Efficient Quantization-Aware Adaptation for Visual Foundation Models

Yinglong Li[1], Xiaoyu Liu[1], Yutong Liu[1], Yueyi Zhang[2], and Zhiwei Xiong[1(✉)]

[1] University of Science and Technology of China, China
yllee@mail.ustc.edu.cn, zwxiong@ustc.edu.cn
[2] Midea Group, China

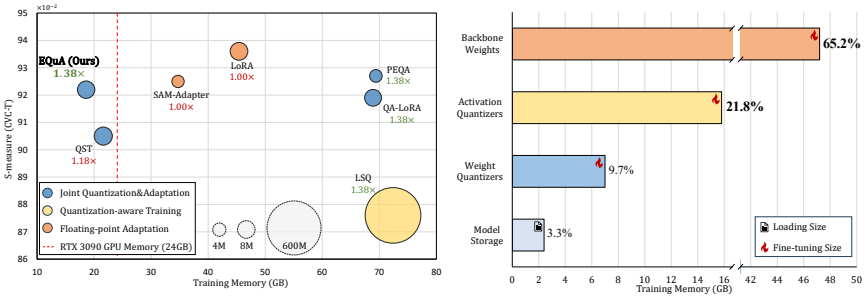
Abstract. Efficient strategies to jointly adapt and deploy large language models have seen a growing need under resource-limited conditions for downstream applications. However, when applied to visual foundation models, existing methods typically incur either high GPU memory consumption during adaptation or extra computation costs introduced by the adapters at deployment. In this paper, we propose **E**fficient **Qu**antization-aware **A**daptation (EQuA) that achieves high efficiency in both adaptation and deployment for visual foundation models. We observe that dominant memory consumption arises from intermediate activations cached for backpropagation in the deep backbone and activation quantizers. To address this issue, we split a lightweight sub-network from the backbone during adaptation as a side adapter branch, and tailor two adaptation strategies to eliminate these cached activations, thereby significantly reducing memory consumption. At deployment, the side adapter branch is merged back into the backbone, yielding a quantized model without any extra computation costs. Extensive experiments on representative visual foundation models and diverse downstream tasks exhibit that EQuA achieves an elegant trade-off between performance and efficiency. For example, EQuA yields over 70% GPU memory reduction compared to state-of-the-art baselines while maintaining competitive performance. Code: https://github.com/leenas233/EQuA.

Keywords: Quantization · Model Adaptation · Visual Foundation Model

1 Introduction

Foundation models have recently exhibited impressive performance across diverse tasks. Pre-trained on massive datasets, these models can be effectively adapted, *i.e.*, fine-tuned, for diverse downstream applications. Thanks to the strong generalization ability of foundation models, there is a growing need for an efficient solution that enables their adaptation and quantization under resource-limited conditions for downstream deployment.

In the field of Large Language Models (LLMs) [32], some recent methods [18, 39] leverage parameter-efficient fine-tuning (PEFT) techniques to enable joint quantization and adaptation with minimal trainable parameters, avoiding the



(a) Performance-efficiency trade-off **(b)** Training memory components on SAM-H

Fig. 1: (a) Performance-efficiency trade-off for quantizing and adapting SAM-H (2.4 GB) on the CVC-T dataset [34] under the 4-bit setting. The bubble size represents number of trainable parameters. The number below the method name denotes inference speedup compared to the original floating-point model. (b) The analysis is conducted on performing quantization-aware training for SAM-H with a batch size of 2. *Fine-tuning Size* represents memory consumption caused by cached intermediate tensors when fine-tuning weight quantizers, activation quantizers, or backbone weights. *Loading Size* represents the memory consumption occupied by loading the model on GPU. Total memory consumption is 72.4 GB.

overhead of full-parameter fine-tuning. These methods typically quantize massive pre-trained floating-point weights (*e.g.*, 138.0 GB for floating-point LLaMA2-70B [33]) into low-bit integers (*e.g.*, 35.3 GB for 4-bit LLaMA2-70B) to reduce training memory on GPU and then apply adaptation strategies via adapters.

While being effective for LLMs, however, these methods exhibit limited efficiency on visual foundation models. LLMs process token sequences and substantial memory consumption comes from pre-trained floating-point weights. In contrast, visual foundation models typically consume far less GPU memory for floating-point weights compared to LLMs (*e.g.*, 2.4 GB for SAM-H [20] *vs.* 138.0 GB for LLaMA2-70B), but they often process high-resolution images and perform dense prediction tasks, which generate large intermediate activations. As a result, cached activations during backpropagation become the main source of memory costs, further exacerbated by deep backbones and activation quantizers in the backward pass. As shown in Fig. 1a, QA-LoRA [37] fine-tunes only about 1.0% of the parameters in SAM-H [20], but the GPU memory costs during adaptation exceeds 60 GB, making it infeasible to perform joint quantization and adaptation of SAM-H on a single RTX 3090 GPU (24 GB). Although QST [39] reduces memory consumption by blocking gradient flow through the quantized backbone via side adapters, these adapters introduce extra computations and inherit quantization errors from the quantized backbone, limiting both the inference efficiency and performance of the quantized model.

In this work, we aim to design an efficient joint quantization and adaptation method for visual foundation models that reduces training memory consumption during adaptation and incurs no extra computational overhead at deployment.

To this end, we propose **E**fficient **Qu**antization-aware **A**daptation (EQuA). We first evaluate weight importance by scaling the magnitude of each weight with the corresponding input activation and split important weights from the backbone to construct a lightweight Sub-network Side Adapter (SSA) branch, while the less important weights remain in the Memory-Intensive Backbone (MIB) branch. During adaptation, we propose a Side Adapter Quantization-aware Fine-tuning (SAQF) strategy by freezing MIB and fine-tuning SSA end to end, eliminating intermediate activations cached in MIB and substantially reducing memory usage. We also propose a Block-wise Activation Quantization Fine-tuning (BAQF) strategy to optimize activation quantizers by minimizing block-wise quantization errors. SAQF and BAQF are executed alternately, alleviating the memory consumption and quantization errors exacerbated by the deep backbone and activation quantizers. At deployment, we merge the SSA with the MIB, yielding a fully quantized model with no extra computational overhead.

We evaluate the effectiveness of our EQuA on representative visual foundation models including SAM [20] and ViT [8]. We conduct experiments on SAM across a diverse set of downstream tasks, including medical images, natural images, and agricultural images. For ViT, we evaluate on the VTAB [38] benchmark, which comprises 19 datasets spanning three categories: Natural, Specialized, and Structured. Extensive experiments demonstrate that our EQuA achieves an elegant trade-off between performance and efficiency. For example, the 4-bit SAM model adapted and quantized with EQuA reduces memory consumption by over 70% during adaptation compared to state-of-the-art joint quantization and adaptation methods, while maintaining competitive performance.

The main contributions of this work are as follows:

1) We propose a novel method, EQuA, to achieve efficient joint quantization and adaptation. To the best of our knowledge, this is the first work specifically designed for visual foundation models.

2) We design a unique side adapter, SSA, by splitting a sub-network from the backbone, which can save GPU memory during adaptation and can be merged back into the backbone at deployment.

3) We tailor two adaptation strategies, SAQF and BAQF, to enable efficient joint quantization and adaptation in a quantization-aware manner, while preserving memory efficiency and mitigating quantization errors.

4) Extensive experiments demonstrate that our EQuA achieves a superior performance-efficiency trade-off for visual foundation models compared with existing methods.

