AdapMTL: Adaptive Pruning Framework for Multitask Learning Model

Mingcan Xiang
mingcanxiang@umass.edu
University of Massachusetts Amherst
Amherst, MA, USA

Jiaxun Tang
jtang@umass.edu
University of Massachusetts Amherst
Amherst, MA, USA

Qizheng Yang
qizhengyang@umass.edu
University of Massachusetts Amherst
Amherst, MA, USA

Hui Guan
huiguan@umass.edu
University of Massachusetts Amherst
Amherst, MA, USA

Tongping Liu
tongping@umass.edu
University of Massachusetts Amherst
Amherst, MA, USA

Abstract

In the domain of multimedia and multimodal processing, the efficient handling of diverse data streams such as images, video, and sensor data is paramount. Model compression and multitask learning (MTL) are crucial in this field, offering the potential to address the resource-intensive demands of processing and interpreting multiple forms of media simultaneously. However, effectively compressing a multitask model presents significant challenges due to the complexities of balancing sparsity allocation and accuracy performance across multiple tasks. To tackle the challenges, we propose AdapMTL, an adaptive pruning framework for MTL models. AdapMTL leverages multiple learnable soft thresholds independently assigned to the shared backbone and the task-specific heads to capture the nuances in different components' sensitivity to pruning. During training, it co-optimizes the soft thresholds and MTL model weights to automatically determine the suitable sparsity level at each component to achieve both high task accuracy and high overall sparsity. It further incorporates an adaptive weighting mechanism that dynamically adjusts the importance of task-specific losses based on each task's robustness to pruning. We demonstrate the effectiveness of AdapMTL through comprehensive experiments on popular multitask datasets, namely NYU-v2 and Tiny-Taskonomy, with different architectures, showcasing superior performance compared to state-of-the-art pruning methods.

CCS Concepts

• **Computing methodologies** → **Multi-task learning**.

Keywords

Pruning, Multitask Learning

1 Introduction

In the landscape of multimedia and multimodal processing [2, 40], Deep Neural Networks (DNNs) [46] have emerged as a pivotal technology, powering advancements across a spectrum of applications from image and video analysis to natural language understanding and beyond. Their profound ability to learn and abstract complex features from a range of media forms underpins their utility in diverse domains, including content categorization, recommendation systems, and interactive interfaces. However, as the complexity of tasks grows, so does the demand for larger and more powerful models, which in turn require substantial computational resources, memory usage, and longer training times. This trade-off between performance and model complexity has led to a continuous pursuit of more efficient and compact CNN [24] architectures, as well as innovations in pruning techniques that can maintain high performance without compromising the benefits of the model's scale.

Pruning techniques [13, 19, 23, 25–27, 36, 48] have emerged as a promising approach to compress large models without significant loss of performance. These techniques aim to reduce the size of a model by eliminating redundant or less important parameters, such as neurons, connections, or even entire layers, depending on the method employed [9, 28, 62]. Parameter-efficient pruned models can provide significant inference time speedups by exploiting the sparsity pattern [14, 31, 57, 61]. These models are designed to have fewer parameters, which translates into reduced memory footprint and lower computational complexity (FLOPs) [31]. By leveraging specialized hardware and software solutions that can efficiently handle sparse matrix operations, such as sparse matrix-vector multiplication (SpMV), these models can achieve faster inference times [14, 39, 56]. Additionally, sparse models can benefit from better cache utilization, as they require less memory bandwidth, thereby reducing the overall latency of the computation [41, 61].

Although many techniques have been proposed in the past for pruning a single-task model, there is much less work in pruning a multitask model. Multitask models, which are designed to simultaneously handle multiple tasks, have become increasingly popular due to their ability to share representations and learn more effectively from diverse data sources [16, 65, 68]. These models have found wide-ranging applications where tasks are often related
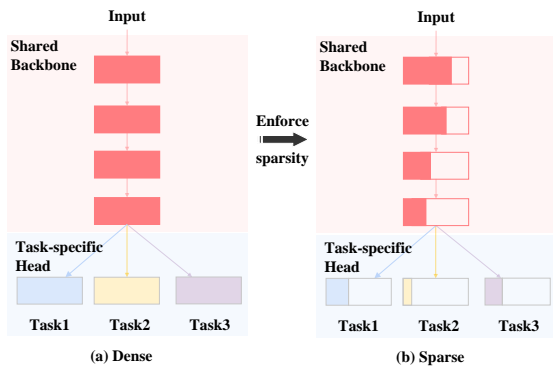
Figure 1: Overview of pruning a dense multitask model. The red parts represent the shared backbone, and the leaf boxes represent the task-specific heads. In the sparse model, the blank spaces indicate the pruned parameters.

and can benefit from shared knowledge [66]. A compact multitask model, which is shown in Figure 1, has the potential to deliver high performance across various tasks while minimizing resource requirements, making it well-suited for deployment on resource-constrained devices or in real-time scenarios.

Traditional pruning techniques, which are primarily focused on single-task models, may not be directly applicable or sufficient for multitasking settings. Recent work has started to explore the intersection of multitask learning and pruning. Disparse [52] considered each task independently by disentangling the importance measurement and taking the unanimous decisions among all tasks when performing parameter pruning and selection. A parameter is removed if and only if it's shown to be not critical for any task. However, as the number of tasks increases, it becomes challenging to achieve unanimous selection agreement among all tasks, which could negatively affect the average performance across tasks. Thus, there is a need for novel compression approaches that cater to the complexities of multitask models, taking into account the inter-dependencies between tasks, the sharing of representations, and the different sensitivity of task heads.

To tackle the challenges, we conduct extensive experiments that reveal two valuable insights on designing an effective multitask model pruning strategy. *First, the shared backbone and the task-specific heads have different sensitivity to pruning and thus should be treated differently.* However, current state-of-the-art approaches do not adequately recognize this aspect, leading to equal treatment of each component during pruning, rather than accounting for their varying sensitivities. *Second, the change in training loss could serve as a useful guide for allocating sparsity among different components.* If the training loss of a specific task tends to be stable, we can prune more aggressively on that component, as the task head is robust to pruning. On the contrary, if the loss of a specific task fluctuates significantly, we should consider pruning less on that component since the training is less likely to converge at higher sparsity levels.

Motivated by these observations, we propose AdapMTL, an adaptive pruning framework for MTL models. AdapMTL dynamically adjusts sparsity across different components, such as the shared backbone and task-specific heads based on their sensitivity to pruning,

while preserving accuracy for each task. This is achieved through a set of learnable soft thresholds [10, 23] that are independently assigned to different components and co-optimized with model weights to automatically determine the suitable sparsity level for each component during training. Specifically, we maintain a set of soft thresholds $\alpha = \{\alpha_B, \alpha_1, \alpha_2, ..., \alpha_T\}$ in each component, where α_B represents the threshold for the shared backbone and α_t represents the threshold for the t-th task-specific head. In the forward pass, only the weights larger than the threshold α will be counted in the model, while others are set to zero. In the backward pass, we automatically update all the component-wise thresholds α, which will smoothly introduce sparsity. Additionally, AdapMTL employs an adaptive weighting mechanism that dynamically adjusts the importance of task-specific losses based on each task's robustness to pruning. AdapMTL does not require any pre-training or pre-pruned models and can be trained from scratch.

We conduct extensive experiments on two popular multitask datasets: NYU-v2 [47] and Tiny-Taskonomy [63], using different architectures such as Deeplab-ResNet34 and MobileNetV2. When compared with state-of-the-art pruning and MTL pruning methods, AdapMTL demonstrates superior performance in both the training and testing phases. It achieves lower training loss and better normalized evaluation metrics on the test set across different sparsity levels. The contributions of this paper are summarized as follows:

(1) We conduct extensive experiments that reveal valuable insights in designing effective MTL model pruning strategies. These findings motivate the development of novel pruning strategies specifically tailored for multitasking scenarios.

(2) We propose AdapMTL, an adaptive pruning framework for MTL models that dynamically adjusts sparsity levels across different components to achieve high sparsity and task accuracy. AdapMTL features component-wise learnable soft thresholds that automatically determine the suitable sparsity for each component during training and an adaptive weighting mechanism that dynamically adjusts task importance based on their sensitivity to pruning.

(3) We demonstrate the effectiveness of AdapMTL through extensive experiments on multitask datasets with different architectures, showcasing superior performance compared to SOTA pruning and MTL pruning methods. Our method does not require any pre-training or pre-pruned models.

2 Related Work

Multitask Learning. Multitask learning (MTL)[1, 4, 12, 65] aims to learn a single model to solve multiple tasks simultaneously by sharing information and computation among them, which is essential for practical deployment. Over the years, various MTL approaches have been proposed, including hard parameter sharing[3], soft parameter sharing [59], and task clustering [22]. In hard parameter sharing, a set of parameters in the backbone model are shared among tasks while in soft parameter sharing, each task has its own set of parameters, but the difference between the parameters of different tasks is regularized to encourage them to be similar. MTL has been successfully applied to a wide range of applications, such as natural language processing [8, 18, 29], computer vision [17, 30, 44, 58], and reinforcement learning [42, 54].

