CRYPTPEFT: Efficient and Private Neural Network Inference via Parameter-Efficient Fine-Tuning

Saisai Xia*†, Wenhao Wang*†✉, Zihao Wang‡✉, Yuhui Zhang*†, Yier Jin§, Dan Meng*†, Rui Hou*†

*State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, CAS
†School of Cyber Security, University of Chinese Academy of Sciences
‡Nanyang Technological University
§University of Science and Technology of China

Abstract*—Publicly available large pretrained models (i.e., backbones) and lightweight adapters for parameter-efficient fine-tuning (PEFT) have become standard components in modern machine learning pipelines. However, preserving the privacy of both user inputs and fine-tuned adapters—often trained on sensitive data—during inference remains a significant challenge. Applying cryptographic techniques, such as multi-party computation (MPC), to PEFT settings still incurs substantial encrypted computation across both the backbone and adapter, mainly due to the inherent two-way communication between them. To address this limitation, we propose CRYPTPEFT, the first PEFT solution specifically designed for private inference scenarios. CRYPTPEFT introduces a novel *one-way communication (OWC)* architecture that confines encrypted computation solely to the adapter, significantly reducing both computational and communication overhead. To maintain strong model utility under this constraint, we explore the design space of OWC-compatible adapters and employ an automated architecture search algorithm to optimize the trade-off between private inference efficiency and model utility. We evaluated CRYPTPEFT using Vision Transformer backbones across widely used image classification datasets. Our results show that CRYPTPEFT significantly outperforms existing baselines, delivering speedups ranging from $20.62\times$ to $291.48\times$ in simulated wide-area network (WAN) and local-area network (LAN) settings. On CIFAR-100, CRYPTPEFT attains 85.47% accuracy with just 2.26 seconds of inference latency. These findings demonstrate that CRYPTPEFT offers an efficient and privacy-preserving solution for modern PEFT-based inference.

I. INTRODUCTION

Pre-trained models have revolutionized machine learning, serving as the foundation for a variety of downstream applications [11], [12], [19]. By leveraging large-scale datasets and substantial computational resources, these models often learn representations that rival or exceed human-level performance. Subsequently, fine-tuning [55] re-purposes a pre-trained network for a specific task without necessitating training from scratch. For example, a model initially designed for general image classification can be adapted to detect tumors in medical scans [35], [2], [37], automatically tag images on social media [27], [49], or recognize traffic signs in autonomous vehicles [13], [45]. Such flexibility facilitates the specialization of foundational models and yields highly personalized services.

Despite these advantages, deploying fine-tuned models in user-facing contexts raises pressing privacy concerns. From the service provider's perspective, there is an incentive to maintain confidentiality of proprietary model parameters, while users often hesitate to disclose potentially sensitive input data. As a result, privacy-preserving inference[1] based on secure multi-party computation (MPC) [31] has emerged as a promising solution to protect both the intellectual property of service providers and the confidentiality of user queries. However, existing secure computation protocols are often burdened by substantial communication and computational overhead [38], [39], [47], [48]. To alleviate these overheads, prior work has primarily focused on optimizing the resource-intensive low-level operations within MPC protocols, leading to the development of MPC-friendly approximations intended to expedite the inference process [34], [43], [42], [56], [4], [54]. While such optimizations can reduce overhead, their effectiveness diminishes as models increase in complexity and size. When applied to extremely large architectures, these approaches may become impractical or infeasible.

In contrast, our study focuses on the development of MPC-friendly architectures tailored for fine-tuning, thereby addressing the fundamental challenges posed by the increasing size of modern models. By rethinking model design at the architectural level, we aim to ensure that privacy-preserving inference can scale effectively with minimal overhead. This emerging research paradigm has recently attracted attention in pioneering works [32], [33], [52], [46]. Some of these studies

Corresponding authors: Wenhao Wang (wangwenhao@iie.ac.cn) and Zihao Wang (zihao.wang@ntu.edu.sg).