2 Related Work

Adaptation for Foundation Models. Foundation models, such as GPT-3 [1] and BERT [6], have transformed natural language processing by enabling strong generalization across diverse tasks through large-scale pre-training. This paradigm has extended to the visual domain, resulting in powerful visual foundation models based on Vision Transformers (ViTs), including ViT-Base [8],

CLIP [27], Segment Anything Model (SAM) [20], and Stable Diffusion [28]. Pre-trained on large-scale image datasets, these visual foundation models exhibit strong performance across classification, segmentation, detection, and generation tasks. To adapt them efficiently, parameter-efficient fine-tuning (PEFT) techniques, such as Adapter [3, 12, 39], LoRA [13, 40], and Prompt Tuning [16], have been proposed, which insert or adjust small trainable components while keeping the backbone frozen. However, deploying such models on edge devices requires jointly addressing adaptation and quantization challenges. In this work, we propose an efficient quantization-aware adaptation method tailored for visual foundation models.

Quantization. Quantization is a widely used compression and acceleration technique for deploying models on resource-constrained devices such as smartphones and TVs, by converting floating-point weights and activations into low-bit integers. It typically includes quantization-aware training (QAT) and post-training quantization (PTQ). QAT [9,29] optimizes both model weights and quantizers to achieve near full-precision accuracy, but is computationally expensive for large-scale foundation models. PTQ [7, 22–24, 36] is more lightweight, requiring only a small calibration set, but lacks end-to-end adaptation for downstream tasks and suffers from larger errors at low bit widths. In this work, we aim to enable efficient joint quantization and adaptation process for visual foundation models.

Joint Quantization and Adaptation. Joint quantization and adaptation [2, 5], which integrates quantization into PEFT techniques, has attracted increasing attention. QLoRA [5] compresses foundation models to 4-bit precision and recovers performance with LoRA [13]. QA-LoRA [37] merges LoRA parameters into quantization parameters for efficient deployment. PEQA [18] directly fine-tunes quantization parameters on downstream tasks. These methods are significant for LLMs where memory usage is dominated by floating-point weights. In visual foundation models, however, memory is dominated by activations cached for backpropagation in quantized deep backbones and activation quantizers, which limits the significance of existing methods. Although QST [39] mitigates this issue by attaching side adapters to a quantized backbone, reducing memory from backward gradients, these adapters introduce extra computation and inherit quantization errors from the backbone, limiting inference efficiency and performance. In this work, our method introduces no extra modules and achieves memory-efficient adaptation for visual foundation models.

3 Motivation

Visual foundation models with activation quantization face significant training memory challenges during adaptation. As shown in Fig. 1b, we conduct a preliminary experiment on SAM-H, a representative visual foundation model with numerous parameters, by fine-tuning its weights and quantizers under a quantization setting. We observe that floating-point weights of SAM-H occupy only 3.3% of total GPU memory, making mainstream strategies (*e.g.*, QA-LoRA [37]) in LLMs that save memory by quantizing the backbone largely ineffective for

visual foundation models. Instead, the dominant consumption arises from two sources. First, fine-tuning the backbone weights alone consumes over 45 GB because multiple memory-intensive ViT blocks of the deep backbone cache large amounts of intermediate activations for backpropagation. Such observation is consistent with prior studies [26]. Second, fine-tuning numerous activation quantizers introduces additional cached activations, further leading to an extra 21.8% memory consumption.

QST [39], an inspiring solution, reduces memory by attaching additional side adapters to a quantized LLM backbone. However, these side adapters introduce extra computation costs, limiting inference efficiency, as shown in Fig. 1a. Moreover, QST does not optimize quantizers. In vision tasks, the activation quantization is often sensitive and incurs substantial quantization errors. These errors accumulate throughout the backbone and are absorbed by the side adapters, degrading the performance of visual foundation models. Further analysis is provided in the supplementary material.

In this work, we design EQuA to address memory issues and preserve performance in a quantization-aware manner while introducing no additional modules.

4 Method

4.1 Preliminaries

Uniform Quantization. Uniform quantization is widely used in various quantization methods due to its compatibility with hardware acceleration. The formulation of b-bit uniform quantization of the floating-point value x is as follows:

$$\begin{aligned} Quant(x, s, z): &\quad x_q = clip(\lfloor \frac{x}{s} \rceil + z, 0, 2^b - 1), \\ Dequant(x_q, s, z): &\quad \hat{x} = s \cdot (x_q - z) \approx x, \end{aligned} \tag{1}$$

where x_q and $\hat{x}$ denote the quantized and de-quantized values, respectively. During inference, $\hat{x}$ can be replaced by x_q to enable integer-only computation [14]. $\lfloor \cdot \rceil$ denotes the rounding function, and $clip(x, l, u) = min(max(x, l), u)$. The scaling factor s and zero point z are computed from the observed bounds of x:

$$s = \frac{x_{ub} - x_{lb}}{2^b - 1}, \quad z = \lfloor -\frac{x_{lb}}{s} \rceil. \tag{2}$$

Using separate s and z for each channel defines channel-wise quantization, while sharing single s and z across all channels defines tensor-wise quantization. In this work, we apply the channel-wise quantization on weights and the tensor-wise quantization on activations. For clarity, we omit explicit quantization and de-quantization operators in the following equations unless otherwise specified.

ViT Block. Visual foundation models are mainly based on the ViT block [8], which consists of a Multi-Head Self-Attention (MHSA) module and a Multi-Layer Perceptron (MLP) module. The formulas are below:

$$\begin{aligned} Y_{l-1} &= MHSA(LN(F_{l-1})) + F_{l-1}, \\ F_l &= MLP(LN(Y_{l-1})) + Y_{l-1}, \end{aligned} \tag{3}$$

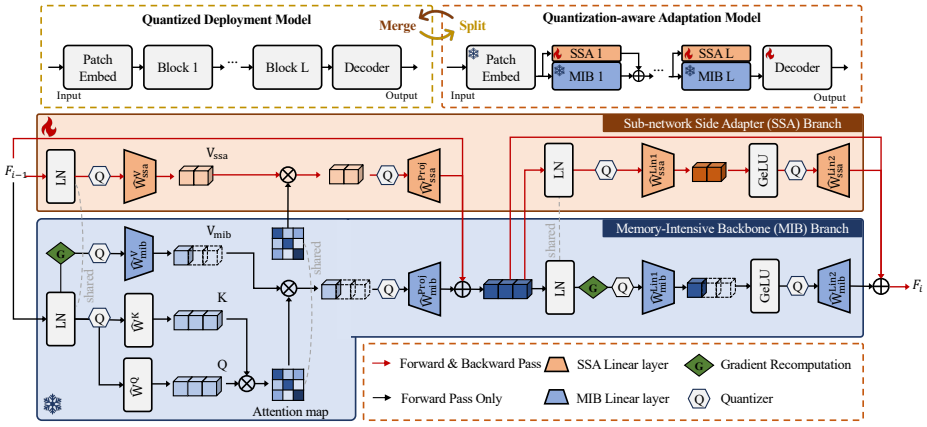


Fig. 2: Overview of our EQuA. During quantization-aware adaptation for downstream tasks, we split each ViT block into a Memory-Intensive Backbone (MIB) branch and a Sub-network Side Adapter (SSA) branch. To reduce memory consumption, the computation graph is constructed only for the SSA branch to cache intermediate activations. At deployment, the two branches are merged back into the original ViT block, introducing no additional computational overhead. $\hat{W}$ represents the quantized weights of linear layers. The split layers share the same activation quantizer.

where LN is layer normalization. F_{l-1}/F_l denote input/output of l-th ViT block. Here, the bias term is omitted for simplicity. The calculation of MHSA module is as follows:

$$MHSA(X) = AXW^V W^{Proj}, \tag{4}$$

where A denotes the attention map and is calculated by W^Q and W^K. W^Q, W^K, $W^V \in \mathbb{R}^{D\times HD_h}$ and $W^{Proj} \in \mathbb{R}^{D\times D}$. H is the number of attention heads and D_h is the dimension size of each head, where $D = D_h \cdot H$. The calculation of MLP module is as follows:

$$MLP(X) = GeLU(XW^{Lin1})W^{Lin2}, \tag{5}$$

where $W^{Lin1} \in \mathbb{R}^{D\times D_{mlp}}$ and $W^{Lin2} \in \mathbb{R}^{D_{mlp}\times D}$. D_{mlp} is the hidden dimension size.

