inference efficiency, while Arora et al. (2018) and Guo et al. (2020a) have explored overparameterized factorizations for better generalization and faster convergence. In contrast, we use factorization to augment the representational capacity without decreasing or increasing the FLOPs. More precisely, let $\theta_l^{sf} \in \{U_l, V_l\}$ denote the parameters of this transformation, where $U_l \in \mathbb{R}^{D_{in} \times d_{sf}}$, $V_l \in \mathbb{R}^{d_{sf} \times D_{out}}$ are sparse matrices with s fraction of their weights being

Table 1: Cardinality of search space for sparsity masks of different members of the Sparse-IFT family.

Transformation	Cardinality of Search Space
Sparse Wide	$(k_{sw})^2 \cdot (D_{in} \cdot D_{out})$
Sparse Parallel	$k_{sp} \cdot (D_{in} \cdot D_{out})$
Sparse Factorized	$d_{sf} \cdot (D_{in} + D_{out})$
Sparse Doped	$D_{in} \cdot D_{out}$

sparse. The functional transformation in this case is $\psi_l^{sf} = V_l^T \sigma(U_l^T z_l)$. The computational FLOPs of this transformation is $d_{sf} \cdot B \cdot (D_{in} + D_{out}) \cdot (1-s)$. Setting these FLOPs equal to FLOPs of f_{θ_l}, we obtain $d_{sf} = \frac{D_{in} \cdot D_{out}}{(D_{in}+D_{out}) \cdot (1-s)}$. Note, setting sparsity $s = 0$, we recover a non-linear low-rank factorization with dense matrices.

Sparse Doped is inspired by previous works which approximate a dense matrix with a combination of low-rank factorization and sparse matrix (Chen et al., 2021a; Thakker et al., 2021; Udell & Townsend, 2019; Candès et al., 2011). In our approach, we replace the feedforward function with low-rank factorization (with rank d_{sd}) and an unstructured sparse weight matrix (with sparsity s). Let $U_l \in \mathbb{R}^{D_{in} \times d_{sd}}, V_l \in \mathbb{R}^{d_{sd} \times D_{out}}$ denote the low-rank matrices, and $\theta_l^{sd} \in \mathbb{R}^{D_{in} \times D_{out}}$ denote the matrix with unstructured sparsity. The functional transformation, in this case, is given by $\psi_l^{sd} = V_l^T (U_l^T z_l) + \sigma((\theta_l^{sd})^T z_l)$. The computational FLOPs associated with this transformation are $B \cdot d_{sd} \cdot (D_{in} + D_{out}) + (1-s) \cdot B \cdot D_{in} \cdot D_{out}$. Setting these FLOPs equal to FLOPs of f_{θ_l}, we obtain $d_{sd} = \frac{s \cdot D_{in} \cdot D_{out}}{(D_{in}+D_{out})}$. Note, as $s \to 0$ and $d_{sd} \to 0$, the low-rank component of disappears, and we can recover the dense feedforward function as a special case by setting σ to Identity.

Cardinality of Search Space Increasing the sparsity mask search space with Sparse-IFT is anticipated to enhance training efficiency, as indicated by prior works (Ramanujan et al., 2020; Liu et al., 2022c; Stosic & Stosic, 2021). The likelihood of finding a lottery ticket in a randomly initialized network increases with network width (Ramanujan et al., 2020). Both Liu et al. (2022b) and Stosic & Stosic (2021) show that expanding the search space through increased width or depth improves accuracy. The search space cardinality, defined as the weights a sparse training method can explore, is detailed in Table 1. Sparse Wide, Sparse Parallel, and Sparse Factorized scale with width, parallel branches, and hidden dimension size, respectively. Sparse Doped maintains a constant search space by allocating FLOPs between a low-rank and an unstructured sparse weight matrix. Therefore, DST becomes crucial for effectively traversing this larger parameter subspace, as discussed in Section 3.1.

3. Sparse-IFT Ablation Studies

In this section, we present a comprehensive analysis of Sparse-IFT networks, focusing on their training methodologies and design considerations. First, we compare static sparse training with DST, highlighting DST's superior performance in handling larger parameter spaces through empirical results using the ResNet-18 architecture on CIFAR-100. Then, we explore critical design aspects of Sparse-IFT, including the role of non-linearities, and the benefits of dynamic unstructured sparsity over structured sparsity. Finally, we evaluate the efficacy of DST by comparing it against densely trained Sparse-IFT models.

Table 2: Sparse Wide IFT with ResNet-18 trained using various sparse training methods on CIFAR-100 across different sparsity levels (columns). Best accuracy for each sparse training method is highlighted in bold.

Dense	Sparse Method	0.50	0.75	0.90
77.0 ± 0.2	Static	**78.5 ± 0.3**	78.3 ± 0.1	78.2 ± 0.3
	SNIP	**77.8 ± 0.3**	77.0 ± 0.2	75.8 ± 0.2
	GraSP	**77.7 ± 0.3**	76.5 ± 0.3	76.5 ± 0.3
	FORCE	**77.2 ± 0.3**	76.9 ± 0.3	75.4 ± 0.4
	SET	78.8 ± 0.1	79.2 ± 0.2	**79.8 ± 0.2**
	RigL	79.1 ± 0.2	79.5 ± 0.1	**80.1 ± 0.2**
	GraNet	79.2 ± 0.2	79.6 ± 0.2	**80.0 ± 0.2**

3.1. Impact of Sparse Training Techniques

This section provides a comparative analysis of Sparse-IFT networks trained with two classes of methods: static sparse training and DST. The focus is on demonstrating DST's effectiveness in navigating larger parameter spaces, as evidenced by previous research (Huang et al., 2023; Tai et al., 2022). Our empirical results consistently show DST's superiority over static sparse training. All experiments utilize the ResNet-18 architecture on CIFAR-100 with published settings (DeVries & Taylor, 2017). Detailed model information and hyperparameters are available in Appendix C.1, and all results are averaged over 3 seeds.

Sparse-IFTs employ unstructured sparsity in its transformations. This study investigates the impact of sparse training methods on various Sparse-IFT configurations, focusing on Sparse Wide IFT with sparsity $\in \{50\%, 75\%, 90\%\}$. In Table 2, we evaluate: random static sparsity, SNIP (Lee et al., 2018), GraSP (Wang et al., 2020a), FORCE (de Jorge et al., 2020), SET (Mocanu et al., 2018), RigL (Evci et al., 2020) and GraNet (Liu et al., 2021a). SET, RigL, and GraNet are DST methods, with SET updating the mask randomly, RigL updating it with gradient information and GraNet incorporating gradual magnitude pruning (Zhu & Gupta, 2017) with RigL. Pruning at Initialization (PaI) methods (e.g., SNIP, GraSP, FORCE) and GraNet increase training FLOPs due to non-uniform sparsity and dense-to-sparse training. We address this by adjusting target sparsity levels to align Sparse-IFT training FLOPs with the dense baseline (see Appendix A.2). In Iso-FLOP scenarios, PaI methods underperform because they heavily prune parameter-rich layers to match target sparsity levels, leading to layer-collapse and poor gradient flow. Furthermore, DST methods consistently

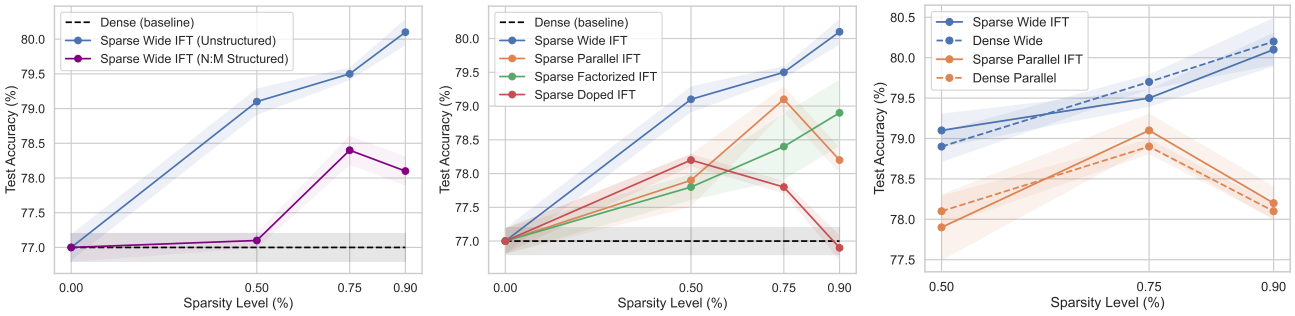

Figure 3: Ablation studies with Sparse-IFT on the ResNet-18 model for CIFAR-100 across sparsity $\in \{50\%, 75\%, 90\%\}$. (left) Sparse Wide IFT trained with dynamic unstructured and structured sparsity. (middle) Sparse-IFT family members trained with RigL, where Sparse Wide performs the best. (right) Sparse Wide IFT trained in a sparse and dense manner.

outperform static sparsity, with improvements persisting at higher sparsity levels. Sparse-IFTs expand the sparse mask-weight space $\propto$ sparsity, benefiting DST in thorough exploration and exploitation within this space. While RigL and GraNet attain similar performance, RigL is chosen as the sparse training method for simplicity in all experiments.

3.2. Assessing the Effects of Architecture Variations

This section analyzes different design considerations for Sparse-IFT by first, exploring the role of non-linearities in enhancing representational capacity. Then, the advantage of training with dynamic unstructured sparsity over structured is investigated. Next, we compare between densely and sparsely trained Sparse-IFT models. Finally, by applying top-performing Sparse-IFTs to efficient vision models, these insights contribute to a synthesized framework.

Importance of Using Non-Linear Activations For some of the Sparse-IFT members, we draw inspiration from linear overparameterization methods, which fold the feedforward function into a dense matrix post-training (Ding et al., 2021b;a; Guo et al., 2020a; Ding et al., 2019). Our method enhances representational capacity through an Iso-FLOP transformation without increasing training FLOPs. Maintaining original dense FLOP levels eliminates the need for post-training modifications, enabling efficient inference and incorporation of non-linearities (i.e., ReLU) in Sparse-IFT. Experiments on ResNet-18 on CIFAR-100 show notable accuracy gains across all sparsity levels with non-linear activations. For example, at 90% sparsity, using non-linearities in Sparse Factorized IFT yields a 1.8% accuracy increase over the dense baseline, in contrast to a 0.5% decrease without non-linearities. These findings extend to all Sparse-IFT members (see Appendix C.2 for details). The accuracy improvements at all sparsity levels highlight the effectiveness of incorporating non-linear activations in Sparse-IFT.

