

Accelerated Segmentation with Mixed-Precision Quantization of EfficientViT-SAM

Bernard Ryhede Bengtsson
Joel Bengs



LUND
UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6250
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

© 2024 Bernard Ryhede Bengtsson & Joel Bengs. All rights reserved.
Printed in Sweden by Tryckeriet i E-huset
Lund 2024

Abstract

The wide adoption of machine learning (ML) models in resource-constrained devices for complex vision tasks is contingent on improved computational efficiency. Quantization is a tool for model compression in which weights and activations are approximated with low-precision datatypes, to achieve a smaller model with reduced latency and memory usage. In this thesis, we seek to apply post-training mixed-precision quantization to the image encoder of EfficientViT-SAM, a state-of-the-art Segment Anything Model that produces segmentation masks for any object in an image.

We study the architecture and latency profile of EfficientViT-SAM, and conduct experiments to understand the sensitivity of different architecture components, like layers, on the accuracy achieved. Results show that some convolution layers, in particular depth-wise convolutions, are difficult to quantize without degrading accuracy, while other layers such as the ReLU-based self-attention mechanism used in EfficientViT-SAM are relatively robust under quantization.

From these insights, we design and evaluate several quantization schemes that quantize only non-sensitive layers. More specifically, using TensorRT optimization engine, we reduce the latency of EfficientViT-SAM-XL1 by 2.4%, with a loss of 6.9% accuracy. Using a simulation framework, we reduce the size of EfficientViT-SAM-L0 by 31% with a loss of 9% accuracy. Our extensive experiments show that a desirable accuracy-latency trade-off is hard, if not infeasible, to reach through uniform quantization to INT8. From our qualitative results, we found that the poorly performing models matched baseline performance in simpler scenarios but failed in predicting complex segmentation masks.

Acknowledgements

We wish to thank the Device Platform Research (DPR) Team at Ericsson for providing the opportunity and resources to conduct this master's thesis.

A special thanks to our supervisors Saeed Bastani and Puren Guler - your never-ending knowledge was a guiding light in difficult times. We are sure that we will cross paths again in future endeavors.

To our academic supervisor Pontus Giselsson; thank you not only for your efforts but also for the course in optimization at which We the authors first met. It was perhaps the most insightful (and difficult) course in our engineering degrees, and it paved the path for this opportunity with Ericsson.

Lastly, thank you to our families and friends, whose encouragement and Power-Point assistance have been invaluable.

Contents

1. Introduction	1
1.1 Background	1
1.2 Segment Anything Model	2
1.3 Research questions	3
1.4 Delimitation	4
1.5 Thesis Outline	4
1.6 Individual Contributions	5
2. Background and Related Work	6
2.1 Neural Networks	6
Convolutional Neural Networks	7
Depth-Wise Separable Convolution Layers	7
Vision Transformers	9
2.2 Precision formats	10
2.3 Quantization	11
The Quantization Operator	11
Calibration of Optimal Scaling Factor	12
Granularity of Quantization	13
Uniform and Non-Uniform Quantization	13
Quantization-Aware Training	14
Hardware Acceleration	14
2.4 Segment Anything	15
SA-1B Dataset	15
Promptable Segmentation Task	16
Architecture of SAM	17
2.5 EfficientViT-SAM	17
Architecture of EfficientViT-SAM’s image encoder	19
Residual Connections and ResBlocks	19
Mobile Inverted Convolutions	21
Fused Mobile Inverted Convolution Block	22
The Neck	22

	Global Receptive Field and Multi-Scale Learning	22
	EfficientViT Module: Multi-scale ReLU Linear Attention	23
	Architecture summary	25
	Training of EfficientViT-SAM	28
2.6	Related Work	28
	Related Model Compression Works	28
	Related Quantization Works	29
	Quantization of CNNs	29
	Quantization of ViTs	30
	Related Works on EfficientViT	31
3.	Method	32
3.1	Methodology Overview	32
	Configuration of Quantization Experiments	33
	Latency Analysis with TensorRT	34
	Simulated Layer-Wise Accuracy Analysis	37
3.2	Experimental Setup	40
	Datasets	40
	Evaluation Metrics	40
	Efficiency Metrics	41
	Box-Prompted Segmentation Task	42
	Error Sources	42
	COCO Dataset	42
	Hardware stability	42
4.	Results	44
4.1	Quantitative results	44
	Stage-wise sensitivity results	46
	Type-wise sensitivity results	47
	Layer-wise sensitivity results	48
	Evaluation of mixed-precision schemes	52
	Performance trade-offs	55
4.2	Qualitative results	57
	Object Size	57
	Object Occlusion	61
	Object Density	61
4.3	Method discussion	64
	Discrepancies Between Simulation and TensorRT	64
	Compilation issues	65
5.	Conclusion	66
5.1	Future Works	67
	Bibliography	68

Introduction

Human beings ponder about existential questions. What is the purpose of life? Does an afterlife exist? What is the essence of consciousness? Unable to provide philosophical arguments, this thesis instead investigates the effects of quantization, a compression technique, on EfficientViT-SAM, a state-of-the-art vision model for promptable segmentation tasks. This chapter gives a background of the domain and outlines the research questions examined in this thesis, along with the delimitations.

1.1 Background

One important domain of machine learning (ML) is computer vision. Many value-unlocking applications, such as autonomous driving and robotics, rely on ML-models that can understand a feed of images from a camera. Understanding an image can have several definitions. A stepping stone to complete understanding is the ability to mask out any object given a prompt, even on unfamiliar images. This task is known as promptable segmentation; to take in some prompt (such as a bounding box) and mask out the outline of the object that the prompt is referring to. For example, in an autonomous driving system, this task translates to the ability to segment other cars or obstacles at the pixel level. The prompt for segmentation may originate from another component of the system, such as an object detection model.

Deep Neural Networks (DNNs) are machine learning tools used extensively for segmentation tasks, as their accuracy surpasses previous tools, such as bag-of-features. The capabilities of DNNs have increased steadily over the past decade, as a result of increased model sizes, more computing power, smarter architectures, and larger datasets. EfficientViT-SAM is one such deep neural network capable of promptable segmentation, and the subject model for this study.

Other noteworthy use-cases of an intelligent vision model are those in Extended Reality (XR) — an umbrella term encompassing virtual reality (VR), mixed reality (MR), and augmented reality (AR) [6]. With wearable headsets, a consumer can perceive its surroundings augmented with digital overlays, for example showing a set of virtual persons to hold a virtual meeting. A DNN vision model is deployed in

the headset, with one of its core tasks being to understand the world around the user. The user experience depends directly on the accuracy of the model’s predictions and the latency being unnoticeable. However, achieving high performance with excessive computational power is impractical, as it would necessitate an expensive, heavy device with a short battery life. Instead, it is essential that the model is *efficient* in terms of compute, energy, network bandwidth and memory [56]. This motivates research into vision models that run efficiently on resource-constrained devices. Preferably, models should retain their original accuracy when optimized for efficiency.

There are several ways to make a model more efficient, such as by removing redundant parts of the model (called pruning) or by distilling the knowledge of a large network into a smaller version. Quantization, the subject of this work, is yet another way of compressing the size and accelerating the inference of a model. Quantization targets the weights and the non-linear functions that make up a neural network. To quantize a model’s *weights* is to represent them in fewer bits. Further, a model processing an image (called inference) causes millions, if not billions, of multiplication- and accumulation calculations (MACs). To quantize a model’s *activations* is to reduce the precision of these calculations and their outputs. A quantized model ideally uses less memory at a reduced latency to perform inference. The drawback is that the noise introduced (or, equivalently, the information lost) from quantization causes performance degradation. There is a trade-off to be made between a more efficient or a more accurate model.

We note that different compression methods have been shown to complement each other with impressive results [19, 43, 51, 56, 57, 63], and we encourage practitioners to combine the techniques. This work investigates quantization only. More specifically, we study mixed-precision quantization to INT8 and FP16, which means to quantize some subset of the model to the lower precision and the rest to the baseline precision. We consider FP16 the baseline (as opposed to FP32) since EfficientViT-SAM is already fully deployed in FP16 in the original publication [80]. Note that there are other potential low-precision formats available: FP8, INT16, and INT4 [27]. We exclude FP8, as evidence suggests it is less efficient than INT8 during inference [2]. Additionally, we exclude INT16 because it is not supported by the framework `TensorRT` we use [68]. Lastly, although INT4 has the potential to improve on INT8, we leave this for future work—assuming the mixed-precision using INT8 leads to negligible accuracy degradations.

1.2 Segment Anything Model

The Segment Anything Model (SAM) is a foundation model capable of impressive results when challenged with segmentation of high-resolution images drawn from a previously unseen dataset (zero-shot inference). While highly capable, its image encoder is too heavy to run on resource-constrained devices. With almost 3000G

MAC operations per image inference, SAM processes 11 images per second when run on a single A100 GPU [80], which is a more powerful GPU than any contemporary resource-constrained hardware. As the human eye registers far more than 11 images per second, SAM is not real-time even on the A100.

EfficientViT-SAM [80] is an extension of SAM designed to address the slow image encoder by replacing it with a more efficient architecture (keeping the rest of SAM unchanged). The new architecture is a hybrid design, incorporating architectural components from vision transformers (ViTs) and convolutional neural networks (CNNs). EfficientViT-SAM was developed using several of the previously mentioned compression methods, including sparse operations co-designed for hardware, pruning, knowledge distillation, and quantization to 16-bit floating point (FP16). While it achieves an order of magnitude speed-up without performance loss, it has not yet been quantized to lower precision than FP16. This thesis aims to explore quantization to lower-precision formats, specifically mixed-precision schemes to INT8, to further reduce the latency and size of EfficientViT-SAM.

As EfficientViT-SAM is a novel architecture, it is not trivial that earlier quantization methods transfer well due to both local factors (the characteristics of the novel modules) and global factors (effects that propagate through the network). We anyhow note that most state-of-the-art vision models have been quantized with success, including for CNN-based models [59], for ViT-based models [56], and for some other hybrid models [39]. As this is the first quantization study of EfficientViT-SAM, our work is focused on evaluating what can be achieved with a mixed-precision approach in deployment (as opposed to designing tailored algorithms), and identifying the challenges ahead.

1.3 Research questions

EfficientViT-SAM is substantially more efficient than both the original SAM model [32] and other comparable works [75, 78, 82] that has attempted to accelerate SAM. However, as suggested by Cai et al. [80] (the authors of EfficientViT), quantization of the model is an open question that may lead to further reduction of latency and size. Further, we seek to understand the model’s behaviour under quantization, to identify what parts of the architecture are sensitive or robust to quantization. In detail, our research questions are,

1. RQ1: To what extent can mixed-precision quantization to INT8 reduce the latency and size of EfficientViT-SAM’s image encoder, without causing severe accuracy degradation?
2. RQ2: What layers of EfficientViT-SAM’s image encoder are sensitive or robust when quantized to INT8?

We assess RQ1 by four experiments. First, we profile the latency of the model, to identify opportunities for improvements. We then break down the model into its parts at three different depths (stage, operation type, and layer), and measure the sensitivity of each part. We design several quantization schemes that quantize the parts at the intersection of potential latency improvement and low sensitivity to quantization. We assess RQ2 by drawing insights from the fourth experiment; the layer-wise sensitivity analysis.

1.4 Delimitation

This work investigates only mixed-precision post-training quantization to INT8, limited to uniform, symmetric, and static calibration. In other words, the work does not consider other specialized methods, such as re-parameterization tricks, or methods that require further model training after quantization. Evaluation is limited to bounding box segmentation tasks, using prompts from two sources and evaluated on two datasets. The TensorRT engines were built and evaluated using TensorRT-v8.6.3 on a single RTX 4090. Simulation experiments were conducted on double RTX 3090, using PyTorch v2.0.7 with CUDA v11.7.

To limit the scope in favor of in-depth analysis, the thesis only considers quantization, acknowledging the progress made with other compression techniques such as pruning, as well as the potential of combining compression methods. Other aspects, not in the scope of this thesis, include novel model architecture designs and algorithm-hardware co-design. We refer the interested reader to Menghani [51] for an overview of Efficient Deep Learning, and Shouvo et al. [63] for further reading on acceleration methods specifically for AI at the Edge.

1.5 Thesis Outline

This work is outlined as follows: Chapter 2 gives a theoretical background and discusses related works, Chapter 3 describes the methodology and experiments conducted, Chapter 4 presents and discusses results obtained, and Chapter 5 concludes our findings.

The main contributions of this work are,

1. We provide a detailed survey of the architecture of EfficientViT-SAM’s image encoder, beyond what is presented in the model’s publication. This overview can aid further research into and use of the architecture.
2. We conduct latency analysis of EfficientViT-SAM deployed with TensorRT, identifying slow layers, stages, and operations.

3. We carry out an extensive study on the sensitivity of the model, i.e., we identify what layers, stages, and operations cause more or less accuracy degradation when quantized.
4. We implement a simulation framework that can be used for future research into quantization of EfficientViT-SAM.

1.6 Individual Contributions

This thesis work was conducted as a joint project by the authors, but is examined individually. We therefore outline our individual contributions. Bernard Ryhede Bengtsson built and evaluated TensorRT engines, and thus also conducted the latency measurements and sensitivity experiments by stage and by type. Joel Bengs performed the architecture survey, implemented the simulation framework, and conducted the layer-wise sensitivity experiments. Both students contributed to the analysis and discussion.

2

Background and Related Work

To refresh the basics, we here briefly discuss neural networks and quantization theory. The experienced reader may skip ahead to Section 2.4, where we review the Segment Anything Model. We then extensively survey the architecture of EfficientViT-SAM, and lastly discuss Related Works.

2.1 Neural Networks

Machine Learning, a subset of Artificial Intelligence (AI), is the field concerned with algorithms that learn the characteristics of some data and automate a task or decision-making. Deep Learning is a subfield of ML concerned with algorithms organized in many sequential layers. The basic building block of a neural network is the perceptron [60], defined as

$$y = \sigma \left(\sum_{i=1}^n w_i x_i + b \right)$$

where w is the learnable weight vector, x is the input vector, $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ is the non-linear activation function, b is the bias term, and y is the output to the next layer of the network. A single perceptron can approximate a simple function, and a sufficiently large network of perceptrons can approximate any function. Deep Learning is not a new field, but research and success have taken off in recent years thanks to the introduction of viable training methods coupled with the growth in computing power availability. When training a neural network, one is traversing the non-convex solution space in search for a set of weights w that minimizes some loss function.

Convolutional Neural Networks

The convolutional neural network (CNN) is a special type of DNN targeted at multidimensional inputs where spatial relations are important, such as images [35]. An image can in general be represented by a tensor $X \in \mathbb{R}^{i \times j \times l}$ with three dimensions i (width), j (height), and l (channels). A convolutional layer with kernel size k is then

$$Y = \sigma((X * K) + b_m)$$

where $*$ denotes the convolution operation, $K \in \mathbb{R}^{k \times k \times l \times m}$ is the learnable weight vector (called the kernel), b_m is the bias vector for the filter m , $\sigma : \mathbb{R}^{i \times j \times m} \rightarrow \mathbb{R}^{i \times j \times m}$ is a non-linear activation function, $Y \in \mathbb{R}^{i \times j \times m}$ is the output tensor, and m is the filter index. For each filter m , a separate convolution operation is performed to produce a new channel in the output tensor $Y \in \mathbb{R}^{i \times j \times m}$. Intuitively, a filter scans each part of the input to produce a new representation of that image, a feature map. Convolutions are designed to extract local patterns such as shapes and edges, wherever they are in an image. A single convolution would be able to extract global patterns if the kernel size was as large as the input $(k, k) = (i, j)$, but this would reduce the ability to extract local patterns [3]. In practice, $(k, k) \ll (i, j)$ to benefit local feature extraction, and global understanding is achieved by having many convolutional layers in an hierarchical sequence [35].

The concepts and the architecture of CNNs were introduced in the 1990s [37, 38] and popularized in 2012 by AlexNet [35], the first “deep” CNN utilizing five layers. In the consequent years, CNNs evolved rapidly with several noticeable works [12, 48, 64, 21, 58, 20, 24, 62, 65]. To a varying degree, all of these works have contributed to the design of the convolutional parts of EfficientViT-SAM.

Depth-Wise Separable Convolution Layers

In contemporary models such as EfficientViT, the standard convolution is often replaced with some more effective design, such as the Depth-wise Separable Convolution (DWSC) [62]. The DWSC consists of two sparse operations; one depth-wise convolution (DWC) followed by one point-wise convolution, as illustrated in Figure 2.1. A standard convolution layer involves $ijklm^2$ operations (excluding summations and activations), where i, j, l are the dimensions of the input, m is the output channel count (i.e., the number of filters in the layer) and k is the kernel size of each filter. Note that each variable contributes to the cost by multiplication, something that the DWSC addresses.

First, a DWC is applied to produce an intermediate representation. Each filter of the DWC only operates on one input channel, not across all input channels, using a kernel $K_{\text{depth}} \in \mathbb{R}^{k \times k \times l}$. Note that K_{depth} lacks the dimension m since the operation simply preserves the channel depth. The cost of the depth-wise operation is $ijlk^2$, avoiding multiplication by m .

Second, a point-wise convolution is applied to the intermediate representation. A point-wise convolution has kernel size $(k, k) = (1, 1)$, and computes linear combina-

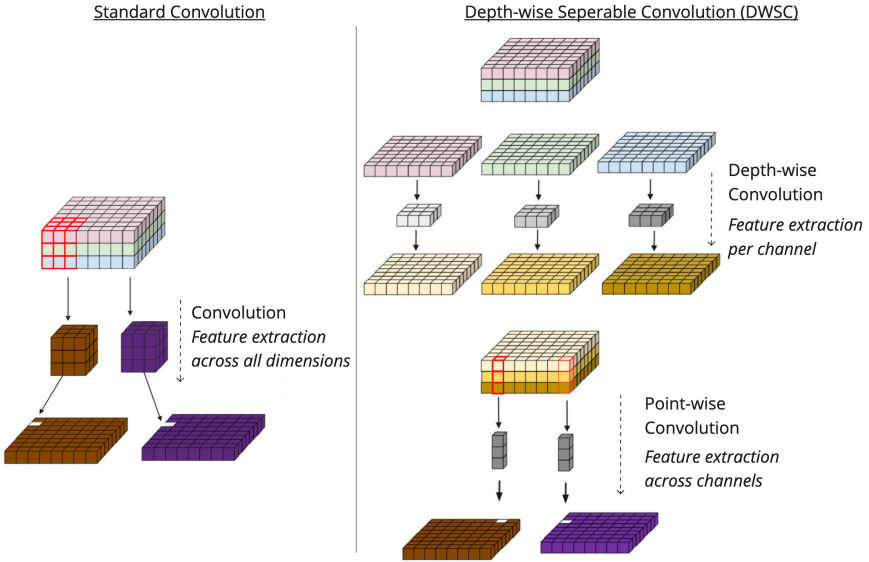


Figure 2.1 *Left:* The standard convolution layer, illustrated with two filters. *Right:* The Depth-wise Seperable Convolution [62], an efficient substitute for the convolution with comparable feature extraction capability. The DWSC consists of two convolutions in sequence. The first operates only on individual channels of the input, not across channels. The second operates only across channels of the intermediate tensor. Depending on dimensions, the number of operations is reduced by up to k^2 , where k is the kernel size [62].

tions *across* channels, but not within a channel. One can use m number of point-wise filters to scale the number of output channels m up or down. As such, point-wise convolutions are often used for dimension expansion or compression. The point-wise operation’s kernel is $K_{point} \in \mathbb{R}^{i \times j \times l \times m \times k^2}$. Since $k^2 = 1^2 = 1$, the computational cost for the point-wise operation is reduced by 9x as compared to a normal convolution, assuming a kernel size $k = 3$.

Replacing a normal convolution with a DWSCnv reduces the total computation cost by factor $k^2/(k^2 + m)$ [62]. As EfficientViT-SAM has $k = 3$ in all DWConvs and a channel depth $m \geq 64$ in stages where the architecture is used, the save in compute is $7.9 - 9x$ as compared to classic convolutions. Empirically, DWSCnvs can be used as drop-in replacements for normal convolutions with only a small reduction in performance [62]. It should however be noted that the reduction of parameter count equals a reduction in expressiveness, which may limit an already small model’s ability to learn [18]. As we will discover, the sparse convolutions are sometimes challenging to quantize [77].

Vision Transformers

An alternative design of computer vision machine learning models is the visual transformer (ViT) introduced in 2021 in [5], inspired by the transformer architecture from large language models [70]. In a vision transformer, a multidimensional input is converted to a 1D input sequence by a patch embedding. The 3D tensor is divided into a fixed number of patches, and each patch is flattened and projected to some higher dimensions space. All embedded patches are fed to the transformer backbone as if they were tokens in a sequence, similar to words in a sentence in the language domain.

Vision transformers have a global context and can establish long-range dependencies, without losing the ability to extract local patterns [5, 56]. A convolution can theoretically have a global context using a large kernel size, but then local pattern extraction is lost. The transformer is based on *self-attention*, which means that the processing of a token (a part of an image) is directly influenced by the all other tokens in the input sequence (the rest of the global image). Each token attends to each and every other token through a matrix multiplication. Learnable parameters (weights) are used to calculate three matrices, Q (query), K (key), and V (value) from the input x . In very simple terms, one can think of the query as holding the question that the transformer is currently trying to answer, of the key as holding the context (how all other tokens relate to the query), and the value as the importance that can focus attention to what matters the most. The attention matrix O is then calculated according to

$$O_i = \sum_{j=1}^N \text{Softmax} \left(\frac{Q_i K_j^T}{\sqrt{d}} \right) V_j, \quad (2.1)$$

where O_i represents the i -th row of the attention matrix O and d is the dimensionality of K . If the dot product $Q_i K_j^T$ is large, the intuition is that token j is relevant for the evaluation of query i . The Softmax function normalizes scores to a probability distribution that sums to one, and V_j further blends global context by a weighted sum. The Softmax self-attention in (2.1) has quadratic complexity to the length of the input sequence, but EfficientViT-SAM reduces the complexity to linear, as detailed in Section 2.5. Several self-attention mechanisms compute in parallel, and their respective attention matrices O are concatenated to form an attention map, which is fed to the next layer in the network. For details on the complete architecture of ViTs, we refer the reader to [5].

2.2 Precision formats

The standardized floating-point formats are defined by the IEEE 754-2019 standard [26]. The binary encoding of a floating-point number is characterized by the following sequence of bits: 1-bit sign field S , W_E -bit exponent field E , and a $(p - 1)$ -bit precision field T , as seen in Figure 2.2 [52].

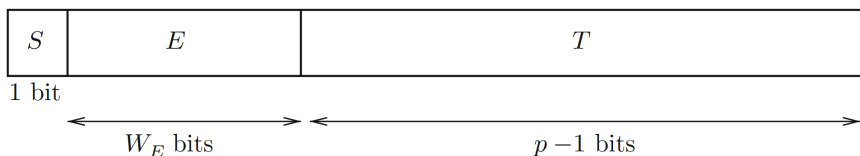


Figure 2.2 Binary encoding of floating-point number [26](©IEEE, 2019, with permission).

The floating-point formats, FP32 and FP16, are characterized by their bit-formats and representable ranges as seen in Figure 2.2 and Table 2.1.