4.2 Sub-network Side Adapter

To mitigate the substantial memory overhead from the deep backbone of visual foundation models during adaptation and avoid additional computations and parameters during deployment, we design an SSA inspired by recent works [17, 39]. We split a lightweight sub-network from each of the two modules in Eq. 3 to construct the SSA for the ViT block. The MHSA module can be split into

two terms, which correspond to the MIB and SSA branches, respectively:

$$\begin{aligned} MHSA(X) &= AX \begin{bmatrix} W^V_{mib} & W^V_{ssa} \end{bmatrix} \begin{bmatrix} W^{Proj}_{mib} \\ W^{Proj}_{ssa} \end{bmatrix} \\ &= AXW^V_{mib}W^{Proj}_{mib} + AXW^V_{ssa}W^{Proj}_{ssa}, \\ &= MHSA_{mib}(X) + MHSA_{ssa}(X), \end{aligned} \tag{6}$$

where $W^V_{mib} \in \mathbb{R}^{D\times(D-D_s)}$ and $W^{Proj}_{mib} \in \mathbb{R}^{(D-D_s)\times D}$ represent weights of linear layer in the backbone branch, and $W^V_{ssa} \in \mathbb{R}^{D\times D_s}$ and $W^{Proj}_{ssa} \in \mathbb{R}^{D_s\times D}$ represent weights of linear layer in the SSA branch. The attention map A is shared in $MHSA_{mib}$ and $MHSA_{ssa}$. W^Q and W^K cannot be expressed in the block-matrix form described in Eq. 6, and are therefore kept unsplit. During backpropagation, we update W^V_{ssa} and W^{Proj}_{ssa}, while preventing gradient flow through the path involving $MHSA_{mib}$ and attention map A. The MLP module can also be split into two terms:

$$\begin{aligned} MLP(X) &= GeLU(X \begin{bmatrix} W^{Lin1}_{mib} & W^{Lin1}_{ssa} \end{bmatrix}) \begin{bmatrix} W^{Lin2}_{mib} \\ W^{Lin2}_{ssa} \end{bmatrix} \\ &= GeLU(XW^{Lin1}_{mib})W^{Lin2}_{mib} + GeLU(XW^{Lin1}_{ssa})W^{Lin2}_{ssa}, \\ &= MLP_{mib}(X) + MLP_{ssa}(X), \end{aligned} \tag{7}$$

where $W^{Lin1}_{mib} \in \mathbb{R}^{D\times(D_{mlp}-D_s)}$, $W^{Lin2}_{mib} \in \mathbb{R}^{(D_{mlp}-D_s)\times D}$, $W^{Lin1}_{ssa} \in \mathbb{R}^{D\times D_s}$, and $W^{Lin2}_{ssa} \in \mathbb{R}^{D_s\times D}$. D_s is the SSA dimension. We update W^{Lin1}_{ssa} and W^{Lin2}_{ssa} and freeze MLP_{mib}.

We propose a strategy to split the SSA from the original linear layer based on weight importance, assuming that weight channels with higher importance have greater impact on performance. A simple and effective way to estimate weight importance is by measuring the magnitude of the weights, as channels with larger magnitudes are generally more important [11]. However, for visual foundation models adapted to downstream tasks, input activation distributions may shift across domains. Inspired by existing work [31], we scale weight magnitudes by input activations to better capture the relative importance of each channel for downstream adaptation. Consider an input activation X of shape $(B \times N, D_{in})$ and a linear layer with weight $W \in \mathbb{R}^{D_{in}\times D_{out}}$, where B and N are batch size and sequence length, respectively. The scaled magnitude used to estimate weight importance is defined as follows. Detailed mathematical derivation about weight importance is provided in the supplementary material.

$$M_i = \sum_{j=1}^{D_{out}} \|X_{:,i}\|_2 \cdot |W_{i,j}|, \; P^M_i = Softmax(M_i), \tag{8}$$

where M_i denotes the weight importance of i-th input channel, P^M_i denotes the probability of importance, $|\cdot|$ represents the absolute value function, and $\|X_{:,i}\|_2$ denotes the L_2 norm computed over the i-th channel across all $B\times N$ tokens. We apply Eq. 8 to W^{Proj} and W^{Lin2} to compute their respective weight importance

probabilities, guided by which a subset of channels, referred to as SSA channels, is stochastically selected to construct the SSA:

$$\begin{aligned} MHSA_{ssa}: \quad & W^{V}_{ssa} = W^{V}_{:,\mathcal{I}^{MHSA}_{ssa}}, \qquad W^{Proj}_{ssa} = W^{Proj}_{\mathcal{I}^{MHSA}_{ssa},:}, \\ MLP_{ssa}: \quad & W^{Lin1}_{ssa} = W^{Lin1}_{:,\mathcal{I}^{MLP}_{ssa}}, \qquad W^{Lin2}_{ssa} = W^{Lin2}_{\mathcal{I}^{MLP}_{ssa},:}, \end{aligned} \tag{9}$$

where $\mathcal{I}^{MHSA}_{ssa}$ and $\mathcal{I}^{MLP}_{ssa}$ denote the index sets of SSA channels for MHSA and MLP modules. In practice, both sets are sampled according to their corresponding probabilities P^{M}_{i}, rather than by selecting the top-D_s M_i, for a more stable convergence. Since the original linear layer weights are split into W_{ssa} and W_{mib}, they can be merged at deployment to obtain a fine-tuned weight with the same shape as the original one, without introducing extra parameters. Further analysis about SSA channel selection is provided in the supplementary material.

4.3 Side Adapter Quantization-aware Fine-tuning

Fig. 2 illustrates the SAQF process of our EQuA. During the adaptation phase, each ViT block is split into a backbone branch and an SSA branch. We fine-tune weights and weight quantizers of the quantized linear layers in the SSA branch while keeping the MIB branch frozen. Since the computation graph is constructed only for the low-dimensional SSA branch during adaptation, which caches low-dimensional intermediate activations, and the high-dimensional MIB branch is excluded from the graph construction, this results in substantial memory savings.