Unstructured vs. Structured Sparsity We compare dynamic unstructured and structured sparsity using Sparse-IFT. Unstructured sparsity explores all mask variations, but most hardware accelerators do not support unstructured sparse acceleration. Prior works have investigated structured sparsity, such as low-rank and block-sparse matrices, for wall-clock speed-ups (Khodak et al., 2020; Chen et al., 2021b; Hubara et al., 2021; Dao et al., 2022). We explore structured sparsity through Iso-FLOP configurations with Sparse Wide IFT, employing low-rank factorization and N:M sparsity for GPU acceleration. In Figure 3 (left plot), we compare dynamic unstructured sparsity with N:M transposable structured sparsity (Hubara et al., 2021) using Sparse-IFT. The latter demonstrates improvements over the dense baseline at 75% and 90% sparsity levels. Results also indicate that N:M block sparsity outperforms low-rank factorization (see Appendix C.3.3). However, unstructured sparsity still gives the highest gains, as N:M sparsity has reduced mask diversity in block-sparse matrices (Hubara et al., 2021), therefore, we adopt unstructured sparsity in all subsequent experiments.

Sparse-IFT ResNet-18 We assess all Sparse-IFT family members with ResNet-18 on CIFAR-100 across different sparsity levels. The middle plot of Figure 3, highlights the best accuracy achieved by each Sparse-IFT member. All members exhibit substantial accuracy improvements compared to the dense baseline (77%), using the same FLOPs. Sparse Wide consistently performs the best, while Sparse Doped is the only member not gaining accuracy at higher sparsity. This is attributed to Sparse Doped maintaining constant search space by distributing FLOPs between low-rank and unstructured sparse matrices (see Table 1), leading to a decrease in active weights in the unstructured matrix. In Appendix C.3.1, we compare Sparse-IFT against other DST baselines under the same training efficiency setup by extending the training steps, showing Sparse-IFT outperforms them significantly at $s \in \{50\%, 75\%, 90\%\}$. Since, Sparse Parallel and Sparse Wide perform the best across ablations, we use these two IFTs for the main experiments.

Sparse-IFT vs. Dense Overparametrization A crucial element in the success of Sparse-IFT lies in its efficient exploration of the search space. In this section, to benchmark this exploration, we establish an upper bound by training the Sparse-IFT architectures in a dense manner (with sparsity levels $s \in$ 50%, 75%, 90%). In Figure 3, the right plot compares the sparse and dense versions of Sparse Wide and Sparse Parallel IFTs. Both Sparse-IFT members ex-

Table 3: Sparse Wide IFT with various efficient architectures on CIFAR-100 across different levels of sparsity (columns).

	DENSE	0.50	0.75
MOBILENETV2	72.4 ± 0.2	73.4 ± 0.2	$\mathbf{73.7} \pm \mathbf{0.2}$
MOBILEVIT-S	73.5 ± 0.1	74.6 ± 0.2	$\mathbf{74.8} \pm \mathbf{0.2}$
BOTNET-50	79.8 ± 0.2	80.3 ± 0.3	$\mathbf{80.9} \pm \mathbf{0.3}$

cel in exploring a large search space with DST, achieving accuracy comparable to their dense counterparts without the computational overhead. These results highlight that the sparsity search in DST approaches optimality and can achieve accuracy comparable to that of densely trained models. This efficiency does not compromise accuracy and offers substantial computational benefits, especially on hardware optimized for sparsity (further discussed in Section 6).

Efficient Architectures To assess Sparse-IFT's robustness across diverse set of models, we evaluate it on architectures optimized for efficient inference (MobileNetV2 (Sandler et al., 2018) and MobileViT (Mehta & Rastegari, 2021)) and efficient training (BotNet (Srinivas et al., 2021)). Applying Sparse Wide IFT to dense layers significantly improves test accuracy across all architectures (refer to Table 3). Similarly, utilizing the Sparse Parallel IFT consistently enhances performance across all architectures (see Appendix C.3.2). We evaluate the best-performing model, BotNet-50, on ImageNet, where the Sparse-IFT variant outperforms dense by 1% (see Section 5.1). We provide additional experimental setup details in Appendix C.1. In summary, Sparse-IFT significantly improves test accuracy across all efficient architectures, demonstrating its robustness and effectiveness.

4. Spectral Analysis of DST in Sparse-IFT

In this study, we investigate the intricate properties of Sparse-IFT networks and their training dynamics. We analyze the benefits of Sparse-IFT networks trained with DST by analyzing the Ramanujan Gap and Spectral Gap characteristics. Ramanujan graph structures which are known to exhibit sparsity and high connectivity like expander graphs, are investigated to reveal their correlation with the final performance of sparse networks. Our analysis evaluates the impact of model parameters and graph connectivity on the effectiveness of DNNs with Sparse-IFTs, aiming to provide insights into the training dynamics of Sparse-IFT models. Inspired by Hoang et al. (2023b;a), in this analysis, we interpret the ResNet-18 model as a series of bipartite compute graphs, where each layer, $\{\theta_1, \ldots, \theta_L\}$ in an L layered sparse DNN, takes the form of a square adjacency matrix A. Hoang et al. (2023b) proposed several graph metrics inspired by Ramanujan properties for characterizing sparse networks, via: 1) **Ramanujan Gap**: $\Delta r = 2 * \sqrt{d-1} - \hat{\mu}(A)$, and $\Delta r_{imdb} = \frac{1}{|K|}\sum_{i=1}^{|K|}(2\sqrt{d_i - 1} - \hat{\mu}(A_{K_i}))$, where d is the average edge per node, and $\hat{\mu}(A)$ is the non-trivial eigenvalue of A. Here, Δr is the conventional view of measuring

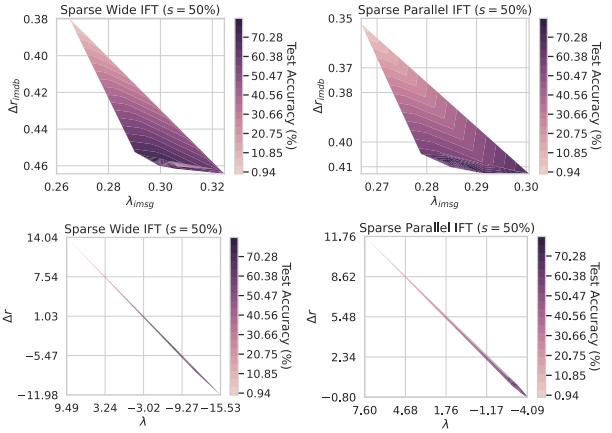

Figure 4: The relationship between the structure and weights of Sparse-IFT ResNet-18 networks are analyzed through a graph perspective in terms of performance. Top row: we assess the relationship between Δr_{imdb} and λ_{imsg}. Bottom row: investigates the correlation between Δr and λ. The Pareto curvature heatmap visually represents the classification performance, with varying color gradients symbolizing the spectrum from low to high test accuracy on CIFAR-100.

gap between Ramanujan's upper bound $2 * \sqrt{d-1}$ and $\hat{\mu}(A)$. Δr measures the network's degree of connectivity to reveal the flow of information propagation. Δr_{imdb} (Hoang et al., 2023b), the Iterative Mean Difference Bound (imdb), evaluates the average connectivity boundary across all subgraphs K within A. A higher Δr in sparse networks signifies efficient information flow, gradient propagation, and a well-separated spectrum in the adjacency matrix of sparse weights; indicating robust and efficient representation. In addition, an increasing Δr_{imdb} indicates more extensive connectivity boundaries within subgraphs, enhancing communication among nodes and promoting stronger connections. 2) **Weighted Spectral Gap**: $\lambda = \mu_0(|\boldsymbol{W}|) - \hat{\mu}(|\boldsymbol{W}|)$, and $\lambda_{imsg} = \frac{1}{|K|}\sum_{i=1}^{|K|}(\mu_0(|\boldsymbol{W}_{K_i}|) - \hat{\mu}(|\boldsymbol{W}_{K_i}|))$. Here, the gap between μ_0, the trivial parameters, and $\hat{\mu}$, the non-trivial eigenvalues of $\boldsymbol{W}$, the weighted adjacency matrix, is denoted as λ, the weighted spectral gap. Then, λ_{imsg} (Hoang et al., 2023a) is the iterative version which takes into account all subgraphs K within $\boldsymbol{W}$. A higher λ_{imsg} indicates enhanced spectral separation between μ_0 and $\hat{\mu}$ of $\boldsymbol{W}$, implying a more distinct and well-defined spectral structure within subgraphs. This improved separation in the spectrum, represented by a higher λ, facilitates better isolation of meaningful signals. We train Sparse Wide and Sparse Parallel ResNet-18 models at 50% sparsity on CIFAR-100. Then, we generated a Pareto curvature heatmap, considering weight magnitudes and graph topological structure details (see Figure 4). See Appendix B for a detailed analysis.

Δr_{imdb} and λ_{imsg} Analysis: In the initial to middle stages of training, RigL's dynamic pruning and regrowth increases Δr_{imdb} for the network to *explore* diverse connec-

tivity patterns (see top row, Figure 4). Subsequent pruning removes less critical connections, diversifying subgraphs in the adjacency matrix A. Later stages witness a decrease in Δr_{imdb} as the network converges to more focused and organized connectivity patterns. RigL prioritizes crucial connections, *exploiting* an efficient subgraph structure linked with highly accuracy regions of Sparse-IFT models. Early in training, increasing λ_{imsg} suggests successful isolation of different modes. Pruning leads to a distinct separation between dominant and less dominant modes. The subsequent λ_{imsg} decrease signals the network's convergence to a more specialized representation, emphasizing key spectral components and diminishing the influence of less critical modes. While both Sparse Wide and Sparse Parallel IFTs show increasing Δr_{imdb}, the larger search space cardinality in Sparse Wide facilitates the emergence of diverse subgraph structures within each layer, allowing for a richer set of connections between nodes; resulting in a higher maximum Δr_{imdb}. Similarly, Sparse Wide has a higher maximum λ_{imsg} compared to Sparse Parallel, indicating the emergence of subgraphs with more distinct spectral properties.

Δr and λ Analysis: Figure 4's bottom row reveals a strong correlation between Δr and λ. Δr initially decreases, indicating a temporary relaxation of spectral constraints during dynamic pruning with RigL. Subsequently, it maximizes in the final training stages, signifying RigL's ability to guide the network to reorganize its connectivity, promoting more structured and favorable spectral characteristics. Similarly, λ follows a pattern of initial decrease and later maximization. This implies that RigL's dynamic sparsity initially results in less optimal weight organization concerning spectral properties. However, RigL's iterative pruning and rewiring dynamically adapts the network, aligning weights to enhance spectral characteristics and increase the spectral gap. Our analysis demonstrates that DST, as exemplified by RigL, outperforms static sparse training by optimizing spectral characteristics for Sparse-IFT; facilitating improved connectivity patterns and a more favorable spectral profile.

5. Empirical Evaluation

Building on insights gained from our ablations discussed in Section 3.2, we apply Sparse-IFTs to ImageNet, also demonstrating its advantages for transfer learning in various computer vision tasks. Additionally, we highlight the benefits of Sparse-IFT in the domain of NLP by presenting results on pre-training GPT (Brown et al., 2020).

