Parameter-Efficient Multi-Task Learning via Progressive Task-Specific Adaptation

Neeraj Gangwar[†] **Anshuka Rangi**[§] **Rishabh Deshmukh**[§] **Holakou Rahmanian**[§]
Yesh Dattatreya[§] **Nickvash Kani**[†]

[†]University of Illinois Urbana-Champaign [§]Amazon
gangwar2@illinois.edu, anshuka@amazon.com, derishab@amazon.com

Abstract

Parameter-efficient fine-tuning methods have emerged as a promising solution for adapting pre-trained models to various downstream tasks. While these methods perform well in single-task learning, extending them to multi-task learning exacerbates common issues, such as task interference and negative transfer, due to the limited number of trainable parameters. To address these challenges, we introduce progressive task-specific multi-task adaptation, a novel parameter-efficient approach for multi-task learning. Our approach introduces adapter modules that are shared in early layers and become increasingly task-specific in later layers. Additionally, we propose a gradient-based approach for computing task similarity and use this measure to allocate similar tasks to the shared adapter modules. To evaluate our approach, we adapt Swin and Pyramid Vision Transformers on PASCAL and NYUD-v2. On both datasets, our approach outperforms prior parameter-efficient multi-task methods while using fewer trainable parameters.

1 INTRODUCTION

Large language and vision models pre-trained on extensive datasets have demonstrated an unprecedented ability to understand and generate human-like text and images, perform complex reasoning, and analyze data (e.g., Achiam et al., 2023; Team et al., 2023; Gao et al., 2023; Touvron et al., 2023; Bai et al., 2024). These models can learn from a few demonstrations through in-context learning or can be adapted to various downstream tasks through targeted fine-tuning. Although fine-tuning may result in better performance compared to in-context learning, updating all model parameters demands substantial computational resources. Moreover, a separate copy of the model must be stored for each task, and this challenge only worsens as model sizes increase.

Several parameter-efficient methods have been proposed to address these challenges (e.g., Hu et al., 2022; Li and Liang, 2021; Lester et al., 2021; Houlsby et al., 2019; Sung et al., 2021; Zhang et al., 2023; Liu et al., 2024). These approaches either fine-tune a subset of the model's existing parameters or add new layers, training only these layers. In both cases, the number of trainable parameters remains small relative to the overall model size. These methods have achieved performance levels that nearly match those of full fine-tuning, offering a more favorable trade-off between trainable parameter count and downstream task performance. They also reduce storage requirements, as only the updated parameters must be stored separately for each task. These methods have been widely used for adapting large pre-trained models to various natural language processing (NLP) and vision tasks (e.g., Hu et al., 2023; Chen et al., 2022; Jia et al., 2022; Chen et al., 2023; Xin et al., 2024). While they have proven highly effective for adapting single-task models, their applicability to multi-task learning (MTL) remains limited. One reason is the unique challenges associated with MTL. Due to parameter sharing across tasks, MTL can suffer from conflicting and dominating gradients. This results in interference and negative transfer between tasks, leading to significantly degraded performance (Yu et al., 2020). These problems become more pronounced in parameter-efficient multi-task learning (PEMTL), as the limited number of trainable parameters restricts the model's capacity to effectively separate task-specific

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300. Copyright 2026 by the author(s).

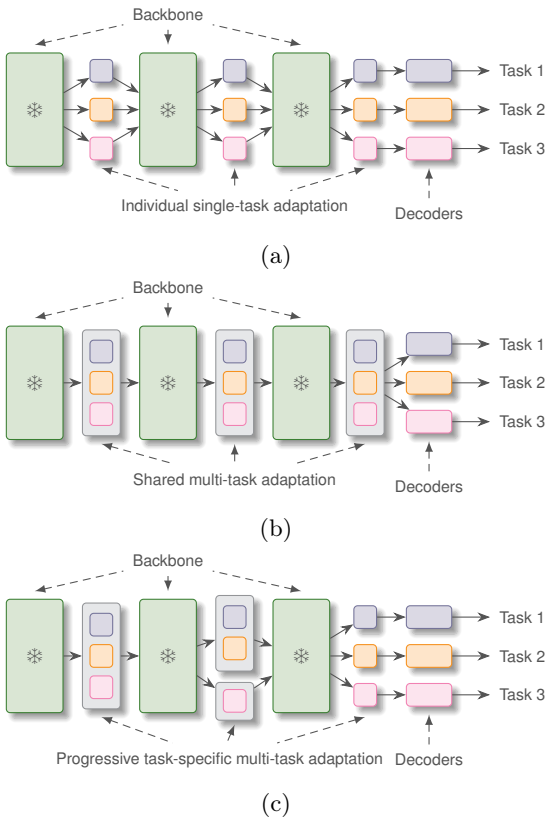


Figure 1: (a) Single-task adaptation uses separate adapters per task, preventing knowledge transfer and increasing inference cost with more tasks. (b) Shared multi-task adaptation uses common adapters, reducing inference cost at the expense of potential task interference. (c) *Progressive task-specific multi-task adaptation*, where adapters become increasingly task specific near the decoders, enables both knowledge transfer and task specialization.

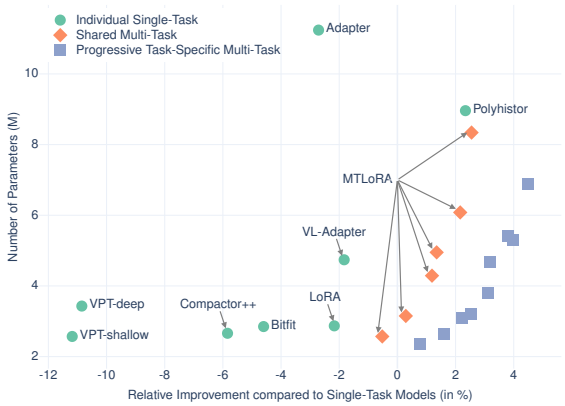


Figure 2: Comparison of our proposed approach, progressive task-specific multi-task adaptation, with methods that use individual single-task and shared multi-task adaptations. These experiments are performed on the PASCAL dataset with Swin-Tiny. The numbers for individual single-task and shared multi-task adaptations are taken from Agiza et al. (2024).

knowledge. Although PEMTL is relatively underexplored, several works have made notable advancements (e.g., Pfeiffer et al., 2021; Wang et al., 2022; Mahabadi et al., 2021; Liu et al., 2022; Agiza et al., 2024; Baek et al., 2025). Existing methods for adapting pre-trained models to perform multiple tasks fall into two main categories: (1) individual single-task adaptation and (2) shared multi-task adaptation.