Although the MIB branch parameters are frozen, gradients with respect to its input activations can still be recomputed to propagate supervision to the SSA branch. These gradients carry quantization-aware information from MIB, allowing SSA to compensate for performance degradation. Naively caching intermediate activations for this purpose would reintroduce substantial memory overhead, which contradicts our effort to reduce training memory. Thanks to the absence of learnable parameters in the MIB branch, the gradient recomputation of the input activations of $Lin1$ layer and V layer in the MIB branch, denoted as X^{Lin1}_{mib} and X^{V}_{mib}, can be achieved in a memory-efficient way with a one-step computation. The computation of our gradient recomputation is as follows:

$$\begin{aligned} \frac{\partial L}{\partial X^{Lin1}_{mib}} &= \left(\frac{\partial L}{\partial Y^{Lin2}_{mib}} \hat{W}^{Lin2,T}_{mib} \odot \mathbb{1}^{Lin2}_{mib}\right) \odot \phi(\hat{X}^{Lin1}_{mib} \hat{W}^{Lin1}_{mib}) \hat{W}^{Lin1,T}_{mib} \odot \mathbb{1}^{Lin1}_{mib}, \\ \frac{\partial L}{\partial X^{V}_{mib}} &= \left(A^{T}\left(\left(\frac{\partial L}{\partial Y^{Proj}_{mib}} \hat{W}^{Proj,T}_{mib}\right) \odot \mathbb{1}^{Proj}_{mib}\right) \hat{W}^{V,T}_{mib}\right) \odot \mathbb{1}^{V}_{mib}, \end{aligned} \tag{10}$$

where $\odot$ is the element-wise multiplication. $\hat{W}^{Proj,T}_{mib}$ denotes the transpose of de-quantized $\hat{W}^{Proj}_{mib}$, and the other notations follow analogously. $\hat{X}^{Lin1}_{mib}$ is the de-quantized X^{Lin1}_{mib}. Y^{Proj}_{mib} and Y^{Lin2}_{mib} denote the output of $Proj$ layer and $Lin2$ layer in the MIB branch, respectively. $\phi(\cdot)$ represents the analytical gradients of GeLU function. $\mathbb{1}^{V}_{mib}$, $\mathbb{1}^{Proj}_{mib}$, $\mathbb{1}^{Lin1}_{mib}$, and $\mathbb{1}^{Lin2}_{mib}$ are boolean mask tensors, which are computed from activation quantizers of X^{V}_{mib}, X^{Proj}_{mib}, X^{Lin1}_{mib}, and X^{Lin2}_{mib},

respectively. For example, the mask $\mathbb{1}^{V}_{mib}$ is computed from the pre-clipping quantized value, and the other notations follow analogously:

$$[\mathbb{1}^{V}_{mib}]_{ij} = \begin{cases} 1, & \text{if } 0 \leq [\lfloor \frac{X^{V}_{mib}}{s_{X^V}} \rceil + z_{X^V}]_{ij} \leq 2^b - 1 \\ 0, & \text{otherwise} \end{cases} \quad X^{V}_{mib} \in \mathbb{R}^{BN \times D_{in}}, \tag{11}$$

where s_{X^V} and z_{X^V} denote scaling factor and zero point of the activation quantizer for X^{V}_{mib}. Note that the activation quantizers and input activations of V and $Lin1$ layers in MIB branch are also the same activation quantizers and input activations of V and $Lin1$ layers in SSA branch, *i.e.*, $X^{V}_{mib} = X^{V}_{ssa}$, $X^{Lin1}_{mib} = X^{Lin1}_{ssa}$, $\hat{X}^{Lin1}_{mib} = \hat{X}^{Lin1}_{ssa}$, $\mathbb{1}^{V}_{mib} = \mathbb{1}^{V}_{ssa}$, and $\mathbb{1}^{Lin1}_{mib} = \mathbb{1}^{Lin1}_{ssa}$. Therefore, tensors in Eq. 10, except for $\mathbb{1}^{Proj}_{mib}$ and $\mathbb{1}^{Lin2}_{mib}$, are already shared by the SSA branch and do not need to be recomputed. We manually cache $\mathbb{1}^{Proj}_{mib}$ and $\mathbb{1}^{Lin2}_{mib}$ during the forward pass before loss computation. The recomputed gradients are merged into the SSA gradient flow, allowing SSA to learn quantization-aware information from the frozen MIB branch and improving adaptation performance. More details about gradient recomputation can be found in the supplementary material.

To further reduce trainable parameters, we introduce LoRA [13] into the linear layers of the SSA branch, enabling a low-rank quantization-aware adaptation:

$$\hat{W}_{ssa} = Dequant(Quant(W_{ssa} + \alpha \cdot A_r B_r, s_{w_{ssa}}, z_{w_{ssa}}), s_{w_{ssa}}, z_{w_{ssa}}), \tag{12}$$

where $Dequant(Quant(\cdot))$ is the uniform quantization in Eq. 1, $s_{w_{ssa}}$ and $z_{w_{ssa}}$ are scaling factors and zero points of W_{ssa}, respectively, $A_r \in \mathbb{R}^{d_{in} \times r}$ and $B_r \in \mathbb{R}^{r \times d_{out}}$ are low-rank matrices, and α is a scalar. During adaptation, we freeze W_{ssa} and fine-tune A_r, B_r, and $s_{w_{ssa}}$. At deployment, the SSA and MIB branches can be merged into a single ViT block, with all linear layers preserving the shape of the original model. This design introduces no additional parameters or computations, thereby maintaining inference efficiency.

4.4 Block-wise Activation Quantization Fine-tuning

We design the BAQF strategy to mitigate quantization errors and reduce memory consumption incurred by activation quantizers in the backbone. After merging the SSA and MIB branches into a unified ViT block, we perform BAQF by minimizing the Mean Squared Error (MSE) between the outputs of each quantized block $\mathcal{F}_l$ and the corresponding block $\mathcal{F}^{fp}_l$ with quantizers disabled:

$$s_a \leftarrow s_a - \eta \cdot \nabla_{s_a} \mathbb{E}[\|\mathcal{F}^{fp}_l(Y_{l-1}) - \mathcal{F}_l(\hat{Y}_{l-1})\|^2_2] \tag{13}$$

where Y_{l-1} and $\hat{Y}_{l-1}$ denote floating-point input and de-quantized input of l-th block, respectively, s_a denotes scaling factors of activation quantizers. Here, we update only s_a while keeping all other parameters frozen. Since the BAQF strategy operates in a block-wise manner, the memory introduced by activation quantizers is negligible, as verified in Table 3. BAQF is performed once every K epochs of SAQF, where K denotes the block-wise fine-tuning frequency. More details of the processing pipeline are provided in the supplementary material.

Table 1: Semantic segmentation results of SAM-H on various downstream datasets. *Param* and *Mem* indicate the number of trainable parameters and training memory consumption, respectively. The best, second-best, and third-best results are marked in **bold**, underlined, and double underlined, respectively. The 🪶 denotes efficiency in the corresponding *Param* or *Mem* column, *e.g.*, the model can be trained on a single RTX 3090 GPU (24 GB).