5.1. ImageNet and Transfer Learning

We apply the best-performing Sparse-IFT transformations (Sparse Wide IFT and Sparse Parallel IFT) from CIFAR-100 to ImageNet using ResNet-18. We follow published training settings for ImageNet (Nvidia, 2023). Both Sparse-IFT families achieve significantly higher accuracy compared to the dense baseline (see Table 4). Specifically, Sparse Wide IFT ResNet-18 at 90% sparsity improves over the dense baseline by 3.5% and matches the accuracy of a dense ResNet-34 with 2× fewer training FLOPs (refer to Figure 1). We also apply the best-performing transformation (Sparse Wide IFT) to ResNet-34 and BotNet-50. Increasing sparsity consistently improves accuracy, indicating enhanced training efficiency at higher sparsities. On BotNet-50, a hybrid ViT model, there is a 1.1% improvement at 90% sparsity.

Table 4: Sparse-IFT on ImageNet. Best result for each transformation and architecture is highlighted in bold.

Model	Dense	Transformation	0.50	0.75	0.90
ResNet-18	70.9 ± 0.1	Sparse Wide	72.7 ± 0.1	73.8 ± 0.2	$\mathbf{74.4 \pm 0.2}$
		Sparse Parallel	72.7 ± 0.2	73.2 ± 0.2	$\mathbf{74.0 \pm 0.2}$
ResNet-34	74.2 ± 0.1	Sparse Wide	75.6 ± 0.2	76.4 ± 0.1	$\mathbf{76.8 \pm 0.3}$
BotNet-50	77.5 ± 0.1	Sparse Wide	77.9 ± 0.2	78.3 ± 0.2	$\mathbf{78.6 \pm 0.3}$

Table 5: Sparse Wide IFT variants of ResNet-18 as backbones for: (a) object detection on MS COCO, (b) semantic segmentation on Cityscapes.

	Metric	Dense	0.50	0.75	0.90
MS COCO	AP	29.3 ± 0.1	31.3 ± 0.1	32.8 ± 0.2	$\mathbf{34.5 \pm 0.2}$
	AP_{50}	46.2 ± 0.2	49.0 ± 0.2	51.0 ± 0.2	$\mathbf{53.5 \pm 0.2}$
	AP_{75}	30.9 ± 0.2	33.0 ± 0.2	34.8 ± 0.2	$\mathbf{36.5 \pm 0.3}$
CityScapes	mIoU	76.7 ± 0.2	77.9 ± 0.2	78.9 ± 0.2	$\mathbf{79.1 \pm 0.2}$
	mAcc	84.4 ± 0.2	85.1 ± 0.2	85.7 ± 0.2	$\mathbf{86.0 \pm 0.2}$

Transfer Learning on Downstream To show the effectiveness of pre-training our Sparse-IFT classification backbones, we evaluate them on 1) object detection on MS COCO 2017 (Lin et al., 2014b), and 2) semantic segmentation on CityScapes (Cordts et al., 2016). For object detection, we adopt RetinaNet (Lin et al., 2017b) from the MMDetection open-source toolbox (Chen et al., 2019) and report results in the standardized training setting. For semantic segmentation, we utilize DeepLabV3+ (Chen et al., 2018) in the MMSegmenation open-source toolbox (Contributors, 2020). We evaluate ResNet-18 with Sparse Wide IFT and to ensure FLOP-equivalent comparisons with the dense backbone, the Sparse-IFT backbones remain sparse during fine-tuning. Appendix C.3.4 provides more details on the training setup. We summarize our findings in Table 5, where using Sparse Wide IFT ResNet-18 backbone leads to significant accuracy gains across all metrics on both tasks.

5.2. Language Modeling

We pre-train the Sparse Wide IFT GPT-3 Small model at $s \in \{50\%, 75\%\}$ from scratch on the Pile (Gao et al., 2020) dataset using SET (Mocanu et al., 2018), and compare against the standard dense model. All models were trained on the Cerebras CS-2 (Cerebras, 2023) following Chinchilla (Hoffmann et al., 2022) for obtaining loss-optimal pre-trained baseline configurations of models. We evaluate the models on 5 tasks from the Open LLM leader-

Table 6: Average accuracy of Sparse Wide IFT with GPT-3 Small across ARC, HellaSwag, TruthfulQA, MMLU and Winogrande tasks on the Open LLM Leaderboard.

MODEL	DENSE	0.50	0.75
GPT-3 SMALL	33.8 ± 0.1	34.1 ± 0.2	$\mathbf{34.7 \pm 0.2}$

board (Beeching et al., 2023) (i.e., ARC (Clark et al., 2018), HellaSwag (Zellers et al., 2019), MMLU (Hendrycks et al., 2021), TruthfulQA (Lin et al., 2022) and Winogrande (Sakaguchi et al., 2019)), and show that the Sparse Wide IFT GPT-3 Small at 75% sparsity improves the average accuracy by a noticeable 0.9% (see Table 6). In Appendix D.1, we provide details on the models and hyperparameters.

6. Wall-Clock Acceleration with Sparse-IFT

Our studies in Section 5 show noticeably improved training efficiency (test accuracy w.r.t training FLOPs) for Sparse-IFT models. In this section, we aim to showcase the practicality of Sparse-IFT models, providing unique hardware insights for accelerating DNNs with unstructured sparsity, a perspective notably absent in most existing works. Recent developments, like specialized software kernels and hardware (e.g., DeepSparse (NeuralMagic, 2021) and Cerebras CS-2 (Lie, 2023)) indicate promising gains in realizing unstructured sparsity benefits during training and inference (Thangarasa et al., 2023). This sets the stage for examining the impact on inference and training acceleration.

Real-World Inference Acceleration We assess Sparse-IFT's inference efficiency using DeepSparse[1]. Our setup employs a ResNet-18 model and performs batched inference of 64 images from ImageNet at 224 × 224 resolution on Intel Cascade Lake CPUs, known for their AVX-512 support. The latency (i.e., seconds per batch) is compared between the dense ResNet-18 model and the Sparse Wide IFT variants at $s \in 50\%, 75\%, 90\%\}$. On an ideal hardware, FLOPs should directly translate to wall clock time. Therefore, the inference latency or training time for all Sparse-IFT models should match that of the dense model, as all models are Iso-FLOP. This baseline is illustrated by the black dashed line in the left plot of Figure 5. However, the blue line shows the expected increases in latency on hardware *without* unstructured sparse acceleration support, like the CPUs we benchmarked on, with a notable 19.5x increase at $s = 90\%$. In contrast, the green line demonstrates a significant reduction in latency using DeepSparse, decreasing the latency increase from 19.5x to 3.5x, and showing minimal overhead up to 75% sparsity. This emphasizes the benefits of optimized kernel support for sparse inference acceleration, showcasing the potential for practical deployment of Sparse-IFT models.

[1]Neural Magic DeepSparse

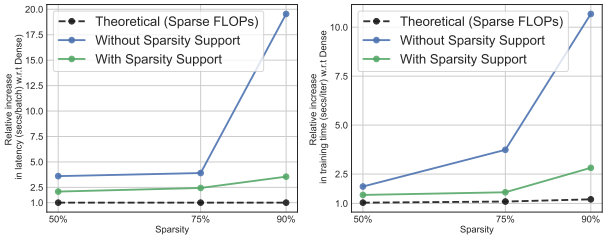

Figure 5: Benchmarking unstructured sparsity during (left) inference on Neural Magic's DeepSparse runtime and (right) training acceleration on the Cerebras CS-2. In both setups, we measure the relative increase in latency or training speed for Sparse-IFT variants against the dense model.

Real-World Training Acceleration In the right plot of Figure 5, we evaluate the training efficiency of Sparse-IFT on the Cerebras CS-2 system, which supports unstructured sparse training for LLMs[2]. Our experimental setup involves pre-training a GPT-3 model on the CS-2. We measured and compared throughput (i.e., iterations per second), between the dense GPT-3 model and Sparse Wide IFT variants at sparsity levels of 50%, 75%, and 90%. As previously mentioned, the theoretical baseline (black dashed line) suggests that since both the dense model and Sparse Wide IFT configurations are Iso-FLOP, training time should not increase with increasing sparsity. The blue line shows the throughput for Sparse Wide IFT variants run *without* any unstructured sparse acceleration support on the Cerebras CS-2, indicating a ∼10x increase in training time for the model at 90% sparsity. A similar degradation in performance would be expected on traditional Nvidia GPU or Google TPU hardware as well. In contrast, the green line demonstrates the effect of utilizing the Cerebras CS-2's unstructured sparse training support. Here, we observe a significant reduction in relative training time, bringing down the increase from ∼10x to 2.82x at 90% sparsity. Additionally, for sparsity levels up to 75%, we note minimal overhead compared to the dense model. Detailed benchmarking setups are in Appendix E.

While we do not achieve perfect FLOPs translation with Sparse-IFT models, our promising results highlight the importance of ML software and hardware co-design for leveraging sparsity. The interaction of layer dimensions, sparsity, and overhead, influenced by hardware architecture, necessitates co-designed sparse techniques for optimal performance. Our work showcases algorithmic advancements over prior sparse methods, emphasizing the benefits of sparse training and winning *the hardware lottery* (Hooker, 2020).

7. Related Work

Our work aligns with research on overparameterization and sparsity in DNN training. The required modeling capacity for a given task is often unknown, leading to training over-

[2]Cerebras CS-2 (R2.1.1): Train a Model with Weight Sparsity

parameterized models to fully exploit learning capabilities before compressing them into smaller subnetworks.

Overparameterization Nakkiran et al. (2021) show that DNNs benefit from overparameterization. Subsequently, several studies have capitalized on overparameterization by scaling the size of models (Rae et al., 2021; Goyal et al., 2022) and augmenting existing DNNs to boost modeling capacity and the accuracy of trained networks (Guo et al., 2020b; Ding et al., 2019; 2021b; Cao et al., 2022; Vasu et al., 2022; Liu et al., 2022a). These methods use linear parameterizations of the model, making them highly inefficient to train, and are focused on improving inference throughput. In contrast, our work is focused on improving the modeling capacity using sparse non-linear parameterizations. Sparse-IFT enhances accuracy without increasing training and inference FLOPs compared to the baseline dense model.