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.231102
www.ndss-symposium.org

[1] We use the terms "privacy-preserving inference" and "private inference" interchangeably throughout this paper.

emphasize constructing compact models through techniques such as compressing or distilling large and powerful networks [32], [33], [52]. While these methods achieve improved efficiency, they often involve trade-offs that compromise model utility, which can be untenable in many practical applications. More recently, Rathee et al. [46] proposed an alternative approach that retains the foundational architecture of the model, fine-tuning only the final layers to indirectly reduce the overall model size for privacy-preserving inference. While this method mitigates some utility losses, it remains hindered by persistent efficiency bottlenecks (Sec. VI-C).

Our solution. To address this challenge, we draw inspiration from Parameter-Efficient Fine-Tuning (PEFT) methodologies [26], [13], [9], [53], [24], which introduce a small set of trainable parameters—commonly referred to as *adapters*—into a pre-trained *backbone model* while keeping most of the original weights frozen. In standard (i.e., unencrypted plaintext) fine-tuning settings, PEFT has demonstrated both efficiency and robustness, often outperforming approaches that fine-tune only the final layers [9]. Given these advantages, it is natural to consider extending PEFT to private inference scenarios, where one might hope to perform the bulk of computation in plaintext on the public backbone, while limiting MPC-based encrypted computation to the adapter components—thus significantly improving overall inference efficiency. Unfortunately, our study reveals that *existing PEFT methods are fundamentally incompatible with private inference due to architectural limitations*. Specifically, conventional adapter designs require their (encrypted) outputs to be decrypted before being fed back into the backbone, creating a potential leakage point (c.f., Sec. III-B). These limitations highlight the need for novel PEFT architectures that can operate entirely on encrypted data, without sacrificing privacy or efficiency.

In this paper, we propose CRYPTPEFT, the first PEFT-based solution specifically designed for private inference. Our key insight is that conventional PEFT techniques inherently introduce a *two-way communication (TWC)* mechanism between the backbone and the adapter, necessitating multiple encrypt-decrypt cycles. In contrast, CRYPTPEFT enforces a *one-way communication (OWC)* policy, ensuring that data flows unidirectionally from the backbone to the adapter without being fed back. By eliminating intermediate decryption, CRYPTPEFT allows all computations to remain fully encrypted, thereby preserving strict privacy guarantees.

A natural concern arising from the removal of feedback loops, as mandated by the OWC policy, is whether such a restriction compromises the model's expressive power. To answer this, we begin with an empirical study to evaluate the impact of OWC on model performance across various downstream tasks. The results show that existing adapter architectures often suffer from reduced utility under OWC constraints. Based on this observation, we analyze how adapter structure and placement affect both utility and efficiency in private inference scenarios (Sec. IV-A). Guided by these insights, we design a new adapter architecture specifically tailored for OWC-compliant settings. A central component of this design is `LinAtten`, a lightweight attention mechanism that replaces the traditional Softmax with MPC-friendly operations such as addition and multiplication. `LinAtten` significantly reduces the cost of private inference while maintaining high utility. To further enhance efficiency, CRYPTPEFT adopts the Low-Rank Adaptation (LoRA) framework [26], which reduces computational and communicational overhead through low-rank decomposition.

Finally, CRYPTPEFT integrates a Neural Architecture Search (NAS) mechanism [20] to automatically identify adapter configurations that optimize the trade-off between efficiency and utility for a given downstream task. Unlike conventional NAS in plaintext setting, which typically assumes computational cost scales proportionally with model size, private inference introduces fundamentally different cost dynamics. To enable cost-aware NAS, we profiled all key operations—including linear layers, matrix multiplications, normalization, and activation functions—under our target network environment, and derived cost estimation models based on the profiling results. Guided by these cost models, our NAS strategy is tailored to identify architectures that satisfy a target utility while minimizing private inference cost (Sec. IV-D).

Evaluations. In this study, we primarily focus on vision-related tasks, aligning with the majority of existing research in the field of private inference. Nonetheless, we view language models as a natural extension of our framework and plan to explore them in future work. Specifically, we built CRYPTPEFT using public vision backbones (i.e., ViT-B), and evaluated its performance on widely-used image datasets, including CIFAR-10 [28], CIFAR-100 [3], Food-101 [6], SVHN [22] and Flowers-102 [41]. For each dataset, we identified the most suitable adapters through our NAS-based search procedure.