Table 2.1 Range of representable values for FP16 and FP32 formats. Note that values are approximated for easier reading

	Smallest positive	Largest finite
FP16	5.96×10^{-8}	65504
FP32	1.401×10^{-45}	3.403×10^{38}

Table 2.2 IEEE 754 binary formats

	FP16	FP32
storage width	16	32
$p - 1$	10	23
W_e	5	8

Numbers do not necessarily have to be represented using floating-point formats, there are also fixed-point formats. A fixed-point number consists of an integer-part, and a fractional-part. The number of bits allocated to the fractional-part decides the resolution of the fixed-point number [1]. Likewise, the number of bits allocated to the integer-part decides the range of the fixed-point number. In this thesis, we will use the 8-bit fixed-point format referred to as INT8. The placement of the radix-point, the point separating the integer and fractional parts, can be abstracted to a scaling factor s [44].

2.3 Quantization

Quantization is the process of approximating a continuous signal with a discrete signal [11]. Mathematically, it involves mapping a large set Θ , often continuous and infinite, to a smaller set Ω , often discrete and countable. In this thesis work, the large set Θ is the set of numbers representable by the 32-bit floating-point format, and the small set Ω is the set of numbers representable by the fixed-point format INT8. In other words, $\Omega \subset \Theta \subset \mathbb{R}$. The mapping $\Theta \rightarrow \Omega$ is defined by a quantization operator Q , controlled by a set of parameters. The process of selecting parameter values is called *calibration* and can be performed at different granularity levels. In general, each layer of a neural network has a dedicated quantization operator Q .

The Quantization Operator

Adapted from Gholami et al. [11], the quantization operator $Q : \Theta \rightarrow \Omega$ maps a signal $x \in \Theta \subset \mathbb{R}$ (or equivalently a set of numbers x , a tensor x , or a vector x) to Ω according to

$$Q(x) = \text{Int}\left(\frac{x}{S}\right) - Z, \quad (2.2)$$

where S is a real-valued scaling factor, Z is an integer zero point, and Int is a rounding operator that rounds to the nearest integer. Z provides flexibility to adjust the mapping of zero, while S can compress or expand the mapping. Note that (2.2) has evenly spaced steps in the goal domain, called *uniform* quantization. The scaling factor S is defined by

$$S = \frac{\beta - \alpha}{2^b - 1}, \quad (2.3)$$

where $[\alpha, \beta]$ denotes the bounded range of Θ that will be represented by the limited and discrete set Ω , and b is the number of bits in the goal representation ($b = 8$ in the case of INT8). Any input value x smaller than α or larger than β will be mapped by $Q(x)$ to a value outside Ω and therefore clipped to the set boundary, thus $[\alpha, \beta]$ is called the clipping range. This clipping (or projection) onto Ω is not included in the mathematical definitions, but occurs in practice. Information in Θ outside $[\alpha, \beta]$ will be lost after quantization. Having encoded the signal x with $Q(x)$, it is of interest to recover the original signal. The inverse operation

$$Q^{-1}(\tilde{x}) = S(\tilde{x}) + Z \quad (2.4)$$

is called dequantization, where $\tilde{x}$ is the previously quantized representation, and the parameters S and Z need to be the same values as when quantizing the range with (2.2). Information lost due to the rounding operator or the clipping range is non-recoverable.

Restricting the case to uniform quantization (i.e., evenly distributed levels in the smaller set Ω), there are three design choices: range width (affecting scaling factor S), range placement (using zero-point Z), and precision format (which directly decides

goal range and goal resolution). A narrower range $[\alpha, \beta]$ decreases loss for signal within the range (since resolution is increased), but completely erases information outside the range due to clipping. The range is selected by some algorithm during calibration as an optimization problem. The range placement is *symmetric* if $\alpha = -\beta$ and $Z = 0$, in which case the zero point in Θ is mapped to zero in Ω . Setting $\alpha \neq -\beta$ results in *asymmetric* quantization, in which case the zero point Z may be set to the middle of the calibration range. Symmetric quantization has the benefit of reduced complexity. Lastly, a practitioner must choose the precision format of Ω . A smaller goal representation enables greater compression but with more information loss. In this work, INT8 is used as goal representation, only symmetric quantization is deployed, and the clipping range is selected by calibration methods as discussed further in Section 3.1.

Calibration of Optimal Scaling Factor

Each model layer has a dedicated operator Q . Calibration refers to calculating the optimal scale factor S to be used in Q (and zero-point Z , but $Z = 0$ in this work due to symmetric quantization). Quantization aims to compress information from a source, defined by its probability density function, into the fewest bits while allowing minimal loss during retrieval.[15]. The loss can be quantified in several ways. An intuitive method is to minimize the Euclidean norm between two tensors; $\min_Q ||Q(x) - x||^2$. A more sophisticated alternative is to optimize the parameters such that the Kullback–Leibler (KL) divergence between the distributions before or after quantization is minimized [36], called *entropy* calibration. Yet another method is to calculate an exponential moving average (EMA) of the observed distributions [28]. In either case, it is an optimization problem.

Quantization is applied to both the weights of each neural network layer, and the operators (the activations) of each layer. After model training, the model’s weights are fixed and known. The optimal scale factor for each layer’s weights can thus be calculated before deployment, without uncertainty according to the chosen calibration strategy, void of any data samples. The probability density function of each layer’s activations, on the other hand, is not known before inference and will be unique for each input. The optimal scaling factor can therefore not be calculated without data. To pass sample data through the network, and estimate the probability density function at each activation, is called *static* calibration. It is static because the optimal scaling factor is estimated before deployment, and then fixed. To perform the calibration during inference is called *dynamic* calibration. The dynamic approach calculates and applies individual loss-minimizing quantization for each data sample at inference. This approach is more accurate but is seldom used due to the heavy computational burden [11]. With static calibration, about 100 mini-batches are sufficient to observe and estimate the distribution of the activation functions [34]. Static calibration is used in this work, i.e., weights are calibrated independent of any data samples (as is always the case), and activations are calibrated with sample data from the training

dataset before deployment. During deployment, all quantization parameters are fixed.

Granularity of Quantization

In a convolutional neural network, each layer contains several filters, and each filter holds several kernels. A granularity decision naturally arises; at what level of detail should calibration occur? That is, should individual kernels have individual scale parameters (called sub-channel-wise calibration), or should they be grouped by filters (channel-wise), or grouped by layer (layer-wise)? Recall that calibration depends on the statistics of the set of weights and the distribution of the output. More granularity causes more overhead computation but reduces the information loss. Empirical investigations [34] have found that, in general, layer-wise is preferable for activations while channel-wise is preferable for weights as a default. Recall that weight calibration always occurs before deployment (as weights are fixed and known), so the extra compute is often acceptable. Sub-channel-wise is rarely worth the overhead, since kernels within a channel tend to have similar distributions [11]. These are general guidelines; if e.g. activations exhibit channel-wise fluctuating distributions, a practitioner should move from layer-wise to channel-wise to combat this. Several advanced quantization methods observe channel-wise fluctuations and implement channel-wise calibration in hardware-efficient ways, such as [45] and [47].

Uniform and Non-Uniform Quantization

The quantization operator Q as defined in (2.2) is uniform (the levels are evenly distributed) and thus simple, but the method is sub-optimal concerning minimizing information loss. In neural networks, the distributions of weights and activations are rarely uniform, and there is headroom to reduce the information loss by assigning more resolution bits to selected range parts. Non-uniform quantization uses various methods to achieve lower loss through clever quantization steps. $Q(x) = \Delta_i$, if $x \in [\lambda_i, \lambda_{i+1})$ is a general definition of non-uniform quantization, where $\Delta_i \in \Delta$ are discrete levels in the goal range and $\lambda_i \in \lambda$ are the quantization steps. Ideally, the distribution of λ follows the distribution of numbers inside the neural network during inference. If a non-uniform distribution is observed when studying a network’s layer, one can design a new quantizer that fits the observed distribution better, such as letting λ model a logarithmic distribution. The Softmax activation function used in vision transformers is an example where non-uniform quantization can be and has been applied successfully [47, 41].

Uniform quantization is the default in most quantization frameworks [11], because while non-uniform quantization is preferable from a theoretical perspective, it is challenging to implement in hardware efficiently. In this work, only uniform quantization is implemented due to its simplicity and wide hardware support.

Quantization-Aware Training

Quantization methods can be broadly divided into two fields. Post-Training Quantization (PTQ) is to apply quantization to a pre-trained model, and Quantization-Aware Training (QAT) train a model using quantized weights and operators. Often, a model is pre-trained in floating point and then further fine-tuned with QAT [11]. During QAT, (some version of) back-propagation is performed with both passes in floating point precision, and the updated weights are quantized to the goal precision after each update [11]. As characteristics of the loss-landscape are altered by quantization [10], QAT is a way to find a new optimum in the non-smooth landscape. Gradient-based optimizers work only to a degree, but refined methods are often capable of recovering all performance lost from quantization [59, 29, 8]. The drawback to QAT is the cost of re-training, which is not negligible. In many use-cases, QAT is not feasible due to a lack of time, training data, compute resources or funds. This is especially true when one seeks to deploy a pre-trained open-source model like EfficientViT-SAM to resource-constrained devices. Therefore this thesis focuses on Post-Training Quantization only.

Hardware Acceleration

The performance gains of quantization stems from the fixed-point accelerators within the graphical processing unit (GPU) that enables faster computations and smaller memory transfers from the shorter bit-width. The basic operation in a neural network is the Multiply-Accumulate (MAC) operation. In short, it consists of a matrix multiplication between the activations and weights, where the subsequent result matrix is row-wise accumulated into a vector. An *accelerator* refers to a specialized hardware design meant to accelerate a specific operation [53], such as MAC. On a high level, the accelerator can be described by the schematic overview in Figure 2.3. Weights and activations are loaded from memory, multiplied, accumulated, and, depending on the preferred output format, quantized. In Figure 2.3, the precision formats used throughout the operation are written in the figure next to the corresponding operation. The precision format can be specified at five stages in the accelerator: activation format, weight format, calculation format, accumulator format, and output format. A two-fold reduction in the size of the memory transfers is achieved if the activations, weights, and outputs are quantized from FP16 to INT8. This is a trivial conclusion, since 8-bit is half the size of 16-bit. Further, the matrix multiplication is significantly faster when performed in fixed-point as compared to its floating-point counterpart [23]. Nagel et al. [53] states that using a fixed-point accelerator with INT8 reduces the computational cost by a factor of 8 compared to floating-point accelerator with FP16. Note that the accumulator has to use an INT32 format instead of INT8 to avoid overflow, and therefore the quantization step is necessary to again reduce the output format to INT8.

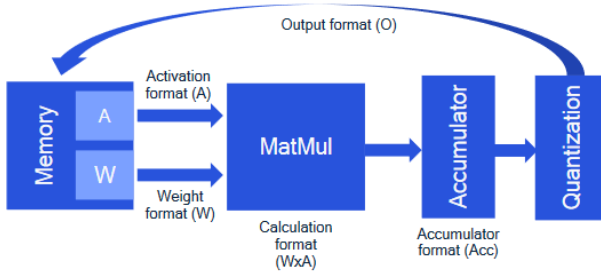


Figure 2.3 Data Flow Diagram for Matrix Multiplication in Hardware Accelerators. The diagram illustrates the flow from memory where activation (A) and weight (W) matrices are stored, through matrix multiplication, accumulation (Acc), and quantization stages, resulting in the output format (O). By [53], licensed under CC BY 4.0 <https://creativecommons.org/licenses/by/4.0/deed.en>. No modifications were made.

When quantizing a neural network, one can target any of the formats in Figure 2.3. To reduce the precision of the weight format is called *weight-only* quantization, and its benefit is a reduction of the model’s size. To reduce the precision for all stages in Figure 2.3 is called *weight-and-activation* quantization. If the chosen precision is fixed-point for all stages, it is sometimes called *integer-only* quantization. *Mixed precision* quantization is to mix precision format used throughout different layers of a network.

2.4 Segment Anything

The “Segment Anything” project, by Kirillov et al. [32], aims to advance the field of segmentation by developing a foundation model. A foundation model is trained extensively on a broad dataset, with focus on generalizability and adaptability to new downstream tasks. In other words, a foundation model’s objective is to serve as the basis for the development of specialized models, reducing the cost of training them and the time to develop them. This initiative led to the creation of SA-1B, the largest dataset for segmentation at the time of writing this thesis report [32]. Additionally, the project has developed a new model architecture that supports a new promptable segmentation task. The task can be tailored to downstream tasks, given an appropriate prompt.

SA-1B Dataset

The most significant contribution of [32] is the SA-1B dataset, with 400x more masks than the second-largest segmentation dataset, OpenImages [33]. A mask is a pixel-granular portion of the image that represents an object. An image and its masks can be seen in Figure 2.4. According to [32], previous studies have found that a foundation model’s ability to generalize to any domain-task scales with the size of

the dataset, so a large high-quality dataset is essential. With 1.1 billion masks from 11 million images, SA-1B enables large-scale training for segmentation foundation models, as already done in the domain of large language models.



Figure 2.4 A sample image from SA-1B dataset. *Left:* The raw image. *Right:* The same image with mask annotations. Adapted from [32]. Modifications include resizing.

Promptable Segmentation Task

A promptable segmentation task allows the foundational model to adapt to downstream tasks by engineering a suitable prompt. Kirillov et al. [32] define the model prompts as a coordinate with a flag to indicate whether the object is in the background or foreground, a bounding box, free-form text, or a dense mask. In our case, we will constrain our investigation to points and boxes. The goal of the task is to output a valid mask, given an image and a prompt. In this context, “valid” means that the model should output a reasonable mask when the prompt is ambiguous, i.e., could be specifying one of multiple potential objects or parts of objects. As an example, the task can be to segment different parts of a car by providing point or box prompts, as seen in Figure 2.5. Note that all prompts in Figure 2.5 could also refer to the full car. An invalid mask is a mask that fails to reasonably encapsulate any of the objects or object-parts that the prompt could be referring to.

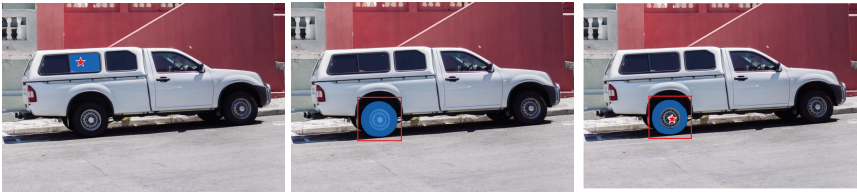


Figure 2.5 Three examples of prompts provided on the *GitHub* page belonging to [32]. The content is licensed under the Apache License, Version 2.0. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>. *Left:* A point prompt. *Middle:* A box prompt. *Right:* A box prompt and a point prompt.

Architecture of SAM

The SAM model consists of three distinct parts, the image encoder, the prompt encoder, and the mask decoder, as seen in Figure 2.6. The image encoder takes an image as input, and outputs an embedding of that image. The prompt encoder takes one or more prompts as input, and outputs an embedding for each prompt. The mask decoder takes both the embedding from the image encoder and the prompt embedding from the prompt encoder, and outputs a set of masks alongside a ranking of the quality of the masks. Since prompts can be ambiguous, multiple masks are given as output with their respective predicted scores. Kirillov et al. [32] state that mask decoder and prompt encoder are implemented to perform efficiently; with a pre-computed image embedding the combined latency is $\sim 50\text{ms}$ when run in a web browser, on a CPU. However, computing the image embedding is the latency bottleneck for SAM, and thus makes run-time performance poor on less powerful machines.

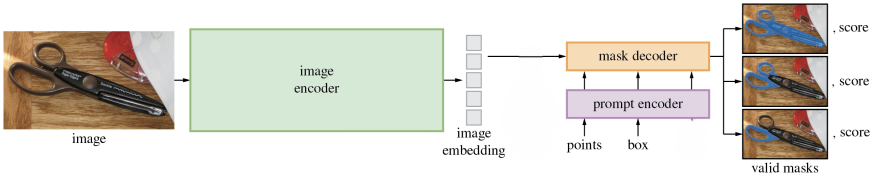


Figure 2.6 SAM architecture illustrated provided on the GitHub page belonging to [32]. The content is licensed under the Apache License, Version 2.0. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>.

2.5 EfficientViT-SAM

Several attempts have been made to accelerate the heavy image encoder of the Segment Anything Model. The most successful such acceleration to date is EfficientViT-SAM [80], a family of models where the image encoder has been replaced by a slightly tweaked version of the EfficientViT¹ [3] architecture, explained in detail below. The relatively lightweight prompt encoder and mask decoder in Figure 2.6 are kept unaltered from the original SAM foundation model. EfficientViT-SAM comes in five variants; L0, L1, L2, XL0, and XL1. Hereinafter, we will use the version names to refer to each variant of the entire model. The number of MACs needed per image is reduced from SAM-ViT-H’s benchmark by one or two orders of magnitude as outlined in Table 2.3, for the largest and smallest version of EfficientViT-SAM respectively. As per Figure 2.7, EfficientViT-SAM is 16.5x faster than the largest

¹ The name EfficientViT is contested, as [49] used the same name for an unrelated accelerated ViT-based model.

version of SAM with improved performance (XL1), and 48.9x faster with matched performance (L2). In the interest of acceleration, the smallest version (L0) offers 84x speed-up with a moderate accuracy loss. We have verified the state-of-the-art accuracy levels on zero-shot COCO mAP.

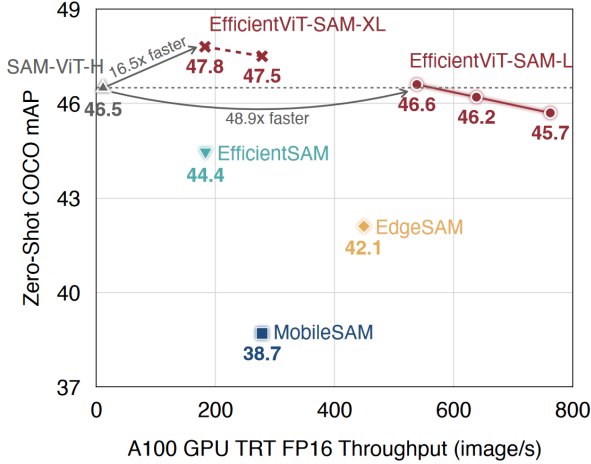


Figure 2.7 Speed and performance of EfficientViT-SAM (red), compared to other models (mixed colors) and original SAM (grey). By [80], licensed under CC BY 4.0 <https://creativecommons.org/licenses/by/4.0/deed.en>. No modifications were made.

Table 2.3 Footprint, speed and accuracy comparison. Note that EfficientViT-SAM has a smaller footprint than SAM-ViT-H, with increased speed and retained accuracy. Throughput measured on a single NVIDIA A100 GPU with TensorRT, quantized to FP16. By [80], licensed under CC BY 4.0 <https://creativecommons.org/licenses/by/4.0/deed.en>. Modifications were made: three rows excluded.

	#Params(M)	#MACs(G)	Throughput (image/s)	COCO mAP
SAM-ViT-H [32]	641.1	2973	11	46.5
EfficientViT-SAM-L0	34.8	35	762	45.7
EfficientViT-SAM-L1	47.7	49	638	46.2
EfficientViT-SAM-L2	61.3	69	538	46.6
EfficientViT-SAM-XL0	117.0	185	278	47.5
EfficientViT-SAM-XL1	203.3	322	182	47.8

The authors of EfficientViT-SAM [3] and EfficientViT [80] are concise when presenting the architecture of the model, omitting any detailed explanation of the architecture. A detailed understanding of the architecture will aid the investigation of the model's sensitivity to quantization, and was an essential step to implementing the simulation framework used in this work. We therefore survey the architecture of the image encoder, including the building blocks and design practices that EfficientViT-SAM inherits from earlier models. We also briefly review the novel ReLU Linear Attention module, which is key to EfficientViT-SAMs drastic state-of-the-art performance.

Architecture of EfficientViT-SAM's image encoder

An overview of the architecture for the XL-series and the L-series is provided in Figure 2.8. The XL-series is divided into a total of seven stages. Stage 0 through stage 3 are convolution-based, as is the Neck. Stage 4 and 5 are attention-based, but include convolution operations. The L-series architecture drops the last attention stage (stage 5), alters the design in stage 3, and reduces depth and width overall. The three model variants in the L-series (L0, L1, and L2) share the same architecture, the only difference is model depth (number of layers). The two variants XL0 and XL1, on the other hand, are not directly comparable with each other since XL0 omits the ResBlock in the first stage.

Residual Connections and ResBlocks

The ResBlock, present only in the first stage of EfficientViT-SAM, is the basic building block from the ResNet family [21]. A (non-bottleneck) ResBlock is two normal 3×3 convolutions and a shortcut connection from the block's input to its output. Via this "residual" connection, a copy of the input is forwarded and element-wise added to the output of the convolutions. It has been shown by [21] that residual connections in CNNs improve learning convergence and enable greater model depth. Residual connections are parameter-free and add negligible computational complexity. Adding residual connections to CNNs is a widely adopted practice. In addition to the ResBlock in the first stage, EfficientViT-SAM adopts residual connections for almost all blocks, excluding the first block of each stage. This exclusion choice is not motivated by [3] or [80].

Intuitively, given two models where one's architecture is a subset of the other's, the larger model could imitate the smaller one by setting the extra layers to the identity mapping $F(x) = x$. Thus, the larger model should achieve at least the same accuracy as the smaller model. Empirically, the inventors of the residual connection [21] found the opposite; excessively deep networks performed worse than their shallower counterparts. Their hypothesis was that optimizers struggled to find the needed identity mappings in the non-convex solution space. By introducing residual connections, they reformulated the problem to a form that, empirically, optimizers performed better on. Adapted from [21], given that a residual free block should map input to output according to some underlying mapping $H(x)$, the block with a residual

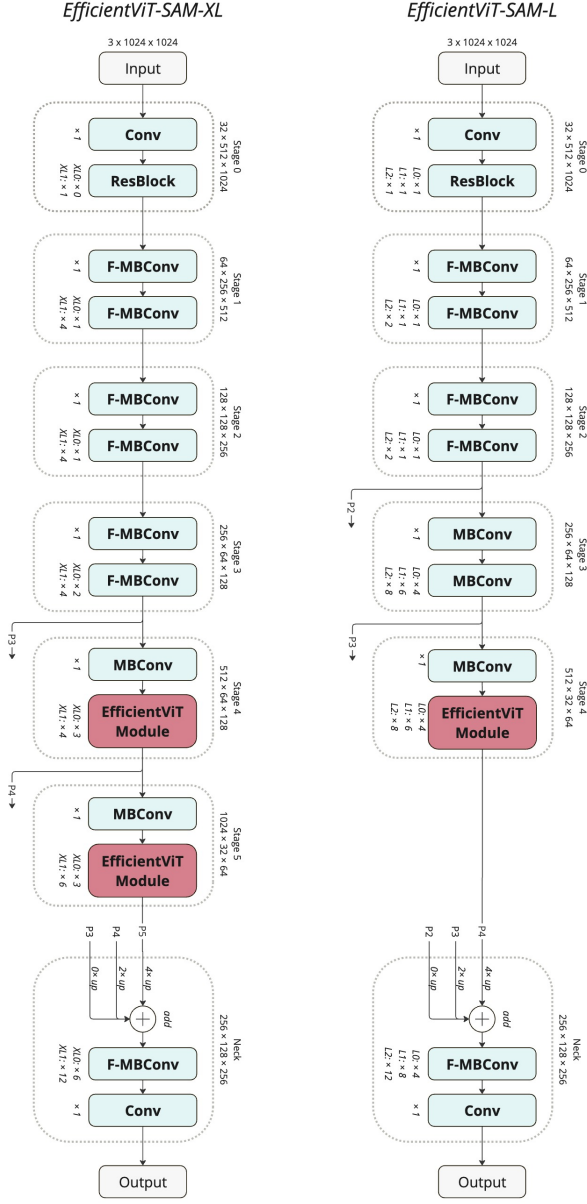


Figure 2.8 Macro architecture of EfficientViT-SAM’s image encoder. Fused-MBConv from [18] is the fused version of the Mobile Inverted Convolution block (MBConv) from [62]. The EfficientViT Module is a novel accelerated vision transformer module from [3]. ResBlock is the simple residual block from [21]. The depth of each block is noted below it. The tensor dimension at the output of each stage is noted above the stage, on the format *channels* $\times$ *height* $\times$ *width*. *Top*: the L-series (L0, L1, L2). *Bottom*: the XL-series (XL0, XL1). Compared to XL1, Large models omit Stage 5 to reduce footprint, Large models reduce depth in several stages, and they use MBConvs instead of Fused-MBConvs in stage 3. Note that the *only* difference between the three models in the L-series (L0, L1, L2) is their block depth. This is an extended version of the overview presented in [80]. For detailed architectures, refer to Figure 2.10 (XL), Figure 2.11 (L), and Figure 2.9 (the EfficientViT Module).

connection instead searches for $F(x) := H(x) + x$. To find the identity mapping is then to find $H(x) = 0$, instead of $H(x) = x$. Using residual blocks, ResNet could continue increasing accuracy by increasing model depth beyond other state-of-the-art models at the time.

