TaskIT: Memory-Efficient Fine-Tuning of Multi-LoRA LLMs via Cross-Task Importance Transfer

Cheng Fang[1] Zimu Zhou[1*] Ke Ma[2] Bin Guo[2]
[1]City University of Hong Kong [2]Northwestern Polytechnical University

Project Page: https://github.com/haojuns/TaskIT

Abstract

On-device AI systems increasingly adopt a single foundation model equipped with task-specific Low-Rank Adaptation (LoRA) modules, forming a multi-LoRA LLM that supports multiple tasks. We study how to adapt such a model to a new task on memory-constrainted devices. Although LoRA reduces trainable parameters, fine-tuning a full set of modules remains memory-intensive. To improve efficiency, we apply sparse updating, training a subset of LoRA modules within the memory budget. However, existing sparse updating methods assume all candidate parameters are instantiated and cannot estimate the importance of modules that do not yet exist, while prior memory models designed for sequential networks fail to capture the block-wise parallel structure of Transformers. We propose TaskIT, a framework for memory-efficient fine-tuning via cross-task importance transfer. TaskIT predicts pre-insertion module importance by transferring from previously tuned tasks and employs a block-based memory predictor that captures parallel and sequential dependencies of Transformer blocks. A dynamic programming scheduler then selects module locations, numbers, and ranks to maximize accuracy within the memory budget. Experiments on uni-modal and cross-modal benchmarks show that TaskIT achieves superior accuracy-memory tradeoffs compared with Zero-FT, non-LoRA, and LoRA-based fine-tuning methods.

1. Introduction

On mobile and edge devices, AI is increasingly built around a single foundation model rather than separate models per task. A pre-trained backbone is deployed once and kept frozen, while each downstream task is customized by a set of Low-Rank Adaptation (LoRA) modules [29] attached to the backbone. These modules are the only parameters trained per task and are merged into the backbone at inference, so all tasks reuse the same heavy computation while differing only in their LoRA states. For example, recent research envisions the foundation model as OS-managed firmware that apps extend with task-specific modules [74], and Apple's on-device foundation models adopt modular fine-tuning for in-app assistance [2]. This **multi-LoRA LLM** paradigm minimizes storage duplication and enables scalable on-device adaptation to new tasks.

Although LoRA fine-tuning is parameter-efficient, it remains **memory-intensive**. For example, compared with full fine-tuning of ViT-g [77], LoRA updates 1.8% of the parameters yet consumes 56% of the peak memory [48]. This is because it still needs to store LoRA parameters, activations, gradients, and optimizer states. Freezing the backbone reduces gradient and optimizer memory but barely lowers activation memory. Thus, training a full LoRA set for each new task easily exhausts device memory and limits the number of supported tasks.

A natural way to enable memory-efficient fine-tuning is to **sparsely** insert trainable LoRA modules into the backbone. *(i)* The configuration of LoRA modules affects the peak memory during fine-tuning. Specifically, the number and placement of LoRA modules determine the activation memory during backpropagation, while their number and ranks determine the overall LoRA parameters, gradients, and optimizer states memory [78]. *(ii)* LoRA modules contribute unevenly to fine-tuning accuracy, and their importance varies across layers, ranks, and tasks [5, 19, 73]. For example, visual question answering benefits more from deeper-layer modules [73], whereas sentiment analysis relies more on shallower ones [19].

In this paper, we study **sparse fine-tuning of multi-LoRA LLMs for an unseen task**, where only a subset of important LoRA modules are inserted and trained under a memory budget. Prior work on sparse updating [31, 42] does not apply to this setting, which raises two key challenges. **Challenge #1: Predicting module importance before insertion.** Most importance estimation methods [4, 27] assume all candidate parameters already exist, as they rely on quantities such as the parameters or their gradients ex-

*Corresponding author: zimuzhou@cityu.edu.hk

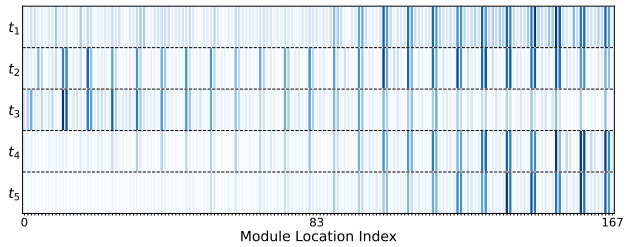

Figure 1. LoRA module importance estimated by TaskIT (§ 4.2) at different backbone locations (x-axis) across five example tasks (y-axis), with darker colors indicating higher importance. t_1: text-to-image [62]; t_2: image captioning [25]; t_3: image classification [18]; t_4 question answering [53]; t_5: text summary [24].

tracted from a trained model. However, we cannot insert and train a full set of LoRA modules merely to measure their importance. **Challenge #2: Predicting module memory in Transformers.** Existing memory modeling [42] assumes that memory scales linearly with the module index, which fits sequential networks but fails for Transformers, whose attention blocks contain parallel projection matrices across multiple branches that break this linearity.

We propose TaskIT (Cross-Task Importance Transfer), a memory-efficient fine-tuning framework for multi-LoRA LLMs. **First**, relevant tasks induce similar distributions of LoRA module importance along the backbone (Fig. 1). Hence, TaskIT treats importance prediction as **cross-task transfer** from previously learned tasks in the multi-LoRA model. It estimates task similarity, transfers module importance from similar tasks, and assigns importance to modules of the new task prior to their training. **Second**, we propose a block-based memory predictor that separates parallel dependencies **within a block** from sequential dependencies **across blocks**, and accurately predicts how LoRA module insertion affects activation memory. **Finally**, TaskIT feeds module importance and memory predictions into a **dynamic-programming** (DP) scheduler that configures the locations, number, and ranks of LoRA modules to maximize accuracy within the memory budget.

Our main contributions are as follows.

- We advance sparse updating to a new setting of memory-constrained fine-tuning of multi-LoRA LLMs on unseen tasks. Prior studies assume candidate parameters already exist and cannot estimate module importance before insertion. We close this gap by cross-task importance transfer, which leverages previously tuned tasks to predict the importance of uninserted modules for a new task.
- We develop TaskIT, an effective and memory-efficient framework that implements this idea with three components: a parameter-free importance predictor that avoids training full LoRA sets, a block-based memory predictor tailored to Transformers, and a DP-based scheduler that efficinetly searches for sparse insertion strategies.
- Experiments show that TaskIT achieves a better accuracy-memory tradeoff than Zero-FT [30, 35], non-LoRA [9, 48], and LoRA-based fine-tuning [78, 79], and that it generalizes across uni- and cross-modal tasks.

2. Related Work

2.1. Efficient LoRA-based Fine-tuning

On-device adaptation of foundation models has driven extensive research on efficient fine-tuning. Zero-FT schemes such as prompt tuning [35] and model merging [11, 30, 51] adapt to new tasks **without updating model parameters**. Parameter-efficient fine-tuning (PEFT) methods, including adapter tuning [9, 28, 35], selective backbone tuning [33, 49, 75], side tuning [15, 34, 44, 48], and LoRA [29], update only **lightweight task-specific components** while keeping most parameters frozen. We apply LoRA-based fine-tuning as it strucually aligns with our multi-LoRA LLM setting, and empirically achieves better performance than Zero-FT and non-LoRA baselines (§ 5.2).

Several studies further improve the efficiency of standard LoRA. π-Tuning [68] merges existing LoRA modules through gradient-based weighting, avoiding training from scratch but still incurring memory overhead comparable to standard LoRA. Other works adjust LoRA ranks across layers, assigning higher rank to important modules and lower rank to less important ones [16, 78, 79]. NOAH [80] further employs neural architecture search to pick optimal per-layer ranks. These rank-based strategies balance **parameter count** and accuracy but remain memory-intensive, since LoRA's peak memory depends on both rank and insertion location within the backbone. In contrast, we jointly optimize the **rank, number, and location** of LoRA modules by predicting their importance and memory usage, achieving memory-efficient LoRA-based fine-tuning.