We developed an end-to-end inference system for CRYPTPEFT using the MPC framework CRYPTEN [1] and evaluated its private inference efficiency. Our experimental results show that CRYPTPEFT achieves significant reductions in inference time compared to the baseline methods. Specifically, in a simulated wide-area network (WAN) environment characterized by a 400 Mbps bandwidth and 4 ms latency, it achieves an average speedup of $250.39\times$ over the traditional PEFT baseline method, and an average speedup of $20.62\times$ compared to the baseline that simply fine-tunes only the last layer. CRYPTPEFT also improves the model accuracy by 0.83% on average over fine-tuning the last layer. Taking the CIFAR-100 dataset as an example, CRYPTPEFT achieves a model accuracy of 85.47% and an inference time of just 2.26 seconds.

Contributions. Our key contributions are outlined below:

- *New observations*. We propose CRYPTPEFT, the first PEFT approach tailored to private inference scenarios. CRYPTPEFT employs a one-way communication (OWC) strategy, completely removing the need for decryption within the network, guaranteeing strict end-to-end privacy.
- *New designs*. Building on insights into adapter behavior under OWC constraints, we propose a redesigned adapter ar-

chitecture and a NAS-guided search strategy to jointly enhance the utility and efficiency of private inference in CRYPTPEFT.
• *Extensive empirical studies*. We implement CRYPTPEFT on a standard vision backbone model, and evaluate it on widely used image datasets, including CIFAR-10, CIFAR-100, Food-101, SVHN and Flowers-102. CRYPTPEFT surpasses existing solutions in both classification accuracy and inference efficiency, highlighting its potential for real-world, privacy-preserving deployment. The source code for our research is publicly available at https://github.com/Saisai-Xia/CryptPEFT.

II. BACKGROUND

A. Vision Transformer (ViT)

Vision Transformer (ViT) is a neural network architecture that applies the Transformer model—originally developed for natural language processing—to image recognition tasks by treating images as sequences of patch embeddings. It has demonstrated competitive or superior performance compared to convolutional neural networks (CNNs), especially when trained on large-scale datasets. A typical ViT consists of the Patch Embedding layer, Position Embedding layer, Transformer Encoder layer, and Classifier [19]. Given an image $\mathbf{x} \in \mathbb{R}^{H\times W\times C}$ with C channels and resolution (H, W), the Patch Embedding layer divides $\mathbf{x}$ into multiple patches and flattens them into $\mathbf{x}_p \in \mathbb{R}^{N\times(P^2\cdot C)}$, where (P, P) denotes the resolution of each image patch, and $N = \frac{H\times W}{P^2}$ represents the number of image tokens. Each row vector in $\mathbf{x}_p$ is then mapped to a D-dimensional space and concatenated with a learnable `[CLS]` token for further processing. The Position Embedding layer adds a set of learnable positional encodings to the output of the Patch Embedding layer to retain spatial information, resulting in a sequence of image tokens.

The Transformer Encoder layer consists of multiple identical Transformer layers. Each Transformer layer includes a Multi-Head Self-Attention (MHSA) layer and a Multi-Layer Perceptron (MLP) layer. For example, the image tokens $\mathbf{x}_{j-1}$ output by the $(j-1)$-th Transformer layer are first transformed into three vectors: Q, K, and V. These vectors are then used in the attention calculation within the MHSA layer of the j-th Transformer layer:

$$\mathbf{x}_j^{'} = \text{Attention}(Q, K, V) = \text{Softmax}(QK^T/\sqrt{d_k})V,$$

where d_k represents the feature dimension of each head in the MHSA. The output $\mathbf{x}_j^{'}$ is then passed through LayerNorm and the MLP. The computation can be formalized as:

$$\mathbf{x}_j = \text{MLP}(\text{LN}(\mathbf{x}_j^{'})) + \mathbf{x}_j^{'}.$$

For the subsequent Transformer layers, the computation of $\mathbf{x}_j$ continues as described above. The final Transformer layer outputs a set of image tokens, from which the previously appended `[CLS]` token is extracted. This `[CLS]` token, having captured global information, is then passed to the Classifier to complete the classification task. For more details on ViT, please refer to [19].