Mobile Inverted Convolutions

For efficiency gains, ResNet further evolved the ResBlock into a bottleneck design, using three convolutions. A point-wise convolution (i.e., kernel size 1×1) would *reduce* the dimensions, the following 3×3 convolution would do feature extraction, and another point-wise convolution would *expand* dimensions again. The clear benefit of the design is the reduction in computation in the heavy 3×3 layer (which is a function of the size of the input tensor). A residual connection avoids all three layers, connecting back at the output with addition. While a successful architecture (1st place in the competition ILSVRC 2015), it was further evolved when [62] introduced the Mobile Inverted Convolution Block with a linear bottleneck, denoted MBConv. The design utilizes an "inverted bottleneck" design, i.e., the channel count is *expanded* in the block's first layer, kept constant in the middle layer(s), and *compressed* in the output layer. Similar to the ResBlock, pointwise convolutions handle expansion and compression. The ratio of expansion and compression is called *expansion ratio*, and should be greater than 1. Setting the expansion ratio to zero gives an identity mapping (via the residual connection), and setting the expansion ratio less than one gives a non-inverted block. Since the shortcut is between shallow-channel layers instead of wide-channel layers, it is aptly named an inverted residual. Further, a depth-wise convolution replaces the normal convolution in the middle layer. Joined with the following point-wise convolution, these form a depth-wise separable convolution block, reducing computations by 7.9-9x [62].

The inverted residual block improves the memory efficiency of the architecture if the intermediate operations are performed in sequence on hardware (thus eliminating the need for memory-write of the large intermediate tensors). A copy of the input tensor still has to be stored in memory when passed through the residual shortcut, before addition to the output tensor. As both these tensors are compressed along the channel dimension, they are cheap to store compared to normal convolution blocks. This is especially important for mobile applications, where memory access is often a limiting factor.

The MBConv is unique in that it separates a network's *expressiveness* from its *capacity*. The intermediate layer introduces flexibility and non-linearity, i.e., expressiveness, while the bottleneck layers reduce memory requirements but also limit the capacity of the network. EfficientViT-SAM uses MBConv extensively in the attention stages, one MBConv is inserted between every attention layer. In the L-series, the entire third stage consists of MBConv. As we will discover, the MBConv is sensitive to quantization.

Fused Mobile Inverted Convolution Block

The MBConv is efficient in theory, but in practice, the Fused Mobile Inverted Convolution (Fused-MBConv) from [18] can be even more performing. The depth-wise convolution in the MBConv is sparse and thus does not benefit the full extent of specialized hardware. The first two layers of the MBConv can be merged into a full convolution (retaining an expansion ratio greater than one) to solve this problem. The merged convolution, which has a 3×3 -kernel in EfficientViT-SAM, has lower inference latency under some conditions despite its large parameter count (missing out on the 7.9-9x saving) [18]. In applications that are latency-bound but not memory-bound, this is a good trade-off, since more parameters translate to greater learning capacity without impacting latency.

Realized gains are conditional on input/output tensor shapes, location in the model, and hardware mapping, so it is not obvious which MBConv blocks should be fused when designing a new model. Neural architecture search (to search a model space for an optimal architecture) can be used to find a suitable combination of MBConv and Fused-MBConv blocks. It has been shown empirically by [18] and [66] that Fused-MBConv blocks are preferable in a model's early layers, but not in late layers. This is intuitive since the cost of Fused-MBConv grows as channel width increases later in the model. EfficientViT-SAM adopts the Fused-MBConv in its early stages and the Neck, and the non-fused version in later stages. Additionally, EfficientViT-SAM mimics earlier efficient works [18, 66], and removes the swish non-linearity and the Squeeze-and-Excite blocks from [25] that can otherwise be found in similar architectures, because they lack implementation for specialized hardware [18].

The Neck

The Neck of the image encoder is responsible for turning the feature maps into an image encoding interpretable by the mask decoder. It receives the feature maps from the last three stages as input, matching their dimension using up-sample layers and element-wise addition. The up-sample layers are preceded by one convolutional kernel for each input. The majority of the Neck consists of Fused-MBConv blocks with residual connections. The image encoder's last layer is a point-wise convolution without normalization or activation. Having reviewed the basic blocks of the model, we now examine the novel attention modules and the motivation behind the new design.

Global Receptive Field and Multi-Scale Learning

Models that perform well on challenging vision tasks must exhibit two traits; multiscale learning and a global receptive field [3]. Multiscale learning is the ability to learn from small and large input features. A global receptive field is the ability of a neuron in the network to be connected to the entire input image and thus understand how any feature is affected by the global context. Both skills are intuitively important for understanding an image. A model that could only identify large objects would

perform poorly when the object of interest is small relative to the image. A model without a global receptive field would fail to understand an object that takes up the whole image.

There are various tools to achieve global receptive field in a vision model, transformers being the popular choice. Recall that the attention mechanism in transformers by design incorporates the entire input sequence, and will learn to attend to the important image features anywhere in the sequence [5]. This is a trait from the original task transformers were designed for; natural language translation. In language, a word at the end of a sentence may alter the meaning of a word in the beginning. However, due to the Softmax function being used in transformers, the computational complexity grows $\mathcal{O}(n^2)$ to the number of input tokens [3]. Large receptive fields can also be achieved with transformer-free CNN models using large kernel operations, such as SegNeXt [16], but the large kernels are inefficient on existing hardware [3]. EfficientViT mitigates the inefficiency of transformers and avoids large kernel operations.

EfficientViT proposes a novel attention module where Softmax attention is substituted for ReLU linear attention (where *linear* highlights the linear complexity, not any linear function). This reduces the computational complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$ while maintaining the important global receptive field from the attention design. However, substituting Softmax causes a loss of non-linearity in attention scores. Thus the design fails to extract localized information and to learn from low-resolution features (i.e., small patterns). EfficientViT introduces a small-kernel convolution scaling layer to mitigate this. This convolution aggregates nearby tokens before the ReLU linear attention layer, enabling multi-scale learning and sufficient non-linearity [3]. Intuitively, the self-attention layer receives a zoomed-in version of the feature map concatenated to the original feature map. The zoom enables the self-attention to attend to small features. The novel attention mechanism is at the heart of the EfficientViT Module, explained next.

EfficientViT Module: Multi-scale ReLU Linear Attention

Illustrated in Figure 2.9, the EfficientViT Module is an evolution of the vanilla vision transformers. It inherits the linear layer that generate the Q, K, and V matrices from the input, and a linear projection at the output. The multi-scale operation consists of two convolution kernels without normalization or activation functions. The EfficientViT Module also includes a trailing MBConv block, unlike ViT.

At the core of the novel module, ReLU is inserted into the Softmax function of the attention layer. It is, however, a more complex swap than just a substitution, since vanilla ReLU can not calculate the probabilities as Softmax is designed to do. Instead, the ReLU function is injected inside the general definition of the Softmax function. An explanation follows.

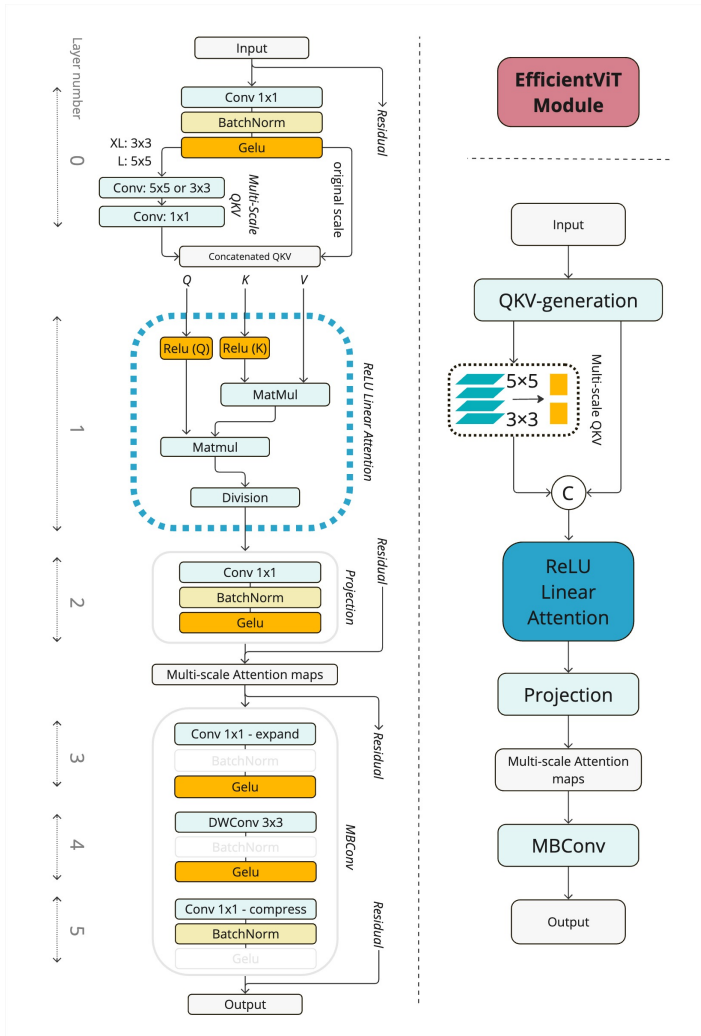


Figure 2.9 Architecture of the EfficientViT attention module. As with any transformer [70], [5], the design uses learnable parameters to generate three vectors from the input, called Query (Q), Key (K), and Value (V). These are fed through the self-attention mechanism with a novel ReLU implementation. To introduce additional non-linearity, two convolutions perform feature extraction before the attention block. Only one scaling convolution is used per model; the L-family uses kernel size 5×5 , and the XL-family uses kernel size 3×3 . Operations that are defined in PyTorch modules but which are instantiated to None at model creation are transparent. The layer numbers are used for identification in layer-wise experiments. This illustration extends the overview presented in [80].

Recall that the vanilla vision transformer with Softmax attention computes an attention matrix O according to

$$O_i = \sum_{j=1}^N \text{Softmax} \left(\frac{Q_i K_j^T}{\sqrt{d}} \right) V_j, \quad (2.5)$$

where Q_i , K_j , and V_j are vectors stemming from the input to the self-attention layer and the layer’s weight matrix. O_i represents the i -th row of the attention matrix O . Adopted from [3] and simplified, the classic transformer in (2.5) can be expressed in a more general form as

$$O_i = \sum_{j=1}^N \frac{\text{Sim}(Q_i, K_j)}{\sum_{j=1}^N \text{Sim}(Q_i, K_j)} V_j \quad (2.6)$$

where, $\text{Sim}(Q, K)$ is the similarity function, i.e., a function that computes some form of similarity measure between two vectors. The choice of similarity function is arbitrary, and $\text{Sim}(Q, K) = \exp(\frac{QK^T}{\sqrt{d}})$ is the common version which makes (2.6) equal to (2.5). The authors [3] replaces the common version with $\text{Sim}(Q, K) = \text{ReLU}(Q)\text{ReLU}(K)^T$, where $\text{ReLU}(x) = \max(0, x)$. (2.6) then becomes

$$O_i = \sum_{j=1}^N \frac{\text{ReLU}(Q_i)\text{ReLU}(K_j)^T}{\sum_{j=1}^N \text{ReLU}(Q_i)\text{ReLU}(K_j)^T} V_j. \quad (2.7)$$

This ReLU-based version of Softmax in (2.7) can be reformulated with equivalence to the form

$$O_i = \frac{\text{ReLU}(Q_i)(\sum_{j=1}^N \text{ReLU}(K_j)^T V_j)}{\text{ReLU}(Q_i)(\sum_{j=1}^N \text{ReLU}(K_j)^T)}. \quad (2.8)$$

Please refer to [3] for details. Note that the summations over $j \in 1, \dots, N$ are independent of the running variable i . As such, they can be calculated once and then reused as constants for each query Q_i . This is the key to the complexity reduction from $\mathcal{O}(n^2) \rightarrow \mathcal{O}(n)$. The new choice of similarity function is more linear than the common choice $\text{Sim}(Q, K) = \exp(\frac{QK^T}{\sqrt{d}})$. As non-linearity is fundamental in a network’s ability to approximate non-linear functions, this is an issue. The problem is mitigated by the multiscale convolution layer, as explained above. We note that ReLU is a quantization-friendly function, especially compared to Softmax. (2.8) will be referenced in Section 3.1 when discussing how simulated quantization is implemented in the ReLU linear attention module.

Architecture summary

In summary, EfficientViT-SAM’s image encoder uses convolutions in the first stages and self-attention in the later stages, all to produce three feature maps at different scales. The neck processes all three feature maps to create a final image encoding. A detailed view is presented in Figure 2.10 and, Figure 2.11 respectively.

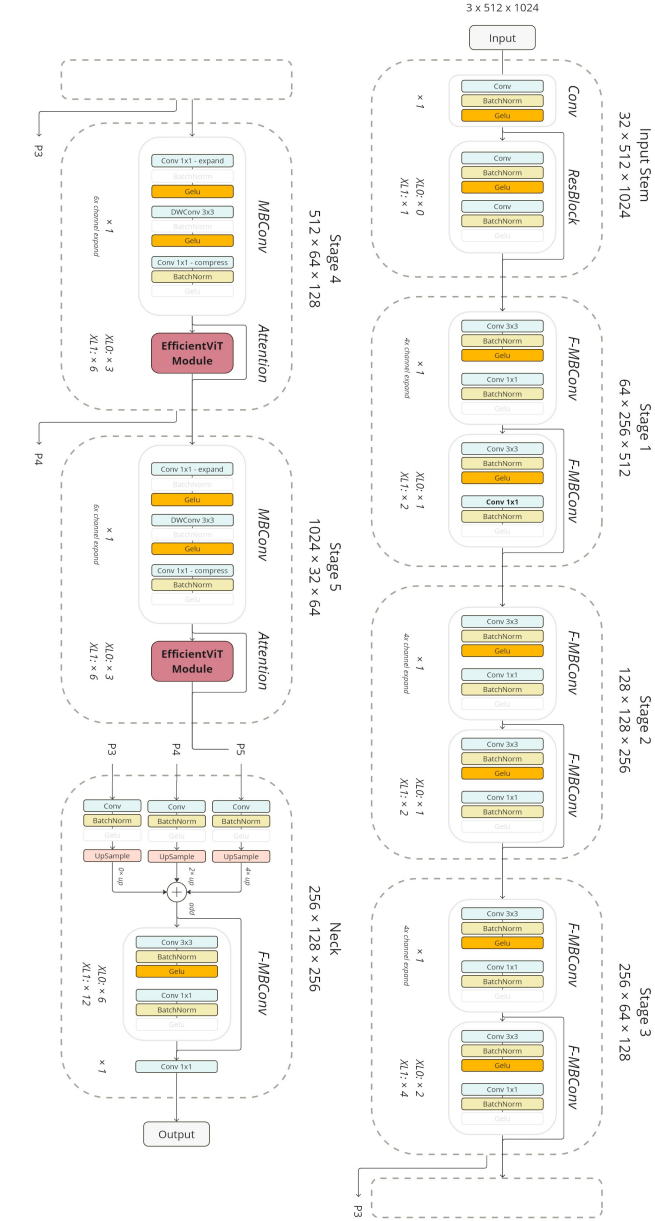


Figure 2.10 Detailed architecture of EfficientViT-SAM-XL’s image encoder. Derived from the code implementation in [80], and intended as a support for understanding said code. Operations defined in PyTorch modules but instantiated to None at model creation are transparent. Residual connections are marked with shortcuts. Note the absence of residual connections between stages and around the first block of each stage. The EfficientViT Module (which wraps the transformers) is detailed in Figure 2.9.

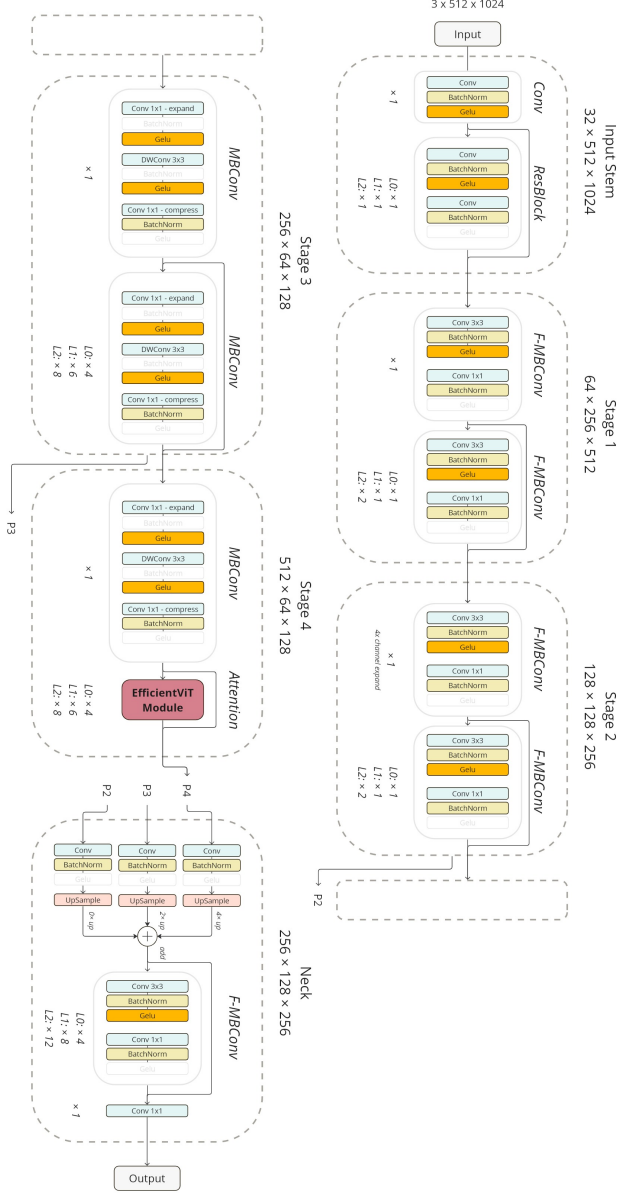


Figure 2.11 Detailed architecture of EfficientViT-SAM-L’s image encoder. Derived from the code implementation in [80], and intended as a support for understanding said code. Operations defined in PyTorch modules but instantiated to None at model creation are transparent. Residual connections are marked with shortcuts. Note the absence of residual connections between stages and around the first block of each stage. The EfficientViT Module (which wraps the transformers) is detailed in Figure 2.9.

Training of EfficientViT-SAM

To highlight the use of knowledge distillation, we briefly outline the training of EfficientViT-SAM. To train EfficientViT for adoption as an image encoder, the authors in [80] utilize knowledge distillation and end-to-end training from random initialization. Knowledge distillation in supervised learning is to leverage the image embedding predicted by the larger SAM-ViT-H model as a secondary ground truth for what the image embedding the *image encoder* should output, alongside the dataset’s label [22]. The loss function in training is the sum of two loss functions; one over each label. Naturally, SA-1B holds no labels for the intermediate image embedding, so the prompt encoder and mask decoder are needed to translate the image embedding to a mask output. EfficientViT-SAM’s image encoder is trained for two epochs on the full SA-1B dataset, alternating between box-prompt and point-prompt. Note that the prompt encoder and the mask decoder are not subject to further training. For more training details, we refer the reader to [80]. Like SAM, EfficientViT-SAM is tasked with zero-shot image segmentation, so no fine-tuning to the evaluation dataset is allowed.

2.6 Related Work

There are many approaches to make DNNs more efficient. While quantization is the focus of this work, this chapter provides brief overview of the methods presented by [11, 43, 51, 56, 63], after which follows a review of related works specifically in the field of quantization.

Related Model Compression Works

Naturally, the model’s core building blocks can be re-designed architecturally with efficiency in mind, e.g. by substituting convolution kernels for sparse versions that reduce the number of computations required [24]. Recently, efficient designs of neural networks have been found *using* neural networks, known as Automated Machine Learning (AutoML) or Neural Architecture Search (NAS) [7].

A continuation is to design network architectures specifically for a target hardware. As it is becoming increasingly difficult to make integrated circuits more powerful by making transistors smaller, innovation in hardware turns to task-specific designs. When hardware and model are co-designed, otherwise unattainable bottlenecks such as memory bandwidth can be removed.

Knowledge Distillation involves the distillation of a large (and thus accurate) model’s knowledge into a smaller (and thus more efficient) model. In this compression method by [22], the intermediate representations in the large network (the teacher) can be used as a target when training the smaller model (the student). Specifically, the output of the teacher’s second-to-last layer is often used as a ‘soft label’ rich in information, in compliment to the dataset’s original ‘hard’ label that binary

assigns labels to data points. Training against both labels, the student can learn to reproduce the internal dynamics of the teacher, i.e. the teacher’s knowledge.

Pruning can improve the efficiency of an existing model without redesigning its fundamentals. To prune a model is to remove nodes, connections or entire layers, saving memory and compute. The criterion for removing nodes is often their impact on the model’s performance, referred to as their *saliency*. Removing nodes with low saliency can reduce model size, but the resulting sparse matrix operations are difficult to accelerate on hardware. An alternative approach is to prune entire layers to avoid creating a sparse network.

Related Quantization Works

Quantization has been extensively applied to deep neural networks of both CNN architectures [59] and ViT architectures [42, 41, 47, 76]. In a wider context, quantization is often used together with the above-mentioned compression techniques [19] to accelerate DNNs in many domains, including natural language processing [45, 67, 74, 83]. The common denominator is that a practitioner looking to quantize a novel model is faced with two challenges. First, any novel operation may be sensitive to quantization in itself, e.g. by having a skewed output distribution. Second, sensitive cross-layer dependencies may be at play, causing quantization errors to propagate through a network in new ways [41, 13, 77]. Applying earlier methods to a novel model is not trivial. Both of these challenges are present when looking to quantize EfficientViT-SAM. The model has new operations (the ReLU linear attention module) and new architecture in a new context. Keeping in mind that methods are not straightforwardly transferable, we here review quantization methods for CNNs and ViTs respectively.

