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.

B Computer Vision: Experimental Settings

B.1 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 10 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 10 because here we increase the number of channels in the convolutional layers while maintaining the existing architecture.

Table 10: 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
	✓	$\mathbf{77.8 \pm 0.4}$	$\mathbf{78.4 \pm 0.5}$	$\mathbf{78.9 \pm 0.5}$
Sparse Parallel	✗	77.1 ± 0.1	77.2 ± 0.2	77.6 ± 0.1
	✓	$\mathbf{77.9 \pm 0.2}$	$\mathbf{79.1 \pm 0.2}$	$\mathbf{78.2 \pm 0.2}$
Sparse Doped	✗	77.3 ± 0.2	77.1 ± 0.1	76.5 ± 0.2
	✓	$\mathbf{78.2 \pm 0.1}$	$\mathbf{77.8 \pm 0.1}$	$\mathbf{76.9 \pm 0.2}$

B.2 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. (2017); Simonyan & Zisserman (2014). The image is resized with its shorter side randomly sampled in [256, 480] for scale augmentation Simonyan & Zisserman (2014). A 224×224 crop is randomly sampled from an image or its horizontal flop, and then normalized. For evaluation, the image is first resized to 256×256, followed by a 224×224 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 (2019b), 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 (2019b) 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. (2022b).

B.3 Computer Vision

B.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

Table 11: 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 Parallel	RigL	200	79.1 ± 0.2	79.5 ± 0.1	**80.1 ± 0.2**

respect to this standard sparse baselines. As shown in the Table 11, 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.

B.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 12).

Table 12: 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**

B.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 2.4) 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. 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 13.

Table 13: 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**

B.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 14: 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 Segmenation

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.

C Natural Language Processing: Experimental Settings

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

Our end-to-end training setup for GPT-3 on WikiText-103 follows a similar procedure to Dao et al. (2022). We use a batch size of 512 and train with the AdamW optimizer for 100 epochs. Also, we use a learning rate warmup for 10 epochs and a weight decay of 0.1. To discover good hyperparameters, we perform a grid search to discover an appropriate learning rate among {8e-3, 6e-3, 5.4e-3, 1.8e-3,

Table 15: 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**

6e-4, 2e-4, 6e-5} that led to the best perplexity for a given compute budget on the validation set. In Table 16, we outline the architecture configurations for the original dense model and its Sparse Wide IFT 50% and 75% variants.

Table 16: 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

WikiText-103 End-to-End Training Results We highlight that in Table 17, the Sparse Wide IFT GPT-3 Small at 50% sparsity attains a better perplexity on WikiText-103 while using 2.4x fewer training FLOPs than the GPT-3 Medium dense model. In this setup, using Sparse Wide transformation does not change the FLOP of the dense layer, but this leads to a slight increase in the attention FLOPs. This explains the 1.17x increase in FLOPs between the GPT-3 Small Sparse Wide at 50% sparsity and the dense GPT-3 Small model. Note, out of all the Sparse-IFT transformations, this increase only occurs in the Sparse Wide IFT.

Table 17: Details on the total training FLOPs for each GPT-3 model tested. We note that the reported FLOPs per sequence (seq) include both forward and backward passes. The reported perplexity (lower is better) is on the WikiText-103 test set over 3 random seeds.

Model	Transformation	Sparsity	Total Seqs	Total FLOPs/ Seq	Total FLOPs	Total exaFLOPs	Perplexity
GPT-3 Small	Dense	0%	2.28e6	8.763e11	2.0011e18	2.00	20.8 ± 0.3
GPT-3 Small	Sparse Wide	50%	2.28e6	1.029e12	2.3498e18	2.35	$\mathbf{20.4 \pm 0.2}$
GPT-3 Medium	Dense	0%	2.28e6	2.4845e12	5.6734e18	5.67	20.5 ± 0.2

C.2 Details for Sparse Pre-training and Dense Fine-tuning (Thangarasa et al., 2023)

We provide an extended set of results that showcase the added benefit of using Sparse-IFT transformations. Here, we apply the Sparse Pre-training and Dense Fine-tuning (SPDF) framework introduced by Thangarasa et al. (2023). In this setup, all models are pre-trained under a similar FLOP budget. However, during the fine-tuning stage, Sparse-IFT models have extra representational capacity which can be enabled by allowing the zeroed weights to learn (i.e., dense fine-tuning). Even though the fine-tuning FLOPs are more than the original dense model, we leverage Sparse-IFT method's extra capacity to obtain accuracy gains on the downstream task. To ensure a fair baseline, we also compare dense fine-tuning to sparse fine-tuning (i.e., pre-trained model remains as-is) similar to Thangarasa et al. (2023).