Subsequently, the integration of neural architecture search (NAS) with MTL has emerged as a promising direction. NAS for MTL, exemplified by works like MTL-NAS [15], Learning Sparse Sharing Architectures for Multiple Tasks [51], and Controllable Dynamic Multi-Task Architectures [43], focuses on discovering optimal architectures that can efficiently learn shared and task-specific features. These approaches, including Adashare [53] and AutoMTL [64], demonstrate the potential of dynamically adjusting architectures to the requirements of multiple tasks, optimizing both performance and computational efficiency.

Pruning. Pruning techniques have been widely studied to reduce the computational complexity of deep neural networks while maintaining their performance. Early works on pruning focused on unstructured weight pruning [20, 25], where unimportant weights were removed based on a given criterion, and the remaining weights were fine-tuned. There are different kinds of criterion metrics, such as magnitude-based [20, 27], gradient-based [36, 37], Hessian-based [21], connection sensitivity-based [26, 33, 48], and so on. Other works explored structured pruning [57, 67], which removes entire filters or channels, leading to more efficient implementations on hardware platforms. Recently, the lottery ticket hypothesis [13] has attracted considerable attention, suggesting that dense, randomly initialized neural networks contain subnetworks (winning tickets) that can be trained to achieve comparable accuracy with fewer parameters. This has led to follow-up works [13, 32, 38] that provide a better understanding of the properties and initialization of winning tickets. Single-Shot Network Pruning (SNIP) [26] is a data-driven method for pruning neural networks in a one-shot manner. By identifying an initial mask to guide parameter selection, it maintains a static network architecture during training. Some other work, like the layer-wise pruning method [23], inspiringly attempts to learn a layer-wise sparsity for individual layers rather than considering the network as a whole. This approach allows for fine-grained sparsity allocation across layers. To reduce the total time involved in pruning and training, pruning during training techniques [11, 35, 39] have been proposed to directly learn sparse networks without the need for an iterative pruning and finetuning process. These methods involve training networks with sparse connectivity from scratch, updating both the weights and the sparsity structure during the training process.

Pruning for Multitask Learning. Recently, attention has shifted to the intersection of MTL and pruning techniques. A compact multitask model has the potential to deliver high performance across various tasks while minimizing resource requirements, making it well-suited for deployment on resource-constrained devices or in real-time scenarios. For example, MTP [6] focuses on efficient semantic segmentation networks, demonstrating the potential of multitask pruning to enhance performance in specialized domains. Similarly, the work by Cheng et al.[7] introduces a novel approach to multi-task pruning through filter index sharing, optimizing model efficiency through a many-objective optimization framework. Additionally, Ye et al.[60] propose a global channel pruning method tailored for multitask CNNs, highlighting the importance of performance-aware approaches in maintaining accuracy while reducing model size. Disparse [52] proposes joint learning and pruning methods to achieve efficient multitask models. However, these methods often neglect the importance of the shared backbone, leading to equal treatment of each component during pruning, rather than accounting for their varying importance. Our work aims to address this limitation by adaptively allocating sparsity across the shared backbone and task-specific heads based on their importance and sensitivity.

3 Methodology

3.1 Preliminary

We formulate multitask model pruning as an optimization problem. Given a dataset $\mathcal{D} = \{(x^i;\ y_1^i, y_2^i, ..., y_t^i),\ i \in [1, N]\}$, a set of T tasks $\mathcal{T} = \{t_1\ t_2\ ..., t_T\}$, and a desired sparsity level s (i.e. the percentage of zero weights), the multitask model pruning aims to find a sparse weight W that minimizes the sum of task-specific losses. Mathematically, it is formulated as:

$$
\min_{W} \mathcal{L}(W; \mathcal{D}) = \min_{W} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \mathcal{L}_t(f(W, x^i); y_t^i)
$$
$$
\text{s. t.} \quad W \in \mathbb{R}^d, \quad \|W\|_0 \le (1 - s) \cdot P,
$$
(1)

where the $\mathcal{L}(\cdot)$ is the total loss function, $\mathcal{L}_t(\cdot)$ is the task-specific loss for each individual task t, W are the parameters of neural network to be learned, P is the total number of parameters and $\| \cdot \|_0$ denotes the ℓ_0-norm, i.e. the number of non-zero weights. The key challenge here is how to enforce sparsity on weight W while minimizing the loss. This involves finding an optimal balance between maintaining the performance of each task and pruning the model to achieve the desired sparsity level. We next describe our proposed adaptive pruning algorithm that can effectively handle the unique characteristics of multitask models and efficiently allocate sparsity across different components to preserve the overall model performance.

3.2 Adaptive Multitask Model Pruning

Multitask models typically have a backbone shared across tasks and task-specific heads. We observe that these different model components have different sensitivities to pruning and thus should be treated differently. The challenge lies in how to automatically capture the sensitivity of each model component to pruning and leverage the signal to automatically allocate sparsity across components. To address the challenge, we propose a component-wise pruning framework that assigns different learnable soft thresholds to each component to capture its sensitivity to pruning. The framework then co-optimizes the thresholds with model weights to automatically determine the suitable sparsity level for each component during training.

Specifically, we introduce a set of learnable soft thresholds $\alpha = \{\alpha_B, \alpha_1, \alpha_2, ..., \alpha_T\}$ for each component, where α_B represents the threshold for the shared backbone and α_t represents the threshold for the t-th task-specific head. The thresholds α are determined based on the significance and sensitivity of the respective components and are adaptively updated using gradient descent during the backpropagation process. The soft threshold α_t and sparse weight W_t for each component can be computed as follows:
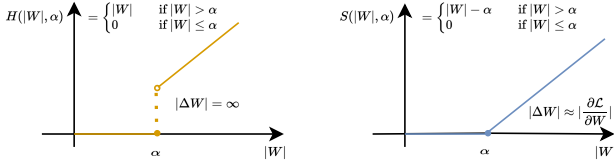
Figure 2: Difference between hard and soft thresholding. Hard thresholding causes abrupt weight discontinuities during training, while soft thresholding ensures a smooth relationship for consistent learning.

$$S(W_t, \alpha_t) = sign(W_t) \cdot ReLU(|W_t| - \alpha_t)$$
$$\alpha_t = sigmoid(\theta_{\text{init}}), \tag{2}$$

where θ_{init} is a learnable parameter that controls the initial pruning threshold α_t. We will discuss the choice of θ_{init} in the supplementary material. The $ReLU(\cdot)$ function here is used to set zero weights. In other words, if some weights $|W_t|$ are less than the threshold α_t, then the sparse version of this weight $S(w_t, \alpha_t)$ is set to 0. Otherwise, we obtain the soft-thresholding version of this weight.

The reason why we choose soft thresholding [55] rather than hard thresholding is illustrated in Figure 2. Soft parameter sharing is the best fit for our approach as it allows us to calculate the gradient and perform the backpropagation process more effectively.

AdapMTL reformulates the pruning problem in Equation 1 to find a set of optimal thresholds $\alpha = \{\alpha_B, \alpha_1, \alpha_2, ..., \alpha_T\}$ across different components as follows:

$$\min_{W, \alpha} \mathcal{L}(W, \alpha; \mathcal{D}) = \min_{W_t, \alpha_t} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \beta_t \cdot \mathcal{L}_t(f(S(W_t, \alpha_t), x^i); y_t^i)$$
$$\text{s. t. } \alpha = sigmoid(\theta_{\text{init}}), \quad W \in \mathbb{R}^d, \quad \|W\|_0 \leq (1-s) \cdot P, \tag{3}$$

where the β_t represents the adaptive weighting factor for t-th task, which will be elaborated in Section 3.3.

We next describe how AdapMTL optimizes the problem in Equation 3. Considering a multitask model with T tasks, we divide the weight parameters into $W = \{W_B, W_1, W_2, ..., W_T\}$, where W_B represents the weight parameters for the shared backbone and W_t represents the weight parameters for the t-th task-specific head. We derive the gradient descent update equation at the n-th epoch for W_t as follows:

$$W_t^{n+1} = W_t^n - \eta^n \frac{\partial \mathcal{L}(W, \alpha; \mathcal{D})}{\partial W_t^n}$$
$$= W_t^n - \eta^n \frac{\partial \mathcal{L}(W, \alpha; \mathcal{D})}{\partial S(W_t^n, \alpha_t^n)} \odot \frac{\partial S(W_t^n, \alpha_t^n)}{\partial W_t^n} \tag{4}$$
$$= W_t^n - \eta^n \frac{\partial \mathcal{L}(W, \alpha; \mathcal{D})}{\partial S(W_t^n, \alpha_t^n)} \odot \mathcal{B}_t^n,$$

where η^n is the learning rate at the n-th epoch. We use the partial derivative to calculate the gradients. As mentioned earlier, different task heads may have varying sensitivities to pruning and, consequently, may require different levels of sparsity to achieve the best accuracy. By setting a set of learnable parameters for each component and treating them separately during the backpropagation
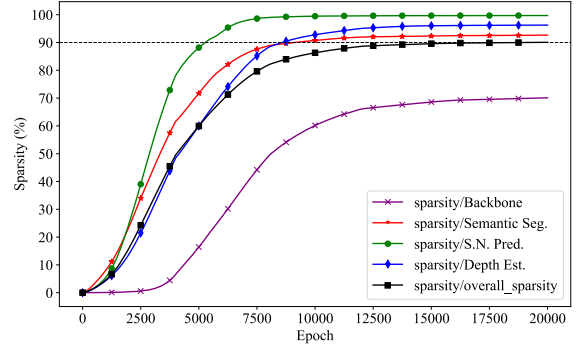

Figure 3: Breakdown of component-wise sparsity allocation during training. We use the ResNet34 backbone and achieve 90% overall sparsity in the end.

process, our component-wise pruning framework can effectively account for these differences in sensitivity and adaptively adjust the sparsity allocation for each component.