Sparse Network Training The Lottery Ticket Hypothesis (Frankle & Carbin, 2018; Frankle et al., 2020) shows that accurate sparse subnetworks exist in overparameterized dense networks but require training a dense baseline to find. Other approaches have proposed frameworks for identifying lottery tickets (Zhou et al., 2019; Ma et al., 2022) but still require a lot of compute resources. Following this, various attempts have been made to find the optimal sparse subnetwork in a single shot. These methods either try to find the subnetworks at initialization (Tanaka et al., 2020; Wang et al., 2020a; de Jorge et al., 2020; Lee et al., 2018) or dynamically during training (Mocanu et al., 2018; Evci et al., 2020; Jayakumar et al., 2020; Raihan & Aamodt, 2020). However, given a fixed model capacity, these methods tradeoff accuracy relative to the dense baseline to save training FLOPs. Stosic & Stosic (2021) and Ramanujan et al. (2020) increase the search space during sparse training to retain accuracy; however, do not guarantee FLOPs savings. In contrast to these methods, our work introduces a set of non-linear sparse transformations, which increase the representational capacity of the network. This approach does not introduce a new sparse training algorithm, but instead improves the search space of existing methods, leading to improved generalization while being efficient to train.

Iso-Parameter vs. Iso-FLOP Recent works have focused on improving generalization at high sparsity levels. Techniques such as the Erdös-Rényi-Kernel (Evci et al., 2020), Ideal Gas Quota (Chen et al., 2022), and parameter leveling (Golubeva et al., 2021) employ layer-wise sparsity distributions in sparse training to boost accuracies. These methods, however, target scenarios where models have a fixed parameter budget (i.e., Iso-Parameter), which does not equate to similar training FLOPs as the original dense model. Our work highlights that while transformer-based NLP networks may not show significant differences between Iso-Parameter and Iso-FLOP optimization, this distinction becomes critical in CV networks. In CNNs and heterogeneous ViTs (Wang et al., 2021; Pan et al., 2021; Wu et al., 2021), the uneven distribution of parameters and computational costs across layers necessitates a distinct approach. Optimizing for Iso-Parameter typically involves pruning later, parameter-rich layers, thus maintaining performance but not significantly reducing computational costs. Conversely, optimizing for Iso-FLOP shifts pruning to early, FLOP-intensive layers, enhancing performance by addressing both computational demands and pruning needs. Unlike variable sparsity techniques that adapt to different computational and memory demands across layers, our method employs a uniform sparsity approach, ensuring consistent FLOP reductions across all layers. This aligns computational costs closely with those of a fully dense model, achieving significant computational efficiencies without compromising performance.

Sparse-IFT and Scaling Laws Recent advances in deep learning highlight the importance of scaling laws, which provide a systematic framework for optimizing model performance as model size increases. Pioneering scaling laws work such as ConvNeXt (Liu et al., 2022d), EfficientNet (Tan & Le, 2019), large language models (Kaplan et al., 2020), and vision transformers (Alabdulmohsin et al., 2023) demonstrate that achieving optimal performance typically involves tuning multiple training (e.g., learning rate, batch sizes, etc.) and architectural (e.g., depth, width, resolution, etc.) hyperparameters. This intricate balance necessitates extensive experimentation and hyperparameter tuning. Sparse-IFT introduces a streamlined approach for scaling DNNs, leveraging a single hyperparameter, the sparsity level, to enhance model efficiency and accuracy. This method simplifies the optimization process by eliminating the need to tune multiple factors concurrently. Future research will explore the integration of Sparse-IFT with scaling laws to address the challenges of scaling large models. This involves examining the interplay between Sparse-IFT and various architectural elements, such as depth and width, while maintaining a constant computational FLOP budget.

8. Conclusion

We introduced Sparse-IFT as a drop-in replacement for dense layers in DNNs, enhancing test accuracy w.r.t training FLOPs by increasing representational capacity through sparsity. The expanded weight space enables effective exploration and exploitation by DST algorithms, facilitating the discovery of optimal sparse subnetworks. Our spectral analysis of Sparse-IFT models trained with DST reveals efficient connectivity and information propagation, correlated with high-performance networks. Notably, Sparse-IFT consistently outperforms dense models in vision and NLP domains. Despite current hardware limitations, promising benchmarks on Cerebras CS-2 and Neural Magic DeepSparse runtime highlight the need for improved support for unstructured weight sparsity. We hope our findings encourage the community to explore unstructured sparsity for improved model efficiency and performance across applications.

Acknowledgements

We thank Anshul Samar for his helpful comments and edits that improved our manuscript. We also thank Chen-Yu Kevin Leong for assisting with the Cerebras CS-2 GPT-3 experiments and Dylan Finch for performance evaluations on CS-2. Finally, we provide details on each author's contributions in Appendix F.

Impact Statement

The landscape of machine learning (ML) has witnessed an exponential growth in models, particularly in domains such as natural language processing and computer vision. However, this surge in model size has come with a considerable cost in terms of compute, memory, and energy requirements. Our approach, Sparse Iso-FLOP Transformations (Sparse-IFT), represents a significant stride toward mitigating these resource-intensive demands. Sparse-IFT introduces a novel approach that enhances the efficiency of training large neural networks. Remarkably, it achieves improved accuracy while maintaining the same FLOPs as the original dense baseline model. Our method holds promise for positive environmental impacts, given the substantial computational resources typically associated with training large neural networks.

Models trained with Sparse-IFT require less computing resources and energy to achieve higher model quality, directly translating to lower deployment costs for real-world applications. Furthermore, training models with sparsity has been shown to lead to better generalization (Chen et al., 2022), a benefit supported by our transfer learning results on computer vision tasks. An additional advantage is the enhanced efficiency in training larger sparse models, facilitated by the widespread adoption of AI hardware, such as the Cerebras CS-2, which accelerates unstructured sparsity. The key here is achieving sparsity acceleration through a harmonious collaboration between hardware support and the development of sparse ML techniques.

The potential sustainability contribution lies in the fact that, as sparse ML software and hardware co-design continues to evolve, we may be able to train more accurate "larger sparse" networks within the confines of the same computational budget as a smaller dense model. This paradigm shift could usher in a more environmentally conscious approach to deep learning, addressing the concerns associated with the escalating resource requirements of ever-expanding models. The seamless integration of these elements ensures that hardware architectures are optimized to complement sparse techniques, fostering a sustainable and efficient trajectory for the future of deep learning.

As we continue to explore the intersection of hardware support for sparsity and the evolution of sparse ML techniques, our benchmarking analysis in Section 6 and Appendix E serves as a practical illustration of the transformative potential of Sparse-IFT. It not only substantiates the theoretical promises but also offers a roadmap for future developments in the pursuit of sustainable and efficient deep learning practices.

References

Alabdulmohsin, I., Zhai, X., Kolesnikov, A., and Beyer, L. Getting vit in shape: Scaling laws for compute-optimal model design. In *NeurIPS*, 2023.

Arora, S., Cohen, N., and Hazan, E. On the optimization of deep networks: Implicit acceleration by overparameterization. In *ICML*, 2018.

Beeching, E., Fourrier, C., Habib, N., Han, S., Lambert, N., Rajani, N., Sanseviero, O., Tunstall, L., and Wolf, T. Open llm leaderboard. https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard, 2023.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. In *NeurIPS*, 2020.

Candès, E. J., Li, X., Ma, Y., and Wright, J. Robust principal component analysis? *Journal of the ACM*, 2011.

Cao, J., Li, Y., Sun, M., Chen, Y., Lischinski, D., Cohen-Or, D., Chen, B., and Tu, C. Do-conv: Depthwise over-parameterized convolutional layer. *IEEE Transactions on Image Processing*, 2022.

Cerebras. Train a model with weight sparsity. *Cerebras Wafer-Scale cluster (R2.1.1) Documentation*, Jan 2023. URL https://docs.cerebras.net/en/2.1.1/wsc/how_to_guides/sparsity.html.

Chen, B., Dao, T., Winsor, E., Song, Z., Rudra, A., and Ré, C. Scatterbrain: Unifying sparse and low-rank attention approximation. In *NeurIPS*, 2021a.

Chen, K., Wang, J., Pang, J., Cao, Y., Xiong, Y., Li, X., Sun, S., Feng, W., Liu, Z., Xu, J., Zhang, Z., Cheng, D., Zhu, C., Cheng, T., Zhao, Q., Li, B., Lu, X., Zhu, R., Wu, Y., Dai, J., Wang, J., Shi, J., Ouyang, W., Loy, C. C., and Lin, D. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv*, 2019.

Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., and Adam, H. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018.

Chen, P., Yu, H.-F., Dhillon, I., and Hsieh, C.-J. Drone: Data-aware low-rank compression for large nlp models. In *NeurIPS*, 2021b.

Chen, T., Frankle, J., Chang, S., Liu, S., Zhang, Y., Wang, Z., and Carbin, M. The lottery ticket hypothesis for pre-trained bert networks. In *NeurIPS*, 2020.

Chen, T., Zhang, Z., pengjun wang, Balachandra, S., Ma, H., Wang, Z., and Wang, Z. Sparsity winning twice: Better robust generalization from more efficient training. In *ICLR*, 2022.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018.

Contributors, M. MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. https://github.com/open-mmlab/mmsegmentation, 2020.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016.

Dao, T., Chen, B., Sohoni, N. S., Desai, A., Poli, M., Grogan, J., Liu, A., Rao, A., Rudra, A., and Ré, C. Monarch: Expressive structured matrices for efficient and accurate training. In *ICML*, 2022.

de Jorge, P., Sanyal, A., Behl, H. S., Torr, P. H., Rogez, G., and Dokania, P. K. Progressive skeletonization: Trimming more fat from a network at initialization. *arXiv*, 2020.

DeVries, T. and Taylor, G. W. Improved regularization of convolutional neural networks with cutout. *arXiv*, 2017.

Ding, X., Guo, Y., Ding, G., and Han, J. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In *ICCV*, 2019.

Ding, X., Zhang, X., Han, J., and Ding, G. Diverse branch block: Building a convolution as an inception-like unit. In *CVPR*, 2021a.

Ding, X., Zhang, X., Ma, N., Han, J., Ding, G., and Sun, J. Repvgg: Making vgg-style convnets great again. In *CVPR*, 2021b.

Evci, U., Gale, T., Menick, J., Castro, P. S., and Elsen, E. Rigging the lottery: Making all tickets winners. In *ICML*, 2020.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2018.

Frankle, J., Dziugaite, G. K., Roy, D., and Carbin, M. Linear mode connectivity and the lottery ticket hypothesis. In *ICML*, 2020.

Gale, T., Elsen, E., and Hooker, S. The state of sparsity in deep neural networks. *arXiv*, 2019.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv*, 2020.

Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., Phang, J., Reynolds, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, September 2021.

Golubeva, A., Gur-Ari, G., and Neyshabur, B. Are wider nets better given the same number of parameters? In *ICLR*, 2021.

Goyal, P., Duval, Q., Seessel, I., Caron, M., Singh, M., Misra, I., Sagun, L., Joulin, A., and Bojanowski, P. Vision models are more robust and fair when pretrained on uncurated images without supervision. *arXiv*, 2022.

Guo, S., Alvarez, J. M., and Salzmann, M. Expandnets: Linear over-parameterization to train compact convolutional networks. In *NeurIPS*, 2020a.