Quantization of CNNs

Gluska et al. [13] perform a layer-wise accuracy profiling of four models that suffer large accuracy degradation under quantization, ResNets, MobileNets (V1 & V2). They find that the majority of degradation can be attributed to one or a few "ultra-sensitive" layers. By finding an alternate calibration scheme tailored at these sensitive layers, [13] is able to recover virtually all lost accuracy. Wu et al. [73] apply a similar layer-wise sensitivity analysis as [13] for additional models; MobileNetV1, EfficientNet, and Transformer. Their work verifies the method and findings from [13] in both identifying the sensitive layers and recovering the lost accuracy by attending to them. Whereas Gluska et al. apply a tailored quantization scheme for each layer, Wu et al. simply retain the sensitive layers in full precision (i.e., a mixed-precision scheme). EfficientViT-SAM inherits several design choices in the convolution layers from ResNets, MobileNets, and EfficientNet, so we hypothesize that EfficientViT-SAM might also contain ultra-sensitive layers. We adopt the methodology proposed by Gluska et al. and Wu et al. to verify this hypothesis, and find that the depth-wise separable convolution from [62] is ultra-sensitive in EfficientViT-SAM. We therefore review the following relevant work in particular.

EfficientViT-SAM uses the MBConv block presented in MobileNetV2 [62]. The block inherits the depthwise separable convolution (DWSCConv) from MobileNetV1, which (initially) was difficult to quantize. The sensitivity of MobileNetV1 has been attributed to the characteristics of the DWSCConv operation [77]. Comparing CNNs built with or without DWSCConv operations, [77] observed abnormally fluctuating dynamic ranges within each DWSCConv layer, meaning that layer-wise quantization would introduce a large amount of noise. Further, [77] observed a large accumulation of quantization errors from layer to layer in the networks with DWSCConv present. While [77] does not present any solution for how to quantize the design successfully, their work indicates that the design may be sensitive to quantization also in a hybrid model, which is in line with our findings.

Other works on the quantization of CNNs are numerous and revolve around applying the concepts described in Section 2.3 in efficient ways [11, 59]. Since convolution architectures are diverse in their designs, methods rarely target a specific problematic non-linear operation, as is the case for ViTs. Quantization of classic CNN models (such as ResNet, MobileNet, SqueezeNet, and DenseNet) is refined to the level where full accuracy is retained with quantization to less 4 bits, both for weight-only and integer-only [59]. Often this performance is achieved with quantization-aware training (QAT), as in [29] and [8]. For a comprehensive review of methods, we refer the reader to [59].

Quantization of ViTs

The vanilla vision transformer [5] holds two key non-linear operations; Softmax and LayerNorm. The output distribution of Softmax is by nature difficult to quantize with a uniform quantizer, as the bulk of vales is almost zero, and a few (important) outliers are closer to one. The output of LayerNorm has a peculiar behavior where the dynamic range of channels within a layer varies greatly [42], so channel-wise calibration is needed (which is costly and thus infeasible). Quantization works on ViTs often target one or both of these issues. However, neither operation is present in EfficientViT-SAM, as ReLU and BatchNorm are used respectively.

Methods that target Softmax include RepQViT [42], a method that deploys a bit-shift operation to efficiently use a non-uniform $\log \sqrt{2}$ quantizer. Recall that the challenge with non-uniform quantizers is to implement them efficiently on hardware. The quantizer better captures the Softmax distribution, while also being efficient on hardware thanks to the bit-shift operation. Further, PTQ4ViT [76] introduced a 'twin uniform strategy' in which two independent quantizers target two groups of values. For Softmax, the method uses one calibration range for small and large values, respectively. Another method, FQ-ViT by Lin et al. [47] designs a novel transformer design called Log-Int-Softmax, which is more robust to quantization as far as 4 bits. Notably, FQ-ViT is implemented using a custom framework for simulated quantization, which we adopt when performing simulated quantization experiments.

RepQViT [42] also introduces a re-parameterization trick targeting the Layer-

Norm where the weights are changed off-line to mitigate the channel-wise variation that a LayerNorm exhibits. During inference, calibration can be applied layer-wise, resulting in the acceptable inference overhead coupled with the performance of channel-wise calibration. FQ-ViT [47] also targets the weights of the LayerNorm, but is restricted to only scale each channel with an individual 2-factor which accelerates the scaling using bit-shift. As with RepQViT, this scheme performs as well as quantizing channel-wise even though the entire layer can share quantization parameters S and Z .

There are also more general works. NoisyQuant [50] makes the distributions more uniform by adding noise to activations, which flatten peaks. It can be put on top of other quantization methods for better performance with little overhead. Assuming the input from a previous layer is X , NoisyQuant samples a bias N_b from a Uniform distribution, adds it to the activation X before quantization, and then subtracts the same value N_b from $Q(X)$ after the quantization. The input to the next operation is then $Q(X + N_b) - N_b$, instead of just $Q(X)$. The authors showed empirically that NoisyQuant made the output after the next operation more closely resemble the full precision version when applied to the vanilla vision transformer model from [5].

I-ViT [41] is the most complete ViT-quantization work. It achieves full integer-only quantization of common ViT-based (non-hybrid) models. Linear operations are quantized with dyadic arithmetic (which avoids integer division using bit-shift [11]), and the non-linear operations are approximated with novel shift-based operations (ShiftMax, ShiftGELU, and I-LayerNorm). I-ViT is targeted at and only evaluated on pure ViT models, not hybrid models. Further, I-ViT and its predecessors have not yet been extended to the complex vision task of dense semantic segmentation.

Related Works on EfficientViT

Current research on the EfficientViT architecture is limited to benchmark baselines and two applications in downstream tasks. Feng et al. [9] use the EfficientViT module in LRTransDet, a real-time SAR ship-detection model. Zheng et al. [81] enhance the YOLOv5 model with EfficientViT modules for a real-time helmet-less detection system for lift truck operators. Additionally, Wang et al. [71] benchmark a new compression technique, TinySaver, with EfficientViT among other ViT models. In this thesis, we will continue in a similar vein as [71], and explore how compression techniques can be applied to EfficientViT to improve performance.

3

Method

3.1 Methodology Overview

This thesis explores to what extent quantization can reduce the latency and the size of EfficientViT-SAM’s image encoder without causing significant accuracy degradation (RQ1). It also seeks insights into what layers of the architecture are more sensitive or robust to quantization (RQ2).

We assess RQ1 by four experiments. First, we identify opportunities for improvements by profiling the latency of the model. Second, we measure the sensitivity of each model *stage*, i.e., how much accuracy is degraded when applying quantization to only that stage. Third, we measure the sensitivity of different operation *types*. From these insights, we design quantization schemes that quantize those parts that both exhibit a potential for latency improvement and low sensitivity to quantization. These three experiments are carried out in TensorRT. The fourth experiment further breaks down the model into individual *layers* and measures sensitivity. This granularity enables us to find patterns and insights otherwise not observable, from which we design additional schemes (addressing RQ1), and address RQ2 directly. An overview is presented in Figure 3.1.

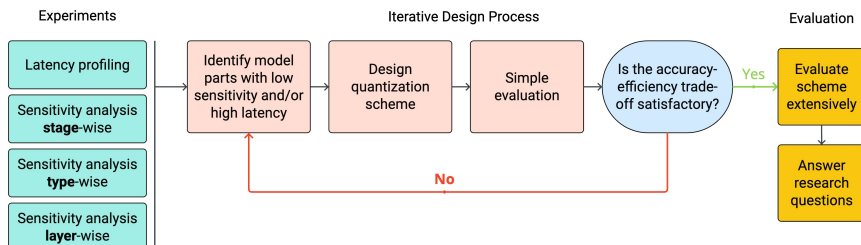


Figure 3.1 Overview of the methodology. Four experiments and a systematic search for mixed-precision quantization schemes address research question RQ1. The insights from all experiments, but primarily the layer-wise analysis, address RQ2.

The layer-wise experiment is implemented in simulation, not deployment, because in simulation we could iterate faster on ideas within the time constraints of this thesis work. Further, TensorRT is a commercial tool that performs an array of optimizations beyond quantization, such as fusing of layers and approximation of operators, which clouds the results obtained. In comparison, simulation provides a platform with more reproducible context where effects can be isolated. Additionally, our simulation framework can be used in future works to evaluate methods not yet available in TensorRT. However, the results showed that there are discrepancies between the two approaches, namely that accuracy results differ greatly. This is discussed in Section 4.3.

Configuration of Quantization Experiments

A quantization scheme involves several configuration choices, e.g. the choice of calibration method, precision format, and granularity of calibration. As the search space grows exponentially with each dimension, we adopt a fixed set of configurations according to the limitations of TensorRT, an engine compiler optimized for NVIDIA GPUs. Table 3.1 outlines our configurations.

This thesis defines performance as the chosen accuracy metrics we use during the evaluation. Efficiency can refer to either inference latency or memory size. We deploy static quantization of activations (as is preferable for reduced latency), and weights are calibrated without the need for calibration samples since their distribution is deterministic.

For latency experiments, TensorRT necessitates channel-wise uniform symmetric quantization for weights and tensor-wise (i.e. layer-wise) uniform symmetric quantization for activations [27]. This granularity level is the one recommended by [73, 11] as a reasonable trade-off between efficiency and accuracy. Any granularity can be simulated in our simulation framework (soon to be described), but we apply the same choice as in TensorRT for comparability.

Calibration involves running a set of representative input data through the network, collecting distributions of the activations of all layers, and finally calculating the optimal calibration parameters. In TensorRT, we employ the entropy method for calibration, utilizing 4500 images from the SA-1B dataset and a calibration batch size of 500. This is the dataset on which both SAM and EfficientViT-SAM are trained. The calibration aims to minimize information loss by utilizing KL-divergence [73]. Large batches are preferred because they prevent the significant expansion of the quantized histogram range, which can occur if a batch contains values exceeding the histogram’s absolute maximum. Calibration in simulated experiments follows the same outline, but substitutes KL-divergence for the exponential moving average method (EMA) [28] as a measure of information loss, due to EMA being readily implemented in the simulation framework adopted from [47].

Table 3.1 Configuration used in quantization experiments.

	TensorRT	Simulation
Approach	Mixed-precision	Mixed-precision
Quantization target	Weights & Activations	Weights & Activations
Precision format	FP16/INT8	FP32/INT8
Weight calibration	Static	Static
Activation calibration	Static, 4500 samples from SA-1B	Static, 4500 samples from SA-1B
Calibration loss metric	KL-divergence	EMA
Granularity for weights	Channel-wise	Channel-wise
Granularity for activations	Layer-wise	Layer-wise
Uniform or non-uniform	Uniform	Uniform
Symmetric or asymmetric	Symmetric	Symmetric

Latency Analysis with TensorRT

It is crucial to systematically profile the network layers to develop an effective mixed-precision strategy. Using the profiling tool, `trt-engine-explorer` we assessed the latency of each layer, measuring it in milliseconds [69]. The results were then visualized to identify which layers were critical to network performance, guiding our development of a mixed-precision scheme.

We began our latency analysis by profiling each layer type and stage across all models, as detailed for the L and XL model series separately. Recall that model L0, L1, and L2 are different depths of the same architecture, with the same width (i.e. the same number of channels), and that XL0 and XL1 have almost the same relation to each other. For a refresher on the specific architectures, see Section 2.5. Utilizing this profiling data, we calculated the percentage latencies for each stage, which are measured in milliseconds. In Figure 3.2 and Figure 3.3, the graphs illustrate the percentage latency contribution per stage. For the XL models, the latency contribution is more evenly split between the stages. On the other hand, for the L models, the latency contribution is more skewed towards the later stages. Importantly, all stages contribute to some degree to the latency of the model, so all stages are subject for the subsequent sensitivity analysis.

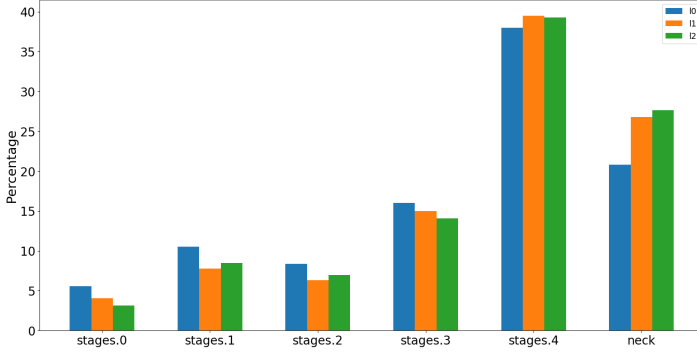


Figure 3.2 Percentage contribution to latency per stage for L-series

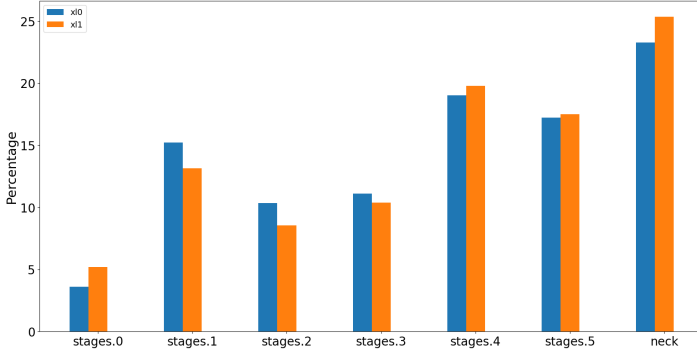


Figure 3.3 Percentage contribution to latency per stage for XL-series

TensorRT identifies eight different layer types of EfficientViT-SAM: convolution, point-wise, reformat, matrix multiply, slice, shuffle, and resize. Analyzing them in Figure 3.4, we see that the number of layers per type has this descending order: convolution, point-wise, matrix multiply, shuffle, slice, reformat, and resize. A point-wise layer in TensorRT refers most often to an activation layer, but can also be any other element-wise operation. A reformat layer either changes the precision of the input or changes the shape of the input. This happens because of precision constraints created by the user or that the compiler has found a faster kernel implementation, but that requires a reformat such that input or output types remain the same for the next layer [79]. When applying mixed-precision quantization in TensorRT, the number of reformat layers will increase due to this.

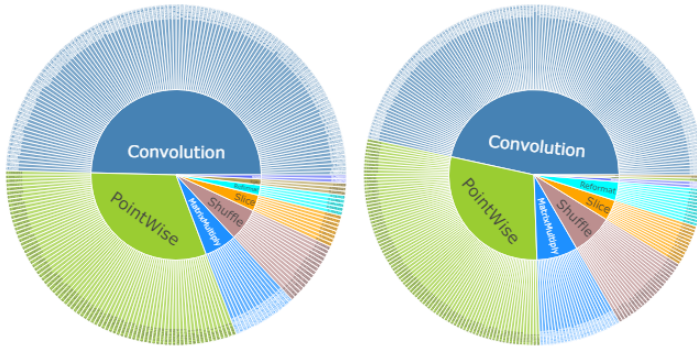


Figure 3.4 Distribution of number of layers per type for L and XL series. *Left: L2, Right: XL1.* Color-coding in order of size: Dark Blue (Convolution), Green (PointWise), Light Blue (MatrixMultiply), Dark Brown (Shuffle), Orange (Slice), Turquoise (Reformat), Purple (Resize), Light Brown (kgen)

Convolutions are the largest contributors to latency across both model series, accounting for 76 – 78% of total latency. Point-wise layers follow as the next significant contributors, with 17 – 19% of the latency, while other layer types each account for less than 2%. This is evident across Figure 3.5 and Figure 3.6, where color coding used corresponds to those in Figure 3.4. Consequently, quantizing layers that contribute minimally to latency is likely to have only a small effect on the model’s total latency. This leads us to focus our efforts only on the slowest layers: convolutional, and point-wise layers.

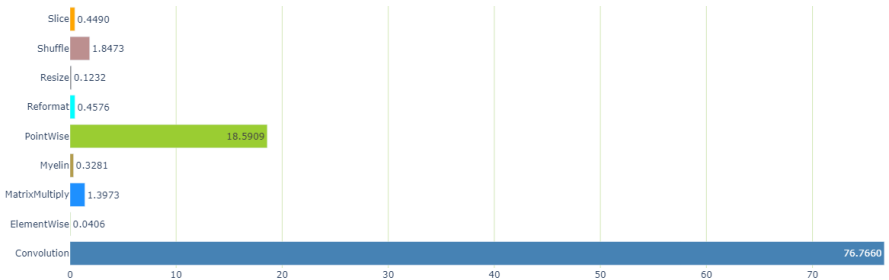


Figure 3.5 Percentage-wise latency contribution per layer for XL1. The x-axis range is percentage contribution of type to total latency. Descending order of latency contributors is convolutions (76.77%), point-wise (18.59%), and all other types (>2%). Color-coding is the same as in Figure 3.4

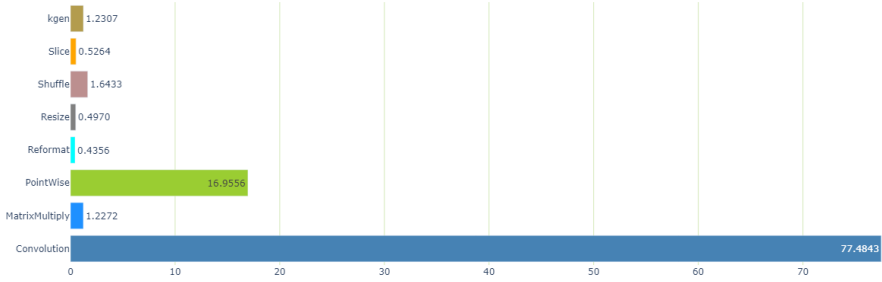


Figure 3.6 Percentage-wise latency contribution per type for L2. The x-axis range is percentage contribution of type to total latency. Descending order of latency contributors is convolutions (77.48%), point-wise (16.96%), and all other types (>2%). Color-coding is the same as in Figure 3.4

Simulated Layer-Wise Accuracy Analysis

Inspired by the methodologies proposed by Gluska et al. [13] and Wu et al. [73], and driven by the hypothesis that EfficientViT-SAM may contain a set of layers that are ultra-sensitive to quantization, we perform a layer-wise accuracy profiling of L0, L2, and XL1 (omitting the other two variants to reduce the scope). Both these works suggest quantizing each layer in isolation with the purpose of identifying sensitive layers of the model that cause large performance degradation. Having found such layers, we intend to follow the lead of Wu et al. [73] and design a mixed-precision quantization scheme that protects the ultra-sensitive layers (i.e., a mixed-precision scheme). This analysis also addresses research question number two: *What layers of EfficientViT-SAM’s image encoder are sensitive or robust when quantized to INT8?*

In deployment, quantization is performed with some software tool, such as NVIDIA’s commercial tool TensorRT. A researcher may also simulate quantization behavior without deploying the model, by artificially compressing and decompressing the tensors in the model’s computational graph. One can use available simulation frameworks such as the PyTorch Quantization API¹. We instead opt for a custom implementation, due to uncertainty about the maturity of existing simulation frameworks (the PyTorch Quantization API was in Beta release at the time of study). Lin et al. [47], the authors of the FQ-ViT quantization method presented in Section 2.6, designed a framework for simulated quantization which we adopt for these experiments. We now present the framework in detail.

¹ Documentation available at: <https://pytorch.org/docs/stable/quantization-support.html>

The simulation applies an operator F to a tensor x , which uses both the quantizer Q from (2.2) and its inverse operation Q^{-1} from (2.4) to introduce the simulated quantization noise to the tensor. F is defined as

$$F(x) = Q^{-1}(Q(x)) = S(\text{Int}(x/S) - Z) + Z, \quad (3.1)$$

where S is the scale factor and Z is the zero point. In practice, F takes an FP32 tensor x , maps each number in x to one of the 256 equidistant steps in INT8, and then uses the same scale factor and zero point to map the INT8 tensor to the FP32 domain. The information content of the output tensor $\tilde{x}$ is reduced to the equivalent of INT8, while the datatype is compatible with the floating-point arithmetic in the next operation of the network.

As the original implementation by Lin et al. [47] contains neither convolutions nor the novel ReLU Linear Attention layer, we must decide on where to apply quantizers F in EfficientViT-SAM. The pattern in [47], as well as in other works [11, 31, 67], is to apply quantizers in between every operation in the computational graph of a model. This is the pattern that guide our implementation.

Assume a general layer L takes x as input from the previous layer and computes the output y as

$$y = \sigma(w \cdot x) \quad (3.2)$$

where σ is an activation function, w is the weights of L , and $\cdot$ is the scalar product. In literature, both x and y may be referred to as *activations*, but we separate notation for clarity. To simulate weight-only quantization of a layer L , a quantizer F_w is inserted according to

$$y = \sigma(F_w(w) \cdot x), \quad (3.3)$$

i.e., the weight vector is quantized before being used in the operation. To simulate integer-only quantization of the layer L , three additional quantizers F_a , F_b , and F_i are introduced according to

$$y = F_a(\sigma(F_b(F_w(w) \cdot F_i(x))))), \quad (3.4)$$

such that F_i quantize the incoming tensor x from the previous layer, F_b quantize the intermediate tensor after the scalar product but before σ , and F_a quantize the output of the layer L . The input quantizer F_i is not necessary if the previous layer's output in the neural network is quantized, as is always the case in FQ-ViT [47]. We include F_i , since we intend to simulate mixed-precision schemes, under which one can not assume that the input x was quantized in the previous layer. Further, if a layer L holds several operations $\sigma_1, \sigma_2, \dots$, then several intermediate quantizers $F_{b1}, F_{b2}, \dots$ are necessary. This is the case for the ReLU Linear Attention layer, which was presented in (2.8) in the survey of the model architecture. With simulated quantization, (2.8) becomes (3.5)

$$O = F \left(\frac{F(F(\text{ReLU}(F(Q)))F(F(\text{ReLU}(F(K))^T)F(V)))}{F(F(\text{ReLU}(F(Q)))F(\text{ReLU}(F(K))^T))} \right). \quad (3.5)$$

where sub-notation and summation has been omitted for clarity. The implementation of (3.5) is illustrated in Figure 3.7. The implementation of (3.4) to the model’s convolution layer is illustrated in Figure 3.8. The figures also highlight the baseline FP32 case and the desired integer-only case in INT8 which the simulation is approximating. As per the survey of the model’s architecture in Section 2.5, all operations of the model are implemented in PyTorch using only these two layer types, aggregated into the various block types. Quantizing convolution and attention layers therefore attends to the entire architecture of the image encoder.

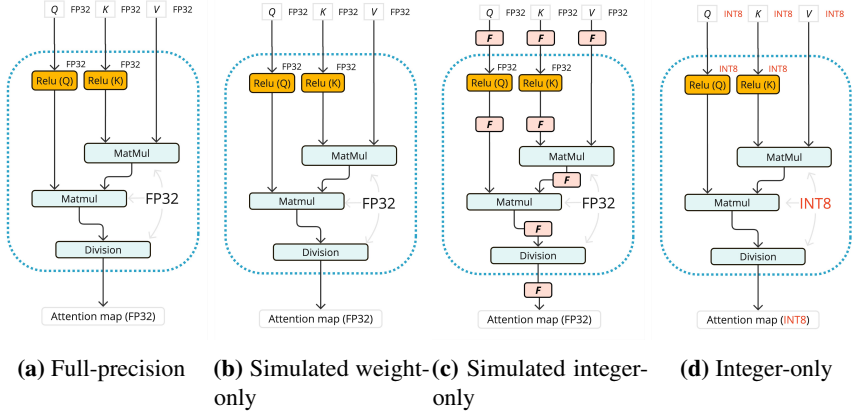


Figure 3.7 Comparison of quantization schemes for the ReLU Linear Attention module from [3]. Simulation. Operators F are inserted to introduce noise. Note that there are no learnable parameters inside the self-attention module, i.e., no weights to quantize. The incoming tensors Q , K , and V are the results of a convolution layer, Figure 3.8.