In individual single-task adaptation (Figure 1a), separate adapter modules[1] are added for each downstream task, making it particularly effective when tasks are not related or have specific requirements. However, there are two major drawbacks. First, this method does not allow knowledge transfer among tasks, which may lead to suboptimal performance compared to models with shared parameters. Second, the computational cost of inference increases linearly with the number of tasks, as the adapter modules for each task must be executed individually. In shared multi-task adaptation (Figure 1b), the adapter modules are shared across all tasks. This setup enables knowledge transfer among tasks and reduces inference cost compared to individual single-task adaptation. However, due to gradient conflicts, shared adapter modules can suffer from task interference and negative transfer, limiting task-specific learning.

To address these issues, we introduce *progressive task-specific multi-task adaptation*, a parameter-efficient approach, illustrated in Figure 1c, that balances the trade-off between individual single-task and shared multi-task adaptations. In our approach, adapter modules are shared across all tasks in the initial layers and become increasingly task-specific toward the decoders. Structuring adapter modules in this manner effectively forms a branched multi-task network. Branched networks have been effective in full fine-tuning settings (Lu et al., 2016; Vandenhende et al., 2019; Brüggemann et al., 2020; Guo et al., 2020); however, branching significantly increases trainable parameters and storage requirements. Adapter modules offer a more efficient way of implementing branching while maintaining strong multi-task performance with substantially fewer trainable parameters. Our approach achieves an inference cost that falls between those of individual single-task and shared multi-task adaptations. To reduce conflicts among tasks, we assign *similar* tasks to the shared adapter modules. We use a gradient-based method to compute task similarity, which introduces

[1] We use the term "adapter module" to refer to the parameters added to a model to adapt it for a downstream task.

minimal computational overhead. We hypothesize that a progressive task-specific architecture could better handle task conflicts and improve performance with fewer trainable parameters. We compare our approach with existing PEMTL methods on the PASCAL and NYUD-v2 datasets. Note that our work complements methods that use gradient surgery to reduce gradient conflicts among tasks and can therefore be combined with any such method. In our evaluation, however, we assess our approach without incorporating such methods and employ standard fine-tuning.

Contributions. Our contributions are as follows:

1. We introduce progressive task-specific adaptation, a parameter-efficient method for multi-task learning that bridges the gap between individual single-task and shared multi-task adaptations. It facilitates knowledge transfer among tasks and reduces interference through intelligent task grouping.

2. We propose a gradient-based task similarity approach to assign similar tasks to the shared adapter modules. Our experiments show that grouping less similar tasks, identified by this approach, adversely impacts the multi-task model's performance, proving its effectiveness.

3. We create a LoRA-based (Hu et al., 2022) layer named TGLoRA, which is used to implement the progressive task-specific architecture.

4. Our approach achieves a better relative improvement over single-task fine-tuning and uses fewer trainable parameters than the current state-of-the-art PEMTL approaches. Figure 2 shows a comparison of our approach with existing approaches on the PASCAL dataset.

Our code is publicly available on GitHub.[2]

2 RELATED WORKS

Parameter-Efficient Fine-Tuning. To achieve a better trade-off between the number of trainable parameters and performance on downstream tasks, various methods have been proposed. These approaches fine-tune a small set of parameters while the rest of the model is frozen. One set of approaches adds new layers to the model and fine-tunes only these. For example, Houlsby et al. (2019) and Pfeiffer et al. (2020) place Adapter layers after the attention and MLP layers and fine-tune these newly added layers to achieve a comparable performance to full fine-tuning on NLP datasets. Hu et al. (2022) propose LoRA, a method to efficiently fine-tune large language models by adding low-rank decomposition matrices to their weights, significantly reducing the number of trainable parameters. Several subsequent works have proposed variations of LoRA, for example, AdaLoRA (Zhang et al., 2023), LoRA Dropout (Lin et al., 2024a), DoRA (Liu et al., 2024), and LoRA+ (Hayou et al., 2024), among others. Another line of research adds trainable prompts to the input and fine-tunes only these while keeping the model frozen (Li and Liang, 2021; Lester et al., 2021). Methods like BitFit (Zaken et al., 2022) and FISH (Sung et al., 2021) fine-tune a small subset of the model's parameters without adding any new parameters. While parameter-efficient methods have been widely applied to NLP tasks, several works have explored their application for vision tasks. Jia et al. (2022) propose visual prompt tuning for adapting Vision Transformers. Gao et al. (2023) propose Llama-Adapter V2, which leverages both prefix-tuning and adapter techniques.

Multi-Task Learning. MTL in computer vision aims to train a model to perform multiple related tasks simultaneously, such as object detection, segmentation, depth estimation, etc. By sharing representations across tasks, MTL allows models to leverage shared information, potentially improving performance and efficiency compared to training separate models for each task (Crawshaw, 2020; Vandenhende et al., 2021; Yu et al., 2024). However, MTL faces common challenges, such as task interference, negative transfer, and task dominance (Yu et al., 2020). Several methods have been proposed to mitigate these issues in a full fine-tuning setting (Chen et al., 2018; Kendall et al., 2018; Maninis et al., 2019; Senushkin et al., 2023; Ban and Ji, 2024). Another line of research has focused on branched multi-task networks (Lu et al., 2016; Vandenhende et al., 2019; Brüggemann et al., 2020; Guo et al., 2020). The branching aims to balance shared representations and task-specific learning to improve performance and efficiency. While some works use neural architecture search (Brüggemann et al., 2020) or reinforcement learning (Guo et al., 2020), our approach aligns with methods that leverage task affinity to group similar tasks (Vandenhende et al., 2019; Fifty et al., 2021). However, unlike these methods, our approach does not train separate models to compute task similarity, making it more efficient. Other methods to compute task similarity have also used Fisher information (Achille et al., 2019). Instead, we compute similarity directly from gradients.

Parameter-Efficient MTL. Although PEMTL remains relatively underexplored, several works have made notable contributions. To address the lack of

[2] https://github.com/neerajgangwar/progressive-task-specific-adaptation

knowledge transfer in individual single-task adaptation, Hyperformer (Mahabadi et al., 2021) uses a common hypernetwork to generate Adapter module parameters for different tasks, training only the hypernetwork. This approach facilitates knowledge transfer across tasks via the shared hypernetwork. Yang et al. (2025) propose MTL-LoRA for NLP tasks, which uses a shared down-projection matrix, task-specific transformation matrices, and multiple up-projection matrices. Polyhistor (Liu et al., 2022) is a parameter-efficient version of Hyperformer for dense vision tasks. While this approach enables knowledge transfer, the inference cost still increases linearly with the number of tasks. To address this, Agiza et al. (2024) introduce MTLoRA, closely approximating shared multi-task adaptation. Unlike shared multi-task adaptation, which relies solely on shared modules, MTLoRA incorporates task-specific modules in certain layers to enable task-specific learning. Although this mitigates some interference, most layers in MTLoRA remain shared across tasks, leading to task conflicts, and the number of task-specific modules scales linearly with the number of tasks. TADFormer (Baek et al., 2025) uses task-aware feature adaptation for dense prediction tasks. Additionally, mixture-of-experts has also been used to implement PEMTL (e.g., Yang et al., 2024; Lin et al., 2024b).