Although $\frac{\partial S(W_t^n, \alpha_t^n)}{\partial W_t^n}$ is non-differentiable, we can approximate the gradients using the sub-gradient method. In this case, we introduce $\mathcal{B}t^n$, an indicator function that acts like a binary mask. The value of $\mathcal{B}_t^n$ should be 0 if the sparse version of the weight $S(W_t^n, \alpha_t^n)$ is equal to 0. This indicator function facilitates the approximation of gradients and the update of the sparse weights and soft thresholds during the backpropagation process. Mathematically, the indicator function is:

$$\mathcal{B}_t^n = \begin{cases} 0, & \text{if } S(W_t^n, \alpha_t^n) = 0, \\ 1, & \text{otherwise.} \end{cases} \tag{5}$$

By updating the sparse weights W_t, and similarly the soft thresholds α_t, for each component in this manner (the derivation process is provided in the supplementary material), the framework can effectively and discriminatively allocate sparsity across the multitask model. By taking into account the significance and sensitivity of each component, this approach ultimately leads to more efficient and accurate multitask learning.

3.3 Adaptive Weighting Mechanism

This subsection introduces the adaptive weighting mechanism that dynamically adjusts the weight of each task loss based on each task's robustness to pruning. The adaptive weighting mechanism determines the β^t for the t-th task in Equation 3 during training.

The rationales behind the proposed adaptive weighting mechanism are two folds. First, if the training loss of a specific task t tends to be stable, then we can assign a higher weighting factor β_t and subsequently prune more aggressively on that component, as the task head is robust to pruning. On the contrary, if the loss of a specific task fluctuates significantly, we should consider pruning less on that component by lowering the weighting factor since the training is less likely to converge at higher sparsity levels. The weighting factor is learned in an adaptive way, eliminating the need for manual effort to fine-tune the hyper-parameters elaborately.

Second, the adaptive weighting mechanism should automatically consider different multitask model architectures as well. The ratio of backbone to task head weights, $\frac{W_{backbone}}{W_{head}}$, matters because it may be beneficial to focus more on pruning the task heads instead if the backbone is already highly compact. For example, in MobileNet-V2, the backbone has only 2.2M parameters, which is 25 times fewer than the task head.

We define a set of adaptive weights $\beta = \{\beta_B, \beta_1, \beta_2, ..., \beta_T\}$, where β_B represents the weighting factor for the shared backbone, β_t represents the weighting factor for the t-th task-specific head. The weighting factor can be formulated as follows:

$$\beta_t = \left(\frac{\sigma \mathcal{L}_t^{\text{window}} / \mathcal{L}_t}{\frac{1}{T} \sum_{t=1}^{T} (\sigma \mathcal{L}_t^{\text{window}} / \mathcal{L}_t)} \right)^{-1} \cdot \lambda \frac{|W_B|_{backbone}}{\sum_{t=1}^{T} |W_t|_{head}}. \quad (6)$$

Here, $\sigma \mathcal{L}_t^{\text{window}}$ is the average deviation of the loss within the sliding window for the t-th task, which is then divided by $\mathcal{L}_t$ to normalize the scale. We divide it by the sum of all tasks to normalize between different tasks. The $(\cdot)^{-1}$ is a multiplicative inverse. λ is a scaling factor, and we will discuss the choice of λ for different architectures in the supplementary material. $|W_B|_{backbone}$, $|W_t|_{head}$ represent the weight parameters of shared backbone and t-th task-specific head, separately. The right ratio in the equation reveals the importance of each component by considering their relative parameterizing contributions to the overall model structure. The weighting factor β_t is used to guide the pruning for the task-specific head, depending on the stability of its loss and its contribution to the model.

To make the multitask pruning more robust, we incorporate a sliding window mechanism that tracks the past loss values to calculate the average $\sigma \mathcal{L}_{\text{window}}$ in Equation 6 instead of relying solely on the variance between two adjacent epochs. This approach provides a more stable and reliable estimation of the fluctuations in the task losses, as it accounts for a larger number of samples and reduces the impact of potential outliers or short-term variations.

4 Experiments

4.1 Experiment Settings

4.1.1 Datasets and tasks. We conduct the experiments on two popular multi-task datasets: NYU-v2 [47], and Tiny-Taskonomy [63]. The NYU-v2 dataset is composed of RGB-D indoor scene images and covers three tasks: 13-class semantic segmentation, depth estimation, and surface normal prediction. The training set consists of 795 images, while the testing set includes 654 images. For the Tiny-Taskonomy dataset, the experiments involve joint training on five tasks: Semantic Segmentation, Surface Normal Prediction, Depth Prediction, Keypoint Detection, and Edge Detection. The training set includes 1.6 million images, while the test set comprises 0.3 million images. The training set includes 1.6 million images from 25 different classes, while the test set comprises 0.3 million images across 5 classes.

4.1.2 Evaluation Metrics and Loss Functions. We adopt a range of evaluation metrics for different tasks, evaluating the model performance at different sparsity levels to provide a comprehensive view of the model's effectiveness and robustness across tasks. On the

NYUv2 dataset, there are totally three tasks. For Semantic Segmentation, we employ the mean Intersection over Union (mIoU) and Pixel Accuracy (Pixel Acc) as our primary evaluation metrics and use cross-entropy to calculate the loss. Surface normal prediction uses the inverse of cosine similarity between the normalized prediction and ground truth, and is performed using mean and median angle distances between the prediction and the ground truth. We also report the percentage of pixels whose prediction is within the angles of 11.25°, 22.5°, and 30° to the ground truth. Depth estimation utilizes the L1 loss, with the absolute and relative errors between the prediction and ground truth being calculated. Again, We also present the relative difference between the prediction and ground truth by calculating the percentage of $\delta = max(\frac{y_{pred}}{y_{gt}}, \frac{y_{gt}}{y_{pred}})$ within the thresholds of 1.25, 1.25^2, and 1.25^3. On the Taskonomy dataset, there are two more tasks. In the context of both the Keypoint and Edge Detection tasks, the mean absolute error compared to the provided ground-truth map serves as the main evaluation metric.

In multitask learning scenarios, tasks involve multiple evaluation metrics with values potentially at different scales. To address this, we compute a single relative performance metric following the common practice [34] [50].

$$\triangle_{T_i} = \frac{1}{|M|} \sum_{j=1}^{|M|} (-1)^{l_j} \cdot (M_{T_i,j} - M_{DM,j})/M_{DM,j} * 100\% \quad (7)$$

where $l_j = 1$ if a lower value shows better performance for the metric M_j and 0 otherwise. $M_{T_i,j}, M_{DM,j}$ are the sparse and dense model value of metric j, respectively. The $\triangle_{T_i}$ is defined to compare results with their equivalent dense task values and the overall performance is obtained by averaging the relative performance across all tasks, denoted as $\triangle_T = \frac{1}{T} \sum_{i=1}^{T} \triangle_{T_i}$, This metric provides a unified measure of relative performance across tasks. Eventually, by employing these diverse evaluation metrics, we can effectively assess the performance of our method as well as the counterparts across various tasks and datasets.

4.1.3 Baselines for Comparison. We compare our work with LTH [13], IMP [19], SNIP [26], and DiSparse [52]. For LTH, we first train a dense model and subsequently prune it until the desired sparsity level is reached, yielding the winning tickets (sparse network structure). We then reset the model to its initial weights to start the sparse training process. For IMP, we iteratively remove the least important weights, determined by their magnitudes. For SNIP and IMP, we directly use the official implementation provided by the authors from GitHub. For DiSparse, the latest multitask pruning work and first-of-its-kind, we utilize the official PyTorch implementation and configure the method to use the DiSparse dynamic mechanism, which is claimed as the best-performing approach in the paper. We also train a fully dense multitask model as our baseline, which will be used to calculate a single relative performance metric Norm. Score.

We use the same backbone model at the same sparsity level across all methods for a fair comparison. In our work, we define overall sparsity as the percentage of weights pruned from the entire MTL model, which includes both the shared backbone and task-specific heads. We utilize Deeplab-ResNet34 [5] and MobileNetV2 [45] as the

Table 1: Comparison with state-of-the-art pruning methods on the NYU-V2 dataset using the Deeplab-ResNet34 backbone. Each pruning method enforces a consistent overall sparsity of 90%, with the $\triangle_T$ indicating the normalized performance of all three tasks to the baseline dense model's performance. We also report the evaluation metrics for each task and the sparsity allocation for each component.

