A. Private Vision Transformer Inference

A.1. Overview of Private ViT Inference

We illustrate the framework of private ViT inference to help readers to better understand our architecture. As shown in Figure 2, the server holds the model weight while the client holds the input data. The client only send the secret share of data to the server to keep data private and the server keeps the weight private. In MPC, two parties, i.e., server and client, compute functionalities, e.g., linear and non-linear layers jointly with MPC protocols like Cheetah [22] and SEMI-2K [9], and finally the client learns the inference results without extra information about the model weights while the server does not learn any information about the client's data.

A.2. ViT Architecture

For the ViT [11] architecture, it takes image patches as input and is composed of an input projection layer, a stack of Transformer layers, and a task-specific multi-layer perceptron (MLP) head. Each layer consists of an multi-head attention (MHA) layer and an MLP block. With MPC protocols, we have demonstrated that the communication bottleneck mainly comes from non-linear Softmax and GeLU.

Softmax Softmax includes max, exponential and reciprocal operations, all of which are very expensive in MPC:

$$\text{Softmax}(x_i) = \frac{e^{x_i - x_{max}}}{\sum_{j=1}^{n} e^{x_j - x_{max}}}.$$

Note that max is widely used in Softmax to improve numerical stability [60]. Figure 11(a) shows our method to optimize Softmax in MHA.

GeLU GeLU is an activation function based on the Gaussian error function [20], which is defined as:

$$\text{GeLU}(x) = x \cdot \frac{1}{2}[1 + \text{erf}(\frac{x}{\sqrt{2}})],$$

where $\text{erf}(\cdot)$ is the Gaussian error function. For MPC, GeLU is usually approximated with tanh:

$$\text{GeLU}(x) = 0.5x(1 + \tanh[\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)]),$$

or

$$\text{GeLU}(x) = x \cdot \sigma(1.702x).$$

Figure 11(b) shows our method to optimize GeLU in MLP.

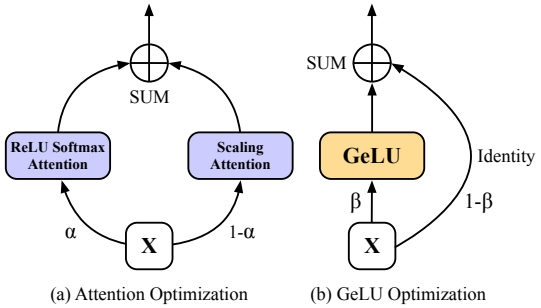

Figure 11. Visualization of our proposed MPCViT and MPCViT$^+$.

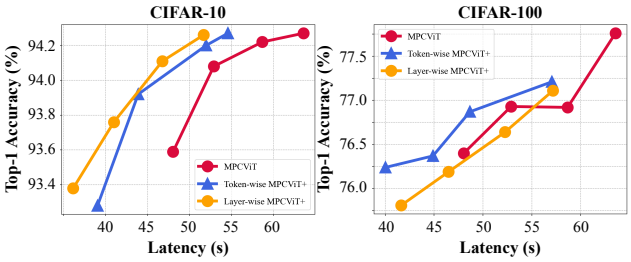

Figure 12. Comparison of layer-wise and token-wise GeLU optimization of MPCViT$^+$.

Exponential Exponential is used in Softmax, but exponential cannot be directly computed in MPC, so it is generally approximated as

$$e^x = \lim_{n \to \infty} (1 + \frac{x}{2^n})^{2^n},$$

where n is the number of approximation iterations.

Reciprocal Reciprocal is widely used in various functions, including Softmax, ReLU Sofmax, 2Quad, etc. Reciprocal in MPC is usually approximated using Newton-Raphson iteration [1]:

$$\frac{1}{x} = \lim_{n \to \infty} y_n = y_{n-1}(2 - xy_{n-1}),$$

where $y_0(x) = 3e^{0.5-x} + 0.003$ which makes the approximation work for a large input domain [30].