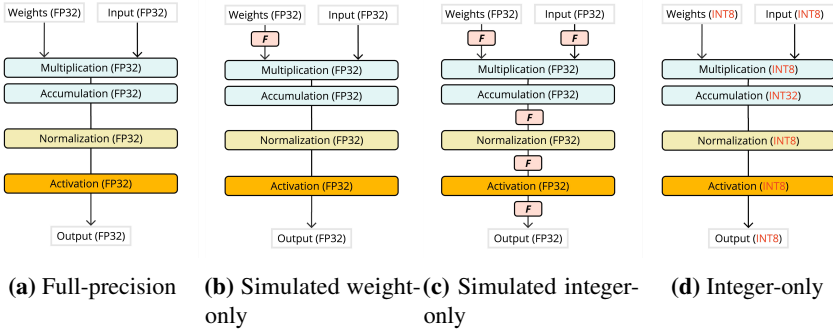


Figure 3.8 Comparison of quantization schemes for the convolutional layer. Simulation. Operators F are inserted to introduce noise. Multiply-Accumulate is one operation in practice, therefore no operator F is inserted between them.

In simulation, it is not meaningful to measure a model’s latency due to the extensive overhead of computation caused by F , and since no actual latency-reduction is achieved when the operations are carried out in full precision with noise introduced. Instead, we only measure various forms of accuracy in simulation, and calculate the theoretical size reduction that would stem from quantizing the weights. Further, FP32 is used as the baseline floating-point format in simulation, not FP16. Layer-wise sensitivity analysis is performed for L0, L2, and XL1, and only for integer-only schemes.

3.2 Experimental Setup

In line with previous benchmarks on the Segment Anything task, we tested our models on zero-shot tasks evaluated on two datasets. "Zero-shot" means that the models are evaluated on previously unseen tasks and datasets [32], in contrast to just an unseen part of the same dataset. The purpose is to evaluate the model’s ability to transfer its foundational knowledge to new tasks and datasets.

Datasets

The datasets used were COCO, and LVIS. Common Objects in Context, or COCO, made by Microsoft, is an instance segmentation dataset with a validation dataset of 5000 annotated images [46]. Instance segmentation refers to the segmentation and classification of a specific instance of an object. However, our aim is not to classify instances, but to improve the segmentations. Large Vocabulary Instance Segmentation, or LVIS, inherits the same task and images [17]. Nonetheless, the annotations are different from COCO.

While we benchmark on COCO, note that any calibration occurs on SA-1B. For zero-shot tasks, the model may not be trained, fine-tuned, or calibrated on any part of the evaluation dataset.

Evaluation Metrics

The metrics used are Mean of Intersection over Union (mIoU), and Average Precision (AP). The task of segmentation is to determine what pixels belong to an object. We can evaluate our model’s ability by calculating the pixel-wise correspondences with the ground-truth segmentation. The metric mIoU measures the mean overlap between the ground-truth mask and the predicted mask across all classes [72]. Let A and B be two masks. The Intersection over Union (IoU) is defined as,

$$\text{IoU}(A, B) = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives} + \text{False Negatives}} \quad (3.6)$$

where True Positives are the correctly included pixels, False Positives are the incorrectly included pixels, and False Negatives are the missed pixels. It returns a value between 0 and 1, where 0 means that there is no overlap, and 1 means there is

perfect overlap. Mean Intersection over Union (mIoU) is then simply the mean IoU over the number of classes,

$$\text{mIoU} = \frac{1}{N} \sum_{i=1}^N \text{IoU}(A_i, B_i) \quad (3.7)$$

where N is the number of classes and $\text{IoU}(A_i, B_i)$ is the IoU for the i -th class.

Average Precision measures the robustness of segmentation models by calculating the precision and recall across multiple IoU thresholds. We first define precision as,

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \quad (3.8)$$

and recall as,

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \quad (3.9)$$

A correct prediction, in this context, is a mask with an Intersection over Union (IoU) over a threshold T . Precision and recall are calculated over an interval of k thresholds: 0.5, 0.55, 0.6, ..., 0.95, to avoid being misled by a poorly chosen T .

Average Precision is then defined as,

$$\text{AP} = \frac{\sum_{k=1}^n (\text{Precision}_k \times \text{rel}(k))}{N} \quad (3.10)$$

where Precision_k is the precision at the k -th threshold, $\text{rel}(k)$ is an indicator function that is 1 if the item at the threshold k is relevant, and N is the total number of relevant items. The mean Average Precision (mAP) is to take the mean over the number of classes, as was the case for mIoU. We further report mIoU and AP for small, medium and large objects separately for evaluation of mixed-precision schemes. This highlights if a model is unable to segment objects of different sizes.

Efficiency Metrics

We evaluate the efficiency of the network based on two metrics: memory-size, and latency. Quantization lowers the storage requirement for the weights by representing them in a reduced bit-precision as explained earlier. During inference, three things are spent on: memory transfer of input from the host to GPU, GPU execution time, memory transfer of output from GPU to host. Since we do not quantize input or output tensors, the duration of their memory transfers will not change. Therefore, we define latency as the average number of milliseconds (ms) spent during the GPU execution time.

Latency measurements are not by default deterministic, and can thus lead to performance measures that are unstable [4]. The source of non-determinism is a floating clock frequency inside the GPU. One can lock it, to ensure consistent measurements, but that is worse on average. According to [14], there is no standard

regarding locking or not locking the clock frequency. They also point to the issues with choosing a suitable clock frequency. During latency measurements, we ensured variance in the clock frequency during the measurements were in the single percent points before we reported the results.

Box-Prompted Segmentation Task

We evaluate box-prompted segmentation on the COCO and LVIS dataset, with ground-truth boxes and detected boxes as prompts. We use detected boxes from ViTDet [40], i.e., boxes that are the result of an object detection model finding objects in the images. Using detected boxes instead of ground-truth boxes is meant to better simulate real-world applications where the SAM model is part of a larger model pipeline where several models work together on their respective task [32]. To measure accuracy on the detected boxes, we report Average Precision for small, medium, and large bounding boxes as well as the mean over all sizes. For the ground-truth boxes, we report accuracy using mIoU instead of AP.

Error Sources

We discuss various sources of error that could skew our results. These errors are common when conducting experiments involving complex combinations of software and hardware. Specifically, we have identified two primary sources of error: issues within the COCO Dataset and hardware stability.

COCO Dataset

As discussed by Kirillov et al. [32], the segmentation masks in the COCO dataset are generally coarse. This increases the risk of qualitatively worse-performing models achieving similar results to other models on quantitative benchmarks. Their study found that SAM’s predicted segmentations are generally preferred over COCO’s based on a human survey. To mitigate this, we also benchmark our results on the LVIS dataset and perform a qualitative study of the results.

In Figure 3.9, includes a imprecise segmentation mask. Consider the task to be to segment the tennis player. Qualitatively, we see that all masks capture the complete object. However, the ground-truth mask in the left image includes the space between the tennis player’s arms, while our EfficientViT-SAM model does not capture this. This discrepancy is a concern when comparing our quantized models to the ground-truth. Therefore, we include a qualitative results section in our presentation to address this issue, enabling better comparison across models.

Hardware stability

Retaining a stable and consistent environment during building, and evaluation, is essential to good performance on the specific hardware. The latency measurements were made on a GeForce RTX 4090 Graphics Card from NVIDIA using TensorRT-v8.6.3. Repeatedly changing clock frequencies can lead to volatile latency measurements if the GPU reaches the maximum power consumption [14]. Our pre-

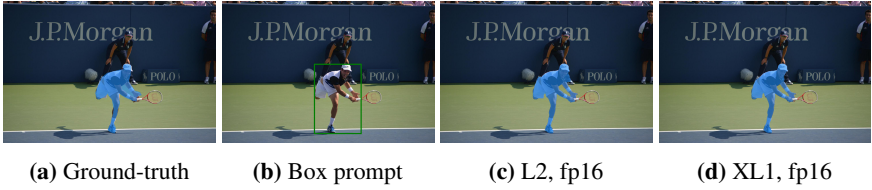


Figure 3.9 Comparison of the ground-truth mask from COCO with baseline models L2 and XL1 using a box prompt in FP16. All masks qualitatively capture the complete object. However, the ground-truth mask has the highest number of false positives, followed by L2, and then XL1.

liminary testing showed that the power consumption on average did not reach the maximum limit, and therefore we did not lock the clock frequency. Locking the clock frequency under a safe threshold can be necessary, if the GPU is busy with other processes in parallel [14]. However, we ensured no other processes were running during testing by monitoring the GPU during evaluation.

4

Results

This chapter presents the results from the three sensitivity experiments: stage-wise, type-wise, and layer-wise. We derive quantization schemes in search of a scheme with a good accuracy-efficiency trade-off, characterized by lower latency and smaller memory size with minimal accuracy degradation. We conclude this quantitative chapter with a trade-off summary of all schemes, followed by a visual comparison of the masks outputted from the quantized models and a critique of the methods used.

4.1 Quantitative results

We first show the discrepancies between a fully quantized INT8 model and the baseline model in FP16. In Table 4.1, we observe $\geq 11\%$ latency reductions across all models, but with a significant drop in mIoU when reducing the precision to INT8. Sizes are reduced by 50% across all models, since INT8 uses half the bits of FP16. The substantial accuracy degradation motivates us to explore fine-grained mixed-precision schemes. We note that the latency and size improvements at INT8 act as upper limits to what we can achieve with a mixed-precision scheme.

We evaluate the obtained schemes on both COCO and LVIS datasets, using both ground-truth bounding boxes and detected bounding boxes from ViTDet as prompts. For evaluation with ground-truth prompts, we measure mean Intersection over Union (mIoU, mIoU^S , mIoU^M , and mIoU^L), while for evaluation with ViTDet prompts we measure Average Precision (mAP, AP^S , AP^M , and AP^L). The superscripts indicate the size of the object’s segmentation mask: ‘small’ for objects with a mask size less than 32×32 , ‘medium’ for objects with a mask size between 32×32 and 96×96 , and ‘large’ for objects with a mask size greater than 96×96 . Performance of the baseline FP16 model on these four tasks is presented in Table 4.2.

A caveat to our results is the failure to compile a subset of the model engines in FP16. As a temporary solution, we replaced occurrences of FP16 with FP32 for these models, marked by a footnote in the table. Despite this issue, we can still perform latency measurements using the affected models, providing a preliminary review of the results. This is further discussed in Section 4.3.

Table 4.1 Baselines mIoU, latency, and size for models in FP16 (or FP32) versus INT8. Latency and mIoU is benchmarked using COCO dataset with ground-truth boxes. mIoU^S, mIoU^M, mIoU^L are omitted as in [80]. Model size refers to only the parameters of the image encoder. Benchmarks generated using TensorRT.

	mIoU		Latency (ms)		Size (MB)	
	FP16	INT8	FP16	INT8	FP16	INT8
L0	78.4	38.6 (↓ 51%)	0.76	0.65 (↓ 14%)	59	29
L1	78.5	32.9 (↓ 58%)	1.06	0.90 (↓ 15%)	83	42
L2 ¹	79.1	32.2 (↓ 59%)	1.40	1.23 (↓ 12%)	110	55
XL0 ¹	79.8	53.4 (↓ 33%)	1.99	1.78 (↓ 11%)	215	108
XL1 ¹	79.9	43.0 (↓ 46%)	3.75	2.73 (↓ 27%)	380	190

Table 4.2 Baseline accuracy for models in FP16 (or FP32) using COCO and LVIS datasets, prompted with ViTDet detected boxes and ground-truth boxes. The benchmarks using ViTDet box-prompts are from [3]. Otherwise, the benchmarks were made by us in TensorRT. Metrics displayed include mean Intersection over Union (mIoU) and Average Precision (AP). The predictions from the detected boxes are measured using AP, and the predictions from the ground-truth boxes are measured using mIoU. By [80], licensed under CC BY 4.0 <https://creativecommons.org/licenses/by/4.0/deed.en>. Modifications were made: four rows excluded, four additional rows included, and models renamed.

	COCO				LVIS			
	mIoU	mIoU ^S	mIoU ^M	mIoU ^L	mIoU	mIoU ^S	mIoU ^M	mIoU ^L
L0	78.4	74.2	80.6	82.4	77.9	72.3	86.5	90.2
L1	78.5	74.3	80.9	82.3	78.0	72.3	86.7	90.3
L2 ¹	79.1	74.9	81.5	83.1	78.6	72.9	87.5	91.1
XL0 ¹	79.8	75.6	81.9	83.7	79.7	74.3	88.1	91.4
XL1 ¹	79.9	75.8	82.2	83.7	79.9	74.4	88.4	91.6

	COCO				LVIS			
	mAP	AP ^S	AP ^M	AP ^L	mAP	AP ^S	AP ^M	AP ^L
L0	45.7	28.2	49.5	63.4	41.8	28.8	53.4	64.7
L1	46.2	28.7	50.4	64.0	42.5	29.4	54.5	65.0
L2	46.6	28.9	50.8	64.2	42.7	29.4	55.1	65.5
XL0	47.5	30.0	51.5	64.6	43.9	31.2	56.2	65.9
XL1	47.8	30.5	51.8	64.7	44.4	31.6	57.0	66.4

¹ mIoU measurements were done with an engine compiled to FP32. Latency measurements were done with an engine compiled to FP16.

Stage-wise sensitivity results

Our TensorRT analysis evaluates only the largest model of each series, XL1 and L2, to reduce scope. We focus on these models because our initial results suggested that their greater network depth made them more robust to quantization effects. The hypothesis was that if significant performance declines are observed in XL1 and L2, it would likely suggest even more pronounced drops in smaller models within the same series. We begin by quantizing only one stage or one layer type in isolation, to measure its sensitivity and offered latency reduction.

As per Table 4.3, we find that across series that the later stages are more sensitive to quantization. For the L-series, stage 0, and 1 are less sensitive to quantization, but do not offer any latency reductions. Quantizing stage 4, and the Neck, led to increases in latency. For the XL-series, stages 0, 2, and 3 offer an acceptable performance degradation. However, the latency reduction is statistically indistinguishable from the baseline model. The later stages offer a more promising latency reduction, but suffer from more sensitive layers.

Table 4.3 Benchmarks for L-series (L2) and XL-series models (XL1) in stage-wise quantization scenarios using COCO dataset ground-truth boxes. Metrics displayed include mean Intersection over Union (mIoU) and average latency in milliseconds (ms) for all stages. Measurements done in TensorRT.

Stage	L-series (L2)		XL-series (XL1)	
	mIoU (%)	ms	mIoU (%)	ms
0	78.1 ¹ (↓ 1.3%)	1.40	79.2 (↓ 0.9%)	3.78
1	79.0 ¹ (↓ 0.1%)	1.40	71.1 (↓ 11%)	3.67
2	69.7 ¹ (↓ 12%)	1.39	79.3 (↓ 0.8%)	3.71
3	0.0 ¹ (↓ 100%)	1.36	74.8 (↓ 6.4%)	3.65
4	42.5 (↓ 46%)	1.49	76.3 (↓ 4.5%)	3.94
5	-	-	69.7 (↓ 12.8%)	3.58
Neck	31.0 (↓ 61%)	1.70	42.5 (↓ 46.8%)	3.36

¹ Engine compiled with mixed-precision FP32/INT8 for mIoU measurements. Engine compiled with mixed-precision FP16/INT8 in TensorRT for latency measurements

Type-wise sensitivity results

We continue with individual quantization of all convolutional and point-wise operations, and find that point-wise operations should be avoided due to their impact on accuracy. TensorRT defines seven types of operations, but five of these contribute less than 2% to the model’s latency and are not worth targeting. Comparing convolutional and point-wise operations, the former are less sensitive to quantization. As shown in Table 4.4, quantizing only point-wise operations increases latency, whereas quantizing only convolutional operations decreases latency. The point-wise quantization experiment likely led to inefficient compilation. When precision constraints are set, the compiler avoids fusing layers with different precisions, reducing computational efficiency. One hypothesis for the increased latency is that layer fusions decrease latency more effectively than quantizing only point-wise operations.

Table 4.4 Benchmarks for L-series (L2) and XL-series models (XL1) in type-wise quantization scenarios using COCO dataset ground-truth boxes. Metrics displayed include mean Intersection over Union (mIoU) and average latency in milliseconds (ms) for the two most latency contributing types. Measurements done in TensorRT.

Model	Convolution		Point-wise	
	mIoU	ms	mIoU	ms
L2	53.2 (↓ 32.7%)	1.54 (↑ 10.0%)	13.9 (↓ 82.4%)	1.61 (↑ 15.0%)
XL1	54.9 (↓ 31.3%)	2.91 (↓ 22.4%)	32.2 (↓ 59.8%)	3.78 (↑ 0.8%)

Layer-wise sensitivity results

Having simulated the quantization of layers in isolation, we now plot the sensitivity of each layer for three models: L0, L2, and XL1. The inclusion of L0 is possible due to the low cost of simulated experiments. We identify several sensitive layers in all models, as discussed next. Based on observed patterns, we design and evaluate two mixed-precision schemes: Mix-DWSC, tailored to the sensitivity of L0, and Mix-MBC-Neck, tailored to the sensitivity of L2.

In L0, there exists a set of sensitive layers as per Figure 4.1, with layer indexed 3:4:2 being an outlier. The sensitive layers are exclusively part of the Mobile Inverted Convolution architecture (detailed in Section 2.5 and are identifiable by red color coding). With even more granularity, only the Depth-Wise Separable Convolutions inside the MBConvs are sensitive, not the first layer of the MBConv. Recall that an MBConv consists of three convolutions: one point-wise that expands the channel width, one depth-wise that performs inter-channel feature extraction, and lastly one point-wise that both performs intra-channel feature extraction and reduce channel-width. The last two layers together form a DWSCConv, which is sensitive. This insight motivates a mixed-precision scheme that retains all DWSCConvs in floating point, which we call *Mix-DWSC*.

The observation from L0 holds in part when examining L2 in Figure 4.2. L2 has a dozen very sensitive and some moderately sensitive layers in stage 4. In addition to DWSCConvs being sensitive in general, there are three MBConv blocks in which the entire block (including its first convolution) is sensitive to some degree (block 4:0, 4:1, and 4:2). Additionally, the Neck of L2 exhibits some degradation. These insights motivate a mixed-precision scheme that retains all MBConvs and the entire Neck in floating point, which we call *Mix-MBC-Neck*.

Lastly, examining the largest model XL1 in Figure 4.3, there are sensitive layers throughout the model. The XL-series uses Fused-MBConvs in stage 3, unlike the L-series, which uses MBConvs, so direct comparison is not appropriate. The accuracy degradation is severe in the Fused-MBConvs in both stage 2 and stage 3, as well as in the MBConvs that initiate the attention stages (4 and 5). Due to this severe sensitivity, we do not design another mixed-precision scheme for XL1. Instead, we focus on the self-attention layers in stage 4 (for both series) and stage 5 (for the XL-series).

Across the three experiments, only one attention layer causes degradation in one model (XL1:5:1:1) and only one multiscale QKV convolution shows degradation (XL1:4:1:0). Otherwise, attention layers exhibit robustness. This suggests that ReLU linear attention layers and multiscale QKV convolutions are not sensitive to quantization. The novel architecture, in addition to being efficient by design, appears well suited for quantization to INT8. Since softmax-based self-attention is difficult to quantize, as discussed in Section 2.6, this finding supports the use of ReLU linear attention over Softmax attention in future models.

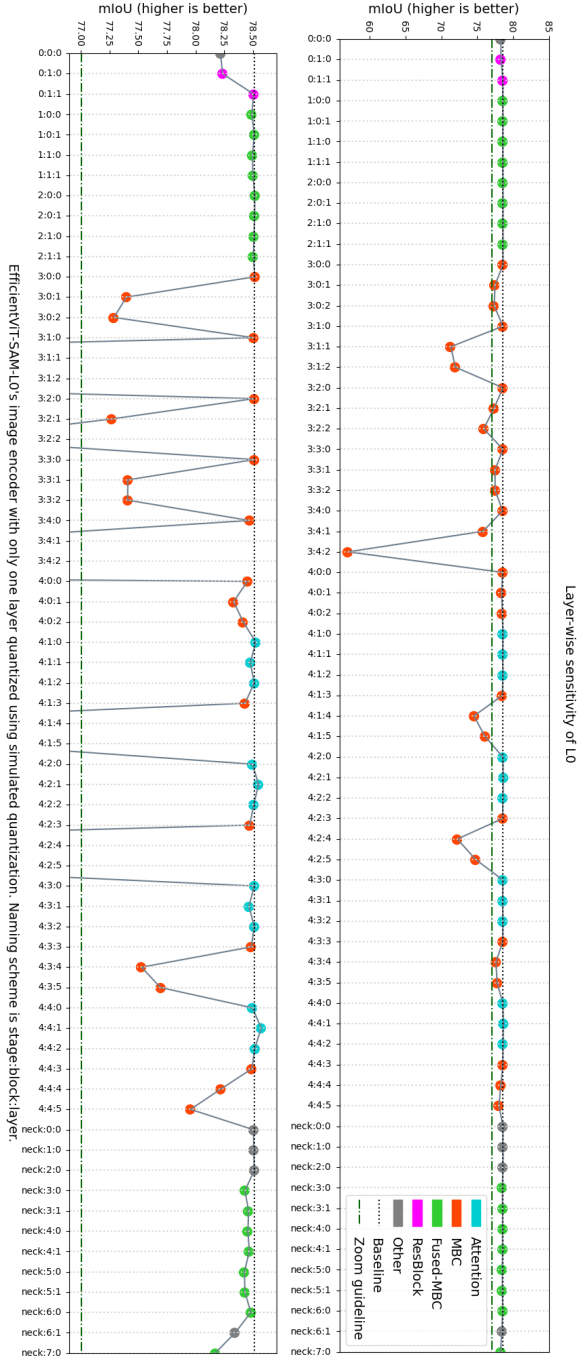
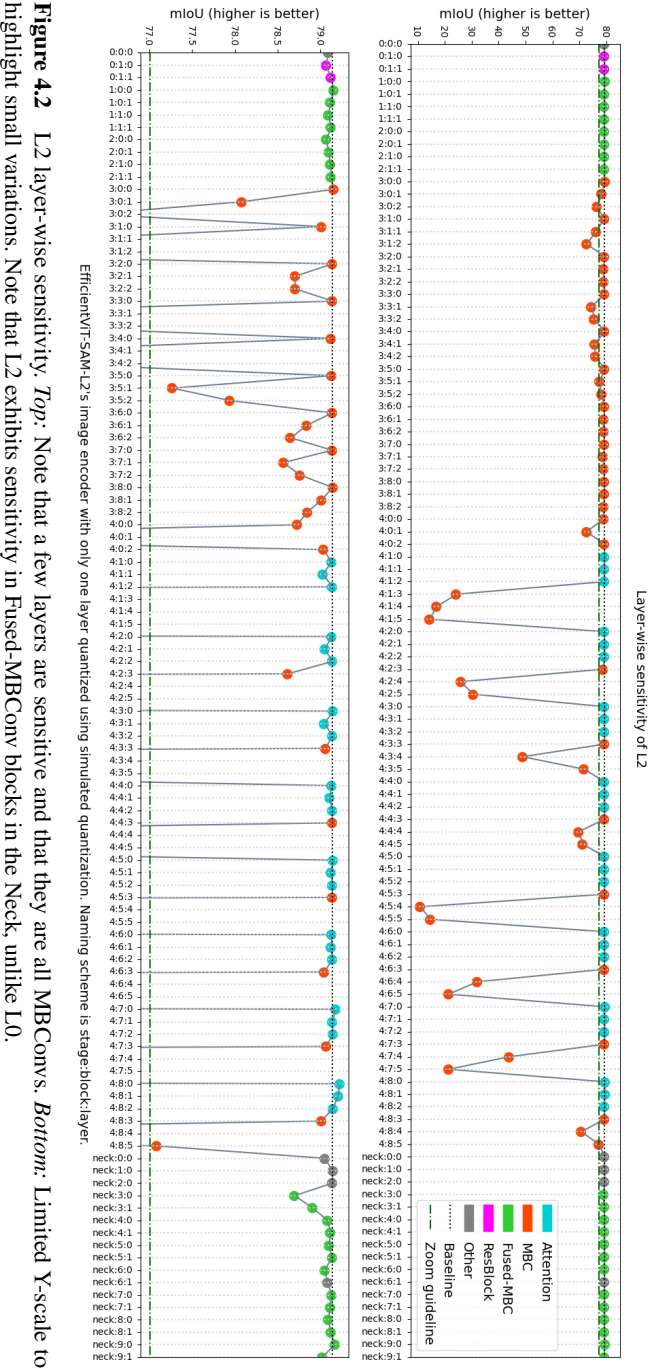


Figure 4.1 L0 layer-wise sensitivity. *Top:* Note that the few sensitive layers are all part of DWSConvs inside MBConvs. *Bottom:* Limited Y-scale to highlight small variations. Apart from the input and output layers of L0, only DWSConvs are sensitive.