3 PROPOSED APPROACH

In our approach, adapter modules are shared across tasks in early layers and gradually become more task-specific in later layers. To reduce task interference, we assign similar tasks to an adapter module. This intelligent parameter sharing enables effective knowledge transfer among tasks while reducing task conflicts, addressing both the knowledge transfer limitations of individual single-task adaptation and the task interference challenges of shared multi-task adaptation. This architecture achieves better inference cost than individual single-task adaptation by significantly reducing the number of dedicated adapter modules. With a moderately higher inference cost than shared multi-task adaptation, this architecture offers significant performance improvement.

We define a *task group* as a set of tasks assigned to a single adapter module. Adding multiple modules of this type within a layer effectively creates copies of that layer, with each copy specialized for a task group. Each copy's parameters are updated based on the combined loss for its respective task group. As a result, the progressive task-specific adaptation shown in Figure 1c is equivalent to transforming a linear architecture into a branched structure, where the task-specific decoders serve as the leaves. The tree-like structure ensures the task specificity of the adapter modules. Notably, the adapter modules in the final layer can be shared across multiple tasks rather than being task-specific. This helps limit the proliferation of adapter modules and manage the resulting tree's width in the case of extreme MTL, addressing the linear expansion of task-specific modules observed in MTLoRA.

The number of task groups in a layer is determined by a predefined computational budget, such as parameter count or floating-point operations per second (FLOPS). To maintain the task-specificity of adapter modules, two tasks in a layer's task group must not belong to different task groups in any preceding layers.

3.1 Task Similarity

We use the notion of gradient conflicts from the MTL literature to compute the similarity between a pair of tasks. Formally, consider a set of tasks $\mathcal{T} = \{1, 2, \ldots T\}$ and model parameters $\{\theta_s\} \cup \{\theta_t | t \in \mathcal{T}\}$, where θ_s and θ_t represent the shared and task-specific parameters, respectively. Let $\mathcal{D}_t = \{x_{t,i} \mid 1 \leq i \leq |\mathcal{D}_t|\}$ represent the training dataset for task t. We define the similarity between two tasks based on the gradient of each task's loss with respect to the shared parameters. Specifically, the similarity between tasks t and t' is given by

$$\begin{gathered} \mathbb{E}_{x \sim \mathcal{D}_t, x' \sim \mathcal{D}_{t'}} \left[S\left(g\left(x,t\right), g\left(x',t'\right)\right)\right] \\ g(x,t) \triangleq \nabla_{\theta_s} \mathcal{L}(\theta_s, \theta_t, x) \end{gathered} \tag{1}$$

where S and $\mathcal{L}(.)$ represent the similarity and loss functions. Following Achille et al. (2019), we use the cosine similarity, indicated by $S_{\cos}$, between the normalized gradients to compute the similarity as follows

$$S(g, g') = S_{\cos}\left(\frac{g}{|g| + |g'|}, \frac{g'}{|g| + |g'|}\right)$$

Here, g and g' are shorthands for $g(x,t)$ and $g(x',t')$, respectively, and $|\cdot|$ computes element-wise absolute values. Note that (1) does not constrain that the inputs must be the same for all tasks and may be applied to tasks not sharing the same input.

We use a pre-trained backbone (θ_s) with task-specific heads (θ_t's) to compute the similarity. The task-specific heads are randomly initialized, so we fine-tune them on the training examples. The backbone remains frozen during this fine-tuning. For our experiments, the expectation, $\mathbb{E}$, in (1) is replaced by the empirical mean. As our task similarity is defined as the expectation of example-level similarity, we ensure that no model parameters are disabled in the forward pass due to dropout layers.

3.2 Task-Grouped LoRA (TGLoRA)

To implement the progressive task-specific architecture, we create a LoRA-based (Hu et al., 2022) layer: Task-

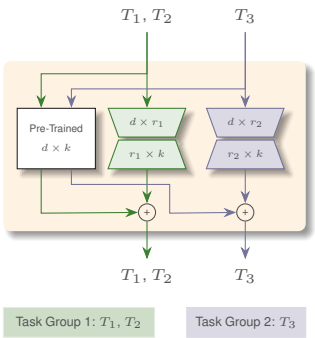


Figure 3: TGLoRA Layer with two task groups. T_1, T_2, and T_3 represent three tasks.

Grouped LoRA (TGLoRA). It consists of multiple low-rank modules, one for each task group. It takes as many inputs as the number of task groups and produces the same number of outputs. Let $W \in \mathbb{R}^{d \times k}$ represent a pre-trained weight matrix and $A_\pi \in \mathbb{R}^{r_\pi \times k}$ and $B_\pi \in \mathbb{R}^{d \times r_\pi}$ represent low-rank matrices with rank r_π for task group π. The input corresponding to task group π, $x_\pi \in \mathbb{R}^k$, is processed as

$$y_\pi = W x_\pi + \gamma_{r,\pi} B_\pi A_\pi x_\pi$$

where $\gamma_{r,\pi} = \frac{\alpha_\pi}{r_\pi}$ represents the scaling factor defined in Hu et al. (2022). Figure 3 shows this layer for two task groups. Note that a TGLoRA layer with one task group reduces to a LoRA layer.

3.3 Computing Task Groups

Creating task groups is a set partitioning problem. Formally, it can be defined as partitioning $\mathcal{T}$ into a partition $P = \{\pi_1 \dots \pi_M\}$ such that

- $\bigcup_{i=1}^{M} \pi_i = \mathcal{T}$; and
- $\pi_i \cap \pi_j = \emptyset \; \forall \; 1 \leq i, j \leq M$ and $i \neq j$

We assign a score to every partition P, based on the tasks in each group π_i, which we define momentarily. Intuitively, this score should be higher when similar tasks are grouped and lower when conflicting tasks are assigned to the same task group. This score is determined by evaluating the affinity of each task with other tasks within its respective group. Our objective is to find the partition that maximizes the score.

First, we assign a score to each task t in a task group π_i as

$$\mathcal{S}_{t,\pi_i} = \begin{cases} 0 & |\pi_i| = 1 \\ \frac{\sum_{t' \in \pi_i, t' \neq t} \text{sim}(t,t')}{|\pi_i| - 1} & \text{otherwise} \end{cases}$$

Algorithm 1 Task Group Merging.