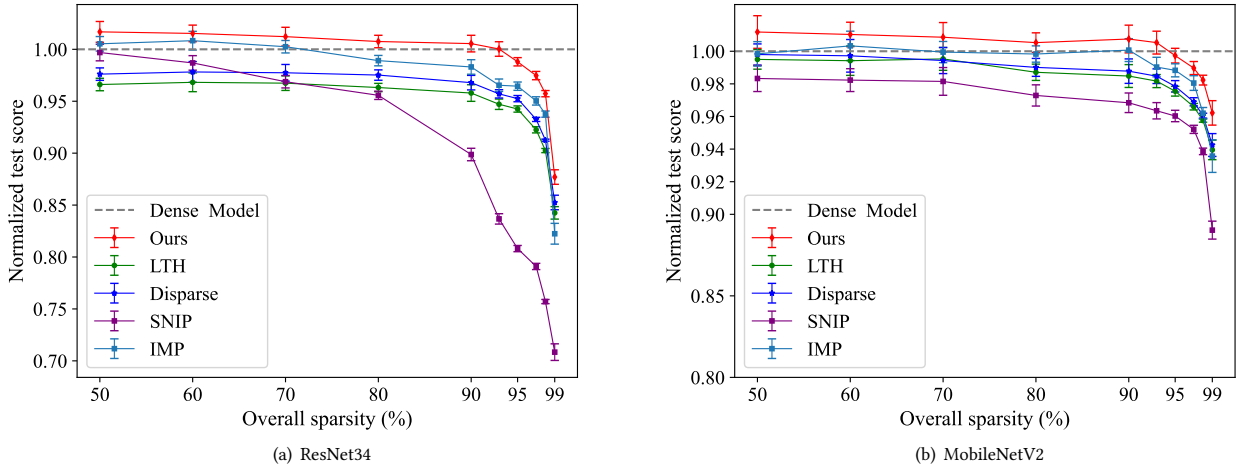| Model	T_1 : Semantic Seg.			T_2 : Surface Normal Prediction						T_3 : Depth Estimation						Sparsity %				$\triangle_T\uparrow$
	mIoU↑	pixel Acc↑	$\triangle_{T_1}\uparrow$	Error↓ Mean	Median	Angle θ,within ↑ 11.25°	22.5°	30°	$\triangle_{T_2}\uparrow$	Error↓ Abs.	Rel.	$\triangle$, within 1.25	1.25^2	1.25^3	$\triangle_{T_3}\uparrow$	Back bone	S. S. head	S.N.P. head	D. E. head	
Dense Model (baseline)	25.54	57.91	0.00	17.11	14.95	36.35	72.25	85.44	0.00	0.55	0.22	65.21	89.87	97.52	0.00		-			0.00
SNIP [26]	24.09	55.32	-10.15	16.94	14.93	36.17	72.39	86.98	2.63	0.61	0.23	60.61	87.88	96.77	-25.49	85.46	90.24	92.28	91.17	-11.00
LTH [13]	25.42	57.98	-0.35	**16.73**	15.08	35.20	72.35	**87.22**	0.41	0.57	0.22	60.93	88.64	96.20	-12.92	78.32	90.54	95.21	95.49	-4.29
IMP [19]	25.68	57.86	0.46	16.86	15.18	35.53	71.96	86.26	-1.77	0.56	0.22	65.23	89.29	97.53	-3.82	74.98	92.34	97.23	95.15	-1.71
DiSparse [52]	25.71	58.08	0.96	17.03	15.23	35.10	71.85	86.22	-4.48	0.57	0.22	64.93	88.64	97.20	-5.76	75.07	90.41	98.51	94.86	-3.10
AdapMTL w/o adaptive thresholds	25.59	57.53	-0.46	17.26	15.75	36.21	71.53	85.91	-7.06	0.58	0.22	62.52	87.12	96.50	-13.68	79.12	89.37	96.85	95.74	-7.07
AdapMTL (ours)	**26.28**	**58.29**	**3.55**	16.92	**14.91**	**36.36**	**72.97**	86.29	**3.41**	**0.55**	**0.22**	**65.39**	**89.93**	**97.58**	**0.38**	71.74	93.18	99.26	96.22	**2.45**

(a) ResNet34

(b) MobileNetV2

Figure 4: Comparison of state-of-the-art methods, including DiSparse [52], LTH [13], SNIP [26], and IMP [19], on the NYUv2 dataset, evaluated with different MTL backbones and under various sparsity settings.

backbone models, and the Atrous Spatial Pyramid Pooling (ASPP) architecture [5] as the task-specific head. Both of them are popular architectures for pixel-wise prediction tasks. We share a common backbone for all tasks while each task has an independent task-specific head branching out from the final layer of the backbone, which is widely used in multitasking scenarios.

4.2 Experiment Results

4.2.1 Results on NYU-V2. We first present the comparison results with state-of-the-art methods on the NYU-V2 dataset in table 1. Overall, AdapMTL outperforms all other methods by a significant margin across most metrics and achieves the highest $\triangle_T$. Recall that the major difference between our method and the baselines lies in our ability to adaptively learn the sparsity allocation across the components adaptively, maintaining a dense shared backbone (71.74%) while keeping the task-specific heads relatively sparse. Within the scope of our research, we characterize overall sparsity as the percentage of weights pruned from the entire MTL model, which includes both the shared backbone and task-specific heads.

SNIP [26] exhibits the lowest performance in the multi-task scenario its pruning mask is determined from a single batch of data's gradient, which treats all components, including the shared backbone, equally. Since all input information passes through the shared backbone, accuracy loss in the shallow layers is inevitable, regardless of how well the task heads perform with relatively high density. LTH's [13] winning tickets do not sufficiently focus on the backbone, as they intentionally create a dense surface normal prediction task head. Although this approach performs well on this specific task, the bias still causes an imbalance in the metrics across all tasks, resulting in a lower $\triangle_T$ score. IMP [19] achieves a good normalized score across all tasks. However, this method is trained in an iterative manner and prunes the model step-by-step, resulting in a significantly longer training time. DiSparse [52] learns an effective dense backbone by adopting a unanimous decision across all tasks. However, it falls short of differentiating the relative sensitivities between specific task heads, leading to an imbalanced normalized score among all tasks. Here, we add an additional row, AdapMTL without adaptive thresholds, to demonstrate the effectiveness of our approach. Rather than using multiple adaptive thresholds, this version utilizes a single shared threshold for all components. As

Table 2: Comparison with state-of-the-art pruning methods on the NYU-V2 dataset using the MobileNetV2 backbone. Each pruning method enforces a consistent overall sparsity of 90%, with the $\triangle_T$ indicating the normalized performance of all three tasks to the baseline dense model's performance. We also report the evaluation metrics for each task and the sparsity allocation for each component.

Model	T_1: Semantic Seg.			T_2: Surface Normal Prediction						T_3: Depth Estimation						Sparsity %				$\triangle_T\uparrow$
	mIoU↑	pixel Acc↑	$\triangle_{T_1}\uparrow$	Error↓ Mean	Median	Angle θ, within↑ 11.25°	22.5°	30°	$\triangle_{T_2}\uparrow$	Error↓ Abs.	Rel.	$\triangle$, within↑ 1.25	1.25^2	1.25^3	$\triangle_{T_3}\uparrow$	Back bone	S. S. head	S.N.P. head	D. E. head	
Dense Model [5] (baseline)	19.94	48.71	0.00	17.85	16.21	29.77	72.19	86.19	0.00	0.64	0.24	58.93	86.27	96.16	0.00	-				0.00
SNIP [26]	18.96	46.93	-8.57	18.33	16.97	28.93	71.21	85.78	-12.03	0.64	0.25	56.75	85.71	95.33	-9.38	78.46	88.19	92.08	90.25	-9.99
LTH [13]	19.14	47.25	-7.01	17.67	16.32	29.67	72.15	86.22	-0.03	0.65	0.25	57.68	85.89	96.13	-8.32	71.32	88.34	92.19	90.52	-5.12
IMP [13]	18.76	48.12	-7.13	18.71	16.68	29.63	71.76	85.91	-9.11	0.64	**0.23**	**59.75**	86.52	96.31	**6.00**	68.49	88.07	95.13	87.74	-3.41
DiSparse [52]	19.87	48.83	-0.10	17.92	16.79	29.87	71.76	85.64	-4.87	0.65	0.24	58.42	85.72	96.28	-2.94	65.22	87.21	93.55	90.53	-2.64
AdapMTL w/o adaptive thresholds	18.93	47.51	-7.53	18.16	16.87	28.37	71.53	86.63	-10.91	0.65	0.24	58.26	85.82	95.92	-3.47	73.61	88.64	92.37	89.82	-7.30
AdapMTL (ours)	**20.16**	**49.14**	**1.99**	**17.53**	**15.96**	**30.16**	**72.36**	86.51	**5.25**	0.64	0.24	59.03	**86.57**	**96.38**	0.75	52.74	86.18	94.72	90.76	**2.66**

expected, performance significantly deteriorates because a uniform threshold makes it hard to capture the nuances in different components' sensitivity.