2.2. Sparse Updating

Sparse updating freezes unimportant layers and trains only accuracy-critical ones, and has been applied in memory-efficient training of **small models**. Its effectiveness depends on estimating a layer's **importance** and **memory cost**.

Prior **importance prediction** approaches fall into three categories. Activation-based methods use statistics [20, 27] or semantic similarity [54, 66] between layer inputs and outputs. Loss-based methods estimate the impact of removing parameters by computing first-order [4, 31, 45, 78, 79] or second-order [57] loss approximations. Proxy-based methods infer importance via an external predictor [41], search algorithm [55], or policy network [22]. However, all of them assume the evaluated parameters **already exist**. In our setting, LoRA parameters for the new task are **unknown before insertion**. Estimating their importance would require training all modules, which defeats the purpose of sparse in-

sertion. We address this dilemma by **transferring** module importance from learned tasks to the new one.

Prior **memory prediction** studies fall into two categories. Activation-based methods [8, 27, 41] predict per-layer memory from activation sizes but fail to model the cumulative backward dependencies during training. Location-based methods [42] approximate per-layer memory by combining activation size with a layer-position index, a proxy for backward-path length, assuming roughly linear growth along network depth. This assumption holds for sequential networks but breaks for Transformers, whose multi-branch attention structure violates linear layer ordering. Instead, we introduce a block-based module memory predictor that models parallel dependencies **within** blocks and sequential dependencies **across** blocks.

3. Problem Statement

On-device **multi-LoRA foundation models** must adapt to **new tasks** on **memory-restricted** platforms. We study how to fine-tune such a model by **selectively** inserting LoRA modules into the frozen backbone under a memory budget. Tab. 4 in Appendix A.1 summarizes the major notations.

Consider a Transformer-based model with N modality-specific encoders, M modality-specific decoders, and a shared backbone θ comprising frozen projection matrices set $\mathcal{K}$. The model has already learned tasks t_1 through t_n, each associated with task-specific LoRA modules $\boldsymbol{w}_i = \{w_i^k\}_{k\in\mathcal{K}}$ inserted into the frozen matrix $\boldsymbol{W}_k$ for task t_i. For a new task t_{n+1}, our goal is to selectively insert and fine-tune LoRA modules while keeping the backbone frozen and respecting a memory budget M_B. Let $\boldsymbol{s}$ be the insertion strategy, where each entry specifies the LoRA rank at a potential module location (with zero indicating no insertion). Let $\boldsymbol{\lambda}_{n+1}$ be the module importance vector, whose entries predict the contribution of each location to the accuracy on t_{n+1}, and let $M(\boldsymbol{s})$ be the resulting memory usage. We formulate the problem as:

$$\max_{\boldsymbol{s}} \ \boldsymbol{s}\cdot\boldsymbol{\lambda}_{n+1} \quad \text{s.t.} \quad M(\boldsymbol{s}) \le M_B. \tag{1}$$

While similar to prior sparse updating [31, 42], Eq.(1) is for a new task whose modules do not yet exist in a multi-LoRA foundation model, which introduces two challenges.

- **Importance prediction without module insertion.** Existing methods (§ 2.2) estimate importance from already inserted modules. For the new task t_{n+1}, no task-specific modules exist beforehand. Obtaining $\boldsymbol{\lambda}_{n+1}$ by training a full set of LoRA modules contradicts the goal of sparse insertion, making these approaches infeasible.
- **Accurate memory prediction in Transformers.** Given insertion strategy $\boldsymbol{s}$, its peak fine-tuning memory $M(\boldsymbol{s})$ is dominated by the parameter and activation memory. While parameter memory is only related to the number and size of inserted modules, activation mememoy also depends on their insertion locations. Previous work [42] assumes the activation memory decreases linearly with layer depth. Yet the multi-branch attention blocks make such linear approximations inaccurate.

4. Method

4.1. TaskIT Overview

We propose TaskIT (Cross-Task Importance Transfer), a framework that predicts module importance for a new task without training a full set of LoRA modules and accurately predicts the resulting activation memory for Transformers.

- To obtain the importance $\boldsymbol{\lambda}_{n+1}$ for uninserted modules, we consider importance prediction as a **transfer** problem. TaskIT measures the importance of existing modules for learned tasks and weigh these importance scores by task similarities. It then uses the weighted average as the importance of the new task's modules at the same locations.
- To predict activation memory for an insertion strategy $\boldsymbol{s}$, we analyze how parallel dependencies **within blocks** and sequential dependencies **across blocks** affect activations to be retained during backpropagation. By aggregating these block-level effects, we accurately predict the contribution of each module to the activation memory usage.

Finally, TaskIT employs a **DP-based** scheduler to effectively search for a near-optimial sparse insertion strategy $\boldsymbol{s}$ that satisfies the memory budget.

Fig. 2 illustrates the workflow of TaskIT. For a new task t_{n+1}, the **module importance predictor** (§ 4.2) predicts the importance of each insertion location, and the **module memory predictor** (§ 4.3) maps a strategy $\boldsymbol{s}$ to its peak fine-tuning memory usage. The DP-based **module insertion scheduler** (§ 4.4) then selects $\boldsymbol{s}$ that optimizes the objective in Eq.(1). The selected modules are inserted and fine-tuned, which are then merged with the frozen backbone for inference on task t_{n+1}.

4.2. Module Importance Predictor

Given a new task t_{n+1}, we aim to predict its module importance vector $\boldsymbol{\lambda}_{n+1}$ without inserting new LoRA modules. We formulate it as a **cross-task transfer** problem, where the importance λ_{n+1}^k of the module at location k is estimated by the importance of modules at k from the n learned tasks,

$$\lambda_{n+1}^k = \sum_{i=1}^{n} \phi(t_{n+1}, t_i)\cdot\lambda_i^k, \tag{2}$$

where λ_i^k is the importance of module w_i^k for task t_i, and $\phi(\cdot,\cdot)$ measures task similarity. Tasks more similar to t_{n+1} contribute more to the predicted importance.

Implementing the idea needs two steps. *(i)* Quantify importance λ_i^k for each already learned module w_i^k at location

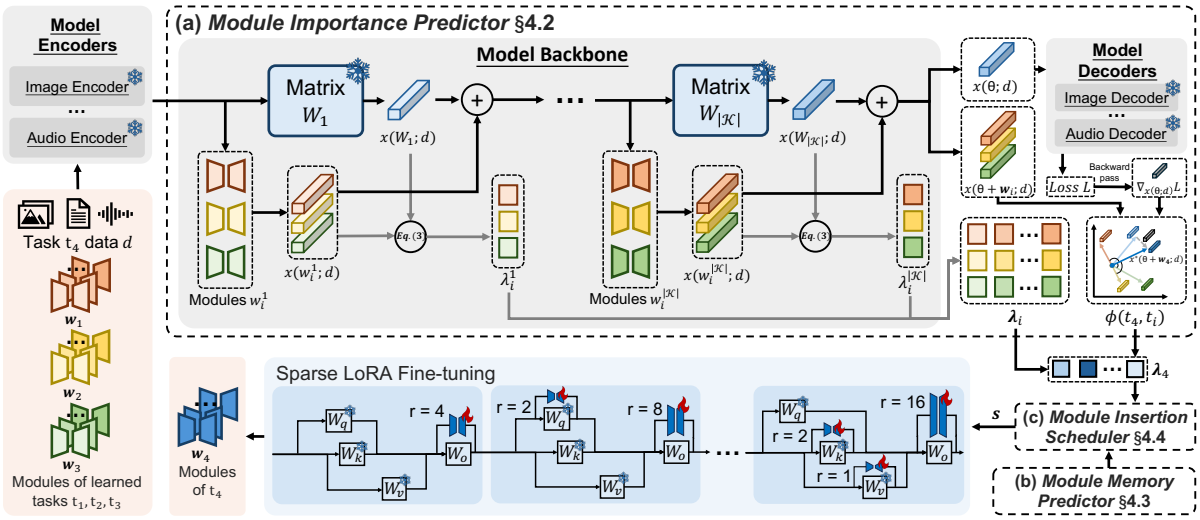

Figure 2. TaskIT overview. (a) The module importance predictor predicts new task importance by transferring weighted importance from existing task modules based on task similarity. (b) The module memory predictor predicts the memory usage of insertion strategy. (c) The module insertion scheduler searches for near-optimial sparse insertion strategy based on results from (a) and (b). The components of the multi-LoRA foundation model are shaded in gray.

k (§ 4.2.1). *(ii)* Compute similarity $\phi(t_{n+1}, t_i)$ between the new and learned tasks as the transfer weight (§ 4.2.2). Both steps should be **accurate** and **memory-efficient**, as they operate on the same memory-limited devices for fine-tuning.