Require: Tasks $\mathcal{T} = \{1 \dots T\}$, a set of task groups $P = \{\pi_1 \dots \pi_M\}$, task similarities $\{S_{t,t'} | t, t' \in \mathcal{T}\}$, $N < M$

```
Q ← P                                  ▷ Partition after merging
repeat
    Q' ← ∅                             ▷ Best partition after merging two groups
    s ← 0                              ▷ Score for the best partition
    for all π_i, π_j ∈ Q do
        P' ← (P \ {π_i, π_j}) ∪ {π_i ∪ π_j}
        s' ← S_P'                      ▷ Defined in (2)
        if s' > s then
            s ← s'
            Q' ← P'
        end if
    end for
    Q ← Q'
until |Q| = N
return Q
```

where $\text{sim}(t, t')$ is the task similarity defined in (1). Subsequently, the score for partition P is defined as

$$\mathcal{S}_P = \sum_{i=1}^{M} \sum_{t \in \pi_i} \mathcal{S}_{t,\pi_i} \tag{2}$$

To convert a pre-trained model to a branched network, task groups are computed in reverse. We first compute the task groups for the layer just before the decoders. If this layer contains task-specific modules with no sharing, we partition $\mathcal{T}$ into subsets of size one and move to the preceding layer. The branch-and-bound algorithm (Fifty et al., 2021) is used to find the partition that maximizes $\mathcal{S}_P$. For other layers, we either keep the same task groups as the succeeding layer or merge them to achieve the required number of groups. Algorithm 1 is used for task group merging.

4 EXPERIMENTS

4.1 Datasets

Following previous works in the MTL literature (Vandenhende et al., 2020; Xu et al., 2018; Ye and Xu, 2022; Liu et al., 2022; Agiza et al., 2024), we evaluate our approach on dense prediction tasks. We use the PASCAL (Everingham et al., 2010) and NYUD-v2 (Silberman et al., 2012) datasets.

For PASCAL, PASCAL-Context split (Chen et al., 2014) is used. It has four dense prediction tasks: (1) semantic segmentation, (2) saliency detection, (3) surface normal estimation, and (4) human part segmentation.

The dataset consists of training and validation splits, with 4,998 and 5,105 images, respectively.

We consider three tasks in the NYUD-v2 dataset: (1) semantic segmentation, (2) depth estimation, and (3) surface normal estimation. The dataset contains 795 training and 654 validation examples.

4.2 Evaluation Metrics

The performance on semantic segmentation, saliency detection, and human part segmentation is evaluated using mean intersection-over-union (mIoU). Surface normal estimation and depth estimation are evaluated using the root mean square error (rmse). Following previous works on MTL (Vandenhende et al., 2021; Liu et al., 2022; Agiza et al., 2024), we use the number of trainable parameters and the average per-task difference in performance compared to single-task full fine-tuning (Δm) to evaluate multi-task models. Δm is defined as

$$\Delta m = \frac{1}{T} \sum_{t=1}^{T} (-1)^{l_t} \, \frac{M_t - M_{st,t}}{M_{st,t}}$$

where M_t and $M_{st,t}$ represent the multi-task and single-task models' performances on task t, respectively. l_t is 0 if a higher score means better performance for task t and is 1 otherwise.

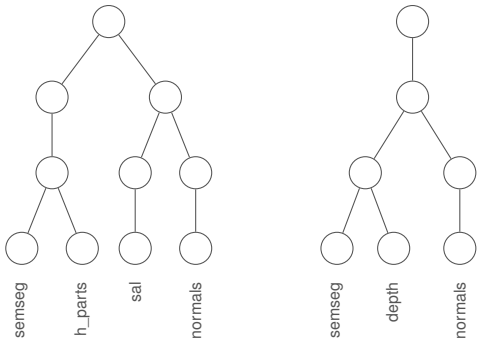
4.3 Setup

Base Model. For PASCAL, we replicate the model architecture from Agiza et al. (2024) for a fair comparison with the existing methods. Specifically, we use the Swin Transformer (Liu et al., 2021) and attach task-specific decoders for each task. These decoders are similar to those used in HR-Net (Wang et al., 2020), consisting of linear and bilinear upsampling layers. We use the Swin-Tiny variant pre-trained on ImageNet-1k. We use the same architecture for NYUD-v2. We also experiment with Pyramid Vision Transformer (PVT; Wang et al., 2021) for PASCAL and use the PVT-Small variant pre-trained on ImageNet-1k.

Furthermore, we employ multi-scale task-specific feature sharing used in MTLoRA, which merges features from different stages for a comprehensive feature representation. See Agiza et al. (2024) for more details on multi-scale task-specific feature sharing.

Optimization. The models are trained to minimize the multi-task loss

$$L_{\text{MTL}} = \sum_{i=1}^{T} w_t \times L_t \tag{3}$$



(a) PASCAL (b) NYUD-v2

Figure 4: Task groups computed for PASCAL (with Swin and PVT) and NYUD-v2 (with Swin).

where w_t and L_t are the task weight and loss for task t. Task weights are the same as in previous works (Liu et al., 2022; Agiza et al., 2024). We use per-pixel cross-entropy loss for semantic and human part segmentation, L1 loss for surface normal and depth estimation, and balanced cross-entropy loss for saliency detection.

Trainable Layers. We conduct our experiments by adding TGLoRA modules to the attention (indicated by ATTN) and MLP (indicated by MLP) layers. We determine the rank of the low-rank matrices in each task group of a TGLoRA layer based on the number of tasks in that group. Using a combined rank r for a TGLoRA layer, we allocate it proportionally across different task groups. This strategy ensures better control of the trainable parameter count across different configurations. Additionally, similar to MTLoRA, we explore unfreezing the patch embedding, patch merging, layer normalization, and position bias layers (indicated by OTHERS).

Training Details. We repeat each experiment three times with different random seeds and report the average scores. Further details on the training setup are provided in Section A of the supplementary material.

Baselines. We consider the following baselines: (1) Polyhistor (Liu et al., 2022), (2) MTLoRA (Agiza et al., 2024), and (3) Hyperformer (Mahabadi et al., 2021; Liu et al., 2022). Additionally, we include the following two baselines:

- *Single Task – LoRA:* We fine-tune single-task models by adding LoRA modules to various layers in the model.
- *MTL – LoRA:* We also fine-tune multi-task models by adding LoRA modules to various layers in the model.

Table 1: Comparison of progressive task-specific adaptation (indicated by TGLoRA) with the existing methods on PASCAL using a Swin backbone. ↑ and ↓ signify that higher and lower values are better, respectively. Results marked with † are from Agiza et al. (2024).