Guo, S., Alvarez, J. M., and Salzmann, M. Expandnets: Linear over-parameterization to train compact convolutional networks. In *NeurIPS*, 2020b.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv*, 2015a.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015b.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *ECCV*, 2016.

He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J., and Li, M. Bag of tricks for image classification with convolutional neural networks. In *CVPR*, 2019.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding, 2021.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv*, 2015.

Hoang, D. N., Kundu, S., Liu, S., and Wang, Z. Don't just prune by magnitude! your mask topology is a secret weapon. In *NeurIPS*, 2023a.

Hoang, D. N., Liu, S., Marculescu, R., and Wang, Z. Revisiting pruning at initialization through the lens of ramanujan graph. In *ICLR*, 2023b.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Vinyals, O., Rae, J. W., Sifre, L., and et al. An empirical analysis of compute-optimal large language model training. In *NeurIPS*, 2022.

Hooker, S. The Hardware Lottery. *arXiv*, 2020.

Huang, S., Lei, B., Xu, D., Peng, H., Sun, Y., Xie, M., and Ding, C. Dynamic sparse training via balancing the exploration-exploitation trade-off. *arXiv*, 2023.

Hubara, I., Chmiel, B., Island, M., Banner, R., Naor, J., and Soudry, D. Accelerated sparse neural training: A provable and efficient method to find n:m transposable masks. In *NeurIPS*, 2021.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Iofinova, E., Peste, A., Kurtz, M., and Alistarh, D. How well do sparse imagenet models transfer? *CoRR*, abs/2111.13445, 2021.

Jayakumar, S., Pascanu, R., Rae, J., Osindero, S., and Elsen, E. Top-kast: Top-k always sparse training. In *NeurIPS*, 2020.

Jiang, P., Hu, L., and Song, S. Exposing and exploiting fine-grained block structures for fast and accurate sparse training. In *NeurIPS*, 2022.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv*, 2020.

Khodak, M., Tenenholtz, N. A., Mackey, L., and Fusi, N. Initialization and regularization of factorized neural layers. In *ICLR*, 2020.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

Kurtz, M., Kopinsky, J., Gelashvili, R., Matveev, A., Carr, J., Goin, M., Leiserson, W., Moore, S., Nell, B., Shavit, N., and Alistarh, D. Inducing and exploiting activation sparsity for fast inference on deep neural networks. In *ICML*, 2020.

Lee, N., Ajanthan, T., and Torr, P. H. Snip: Single-shot network pruning based on connection sensitivity. *arXiv*, 2018.

Lie, S. Thinking outside the die: Architecting the ml accelerator of the future. https://www.microarch.org/micro54/media/lie-keynote.pdf, 2021.

Lie, S. Cerebras architecture deep dive: First look inside the hardware/software co-design for deep learning. *IEEE Micro*, 2023.

Lin, M., Chen, Q., and Yan, S. Network in network. In *ICLR*, 2014a.

Lin, S., Hilton, J., and Evans, O. Truthfulqa: Measuring how models mimic human falsehoods, 2022.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *ECCV*, 2014b.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. Feature Pyramid Networks for Object Detection. In *CVPR*, 2017a.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. Focal loss for dense object detection. In *ICCV*, 2017b.

Littwin, E., Myara, B., Sabah, S., Susskind, J., Zhai, S., and Golan, O. Collegial ensembles. In *NeurIPS*, 2020.

Liu, S., Chen, T., Chen, X., Atashgahi, Z., Yin, L., Kou, H., Shen, L., Pechenizkiy, M., Wang, Z., and Mocanu, D. C. Sparse training via boosting pruning plasticity with neuroregeneration. 2021a.

Liu, S., Mocanu, D. C., Pei, Y., and Pechenizkiy, M. Selfish sparse rnn training. In *ICML*, 2021b.

Liu, S., Chen, T., Chen, X., Chen, X., Xiao, Q., Wu, B., Pechenizkiy, M., Mocanu, D., and Wang, Z. More convnets in the 2020s: Scaling up kernels beyond 51x51 using sparsity. *arXiv*, 2022a.

Liu, S., Chen, T., Chen, X., Shen, L., Mocanu, D. C., Wang, Z., and Pechenizkiy, M. The unreasonable effectiveness of random pruning: Return of the most naive baseline for sparse training. In *ICLR*, 2022b.

Liu, S., Chen, T., Chen, X., Shen, L., Mocanu, D. C., Wang, Z., and Pechenizkiy, M. The unreasonable effectiveness of random pruning: Return of the most naive baseline for sparse training. *arXiv*, 2022c.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, 2021c.

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., and Xie, S. A convnet for the 2020s. In *CVPR*, 2022d.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv*, 2017.

Ma, X., Qin, M., Sun, F., Hou, Z., Yuan, K., Xu, Y., Wang, Y., Chen, Y.-K., Jin, R., and Xie, Y. Effective model sparsification by scheduled grow-and-prune methods. In *ICLR*, 2022.

Mehta, S. and Rastegari, M. Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer. In *ICLR*, 2021.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., and Wu, H. Mixed precision training. In *ICLR*, 2018.

Mocanu, D., Mocanu, E., Stone, P., Nguyen, P., Gibescu, M., and Liotta, A. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 2018.

Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., and Sutskever, I. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021.

NeuralMagic. Deepsparse, 2021. URL https://github.com/neuralmagic/deepsparse.

Nvidia. Resnet v1.5 for pytorch. 2019. URL https://catalog.ngc.nvidia.com/orgs/nvidia/resources/resnet_50_v1_5_for_pytorch.

Nvidia. Nvidia performance documentation. 2023. URL https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html.

Pan, Z., Zhuang, B., Liu, J., He, H., and Cai, J. Scalable vision transformers with hierarchical pooling. In *ICCV*, 2021.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. Improving language understanding by generative pre-training. *OpenAI Blog*, 2018.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019.

Rae, J. W., Borgeaud, S., Cai, T., Millican, K., Hoffmann, J., Song, F., Aslanides, J., Henderson, S., Ring, R., Young, S., et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv*, 2021.

Raihan, M. A. and Aamodt, T. Sparse weight activation training. In *NeurIPS*, 2020.

Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A., and Rastegari, M. What's hidden in a randomly weighted neural network? In *CVPR*, 2020.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. WINOGRANDE: an adversarial winograd schema challenge at scale, 2019.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *Nature*, 2017.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv*, 2014.

Srinivas, A., Lin, T.-Y., Parmar, N., Shlens, J., Abbeel, P., and Vaswani, A. Bottleneck transformers for visual recognition. In *CVPR*, 2021.

Stosic, D. and Stosic, D. Search spaces for neural model training. *arXiv*, 2021.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

Tai, C., Xiao, T., Zhang, Y., Wang, X., and Weinan, E. Convolutional neural networks with low-rank regularization. In *ICLR*, 2016.

Tai, K. S., Tian, T., and Lim, S.-N. Spartan: Differentiable Sparsity via Regularized Transportation. In *NeurIPS*, 2022.

Tan, M. and Le, Q. EfficientNet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

Tanaka, H., Kunin, D., Yamins, D. L., and Ganguli, S. Pruning neural networks without any data by iteratively conserving synaptic flow. In *NeurIPS*, 2020.

Thakker, U., Whatmough, P. N., Liu, Z., Mattina, M., and Beu, J. Doping: A technique for efficient compression of lstm models using sparse structured additive matrices. In *MLSys*, 2021.

Thangarasa, V., Gupta, A., Marshall, W., Li, T., Leong, K., DeCoste, D., Lie, S., and Saxena, S. SPDF: Sparse pre-training and dense fine-tuning for large language models. In *UAI*, 2023.

Udell, M. and Townsend, A. Why are big data matrices approximately low rank? *SIAM Journal on Mathematics of Data Science*, 2019.

Vasu, P. K. A., Gabriel, J., Zhu, J., Tuzel, O., and Ranjan, A. An improved one millisecond mobile backbone. *arXiv*, 2022.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *NeurIPS*, 2017.

Wang, C., Zhang, G., and Grosse, R. Picking winning tickets before training by preserving gradient flow. *arXiv*, 2020a.

Wang, J., Sun, K., Cheng, T., Jiang, B., Deng, C., Zhao, Y., Liu, D., Mu, Y., Tan, M., Wang, X., et al. Deep high-resolution representation learning for visual recognition. In *TPAMI*, 2020b.

Wang, W., Xie, E., Li, X., Fan, D.-P., Song, K., Liang, D., Lu, T., Luo, P., and Shao, L. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *ICCV*, 2021.

Wu, H., Xiao, B., Codella, N., Liu, M., Dai, X., Yuan, L., and Zhang, L. Cvt: Introducing convolutions to vision transformers. In *ICCV*, 2021.

Yuan, G., Ma, X., Niu, W., Li, Z., Kong, Z., Liu, N., Gong, Y., Zhan, Z., He, C., Jin, Q., et al. Mest: Accurate and fast memory-economic sparse training framework on the edge. volume 34, 2021.

Zagoruyko, S. and Komodakis, N. Wide residual networks. In *BMVC*, 2016.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence?, 2019.

Zhao, H., Shi, J., Qi, X., Wang, X., and Jia, J. Pyramid scene parsing network. In *CVPR*, 2017.

Zhou, H., Lan, J., Liu, R., and Yosinski, J. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *NeurIPS*, 2019.

Zhu, M. and Gupta, S. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv*, 2017.

A. Additional Methodology Details

A.1. Sparse-IFT for Convolutional Layers

In this section, we detail the straightforward extension of the Sparse-IFT family for convolutional layers.

Sparse Wide Similar to the setup for fully connected layers, in the case of convolutional layers, we widen the number of input and output channels.

Sparse Parallel Similar to the setup for fully connected layers, in the case of convolutional layers, we can implement this transformation with the use of convolutional branches in parallel.

Sparse Factorized and Sparse Doped Let $\theta_l \in \mathbb{R}^{c_{in} \times c_{out} \times k_h \times k_w}$ represent the weight matrix of a convolutional layer, where $c_{in}, c_{out}, k_h, k_w$ denote the input channels, output channels, kernel height, and kernel width, respectively. We apply low-rank or matrix factorization to the weight matrix by first converting the 4D tensor into a 2D matrix with shape: $(c_{in} \cdot k_h \cdot k_w) \times c_{out}$. In this setup, we can express $\theta_l = UV^T$, where $U \in \mathbb{R}^{c_{in} \cdot k_h \cdot k_w \times d}$, $V \in \mathbb{R}^{c_{out} \times d}$. In this factorization, U learns a lower-dimensional set of features and is implemented as a convolutional layer with d output channels and $k_h \times k_w$ filter. V matrix expands this low-dimensional set of features and is implemented as a convolutional layer with 1×1 filter.