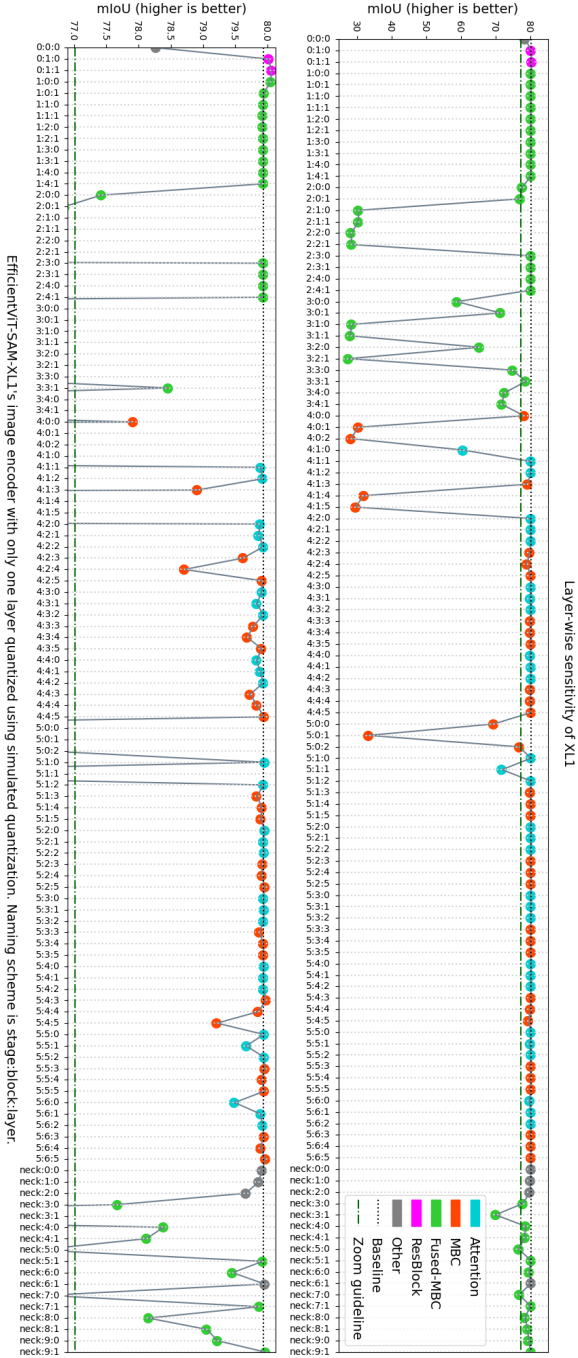


Figure 4.3 XL1 layer-wise sensitivity. *Top:* There are several sensitive layers, concentrated in stages 2, 3 and 4, but also present in stage 5. Fused-MBCs in early stages are also sensitive, unlike in the L-series. Note the single sensitive attention stage, something not seen in L-series. *Bottom:* Limited Y-scale to highlight small variations. Note the few layers left on the baseline: only stage 1 and stage 5 exhibits robustness to quantization, but even in stage 5 there are exceptions.

Evaluation of mixed-precision schemes

Insights from the above experiments have produced two mixed-precision schemes; Mix-DWSC and Mix-MBC-Neck. We now evaluate them across all variants of EfficientViT-SAM. We then define and evaluate two additional schemes named Mix-L and Mix-XL.

We prompt the model in two ways: first, with COCO’s ground-truth bounding boxes, measuring mean Intersection over Union only; second, with bounding boxes outputted from the ViTDet object detection model, measuring mean Average Precision only. The ViTDet prompts represent a realistic scenario where EfficientViT-SAM is used as part of a pipeline in an application. For both mAP and mIoU metrics, we break down the scores across small, medium, and large object sizes to identify if the model struggles with any specific size category. The relative changes in mIoU and mAP compared to the *FP16* baseline models aid the analysis. Note that the simulation results from ViTDet prompts on LVIS are missing (denoted with $-\dagger$) due to dependency conflicts not solvable within the time limitation of this work.

Mix-DWSC evaluation is outlined in Table 4.5. This scheme was designed with only L0 in mind, and quantize everything but the DWSCConv layers. As expected, it performs the best on L0 with a relatively low loss of 3.3% mIoU. However, mAP tells a different story with an accuracy reduction of 9.0%. This could be an effect of errors from the object detection that produced the prompt on which mAP is measured. L1-Mix-DWSC trails close behind in both scores, while L2-Mix-DWSC suffers more intensive accuracy degradation across both scores. The severe degradation of XL1-Mix-DWSC is natural since the scheme quantizes the model’s many sensitive Fused-MBConv layers. Observed effects can not be attributed to small, medium or large objects specifically as the scores are reduced evenly from the baseline.

Table 4.5 Evaluation of Mix-DWSC for L-series and XL-series on COCO and LVIS datasets. *Top*: box-prompted with ground-truth boxes. *Bottom*: box-prompted with ViTDet boxes. Due to conflicting package dependencies, evaluation of mAP for ViTDet box-prompts was unsuccessful in simulation, denoted with $-\dagger$.

	COCO				LVIS			
	mIoU	mIoU ^S	mIoU ^M	mIoU ^L	mIoU	mIoU ^S	mIoU ^M	mIoU ^L
L0 Mix-DWSC	75.8 (↓ 3.3%)	80.2	78.0	71.3	74.3 (↓ 4.6%)	85.4	82.4	69.1
L1 Mix-DWSC	75.3 (↓ 4.1%)	79.5	77.6	71.0	73.8 (↓ 5.4%)	84.7	82.0	68.6
L2 Mix-DWSC	69.8 (↓ 11.8%)	72.3	71.1	67.3	69.2 (↓ 12.0%)	74.7	75.8	65.6
XL0 Mix-DWSC	19.0 (↓ 76.2%)	6.7	18.0	27.0	23.0 (↓ 71.1%)	7.8	18.7	27.3
XL1 Mix-DWSC	24.7 (↓ 69.1%)	5.7	31.0	30.6	25.2 (↓ 68.5%)	6.4	32.4	25.5

	COCO				LVIS			
	mAP	AP ^S	AP ^M	AP ^L	mAP	AP ^S	AP ^M	AP ^L
L0 Mix-DWSC	41.6 (↓ 9.0%)	25.6	45.6	57.5	$-\dagger$	$-\dagger$	$-\dagger$	$-\dagger$
L1 Mix-DWSC	41.2 (↓ 10.8%)	25.3	45.2	57.5	$-\dagger$	$-\dagger$	$-\dagger$	$-\dagger$
L2 Mix-DWSC	32.3 (↓ 30.7%)	22.0	36.8	41.5	$-\dagger$	$-\dagger$	$-\dagger$	$-\dagger$
XL0 Mix-DWSC	0.0 (↓ 100.0%)	0.1	0.0	0.0	$-\dagger$	$-\dagger$	$-\dagger$	$-\dagger$
XL1 Mix-DWSC	0.4 (↓ 99.2%)	0.4	0.8	0.0	$-\dagger$	$-\dagger$	$-\dagger$	$-\dagger$

Mix-MBC-Neck evaluation is outlined in Table 4.6. This scheme was designed for L2 and is less aggressive than Mix-DWSC, so we expect higher scores across the board. This is indeed the case for the entire L-series, as mIoU degradation is limited to $\leq 1.5\%$ and mAP to $\leq 5.4\%$, with effects evenly distributed over small, medium and large objects.

Table 4.6 Evaluation of Mix-MBC-Neck for L-series and XL-series on COCO and LVIS datasets. *Top*: box-prompted with ground-truth boxes. *Bottom*: box-prompted with ViTDet boxes. Due to conflicting package dependencies, evaluation of mAP for ViTDet box-prompts was unsuccessful in simulation, denoted with $^{-\dagger}$.

	COCO				LVIS			
	mIoU	mIoU ^S	mIoU ^M	mIoU ^L	mIoU	mIoU ^S	mIoU ^M	mIoU ^L
L0 Mix-MBC-Neck	77.5 ($\downarrow 1.1\%$)	81.8	79.8	73.1	76.2 ($\downarrow 2.2\%$)	87.7	84.5	70.8
L1 Mix-MBC-Neck	77.6 ($\downarrow 1.1\%$)	81.6	80.2	73.0	76.0 ($\downarrow 2.6\%$)	87.6	84.8	70.5
L2 Mix-MBC-Neck	77.9 ($\downarrow 1.5\%$)	81.4	80.3	73.8	77.0 ($\downarrow 2.0\%$)	87.5	85.6	71.7
Xl0 Mix-MBC-Neck	51.1 ($\downarrow 36.0\%$)	53.1	51.9	49.3	48.3 ($\downarrow 39.4\%$)	53.7	52.6	45.7
Xl1 Mix-MBC-Neck	29.5 ($\downarrow 63.1\%$)	10.3	36.1	35.3	29.9 ($\downarrow 62.6\%$)	11.0	38.0	29.7

	COCO				LVIS			
	mAP	AP ^S	AP ^M	AP ^L	mAP	AP ^S	AP ^M	AP ^L
L0 Mix-MBC-Neck	43.8 ($\downarrow 4.2\%$)	27.0	48.2	60.1	$^{-\dagger}$	$^{-\dagger}$	$^{-\dagger}$	$^{-\dagger}$
L1 Mix-MBC-Neck	44.0 ($\downarrow 4.8\%$)	27.0	48.4	60.5	$^{-\dagger}$	$^{-\dagger}$	$^{-\dagger}$	$^{-\dagger}$
L2 Mix-MBC-Neck	44.1 ($\downarrow 5.4\%$)	27.7	48.7	60.0	$^{-\dagger}$	$^{-\dagger}$	$^{-\dagger}$	$^{-\dagger}$
Xl0 Mix-MBC-Neck	10.1 ($\downarrow 78.7\%$)	6.6	11.4	14.3	$^{-\dagger}$	$^{-\dagger}$	$^{-\dagger}$	$^{-\dagger}$
Xl1 Mix-MBC-Neck	0.9 ($\downarrow 98.1\%$)	1.2	1.5	0.2	$^{-\dagger}$	$^{-\dagger}$	$^{-\dagger}$	$^{-\dagger}$

The fact that XL1-Mix-MBC-Neck only improves marginally over XL-Mix-DWSC despite the former addressing the Neck (which was sensitive in XL1) is noticeable, as it shows that all sensitive layers must be addressed in a mixed-precision scheme, not just a subset of them. We can further give some insights about L1. In evaluation, L1 closer resembles L0 than L2, suggesting that sensitivity in L1 is concentrated to DWSCConv layers and that its Neck is robust.

The robustness of the ReLU linear attention layer was suggested from the layer-wise sensitivity analysis, as it was observed that it could be quantized in isolation without accuracy loss in all models. This hypothesis is further supported by the evaluation of Mix-DWSC and Mix-MBC-Neck. Quantization in isolation did not take into account the possibility of cross layer dependencies that propagate through the model, but these measurements do. In both schemes, all self-attention layers are quantized together (including the layer generating QKV and the trailing linear projection). Yet, limited degradation occurs in the L-series under Mix-DWSC and Mix-MBC-Neck. However, we can only hypothesize, not claim proof, as that would require an in-depth investigation of these layers in a setting where there are no floating point operations between the attention layers.

Using our stage-wise, and type-wise benchmarks, we propose a set of increasingly aggressive mixed-precision schemes as per Table 4.7. We find that for both model series, quantization leads to an unsatisfactory trade-off between the accuracy (mIoU),

and latency (ms). For neither model series, it does not make sense to quantize more aggressively than s_0, s_2 for XL, and s_0, s_1 for L. Using the descriptive statistics from the latency benchmarks, we find, using the Welch’s t-test, that s_0, s_2 for XL1 is distinguishable from the baseline, and that s_0, s_1 for L2 is not. Therefore, we will keep the strategy for XL-series, but use s_0, s_1, s_2 for L-series. However, during benchmarking we find that s_0, s_1 leads to a severe accuracy drop for XL0. For this model, we pick s_0, s_1, s_2, s_4 (Conv-only) instead. We call these mixed-precision quantization schemes *Mix-L* and *Mix-XL* respectively. In Table 4.8, we evaluate them on COCO and LVIS, as done for the previous schemes.

Table 4.7 Evaluation of proposed mixed-precision schemes for L-series (L2) and XL-series models (XL1) using COCO dataset ground-truth boxes. Metrics displayed include mean Intersection over Union (mIoU) and average latency in milliseconds (ms) for the three most latency-contributing types.

L-series (L2)	mIoU	ms
s_0, s_1	78.0* ($\downarrow$ 1.4%)	1.39 ($\downarrow$ 0.7%)
s_0, s_1, s_2	67.8* ($\downarrow$ 14.3%)	1.34 ($\downarrow$ 4.3%)
s_0, s_1, s_2, s_4 (Conv-only)	39.3* ($\downarrow$ 50.3%)	1.35 ($\downarrow$ 3.6%)
s_0, s_1, s_2, s_3, s_4 (Conv-only)	24.7* ($\downarrow$ 68.8%)	1.30 ($\downarrow$ 7.1%)
s_0, s_1, s_2, s_3, s_4 (Conv-only), Neck (Conv-only)	25.6* ($\downarrow$ 67.6%)	1.10 ($\downarrow$ 21.4%)
XL-series (XL1)	mIoU	ms
s_0, s_2	77.8 ($\downarrow$ 2.6%)	3.66 ($\downarrow$ 2.4%)
s_0, s_2, s_4 (Conv-only)	71.4 ($\downarrow$ 10.6%)	3.61 ($\downarrow$ 3.7%)
s_0, s_2, s_3, s_4 (Conv-only)	64.8 ($\downarrow$ 18.9%)	3.45 ($\downarrow$ 8.0%)
s_0, s_1, s_2, s_3, s_4 (Conv-only)	62.6 ($\downarrow$ 21.7%)	3.32 ($\downarrow$ 11.5%)
s_0, s_1, s_2, s_3, s_4 (Conv-only), s_5 (Conv-only)	49.1 ($\downarrow$ 38.5%)	3.27 ($\downarrow$ 12.8%)
s_0, s_1, s_2, s_3, s_4 (Conv-only), s_5 (Conv-only), Neck (Conv-only)	54.8 ($\downarrow$ 31.4%)	2.84 ($\downarrow$ 24.3%)

Table 4.8 Evaluation of s_0, s_1, s_2 L-series (Mix-L) and s_0, s_2 XL-series (Mix-XL) on COCO and LVIS datasets. *Top*: box-prompted with ground-truth boxes. *Bottom*: box-prompted with ViTDet boxes.

	COCO				LVIS			
	mIoU	mIoU ^S	mIoU ^M	mIoU ^L	mIoU	mIoU ^S	mIoU ^M	mIoU ^L
L0 Mix-L	77.8 ($\downarrow$ 0.8%)	73.2	80.0	83.1	76.5 ($\downarrow$ 1.8%)	71.0	84.9	88.8
L1 Mix-L	71.3 ($\downarrow$ 9.2%)	65.8	72.7	78.7	71.3 ($\downarrow$ 8.6%)	65.8	72.7	78.7
L2 Mix-L	67.8* ($\downarrow$ 14.3%)	65.5*	68.4*	71.1*	66.5* ($\downarrow$ 15.4%)	63.4*	71.5*	72.8*
XL0 Mix-XL	46.2* ($\downarrow$ 42.1%)	47.7*	51.7*	42.1*	46.7* ($\downarrow$ 41.4%)	44.8*	52.5*	44.5*
XL1 Mix-XL	77.9 ($\downarrow$ 2.5%)	73.5	79.4	83.3	76.2 ($\downarrow$ 4.6%)	71.4	83.4	86.9
	COCO				LVIS			
	mAP	AP ^S	AP ^M	AP ^L	mAP	AP ^S	AP ^M	AP ^L
L0 Mix-L	44.7 ($\downarrow$ 2.2%)	27.0	48.0	63.0	39.7 ($\downarrow$ 5.0%)	27.2	51.0	62.0
L1 Mix-L	35.3 ($\downarrow$ 23.6%)	20.4	37.6	52.3	29.6 ($\downarrow$ 30.4%)	20.2	38.2	49.0
L2 Mix-L	29.2 ($\downarrow$ 37.3%)	19.9	32.5	39.7	26.3 ($\downarrow$ 38.4%)	20.1	34.4	38.3
XL0 Mix-XL	7.8 ($\downarrow$ 83.6%)	5.6	9.7	10.9	7.8 ($\downarrow$ 82.2%)	5.4	11.4	11.7
XL1 Mix-XL	44.5 ($\downarrow$ 6.9%)	27.7	47.8	61.0	39.4 ($\downarrow$ 11.3%)	28.3	49.9	59.5

Performance trade-offs

The optimal performance trade-off involves a small accuracy degradation combined with a significant reduction in latency or size. Table 4.9 summarize the relative effects on accuracy and size of the Mix-DWSC and Mix-MBC-Neck schemes, and Figure 4.4 visualize these effects as compared to the baseline models. In simulation (where latency can not be measured in a meaningful way), we reduced the size of L0 by 31% to 40 MB at the cost of a 9% reduction in mAP on COCO, using the scheme Mix-DWSC. L1 exhibits a similar trade-off under the same scheme. With the less aggressive scheme Mix-MBC-Neck, the size of L2 is reduced by 12% to 97 MB at a 5.4% reduction in mAP on COCO. As observed earlier, neither the Mix-DWSC nor the Mix-MBC-Neck schemes are successful when applied to the XL-series.

Table 4.9 Accuracy-Size trade-off for Mix-DWSC and Mix-MBC-Neck. Latency is not measurable in simulation, hence size is used as an efficiency metric. Due to conflicting package dependencies, the evaluation of mAP for ViTDet box-prompts was unsuccessful in simulation, denoted with $-\dagger$.

	Mix-DWSC				Mix-MBC-Neck			
	COCO mAP	LVIS mAP	Latency ms	Size MB	COCO mAP	LVIS mAP	Latency ms	Size MB
L0	(↓ 9.0%)	$-\dagger$	n/a	40 (↓ 31%)	(↓ 4.2%)	$-\dagger$	n/a	52 (↓ 12%)
L1	(↓ 10.8%)	$-\dagger$	n/a	56 (↓ 33%)	(↓ 4.8%)	$-\dagger$	n/a	73 (↓ 12%)
L2	(↓ 30.7%)	$-\dagger$	n/a	73 (↓ 33%)	(↓ 5.4%)	$-\dagger$	n/a	97 (↓ 12%)
XL0	(↓ 100.0%)	$-\dagger$	n/a	143 (↓ 34%)	(↓ 78.7%)	$-\dagger$	n/a	186 (↓ 13%)
XL1	(↓ 99.2%)	$-\dagger$	n/a	247 (↓ 35%)	(↓ 98.1%)	$-\dagger$	n/a	326 (↓ 14%)

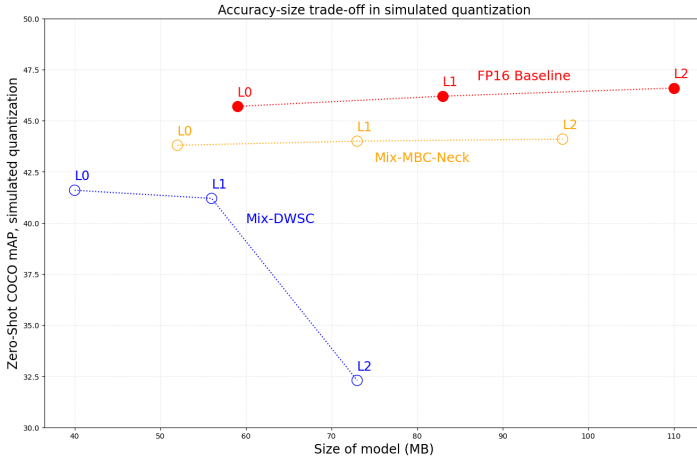


Figure 4.4 Accuracy-Size trade-off for Mix-DWSC and Mix-MBC-Neck, in simulation. XL-series is omitted to aid visualization.

Table 4.10 summarize the relative effects on accuracy and latency for Mix-L and Mix-XL. Figure 4.5 visualize the findings. Only L0-Mix-L and XL1-Mix-XL offer reasonable trade-offs between accuracy and latency, the other model variants suffer extensive accuracy degradation on both COCO and LVIS under their respective quantization schemes. Latency improvements are however small at single-digit percentages. In most use cases, the observed trade-offs in all schemes are not acceptable since the accuracy loss is simply disproportionately large.

Table 4.10 Accuracy-Latency trade-off for Mix-L and Mix-XL.

	COCO mAP	LVIS mAP	Latency ms
L0 Mix-L	(↓ 2.2%)	(↓ 5.0%)	0.72 (↓ 5.3%)
L1 Mix-L	(↓ 23.6%)	(↓ 30.4%)	1.02 (↓ 3.8%)
L2 Mix-L	(↓ 37.3%)	(↓ 38.4%)	1.34 (↓ 4.3%)
XL0 Mix-XL	(↓ 83.6%)	(↓ 82.2%)	1.94 (↓ 2.6%)
XL1 Mix-XL	(↓ 6.9%)	(↓ 11.3%)	3.66 (↓ 2.4%)

The natural progression is to transfer the schemes designed in simulation experiments to the deployment phase by compiling TensorRT engines that quantize the same layers. This was done for all model variants. The blue and yellow plots in Figure 4.5 depict the accuracy-latency trade-off in TensorRT for Mix-DWSC and Mix-MBC-Neck, respectively. There are two observations to be made. First, latency is *increased* for both Mix-DWSC and Mix-MBC-Neck for the L-series. As mixed-precision schemes prohibit the fusion of layers with different precision formats, the negative latency effect could be due to the fusion of layers decreasing latency more effectively than quantizing some layers. Second, measurements of zero-shot mAP on COCO disagree in simulation versus TensorRT. For the L-series, simulation is overly-optimistic, e.g. L0-DWSC drops 42% in TensorRT (to 26.7) but only 9% in simulation. The case is reversed for the XL series, where TensorRT measurements are more optimistic. Discrepancies are further discussed in Section 4.3.