Method	Bit	Param↓	Mem↓	Medical						Natural				Agricultural	
				CVC-T		Kvasir		ETIS		CAMO		COD10K		Leaf	
				$S_\alpha\uparrow$	$E_\phi\uparrow$	$S_\alpha\uparrow$	$E_\phi\uparrow$	$S_\alpha\uparrow$	$E_\phi\uparrow$	$S_\alpha\uparrow$	$E_\phi\uparrow$	$S_\alpha\uparrow$	$E_\phi\uparrow$	mIoU↑	mDice↑
Full	W32A32	641.09 M	49.6 GB	0.917	0.940	0.917	0.954	0.815	0.833	0.805	0.860	0.837	0.892	0.710	0.818
SAM-Adapter	W32A32	4.25 M	34.7 GB	0.925	0.939	0.915	0.941	0.829	0.838	0.867	0.903	0.894	0.930	0.671	0.780
LoRA	W32A32	8.03 M	45.4 GB	0.937	0.963	0.936	0.962	0.845	0.857	0.889	0.933	0.920	0.959	0.730	0.831
LSQ	W8A8	641.46 M	72.4 GB	0.920	0.958	0.900	0.937	0.793	0.825	0.808	0.864	0.843	0.900	0.712	0.818
NIPQ	W8A8	641.46 M	72.4 GB	0.916	0.956	0.904	0.936	0.797	0.819	0.805	0.859	0.841	0.898	0.702	0.811
QA-LoRA	W8A8	🪶 7.13 M	68.9 GB	0.921	0.936	**0.930**	**0.953**	**0.865**	**0.895**	**0.885**	**0.928**	**0.919**	**0.959**	0.720	0.823
PEQA	W8A8	🪶 4.43 M	69.4 GB	**0.929**	**0.969**	0.929	**0.953**	0.827	0.844	0.877	0.923	0.909	0.951	**0.732**	**0.835**
QST	W8A8	🪶 8.30 M	🪶 21.6 GB	0.918	0.939	0.899	0.925	0.799	0.819	0.839	0.884	0.885	0.930	0.678	0.792
EQuA (Ours)	W8A8	🪶 7.67 M	🪶 18.5 GB	0.925	0.961	0.919	0.948	0.834	0.848	0.863	0.900	0.902	0.942	0.717	0.820
LSQ	W4A4	641.46 M	72.4 GB	0.876	0.895	0.895	0.931	0.753	0.769	0.760	0.804	0.806	0.867	0.697	0.807
NIPQ	W4A4	641.46 M	72.4 GB	0.884	0.908	0.901	0.934	0.771	0.791	0.762	0.808	0.818	0.875	0.699	0.809
QA-LoRA	W4A4	🪶 7.13 M	68.9 GB	0.919	0.929	0.923	0.948	**0.830**	**0.862**	**0.858**	**0.903**	**0.903**	**0.945**	0.705	0.815
PEQA	W4A4	🪶 4.43 M	69.4 GB	**0.927**	**0.959**	**0.926**	**0.952**	0.823	0.842	0.855	0.899	0.892	0.937	**0.718**	**0.819**
QST	W4A4	🪶 8.30 M	🪶 21.6 GB	0.905	0.915	0.894	0.921	0.729	0.754	0.788	0.836	0.833	0.884	0.656	0.772
EQuA (Ours)	W4A4	🪶 7.67 M	🪶 18.5 GB	0.922	0.938	0.915	0.947	0.799	0.837	0.833	0.878	0.873	0.920	0.709	0.816

5 Experiments and Results

5.1 Experimental Setup

Settings. To evaluate our EQuA, we perform downstream quantization and adaptation experiments on two representative visual foundation models, SAM-Huge (SAM-H) [20] and ViT-Base (ViT-B) [8], both of which are pre-trained on large scale datasets. We first use RepQ-ViT [22] to perform uniform quantization on the pre-trained models as a PTQ initialization by jointly calibrating quantization parameters on the original full-precision weights, and then fine-tune the quantized models on downstream datasets. We include different bit-width configurations supported by the hardware, including W8A8 (8-bit weights and activations) and W4A4 (4-bit weights and activations). We fine-tune quantized pre-trained models using the Adam [19] and the CosineAnnealingLR [25]. We set the batch size to 2 for SAM-H and 32 for ViT-B. The SSA dimension D_s (Eq. 7) is set to 64, the LoRA rank r (Eq. 12) is set to 4, the scalar α (Eq. 12) is set to 32 for SAM-H and 8 for ViT-B. We use multiple LoRA branches for the V and $Proj$ layers with shapes $(10, r, d_{in})$, $(10, r, d_{out})$ and $(2, r, d_{in})$, $(2, r, d_{out})$, respectively, which are summed into an effective (r, d_{in}) and (r, d_{out}) LoRA weight; in the ablation, only r is varied. The K (Sec. 4.4) is set to 10 for SAM-H and 20 for ViT-B. More detailed setups are in the supplementary material.

Datasets and Metrics. Following recent studies [3, 40], we conduct experiments on SAM-H across three downstream semantic segmentation scenarios: polyp segmentation for medical images [15, 30, 34], camouflaged object detection for natural images [10, 21], and leaf disease segmentation for agricultural

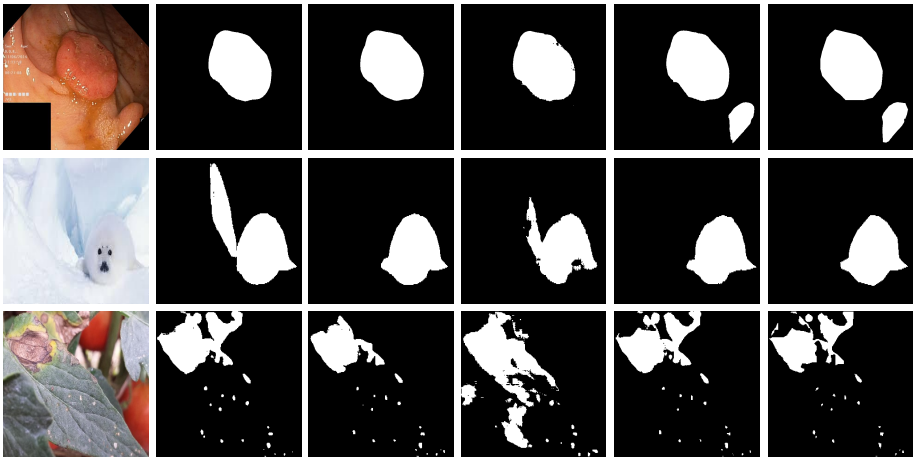


Fig. 3: Visualization results of SAM-H on semantic segmentation under the W4A4 setting, with cases from three different downstream datasets: Kvasir, CAMO, and Leaf.

images [40]. For evaluation of semantic segmentation results, we use the widely-used S-measure (S_α) and mean E-measure (E_ϕ) for medical images and natural images, and use mean IoU (mIoU) and mean Dice (mDice) for agricultural images. We also conduct experiments on ViT-B using the VTAB [38] benchmark, which comprises 19 image classification datasets across three categories and is widely adopted in the vision community [17, 26]. We use the Top-1 accuracy to evaluate image classification results. More details about datasets and metrics are provided in the supplementary material.

Baselines. We include following baselines: 1) floating-point methods: Full (all parameters), Linear (the classification head), SAM-Adapter [3], LoRA [13]; 2) standard QAT methods: LSQ [9] and NIPQ [29]; 3) joint quantization and adaptation methods: PEQA [18], QA-LoRA [37], and QST [39]. In the Full fine-tuning, we use the same learning rate for the backbone and the decoder or classification head. Floating-point methods act as the upper-bound reference.

5.2 Semantic Segmentation Results

Table 1 presents the quantitative results on SAM-H. Our EQuA consistently achieves a superior trade-off between performance and efficiency across three segmentation domains: medical, natural, and agricultural. Standard QAT methods fine-tune all parameters and incur substantial memory overheads (over 70 GB), making them inefficient in both parameter and memory. While QA-LoRA and PEQA reduce the number of trainable parameters, their memory consumption remains above 60 GB. In contrast, our EQuA achieves comparable performance with significantly lower memory cost. For example, on the CVC-T dataset under the W4A4 setting, EQuA achieves an S_α of 0.922, closely matching the best

Table 2: Classification results (%) of adapting ViT-B on the VTAB-1K benchmark. The best, second-best, and third-best results are marked in **bold**, underlined, and double underlined, respectively. FP represents full-precision (W32A32). The 🖇 denotes efficiency in the corresponding *Param* or *Mem* column.