A.1.1. Sparse-IFT for Depthwise Convolution Layers

For a normal convolution layer, all inputs are convolved to all outputs. However, for depthwise convolutions, each input channel is convolved with its own set of filters. Let $\theta_l \in \mathbb{R}^{c_{in} \times c_{out} \times k_h \times k_w}$ represent the weight matrix of a normal convolution layer, where $c_{in}, c_{out}, k_h, k_w$ denote the input channels, output channels, kernel height, and kernel width, respectively. An equivalent depthwise convolution layer will have weights $\theta_{dw,l} \in \mathbb{R}^{1 \times c_{out} \times k_h \times k_w}$.

Sparse Wide A Sparse Wide depthwise convolution will have weights $\theta^{sw}_{dw,l} \in \mathbb{R}^{1 \times k_{sw} \cdot c_{out} \times k_h \times k_w}$. Since the fraction of non-sparse weights is given by $1 - s$, the FLOPs required by this transformation are $B \cdot (k_{sw} \cdot c_{out}) \cdot k_h \cdot k_w \cdot (1 - s)$. Setting these equal to the FLOPs of the original dense $\theta_{dw,l}$, we obtain the widening factor $k_{sw} = \frac{1}{(1-s)}$. In this case, we do not scale the input channels as it converts the depthwise convolution to a grouped convolution without an equivalent scaling in the number of groups.

Other Sparse-IFT Transformations The Sparse Wide IFT generally changes a layer's input and output channels, subsequently scaling the following layers in a CNN. However, the other Sparse-IFT transforms (Sparse Parallel, Sparse Factorized, and Sparse Doped) do not modify a convolution layer's input or output channels (as seen in Figure 2). This allows for fine-grained control of what layers to apply the Sparse-IFT transformations. Since depthwise convolutions are an extreme form of structured sparsity, where some filters interact with only specific input channels, we opt not to sparsify them when using the other Sparse-IFT transformations and leave the layer unchanged while still maintaining FLOPs equivalent to the dense baseline. Note that the different convolution layers surrounding the depthwise convolution are still transformed with Sparse-IFT to increase their representational capacity.

A.2. Controlling for Iso-FLOP

As mentioned before, in our work, we mainly apply a uniform sparsity distribution to the model, which essentially means each layer is allocated the same level of sparsity. Let $\mathcal{N}$ denote a L layered DNN parameterized by $\Theta_{\mathcal{N}}$. Let $\Theta_{\mathcal{N}} \in \{\theta_1, \ldots, \theta_L\}$ denote the parameters of the DNN. Now, let M_l be the binary mask for layer $l \in \{1, \ldots, L\}$ with dimensions corresponding to the parameters of that layer. The binary mask m_l has values of 1 for active weights and 0 for non-active weights. Let θ_l be the total number of parameters in l, hence, the sparsity level per layer, s_l, is defined as $\frac{\sum_{i,j} \mathbb{I}(m_l(i,j) \neq 0)}{\theta_l}$. The average sparsity level in the network, s, is then defined as the ratio of the total number of zero parameters to the total number of parameters. This is expressed as $s = \frac{\sum_{l=1}^{L} \sum_{i,j} \mathbb{I}(m_l(i,j) \neq 0)}{\Theta_{\mathcal{N}}}$. Below, we characterize the different scenarios when training with different sparse training methods:

- **Random Static Sparsity:** In this case, the sparsity distribution is uniform, ensuring that the sparsity in each layer matches the target sparsity level. Consequently, the application of Sparse-IFT, parameterized by the sparsity level, maintains Iso-FLOP equivalence to the original dense model. However, adhering to common practice for computer

vision networks (e.g., ResNet), we retain the first and last layers (input convolution and output linear layer) as dense to prevent a significant decline in model quality during pre-training. Consequently, the Sparse-IFT network deviates from Iso-FLOP to the dense model, introducing additional FLOPs that need consideration.

- **Pruning at Initialization:** The algorithms, such as SNIP (Lee et al., 2018), GraSP (Wang et al., 2020a), FORCE (de Jorge et al., 2020), etc., introduce distinct criteria or methods for determining which weights to prune at initialization, influencing the sparsity distribution. Consequently, the inherent characteristics of these algorithms result in changes to the sparsity distribution. In the context of Sparse-IFT, despite having an identical total sparse parameter count to the original dense model, the Sparse-IFT network no longer maintains Iso-FLOP equivalence.

- **Dense-to-Sparse Training:** Sparse training methods, such as GraNet (Liu et al., 2021a), employ dense-to-sparse training, initiating training from either a fully dense state or a state less sparse than the target sparsity level. For instance, GraNet utilizes gradual magnitude pruning (Zhu & Gupta, 2017) at the beginning of training to systematically reduce the network's density to the target sparsity level. Consequently, in the context of Sparse-IFT networks, this configuration no longer maintains Iso-FLOP equivalence to the dense model, as the average training FLOPs surpass those of the original dense model.

To address the FLOPs discrepancy between the Sparse-IFT network trained with non-uniform sparsity distributions (e.g., PaI methods or densifying certain layers) and dense-to-sparse training (e.g., GraNet), we employ a binary search to fine-tune the target sparsity of the network prior to any training. In this process, we set the maximum and minimum values for the target sparsity level. At each iteration, we profile the FLOPs used by the Sparse-IFT network and compare it to the original dense model FLOPs. The target sparsity level is adjusted through the binary search, ensuring that the total FLOPs of the Sparse-IFT network are within 0.0001% of the dense model FLOPs.

B. Graph Analysis of Sparse-IFT with DST

In our analysis, we interpret the Sparse-IFT ResNet-18 models as a series of bipartite compute graphs, where each layer, $\{\theta_1, \ldots, \theta_L\}$ in an L layered sparse DNN, takes the form of a square adjacency matrix A. The Ramanujan gap is defined as $\Delta r = 2 * \sqrt{d-1} - \hat{\mu}(A)$ (Hoang et al., 2023b;a), where d is the average edge per node, and $\hat{\mu}(A)$ is the non-trivial eigenvalue of A. Also, we analyze the Iterative Mean Difference Bound, $\Delta r_{imdb} = \frac{1}{|K|}\sum_{i=1}^{|K|}(2\sqrt{d_i - 1} - \hat{\mu}(A_{K_i}))$ (Hoang et al., 2023b). We train a ResNet-18 model with all members of the Sparse-IFT family using a dynamic sparse training algorithm (i.e., RigL (Evci et al., 2020)).

Δr Analysis: In Figure 6, we observe that Δr decreases over the course of training and then maximizes at later stages, which suggests that the spectral properties of the adjacency matrices are changing dynamically during training. The fact that Δr maximizes at later stages and correlates with the Sparse-IFT ResNet-18 model achieving the highest test accuracy indicates a potential connection between the spectral properties of the adjacency matrices and the model's performance. The dynamic changes in Δr might indicate that the neural network is adapting its structure during training. The network might be pruning less important connections and reinforcing more important ones, leading to an optimized structure. Moreover, the increase in Δr could be related to implicit regularization effects. The spectral properties of the adjacency matrices may play a role in controlling the model's capacity, preventing overfitting, and enhancing generalization. The correlation between the maximization of Δr at the later stages of training and the highest test accuracy suggests that there is a relationship between the identified spectral properties and the performance of the Sparse-IFT ResNet-18 model. The maximization of Δr could represent an optimal point in the trade-off between sparsity and model accuracy for the given task.

Δr_{imdb} Analysis: The increasing trend of Δr_{imdb} during training suggests that the overall connectivity boundary across subgraphs is progressively being enhanced. This could imply that the network is learning to establish more meaningful and relevant connections within its structure as training progresses. The DST algorithm may be facilitating an adaptive refinement of connectivity within the network. The observed increase in Δr_{imdb} could indicate that the model is iteratively adjusting its connectivity boundaries to improve information flow. Δr_{imdb} evaluates the average connectivity boundary across all subgraphs, providing a more comprehensive measure of the network's overall connectivity changes. The correlation with the highest performing models at the final stage of training suggests that the average connectivity enhancements captured by Δr_{imdb} are beneficial for the model's performance.

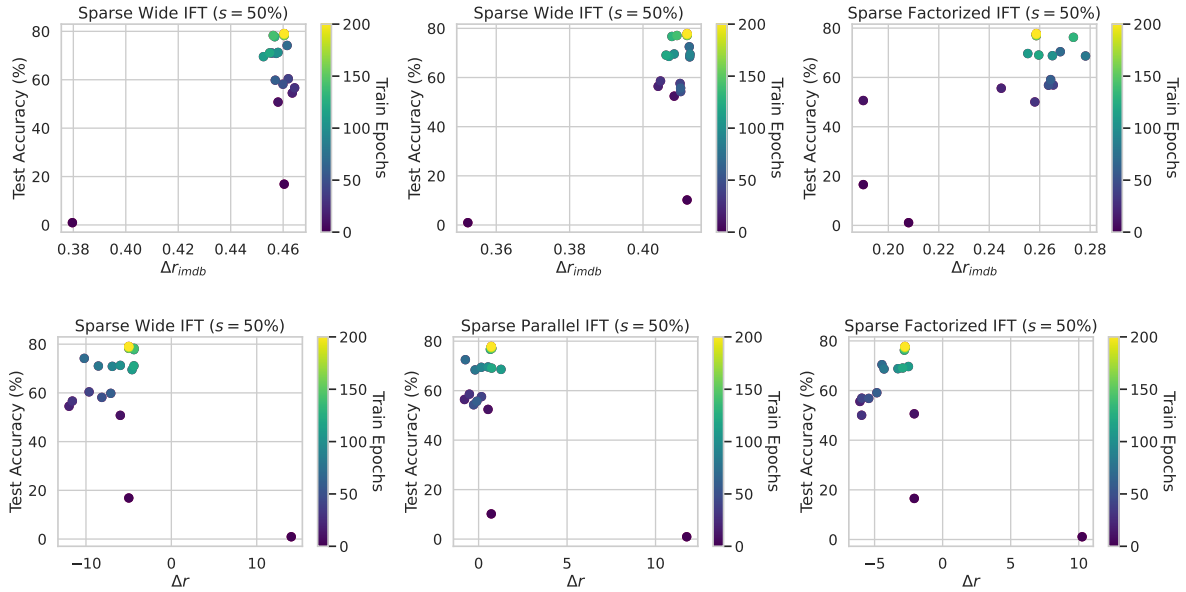

Figure 6: Top row illustrates the dynamic interplay between the Iterative Mean Difference Bound, Δr_{imdb} and test accuracy, and bottom row shows the correlation between the Ramanujan Gap, Δr and test accuracy throughout the training process. This illustrates the evolving relationship between spectral graph properties and network performance, shedding light on the connectivity dynamics of the Sparse-IFT networks trained with DST.