Method	**SemSeg** (mIoU ↑)	**Human Parts** (mIoU ↑)	**Saliency** (mIoU ↑)	**Normals** (rmse ↓)	Δm (% ↑)	**Trainable Params** (M ↓)
Single Task – Full Fine-Tuning†	67.21	61.93	62.35	17.97	0	112.62
MTL – Decoders Only†	65.09	53.48	57.46	20.69	−9.95	1.94
MTL – Full Fine-Tuning†	67.56	60.24	65.21	16.64	+2.23	30.06
Hyperformer†	71.43	60.73	65.54	17.77	+2.64	72.77
Polyhistor†	70.87	59.15	65.54	17.77	+2.34	8.96
MTLoRA† ($r = 64$)	67.90	59.84	65.40	16.60	+2.55	8.34
MTL – LoRA (ATTN + MLP)	$67.57_{\pm 0.19}$	$59.09_{\pm 0.10}$	$65.18_{\pm 0.07}$	$17.08_{\pm 0.03}$	+1.36	5.29
Single Task – LoRA (ATTN + MLP)	$70.80_{\pm 0.08}$	$58.73_{\pm 0.07}$	$66.05_{\pm 0.04}$	$16.65_{\pm 0.06}$	+3.36	5.29
TGLoRA (ATTN + MLP + OTHERS)	$70.53_{\pm 0.08}$	$60.96_{\pm 0.12}$	$66.12_{\pm 0.04}$	$16.43_{\pm 0.03}$	+4.50	6.89
TGLoRA (ATTN + MLP)	$70.39_{\pm 0.29}$	$60.58_{\pm 0.10}$	$65.94_{\pm 0.01}$	$16.61_{\pm 0.01}$	+3.97	5.29
TGLoRA (ATTN)	$70.27_{\pm 0.09}$	$59.36_{\pm 0.04}$	$65.38_{\pm 0.15}$	$17.09_{\pm 0.04}$	+2.54	3.20

Table 2: Comparison of progressive task-specific adaptation (indicated by TGLoRA) with the existing methods on NYUD-v2 using a Swin backbone. ↑ and ↓ signify that higher and lower values are better, respectively.

Method	**SemSeg** (mIoU ↑)	**Normals** (rmse ↓)	**Depth** (rmse ↓)	Δm (% ↑)	**Trainable Params** (M ↓)
Single Task – Full Fine-Tuning	$41.85_{\pm 0.37}$	$24.01_{\pm 0.05}$	$0.6322_{\pm 0.0014}$	0	84.04
MTL – Full Fine-Tuning	$41.17_{\pm 0.44}$	$24.75_{\pm 0.02}$	$0.6217_{\pm 0.0019}$	−1.01	29.00
MTL – Decoder Only	$35.97_{\pm 0.02}$	$32.63_{\pm 0.02}$	$0.8008_{\pm 0.0012}$	−25.54	1.48
MTLoRA ($r = 64$)	$41.52_{\pm 0.27}$	$24.99_{\pm 0.04}$	$0.6212_{\pm 0.0014}$	−1.04	7.81
MTL – LoRA (ATTN + MLP)	$41.05_{\pm 0.28}$	$25.01_{\pm 0.03}$	$0.6231_{\pm 0.0019}$	−1.54	5.93
Single Task – LoRA (ATTN + MLP)	$41.86_{\pm 0.17}$	$24.57_{\pm 0.03}$	$0.6295_{\pm 0.0022}$	−0.63	5.93
TGLoRA (ATTN + MLP + OTHERS)	$41.84_{\pm 0.29}$	$24.38_{\pm 0.06}$	$0.6177_{\pm 0.0030}$	+0.24	7.53
TGLoRA (ATTN + MLP)	$41.89_{\pm 0.24}$	$24.45_{\pm 0.03}$	$0.6236_{\pm 0.0039}$	−0.12	5.93
TGLoRA (ATTN)	$41.92_{\pm 0.28}$	$25.27_{\pm 0.04}$	$0.6319_{\pm 0.0039}$	−1.68	3.15

We scale the ranks of adapter modules in these two baselines to match the number of trainable parameters with TGLoRA. These baselines represent individual single-task and shared multi-task adaptations.

4.4 Results

Task Groups. For both datasets, we share adapter modules across tasks in the first stage, keep the last stage task-specific, and reduce the number of task groups as we move away from the decoders. Figure 4 shows the task groups computed by our method for PASCAL and NYUD-v2. Notably, both the Swin and PVT backbones result in the same task groups for PASCAL. More details on task group creation and the effect of different computational budgets are presented in Sections B and C.1 of the supplementary material.

MTL Performance. Tables 1 and 2 show the per-task performance, the overall MTL performance (Δm), and the number of trainable parameters for PASCAL and NYUD-v2, respectively, with a Swin backbone. For PASCAL, our approach achieves an absolute improvement of more than 2% in overall MTL performance over MTLoRA and Polyhistor and uses significantly fewer trainable parameters. Furthermore, it surpasses Hyperformer by a significant margin, which requires considerably more trainable parameters. Similar to Agiza et al. (2024), we also observe improved performance on unfreezing additional layers compared to fine-tuning only the newly added TGLoRA modules. As a final point, adding TGLoRA only to the attention layers is very effective and achieves comparable performance to the existing methods while requiring less than half the trainable parameters. For NYUD-v2, our approach outperforms MTLoRA by more than 1% with fewer trainable parameters. Moreover, it achieves better overall MTL performance than the existing methods while keeping additional layers frozen during fine-tuning, leading to a significantly better trade-off between the number of trainable parameters and performance.

For both datasets, the progressive task-specific architecture outperforms the single-task (Single Task – LoRA) and multi-task models with a shared backbone (MTL – LoRA) for the same parameter budget. This behavior remains consistent when TGLoRA is added only to the

Table 3: Comparison of progressive task-specific adaptation (indicated by TGLoRA) with the existing methods on PASCAL using a PVT backbone. ↑ and ↓ signify that higher and lower values are better, respectively. Results marked with † are from Agiza et al. (2024).

Method	**SemSeg** (mIoU ↑)	**Human Parts** (mIoU ↑)	**Saliency** (mIoU ↑)	**Normals** (rmse ↓)	Δm (% ↑)	**Trainable Params** (M ↓)
Single Task – Full Fine-Tuning†	68.81	61.27	62.67	17.55	0	97.51
MTL – Decoders Only†	64.86	51.18	61.54	19.55	−8.85	2.11
Hyperformer†	70.81	57.76	65.49	17.75	+0.14	16.14
Polyhistor†	71.00	57.52	65.83	17.83	+0.13	7.32
MTLoRA† ($r = 64$)	69.74	58.08	65.62	17.35	+1.20	8.69
MTLoRA (Reproduced, ATTN + SR)	70.32±0.41	59.06±0.10	66.10±0.07	16.99±0.02	+1.81	8.69
MTLoRA (Reproduced, ATTN)	70.34±0.40	58.83±0.13	65.89±0.08	17.15±0.03	+1.42	4.40
TGLoRA (ATTN)	72.90±0.17	60.15±0.13	66.76±0.02	16.91±0.03	+3.57	3.85

Table 4: Giga multiply-accumulate operations for individual single-task, shared multi-task, and progressive task-specific multi-task adaptations using a Swin backbone.

Dataset	**# Tasks**	**Individual Single-Task**	**Shared Multi-Task**	**Progressive Task-Specific**
PASCAL	1	18.49	18.49	18.49
	4	73.96	21.47	49.95
NYUD-v2	1	18.56	18.56	18.56
	3	55.41	20.53	34.65

attention layer and when additional layers are unfrozen during training. See Section C.2 in the supplementary material for the results with these configurations.