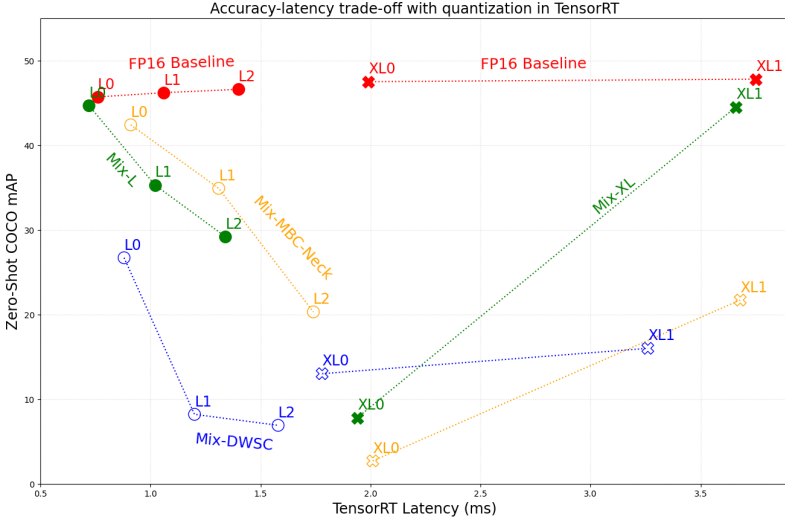


Figure 4.5 Accuracy-Latency trade-off for all our proposed mixed-precision schemes. Accuracy measured in TensorRT.

4.2 Qualitative results

In this section, we visualize the segmentation masks outputted from all model variants under the Mix-L and Mix-XL schemes. We explore the segmentation masks on images with different object sizes, occluded objects, and objects in dense environments, and compare them against the segmentations from the non-quantized baseline models. These qualitative results were produced using TensorRT.

Object Size

We examine three segmentations based on size: small (32×32 pixels), medium (32×32 to 96×96 pixels), and large (greater than 96×96 pixels). As shown in Figure 4.6, the output of the quantized models remains unchanged for the selected large image. The L2 model struggles slightly with edge definition, but this does not significantly impact the overall quality of the mask. In Figure 4.7, the results are more varied. Despite small artifacts, most models accurately segment the inline skater. However, the L2 model has difficulty finding the object’s boundaries, affecting segmentation accuracy. In Figure 4.8, all models accurately capture the object’s mask, with only subtle differences around the boundaries.

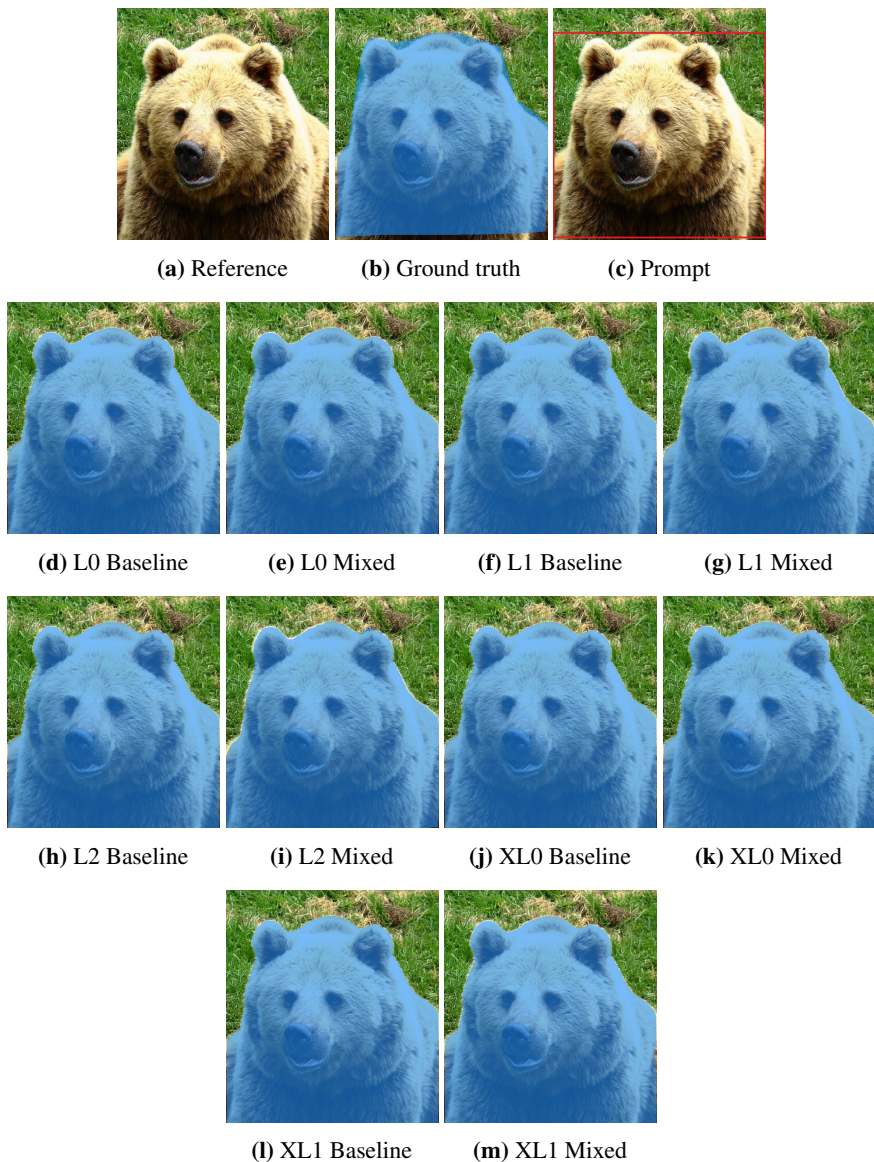


Figure 4.6 Comparison of segmentation masks for a large-sized object produced by all models. (a) Reference image; (b) Ground truth mask; (c) Prompt Baseline; (d) Prompt Mixed; (e) L0 Baseline; (f) L0 Mixed; (g) L1 Baseline; (h) L1 Mixed; (i) L2 Baseline; (j) L2 Mixed; (k) XL0 Baseline; (l) XL0 Mixed; (m) XL1 Baseline; (n) XL1 Mixed.

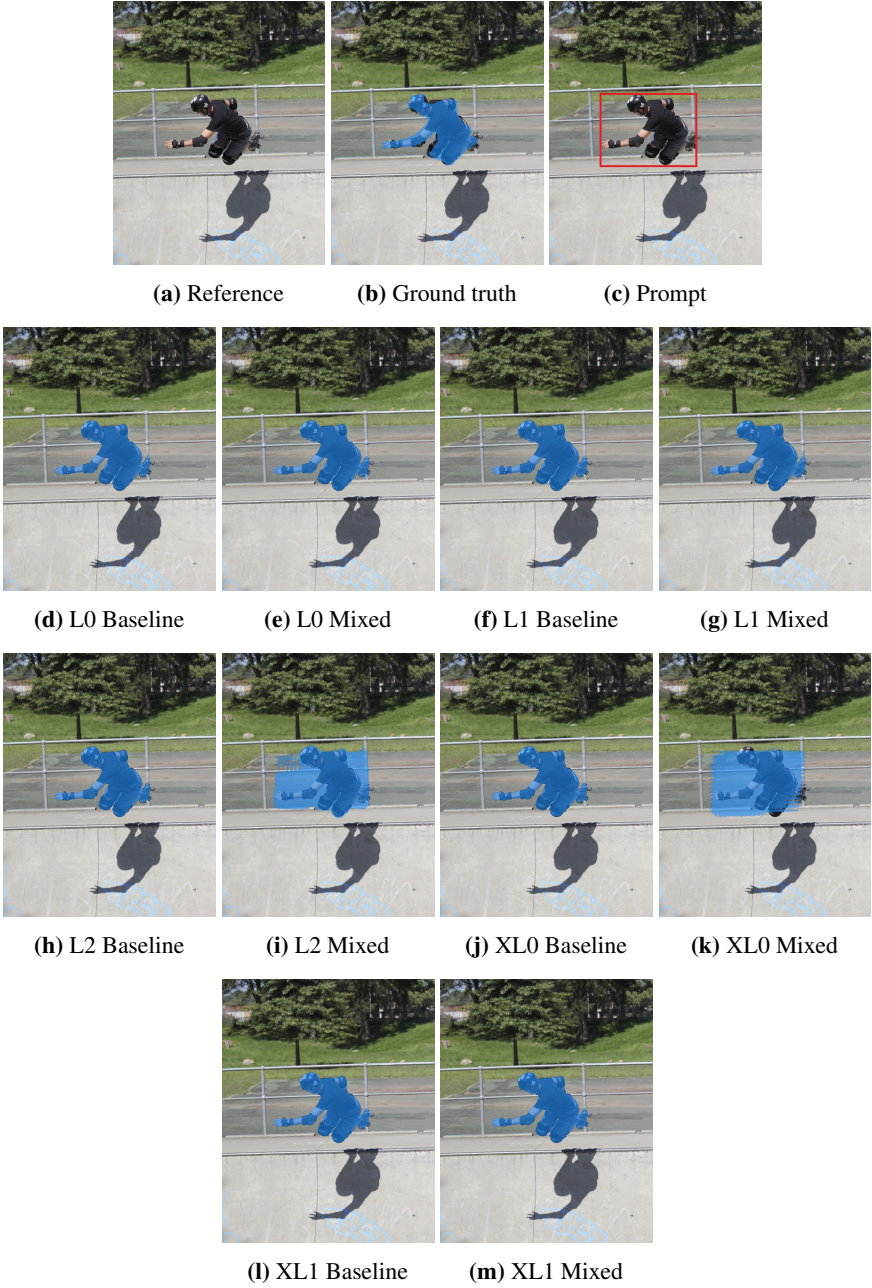


Figure 4.7 Comparison of segmentation masks for a medium-sized object produced by all models. (a) Reference image; (b) Ground truth mask; (c) Prompt Baseline; (d) Prompt Mixed; (e) L0 Baseline; (f) L0 Mixed; (g) L1 Baseline; (h) L1 Mixed; (i) L2 Baseline; (j) L2 Mixed; (k) XL0 Baseline; (l) XL0 Mixed; (m) XL1 Baseline; (n) XL1 Mixed.

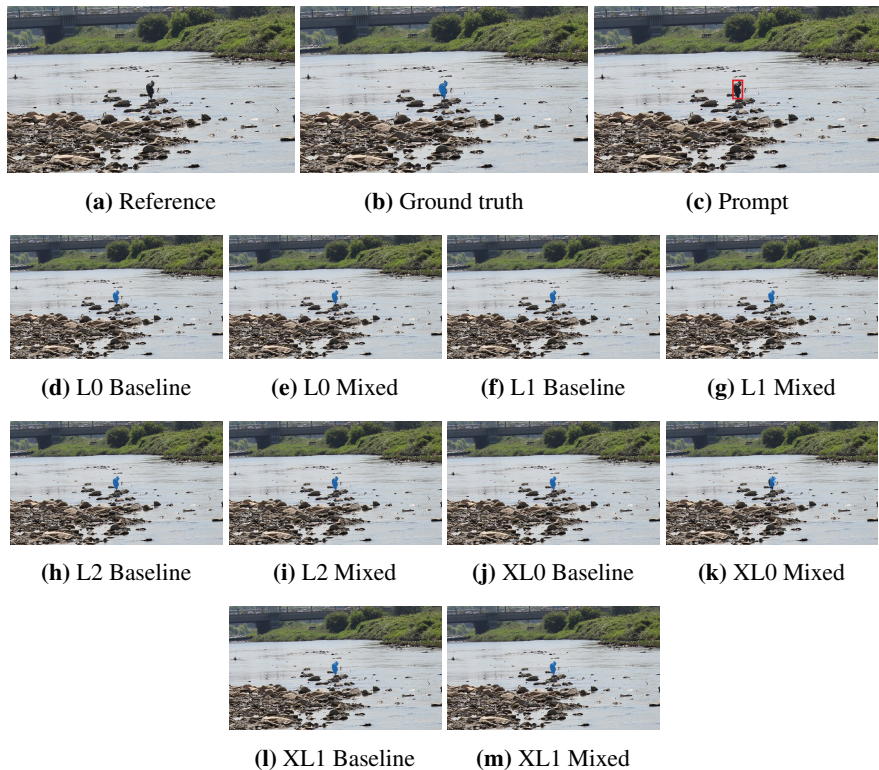


Figure 4.8 Comparison of segmentation masks for a small-sized object produced by all models. (a) Reference image; (b) Ground truth mask; (c) Prompt Baseline; (d) Prompt Mixed; (e) L0 Baseline; (f) L0 Mixed; (g) L1 Baseline; (h) L1 Mixed; (i) L2 Baseline; (j) L2 Mixed; (k) XL0 Baseline; (l) XL0 Mixed; (m) XL1 Baseline; (n) XL1 Mixed.

Object Occlusion

We examine the quantized models’ ability to segment partly occluded objects in Figure 4.9. Occlusion refers to the scenario where the object is partly hidden, requiring the model to infer and segment the visible parts of the object. The quality of the masks varies. While all models successfully capture the object within the mask, they struggle to varying degrees with accurately defining the object boundaries and excluding occluded sections.

Object Density

We examine the quantized models’ ability to segment an object in a crowded environment in Figure 4.10. Both quantized and baseline models struggle to segment the target object. All models correctly exclude the object in front of the oven, but fail to exclude objects on or behind it. There is not much difference in performance between the baseline and quantized models in this scenario. This issue may be attributed to the task type, suggesting that a point-prompt might provide a more precise prompt to the decoder. Since the box-prompt cannot capture the oven without also including other objects, as illustrated by the coarse box in the third image of Figure 4.10.

The quantitative accuracy scores do not tell the whole story of a quantized model’s performance, as illustrated by the above visualizations. A practitioner should evaluate models from several perspectives, and select the model that performs the best on the intended use-case. For example, all quantized model variants could successfully segment the large and isolated bear in Figure 4.6, despite XL0-Mix-XL scoring 80% lower than the baseline on mAP measurements on both LVIS and COCO. Segmenting the occluded horse, only XL1-Mix-XL could produce a reasonable segmentation mask. L0-Mix-L could not, even though this model scored the highest in absolute terms on three out of four quantitative benchmarks in Table 4.8, higher than XL1-Mix-XL.

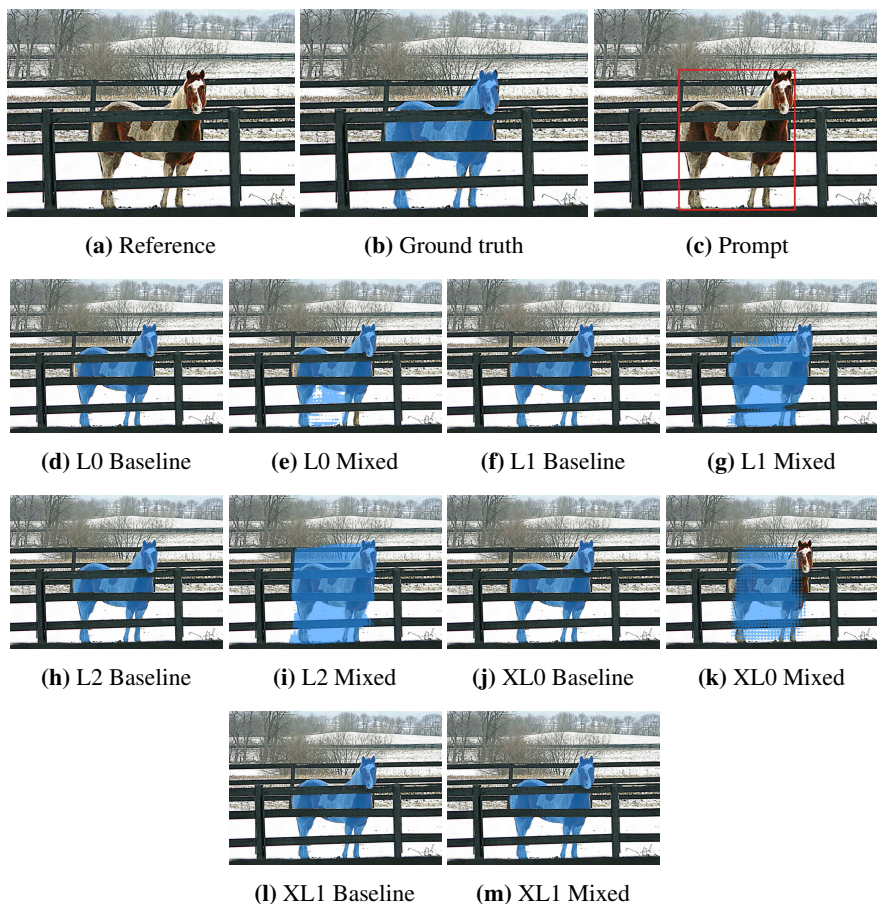


Figure 4.9 Comparison of segmentation masks for a partly occluded object produced by all models. (a) Reference image; (b) Ground truth mask; (c) Prompt Baseline; (d) Prompt Mixed; (e) L0 Baseline; (f) L0 Mixed; (g) L1 Baseline; (h) L1 Mixed; (i) L2 Baseline; (j) L2 Mixed; (k) XL0 Baseline; (l) XL0 Mixed; (m) XL1 Baseline; (n) XL1 Mixed.

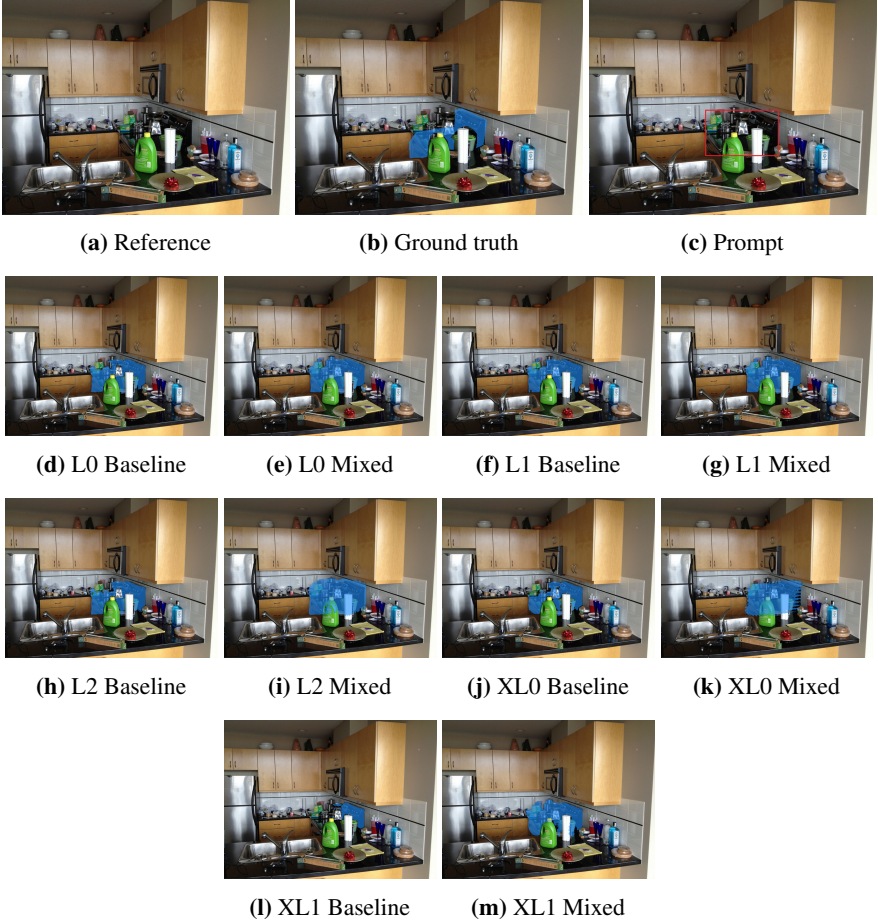


Figure 4.10 Comparison of segmentation masks for an object in a densely packed environment produced by all models. (a) Reference image; (b) Ground truth mask; (c) Prompt Baseline; (d) Prompt Mixed; (e) L0 Baseline; (f) L0 Mixed; (g) L1 Baseline; (h) L1 Mixed; (i) L2 Baseline; (j) L2 Mixed; (k) XL0 Baseline; (l) XL0 Mixed; (m) XL1 Baseline; (n) XL1 Mixed.

4.3 Method discussion

Discrepancies Between Simulation and TensorRT

We compiled the Mix-DWSC and Mix-MBC-Neck schemes from simulation to TensorRT and observed discrepancies between the accuracy measurements in the two domains, as presented in Section 4.3. The measurements are provided in Table 4.11 for completeness, including a comparison of the models fully quantized to INT8 in simulation and TensorRT. Simulation results are higher than TensorRT results for the L-series across all three quantization schemes, but the pattern does not hold for the XL-series.

The observations are alarming since they put the validity of the simulation results into question. The discrepancies may stem from effects in TensorRT, weaknesses in the simulation framework, or a combination of factors. There are configuration differences, such as the use of different optimizers in calibration (KL-divergence and EMA respectively), but this should not create pattern-breaking differences at this magnitude. Concerning TensorRT, the other optimizations occurring under the hood may affect the results of quantization, such as fusing of operations and approximation of operators. Note however that TensorRT can apply its optimizations to the FP16 baseline with negligible, if any, loss in accuracy. The effects are difficult to investigate within the scope of this thesis work due to their complexity. There are also some weaknesses in the simulation framework. As it uses 32-bit floating point, not 16-bit, the results should be slightly optimistic. Further, our framework is not complete since the bias weight of convolutions or the parameters of the batch-normalization layers are not quantized, which may also create overly optimistic results. Simply, we can not provide a definite explanation as to why the simulation framework fails to approximate TensorRT deployment.

Table 4.11 Discrepancies in accuracy between Simulation (Sim) and TensorRT (TRT) for two mixed-precision schemes and the fully quantized INT8 model. Measurements performed on COCO dataset.

Model	Mix-DWSC		Mix-MBC-Neck		INT8	
	mAP Sim	mAP TRT	mAP Sim	mAP TRT	mIoU Sim	mIoU TRT
L0	41.6	26.7	43.8	42.4	38.6	38.6
L1	41.2	8.2	44.0	34.9	46.8	32.9
L2	32.3	6.9	44.1	20.3	49.4	32.2
XI0	0.0	13.0	10.1	2.7	34.8	53.4
XI1	0.4	16.0	0.9	21.7	24.8	43.0

Compilation issues

We used Polygraphy, a debugging tool for TensorRT, to compare the activation distributions between the original model in ONNX format and the TensorRT engine compiled to FP16. Any layer with a relative or absolute error greater than 10^{-3} is considered to be failing. The maintainers of TensorRT recommend this threshold [61]. We found that the network fails as early as the first stage and extracted a minimal failing subset. Similar issues were encountered by [55]. In May 2024, the authors [3] changed the training-script to use the precision format BF16 instead of FP16. BF16 (Brain-float) uses the same number of bits, but has a much greater range [30]. Using this precision format during inference could ensure that the model doesn't overflow, while retaining the speed-up from choosing a precision format half the size of FP32.

5

Conclusion

In this thesis, our overarching aim was to investigate if EfficientViT-SAM could be accelerated with acceptable accuracy losses using quantization. To answer this question, we examined the model’s latency and sensitivity to quantization at three different abstractions: by stage, by operation type, and by layer. We designed and evaluated four mixed-precision quantization schemes, finding that post-training quantization of EfficientViT-SAM causes unacceptable accuracy losses.