C. Computer Vision: Experimental Settings

C.1. Computer Vision: Pre-Training Settings

CIFAR-100 Our implementation of CIFAR-100 follows the setup from (DeVries & Taylor, 2017) for ResNets. We train the models for 200 epochs with batches of 128 using SGD, Nesterov momentum of 0.9, and weight-decay of 5×10^{-4}. The learning rate is initially set to 0.1 and is scheduled to decay to decrease by a factor of 5x after each of the 60th, 120th, and 160th epochs. Following recent advances in improving ResNets, we initialize the network with Kaiming He initialization (He et al., 2016), zero-init residuals (He et al., 2019), and disable weight-decay in biases and BatchNorm (Ioffe & Szegedy, 2015) layers. For CIFAR-100 experiments with MobileNetV2, MobileViT-S, and BotNet-50, we follow the same training setup used for ResNet, but the learning rate is scheduled via cosine annealing.

ImageNet Our implementation of ImageNet follows the standard setup from (Krizhevsky et al., 2012; Simonyan & Zisserman, 2014). The image is resized with its shorter side randomly sampled in [256, 480] for scale augmentation (Simonyan & Zisserman, 2014). A 224 $\times$ 224 crop is randomly sampled from an image or its horizontal flop, and then normalized. For evaluation, the image is first resized to 256 $\times$ 256, followed by a 224 $\times$ 224 center crop, and then normalized. Following recent advances in improving ResNets, we initialize the network with Kaiming He initialization (He et al., 2016) and zero-init residuals (He et al., 2019).

For ResNets, we replicate the settings recommended by Nvidia (Nvidia, 2019), which uses the SGD optimizer with a momentum of 0.875 and weight decay of $3.0517578125\times10^{-5}$. We disable weight-decay for biases and BatchNorm layers. The model is trained with label smoothing (Szegedy et al., 2016) of 0.1 and mixed precision (Micikevicius et al., 2018) for the standard 90 epochs using a cosine-decay learning rate schedule with an initial learning rate of 0.256 for a batch size of 256. Srinivas et al. (2021) follow the same setup as ResNet for training BotNet-50 on ImageNet, therefore we maintain the same hyperparameter settings as Nvidia (2019) for our BotNet-50 ImageNet experiments.

Sparsity Setup For enabling the Sparse-IFT transformations, we use the RigL (Evci et al., 2020) algorithm in its default hyperparameter settings ($\alpha = 0.3, \Delta T = 100$), with the drop-fraction (α) annealed using a cosine decay schedule for 75% of the training run. We keep the first and last layers (input convolution and output linear layer) dense to prevent a significant degradation in model quality during pre-training, which is standard practice. We account for these additional dense FLOPs by increasing the sparsity in the remaining layers, similar to Gale et al. (2019) and Liu et al. (2022c).

C.2. Importance of Non-linearity

We use BatchNorm (Ioffe & Szegedy, 2015) followed by ReLU (Nair & Hinton, 2010) as a non-linearity. We provide an extended set of empirical results in Table 7 to help validate the importance of training with and without non-linearity by training configurations of the Sparse Parallel, Factorized, and Doped IFT families at different levels of sparsity. The results without non-linear activation functions are often worse than the dense accuracy (77%) across all Sparse-IFT family transformations. We omit Sparse Wide in Table 7 because here we increase the number of channels in the convolutional layers while maintaining the existing architecture.

Table 7: Evaluation on the importance of utilizing the non-linear activation across different members of Sparse-IFT with ResNet-18 on CIFAR100 across different values of sparsity (columns). Non-linear activations enhance the representational capacity of Sparse-IFT, leading to higher accuracy. All reported results are the average over 3 random seeds.

Transformation	Non-linear activation	0.50	0.75	0.90
Sparse Factorized	✗	75.9 ± 0.3	76.6 ± 0.4	76.5 ± 0.4
	✓	**77.8 ± 0.4**	**78.4 ± 0.5**	**78.9 ± 0.5**
Sparse Parallel	✗	77.1 ± 0.1	77.2 ± 0.2	77.6 ± 0.1
	✓	**77.9 ± 0.2**	**79.1 ± 0.2**	**78.2 ± 0.2**
Sparse Doped	✗	77.3 ± 0.2	77.1 ± 0.1	76.5 ± 0.2
	✓	**78.2 ± 0.1**	**77.8 ± 0.1**	**76.9 ± 0.2**

C.3. Computer Vision

C.3.1. Sparse-IFT vs. Extended Sparse Training Schedules

We provide a direct comparison with sparse training methods (e.g., RigL and SET) in the Iso-FLOP setting (i.e., training with a longer schedule) to demonstrate the significance of our results with respect to this standard sparse baselines. As shown in the Table 8, Sparse-IFTs outperform dynamic sparse training methods by a significant margin across all levels of sparsity. Note, at higher levels of sparsity (e.g., 90%), sparse training methods obtain worse accuracy compared to the FLOP equivalent dense baseline. In contrast, with Sparse-IFT, we observe higher accuracy across all levels of sparsity evaluated.

C.3.2. Sparse-IFT on Efficient Computer Vision Architectures

Here, we provide an extended set of results on MobileNetV2, MobileViT-S, and BotNet-50 on CIFAR-100. In particular, we enable Sparse Wide and Sparse Parallel IFT at 50% and 75% sparsity values (see Table 9).

C.3.3. Evaluation of Sparse-IFT with Structured Sparsity

Block Sparsity To derive Iso-FLOP configurations with block sparsity, we reuse the analysis done previously with unstructured sparsity (see Section 2.2) and express the width scaling as a function of sparsity. However, we will search for a block sparse mask during training instead of an unstructured sparsity mask. We use the method proposed by Hubara et al. (2021) to search N:M transposable sparsity, which can accelerate both the forward and backward pass during training on NVIDIA GPUs with Tensor Cores. We use 4:8-T, 2:8-T, and 1:8-T block patterns to obtain 50%, 75%, and 87.5% sparsity, respectively. Note the 1:8-T block is the closest approximation to a 90% sparsity pattern attainable with a block size of 8.

Table 8: Results with ResNet-18 on CIFAR-100 across different values of sparsity (columns). Best accuracy for each sparse training method is highlighted in bold. The original dense ResNet-18 model obtains an accuracy of 77.0±0.2. All reported results are over 3 random seeds.

Dense	Transformation	Sparse Training Method	Epochs	0.50	0.75	0.90
77.0 ± 0.2	Sparse Wide	SET	$200 \cdot \frac{1}{1-s}$	**78.7 ± 0.2**	78.4 ± 0.1	76.8 ± 0.1
	Sparse Wide	RigL	$200 \cdot \frac{1}{1-s}$	**78.9 ± 0.1**	78.8 ± 0.1	76.4 ± 0.2
	Sparse Wide	RigL	200	79.1 ± 0.2	79.5 ± 0.1	**80.1 ± 0.2**

Table 9: Evaluation of Sparse Wide and Sparse Parallel IFT with various compute efficient architectures on CIFAR-100 across different values of sparsity (columns). Using Sparse Parallel IFT, all architectures outperform the dense baseline by a significant margin.

	Dense	Transformation	0.50	0.75
MobileNetV2	72.4 ± 0.2	Sparse Wide	73.4	**73.7**
		Sparse Parallel	72.9	**73.3**
MobileViT-S	73.5 ± 0.1	Sparse Wide	74.6	**74.8**
		Sparse Parallel	73.7	**74.4**
BotNet-50	79.8 ± 0.2	Sparse Wide	80.3	**80.6**
		Sparse Parallel	79.7	**80.5**

Table 10: Comparison of structured sparse and unstructured sparse methods on CIFAR-100 test accuracy on ResNet-18.

Transformation	Sparsity Type	Sparsity	Width Scaling Factor 1x	1.41x	2x	3.16x
Low Rank, Linear	Structured	0%	74.1	74.3	74.3	73.4
Low Rank, Non-Linear	Structured	0%	76.8	76.5	76.0	75.3
Sparse Wide	N:M Block Sparse (Hubara et al., 2021)	4:8-T		77.1		
		2:8-T			**78.4**	
		1:8-T				78.1
	Unstructured Sparse (Evci et al., 2020)	50%		79.1		
		75%			79.5	
		90%				**80.1**

We also set up and experimented using the method proposed by Jiang et al. (2022) to train with fine-grained sparse block structures dynamically. However, the algorithm uses agglomerative clustering which led to a much slower runtime and quickly ran out of memory even at 50% sparsity using the Sparse Wide IFT on a single Nvidia V100 (16 GB).

Low Rank Let k_{lr} be the factor with which we widen all layers' input and output dimensions for low-rank factorization. We replace all dense layers with low-rank factorization, i.e. $\theta_l^{lr} = U_l V_l^T$, where $U_l \in \mathbb{R}^{(k_{lr}.D_{in})\times d}$ and $V_l \in \mathbb{R}^{(k_{lr}.D_{out})\times d}$. Given a widening factor and equating the FLOPs of this transformation to that of a dense transformation f_θ, we obtain the following expression for rank d: $\frac{D_{in}.D_{out}.k_{lr}}{(D_{in}+D_{out}}$. We evaluate this factorization across different values of width-scaling k_{lr} in Table 10.

C.3.4. Evaluation on downstream tasks

COCO Object Detection

This dataset contains 118K training, 5K validation (`minival`), and 20K test-dev images. We adopt the standard single-scale training setting (Lin et al., 2017a) where there is no additional data augmentation beyond standard horizontal flipping. For training and testing, the input images are resized so that the shorter edge is 800 pixels (Lin et al., 2017a). The model is trained with a batch size of 16, using the SGD optimizer with a momentum of 0.9 and weight decay of 1×10^{-4}. We follow the standard 1x schedule (12 epochs) using a step learning rate schedule, with a 10x decrease at epochs 8 and 11, an initial learning rate warmup of 500 steps starting from a learning rate of 2×10^{-5}, and a peak learning rate of 0.01.

Table 11: Object detection results on COCO `minival` in the RetinaNet framework. Sparse Wide IFT configurations of RetinaNet outperform the dense baseline by a large margin on all metrics while using similar FLOPs.

Backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Dense	29.3	46.2	30.9	14.7	31.5	39.6
Sparse Wide (50%)	31.3	49.0	33.0	16.6	34.0	42.0
Sparse Wide (75%)	32.8	51.0	34.8	17.3	35.8	43.3
Sparse Wide (90%)	**34.5**	**53.5**	**36.5**	**18.6**	**37.6**	**45.3**

CITYSCAPES SEMANTIC SEGMENTATION

Setup We follow the same training protocol as (Zhao et al., 2017), where the data is augmented by random cropping (from 1024 × 2048 to 512 × 1024), random scaling in the range [0.5, 2], and random horizontal flipping. The model is trained with a batch size of 16, using the SGD optimizer with a momentum of 0.9 and weight decay of 5×10^{-4}. We follow the 80K iterations setup from MMSegmentation with an initial learning rate of 0.01 annealed using a poly learning rate schedule to a minimum of 1×10^{-4}. Similar to most setups that tune hyperparameters (Zhao et al., 2017; Liu et al., 2021c; Wang et al., 2020b) for reporting the best results, we tune the learning rate for all our models. All our results are reported using a learning rate of 0.03 for the sparse backbones and 0.01 for the dense baseline.