4.2.1. Module Importance Estimation for Learned Tasks

We infer the module importance of a learned task t_i ($1 \leq i \leq n$) using an **activation-based**, **LoRA-tailored** estimator evaluated on the **new task's data**. Specifically, the importance of module w_i^k at backbone position k is

$$\lambda_i^k = \frac{\|x(w_i^k; d)\|_1}{\|x(W^k; d)\|_1 + \epsilon}, \tag{3}$$

where d is batched data from the new task t_{n+1}, $x(w_i^k; d)$ and $x(W^k; d)$ are the output activations of the LoRA module w_i^k and frozen matrix W^k on d, and $\epsilon = 10^{-8}$ ensures numerical stability. We use the L1 norm for computational efficiency and robustness to outliers.

This design is both memory-efficient and accurate.

- **Memory efficiency.** Activation-based estimation [20, 56] only needs **forward** passes, whereas gradient- [4, 31] and proxy-based [41, 55] methods require full backpropagation, which are more memory-intensive.
- **LoRA-aware accuracy.** Prior activation-based estimators [20, 56] assess importance solely by the absolute activation of the evaluated layer. In LoRA, the module output is merged with the matrix output, so importance should depend on the module's **relative** contribution to the combined activation. Our **ratio-based** formulation explicitly captures such relativeness and better aligns with LoRA.

4.2.2. Task Similarity Assessment

We measure the similarity $\phi(t_{n+1}, t_i)$ between the new task t_{n+1} and each learned task t_i ($1 \leq i \leq n$) from their **activations** on the same batched input d from t_{n+1}, **without** requiring LoRA parameters trained for t_{n+1}.

Principle. Activation similarity is a direct proxy for task similarity. If two tasks are semantically related, their activations (backbone plus task-specific LoRAs) should exhibit similar responses to the same input [39]. Let $\boldsymbol{w}_i$ be the trained LoRA modules for task t_i, which produce activations $x(\theta + \boldsymbol{w}_i; d)$ optimized for t_i. If LoRA modules $\boldsymbol{w}_{n+1}$ for t_{n+1} were available, they would generate activations $x(\theta + \boldsymbol{w}_{n+1}; d)$ reflecting the objective of t_{n+1}. The cosine similarity between $x(\theta + \boldsymbol{w}_i; d)$ and $x(\theta + \boldsymbol{w}_{n+1}; d)$ would then quantify how similar t_i is to t_{n+1}.

Challenge. In our setting, $\boldsymbol{w}_{n+1}$ is not yet trained, so $x(\theta + \boldsymbol{w}_{n+1}; d)$ is unavailable. This challenge resembles that in module importance prediction (§ 3), where task-specific parameters are umissing. However, our goal here is simpler. Rather than inferring parameters or their importance, we only approximate the **final activation** of a trained model $\theta + \boldsymbol{w}_{n+1}$ on task t_{n+1}.

Solution. We approximate $x(\theta + \boldsymbol{w}_{n+1}; d)$ by **reversing a single, simulated optimization step** in the activation space (upper-right of Fig. 2): *(i)* run a forward pass with the frozen backbone to obtain $x(\theta; d)$; *(ii)* feed $x(\theta; d)$ into the decoder and compute the new-task loss L; *(iii)* backpropagate only to $x(\theta; d)$ to obtain $\nabla_{x(\theta;d)} L$; *(iv)* take a small descent step:

$$x^*(\theta + \boldsymbol{w}_{n+1}; d) = x(\theta; d) - \eta \nabla_{x(\theta;d)} L, \tag{4}$$

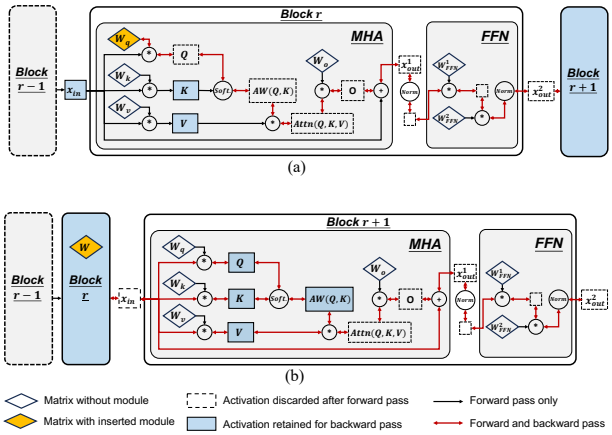

Figure 3. (a) Intra- and (b) inter-block activation dependency.

where $\eta > 0$ is a small step size.

For each learned task t_i, we directly obtain $x(\theta + \boldsymbol{w}_i; d)$ via a single forward pass through the backbone θ merged with $\boldsymbol{w}_i$. The similarity $\phi(t_{n+1}, t_i)$ is then defined as the cosine similarity between $x(\theta + \boldsymbol{w}_i; d)$ and the approximated activation $x^*(\theta + \boldsymbol{w}_{n+1}; d)$. Empirically, we only use tasks with similarity $\phi(t_{n+1}, t_i)$ above 0.35 to ensure positive transfer (Appendix C.1). This yields an **accurate**, **pre-finetuning** estimate of task relatedness using only one forward and one lightweight backward pass per batch.

4.3. Module Memory Predictor

Given a module insertion strategy $\boldsymbol{s}$, we predict its peak fine-tuning memory $M(\boldsymbol{s})$. We take a **block-level** approach that accurately estimates **activation memory** in Transformers and thus the overall peak memory during fine-tuning.

4.3.1. Peak Memory Decomposition

Fine-tuning memory peaks at the end of forward pass [42] and decomposes as: $M(\boldsymbol{s}) = M_{con} + M_w(\boldsymbol{s}) + M_x(\boldsymbol{s})$. Here, M_{con} is the memory of frozen backbone parameters and runtime overhead, independent of $\boldsymbol{s}$ and profiled offline. $M_w(\boldsymbol{s})$ is the memory from inserted LoRA modules (parameters, gradients, and optimizer states), which scales with the parameter size of inserted modules and can be directly estimated (Appendix A.2). $M_x(\boldsymbol{s})$ accounts for cached activations during backpropagation and is difficult to model.

To estimate $M_x(\boldsymbol{s})$, we enumerate candidate cached activations by tracing a dummy forward through the backbone and decoder (encoder excluded since LoRA modules are rarely inserted there), yielding an indexed set $\mathcal{X}$. For each activation $x \in \mathcal{X}$, we record its memory footprint in $\boldsymbol{m}_x$, which scales with batch size and sequence length. The activation memory for strategy $\boldsymbol{s}$ is then:

$$M_x(\boldsymbol{s}) = g_x(\boldsymbol{s})^\top \boldsymbol{m}_x \tag{5}$$

where $g_x(\boldsymbol{s}) \in \{0,1\}^{|\mathcal{X}|}$ indicates which activations must be retained for backpropagation. Obtaining $g_x(\boldsymbol{s})$ for Transformers requires **block-level** analysis, as explained below.