B. Parameter-Efficient Fine-Tuning (PEFT)

ViTs leverage global self-attention mechanisms across image patches, which enables strong representation learning but also results in a higher parameter count compared to traditional convolutional neural networks (CNNs). Fine-tuning ViTs is particularly computationally expensive, as it typically involves updating all model parameters. When adapting a pre-trained model to multiple downstream tasks, conventional fine-tuning necessitates storing separate parameter sets for each task, incurring significant storage and compute overhead. Parameter-Efficient Fine-Tuning (PEFT) offers a practical alternative by introducing a small set of trainable parameters, such as adapters or low-rank matrices, while keeping the backbone frozen. In this section, we detail AdaptFormer [9] and Low-Rank Adaptation (LoRA) [26], two representative PEFT methods that serve as baselines in our evaluations.

AdaptFormer. AdaptFormer [9] enables ViTs to perform downstream tasks by tuning lightweight task-specific adapters, rather than the entire model. These adapters, often implemented as low-rank matrices inserted into selected layers, significantly reduce the number of trainable parameters, improving efficiency in multi-task and transfer learning scenarios.

LoRA. Low-Rank Adaptation (LoRA) [26] is a technique that reduces fine-tuning overhead for large pre-trained models by applying low-rank decomposition. It decomposes the weight matrices of the pre-trained model into low-rank matrices, which reduces the number of parameters requiring optimization during fine-tuning. LoRA is commonly used with large language models like GPT and BERT, and can also be applied to architectures such as ViT. LoRA's key advantage is its ability to adapt to new tasks or datasets with minimal parameter changes, avoiding the need to retrain the entire model.

C. MPC-based Private Inference

Private inference aims to protect sensitive data during model inference by employing cryptographic techniques such as multi-party computation (MPC) and homomorphic encryption (HE). In this work, we build our inference system using the privacy-preserving machine learning framework CRYPTEN [30], which implements secure computation protocols based on secret sharing. This section provides essential background on secret sharing based MPC protocols.

Secret sharing. Arithmetic secret sharing involves dividing a scalar value $x \in \mathbb{Z}/Q\mathbb{Z}$, where $\mathbb{Z}/Q\mathbb{Z}$ represents a finite ring of integers modulo Q, across a set of parties $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$. The sharing of x is represented by $[x] = \{[x]_p\}_{p\in\mathcal{P}}$, where each $[x]_p \in \mathbb{Z}/Q\mathbb{Z}$ denotes the share held by party p. The shares are constructed such that their sum (modulo Q) reconstructs the original value, i.e.,

$$x = \sum_{p\in\mathcal{P}} [x]_p \pmod{Q}.$$

To generate the shares for a value x, the parties first create a *zero-share* vector consisting of $|\mathcal{P}|$ random numbers, ensuring their sum equals zero [16]. One party, typically the one with

access to the secret x, adds the value x to its share and then discards the original value. The sum of all shares will reconstruct x without revealing the secret to any single party.

Binary secret sharing is a specific case of arithmetic secret sharing that operates in the binary field $\mathbb{Z}/2\mathbb{Z}$, where arithmetic is performed modulo 2. In this setting, each party $p \in \mathcal{P}$ holds a share $\langle x \rangle_p \in \{0, 1\}$, such that the reconstruction condition is given by:

$$x = \bigoplus_{p \in \mathcal{P}} \langle x \rangle_p,$$

where $\bigoplus$ denotes the bitwise XOR operation. Binary secret sharing is often used in applications when binary operations are efficient.

Conversions. The conversion from an arithmetic secret share $[x]$ to a binary secret share $\langle x \rangle_p$ is performed by first converting each party's arithmetic share $[x]_p$ into binary shares. Each party creates a binary secret share, $\langle [x]_p \rangle$, which represents the bits of their share $[x]_p$. These binary shares are then combined by summing them to produce the binary secret share $\langle x \rangle$, which represents the original value x in binary form. Specifically, each party $p \in \mathcal{P}$ generates a binary secret share $\langle [x]_p \rangle$ of their share $[x]_p$, and the parties then compute the total binary share $\langle x \rangle$ by summing the individual binary shares:

$$\langle x \rangle = \sum_{p \in \mathcal{P}} \langle [x]_p \rangle.$$

This process typically uses a carry-lookahead adder to handle the binary addition efficiently, and it can be completed in $\mathcal{O}(\log_2 |\mathcal{P}| \cdot \log_2 Q)$ communication rounds, where $|\mathcal{P}|$ is the number of parties and Q is the modulus in the original arithmetic secret sharing scheme [8], [17].