Table 12: Semantic segmentation results on the Cityscapes `val` set using DeepLabV3+. Sparse Wide IFT configurations ResNet-18 backbones outperform the dense baseline on all metrics while using similar FLOPs.

Backbone	mIoU	mAcc
Dense	76.72	84.40
Sparse Wide (50%)	77.90	85.12
Sparse Wide (75%)	78.92	85.68
Sparse Wide (90%)	**79.10**	**86.01**

D. Natural Language Processing: Experimental Settings

D.1. Details for GPT End-to-End Training

We demonstrate the benefits of using Sparse-IFT transformations in the NLP domain by pre-training GPT-3 models and performing zero-shot eval on downstream tasks from the HuggingFace Open LLM leaderboard. Here, we pre-train the models on the Pile (Gao et al., 2020) dataset. To train all GPT models, we use the AdamW optimizer (Loshchilov & Hutter, 2017) with $\beta_1 = 0.9$, $\beta_2 = 0.95$ and $\epsilon = 10^{-8}$. The global norm is clipped at 1.0, and a weight decay of 0.1 is used. There is a learning rate warmup over the first 375M tokens, followed by a cosine decay to 10% of the peak learning rate. We follow the recently published Chinchilla (Hoffmann et al., 2022) recommendations for obtaining loss-optimal pre-trained baseline configurations of models. The context window size is 2048 following (Brown et al., 2020). Table 13 shows a detailed breakdown of the model architectures, learning rate, and training settings.

In Tables 13 and 14, we outline the architecture configurations for Sparse Wide IFT 50% and 75% variants. We train the Sparse Wide GPT-3 models using the dynamic sparse training algorithm, SET (Mocanu et al., 2018) on the Cerebras CS-2 to realize the acceleration from unstructured sparsity. Currently, Cerebras CS-2's specialized kernels support training with dynamic unstructured sparsity via SET; therefore, results in this section are reported with SET. In Table 13, n_{params} is the total number of trainable parameters, n_{layers} is the number of decoder layers, and d_{model} is the base size of the model. The feedforward bottleneck is four times the base size, i.e., $d_{\text{ff}} = 4 \times d_{model}$. Finally, n_{heads} is the number of attention heads, and d_{head} is the dimension of each attention head.

Evaluation We conducted a comprehensive evaluation of both dense and Sparse Wide IFT GPT-3 Small models, assessing their performance at 50% and 75% sparsity levels across five distinct tasks on the Open LLM leaderboard (Beeching et al., 2023) using the LM-eval-harness (Gao et al., 2021). The tasks encompassed ARC (Clark et al., 2018), HellaSwag (Zellers et al., 2019), TruthfulQA (Lin et al., 2022), MMLU (Hendrycks et al., 2021), and Winogrande (Sakaguchi et al., 2019). In Table 15, our results reveal that the Sparse IFT GPT-3 Small model at 75% sparsity achieved a notable 0.9% improvement over the dense baseline, underscoring the efficacy of Sparse Wide IFT in enhancing model performance across a diverse range of language understanding tasks.

Table 13: Size, architecture, and learning hyperparameters (batch size and learning rate) of the GPT-3 Small model, which is trained using Chinchilla optimal configurations (≈ 20 tokens per parameter)

Model	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate	Training Tokens
GPT-3 Small	125M	12	768	12	64	256	6×10^{-4}	2.5B

Table 14: Sizes and architecture definitions of the dense GPT-3 Small model and its Sparse Wide IFT variants.

MODEL	TRANSFORMATION	SPARSITY	n_{layers}	d_{model}	d_{FF}	n_{heads}	d_{head}
GPT-3 SMALL	DENSE	0%	12	768	3072	12	64
GPT-3 SMALL	SPARSE WIDE	50%	12	1092	4344	12	64
GPT-3 SMALL	SPARSE WIDE	75%	12	1536	6144	12	64

Table 15: Performance Evaluation of Dense and Sparse Wide IFT GPT-3 Small Models at 50% and 75% sparsity levels across five tasks (i.e., ARC, HellaSwag, TruthfulQA, MMLU, and Winogrande) on the Open LLM Leaderboard

MODEL	TRANSFORMATION	SPARSITY	SPARSE METHOD	OPEN LLM LEADERBOARD					
				ARC	HELLASWAG	TRUTHFULQA	MMLU	WINOGRANDE	AVERAGE
GPT-3 SMALL	DENSE	0%	-	20.8	27.2	47.0	24.6	49.4	33.8
	SPARSE WIDE	50%	SET	20.6	27.4	47.4	25.6	49.6	34.1
	SPARSE WIDE	75%	SET	**22.1**	**27.8**	**47.5**	**25.6**	**50.4**	**34.7**

E. Benchmarking Efficiency w.r.t Wall-clock

In this section we provide additional details on the benchmarking setups for inference on Neural Magic DeepSparse (Neural-Magic, 2021; Iofinova et al., 2021; Kurtz et al., 2020) sparsity-aware runtime and training on the Cerebras CS-2 (Lie, 2023; Cerebras, 2023) for evaluating the efficiency of Sparse-IFT with respect to the wall-clock time.

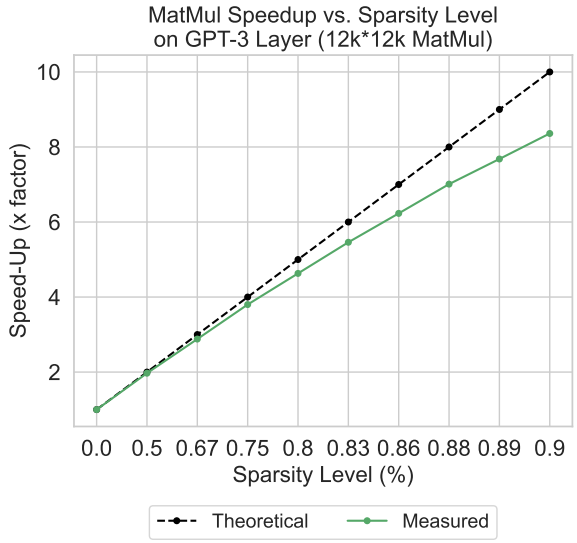

Figure 7: Measured speedup versus theoretical speedup at varying sparsity levels for a GPT-3 layer 12k $\times$ 12k matrix multiplication (MatMul) (Lie, 2021).

Inference Setup We use Neural Magic's DeepSparse tool for benchmarking Sparse-IFT variants. The benchmarking is conducted on the Intel Cascade Lake CPUs found on AWS G4dn cloud instances. These instances support the AVX-512 instruction set, which is used by the DeepSparse inference runtime to accelerate unstructured sparsity. We benchmark different configurations of the Sparse Wide ResNet-18 model with sparsity $\in \{50\%, 75\%, 90\%\}$ for batched inference on ImageNet. We report runtime for batch-inference of 64 images at 224 $\times$ 224 resolution.

Training Setup We evaluate the training efficiency of Sparse-IFT on the Cerebras CS-2 which supports and accelerates training with unstructured sparsity (both forward and backward passes). We benchmark the training speed measured in seconds/iteration. Note that the overall FLOPs of models in the GPT family are comprised of matrix multiplication FLOPs and attention FLOPs. Attention FLOPs (i.e., spent in multi-head attention) scale quadratically with sequence length and are invariant to weight sparsity. To demonstrate the efficacy of sparse kernels for unstructured weight sparsity, we report our

results for dense and Sparse Wide variants of the GPT-3 20B model with a sequence length of 256 and batch size of 256. We benchmark different configurations of Sparse Wide GPT-3 20B with sparsity $\in \{50\%, 75\%, 90\%\}$ and report seconds/iteration.

Benchmarking Analysis Figure 5 in Section 6 presents the results of benchmarking inference and training of Sparse-IFT Sparse Wide family. In both setups, we measure the relative increase in latency or training speed for Sparse-IFT variants against the dense model. Note that configurations of Sparse-IFT at different values of sparsity do not incur a significant change in the FLOPs compared to the dense model. On ideal hardware, FLOPs should translate directly to wall clock time, and hence, the inference latency or training time for all configurations of Sparse-IFT should be the same as that of the dense model (dotted black line). Conversely, when hardware does not support unstructured sparsity, the latency or training time of Sparse-IFT variants increases with sparsity (blue line).

The results in Figure 5 of Section 6 show that up 75%, there is minimal computational overhead compared to training the original dense baseline model. At 90% sparsity, our results lie between these two spectrums (green line). Using Neural Magic's sparse inference runtime, we observe a significant reduction in inference latency, bringing down the relative increase in latency from 19.5x to 3.5x. Similiarly, in the case of training on the Cerebras CS-2, we observe a significant reduction in training-time, bringing down the relative increase from 10.6x to 2.8x. The dense GPT-3 model achieved a throughput of 828.71 iterations per second on the CS-2, while the Sparse Wide IFT variants recorded throughputs of 637.5, 595.3, and 294.3 at respective sparsity levels, resulting in overheads of 1.30x, 1.39x, and 2.82x, respectively.

In Figure 7, we illustrate the achievable benefits of unstructured weight sparsity when utilizing specialized hardware designed for deep learning, such as the Cerebras CS-2. This figure was regenerated based on the plot in (Lie, 2021).

F. Author Contributions

We provide a summary of each author's contributions:

- Vithursan Thangarasa was an integral part of the project by participating in discussions with Shreyas Saxena and contributing to the method. He also implemented all Sparse-IFT transformations in PyTorch, proposed using non-linearity in Sparse-IFT, analyzed DST methods via spectral analysis, conducted experiments for the entire study on CIFAR-100 and its ablations, obtained initial results on ImageNet, extended Sparse-IFT to efficient architectures (e.g., BotNet, MobileViT), and contributed to writing several sections of the manuscript.
- Shreyas Saxena conceived the key idea of matching the FLOPs of Sparse Wide transformation to a compact dense model, extended the idea to other members of the Sparse-IFT family, helped with the implementation, established cardinality of Sparse-IFT members to explain the results, benchmarked Sparse-IFT for inference, and wrote several sections of the manuscript.
- Abhay Gupta validated sparse optimizers in PyTorch, conducted experiments with Sparse-IFT ResNet variants on ImageNet, helped with pre-training of Sparse-IFT variants of GPT on Cerebras CS-2, conducted all experiments of Sparse-IFT on downstream CV tasks, and contributed to writing parts of the manuscript.
- Sean Lie helped with the bring-up of sparsity support on Cerebras CS-2 which was crucial for benchmarking and training Sparse-IFT variants of GPT models, and provided feedback to improve the structuring and presentation of the manuscript.