Moreover, we extended our experiments to different model architectures to assess the model-agnostic nature of our method, using MobileNetV2 as an alternative architecture. The results, detailed in Table 2, show how AdapMTL adeptly manages the dense representation of MobileNetV2's compact backbone, ensuring it remains sufficiently dense (52.74%) while enforcing higher sparsity in the task-specific heads. This is very important, especially with such backbone compact architectures where over-pruning the backbone can easily lead to significant degradation in accuracy. Our approach ensures that the backbone remains dense enough, thereby preserving overall performance.

4.2.2 Results under various sparsity settings. We show a comparison of results under different sparsity settings using different backbones, namely ResNet34 and MobileNetV2, as illustrated in Figure 4, where AdapMTL consistently demonstrates superiority over other methods. The normalized test score, following the common practice [34] [50], is obtained by averaging the relative performance across all tasks with respect to the dense model. We observe a slightly better performance for medium sparsity levels(from 50% to 80%), which even surpasses dedicated dense multitask learning approaches despite the high sparsity enforced. This observation aligns with our assumptions and motivates the research community to further explore and develop sparse models. The score of SNIP drops significantly as higher sparsity levels (>90%) are enforced. This is because it fails to maintain the density of the shared backbone effectively.

4.2.3 Results on Tiny-Taskonomy. On the Tiny-Taskonomy dataset, which encompasses five distinct tasks, AdapMTL exhibits a more consistent performance across all tasks, as detailed in Table 3. Here, we use the ResNet backbone at sparsity 90%. Our method consistently achieved the highest scores in each task, unlike other methods which exhibited noticeable biases. The DiSparse method struggles to achieve unanimous decisions, particularly as the number of tasks increases, highlighting a key limitation in its approach.

The consistent superiority of AdapMTL across both NYUv2 and Tiny-Taskonomy datasets, and with different backbone architectures, highlights the effectiveness of our approach in achieving

Table 3: Results on Tiny-Taskonomy dataset. T1: Semantic Segmentation, T2: Surface Normal Prediction, T3: Depth Prediction, T4: Keypoint Estimation, T5: Edge Estimation.

Model	$\triangle_{T_1}\uparrow$	$\triangle_{T_2}\uparrow$	$\triangle_{T_3}\uparrow$	$\triangle_{T_4}\uparrow$	$\triangle_{T_5}\uparrow$	$\triangle_T\uparrow$
SNIP	-11.2	-15.7	-9.4	+1.2	-2.8	-7.58
LTH	-9.9	-1.3	-10.7	+0.5	+3.1	-3.66
IMP	-6.3	-9.7	+3.1	-1.1	+2.4	-2.32
DiSparse	-1.6	+1.2	-3.9	-1.5	+4.2	-0.32
AdapMTL w/o adaptive thresholds	-8.7	-12.6	-4.7	+0.2	-1.4	-5.44
AdapMTL (ours)	**+2.8**	**+4.7**	**+1.5**	**+0.5**	**+4.9**	**+2.88**

high sparsity with minimal performance degradation for multitask models. More results on the other datasets, using the different architectures, can be found in the supplementary material.
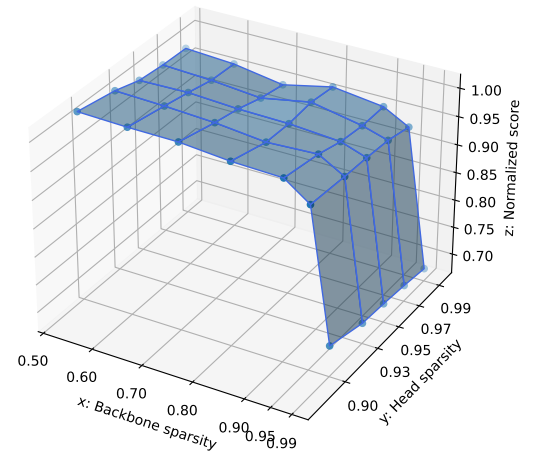

Figure 5: Visualization comparing the sensitivity of the backbone and task head in a MobileNetV2 backbone MTL model. The y-axis represents the total sparsity of all task heads.

4.3 Analysis

4.3.1 Pruning sensitivity. AdapMTL results in different sparsity for backbone parameters and task-specific parameters, indicating that it captures their different sensitivity to pruning. To compare the sensitivity to pruning between the shared backbone and task heads,

Table 4: Computational cost of AdapMTL

Method	Sparsity (%)	Params	$\triangle_T \uparrow$	FLOPs $\downarrow$
Deeplab-ResNet34	0	197.6M	-	56.32G
AdapMTL	79.83	39.52M	6.7	9.04G
AdapMTL	85.01	29.64M	4.3	7.84G
AdapMTL	90.03	19.77M	2.45	5.32G
MobileNetV2	0	155.2M	-	37.32G
AdapMTL	80.12	31.04M	7.8	5.79G
AdapMTL	85.03	23.28M	5.2	4.21G
AdapMTL	89.93	15.51M	2.66	2.98G

we create a 3D plot, as shown in Figure 5. The x-axis represents the shared backbone sparsity from 50% to 99%, while the y-axis represents the total head sparsity for all three tasks from 90% to 99%. The z-axis represents the normalized score.

From the xz-plane, we can observe that the normalized score drops significantly when we prune the backbone at sparsity levels of 90% and higher. In contrast, from the yz-plane, we can see that the task heads are highly robust to pruning, as they maintain a good normalized score even when extreme sparsity levels are reached. This observation highlights the importance of preserving the shared backbone's density and suggests that pruning strategies should prioritize maintaining the backbone's performance while aggressively pruning the task-specific heads.
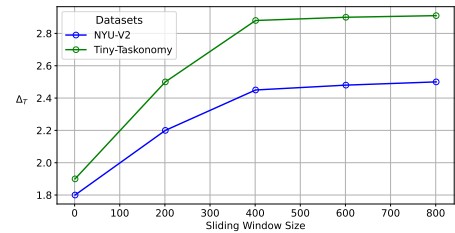
4.3.2 Computational cost. The computational cost of the AdapMTL under varying sparsity levels is detailed in Table 4, which illustrates a significant reduction in both parameters and FLOPs as sparsity increases. These reductions highlight not only the adaptability of AdapMTL across different architectures but also its capability to maintain a balance between performance, measured by $\triangle_T$, and efficiency, evidenced by the substantial decrease in FLOPs. This balance is crucial for deploying high-performance models in resource-constrained environments. By leveraging specialized hardware and software solutions that can efficiently handle sparse matrix operations, such as sparse matrix-vector multiplication (SpMV), these models can achieve faster inference times [14, 39, 56].

4.4 Ablation Studies

We conducted ablation studies to validate the effectiveness of the proposed adaptive multitask model pruning (Section 3.2), and the adaptive weighting mechanism (Equation 6). We tested variations including models without adaptive thresholds, where all components share a single threshold, and models with only two adaptive thresholds, where the backbone has a unique threshold while other task heads share another. The results, presented in Table 5, highlight the critical role of adaptive thresholding. Models without adaptive thresholds showed significantly poorer performance, with a drastic decrease in $\triangle_T$, especially affecting tasks with higher sensitivity to pruning, such as Depth Prediction. Conversely, the full AdapMTL configuration, employing independent thresholds for each component, achieved the best $\triangle_T$ score. These variations help illustrate the impact and necessity of differentiated thresholding in multitask environments. The results confirm that our full AdapMTL setup, with all components active, performs superiorly across different

Table 5: Ablation Study on NYU-V2. T1: Semantic Segmentation, T2: Surface Normal Prediction, T3: Depth Prediction.

Model	$\triangle_{T_1}\uparrow$	$\triangle_{T_2}\uparrow$	$\triangle_{T_3}\uparrow$	$\triangle_T\uparrow$
w/o λ (=5)	1.26	1.74	-1.83	0.39
w/o sliding window	3.07	2.84	-0.49	1.81
w/o adaptive thresholds	-0.46	-7.06	-13.68	-7.07
only 2 adaptive thresholds	-0.32	-3.28	-9.74	-4.45
AdapMTL	3.55	3.41	0.38	2.45

Figure 6: Choice of sliding window size

settings, underscoring the indispensable nature of each proposed component.

We have implemented a sliding window mechanism to enhance the robustness and accuracy of our pruning strategy. This mechanism is pivotal in tracking the loss values over a sequence of epochs to compute the average change in loss, $\sigma\mathcal{L}_{\text{window}}$, as formalized in Equation 6. By integrating this approach, we significantly mitigate the influence of abrupt variations and potential outliers that may occur in task-specific loss calculations. The sliding window, set at a size of 400 as demonstrated in Figure 6, represents an optimal balance between computational memory demands and the need for a comprehensive data scope. This size ensures that the model captures sufficient temporal loss information without excessive memory consumption, thereby maintaining efficiency.