4.3.2. Block-Level Activation Dependency Analysis

For each candidate insertion location k, we derive a binary mask $g_x(k) \in \{0,1\}^{|\mathcal{X}|}$ that specifies which activations in $\mathcal{X}$ must be cached to compute gradients for that module. We obtain $g_x(k)$ by tracing the backward pass and separating **intra-block** and **inter-block** dependencies.

Intra-Block Dependencies. Within an attention block, Q, K, and V are parallel branches whose activations interact in the attention operator, creating cross-branch dependencies in the backward pass.

Consider inserting a module w_q at W_q (Fig. 3a). The forward pass is $Q = x_{\text{in}}(W_q + w_q)$, and the block output is $x^2_{\text{out}} = FFN(x^1_{\text{out}})$, where $x^1_{\text{out}} = W_o \cdot \text{Attn}(Q,K,V) + x_{\text{in}}$, $\text{Attn}(Q,K,V) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$ and W_o is the output projection. To compute $\nabla_{w_q} L$, gradients flow from $\nabla_{x^2_{\text{out}}} L$ to $\nabla_{x^1_{\text{out}}} L$ through FFN, then to $\nabla_{\text{Attn}(Q,K,V)} L$ through W_o, and to $\nabla_Q L$ via the attention operator, which depends on the stored activations K and V. Finally, $\nabla_{w_q} L$ needs the input activation x_{in}. Thus, insertion location k at W_q should set $g_x(k)$ to 1 for entries correponding to K, V, x_{in} for W_q.

We analyze the backward dependency for moduel insertion at all the six location types within an attention block (W_q, W_k, W_v, W_o, W^1_{FFN}, and W^2_{FFN}). The correponding configurations for $g_x(k)$ are deferred to Appendix A.3.

Inter-Block Dependencies. To propagate gradients from the loss L back to a module in block r, we must preserve activations along the backward path in all **subsequent** blocks $r+1$ up to the decoder. These include Q, K, V and attention weights $\text{AW}(Q,K) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)$ in downstream blocks (Fig. 3b). In contrast, the activations of **preceding blocks** $(1, \ldots, r-1)$ can be discarded after the forward pass. Thus, we set to 1 all entries in $g_x(k)$ that correspond to the required backward-path activations in blocks $r+1$ through the decoder.

Efficient Prediction. Given $\{g_x(k)\}$, the activations for strategy $\boldsymbol{s}$ are the union over selected locations $g_x(\boldsymbol{s}) = \bigvee_{k:s_k=1} g_x(k)$, with element-wise OR $\vee$. For a given model, $\{g_x(k)\}$ and $\boldsymbol{m}_x$ are profiled once by tracing the computation graph and analyzing block-level dependencies. Then memory prediction for strategy $\boldsymbol{s}$ reduces to inexpensive OR and dot-product operations plus $M_w(\boldsymbol{s})$, enabling fast evaluation for DP-based scheduler.

4.4. Module Insertion Scheduler

We adopt a DP-based scheduler to search the module insertion strategy $\boldsymbol{s}$ for Eq.(1). Each entry $s^k \in \mathcal{R} = \{0, 2^0, \ldots, 2^6\}$ is the LoRA rank r at location k ($r = 0$ means no insertion). We cap the rank at $2^6 = 64$, as prior

work [5] shows diminishing returns beyond. Our goal is to maximize total importance under the memory budget M_{B}.

The problem in Eq.(1) is a knapsack variant. The incremental memory cost ΔM of inserting at location k depends on the nearest previously inserted location k_s, which breaks the standard assumption of independent items. Thus, we explicitly encode the last insertion location in the DP state.

Let $P[k][M_b]$ be the maximum cumulative importance of any strategy whose rightmost insertion is at k ($k = 0$ means no insertion yet), under total memory cost $\leq M_b$. The memory budget is discretized into U bins of width M_{B}/U. Empirically, we set $U = 500$ to balance search accuracy and overhead. We initialize $P[0][M_b] = 0$ for all feasible M_b.

For each state, we consider inserting a rank $r > 0$ at k and choosing a predecessor $k_s < k$:

$$P[k][M_b] \;=\; \max_{\substack{r\in\mathcal{R},\, r>0\\ 0\le k_s<k}} \Big(P[k_s][M_b-\Delta M] \;+\; h(r)\,\lambda^{k}_{n+1}\Big), \tag{6}$$

where λ^{k}_{n+1} is the predicted importance at k, and $h(r) = \sqrt{\ln(r)+1}/\sqrt{\ln(\max(\mathcal{R})+1)}$ normalizes the contribution of rank r (chosen empirically; see Appendix C.2). $\Delta M = M(\boldsymbol{s}_k) - M(\boldsymbol{s}_{k_s})$ is the extra cost to insert a rank-r module at k given predecessor k_s. Transitions with $M_b - \Delta M < 0$ are skipped. A location is skipped if its state is never used as a predecessor. After filling the DP table, the optimal solution is $\max_{0\le k\le|\mathcal{K}|} P[k][M_{\mathrm{B}}]$, and the corresponding insertion strategy is retrieved via backtracking. Pseudocode is provided in Appendix A.4.

5. Experiments

5.1. Experimental Setups

Datasets. We test three modalities: image, natural language processing (NLP), and vision-language (VL). For image, we use seven representative VTAB [76] datasets: CIFAR100 [40], Caltech101 [21], DTD [12], Camelyon [7], EuroSAT [23], Clevr-Count [37], and DMLab [6]. We also include four larger, more challenging datasets following [48]: Places365 [81], iNaturalist2018 [58], iNaturalist2021 [59], and ImageNet [14]. For NLP, we use four representative GLUE [63] tasks: MNLI [67], QQP [13], MRPC [17], and QNLI [63]. For VL, following [68], we evaluate on VQA (test-std) [3], SNLI-VE (test) [69], RefCOCO+ (val) [46], and COCO (image captioning) [43].

Models. We use the 1.5B-parameter vision-language model Janus-Pro [10] as our pretrained foundation model. To support audio modalities, we extend Janus-Pro with Whisper-tiny [52] as the audio-text encoder and Kokoro-82M [1] as the text-audio decoder. We also test other backbones, *e.g.* the 180M and 470M variants of OFA [64] in Appendix C.3.

Metrics. For image and NLP tasks, we report top-1 accuracy (%). For VL tasks, we report accuracy (%) on VQA (test-std), SNLI-VE (test), and RefCOCO+ (val). For COCO, we report BLEU@4 [50] and CIDEr [60], denoted as COCO(B@4) and COCO(C), respectively.

Wart-start Setup. Given a foundation model already fine-tuned on several base tasks via task-specific modules, we adapt it to a new task [68]. Results following the cold-start VTAB setup [76] is in Appendix C.4.

Compared Methods. We compare TaskIT against three groups of baselines: **(i) Zero-FT**: Prompt Tuning [35], AdapterSoup [11], AdapterFusion [51], and LoraHub [30]; **(ii) Non-LoRA-based PEFT**: Full tuning [33], Linear tuning [33], BitFit [75], RS-Bypass [34], Adapter [28], AdaptFormer [9], FacT-TK [36], RS [34], ConvPass [35], LISA [49], LoSA [48], HST [44], and UniPT [15]; **(iii) LoRA-based PEFT**: LoRA ($r = 8$) [29], NOAH [80], π-Adapter ($r = 8$) [68], AdaLoRA [78], and AutoLoRA [79]. Details of all baselines are in Appendix B.1. All other experimental setups are in Appendix B.2.