We also experiment with PVT to ensure that our method generalizes to different backbones. We follow the training setup from Agiza et al. (2024) and fine-tune PVT-Small on the PASCAL dataset. For both MTLoRA and TGLoRA, adapter modules are added to the attention layer (indicated by ATTN). Additionally, for MTLoRA, the sequence reduction layers are unfrozen (indicated by ATTN + SR). Table 3 shows the performance with PVT. These results demonstrate that TGLoRA outperforms the existing methods with fewer trainable parameters. They further show that unfreezing the sequence reduction layers in MTLoRA contributes little to the performance and significantly increases the trainable parameters.

Lastly, in addition to adding adapter modules to different layers, we control the number of trainable parameters by varying the ranks of individual adapter modules. We observe a similar pattern as reported in Table 1 for different values of ranks. See Section C.2 in the supplementary material for results.

Inference Cost. Table 4 shows the number of giga multiply-accumulate operations for our approach compared to individual single-task and shared multi-task adaptations. These results show that the inference cost for progressive task-specific adaptations lies between the two extremes.

Table 5: Performance of progressive task-specific adaptation with TGLoRA when less similar tasks are in the same task group. Δm is computed based on the single-task performance from Table 1 for PASCAL and Table 2 for NYUD-v2. These experiments use a Swin backbone.

Dataset	**Method**	Δm (% ↑)	**Trainable Params** (M ↓)
PASCAL	ATTN + MLP + OTHERS	+3.71	6.89
	ATTN + MLP	+3.14	5.29
	ATTN	+1.95	3.20
NYUD-v2	ATTN + MLP + OTHERS	−1.11	7.53
	ATTN + MLP	−0.90	5.93
	ATTN	−2.30	3.15

4.5 Sub-Optimal Task Groups

Next, we examine the impact of grouping less similar tasks, as per the task similarities computed in the previous section (shown in Figure 4), while maintaining the number of task groups. For PASCAL, we swap saliency and human part segmentation in the second and third stages of Swin. Similarly, we swap depth and surface normal estimation tasks in the third stage for NYUD-v2. The MTL performance under these configurations is illustrated in Table 5. These results demonstrate that grouping less similar tasks adversely impacts the model's performance, providing strong evidence in favor of our proposed method of computing task similarity and forming task groups.

5 LIMITATIONS

In our experiments, the number of task groups in different stages of Swin and PVT is selected manually. Although we report results for several branching config-

urations, exploring a broader range could yield deeper insights and strengthen our approach. An important future direction is to develop automated methods for selecting the optimal number of task groups in different layers. Another limitation lies in the task grouping algorithm itself. We use a branch-and-bound-style procedure to compute task groups in the final stage. While effective for a small number of tasks, this approach becomes intractable as the number of tasks grows. An approximate variant of the algorithm could make our method more scalable to large-scale MTL scenarios.

6 CONCLUSION

In this work, we introduced progressive task-specific multi-task adaptation, positioned between individual single-task and shared multi-task adaptations. To implement it, we developed a new LoRA-based layer named TGLoRA and a gradient-based metric to measure task similarity. We adapted the Swin Transformer and Pyramid Vision Transformer for dense prediction tasks. Our experiments demonstrate that progressive task-specific adaptation outperforms both individual single-task and shared multi-task adaptations, achieving significantly greater relative improvement over single-task models while using fewer trainable parameters than the current state-of-the-art approaches. Ablation studies further reveal that grouping less similar tasks adversely affects multi-task performance, highlighting the effectiveness of our method. Finally, our approach offers flexible control over the trade-off between computational budget and performance.

Acknowledgements

This work was supported in part by the Illinois Campus Cluster Program and the Delta Advanced Computing and Data Resource. Delta is supported by the National Science Foundation (award OAC 2005572) and the State of Illinois and is a joint effort of the University of Illinois Urbana-Champaign and its National Center for Supercomputing Applications. We also thank the anonymous reviewers for their feedback.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Achille, A., Lam, M., Tewari, R., Ravichandran, A., Maji, S., Fowlkes, C. C., Soatto, S., and Perona, P. (2019). Task2vec: Task embedding for meta-learning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6430–6439.

Agiza, A., Neseem, M., and Reda, S. (2024). MT-LoRA: Low-rank adaptation approach for efficient multi-task learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16196–16205.

Baek, S., Lee, S., Jo, H., Choi, H., and Min, D. (2025). Tadformer: Task-adaptive dynamic transformer for efficient multi-task learning. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 14858–14868.

Bai, Y., Geng, X., Mangalam, K., Bar, A., Yuille, A. L., Darrell, T., Malik, J., and Efros, A. A. (2024). Sequential modeling enables scalable learning for large vision models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22861–22872.

Ban, H. and Ji, K. (2024). Fair resource allocation in multi-task learning. In *Forty-first International Conference on Machine Learning*.

Brüggemann, D., Kanakis, M., Georgoulis, S., and Van Gool, L. (2020). Automated search for resource-efficient branched multi-task networks. In *31st British Machine Vision Conference 2020*, page 359. BMVA Press.

Chen, S., Ge, C., Tong, Z., Wang, J., Song, Y., Wang, J., and Luo, P. (2022). Adaptformer: Adapting vision transformers for scalable visual recognition. *Advances in Neural Information Processing Systems*, 35:16664–16678.

Chen, X., Mottaghi, R., Liu, X., Fidler, S., Urtasun, R., and Yuille, A. (2014). Detect what you can: Detecting and representing objects using holistic models and body parts. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1971–1978.

Chen, Z., Badrinarayanan, V., Lee, C.-Y., and Rabinovich, A. (2018). Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International conference on machine learning*, pages 794–803. PMLR.

Chen, Z., Duan, Y., Wang, W., He, J., Lu, T., Dai, J., and Qiao, Y. (2023). Vision transformer adapter for dense predictions. In *The Eleventh International Conference on Learning Representations*.

Crawshaw, M. (2020). Multi-task learning with deep neural networks: A survey. *arXiv preprint arXiv:2009.09796*.

Everingham, M., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88:303–338.

Fifty, C., Amid, E., Zhao, Z., Yu, T., Anil, R., and Finn, C. (2021). Efficiently identifying task groupings for multi-task learning. *Advances in Neural Information Processing Systems*, 34:27503–27516.

Gao, P., Han, J., Zhang, R., Lin, Z., Geng, S., Zhou, A., Zhang, W., Lu, P., He, C., Yue, X., et al. (2023). Llama-adapter v2: Parameter-efficient visual instruction model. *arXiv preprint arXiv:2304.15010*.

Guo, P., Lee, C.-Y., and Ulbricht, D. (2020). Learning to branch for multi-task learning. In *International conference on machine learning*, pages 3854–3863. PMLR.

Hayou, S., Ghosh, N., and Yu, B. (2024). Lora+: Efficient low rank adaptation of large models. In *Forty-first International Conference on Machine Learning*.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.

Hu, E. J., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. (2022). LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Hu, Z., Wang, L., Lan, Y., Xu, W., Lim, E.-P., Bing, L., Xu, X., Poria, S., and Lee, R. (2023). Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5254–5276.