5 Conclusion

In this paper, we propose a novel adaptive pruning method designed specifically for multitask learning (MTL) scenarios. Our approach effectively addresses the challenges of balancing overall sparsity and accuracy for all tasks in multitask models. AdapMTL introduces multiple learnable soft thresholds, each independently assigned to the shared backbone and task-specific heads to capture the nuances in different components' sensitivity to pruning. Our method co-optimizes the soft thresholds and model weights during training, enabling automatic determination of the ideal sparsity level for each component to achieve high task accuracy and overall sparsity. Furthermore, AdapMTL incorporates an adaptive weighting mechanism that dynamically adjusts the importance of task-specific losses based on each task's robustness to pruning. The effectiveness of AdapMTL has been extensively validated through comprehensive experiments on the NYU-v2 and Tiny-Taskonomy datasets with different architectures. The results demonstrate that our method outperforms state-of-the-art pruning methods, thereby establishing its suitability for efficient and effective multitask learning.

Acknowledgments

References

[1] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. 2006. Multi-task feature learning. *Advances in neural information processing systems* 19 (2006).

[2] Tadas Baltrušaitis, Chaitanya Ahuja, and Louis-Philippe Morency. 2018. Multi-modal machine learning: A survey and taxonomy. *IEEE transactions on pattern analysis and machine intelligence* 41, 2 (2018), 423–443.

[3] Rich Caruana. 1993. Multitask learning: A knowledge-based source of inductive bias. In *Machine Learning Proceedings 1993*. Elsevier, 41–48.

[4] Rich Caruana. 1997. Multitask learning. *Machine learning* 28 (1997), 41–75.

[5] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. 2017. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence* 40, 4 (2017), 834–848.

[6] Xinghao Chen, Yiman Zhang, and Yunhe Wang. 2022. MTP: multi-task pruning for efficient semantic segmentation networks. In *2022 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 1–6.

[7] Hanjing Cheng, Zidong Wang, Lifeng Ma, Xiaohui Liu, and Zhihui Wei. 2021. Multi-task pruning via filter index sharing: A many-objective optimization approach. *Cognitive Computation* 13 (2021), 1070–1084.

[8] Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*. ACM, 160–167.

[9] Xiaohan Dong, Huizi Mao, Tianchen Liu, Yiming Yang, Ji Huang, Sen Chen, Zhang Yang, Geng Yuan Tong, Zhen Lin, Song Tang, et al. 2021. HAWQ: Hessian AWare Quantization of Neural Networks with Mixed-Precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 9234–9243.

[10] David L Donoho. 1995. De-noising by soft-thresholding. *IEEE transactions on information theory* 41, 3 (1995), 613–627.

[11] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, Erich Elsen, Jakob Uszkoreit, and Avital Dubey. 2020. Rigging the lottery: Making all tickets winners. In *International Conference on Learning Representations*.

[12] Theodoros Evgeniou and Massimiliano Pontil. 2004. Regularized multi–task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. 109–117.

[13] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. 2020. The lottery ticket hypothesis at scale. *International Conference on Learning Representations* (2020).

[14] Trevor Gale, Erich Elsen, and Sara Hooker. 2019. The State of Sparsity in Deep Neural Networks. In *arXiv preprint arXiv:1902.09574*.

[15] Yuan Gao, Haoping Bai, Zequn Jie, Jiayi Ma, Kui Jia, and Wei Liu. 2020. Mtl-nas: Task-agnostic neural architecture search towards general-purpose multi-task learning. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*. 11543–11552.

[16] Siddhant Garg, Lijun Zhang, and Hui Guan. 2023. Structured Pruning for Multi-Task Deep Neural Networks. *arXiv preprint arXiv:2304.06840* (2023).

[17] Ross Girshick. 2015. Fast R-CNN. In *Proceedings of the IEEE international conference on computer vision*. 1440–1448.

[18] Hui Guan, Xipeng Shen, and Hamid Krim. 2017. Egeria: A framework for automatic synthesis of HPC advising tools through multi-layered natural language processing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14.

[19] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[20] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*. 1135–1143.

[21] Babak Hassibi and David Stork. 1992. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems* 5 (1992).

[22] Laurent Jacob, Francis R Bach, and Jean-Philippe Vert. 2009. Clustered multi-task learning: A convex formulation. In *Advances in neural information processing systems*. 745–752.

[23] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. 2020. Soft threshold weight reparameterization for learnable sparsity. In *International Conference on Machine Learning*. PMLR, 5544–5555.

[24] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[25] Yann LeCun, John S Denker, and Sara A Solla. 1990. Optimal brain damage. *Advances in neural information processing systems* (1990), 598–605.

[26] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. 2018. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340* (2018).

[27] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710* (2016).

[28] Shaohui Lin, Rongrong Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and Baochang Zhang. 2020. HRank: Filter Pruning using High-Rank Feature Map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1529–1538.

[29] Xiaodong Liu, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh, and Ye-Yi Wang. 2015. Representation learning using multi-task deep neural networks for semantic classification and information retrieval. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*. 912–921.

[30] Yinglu Liu, Mingcan Xiang, Hailin Shi, and Tao Mei. 2021. One-stage Context and Identity Hallucination Network. In *Proceedings of the 29th ACM International Conference on Multimedia*. 835–843.

[31] Zehao Liu, Haoliang Li, Shuicheng Shen, Junjie Yan, Xiaolin Zhang, and Nenghai Wang. 2019. MetaPruning: Meta Learning for Automatic Neural Network Channel Pruning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 3296–3305.

[32] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the value of network pruning. In *Proceedings of the IEEE International Conference on Computer Vision*. 7002–7012.

[33] Jian-Hao Luo and Jianxin Wu. 2020. Neural network pruning with residual-connections and limited-data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1458–1467.

[34] Kevis-Kokitsi Maninis, Ilija Radosavovic, and Iasonas Kokkinos. 2019. Attentive single-tasking of multiple tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1851–1860.

[35] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. 2018. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. In *Proceedings of the 35th International Conference on Machine Learning*. 1125–1143.

[36] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. 2017. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*. PMLR, 2498–2507.

[37] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. Importance estimation for neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 11264–11272.

[38] Ari S Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. 2019. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In *Advances in Neural Information Processing Systems*. 11644–11655.

[39] Hesham Mostafa, Xiaoxiao Wang, and Decebal Constantin Mocanu. 2019. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *Proceedings of the 36th International Conference on Machine Learning*. 4648–4657.

[40] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. 2011. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*. 689–696.

[41] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. 2020. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 907–922.

[42] Emilio Parisotto, Jimmy Lei Ba, and Antoine Bordes. 2016. Actor-mimic: Deep multitask and transfer reinforcement learning. In *International Conference on Learning Representations*.

[43] Dripta S Raychaudhuri, Yumin Suh, Samuel Schulter, Xiang Yu, Masoud Faraki, Amit K Roy-Chowdhury, and Manmohan Chandraker. 2022. Controllable dynamic multi-task architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10955–10964.

[44] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*. 91–99.

[45] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.

[46] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks* 61 (2015), 85–117.

[47] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. 2012. Indoor segmentation and support inference from rgbd images. *ECCV (5)* 7576 (2012), 746–760.

[48] Jingtong Su, Yihang Chen, Tianle Cai, Tianhao Wu, Ruiqi Gao, Liwei Wang, and Jason D Lee. 2020. Sanity-checking pruning methods: Random tickets can win the jackpot. *Advances in Neural Information Processing Systems* 33 (2020), 20390–20401.

[49] Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid. 2019. Videobert: A joint model for video and language representation learning. In *Proceedings of the IEEE/CVF international conference on computer vision*. 7464–7473.

[50] Ruoyu Sun, Dawei Li, Shiyu Liang, Tian Ding, and Rayadurgam Srikant. 2020. The global landscape of neural networks: An overview. *IEEE Signal Processing Magazine* 37, 5 (2020), 95–108.

[51] Tianxiang Sun, Yunfan Shao, Xiaonan Li, Pengfei Liu, Hang Yan, Xipeng Qiu, and Xuanjing Huang. 2020. Learning sparse sharing architectures for multiple tasks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 8936–8943.

[52] Xinglong Sun, Ali Hassani, Zhangyang Wang, Gao Huang, and Humphrey Shi. 2022. DiSparse: Disentangled Sparsification for Multitask Model Compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12382–12392.

[53] Ximeng Sun, Rameswar Panda, Rogerio Feris, and Kate Saenko. 2020. Adashare: Learning what to share for efficient deep multi-task learning. *Advances in Neural Information Processing Systems* 33 (2020), 8728–8740.

[54] Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. 2017. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*. 4496–4506.

[55] Antoine Vanderschueren and Christophe De Vleeschouwer. 2023. Are Straight-Through gradients and Soft-Thresholding all you need for Sparse Training?. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 3808–3817.

[56] Yulong Wang, Yifan Xu, Siyuan Qiao, Hanxiao Liu, Zhijian Yang, Chao Xu, Daiyi Lin, Tong Wang, Xinyu Dai, Yichen Huang, et al. 2020. EagleEye: Fast Sub-net Evaluation for Efficient Neural Network Pruning. In *Proceedings of the 37th International Conference on Machine Learning*. 10016–10026.