5.2. Main Results

5.2.1. Performance on Cross-modal Tasks

Settings. We evaluate how different fine-tuning methods adapt a multi-LoRA LLM to **cross-modal** downstream tasks under a memory budget. We start from a model with LoRAs for five base tasks: image classification (Pascal VOC [18]), question answering (SQuAD [53]), image captioning (Flickr8k [25]), text-to-image (CUB-200 [62]), and visual question answering (GQA [32]). These tasks span uni-modal and multi-modal understanding, generation, and reasoning, providing a realistic starting point for on-device LLMs [74]. We then fine-tune the model on new tasks across image, NLP, and VL modalities under a memory budget M_B of 8 GB, roughly matching modern mobile devices [65].

Results. From Tab. 1, TaskIT achieves the best accuracy-memory trade-off on cross-modal tasks. *(i)* Compared to the best **Zero-FT** baseline, AdapterFusion, TaskIT improves average accuracy by 10.4%. Zero-FT methods adapt to a new task by combining existing task-specific modules without training new parameters, and struggle when the new task is weakly related to learned tasks, especially for image tasks. Moreover, although AdapterFusion does not train parameters, it still needs 10.6 GB of memory to compute combination weights. *(ii)* Compared to the best **non-LoRA** baseline, UniPT, TaskIT reduces average memory usage by 13.8% while improving accuracy by 1.5%. UniPT uniformly allocates memory across modules without importance modeling, wasting budget on non-critical components, and thus needs a 9.4× larger trainable subnetwork to approach TaskIT's accuracy. *(iii)* Compared to the best **LoRA-based** baseline, AutoLoRA, TaskIT saves 45.5% memory with only a 0.3% accuracy drop. AutoLoRA re-

		Image													NLP						VL						
	Param(10^6)	Cifar100	Caltech101	DTD	Camelyon	EuroSAT	Clevr-Count	DMLab	Places365	iNaturalist2018	iNaturalist2021	ImageNet	Image Avg Acc.	Memory(GB)	MNLI	QQP	MRPC	QNLI	NLP Avg Acc.	Memory(GB)	VQA(test-std)	SNLI-VE(test)	RefCOCO+(val)	COCO(B@4)	COCO(C)	VL Avg Acc.	Memory(GB)
Zero-FT																											
Prompt Tuning[35]	0.0	73.7	86.1	59.6	78.1	90.7	65.0	41.2	53.2	72.5	76.9	81.9	70.8	11.7	80.6	84.8	84.1	87.5	84.3	11.6	66.9	77.4	69.4	36.0	134.9	76.9	11.7
AdapterSoup[11]	0.0	54.5	73.9	52.5	67.6	84.5	37.9	36.2	45.6	62.8	63.3	72.9	59.3	0.0	75.8	81.5	81.8	81.3	80.1	0.0	62.4	73.5	64.6	37.5	130.4	73.7	0.0
AdapterFusion[51]	11.1	61.2	77.6	54.2	74.7	83.7	79.1	44.0	45.6	65.5	69.0	73.3	66.2	10.6	78.3	79.2	78.2	87.7	80.9	10.5	67.2	76.7	65.9	38.2	131.1	75.8	10.6
LoraHub[30]	0.0	53.2	76.4	67.5	75.1	87.7	66.2	29.6	52.3	68.2	57.3	69.2	63.9	6.4	74.1	83.2	81.7	84.0	80.8	6.3	64.3	77.1	65.5	36.1	123.7	73.3	6.5
Non-LoRA-based PEFT																											
Full tuning[33]	1720.5	72.7	87.2	64.1	81.4	95.6	75.5	43.5	56.5	76.8	81.9	87.2	74.8	16.5	86.5	88.1	88.6	92.3	88.9	16.1	**76.5**	**86.1**	**80.4**	**42.2**	**142.7**	**85.6**	16.6
Linear tuning[33]	0.0	64.4	79.3	58.2	76.8	82.7	68.3	39.8	55.2	67.7	78.7	82.4	68.5	6.8	81.2	84.0	83.9	86.9	84.0	6.6	68.2	78.7	70.3	36.8	135.4	77.9	6.8
BitFit[75]	1.2	71.6	82.9	59.1	77.9	91.1	61.4	33.4	60.6	77.1	75.4	86.3	70.6	8.1	81.7	84.9	85.7	88.5	85.2	7.9	72.1	82.0	73.9	39.5	139.3	81.4	8.1
RS-Bypass[34]	35.8	61.8	86.7	74.6	82.5	94.0	70.5	38.2	57.4	71.5	67.3	78.3	71.2	8.9	85.8	89.3	84.5	90.9	87.6	8.6	72.5	81.0	72.2	38.3	140.3	80.9	9.0
Adapter[28]	13.1	71.7	90.9	64.7	81.1	94.0	78.0	51.1	56.6	72.1	79.8	83.1	74.8	9.9	85.2	89.3	87.9	92.7	88.8	9.6	73.0	83.6	75.2	41.9	140.7	82.9	10.1
AdaptFormer[9]	13.1	69.2	89.1	71.2	82.9	94.5	82.3	48.8	56.6	78.1	83.2	86.4	76.6	9.5	85.9	90.6	89.1	92.7	89.6	9.2	72.7	84.0	74.9	40.7	142.1	82.9	9.8
FacT-TK[36]	4.9	74.8	88.9	70.5	84.0	93.8	84.0	48.3	55.6	73.8	72.8	85.1	75.6	10.1	85.8	90.4	86.1	92.4	88.7	9.8	73.9	83.7	75.5	41.3	142.2	83.3	10.1
RS[34]	43.1	77.7	91.1	71.1	85.5	95.5	81.5	50.9	57.4	74.3	80.8	85.4	77.4	12.3	86.1	89.6	88.5	92.9	89.3	11.8	73.1	84.3	75.4	41.0	141.6	83.1	12.3
Convpass[35]	26.5	67.1	90.8	70.3	82.4	96.5	82.9	51.3	**62.2**	77.6	79.1	84.7	76.8	9.7	73.2	79.6	74.8	82.7	77.6	9.4	70.6	83.3	74.7	39.5	138.9	81.4	9.8
LISA[49]	170.6	69.8	90.6	72.8	85.7	96.1	82.7	49.5	59.8	78.1	84.7	85.9	77.8	10.0	83.8	88.2	86.2	91.3	87.4	9.6	74.9	83.1	73.6	40.1	140.1	82.4	10.1
LoSA[48]	9.7	78.7	90.2	71.9	**86.7**	93.9	77.1	47.2	58.6	79.9	81.6	85.0	77.3	9.3	83.1	88.1	88.9	91.2	87.8	9.1	71.7	82.2	73.9	39.3	140.1	81.4	9.3
HST[44]	66.9	78.3	91.4	74.9	84.9	96.1	81.6	**53.2**	57.7	74.3	77.5	87.5	78.0	9.6	85.0	88.3	89.6	93.1	89.0	9.2	73.5	82.4	75.0	40.3	139.5	82.1	9.6
UniPT[15]	157.8	78.1	89.0	71.5	85.1	94.4	81.6	48.8	60.2	80.9	84.8	88.6	78.4	8.8	85.5	89.1	87.7	91.8	88.5	8.5	73.8	83.7	74.9	39.9	141.3	82.7	8.8
LoRA-based PEFT																											
LoRA[29]	31.0	67.9	88.0	70.9	84.5	92.0	81.2	47.7	60.2	80.7	82.7	87.6	76.7	10.3	86.3	90.1	87.6	92.0	89.0	10.1	73.4	82.7	73.8	39.6	140.4	82.0	10.2
NOAH[80]	42.3	72.6	92.4	73.2	83.2	96.2	83.1	51.9	59.6	79.2	84.7	88.8	78.6	13.2	85.9	89.4	89.8	92.1	89.3	12.6	74.5	84.2	75.9	40.6	141.9	83.4	13.3
π-Adapter[68]	228.0	74.5	91.3	72.9	85.4	93.0	82.5	50.5	60.6	80.5	**86.0**	**89.4**	78.8	12.8	85.8	90.2	89.5	93.2	89.7	12.3	74.2	85.2	73.2	39.5	140.0	82.4	12.9
AdaLoRA[78]	29.1	82.7	94.3	75.4	85.1	94.3	84.8	52.7	60.6	81.2	85.2	89.2	80.5	12.3	**86.6**	90.7	89.6	93.4	90.1	12.1	73.4	84.7	77.2	41.6	142.1	83.8	12.4
AutoLoRA[79]	29.1	83.3	**94.4**	75.8	86.3	94.1	**84.9**	**53.2**	59.4	81.5	85.3	88.9	**80.7**	14.3	86.5	**91.4**	90.3	92.8	**90.3**	13.7	74.5	83.2	78.6	41.8	142.2	84.1	14.5
TaskIT	16.8	**83.4**	92.1	**76.7**	86.5	**96.6**	80.7	50.6	60.8	**81.9**	85.4	89.3	80.4	7.8	86.3	89.6	**90.6**	**93.7**	90.1	7.9	75.1	85.3	75.2	39.8	142.4	83.6	7.8