				Natural							Specialized				Structured								Averages			
Bit	Method	Param (M)	Mem (GB)	Cifar100	Caltech101	DTD	Flower102	Pets	SVHN	Sun397	Camelyon	EuroSAT	Resisc45	Retinopathy	Clevr-Count	Clevr-Dist	DMLab	KITTI-Dist	dSpr-Loc	dSpr-Ori	sNORB-Azim	sNORB-Ele	Avg Natural	Avg Specialized	Avg Structured	Group Average
FP	Full	85.80	5.0	65.7	89.0	68.1	96.7	86.4	88.8	52.3	84.6	94.6	83.0	74.3	62.4	60.9	47.1	75.9	75.3	36.4	21.9	29.5	78.0	84.1	51.2	71.1
	Linear	0.00	0.6	64.4	85.0	63.2	97.0	86.3	36.6	51.0	78.5	87.5	68.5	74.0	34.3	30.6	33.2	55.4	12.5	20.0	9.6	19.2	69.1	77.1	26.9	57.6
W8A8	LSQ	85.88	8.4	64.0	89.4	66.5	96.6	85.1	87.8	52.4	83.3	94.1	83.7	74.0	64.1	60.7	49.4	75.8	74.3	37.9	21.7	30.4	77.4	83.8	51.8	71.0
	NIPQ	85.88	8.4	64.4	89.1	66.9	97.4	86.7	87.8	53.4	83.0	94.2	81.6	74.7	63.2	60.8	49.0	75.8	**77.2**	39.1	26.0	31.0	78.0	83.4	52.8	71.4
	QA-LoRA	🖇 0.68	6.6	**73.9**	89.0	72.9	**99.2**	**91.2**	81.9	**57.2**	85.0	**95.9**	**86.5**	74.8	70.8	59.4	48.1	76.7	71.9	**56.4**	25.5	36.0	**80.8**	85.5	55.6	74.0
	PEQA	🖇 0.08	6.7	67.9	**90.0**	69.5	98.6	88.4	**88.9**	52.4	87.1	**95.9**	85.6	75.1	76.6	**61.8**	**51.1**	79.5	77.0	51.6	**31.4**	38.9	79.3	**85.9**	**58.5**	**74.6**
	QST	🖇 0.85	🖇 1.2	62.0	88.9	71.0	**99.2**	89.7	70.3	56.0	**87.7**	95.2	80.6	**75.7**	**79.0**	61.1	47.9	**79.7**	74.0	29.8	30.6	**44.5**	76.7	84.8	55.8	72.4
	EQuA (Ours)	🖇 0.64	🖇 1.7	71.7	89.6	**73.5**	99.1	90.8	82.4	56.1	85.9	95.6	84.2	75.2	77.0	60.4	42.0	77.5	74.5	43.3	30.0	42.5	80.5	85.2	55.9	73.9
W4A4	LSQ	85.88	8.4	59.3	86.9	65.3	91.8	82.1	81.6	44.7	79.6	94.1	80.2	72.4	58.4	59.5	45.8	74.9	74.3	33.0	11.5	29.5	73.1	81.6	48.4	67.7
	NIPQ	85.88	8.4	59.3	87.1	65.9	93.6	78.8	85.7	48.8	82.5	93.2	80.5	72.8	60.3	60.6	40.3	74.7	75.2	36.7	18.6	30.0	74.2	82.3	49.6	68.7
	QA-LoRA	🖇 0.68	6.6	**68.4**	**88.8**	71.2	**99.0**	**89.8**	83.2	**52.8**	84.3	95.5	**85.5**	71.0	70.3	59.2	47.8	76.1	71.0	51.0	25.0	35.6	**79.0**	84.1	54.5	72.5
	PEQA	🖇 0.08	6.7	63.0	88.4	67.7	98.0	86.2	**88.4**	47.7	**85.5**	**95.9**	84.6	72.4	73.9	**61.5**	**49.7**	**77.5**	**75.8**	**51.5**	**30.1**	35.8	77.1	**84.6**	**57.0**	**72.9**
	QST	🖇 0.85	🖇 1.2	47.8	87.8	67.6	98.1	86.6	67.9	48.5	84.1	95.0	80.0	**75.4**	**77.2**	60.1	46.6	76.9	73.3	28.3	27.7	**43.4**	72.0	83.7	54.2	70.0
	EQuA (Ours)	🖇 0.64	🖇 1.7	63.6	**88.8**	**72.3**	98.6	88.3	80.5	51.8	85.3	95.1	81.4	73.6	74.1	60.0	40.0	76.1	72.7	41.4	29.7	40.0	77.7	83.9	54.3	72.0

Table 3: Ablation of each component.

Components	Param↓ (M)	Train↓ (hour/epoch)	Mem↓ (GB)	Kvasir $S_\alpha \uparrow$	Kvasir $E_\phi \uparrow$
Base	641.46	0.60	72.4	0.895	0.931
+SSA*	29.90	0.57	67.6	0.918	0.941
+SSA	29.90	0.32	18.6	0.881	0.913
+SSA & GR	29.90	0.50	18.6	0.910	0.932
+SSA & GR & LoRA	7.67	0.50	18.5	0.906	0.926
+SSA & GR & LoRA & BAQF	7.67	0.52	18.5	0.915	0.947

Table 4: The time and memory cost to perform SSA channel selection on SAM-H.

Batch size	Time	Memory	Training Memory
1	13.9 sec	6.1 GB	10.7 GB
2	20.6 sec	9.6 GB	18.5 GB
4	33.2 sec	16.4 GB	33.9 GB
8	57.6 sec	30.2 GB	65.0 GB

result of PEQA (0.927) while reducing memory consumption from 69.4GB to 18.5 GB (a 73.3% reduction), enabling the quantization-aware adaptation of SAM-H on a single RTX 3090 GPU. Compared to the memory-efficient QST, our EQuA achieves better performance. For example, on the Leaf dataset under the W4A4 setting, EQuA outperforms QST by 0.053 mIoU and 0.044 mDice (0.709/0.816 vs. 0.656/0.772). As shown in Fig. 3, we also visualize the comparison results. The results demonstrate our EQuA can also achieve superior qualitative performance. In summary, our EQuA offers a strong balance between performance and efficiency. More quantitative and qualitative results are provided in the supplementary material.

5.3 Image Classification Results

In Table 2, we report the comparison results on ViT-B. We report *Mem* on the *Cifar100*. The results further confirm that our EQuA continues the advantage in balancing performance and efficiency across diverse downstream visual tasks on ViT-B. For example, our EQuA requires only 1.7 GB of training memory, achieving a reduction of over 70% compared to PEQA and QA-LoRA, and over 75% compared to LSQ and NIPQ. Our EQuA achieves comparable or superior

Table 5: Ablation study on SSA dimension D_s and LoRA rank r.

D_s	r	Param↓ (M)	Mem↓ (GB)	CVC-T $S_\alpha\uparrow$	CVC-T $E_\phi\uparrow$	Kvasir $S_\alpha\uparrow$	Kvasir $E_\phi\uparrow$
32	4	7.66	18.3	0.915	0.936	0.906	0.933
64	4	7.67	18.5	0.922	0.938	0.915	**0.947**
128	4	7.69	18.7	0.927	0.947	**0.920**	0.943
64	2	6.33	18.5	0.918	0.935	0.906	0.932
64	8	10.35	18.5	**0.930**	**0.954**	0.919	0.944

Table 6: Ablation on the BAQF frequency K and the number of iterations for fine-tuning each block in each BAQF round.