Jia, M., Tang, L., Chen, B.-C., Cardie, C., Belongie, S., Hariharan, B., and Lim, S.-N. (2022). Visual prompt tuning. In *European Conference on Computer Vision*, pages 709–727. Springer.

Kendall, A., Gal, Y., and Cipolla, R. (2018). Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7482–7491.

Lester, B., Al-Rfou, R., and Constant, N. (2021). The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059.

Li, X. L. and Liang, P. (2021). Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597.

Lin, Y., Ma, X., Chu, X., Jin, Y., Yang, Z., Wang, Y., and Mei, H. (2024a). Lora dropout as a sparsity regularizer for overfitting control. *arXiv preprint arXiv:2404.09610*.

Lin, Z., Fu, H., Liu, C., Li, Z., and Sun, J. (2024b). Pemt: Multi-task correlation guided mixture-of-experts enables parameter-efficient transfer learning. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 6869–6883.

Liu, S.-y., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., Cheng, K.-T., and Chen, M.-H. (2024). Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*.

Liu, Y.-C., Ma, C.-Y., Tian, J., He, Z., and Kira, Z. (2022). Polyhistor: Parameter-efficient multi-task adaptation for dense vision tasks. *Advances in Neural Information Processing Systems*, 35:36889–36901.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022.

Loshchilov, I. and Hutter, F. (2017). Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Lu, Y., Kumar, A., Zhai, S., Cheng, Y., Javidi, T., and Feris, R. S. (2016). Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1131–1140.

Mahabadi, R. K., Ruder, S., Dehghani, M., and Henderson, J. (2021). Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 565–576.

Maninis, K.-K., Radosavovic, I., and Kokkinos, I. (2019). Attentive single-tasking of multiple tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1851–1860.

Pfeiffer, J., Kamath, A., Rücklé, A., Cho, K., and Gurevych, I. (2021). Adapterfusion: Non-destructive task composition for transfer learning. In *European Chapter of the Association for Computational Linguistics*, pages 487–503.

Pfeiffer, J., Rücklé, A., Poth, C., Kamath, A., Vulic, I., Ruder, S., Cho, K., and Gurevych, I. (2020). Adapterhub: A framework for adapting transformers. *EMNLP 2020*, page 46.

Senushkin, D., Patakin, N., Kuznetsov, A., and Konushin, A. (2023). Independent component alignment for multi-task learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20083–20093.

Silberman, N., Hoiem, D., Kohli, P., and Fergus, R. (2012). Indoor segmentation and support inference from rgbd images. In *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part V 12*, pages 746–760. Springer.

Sung, Y.-L., Nair, V., and Raffel, C. A. (2021). Training neural networks with fixed sparse masks. *Advances in Neural Information Processing Systems*, 34:24193–24205.

Team, G., Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., et al. (2023). Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. (2023). Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Vandenhende, S., Georgoulis, S., De Brabandere, B., and Van Gool, L. (2019). Branched multi-task networks: Deciding what layers to share. *Proceedings BMVC 2020*.

Vandenhende, S., Georgoulis, S., Van Gansbeke, W., Proesmans, M., Dai, D., and Van Gool, L. (2021). Multi-task learning for dense prediction tasks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3614–3633.

Vandenhende, S., Georgoulis, S., and Van Gool, L. (2020). Mti-net: Multi-scale task interaction networks for multi-task learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16*, pages 527–543. Springer.

Wang, J., Sun, K., Cheng, T., Jiang, B., Deng, C., Zhao, Y., Liu, D., Mu, Y., Tan, M., Wang, X., et al. (2020). Deep high-resolution representation learning for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(10):3349–3364.

Wang, W., Xie, E., Li, X., Fan, D.-P., Song, K., Liang, D., Lu, T., Luo, P., and Shao, L. (2021). Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 568–578.

Wang, Y., Agarwal, S., Mukherjee, S., Liu, X., Gao, J., Hassan, A., and Gao, J. (2022). Adamix: Mixture-of-adaptations for parameter-efficient model tuning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5744–5760.

Xin, Y., Luo, S., Zhou, H., Du, J., Liu, X., Fan, Y., Li, Q., and Du, Y. (2024). Parameter-efficient fine-tuning for pre-trained vision models: A survey. *arXiv preprint arXiv:2402.02242*.

Xu, D., Ouyang, W., Wang, X., and Sebe, N. (2018). Pad-net: Multi-tasks guided prediction-and-distillation network for simultaneous depth estimation and scene parsing. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 675–684.

Yang, Y., Jiang, P.-T., Hou, Q., Zhang, H., Chen, J., and Li, B. (2024). Multi-task dense prediction via mixture of low-rank experts. In *IEEE/CVF conference on computer vision and pattern recognition*, pages 27927–27937.

Yang, Y., Muhtar, D., Shen, Y., Zhan, Y., Liu, J., Wang, Y., Sun, H., Deng, W., Sun, F., Zhang, Q., et al. (2025). Mtl-lora: Low-rank adaptation for multi-task learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 22010–22018.

Ye, H. and Xu, D. (2022). Inverted pyramid multi-task transformer for dense scene understanding. In *European Conference on Computer Vision*, pages 514–530. Springer.

Yu, J., Dai, Y., Liu, X., Huang, J., Shen, Y., Zhang, K., Zhou, R., Adhikarla, E., Ye, W., Liu, Y., et al. (2024). Unleashing the power of multi-task learning: A comprehensive survey spanning traditional, deep, and pretrained foundation model eras. *arXiv preprint arXiv:2404.18961*.

Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., and Finn, C. (2020). Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836.

Zaken, E. B., Goldberg, Y., and Ravfogel, S. (2022). Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9.

Zhang, Q., Chen, M., Bukharin, A., He, P., Cheng, Y., Chen, W., and Zhao, T. (2023). Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes, Section 3]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes, Sections 3 and 4]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [No]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Not Applicable]
   (b) Complete proofs of all theoretical results. [Not Applicable]
   (c) Clear explanations of any assumptions. [Not Applicable]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [`https://github.com/neerajgangwar/progressive-task-specific-adaptation`]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes, Section A in the supplementary material]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator if your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Not applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Parameter-Efficient Multi-Task Learning via Progressive Task-Specific Adaptation: Supplementary Materials

A TRAINING HYPERPARAMETERS

PASCAL. We replicate the hyperparameters from Agiza et al. (2024) for fine-tuning the models on the PASCAL dataset. Specifically, we use the AdamW optimizer (Loshchilov and Hutter, 2017) with a batch size of 32, a learning rate of 3.125×10^{-5}, and a weight decay of 0.05. The models are fine-tuned for 300 epochs, with evaluations every 20 epochs. We use a linear warmup for the first 20 epochs, followed by a cosine annealing learning rate scheduler. For the LoRA modules, a dropout rate of 0.05 and a scaling ($\gamma_{r,\pi}$) of 4 are used. We also apply the following data augmentations: `RandomHorizontalFlip`, `RandomScale` ranging from 0.75 to 1.25, and `RandomRotate` ranging from -20 to 20. We use combined ranks (r) of 64, 32, and 16 for TGLoRA. For Swin, we use a fixed rank of four for all task-specific modules in the last stage. Although we keep the rank constant in the last stage of Swin, it could also be scaled according to r, which would increase the number of trainable parameters. Our experiments use A100 GPUs (40GB VRAM).