Table 1. Performance of Zero-FT, non-LoRA, and LoRA-based fine-tuning for cross-modal tasks (Image, NLP, and VL). Parameters refers to the number of learnable parameters excluding the final classification layer. Best results are **bolded**, and second-best underlined.

lies on a loss-driven meta-learner for importance evaluation and rank allocation, but its bi-level optimization incurs substantial memory overhead [79].

5.2.2. Performance on Uni-modal Tasks

Settings. We compare the most accurate fine-tuning method per category in the cross-modal setting and evaluate how they adapt a multi-LoRA LLM to **uni-modal (image)** downstream tasks. Following the VTAB benchmark [76], we group tasks into three pools: Natural, Specialized, and Structured. For each pool, we pick one task as the new task, while the remaining tasks serve as base tasks for the foundation model, following the warm-start protocol in [68].

Results. From Tab. 2, TaskIT again achieves the best accuracy-memory trade-off. *(i)* Due to the higher similarity among uni-modal tasks, the Zero-FT baseline AdapterFusion performs better than in the cross-modal setting, with an average increase in accuracy by 2.5%. However, its accuracy remains 6.8% lower than TaskIT, indicating that parameter fine-tuning is still necessary even when adapting to tasks within the same modality as the base tasks. *(ii)* Compared to the non-LoRA baseline UniPT, TaskIT improves average accuracy by 2.1% while reducing memory usage by 11.4%. *(iii)* The accuracy gap to the best LoRA-based baseline, AutoLoRA, is reduced to only 0.1%. This is because TaskIT can exploit the higher similarity between uni-modal tasks for cross-task importance transfer, achieving accuracy comparable to AutoLoRA's meta-learning-based importance estimation but at much lower memory cost.

5.3. Ablation Study

Settings. We assess the contribution of each key component in TaskIT. The set of learned (base) tasks is identical to that in § 5.2.1, and the new tasks are the same image tasks in § 5.2.1. For each ablation, we modify a single component while keeping all other parts of TaskIT unchanged.

Results. From Tab. 3, replacing the **task similarity assessment** (S) (§ 4.2.2) with uniform task similarities reduces accuracy by 5.1%, while replacing the entire **module importance predictor** (I)(§ 4.2) with uniform module importances reduces accuracy by 6.3%. This highlights the necessity of task similarity estimation and module importance prediction. Substituting the **module memory predictor** (M) (§ 4.3) with a uniform memory model increases peak memory from 7.8 GB to 9.9 GB, showing that our block-based profiler is essential for staying within a tight mem-

	Natural Task Pool							Specialized Task Pool				Structured Task Pool											
	Cifar100	Caltech101	DTD	Flowers102	Pets	SVHN	Sun397	Camelyon	EuroSAT	Resisc45	Retinopathy	Clevr-Count	Clevr-Dist	DMLab	KITTI-Dist	dSpr-Loc	dSpr-Ori	sNORB-Azim	sNORB-Ele	Nat. Avg Acc.	Spec. Avg Acc.	Struc. Avg Acc.	Memory(GB)
AdapterFusion[51]	67.6	83.8	70.3	98.2	88.1	82.2	55.8	69.4	78.6	84.4	75.9	79.0	70.6	43.5	83.7	80.6	52.4	42.5	49.1	78.1	77.1	62.7	10.6
UniPT[15]	78.1	89.0	71.5	99.1	89.7	84.1	56.3	85.1	94.4	86.8	76.3	81.6	66.4	48.8	81.6	90.2	58.2	43.5	50.8	81.1	85.7	65.1	8.8
AutoLoRA[79]	**83.3**	**94.4**	**75.8**	**99.7**	90.8	**86.3**	56.5	86.3	94.1	**89.6**	**79.2**	83.5	66.7	53.2	**82.7**	**92.1**	**61.3**	**48.8**	51.7	**83.8**	**87.3**	**67.5**	14.3
TaskIT	79.6	93.5	75.7	**99.7**	**91.4**	86.1	**59.7**	**87.2**	**94.6**	89.3	77.6	**86.3**	**69.9**	**54.5**	81.4	90.8	58.1	44.5	**52.7**	83.7	87.2	67.3	7.8

Table 2. Performance on Image tasks. Only the most accurate Zero-FT, non-LoRA, and LoRA-based baselines in the cross-modal setting are included. Best results are **bolded**, and second-best underlined.

Method	Learned Param (M)	Memory usage (GB)	Accuracy (%)
w/o S	16.8	7.6	75.3
w/o I	16.8	7.6	74.1
w/o M	17.4	9.9	80.6
w/o DP	9.1	7.9	77.5
TaskIT	16.8	7.8	80.4

Table 3. Ablation study.

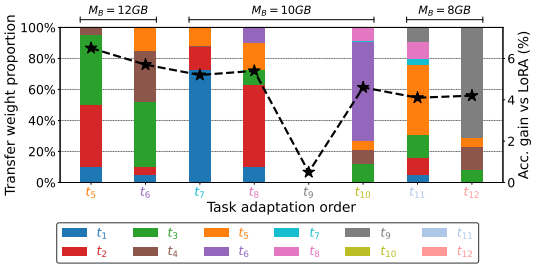

Figure 4. Continual task learning evaluation. The model learns new tasks t_5-t_{12} sequentially. The stacked bar chart (left y-axis) shows the similarity-weighted proportion of importance transferred from existing tasks for each new task. The line plot (right y-axis) shows the accuracy gain of TaskIT over vanilla LoRA.

ory budget M_B while preserving accuracy. Finally, replacing the **DP-based scheduler** (DP) (§ 4.4) with a greedy algorithm [61] reduces the number of learned parameters by 7.7M and drops accuracy to 77.5%. The greedy strategy underutilizes the available memory, whereas the DP scheduler more effectively searches the configuration space to maximize total importance under memory constraints.

5.4. Case Study

Settings. We conduct a **continual task learning** case study to mimic realistic mobile deployment (Fig. 4). We select tasks with high on-device demand [71] and gradually reduce the memory budget M_B from 12 GB to 10 GB and then to 8 GB. The model is initially equipped with LoRAs for four learned tasks: t_1 image classification (Pascal VOC [18]), t_2 object detection (COCO [43]), t_3 question answering (SQuAD [53]), and t_4 text summarization (CNN/Daily Mail [24]). It then sequentially learns t_5 visual question answering (GQA [32]), t_6 natural language inference (MNLI [67]), t_7 image classification (iNaturalist [58]), t_8 referring expression comprehension (RefCOCO [72]), t_9 audio captioning (AudioCaps [38]), t_{10} credit score classification (German Credit [26]), t_{11} video question answering (MSVD-QA [70]), and t_{12} text-to-audio (WavCaps [47]).