B. Cryptographic Primitives for MPC

In this section, we briefly describe the relevant cryptographic primitives for MPC. From the description, we can better understand the communication overhead brought from MPC during model inference.

B.1. Additive Secret Sharing

Additive secret sharing is widely used in arithmetic secret sharing (AS) [52]. Specifically, for an l-bit value $x \in \mathbb{Z}_{2^l}$, it is additively shared between two parties, denoted as $\langle x \rangle_0^l$ and $\langle x \rangle_1^l$, respectively, such that $x = \langle x \rangle_0^l + \langle x \rangle_1^l$

mod 2^l (where $+$ denotes addition in $\mathbb{Z}_{2^l}$). AS is generally implemented by generating a pair $(r, x-r)$, where r is a random number. As illustrated on the left side of Figure 2, we use the share algorithm $\text{Share}^l(x)$ to split an input into two shares. Conversely, we use reconstruction algorithm $\text{Reconst}^l(\langle x\rangle_0^l, \langle x\rangle_1^l)$ recover the actual result to the client. Note that, in AS, the communication of addition operation is free because addition can be locally computed.

B.2. Oblivious Transfer

Oblivious Transfer (OT) is the central cryptographic primitive for building MPC protocols to realize secure ViT private inference. OT [3, 45] enables the receiver to choose one message obliviously from a set of messages sent from the sender without revealing his choice. For 1-out-of-k OT, the sender holds k l-bit messages $m_0, m_1, ..., m_{k-1} \in \{0,1\}$ and the receiver holds a choice bit $b \in [k]$. At the end of OT protocol, the receiver learns m_b but cannot learn any other massages, while the sender learns nothing. Correlated OT (COT) is another form of OT, and 1-out-of-2 correlated OT is widely used, e.g., SiRNN [50]. Specifically, the sender inputs a correlation $x \in \mathbb{Z}_{2^l}$ and the receiver inputs a choice bit $b \in \{0,1\}$. The protocol generates a random value $r \in \mathbb{Z}_{2^l}$ to the sender and $-r + b \cdot x$ to the receiver. $\binom{k}{1}$-COT$_l$ requires $(2\lambda + kl)$-bit and 2 rounds communication.

B.3. Garble Circuit

Garble Circuit (GC) [63] enables two parties to jointly compute an arbitrary function $f(\cdot)$ without revealing their private information. GC has three main phases: 1) garbling, 2) transferring and 3) evaluation. First, the function $f(\cdot)$ is represented as a boolean circuit C. Then, the Garbler encoded the boolean circuit as a garbled circuit $\tilde{C}$ and a set of input-correspondent labels in the first phrase. After garbling phrase, the Garbler sends $\tilde{C}$ to another party who acts as the Evaluator together with the correct labels for the input wires of the circuit. The Evaluator computes the circuit gate-by-gate and produces an encoding of the output. Finally, the Evaluator shares this encoding with the Garbler and learns the actual plaintext result.

C. Details of Attention Variants

In this section , we formally describe the formulations of different attention variants mentioned in §3.

Linformer [58] [60] takes Linformer as an efficient Transformer variant because Linformer significantly reduces the dimenstion of matrix QK^T.

$$\text{Linformer}(Q,K,V) = \text{Softmax}(\frac{Q(EK)^T}{\sqrt{d_k}}) \cdot (FV),$$

where $Q, K, V \in \mathbb{R}^{n \times d_k}$ are queries, keys and values, respectively. $E, F \in \mathbb{R}^{k \times n}$ are two linear projection matrices added on K, V to compress the tensor size of QK^T.