Our mixed-precision schemes achieved only minor latency reductions, as aggressive precision reduction resulted in significant accuracy degradation. Despite this, our latency and sensitivity analyses provide valuable insights for future improvements. We validated our models on the widely recognized COCO and LVIS benchmarks, suggesting that our results are generalizable for downstream box-prompt tasks. However, further benchmarking is required for point-prompt and text-prompt tasks. This report examines to what extent EfficientViT-SAM can be quantized without severe accuracy loss and offers a partial answer. Our results indicate that the model is sensitive to reduced precision, but additional experiments are needed to confirm these findings.

For the smallest model EfficientViT-SAM-L0 under the Mix-DWSC scheme, we reduced the size by 31% to 40 MB at the cost of a 9% reduction in mAP. With the less aggressive scheme Mix-MBC-Neck, the size of EfficientViT-SAM-L2 was reduced by 12% to 97 MB at a 5.4% reduction in mAP. However, neither of these results were transferable from simulation to TensorRT. From our qualitative results, we found that the poorly performing models matched baseline performance in simpler scenarios but failed in predicting complex segmentation masks.

With regard to the second research question, we verified the hypothesis that a few (10%) of layers in the L-series are sensitive, while the other layers are robust to quantization. The pattern was not as clear for the larger XL-series. More interestingly, we identified that the Mobile Inverted Convolution block (MBConv) is sensitive to quantization in the L-series. Specifically, the Depth-Wise Separable Convolution (DWSCConv) inside the MBConv is causing accuracy degradation when quantized. The Fused-MBConv is an alternative design that avoids the problematic depth-wise operation by fusing it into the layer prior. To facilitate quantization, there is an

argument for avoiding the MBConv in favor of the Fused-MBConv when designing models.

EfficientViT-SAM uses a novel self-attention mechanism that substitutes Softmax for ReLU to achieve linear complexity. It also uses convolutions to scale up the input to the self-attention layer. An effect of ReLU is that its distribution is more friendly to quantize than the skewed Softmax distribution. Our simulation results indicate that the novel mechanism is indeed robust to quantization, as the layers can be quantized individually with no loss or together with minimal loss. However, this is conditional on retaining the MBConvs in 32-bit floating point.

5.1 Future Works

Future work could consist of reducing accuracy degradation using Quantization-Aware Training (QAT). This is a recommendation given by most practitioners as the next step to attempt to improve post-training quantization. This strategy entails simulating quantization, and calibrating the quantized scale factors during training [54]. This thesis focused on applying common quantization techniques only, not specialized methods that often require custom hardware implementations. Future work may include a more in-depth analysis of the distributions of the sensitive MBConv and the novel ReLU linear attention mechanism, to design specialized quantization schemes targeted at these layers. To investigate quantization of a model that uses ReLU linear attention but not MBConv may give additional insights on this novel design.

In this thesis, we focused on quantizing the image encoder of EfficientViT-SAM, however, there are other compression techniques that could be tested. The authors of EfficientViT-SAM, Han Cai et al. [3], mention pruning and efficient hardware hardware-aware neural architecture search as two areas to further investigate.

Bibliography

- [1] T. Akenine-Möller. *Appendix A - Fixed-point Mathematics*. 2007. URL: https://fileadmin.cs.lth.se/cs/Education/EDA075/notes/mgh_appA_fixed.pdf (visited on 2024-05-18).
- [2] M. van Baalen, A. Kuzmin, S. S. Nair, Y. Ren, E. Mahurin, C. Patel, S. Subramanian, S. Lee, M. Nagel, J. Soriaga, and T. Blankevoort. *FP8 versus INT8 for efficient deep learning inference*. arXiv:2303.17951 [cs]. 2023.
- [3] H. Cai, J. Li, M. Hu, C. Gan, and S. Han. *EfficientViT: Multi-Scale Linear Attention for High-Resolution Dense Prediction*. arXiv:2205.14756 [cs]. 2023.
- [4] *Deterministic Tactic Selection - TensorRT Developer Guide*. URL: <https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#deterministic-tactic-selection>.
- [5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. arXiv:2010.11929 [cs]. 2021. URL: <http://arxiv.org/abs/2010.11929> (visited on 2024-01-31).
- [6] E. Eklunden. “Ericsson Technology Review Magazine 2023 – Spotlight on extended reality”. *Ericsson* **110** (2023). ISSN: 0014-0171. URL: <http://www.ericsson.com/ericsson-technology-review> (visited on 2024-03-29).
- [7] T. Elsken, J. H. Metzen, and F. Hutter. *Neural Architecture Search: A Survey*. arXiv:1808.05377 [cs, stat]. 2019.
- [8] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha. *Learned Step Size Quantization*. arXiv:1902.08153 [cs, stat]. 2020.
- [9] K. Feng, L. Lun, X. Wang, and X. Cui. “LRTransDet: A Real-Time SAR Ship-Detection Network with Lightweight ViT and Multi-Scale Feature Fusion”. *en. Remote Sensing* **15**:22 (2023), p. 5309. ISSN: 2072-4292. DOI: 10.3390/rs15225309. URL: <https://www.mdpi.com/2072-4292/15/22/5309> (visited on 2024-05-18).

- [10] N. Frumkin, D. Gope, and D. Marculescu. *Jumping through Local Minima: Quantization in the Loss Landscape of Vision Transformers*. arXiv:2308.10814 [cs]. 2023.
- [11] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer. *A Survey of Quantization Methods for Efficient Neural Network Inference*. arXiv:2103.13630 [cs]. 2021.
- [12] R. Girshick, J. Donahue, T. Darrell, and J. Malik. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Columbus, OH, USA, 2014, pp. 580–587. ISBN: 978-1-4799-5118-5. DOI: 10.1109/CVPR.2014.81.
- [13] S. Gluska and M. Grobman. *Exploring Neural Networks Quantization via Layer-Wise Quantization Analysis*. arXiv:2012.08420 [cs, stat]. 2020.
- [14] *GPU Clock Locking and Floating Clock - TensorRT Developer Guide*. URL: <https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#gpu-clock-lock-float> (visited on 2024-06-16).
- [15] R. Gray and D. Neuhoff. “Quantization”. *IEEE Transactions on Information Theory* **44**:6 (1998), pp. 2325–2383. ISSN: 00189448. DOI: 10.1109/18.720541.
- [16] M.-H. Guo, C.-Z. Lu, Q. Hou, Z. Liu, M.-M. Cheng, and S.-M. Hu. *SegNeXt: Rethinking Convolutional Attention Design for Semantic Segmentation*. arXiv:2209.08575 [cs]. 2022.
- [17] A. Gupta, P. Dollár, and R. Girshick. *LVIS: A Dataset for Large Vocabulary Instance Segmentation*. arXiv:1908.03195 [cs]. 2019.
- [18] S. Gupta and B. Akin. *Accelerator-aware Neural Network Design using AutoML*. arXiv:2003.02838 [cs, eess, stat]. 2020.
- [19] S. Han, H. Mao, and W. J. Dally. *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. arXiv:1510.00149 [cs]. 2016.
- [20] K. He, G. Gkioxari, P. Dollar, and R. Girshick. “Mask R-CNN”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Venice, 2017, pp. 2980–2988. ISBN: 978-1-5386-1032-9. DOI: 10.1109/ICCV.2017.322. URL: <http://ieeexplore.ieee.org/document/8237584/> (visited on 2024-02-05).
- [21] K. He, X. Zhang, S. Ren, and J. Sun. “Deep Residual Learning for Image Recognition”. en. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Las Vegas, NV, USA, 2016, pp. 770–778. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.90.
- [22] G. Hinton, O. Vinyals, and J. Dean. *Distilling the Knowledge in a Neural Network*. arXiv:1503.02531 [cs, stat]. 2015.

- [23] M. Horowitz. “1.1 Computing’s energy problem (and what we can do about it)”. In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, San Francisco, CA, USA, 2014, pp. 10–14. ISBN: 978-1-4799-0920-9. DOI: 10.1109/ISSCC.2014.6757323.
- [24] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. arXiv:1704.04861 [cs]. 2017.
- [25] J. Hu, L. Shen, and G. Sun. “Squeeze-and-Excitation Networks”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. ISSN: 2575-7075. 2018, pp. 7132–7141. DOI: 10.1109/CVPR.2018.00745.
- [26] *IEEE Standard for Floating-Point Arithmetic*. ISBN: 9781504459242. DOI: 10.1109/IEEESTD.2019.8766229.
- [27] *Intro to Quantization - TensorRT Developer Guide*. URL: <https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#intro-quantization> (visited on 2024-06-15).
- [28] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. *Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference*. arXiv:1712.05877 [cs, stat]. 2017.
- [29] Q. Jin, L. Yang, and Z. Liao. *Towards Efficient Training for Neural Network Quantization*. arXiv:1912.10207 [cs]. 2019.
- [30] D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen, J. Yang, J. Park, A. Heinecke, E. Georganas, S. Srinivasan, A. Kundu, M. Smelyanskiy, B. Kaul, and P. Dubey. *A Study of BFLOAT16 for Deep Learning Training*. arXiv:1905.12322 [cs, stat]. 2019.
- [31] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer. *I-BERT: Integer-only BERT Quantization*. arXiv:2101.01321 [cs]. 2021.
- [32] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick. *Segment Anything*. arXiv:2304.02643 [cs]. 2023.
- [33] I. Krasin, T. Duerig, N. Alldrin, V. Ferrari, S. Abu-El-Haija, A. Kuznetsova, H. Rom, J. Uijlings, S. Popov, S. Kamali, M. Mallocci, J. Pont-Tuset, A. Veit, S. Belongie, V. Gomes, A. Gupta, C. Sun, G. Chechik, D. Cai, Z. Feng, D. Narayanan, and K. Murphy. “OpenImages: A public dataset for large-scale multi-label and multi-class image classification.” *Dataset available from <https://storage.googleapis.com/openimages/web/index.html>* (2017).
- [34] R. Krishnamoorthi. *Quantizing deep convolutional networks for efficient inference: A whitepaper*. arXiv:1806.08342 [cs, stat]. 2018.
- [35] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “ImageNet classification with deep convolutional neural networks”. en. *Communications of the ACM* **60**:6 (2017), pp. 84–90. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3065386.

- [36] S. Kullback and R. A. Leibler. “On Information and Sufficiency”. en. *The Annals of Mathematical Statistics* **22**:1 (1951), pp. 79–86. ISSN: 0003-4851. DOI: 10.1214/aoms/1177729694. URL: <http://projecteuclid.org/euclid.aoms/1177729694> (visited on 2024-05-24).
- [37] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition”. *Proceedings of the IEEE* **86**:11 (1998), pp. 2278–2324. ISSN: 1558-2256. DOI: 10.1109/5.726791. URL: <https://ieeexplore.ieee.org/abstract/document/726791> (visited on 2024-02-01).
- [38] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. “Handwritten Digit Recognition with a Back-Propagation Network”. en ().
- [39] J. Lee, Y. Kwon, S. Park, M. Yu, J. Park, and H. Song. *Q-HyViT: Post-Training Quantization of Hybrid Vision Transformers with Bridge Block Reconstruction for IoT Systems*. arXiv:2303.12557 [cs]. 2024.
- [40] Y. Li, H. Mao, R. Girshick, and K. He. “Exploring Plain Vision Transformer Backbones for Object Detection”. en. In: S. Avidan et al. (Eds.). *Computer Vision – ECCV 2022*. Vol. 13669. Series Title: Lecture Notes in Computer Science. Springer Nature Switzerland, Cham, 2022, pp. 280–296. ISBN: 978-3-031-20076-2. DOI: 10.1007/978-3-031-20077-9_17. (Visited on 2024-06-16).
- [41] Z. Li and Q. Gu. *I-ViT: Integer-only Quantization for Efficient Vision Transformer Inference*. arXiv:2207.01405 [cs]. 2023.
- [42] Z. Li, J. Xiao, L. Yang, and Q. Gu. *RepQ-ViT: Scale Reparameterization for Post-Training Quantization of Vision Transformers*. arXiv:2212.08254 [cs]. 2023.
- [43] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang. *Pruning and Quantization for Deep Neural Network Acceleration: A Survey*. arXiv:2101.09671 [cs]. 2021.
- [44] D. D. Lin, S. S. Talathi, and V. S. Annapureddy. *Fixed Point Quantization of Deep Convolutional Networks*. arXiv:1511.06393 [cs]. 2016.
- [45] J. Lin, J. Tang, H. Tang, S. Yang, X. Dang, C. Gan, and S. Han. *AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration*. arXiv:2306.00978 [cs]. 2023. URL: <http://arxiv.org/abs/2306.00978> (visited on 2024-03-06).
- [46] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. *Microsoft COCO: Common Objects in Context*. arXiv:1405.0312 [cs]. 2015.
- [47] Y. Lin, T. Zhang, P. Sun, Z. Li, and S. Zhou. *FQ-ViT: Post-Training Quantization for Fully Quantized Vision Transformer*. arXiv:2111.13824 [cs]. 2023.

- [48] S. Liu and W. Deng. “Very deep convolutional neural network based image classification using small training sample size”. In: *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*. IEEE, Kuala Lumpur, Malaysia, 2015, pp. 730–734. ISBN: 978-1-4799-6100-9. DOI: 10.1109/ACPR.2015.7486599. URL: <http://ieeexplore.ieee.org/document/7486599/> (visited on 2024-02-05).
- [49] X. Liu, H. Peng, N. Zheng, Y. Yang, H. Hu, and Y. Yuan. *EfficientViT: Memory Efficient Vision Transformer with Cascaded Group Attention*. arXiv:2305.07027 [cs]. 2023. URL: <http://arxiv.org/abs/2305.07027> (visited on 2024-05-18).
- [50] Y. Liu, H. Yang, Z. Dong, K. Keutzer, L. Du, and S. Zhang. *NoisyQuant: Noisy Bias-Enhanced Post-Training Activation Quantization for Vision Transformers*. arXiv:2211.16056 [cs]. 2023.
- [51] G. Menghani. *Efficient Deep Learning: A Survey on Making Deep Learning Models Smaller, Faster, and Better*. arXiv:2106.08962 [cs]. 2021.
- [52] J.-M. Muller, N. Brunie, F. De Dinechin, C.-P. Jeannerod, M. Joldes, V. Lefèvre, G. Melquiond, N. Revol, and S. Torres. *Handbook of Floating-Point Arithmetic*. en. Springer International Publishing, Cham, 2018. ISBN: 978-3-319-76526-6. DOI: 10.1007/978-3-319-76526-6.
- [53] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort. *A White Paper on Neural Network Quantization*. arXiv:2106.08295 [cs]. 2021.
- [54] . *Nvidia TensorRT Developer Guide*. URL: <https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html> (visited on 2024-05-05).
- [55] o. bc ovunctuzel. *NaN values with FP16 TensorRT Inference*. 2024. URL: <https://github.com/mit-han-lab/efficientvit/issues/116> (visited on 2024-05-30).
- [56] L. Papa, P. Russo, I. Amerini, and L. Zhou. *A survey on efficient vision transformers: algorithms, techniques, and performance benchmarking*. arXiv:2309.02031 [cs]. 2023.
- [57] A. Polino, R. Pascanu, and D. Alistarh. *Model compression via distillation and quantization*. arXiv:1802.05668 [cs]. 2018.
- [58] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. “You Only Look Once: Unified, Real-Time Object Detection”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Las Vegas, NV, USA, 2016, pp. 779–788. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.91.

- [59] B. Rokh, A. Azarpeyvand, and A. Khanteymoori. “A Comprehensive Survey on Model Quantization for Deep Neural Networks in Image Classification”. *ACM Transactions on Intelligent Systems and Technology* **14**:6 (2023). arXiv:2205.07877 [cs], pp. 1–50. ISSN: 2157-6904, 2157-6912. DOI: 10.1145/3623402.
- [60] F. Rosenblatt. “The perceptron: A probabilistic model for information storage and organization in the brain.” en. *Psychological Review* **65**:6 (1958), pp. 386–408. ISSN: 1939-1471, 0033-295X. DOI: 10.1037/h0042519. URL: <http://doi.apa.org/getdoi.cfm?doi=10.1037/h0042519> (visited on 2024-02-05).
- [61] B. Ryhede Bengtsson. *FPI6 Accuracy failure of TensorRT 8.6.3 when running trtexec built engine on GPU RTX4090*. Issue. 2024. URL: <https://github.com/NVIDIA/TensorRT/issues/3893> (visited on 2024-05-30).
- [62] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. arXiv:1801.04381 [cs]. 2019.
- [63] M. M. H. Shuvo, S. K. Islam, J. Cheng, and B. I. Morshed. “Efficient Acceleration of Deep Learning Inference on Resource-Constrained Edge Devices: A Review”. *Proceedings of the IEEE* **111**:1 (2023), pp. 42–91. ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2022.3226481.
- [64] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. “Going deeper with convolutions”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Boston, MA, USA, 2015, pp. 1–9. ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298594.
- [65] M. Tan and Q. V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. arXiv:1905.11946 [cs, stat]. 2020.
- [66] M. Tan and Q. V. Le. *EfficientNetV2: Smaller Models and Faster Training*. arXiv:2104.00298 [cs]. 2021.
- [67] Y. Tang, Y. Wang, J. Guo, Z. Tu, K. Han, H. Hu, and D. Tao. *A Survey on Transformer Compression*. arXiv:2402.05964 [cs]. 2024.
- [68] *Troubleshooting - TensorRT Developer Guide*. URL: <https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#troubleshooting> (visited on 2024-06-15).
- [69] *Trt_engine_explorer -TensorRT Github*. URL: <https://github.com/NVIDIA/TensorRT/tree/release/10.0/tools/experimental/trt-engine-explorer>.
- [70] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. *Attention Is All You Need*. arXiv:1706.03762 [cs]. 2023.

- [71] Q. Wang, B. Cardiff, A. Frappé, B. Larras, and D. John. *Tiny Models are the Computational Saver for Large Models*. arXiv:2403.17726 [cs]. 2024.
- [72] Z. Wang, M. Berman, A. Rannen-Triki, P. H. S. Torr, D. Tuia, T. Tuytelaars, L. Van Gool, J. Yu, and M. B. Blaschko. *Revisiting Evaluation Metrics for Semantic Segmentation: Optimization and Evaluation of Fine-grained Intersection over Union*. arXiv:2310.19252 [cs]. 2023.
- [73] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius. *Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation*. arXiv:2004.09602 [cs, stat]. 2020.
- [74] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han. *SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models*. arXiv:2211.10438 [cs]. 2023.
- [75] Y. Xiong, B. Varadarajan, L. Wu, X. Xiang, F. Xiao, C. Zhu, X. Dai, D. Wang, F. Sun, F. Iandola, R. Krishnamoorthi, and V. Chandra. *EfficientSAM: Leveraged Masked Image Pretraining for Efficient Segment Anything*. arXiv:2312.00863 [cs]. 2023.
- [76] Z. Yuan, C. Xue, Y. Chen, Q. Wu, and G. Sun. “PTQ4ViT: Post-training Quantization for Vision Transformers with Twin Uniform Quantization”. en. In: S. Avidan et al. (Eds.). *Computer Vision – ECCV 2022*. Vol. 13672. Series Title: Lecture Notes in Computer Science. Springer Nature Switzerland, Cham, 2022, pp. 191–207. ISBN: 978-3-031-19775-8. DOI: 10.1007/978-3-031-19775-8_12. URL: https://link.springer.com/10.1007/978-3-031-19775-8_12 (visited on 2024-02-09).
- [77] S. Yun and A. Wong. *Do All MobileNets Quantize Poorly? Gaining Insights into the Effect of Quantization on Depthwise Separable Convolutional Networks Through the Eyes of Multi-scale Distributional Dynamics*. arXiv:2104.11849 [cs, eess]. 2021.
- [78] C. Zhang, D. Han, Y. Qiao, J. U. Kim, S.-H. Bae, S. Lee, and C. S. Hong. *Faster Segment Anything: Towards Lightweight SAM for Mobile Applications*. arXiv:2306.14289 [cs]. 2023.
- [79] G. Zhang. *Questions about the Reformat layer*. English. 2023. URL: <https://github.com/NVIDIA/TensorRT/issues/2921> (visited on 2024-05-15).
- [80] Z. Zhang, H. Cai, and S. Han. *EfficientViT-SAM: Accelerated Segment Anything Model Without Performance Loss*. arXiv:2402.05008 [cs]. 2024.
- [81] Y. Zheng, M. Wang, Y. Liu, C. Li, and Q. Chang. “Real-Time Helmetless Detection System for Lift Truck Operators Based on Improved YOLOv5s”. *IEEE Access* **12** (2024), pp. 4354–4369. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2024.3349471. URL: <https://ieeexplore.ieee.org/document/10379807/> (visited on 2024-05-18).

- [82] C. Zhou, X. Li, C. C. Loy, and B. Dai. *EdgeSAM: Prompt-In-the-Loop Distillation for On-Device Deployment of SAM*. arXiv:2312.06660 [cs]. 2023.
- [83] X. Zhu, J. Li, Y. Liu, C. Ma, and W. Wang. *A Survey on Model Compression for Large Language Models*. arXiv:2308.07633 [cs]. 2023.

Lund University Department of Automatic Control Box 118 SE-221 00 Lund Sweden		<i>Document name</i> MASTER'S THESIS	
		<i>Date of issue</i> June 2024	
		<i>Document Number</i> TFRT-6250	
<i>Author(s)</i> Bernard Ryhede Bengtsson Joel Bengs		<i>Supervisor</i> Saeed Bastani, Ericsson AB, Sweden Puren Guler, Ericsson AB, Sweden Pontus Giselsson, Dept. of Automatic Control, Lund University Karl-Erik Årzén, Dept. of Automatic Control, Lund University, Sweden (examiner)	
<i>Title and subtitle</i> Accelerated Segmentation with Mixed-Precision Quantization of EfficientViT-SAM			
<i>Abstract</i> The wide adoption of machine learning (ML) models in resource-constrained devices for complex vision tasks is contingent on improved computational efficiency. Quantization is a tool for model compression in which weights and activations are approximated with low-precision datatypes, to achieve a smaller model with reduced latency and memory usage. In this thesis, we seek to apply post-training mixedprecision quantization to the image encoder of EfficientViT-SAM, a state-of-the-art Segment Anything Model that produces segmentation masks for any object in an image. We study the architecture and latency profile of EfficientViT-SAM, and conduct experiments to understand the sensitivity of different architecture components, like layers, on the accuracy achieved. Results show that some convolution layers, in particular depth-wise convolutions, are difficult to quantize without degrading accuracy, while other layers such as the ReLU-based self-attention mechanism used in EfficientViT-SAM are relatively robust under quantization. From these insights, we design and evaluate several quantization schemes that quantize only non-sensitive layers. More specifically, using TensorRT optimization engine, we reduce the latency of EfficientViT-SAM-XL1 by 2.4%, with a loss of 6.9% accuracy. Using a simulation framework, we reduce the size of EfficientViTSAM- L0 by 31% with a loss of 9% accuracy. Our extensive experiments show that a desirable accuracy-latency trade-off is hard, if not infeasible, to reach through uniform quantization to INT8. From our qualitative results, we found that the poorly performing models matched baseline performance in simpler scenarios but failed in predicting complex segmentation masks.			
<i>Keywords</i>			
<i>Classification system and/or index terms (if any)</i>			
<i>Supplementary bibliographical information</i>			
<i>ISSN and key title</i> 0280-5316			<i>ISBN</i>
<i>Language</i> English	<i>Number of pages</i> 1-75	<i>Recipient's notes</i>	
<i>Security classification</i>			

<http://www.control.lth.se/publications/>