Results. In Fig. 4, the bar chart shows, for each new task, the proportion of transferred importance from each previously learned task, and the line plot shows TaskIT 's accuracy gain over vanilla LoRA [29]. We highlight three observations. *(i)* TaskIT consistently outperforms vanilla LoRA over the task sequence, though its average gain decreases as M_B shrinks from 12 GB to 8 GB, reflecting tighter memory constraints. *(ii)* Transfer weights adapt to each new task, and newly learned tasks quickly serve as sources. For example, after learning t_9 (audio captioning), over 70% of the transferred importance for t_{12} (text-to-audio) comes from t_9, showing that TaskIT effectively exploits uni-modal audio knowledge once available. *(iii)* When no sufficiently similar tasks exist, TaskIT falls back to non-transfer. For example, t_9 is a new audio modality whose similarity to all existing tasks t_1-t_8 is below 0.35. Then TaskIT applies uniform module importance and relies on the memory predictor and DP scheduler to stay within budget.

6. Conclusion

This paper presents TaskIT, a framework for memory-efficient fine-tuning of multi-LoRA LLMs. TaskIT addresses two key challenges in sparse LoRA updating. It predicts module importance before insertion via cross-task importance transfer, and it accurately models activation memory in Transformers with a block-based predictor. A DP-based scheduler then selects the locations, numbers, and ranks of LoRA modules to maximize fine-tuning accuracy under a given memory budget. Experiments on uni-modal and cross-modal benchmarks show that TaskIT consistently achieves better accuracy-memory tradeoffs than Zero-FT, non-LoRA, and LoRA-based fine-tuning baselines, highlighting its promise for scaling on-device foundation models to many evolving tasks.

Acknowledgments

This research is partially supported by the RGC of Hong Kong SAR, China (Project No. CityU 11206425).

References

[1] Kokoro. https://huggingface.co/hexgrad/Kokoro-82M. 6

[2] Get started with foundation models adapter training. https://developer.apple.com/apple-intelligence/foundation-models-adapter/. 1

[3] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433, 2015. 6

[4] Arash Ardakani, Altan Haan, Shangyin Tan, Doru Thom Popovici, Alvin Cheung, Costin Iancu, and Koushik Sen. Slimfit: Memory-efficient fine-tuning of transformer-based models using training dynamics. In *Proceedings of the ACL*, pages 6218–6236, 2024. 1, 2, 4

[5] Klaudia Bałazy, Mohammadreza Banaei, Karl Aberer, and Jacek Tabor. Lora-xs: Low-rank adaptation with extremely small number of parameters. *arXiv preprint arXiv:2405.17604*, 2024. 1, 6

[6] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016. 6

[7] Babak Ehteshami Bejnordi, Mitko Veta, Paul Johannes Van Diest, Bram Van Ginneken, Nico Karssemeijer, Geert Litjens, Jeroen AWM Van Der Laak, Meyke Hermsen, Quirine F Manson, Maschenka Balkenhol, et al. Diagnostic assessment of deep learning algorithms for detection of lymph node metastases in women with breast cancer. *Jama*, 318(22):2199–2210, 2017. 6

[8] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. Tinytl: Reduce memory, not parameters for efficient on-device learning. *Advances in Neural Information Processing Systems*, 33:11285–11297, 2020. 3

[9] Shoufa Chen, Chongjian Ge, Zhan Tong, Jiangliu Wang, Yibing Song, Jue Wang, and Ping Luo. Adaptformer: Adapting vision transformers for scalable visual recognition. *Advances in Neural Information Processing Systems*, 35:16664–16678, 2022. 2, 6, 7

[10] Xiaokang Chen, Zhiyu Wu, Xingchao Liu, Zizheng Pan, Wen Liu, Zhenda Xie, Xingkai Yu, and Chong Ruan. Janus-pro: Unified multimodal understanding and generation with data and model scaling. *arXiv preprint arXiv:2501.17811*, 2025. 6

[11] Alexandra Chronopoulou, Matthew E Peters, Alexander Fraser, and Jesse Dodge. Adaptersoup: Weight averaging to improve generalization of pretrained language models. *arXiv preprint arXiv:2302.07027*, 2023. 2, 6, 7

[12] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3606–3613, 2014. 6

[13] Kornél Csernai, Nikhil Dandekar, and Shankar Iyer. First Quora Dataset Release: Question Pairs. Kaggle Competition and Data @ Quora Blog, 2017. 6

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 6

[15] Haiwen Diao, Bo Wan, Ying Zhang, Xu Jia, Huchuan Lu, and Long Chen. Unipt: Universal parallel tuning for transfer learning with efficient parameter and memory. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 28729–28740, 2024. 2, 6, 7, 8

[16] Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. Sparse low-rank adaptation of pre-trained language models. In *Proceedings of the ACL*, 2023. 2

[17] William B Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the third international workshop on paraphrasing (IWP2005)*, 2005. 6

[18] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010. 2, 6, 8

[19] Siqi Fan, Xin Jiang, Xiang Li, Xuying Meng, Peng Han, Shuo Shang, Aixin Sun, Yequan Wang, and Zhongyuan Wang. Not all layers of llms are necessary during inference. In *IJCAI*, 2025. 1

[20] Cheng Fang, Sicong Liu, Zimu Zhou, Bin Guo, Jiaqi Tang, Ke Ma, and Zhiwen Yu. Adashadow: Responsive test-time model adaptation in non-stationary mobile environments. In *Proceedings of the 22nd ACM Conference on Embedded Networked Sensor Systems*, pages 295–308, 2024. 2, 4

[21] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *2004 conference on computer vision and pattern recognition workshop*, pages 178–178. IEEE, 2004. 6

[22] Yunhui Guo, Honghui Shi, Abhishek Kumar, Kristen Grauman, Tajana Rosing, and Rogerio Feris. Spottune: transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4805–4814, 2019. 2

[23] Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226, 2019. 6

[24] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28, 2015. 2, 8

[25] Micah Hodosh, Peter Young, and Julia Hockenmaier. Framing image description as a ranking task: Data, models and evaluation metrics. *Journal of Artificial Intelligence Research*, 47:853–899, 2013. 2, 6

[26] Hans Hofmann. Statlog (German Credit Data), 1994. 8

[27] Junyuan Hong, Lingjuan Lyu, Jiayu Zhou, and Michael Spranger. Mecta: Memory-economic continual test-time model adaptation. In *Proceedings of the ICLR*, 2023. 1, 2, 3

[28] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *Proceedings of the ICML*, pages 2790–2799. PMLR, 2019. 2, 6, 7

[29] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022. 1, 2, 6, 7, 8

[30] Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. Lorahub: Efficient cross-task generalization via dynamic lora composition. In *Proceedings of the COLM*, 2024. 2, 6, 7

[31] Kai Huang, Boyuan Yang, and Wei Gao. Elastictrainer: Speeding up on-device training with runtime elastic tensor selection. In *Proceedings of the ACM MobiSys*, pages 56–69, 2023. 1, 2, 3, 4

[32] Drew A Hudson and Christopher D Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6700–6709, 2019. 6, 8

[33] Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning. In *European conference on computer vision*, pages 709–727. Springer, 2022. 2, 6, 7

[34] Zeyinzi Jiang, Chaojie Mao, Ziyuan Huang, Ao Ma, Yiliang Lv, Yujun Shen, Deli Zhao, and Jingren Zhou. Res-tuning: A flexible and efficient tuning paradigm via unbinding tuner from backbone. *Advances in Neural Information Processing Systems*, 36:42689–42716, 2023. 2, 6, 7

[35] Shibo Jie and Zhi-Hong Deng. Convolutional bypasses are better vision transformer adapters. *arXiv preprint arXiv:2207.07039*, 2022. 2, 6, 7

[36] Shibo Jie and Zhi-Hong Deng. Fact: Factor-tuning for lightweight adaptation on vision transformer. In *Proceedings of the AAAI conference on artificial intelligence*, pages 1060–1068, 2023. 6, 7