ReLU/ReLU6 Attention ReLU/ReLU6 attention directly replaces Softmax with ReLU/ReLU6. We take ReLU attention as an example and ReLU6 attention can be obtained by simply replacing ReLU with ReLU6 activation.

$$\text{ReLUAttention}(Q,K,V) = \text{ReLU}(\frac{QK^T}{\sqrt{d_k}}) \cdot V.$$

Sparsemax Attention [40] proposes the Sparsemax activation function to enable to output sparse probabilities. Sparsemax is defined as

$$\text{Sparsemax}(z) = \begin{cases} 1, & \text{if} \quad t > 1; \\ (t+1)/2, & \text{if} \quad -1 \le t \le 1; \\ 0, & \text{if} \quad t < -1. \end{cases}$$

Thus, Sparsemax attention is defined as

$$\text{SparsemaxAttention}(Q,K,V) = \text{Sparsemax}(\frac{QK^T}{\sqrt{d_k}}) \cdot V.$$

Note that Sparsemax can not only used for computing the output possibilities, but also for attention through replacing Softmax with Sparsemax in order to remove the expensive exponential. However, Sparsemax requires more comparison operations.

XNorm Attention [53] XNorm is proposed by UFO-ViT [53] and is also called cross-normalization that normalizes Q and K^TV along two different dimensions to construct the linear attention:

$$\begin{aligned} &\text{XNormAttention}(Q,K,V) \\ =&\text{XNorm}_{\text{dim=filter}}(Q)(\text{XN}_{\text{dim=space}}(K^TV)), \end{aligned}$$

$$\text{XN}(a) := \frac{\gamma a}{\sqrt{\sum_{i=0}^{h} \|a\|^2}},$$

where γ is a learnable parameter and h is the hidden dimension.

2Quad Attention [33] 2Quad approximation is proposed by MPCFormer [33], which replaces e^x with $(x+c)^2$ as follows:

$$\text{2QuadAttention}(Q,K,V) = \frac{(\frac{QK^T}{\sqrt{d_k}} + c)^2}{\sum_{i=1}^{n}(\frac{QK^T}{\sqrt{d_k}} + c)_i^2} \cdot V.$$

In our experiments, we set c to a very small value to make the training process robust.

Table 6. Comparison of MPCViT and ReLU Softmax ViT with different number of heads via head pruning method.

Dataset	CIFAR-10		CIFAR-100	
	Accuracy (%)	Latency (s)	Accuracy (%)	Latency (s)
1-head	92.48	50.88	73.25	51.12
2-head	92.83	57.65	73.99	57.84
3-head	93.03	66.21	74.61	66.70
MPCViT	**93.38**	63.56	**75.38**	63.79

D. The Algorithm Flow of MPCViT

Here, we show the algorithm details of our proposed MPCViT pipeline. As shown in Algorithm 1, the pipeline is mainly divided into two steps: search and retrain. We first initialize a ReLU Softmax ViT and jointly optimize the network θ and architecture parameter α. After searching, we selectively replace a set of ReLU Softmax attention with an MPC-eficient Scaling attention based on the *top-k* rule. Then, in order to boost the performance of MPCViT with heterogeneous attention, we retrain the ViT with knowledge distillation. The algorithm flow is almost the same with MPCViT$^+$, and we can jointly optimize network θ and two architecture parameters α, β during the search.

E. Layer-Wise VS. Token-Wise GeLU Optimizatin of MPCViT$^+$

The choice of GeLU optimization can be different granualirties including layer-wise and token-wise, both of which support to fuse two linear layers for a better efficiency. Experiments in §5.5 use token-wise granularity as an example, and here we compare layer-wise and token-wise GeLU optimization for a better choice. Since the proportion of GeLU and MatMul are small in the ViT model on Tiny-ImageNet, we here consider the model on CIFAR-10 and CIFAR-100 as shown in Figure 12. The results are evaluated with KD. On CIFAR-10, MPCViT$^+$ with layer-wise GeLU optimization has a little better Pareto front than token-wise optimization, while on CIFAR-100, token-wise optimization is a little better than layer-wise optimization.