[57] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*. 2074–2082.

[58] Mingcan Xiang, Yinglu Liu, Tingting Liao, Xiangyu Zhu, Can Yang, Wu Liu, and Hailin Shi. 2021. The 3rd grand challenge of lightweight 106-point facial landmark localization on masked faces. In *2021 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*. IEEE, 1–6.

[59] Yu Yang and Timothy M Hospedales. 2016. Trace norm regularized deep multi-task learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4333–4341.

[60] Hancheng Ye, Bo Zhang, Tao Chen, Jiayuan Fan, and Bin Wang. 2023. Performance-aware Approximation of Global Channel Pruning for Multitask CNNs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023).

[61] Ruichi Yu, Ang Li, Chun-Fu Chen, Jiwen Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. 2018. NISP: Pruning Networks using Neuron Importance Score Propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 9194–9203.

[62] Yujia Yu, Shuaishuai Liu, Anfeng Zhang, and Chunhua Shen. 2019. Playing Lottery Tickets with Vision and Language. In *arXiv preprint arXiv:1912.04488*.

[63] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. 2018. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3712–3722.

[64] Lijun Zhang, Xiao Liu, and Hui Guan. 2022. Automtl: A programming framework for automating efficient multi-task learning. *Advances in Neural Information Processing Systems* 35 (2022), 34216–34228.

[65] Lijun Zhang, Xiao Liu, and Hui Guan. 2022. A Tree-Structured Multi-Task Model Recommender. In *International Conference on Automated Machine Learning*. PMLR, 10–1.

[66] Lijun Zhang, Qizheng Yang, Xiao Liu, and Hui Guan. 2023. An Alternative Hard-Parameter Sharing Paradigm for Multi-Domain Learning. *IEEE Access* 11 (2023), 10440–10452.

[67] Shaokai Zhang, Shanhe Du, Wentai Wang, Yiran Chen, and Hai Li. 2018. A systematic DNN weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 184–199.

[68] Yu Zhang and Qiang Yang. 2018. A Survey on Multi-Task Learning. *arXiv preprint arXiv:1707.08114* (2018).

A Thresholds updating

We detail the process of updating thresholds within the AdapMTL framework in this section. The thresholds are determined by θ_{init}, such that $\alpha_t = \text{sigmoid}(\theta_{\text{init}})$. Consequently, the challenge of updating the thresholds is transformed into the task of updating the θ_t for each specific task. Considering a multitask model with T tasks, we divide the weight parameters W into $W = \{W_B, W_1, W_2, ..., W_T\}$, where W_B represents the weight parameters for the shared backbone and W_t represents the weight parameters for the t-th task-specific head. We derive the gradient descent update equation at the n-th epoch for θ_t as follows:

$$
\begin{aligned}
\theta_t^{n+1} &= \theta_t^n - \eta^n \frac{\partial \mathcal{L}(W,\alpha;\mathcal{D})}{\partial \theta_t^n} \\
&= \theta_t^n - \eta^n \frac{\partial \mathcal{L}(\theta,\alpha;\mathcal{D})}{\partial W_t^n} \odot \frac{\partial W_t^n}{\partial \theta_t^n} \\
&= \theta_t^n - \eta^n \cdot (-sigmoid(\theta_t^n))' \cdot \frac{\partial \mathcal{L}(\theta,\alpha;\mathcal{D})}{\partial W_t^n} \\
&= \theta_t^n - \eta^n \cdot (-sigmoid(\theta_t^n))' \cdot \frac{\partial \mathcal{L}(W,\alpha;\mathcal{D})}{\partial S(W_t^n,\alpha_t^n)} \odot \frac{\partial S(W_t^n,\alpha_t^n)}{\partial W_t^n} \\
&= W_t^n + \eta^n \cdot (sigmoid(\theta_t^n))' \cdot \frac{\partial \mathcal{L}(W,\alpha;\mathcal{D})}{\partial S(W_t^n,\alpha_t^n)} \odot \mathcal{B}_t^n,
\end{aligned}
\tag{8}
$$

where η^n is the learning rate at the n-th epoch. We use the partial derivative to calculate the gradients. Although $\frac{\partial S(W_t^n,\alpha_t^n)}{\partial W_t^n}$ is non-differentiable, we can approximate the gradients using the sub-gradient method. In this case, we introduce $\mathcal{B}t^n$, an indicator function that acts like a binary mask. The value of $\mathcal{B}t^n$ should be 0 if the sparse version of the weight $S(W_t^n,\alpha_t^n)$ is equal to 0. This indicator function facilitates the approximation of gradients and the update of the sparse weights and soft thresholds during the backpropagation process. Mathematically, the indicator function is:

$$
\mathcal{B}_t^n = \begin{cases} 0, & \text{if } S(W_t^n,\alpha_t^n) = 0, \\ 1, & \text{otherwise.} \end{cases}
\tag{9}
$$

B Training Details

We adopt the same training configurations as those used in DiSparse [52], which is the latest multitask pruning work, for fair comparisons. We conduct all our experiments using PyTorch and RTX 8000 GPUs, and we employ the Adam optimizer with a batch size of 16. For the NYUV2 dataset, we run 20K iterations with an initial learning rate of 1e-3, decaying by 0.5 every 4,000 iterations. For the Tiny-Taskonomy dataset, we train for 100K iterations with an initial learning rate of 1e-4, decaying by 0.3 every 12K iterations. The size of the sliding window in our experiments is set to 400 to smooth loss deviations. We utilized cross-entropy loss for Semantic Segmentation, negative cosine similarity between the normalized prediction and ground truth for Surface Normal Prediction, and L1 loss for the remaining tasks. To avoid bias and diversity in different pre-trained models, we trained all models from scratch, ensuring a fair comparison among various methods. It's noteworthy that, unlike many previous works, our method does not require any pre-training or pre-pruned models.

Table 6: Hyper-parameters for training on NYUv2 and Tiny-taskonomy datasets

Dataset	lr	lr decay	epoch
NYUv2	0.001	0.5/ 4,000 ters	20,000
Tiny-Taskonomy	0.0001	0.3/ 10,000 iters	50,000

We use the θ_{init} parameter, set to -20, to regulate the duration of dense training phases. A lower θ_{init} value extends the period dedicated to dense representation, allowing for more comprehensive learning before pruning begins.

C Additional Results

We provide additional results on the Tiny-Taskonomy dataset using Resnet34 and MobileNetV2 architecture, separately. On the Tiny-Taskonomy dataset, which comprises a total of 5 tasks, AdapMTL demonstrates a more balanced performance across tasks. As shown in Table 7, 8, our method achieved the highest $\triangle_T$ score and the lowest absolute error for most tasks.

C.1 Pruning Sensitivity Analysis

Different task heads may have similar amounts of model weights, but their sensitivities to pruning can vary. This observation suggests that a more discriminative pruning approach should be employed for each task, taking into account their unique sensitivities. To verify this observation, we fixed a ResNet34 backbone at a sparsity level of 95% for better visualization and pruned three task heads independently to examine their sensitivity to pruning.

As shown in Figure 7, the head of the surface normal prediction task is the least sensitive, as it maintains good accuracy even when extreme sparsity is enforced. Therefore, AdapMTL learns to keep this task head at a high level of sparsity during pruning, which is aligned with the component-wise sparsity allocation in the table of the manuscript. In contrast, the head of the semantic segmentation task is relatively more sensitive to pruning, so we strive to keep it as dense as possible throughout the training process. This tailored approach to pruning helps AdapMTL achieve better overall performance across different tasks by considering the specific pruning sensitivity of each task head.

C.2 Adaptive weighting factor

The adaptive weighting mechanism is used to decide the head sparsity allocation among different tasks based on their varying sensitivity to pruning. By adaptively learning a weighting factor, we can assign different importance to each task, subsequently pruning discriminatively on different task heads.
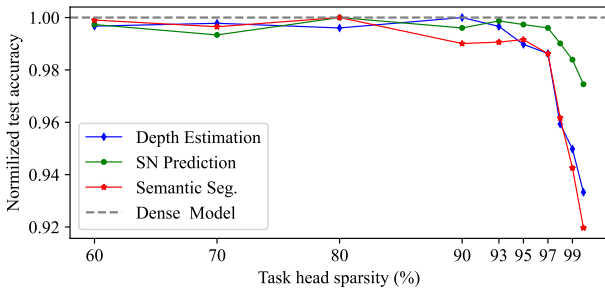
As shown in Figure 8, we initially set the weighting factor of each task equal, such as 1, to ensure sufficient training of each task for 5,000 epochs. Over time, the surface normal prediction task tends to stabilize and converge, leading to small loss fluctuations. This implies that we can prune more aggressively on that component, as the task head is robust to pruning. Consequently, the weighting factor of this task will be larger than the others. This observation is aligned with the results from Table in the main text, where the surface normal prediction achieves higher sparsity compared to other

Table 7: Comparison with MTL pruning methods on the Tiny-Taskonomy dataset using the Deeplab-ResNet34 backbone.