The conversion from $\langle x \rangle_p$ to $[x]$ is done by reconstructing the arithmetic share from the individual bits of the binary share $\langle x \rangle$. Specifically, the arithmetic secret share is computed as:

$$[x] = \sum_{b=1}^{B} 2^b \cdot [\langle x \rangle^{(b)}],$$

where $\langle x \rangle^{(b)}$ denotes the b-th bit of the binary secret share $\langle x \rangle$, and B is the total number of bits needed to represent the binary share. The sum of the weighted bits reconstructs the original arithmetic value x.

Private linear functions. To perform private addition of two secret-shared values $[z] = [x] + [y]$, each party $p \in \mathcal{P}$ adds their respective shares of x and y. That is, each party computes:

$$[z]_p = [x]_p + [y]_p,$$

where $[x]_p$ and $[y]_p$ are the individual shares of x and y held by party p, and $[z]_p$ is the corresponding share of the sum $z = x + y$. Private multiplication between secret-shared values $[x]$ and $[y]$ is more complex and is typically implemented using pre-computed Beaver triples [5]. A Beaver triple consists of three secret-shared values $([a], [b], [c])$, where $c = a \cdot b$. Specifically, the parties compute $[\epsilon] = [x] - [a]$ and $[\delta] = [y] - [b]$, and decrypt ϵ and δ without information leakage due to the masking. They compute the result $[x][y] = [c] + \epsilon[b] + [a]\delta + \epsilon\delta$, using trivial implementations of addition and multiplication of secret shares with public values. Private addition and multiplication can be used to implement a variety of linear functions in neural networks, such as dot products, outer products, matrix products, and convolutions.

Private non-linear functions. Non-linear functions commonly used in neural networks, such as ReLU, Sigmoid, and Softmax, are typically implemented using approximations that combine private additions, multiplications, and comparisons. Private comparisons are implemented using a function that evaluates whether a secret-shared value $[z]$ is less than 0 (i.e., evaluates $[z < 0]$) by following a series of steps: (1) First, the secret-shared value $[z]$ is converted to its binary secret share $\langle z \rangle$. This conversion splits $[z]$ into individual bits, where each bit is shared among the parties. (2) The sign of z can be determined by evaluating the most significant bit (MSB) of the binary representation of z. The sign bit $\langle b \rangle$ is computed by performing a bit shift operation:

$$\langle b \rangle = \langle z \rangle \gg (L - 1),$$

where L is the number of bits used to represent the secret-shared value z, and the right shift operation extracts the sign bit. If z is negative, the MSB will be 1; otherwise, it will be 0. (3) Finally, the resulting binary sign bit $\langle b \rangle$ is converted back to an arithmetic secret share $[b]$. This allows the parties to compute the comparison securely, which can then be used for further operations such as determining the output of the ReLU function.

III. CryptPEFT Overview

A. Parameter-Efficient Fine-Tuning based Private Inference

Secure computation protocols are well-suited for efficiently handling linear operations (e.g., addition) and low-degree polynomial operations (e.g., multiplication). However, they incur substantial overhead when applied to operations such as comparison, division, or complex nonlinear computations. As a result, neural network operators that are efficient in plaintext settings often become prohibitively expensive under private inference. Prior work on accelerating private inference has largely focused on operator-level approximations or low-level protocol optimizations. In comparison, we aim to investigate the architectural compatibility between neural network designs and the constraints imposed by secure computation. Specifically, we focus on Parameter-Efficient Fine-Tuning (PEFT) methods, which have become standard practice in modern machine learning workflows.

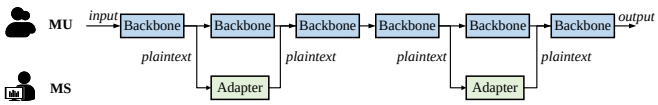
System model. In our settings, we consider a common incentive mechanism in which the model service provider (**MS**) offers secure model inference as a paid service, charging clients per query or based on usage. The model is fine-tuned on proprietary data for specific downstream tasks, while leveraging a publicly available backbone model. These proprietary data—such as medical diagnostic images—are highly

valuable and specifically tailored for certain downstream tasks. Meanwhile, a model user (**MU**) has access to the same public backbone but lacks access to the provider's private training data. To leverage the fine-tuned model's enhanced capabilities for a specific task, the **MU** submits its processed input to the **MS** and receives the corresponding inference results.