K	0	5	10	15
mIoU↑	0.699	0.708	**0.709**	0.707
mDice↑	0.808	**0.816**	**0.816**	0.813
Iteration	10	50	100	200
mIoU↑	0.703	0.705	0.709	**0.710**
mDice↑	0.812	0.813	0.816	**0.817**

performance on all 19 datasets. Under the W4A4 setting, EQuA attains an average Top-1 accuracy of 72.0%, significantly outperforming QST (70.0%) and NIPQ (68.7%), and closely matching PEQA (72.9%) and QA-LoRA (72.5%).

6 Ablation and Analysis

6.1 Effectiveness of Each Component

In order to reveal the effectiveness of each component of our EQuA, we report the ablation results under the W4A4 setting in Table 3. LSQ, the first row in Table 3, serves as the reference baseline without any additional components. *SSA** denotes fine-tuning only the SSA branch, but the MIB branch is included in the computation graph. *SSA* denotes fine-tuning only the SSA branch, but the MIB branch is excluded from the computation graph. *GR* denotes the gradient recomputation. *LoRA* represents using LoRA in Eq. 12. As shown in Table 3, adopting SSA significantly reduces trainable parameters, training time, and memory usage. GR integrates partial quantization-aware information from the MIB branch into the SSA branch, further facilitating convergence and yielding a significant performance gain (0.910 vs. 0.881 on S_α) with negligible memory overhead. Although the training time increases moderately, it is still lower than that of LSQ (0.50 vs. 0.60). Incorporating LoRA maintains performance and reduces trainable parameters to just 1.2% of LSQ, which can achieve efficient model switching across different downstream tasks. Finally, BAQF further improves performance without additional memory cost, by alleviating quantization errors and memory consumption from activation quantizers.

In addition, we also evaluate the cost of the weight importance calculation and SSA selection in Table 4. *Memory* represents memory cost of SSA channel selection. *Training Memory* means the memory cost of training. Since this process is achieved solely through a one-step forward before fine-tuning, and does not require backward gradients, its time cost is negligible, and its memory cost is significantly lower than that of training.

6.2 Ablation on Hyper-Parameters

In order to reveal the impact of different hyper-parameters of our EQuA, we report the ablation study on the SSA dimension D_s in Eq. 7, the LoRA rank

Table 7: Efficiency analysis during adaptation (left) and deployment (right), which are obtained by setting batch size to 1/2/4, respectively. *Speedup*: inference acceleration relative to the original floating-point model. *OOM*: out-of-memory. Models are trained on a single A100 GPU (80 GB). The input image resolution is $1024 \times 1024 \times 3$.

Method	Memory↓ (GB)	Train↓ (hour/epoch)	Storage↓ (GB)	Latency↓ (sec/batch)	Speedup↑
Full	28.6/49.6/OOM	0.57/0.55/OOM	2.39	0.75/1.44/2.94	1.00×/1.00×/1.00×
PEQA	36.7/69.4/OOM	0.63/0.58/OOM	**0.34**	**0.54/1.06/2.14**	**1.38×/1.36×/1.37×**
QA-LoRA	35.9/68.9/OOM	0.58/0.55/OOM	**0.34**	**0.54/1.06/2.14**	**1.38×/1.36×/1.37×**
QST	10.9/21.6/42.0	0.41/0.37/0.35	0.35	0.63/1.24/2.49	1.18×/1.16×/1.18×
EQuA (Ours)	**10.7/18.5/33.9**	0.55/0.52/0.50	**0.34**	**0.54/1.06/2.14**	**1.38×/1.36×/1.37×**

r in Eq. 12, the frequency K of BAQF, and the number of iterations for fine-tuning each block in each BAQF round (denoted as *Iteration*). All studies are evaluated on SAM-H under the W4A4 setting.

SSA dimension D_s. We set r to 4 when studying the effect of D_s. As listed in Table 5, increasing D_s from 32 to 128 results in negligible growth in trainable parameters and only a slight increase in training memory consumption during adaptation. A larger performance gain is observed when increasing D_s from 32 to 64 (*e.g.*, a 0.014 improvement on E_ϕ in the Kvasir dataset), while the performance is comparable between $D_s = 64$ and $D_s = 128$. Thus, we adopt $D_s = 64$ as the default configuration.

LoRA rank r. In this set of experiments, we set D_s to 64 and change r. As shown in Table 5, increasing r from 2 to 8 significantly increases trainable parameters (from 6.33 M to 10.35 M), while training memory consumption keeps unchanged. However, the corresponding performance improvements become marginal beyond $r = 4$, *e.g.*, 0.915 *vs.* 0.919 on S_α in the Kvasir dataset. This suggests that $r = 4$ offers a good balance between parameter efficiency and performance.

Frequency K of BAQF. We conduct ablation study on the Frequency K using the Leaf dataset, fixing *Iteration* to 100 and varying K. $K = 0$ represents we do not perform BAQF. As listed in Table 6, the performance achieves significant improvements when we adopt BAQF. We select $K = 10$ as the final configuration.

Iteration in each BAQF round. When evaluating the effect of *Iteration* in each BAQF round, we set $K = 10$. As shown in Table 6, the best performance is achieved with 200 iterations. However, since the improvement over 100 iterations is marginal, we finally choose 100 as the default setting for better efficiency.

6.3 Analysis of Efficiency

Table 7 summarizes the efficiency analysis on SAM-H. PEQA, QA-LoRA, QST, and our EQuA are under the W4A4 quantization setting, while Full remains in floating-point. *Memory* and *Train* reflect adaptation efficiency, which are evaluated on the polyp segmentation training set. *Storage*, *Latency*, and *Speedup* reflect deployment efficiency. We adopt the CUDA kernels [4] based on the CUTLASS library for 4-bit inference acceleration to estimate *Latency* and *Speedup*.

Table 8: Analysis of combining our EQuA with the direct low-precision training method like COAT [35] on the Leaf dataset with W4A4 SAM-H.

Method	Mem↓ (GB)	mIoU↑	mDice↑
EQuA	18.5	0.709	0.816
EQuA *w.* COAT [35]	12.7	0.702	0.811

Here, we treat PEQA, QA-LoRA, and our EQuA as standard 4-bit quantized models for deployment evaluation. As shown in Table 7, our EQuA achieves a significant advantage in training efficiency, reducing 73.3% training memory and 10.3% training time compared to PEQA (batch size 2). At deployment, EQuA preserves all the benefits of standard 4-bit quantization, achieving a 1.38× inference speedup with a batch size of 1. Although QST shows similar memory savings, its speedup is lower than that of standard 4-bit quantization due to the extra overhead of the additional adapters. In contrast, our EQuA achieves both training and deployment efficiency simultaneously.

6.4 Discussion on Direct Low-Precision Training Method

We further analyze the compatibility of EQuA with the direct low-precision training methods such as COAT [35]. COAT and our EQuA reduce the activation-induced training memory $\mathcal{M}$ from two orthogonal dimensions: $\mathcal{M}{=}\mathcal{N}{\times}\mathcal{B}$. EQuA reduces $\mathcal{N}$ (number of cached activations), while COAT reduces $\mathcal{B}$ (precision bit-width) by compressing cached activations and optimizer states into low-bit formats. As shown in Table 8, combining them can further reduce memory.

7 Conclusion

We propose EQuA, an efficient quantization-aware adaptation framework for visual foundation models. To reduce the high memory overhead of joint quantization and adaptation, we introduce an SSA branch to enable gradient updates without full-model backpropagation and SAQF and BAQF strategies to preserve memory efficiency and minimize quantization errors. At deployment, SSA is merged back into the backbone, incurring no extra computation costs. Extensive experiments validate that EQuA achieves an optimal trade-off between performance and efficiency, achieving practical quantization-aware adaptation and deployment under resource constraints.