F. More Distributions of Attention Architecture Parameters

In §5.4, we enumerate four situations to show the consistency and scalability of our NAS algorithm. To empirically verify the consistency more sufficiently, we supplement more cases of architecture parameter α for each attention head. On CIFAR-10, we fix the number of heads to 4 and modify the hyper-parameter λ to even smaller values, i.e., 10^{-5} and 10^{-6}. As shown in Figure 13, the trend is still similar under different settings as Figure 9.

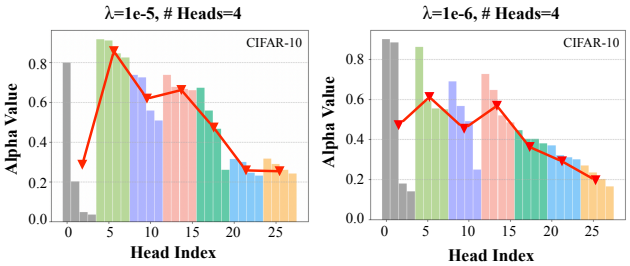

Figure 13. The distribution of architecture parameter α on CIFAR-10 with different Lasso coefficient λ.

G. More Analysis: MPCViT Optimization VS. Pruning Method

Our method is similar to pruning as we aim at removing "unimportant" modules in NNs. Like [8], here we analyze the advantage of our method.

Many methods [41, 56, 62] prune a subset of attention heads to improve the efficiency of Transformers. The formulation of head pruning is defined as

$$\text{MHA} = \sum_{i=1}^{N} z^{(i)} \text{Att}(Q^{(i)}, K^{(i)}, V^{(i)}),$$

where $z^{(i)} \in \{0, 1\}$ is a mask variable for MHA. However, this way losses the benefit of multi-head, leading to a worse representation ability. Here, we give an example shown in Table 6. Note that the ViT architecture on CIFAR-10/100 cannot support three heads since the hidden dimension is 256, so we just modify 256 to 258 with a negligible latency change. As we can observe, MPCViT outperforms head pruning with higher accuracy and lower latency. The result also indicates the necessity of including the ScaleAttn in MPCViT. Instead of cutting attention heads, our method selectively replaces expensive attention with MPC-efficient attention without compromising the accuracy.

For MPCViT$^+$, according [8], our proposed GeLU linearization actually reduces the GeLU count while unstructured pruning still remains more GeLUs. Compared with structured pruning, MPCViT maintains more parameters in the network [8], achieving a better performance.

Algorithm 1: Pipeline of Our Proposed MPCViT

Input: ViT with ReLU Softmax attention: f_θ; ratio of RSAttn budget μ; searching epochs: E_s; training epochs: E_t; Lasso coefficient: λ; total number of ViT heads: N.

Initialize the architecture parameter $\alpha = 1.0$ for all attention heads in f_θ.

$\overline{\theta} \leftarrow (\theta, \alpha)$.

while *epoch* $\leq E_s$ **do**

  Compute loss: $\mathcal{L}_{search}$ with the $\ell 1$-penalty term;

  Update $\overline{\theta}$ with AdamW optimizer;

  Adjust learning rate with the cosine scheduler.

Sort the alpha values across the heads in $f_{\overline{\theta}}$, and find the μN-th largest α, denoted as α^*.

if $\alpha \geq \alpha^*$ **then**

  $\alpha \leftarrow 1.0$

else if $\alpha < \alpha^*$ **then**

  $\alpha \leftarrow 0.0$

Obtain the searched heterogeneous ViTs under different latency constraints with binarized α : $f_{\theta'}$.

Fix α and retrain $f_{\theta'}$ to improve its accuracy as follows:

while *epoch* $\leq E_t$ **do**

  Compute loss: $\mathcal{L}_{train}$ with two types of KD techniques, i.e., $\mathcal{L}_{logists}$ and $\mathcal{L}_{feature}$;

  Update θ' with AdamW optimizer;

  Adjust learning rate with the cosine scheduler.

Output: Accurate and efficient MPC-friendly ViTs with heterogeneous attention $f_{\theta'}$.

Linearize ScaleAttn by scaling factor decomposition during inference time to accelerate computation. // Optional