C.2.1 SPDF ON BERT

Experimental Setup We train BERT models using the open-source LAMB (You et al., 2020) implementation provided by Nvidia (2019a). In this setup, BERT is pre-trained on the BookCorpus (Zhu et al., 2015) and Wikipedia datasets in two phases. In the first phase, models are trained for 82% of total iterations with a sequence length of 128. In the second phase, models are trained for the remaining 18% of iterations with sequence length 512. We use a batch size of 8192 and 4096 in phase 1 and phase 2, respectively. Table 18 shows details of the size and architecture of the BERT Small model. For finetuning models on SQuADv1.1 (Rajpurkar et al., 2016), we train for two epochs with AdamW optimizer and use a grid search to tune the learning rate and batch size.

Table 18: Size and architecture of the BERT Small model, which is trained using the setup from Nvidia (2019a)

Model	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}
BERT Small	29.1M	4	512	8	64

SPDF on SQuADv1.1 Results We evaluate BERT Small with Sparse Wide, Sparse Parallel, and Sparse Factorized members of the Sparse-IFT family. All transformations, except Sparse Parallel, perform comparably to the dense baseline on SQuAD. Unlike CV architectures, BERT initializes the layers with a normal distribution, which has an adverse effect when layers undergo shape transformations (e.g., changes in depth Zhang et al. (2019), or width Yang et al. (2022)). In our initial experiments, we found changing the initialization of BERT enables other families to outperform the dense baseline. In addition to initialization, BERT training has over six hyperparameters. We leave optimizing and analyzing the effect of these hyperparameters on Sparse-IFT for future work and restrict our current scope to demonstrating gains without tuning any hyperparameters. Using the Sparse Parallel IFT with 50% sparsity leads to a 0.7% improvement in the exact match (EM) accuracy over the dense baseline (see Table 19).

Table 19: Evaluation of Sparse Parallel IFT for pre-training BERT Small. We report EM (higher is better) obtained by sparse fine-tuning and dense fine-tuning BERT models on SQuADv1.1, respectively.

Dense	Transformation	Fine-Tuning Method	0.50	0.75
70.6	Sparse Parallel	Sparse	**70.7**	69.9
		Dense	**71.3**	70.8

C.2.2 SPDF ON GPT

Pre-training Experimental Setup Here, we pre-train the models on the Pile Gao et al. (2020) dataset. To train all GPT models, we use AdamW optimizer Loshchilov & Hutter (2017) with $\beta_1 = 0.9$, $\beta_2 = 0.999$ 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 20 shows a detailed breakdown of the model architectures, learning rate, and training settings. In Table 16, we outline the architecture configurations for Sparse Wide IFT 50% and 75% variants.

Table 20: Size, architecture, and learning hyperparameters (batch size and learning rate) of the GPT-3 Small model, which is trained using Chinchilla optimal configurations ($\approx$ 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

Fine-tuning Experimental Setup We finetune the Sparse Wide IFT variants of GPT-3 Small on the WikiText-103 (Merity et al., 2017) dataset following the setup presented in (Rae et al., 2021). We finetune for ten epochs and perform early stopping once the models overfit. We performed a grid search to discover an appropriate learning rate that led to the best perplexity for a given compute budget. More specifically, on the dense baseline and Sparse Wide IFT variants, we use a batch size of 32 and select the best learning rate among {5e-3, 3e-3, 1e-3, 3e-4, 1e-4, 3e-5, 1e-5} on the validation set.

In Tables 16, 18, and 20, 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.

SPDF on WikiText-103 Results Here, we pre-train a GPT-3 Small architecture with Sparse Wide IFTs at 50% and 75% sparsity. Post pre-training, we finetune our models on WikiText-103. The GPT-3 Small 75% Sparse Wide model reduces the perplexity (PPL) by a noticeable 1.3 points compared to dense (refer to Table 21).

Table 21: Evaluation of Sparse Wide IFT for pre-training GPT-3 Small. We report perplexity (lower is better) obtained by sparse fine-tuning and dense fine-tuning GPT models on Wikitext-103, respectively.

Dense	Transformation	Fine-Tuning Method	0.50	0.75
15.9	Sparse Wide	Sparse	**15.6**	16.0
		Dense	15.1	**14.6**