Acknowledgements

The AI-driven experiments, simulations and model training were performed on the robotic AI-Scientist platform of Chinese Academy of Sciences. This work was supported in part by the National Natural Science Foundation of China under Grant 62131003.

References

1. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. In: NeurIPS (2020)
2. Chen, M., Shao, W., Xu, P., Wang, J., Gao, P., Zhang, K., Luo, P.: Efficientqat: Efficient quantization-aware training for large language models. In: ACL. pp. 10081–10100 (2025)
3. Chen, T., Zhu, L., Deng, C., Cao, R., Wang, Y., Zhang, S., Li, Z., Sun, L., Zang, Y., Mao, P.: Sam-adapter: Adapting segment anything in underperformed scenes. In: ICCV Workshop. pp. 3367–3375 (2023)
4. Cho, Y., Lee, C., Kim, S., Park, E.: PTQ4VM: post-training quantization for visual mamba. In: WACV. pp. 1176–1185 (2025)
5. Dettmers, T., Pagnoni, A., Holtzman, A., Zettlemoyer, L.: Qlora: Efficient finetuning of quantized llms. In: NeurIPS (2023)
6. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers). pp. 4171–4186 (2019)
7. Ding, X., Liu, X., Tu, Z., Zhang, Y., Li, W., Hu, J., Chen, H., Tang, Y., Xiong, Z., Yin, B., et al.: Cbq: Cross-block quantization for large language models. In: ICLR (2025)
8. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An image is worth 16x16 words: Transformers for image recognition at scale. In: ICLR (2021)
9. Esser, S.K., McKinstry, J.L., Bablani, D., Appuswamy, R., Modha, D.S.: Learned step size quantization. In: ICLR (2020)
10. Fan, D., Ji, G., Sun, G., Cheng, M., Shen, J., Shao, L.: Camouflaged object detection. In: CVPR. pp. 2774–2784 (2020)
11. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural network. In: NeurIPS. pp. 1135–1143 (2015)
12. Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M., Gelly, S.: Parameter-efficient transfer learning for NLP. In: ICML. pp. 2790–2799 (2019)
13. Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W.: Lora: Low-rank adaptation of large language models. In: ICLR. OpenReview.net (2022)
14. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A.G., Adam, H., Kalenichenko, D.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: CVPR. pp. 2704–2713 (2018)
15. Jha, D., Smedsrud, P.H., Riegler, M.A., Halvorsen, P., de Lange, T., Johansen, D., Johansen, H.D.: Kvasir-seg: A segmented polyp dataset. In: MultiMedia Modeling - 26th International Conference, MMM 2020, Daejeon, South Korea, January 5-8, 2020, Proceedings, Part II. pp. 451–462 (2020)
16. Jia, M., Tang, L., Chen, B., Cardie, C., Belongie, S.J., Hariharan, B., Lim, S.: Visual prompt tuning. In: ECCV. pp. 709–727 (2022)
17. Jiang, Z., Mao, C., Huang, Z., Ma, A., Lv, Y., Shen, Y., Zhao, D., Zhou, J.: Restuning: A flexible and efficient tuning paradigm via unbinding tuner from backbone. In: NeurIPS (2023)

18. Kim, J., Lee, J.H., Kim, S., Park, J., Yoo, K.M., Kwon, S.J., Lee, D.: Memory-efficient fine-tuning of compressed large language models via sub-4-bit integer quantization. In: NeurIPS (2023)
19. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: ICLR (2015)
20. Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A.C., Lo, W., Dollár, P., Girshick, R.B.: Segment anything. In: ICCV. pp. 3992–4003 (2023)
21. Le, T., Nguyen, T.V., Nie, Z., Tran, M., Sugimoto, A.: Anabranch network for camouflaged object segmentation. Comput. Vis. Image Underst. **184**, 45–56 (2019)
22. Li, Z., Xiao, J., Yang, L., Gu, Q.: Repq-vit: Scale reparameterization for post-training quantization of vision transformers. In: ICCV. pp. 17181–17190 (2023)
23. Lin, J., Tang, J., Tang, H., Yang, S., Chen, W., Wang, W., Xiao, G., Dang, X., Gan, C., Han, S.: AWQ: activation-aware weight quantization for on-device LLM compression and acceleration. In: MLSys (2024)
24. Liu, X., Ding, X., Yu, L., Xi, Y., Li, W., Tu, Z., Hu, J., Chen, H., Yin, B., Xiong, Z.: Pq-sam: Post-training quantization for segment anything model. In: ECCV. pp. 420–437 (2024)
25. Loshchilov, I., Hutter, F.: SGDR: stochastic gradient descent with warm restarts. In: ICLR (2017)
26. Mercea, O., Gritsenko, A.A., Schmid, C., Arnab, A.: Time-, memory- and parameter-efficient visual adaptation. In: CVPR. pp. 5536–5545 (2024)
27. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al.: Learning transferable visual models from natural language supervision. In: ICML. pp. 8748–8763 (2021)
28. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-resolution image synthesis with latent diffusion models. In: CVPR. pp. 10674–10685 (2022)
29. Shin, J., So, J., Park, S., Kang, S., Yoo, S., Park, E.: NIPQ: noise proxy-based integrated pseudo-quantization. In: CVPR. pp. 3852–3861 (2023)
30. Silva, J., Histace, A., Romain, O., Dray, X., Granado, B.: Toward embedded detection of polyps in WCE images for early diagnosis of colorectal cancer. Int. J. Comput. Assist. Radiol. Surg. **9**(2), 283–293 (2014)
31. Sun, M., Liu, Z., Bair, A., Kolter, J.Z.: A simple and effective pruning approach for large language models. In: ICLR (2024)
32. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al.: Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971 (2023)
33. Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al.: Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288 (2023)
34. Vázquez, D., Bernal, J., Sánchez, F.J., Fernández-Esparrach, G., López, A.M., Romero, A., Drozdzal, M., Courville, A.: A benchmark for endoluminal scene segmentation of colonoscopy images. Journal of healthcare engineering **2017**(1), 4037190 (2017)
35. Xi, H., Cai, H., Zhu, L., Lu, Y., Keutzer, K., Chen, J., Han, S.: COAT: compressing optimizer states and activations for memory-efficient FP8 training. In: ICLR (2025)
36. Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., Han, S.: Smoothquant: Accurate and efficient post-training quantization for large language models. In: ICML. pp. 38087–38099 (2023)

37. Xu, Y., Xie, L., Gu, X., Chen, X., Chang, H., Zhang, H., Chen, Z., Zhang, X., Tian, Q.: Qa-lora: Quantization-aware low-rank adaptation of large language models. In: ICLR (2024)
38. Zhai, X., Puigcerver, J., Kolesnikov, A., Ruyssen, P., Riquelme, C., Lucic, M., Djolonga, J., Pinto, A.S., Neumann, M., Dosovitskiy, A., et al.: A large-scale study of representation learning with the visual task adaptation benchmark. arXiv preprint arXiv:1910.04867 (2019)
39. Zhang, Z., Zhao, D., Miao, X., Oliaro, G., Zhang, Z., Li, Q., Jiang, Y., Jia, Z.: Quantized side tuning: Fast and memory-efficient tuning of quantized large language models. In: ACL. pp. 1–17 (2024)
40. Zhong, Z., Tang, Z., He, T., Fang, H., Yuan, C.: Convolution meets lora: Parameter efficient finetuning for segment anything model. In: ICLR (2024)