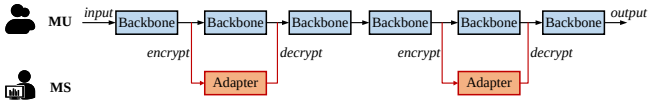
A key challenge in this interaction is preserving the privacy of both parties. The **MU** seeks to protect the confidentiality of its input, ensuring that no sensitive information is leaked during inference. Simultaneously, the **MS** aims to safeguard the proprietary adapter model from unauthorized access or reconstruction. Notably, since the **MU** can observe both the input it provides and the output it receives, some information about the adapter is inherently exposed, potentially making the model susceptible to model inversion attacks. However, we argue that in our setting, the information naturally inferable from the input-output pairs is necessary. Our method ensures that beyond this unavoidable exposure, the **MU** gains no additional knowledge about the adapter's private parameters. As such, we consider the defenses against model inversion attacks orthogonal to our work. Our research aims to develop efficient private inference techniques that protect the sensitive information of both parties. In this scenario, the **MU** and the **MS** collaboratively run the following steps:

- ***MU-side pre-processing.*** The **MU** initially performs local computations on the plaintext backbone model using its input data. It then encrypts the resulting intermediate outputs and sends this encrypted data to the **MS**, who possesses a specialized adapter.
- ***MS-side computation.*** Upon receiving the encrypted intermediate data, the **MS** processes it using the adapter, ensuring that all computations remain in encrypted form. This involves performing private linear and nonlinear operations collaboratively with the **MU**, utilizing secure computation protocols outlined in Sec. II-C.
- ***Result retrieval.*** The **MS** sends the encrypted output to the **MU**, who decrypts it to obtain the final inference result.

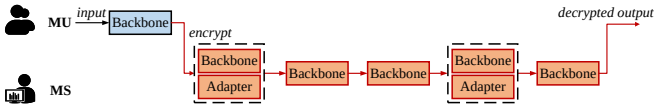
Threat model. To formalize the privacy guarantees, we adopt the semi-honest (i.e., honest-but-curious) adversarial model. This implies that both parties adhere to the prescribed protocol but may attempt to extract additional information from the data they receive or observe during the interaction. This setup is practical since the server is driven by financial incentives to adhere to protocols and deliver high-quality services, while the client is encouraged to comply with the protocol to access those services. As a result, this semi-honest model is widely used in existing research. However, our approach can be extended to support malicious adversaries that may arbitrarily deviate from the protocol. Specifically, the protocols can be strengthened by incorporating homomorphic message authentication codes (MACs) [18], a standard technique in authenticated secret sharing, which enable each party to verify the correctness of computations performed by others without revealing private data. Under this threat model, two key privacy objectives must be met:



(a) Plaintext inference.



(b) Encrypt-then-decrypt inference (*insecure*).



(c) Private inference (*secure but slow*).

Fig. 1: Inference workflows in different scenarios. Orange-shaded components indicate computations performed under secure computation protocols.

- ***Confidentiality of MU's input.*** The **MS** should not learn any information about the private input data of **MU**. This guarantee is especially crucial when the input data includes proprietary or sensitive information, such as personal health records or confidential business data.
- ***Confidentiality of MS's private adapter.*** The **MU** obtains the output corresponding to their own input but cannot obtain any additional information of the **MS**'s private adapter. Any compromise in this regard could undermine the **MS**'s intellectual property and competitive edge.

B. One-way Communication Policy

Motivations. In this paper, we focus on PEFT methods such as AdaptFormer [9] and LoRA [26], which introduce a small number of trainable parameters while keeping the majority of pre-trained weights frozen. This approach has been shown to significantly reduce training time, memory consumption, and fine-tuning overhead. As illustrated in Fig. 1a, inference in plaintext settings typically involves separate computation on the backbone and the adapter, followed by merging intermediate representations before proceeding through subsequent layers. In such environments, PEFT consistently outperforms simpler strategies like fine-tuning only the final layers.