NYUD-v2. We use a batch size of 6 and a learning rate of 10^{-4} for NYUD-v2. We use a combined rank of 64 for TGLoRA. We also apply the following data augmentations: `RandomHorizontalFlip` and `RandomScale` with possible scales of 1, 1.2, and 1.5. All other hyperparameters remain the same as those in PASCAL. Our experiments use A100 GPUs (40GB VRAM).

B TASK GROUPS

We determine the number of task groups in each TGLoRA layer according to a predefined computational budget, such as parameter count or floating-point operations per second (FLOPS). Fixing the computational budget fixes the branching in the architecture. For simplicity, we do not change the number of task groups in the TGLoRA layers within a stage of Swin or PVT.

As discussed in Section 3.1, fine-tuning the decoders is necessary for computing task similarity. We examine the impact of fine-tuning them with a limited subset of training examples versus the full training dataset. Our experiments show that both approaches produce identical task groups. Furthermore, repeating the fine-tuning process three times with different random seeds confirms the consistency of the results. These results confirm that this fine-tuning step introduces minimal overhead in the training pipeline.

C ADDITIONAL EXPERIMENTS

C.1 Computational Budget vs Performance

The tree structure in our approach offers a trade-off between model performance and inference cost. For instance, using 6.89M parameters for PASCAL using a Swin backbone, and assigning one, one, two, and four task groups to the first through fourth stages, respectively, results in $\Delta m = +3.93\%$ with 38.37 GMacs. Similarly, configuring the stages with one, two, two, and four task groups results in $\Delta m = +4.21\%$ and 41.38 GMacs.

C.2 Detailed Results for PASCAL and NYUD-v2

In this section, we present additional experiments on the PASCAL and NYUD-v2 datasets. Continuing from Section 4.4, Table 6 illustrates the performance of TGLoRA for varying trainable parameters. The rank of the low-rank modules in TGLoRA layers controls this number. Ranks for adapter modules in *Single Task – LoRA* and *MTL – LoRA* are chosen to keep the number of trainable parameters the same as TGLoRA. The table also shows

Table 6: Additional results on the PASCAL dataset using a Swin backbone. The number of trainable parameters is controlled by varying the rank of the LoRA modules. Δm is computed based on the single-task performance from Table 1.

Trainable Layers	**SemSeg** (mIoU ↑)	**Human Parts** (mIoU ↑)	**Saliency** (mIoU ↑)	**Normals** (rmse ↓)	Δm (% ↑)	**Trainable Params** (M ↓)
			TGLoRA			
ATTN + MLP + OTHERS	70.66±0.04	60.48±0.11	65.56±0.12	16.65±0.04	+3.82	5.41
	70.66±0.07	59.76±0.08	65.15±0.02	16.79±0.08	+3.17	4.68
ATTN + MLP	70.40±0.07	60.01±0.03	65.34±0.12	16.90±0.03	+3.10	3.81
	70.34±0.26	59.17±0.06	64.84±0.08	17.13±0.03	+2.21	3.08
ATTN	70.33±0.09	58.90±0.10	64.72±0.11	17.45±0.05	+1.61	2.65
	70.27±0.11	58.37±0.03	64.09±0.20	17.69±0.06	+0.79	2.37
			MTL – LoRA			
ATTN + MLP + OTHERS	67.62±0.27	59.22±0.06	65.28±0.09	16.86±0.05	+1.78	6.89
	68.07±0.08	59.07±0.05	64.96±0.05	17.11±0.05	+1.41	5.41
	68.52±0.08	58.78±0.06	64.29±0.03	17.30±0.01	+0.93	4.68
ATTN + MLP	68.02±0.04	58.78±0.05	64.77±0.03	17.33±0.01	+0.89	3.81
	68.35±0.14	58.30±0.07	63.99±0.09	17.62±0.03	+0.10	3.08
ATTN	68.25±0.11	58.42±0.07	64.38±0.09	17.57±0.04	+0.34	3.20
	68.52±0.06	58.11±0.10	63.73±0.05	17.92±0.02	−0.43	2.65
	68.62±0.13	57.65±0.16	63.23±0.12	18.20±0.03	−1.17	2.37
			Single Task – LoRA			
ATTN + MLP + OTHERS	71.16±0.06	59.42±0.09	66.21±0.10	16.41±0.06	+4.17	11.69
	71.32±0.04	59.05±0.08	65.78±0.11	16.58±0.03	+3.68	10.21
	71.37±0.08	58.70±0.03	65.35±0.03	16.67±0.04	+3.25	9.48
ATTN + MLP	70.83±0.13	58.15±0.04	65.54±0.08	16.95±0.10	+2.52	3.81
	70.82±0.08	57.70±0.04	64.97±0.05	17.14±0.05	+1.84	3.08
ATTN	70.65±0.05	57.78±0.03	65.20±0.10	17.25±0.03	+1.75	3.20
	70.63±0.11	57.47±0.09	64.75±0.15	17.48±0.05	+1.12	2.65
	70.52±0.12	56.99±0.03	64.11±0.10	17.74±0.02	+0.26	2.37

Table 7: Additional results on the NYUD-v2 dataset using a Swin backbone. Δm is computed based on the single-task performance from Table 2.

Trainable Layers	**SemSeg** (mIoU ↑)	**Normals** (rmse ↓)	**Depth** (rmse ↓)	Δm (% ↑)	**Trainable Params** (M ↓)
		MTL – LoRA			
ATTN + MLP + OTHERS	41.03±0.33	25.01±0.05	0.6216±0.0040	−1.48	7.53
ATTN	41.92±0.43	25.81±0.05	0.6309±0.0028	−2.37	3.15
		Single Task – LoRA			
ATTN + MLP + OTHERS	42.10±0.15	24.49±0.04	0.6310±0.0047	−0.40	10.73
ATTN	41.69±0.21	25.34±0.02	0.6376±0.0048	−2.26	3.15

the performance of *Single Task – LoRA* and *MTL – LoRA* when the attention and MLP layers are augmented with TGLoRA and when the patch embedding, patch merging, layer normalization, and position bias layers are unfrozen during fine-tuning. Note that unfreezing additional layers results in more trainable parameters in the single-task models, as these layers are unfrozen in the individual models. Hence, the ATTN + MLP + OTHERS configuration has more trainable parameters than the corresponding TGLoRA or *MTL – LoRA* counterparts. Our experiments demonstrate that progressive task-specific adaptation outperforms individual single-task and shared multi-task adaptations when different layers are augmented with TGLoRA or when additional layers are unfrozen during fine-tuning.