Model	T_1 : Semantic Seg.		T_2 : Normal Pred.		T_3 : Depth Estimation		T_4 : Keypoint Det.		T_5 : Edge Det.		sparsity	$\triangle_T\uparrow$
	Abs. ↓	$\triangle_{T_1}\uparrow$	Abs. ↑	$\triangle_{T_2}\uparrow$	Abs. ↓	$\triangle_{T_2}\uparrow$	Abs. ↓	$\triangle_{T_2}\uparrow$	Abs. ↓	$\triangle_{T_2}\uparrow$	%	
Dense Model	0.5053	0	0.8436	0.00	0.0222	0.00	0.1961	0.00	0.2131	0.00	95	0.00
SNIP [26]	0.5659	-11.99	0.7301	-13.45	0.0246	-10.81	0.1972	-0.56	0.2221	-4.22	95	-8.21
LTH [13]	0.5345	-5.78	0.8189	-3.38	0.0234	-5.41	0.2004	-2.19	0.2187	-2.63	95	-3.88
IMP [19]	0.5163	-2.18	0.8371	-0.79	0.0221	0.45	0.1962	-0.05	0.2184	-2.49	95	-1.01
DiSparse [52]	0.5287	-4.63	0.8423	-0.16	0.0217	2.25	0.1987	-1.33	0.2089	1.97	95	-0.38
AdapMTL w/o adaptive thresholds	0.5468	-8.21	0.8059	-4.48	0.02296	-3.42	0.1937	1.22	0.2153	-1.03	95	-3.18
AdapMTL	0.5038	0.30	0.8513	0.96	0.0221	0.45	0.1923	1.94	0.2074	2.67	95	1.26

Table 8: Comparison with MTL pruning methods on the Tiny-Taskonomy dataset using the MobileNetV2 backbone.
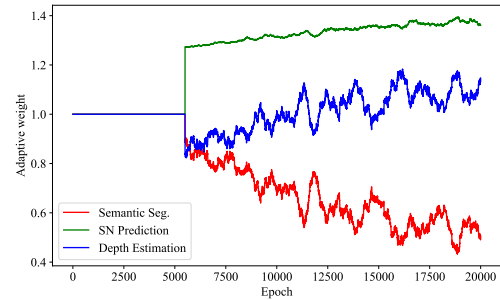
Model	T_1 : Semantic Seg.		T_2 : Normal Pred.		T_3 : Depth Estimation		T_4 : Keypoint Det.		T_5 : Edge Det.		sparsity	$\triangle_T\uparrow$
	Abs. ↓	$\triangle_{T_1}\uparrow$	Abs. ↑	$\triangle_{T_2}\uparrow$	Abs. ↓	$\triangle_{T_2}\uparrow$	Abs. ↓	$\triangle_{T_2}\uparrow$	Abs. ↓	$\triangle_{T_2}\uparrow$	%	
Dense Model	1.0783	0.00	0.7429	0.00	0.0318	0.00	0.203	0.00	0.2242	0.00	95	0.00
SNIP [26]	1.0901	-1.09	0.7243	-2.50	0.0321	-0.94	0.2157	-6.26	0.2364	-5.44	95	-3.25
LTH [13]	1.0869	-0.80	0.7407	-0.30	0.0325	-2.20	0.2118	-4.33	0.2328	-3.84	95	-2.29
IMP [19]	1.0795	-0.11	0.7415	-0.19	0.0327	-2.83	0.2012	0.89	0.2351	-4.86	95	-1.42
DiSparse [52]	1.0781	0.02	0.7423	-0.08	0.0322	-1.26	0.208	-2.46	0.2287	-2.01	95	-1.16
AdapMTL w/o adaptive thresholds	1.0868	-0.79	0.7329	-1.35	0.0344	-8.18	0.2043	-0.64	0.2233	0.40	95	-2.11
AdapMTL	1.0751	0.30	0.7421	-0.11	0.0305	4.09	0.2021	0.44	0.2225	0.76	95	1.10

Figure 7: Comparison of task head sensitivities to pruning for different vision tasks. We use a 95% sparse Resnet34 backbone for better depiction. The dashed line dense model indicates the task head is dense.

Figure 8: The evolution of adaptive weighting factors during training. Equal weights are initially assigned to each task for the first 5000 epochs to ensure sufficient training.

task heads. In contrast, the loss of semantic segmentation fluctuates significantly, indicating that we should consider pruning less on that component by lowering the weighting factor, as the training is less likely to converge at higher sparsity levels. It is worth mentioning that the weighting factor is learned adaptively, eliminating the need for manual effort to fine-tune the hyper-parameters.

To validate the effectiveness of our adaptive weighting mechanism, we also carry out an experiment where we assign equal weights (β^t=1) to each task and then visualize the sparsity allocation across each component under this configuration. For comparison, we also visualize the sparsity allocation with adaptive weighting, where the weighting factor for semantic segmentation, surface normal prediction, and depth estimation is set to 1.35, 1.15, and 0.5 respectively. This configuration is from the one shown in Figure 8.

As illustrated in Figure 9, the adaptive weighting factor results in a denser shared backbone compared to the equal weighting factor,

while simultaneously making the task heads sparser. Notably, even though the task-specific head of semantic segmentation is assigned a lower value, it achieves higher sparsity than with equal weight. This phenomenon arises because the shared backbone is already dense, prompting a sparser task head than before. The right-most sub-figure presents the overall performance under this sparsity allocation. It is evident that our method, with the incorporation of the adaptive weighting mechanism, outperforms the variant of our method without it.

Through this experiment, we demonstrate that the adaptive weighting mechanism plays a crucial role in maintaining high density for the shared backbone and efficiently allocating sparsity among the task-specific heads. By taking into account the sensitivity and importance of different components in the MTL model, the adaptive weighting mechanism allows for better overall performance even when high sparsity is enforced.
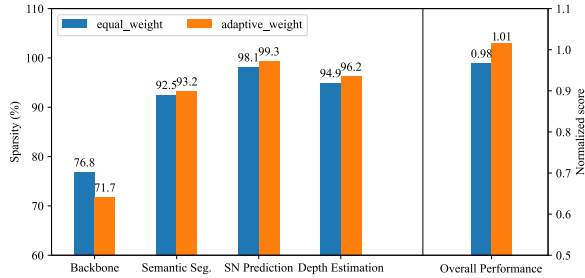
Figure 9: Comparison of sparsity allocation between equal weight (1) and adaptive weight configurations, where we assign weighting factors of 1.35, 1.15, and 0.5 to each task-specific head respectively. The overall sparsity is 90%

Table 9: Video captioning performance on YouCook II using VideoBERT. Each pruning method enforces a sparsity of 80%.

Model	BLEU-3 ↑	BLEU-4 ↑	METEOR ↑	ROUGE-L ↑	CIDEr ↑	$\triangle_T$ ↑
Dense Model	6.74	4.03	10.69	27.35	0.49	0.00
SNIP	6.38	3.72	10.42	26.85	0.47	-4.35
Disparse	6.70	4.04	10.65	27.17	0.49	-0.37
AdapMTL	6.77	4.12	10.64	27.24	0.49	0.45

C.3 Multimodal and multimedia scenarios

While our work primarily focuses on unimodal tasks in computer vision for a fair comparison with the SOTA methods, the AdapMTL framework is versatile and can be applied to multimodal and multimedia scenarios as well.

To demonstrate this, we have conducted new experiments on the video captioning task using VideoBERT model [49]. VideoBERT is a variant of the BERT model (transformer architecture) that targets text-to-video generation. We treat the multi-head attention layers and feed-forward layers as independent components and assign corresponding adaptive soft thresholds α to them. The soft thresholds are learned adaptively and eventually stop at the desired sparsity level. Table 9 shows the accuracy comparison of the pruned model on sparsity 80% for all baselines. AdapMTL outperforms its counterparts while maintaining performance metrics comparable to the dense VideoBERT model. We expect that the proposed methods can scale to larger models, including LLMs. Since LLMs are composed of multiple transformer blocks, our component-wise pruning framework is naturally well-suited for this architecture.

D Discussion

One of the noticeable aspects of AdapMTL is that the final sparsity of our model may not exactly match the requested sparsity. This discrepancy arises due to the intrinsic behavior of the soft thresholds. During pruning, these soft thresholds determine whether a specific parameter should be set to zero, thereby introducing sparsity into the model. However, the soft thresholds do not strictly enforce the exact level of sparsity but rather guide the model to approach the desired sparsity level. This level of flexibility is a design choice made to prevent any undue negative impact on model performance due to overly rigid sparsity constraints. It allows the model to strike a

balance between the targeted sparsity and the necessity to preserve adequate performance levels. Our current strategy to maximally approximate the desired sparsity level involves fixing the pruning mask once the desired sparsity has been reached. Another potential approach could involve a recovery mechanism that regrows some crucial parameters that were pruned in earlier epochs. Future research could explore more precise control mechanisms over the final sparsity while ensuring that the model's performance remains robust.