[37] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2901–2910, 2017. 6

[38] Chris Dongjoo Kim, Byeongchang Kim, Hyunmin Lee, and Gunhee Kim. Audiocaps: Generating captions for audios in the wild. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 119–132, 2019. 8

[39] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International conference on machine learning*, pages 3519–3529. PMlR, 2019. 4

[40] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 6

[41] Sheng Li, Geng Yuan, Yue Dai, Youtao Zhang, Yanzhi Wang, and Xulong Tang. Smartfrz: An efficient training framework using attention-based layer freezing. In *Proceedings of the ICLR*, 2024. 2, 3, 4

[42] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. On-device training under 256kb memory. *Advances in Neural Information Processing Systems*, 35:22941–22954, 2022. 1, 2, 3, 5

[43] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 6, 8

[44] Weifeng Lin, Ziheng Wu, Wentao Yang, Mingxin Huang, Jun Huang, and Lianwen Jin. Hierarchical

side-tuning for vision transformers. *arXiv preprint arXiv:2310.05393*, 2023. 2, 6, 7

[45] Yuhan Liu, Saurabh Agarwal, and Shivaram Venkataraman. Autofreeze: Automatically freezing model blocks to accelerate fine-tuning. *arXiv preprint arXiv:2102.01386*, 2021. 2

[46] Junhua Mao, Jonathan Huang, Alexander Toshev, Oana Camburu, Alan L Yuille, and Kevin Murphy. Generation and comprehension of unambiguous object descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 11–20, 2016. 6

[47] Xinhao Mei, Chutong Meng, Haohe Liu, Qiuqiang Kong, Tom Ko, Chengqi Zhao, Mark D Plumbley, Yuexian Zou, and Wenwu Wang. Wavcaps: A chatgpt-assisted weakly-labelled audio captioning dataset for audio-language multimodal research. *arXiv preprint arXiv:2303.17395*, 2023. 8

[48] Otniel-Bogdan Mercea, Alexey Gritsenko, Cordelia Schmid, and Anurag Arnab. Time-memory-and parameter-efficient visual adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5536–5545, 2024. 1, 2, 6, 7

[49] Rui Pan, Xiang Liu, Shizhe Diao, Renjie Pi, Jipeng Zhang, Chi Han, and Tong Zhang. Lisa: Layerwise importance sampling for memory-efficient large language model fine-tuning. *Advances in Neural Information Processing Systems*, 37:57018–57049, 2024. 2, 6, 7

[50] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002. 6

[51] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. Adapterfusion: Non-destructive task composition for transfer learning. 2021. 2, 6, 7, 8

[52] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *Proceedings of the ICML*, pages 28492–28518. PMLR, 2023. 6

[53] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016. 2, 6, 8

[54] Tianyi Shen, Chonghan Lee, and Vijaykrishnan Narayanan. Simfreeze: Adaptively freeze vision transformer encoders with token similarity. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8266–8270, 2024. 2

[55] Zhiqiang Shen, Zechun Liu, Jie Qin, Marios Savvides, and Kwang-Ting Cheng. Partial is better than all: Revisiting fine-tuning strategy for few-shot learning. In *Proceedings of the AAAI conference on artificial intelligence*, pages 9594–9602, 2021. 2, 4

[56] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. In *Proceedings of the ICLR*, 2024. 4

[57] Yi-Lin Sung, Varun Nair, and Colin A Raffel. Training neural networks with fixed sparse masks. *Advances in Neural Information Processing Systems*, 34:24193–24205, 2021. 2

[58] Grant Van Horn, Oisin Mac Aodha, Yang Song, Yin Cui, Chen Sun, Alex Shepard, Hartwig Adam, Pietro Perona, and Serge Belongie. The inaturalist species classification and detection dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8769–8778, 2018. 6, 8

[59] Grant Van Horn, Elijah Cole, Sara Beery, Kimberly Wilber, Serge Belongie, and Oisin Mac Aodha. Benchmarking representation learning for natural world image collections. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12884–12893, 2021. 6

[60] Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4566–4575, 2015. 6

[61] Andrew Vince. A framework for the greedy algorithm. *Discrete Applied Mathematics*, 121(1-3):247–260, 2002. 8

[62] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011. 2, 6

[63] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP workshop BlackboxNLP: Analyzing and interpreting neural networks for NLP*, pages 353–355, 2018. 6

[64] Peng Wang, An Yang, Rui Men, Junyang Lin, Shuai Bai, Zhikang Li, Jianxin Ma, Chang Zhou, Jingren Zhou, and Hongxia Yang. Ofa: Unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework. In *International conference on machine learning*, pages 23318–23340. PMLR, 2022. 6

[65] Qipeng Wang, Mengwei Xu, Chao Jin, Xinran Dong, Jinliang Yuan, Xin Jin, Gang Huang, Yunxin Liu, and Xuanzhe Liu. Melon: Breaking the memory wall for resource-efficient on-device machine learning. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, pages 450–463, 2022. 6

[66] Yiding Wang, Decang Sun, Kai Chen, Fan Lai, and Mosharaf Chowdhury. Egeria: Efficient dnn training with knowledge-guided layer freezing. In *Proceedings of the eighteenth European conference on computer systems*, pages 851–866, 2023. 2

[67] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long papers)*, pages 1112–1122, 2018. 6, 8

[68] Chengyue Wu, Teng Wang, Yixiao Ge, Zeyu Lu, Ruisong Zhou, Ying Shan, and Ping Luo. π-tuning: Transferring multimodal foundation models with optimal multi-task interpolation. In *Proceedings of the ICML*, 2023. 2, 6, 7

[69] Ning Xie, Farley Lai, Derek Doran, and Asim Kadav. Visual entailment task for visually-grounded language learning. *arXiv preprint arXiv:1811.10582*, 2018. 6

[70] Dejing Xu, Zhou Zhao, Jun Xiao, Fei Wu, Hanwang Zhang, Xiangnan He, and Yueting Zhuang. Video question answering via gradually refined attention over appearance and motion. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 1645–1653, 2017. 8

[71] Mengwei Xu, Wangsong Yin, Dongqi Cai, Rongjie Yi, Daliang Xu, Qipeng Wang, Bingyang Wu, Yihao Zhao, Chen Yang, Shihe Wang, et al. A survey of resource-efficient llm and multimodal foundation models. *arXiv preprint arXiv:2401.08092*, 2024. 8

[72] Licheng Yu, Patrick Poirson, Shan Yang, Alexander C Berg, and Tamara L Berg. Modeling context in referring expressions. In *European conference on computer vision*, pages 69–85. Springer, 2016. 8

[73] Zhuoran Yu and Yong Jae Lee. How multimodal llms solve image tasks: A lens on visual grounding, task reasoning, and answer decoding. 2025. 1

[74] Jinliang Yuan, Chen Yang, Dongqi Cai, Shihe Wang, Xin Yuan, Zeling Zhang, Xiang Li, Dingge Zhang, Hanzi Mei, Xianqing Jia, et al. Mobile foundation model as firmware. In *Proceedings of the ACM MobiCom*, pages 279–295, 2024. 1, 6

[75] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the ACL*, 2022. 2, 6, 7

[76] Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, et al. A large-scale study of representation learning with the visual task adaptation benchmark. *arXiv preprint arXiv:1910.04867*, 2019. 6, 7

[77] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12104–12113, 2022. 1

[78] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. In *Proceedings of the ICLR*, 2023. 1, 2, 6, 7

[79] Ruiyi Zhang, Rushi Qiang, Sai Ashish Somayajula, and Pengtao Xie. Autolora: Automatically tuning matrix ranks in low-rank adaptation based on meta learning. In *Proceedings of the ACL*, 2024. 2, 6, 7, 8

[80] Yuanhan Zhang, Kaiyang Zhou, and Ziwei Liu. Neural prompt search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024. 2, 6, 7

[81] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 40(6): 1452–1464, 2017. 6