However, in the private inference scenario, existing PEFT techniques typically require bidirectional communication and partial decryption between the backbone and adapter, introducing potential leakage points. Specifically, a naive (and inherently *insecure*) approach to applying PEFT in this setting is illustrated in Fig. 1b: the **MU** uses the public backbone to compute intermediate results in plaintext, encrypts these results, and sends them to the **MS**, which holds the adapter. The **MS** then processes it with the adapter using secure computation protocols, and returns the partial output to the **MU** so that the **MU** can decrypt them and continue the inference process. While this design minimizes the computational load by retaining

plaintext operations in the backbone, it directly violates the privacy objectives set forth in Sec. III-A. As a result, conventional PEFT architectures that depend on iterative interactions between the backbone and adapter are fundamentally incompatible with the strict end-to-end privacy guarantees expected in private inference. To fully meet those objectives, most computations—including those performed by the backbone—must be performed over encrypted data (Fig. 1c). This requirement renders the naive design impractically slow and resource-intensive, highlighting the core challenge of applying PEFT in private inference scenarios.

Solution. A critical challenge in private inference lies in ensuring that neither the **MS** nor the **MU** obtains unnecessary information beyond what is essential for producing the final result. Traditional fine-tuning pipelines often rely on iterative or bidirectional data exchange, a process we refer to as *two-way communication* (TWC). To tackle the challenges above, we introduce a key insight for reconciling PEFT with private inference: *one-way communication (OWC)*. This approach involves transferring data *only once and in a single direction*, eliminating the need for decryption or the back-and-forth exchange of intermediate representations.

In TWC, each step typically involves decrypting partial outputs and exchanging them between components (e.g., the adapter and backbone). This intermediate decryption poses significant risks, as it exposes internal features that malicious or curious parties could exploit to extract private information. In contrast, OWC eliminates this vulnerability by enforcing that intermediate activations never return to earlier layers or to the other party in plaintext. Instead, all data processing—including that performed by the adapter—happens in a forward-only manner. Specifically, any intermediate representation remains in encrypted form as it passes through the model. Critically, *no* step in the pipeline requires decrypting these activations until the final output is ready for the legitimate recipient. This design intrinsically aligns with MPC's principle of keeping all intermediate computations locked behind cryptographic methods, thereby eliminating leakage risks.

C. CryptPEFT Workflow

In this paper, we propose CryptPEFT, the first PEFT architecture specifically designed for private PEFT inference scenarios. Fig. 2 provides an overview of CryptPEFT's workflow, illustrating the secure collaboration between the **MU** and the **MS** while ensuring compliance with the OWC policy.

Specifically, the **MU** holds a public backbone network, which it uses to perform local inference on the input data in plaintext. When specialized adapter-based processing, owned by the **MS**, is required, the **MU** encrypts the intermediate results using an MPC-based secure protocol. These encrypted intermediate results are then transmitted to the **MS** for secure processing using the adapter. Notably, the **MU** does not need to wait for the **MS**'s response to continue its local, plaintext computations. The **MS** focuses on receiving ciphertext from the **MU**, performing private inference using the adapter, and returning the encrypted prediction results. Upon receiving these encrypted outputs, the **MU** decrypts them to produce the final inference results. Throughout this workflow, only the final prediction result is transmitted from the **MS** to the **MU** in ciphertext form—a necessary step—while all other intermediate results flow unidirectionally from the **MU** to the **MS** in encrypted form.

A potential concern with removing feedback loops is whether it diminishes the overall expressive power of the model. Traditional architectures often rely on residual or skip connections, effectively implementing partial "feedback" of intermediate activations. In practice, many deep learning models already exhibit strong utility even when most layers remain unaltered. Coupled with PEFT, our experience has shown that carefully designed adapter structures and the strategic placement of adapters in deeper layers (while maintaining a unidirectional flow) can maintain robust model utility. This approach preserves the core objective of fine-tuning, i.e., adapting pre-trained models to new tasks or data distributions, while adhering to the OWC policy essential for efficient private inference. The details for the design of OWC-compliant adapters will be presented in Sec. IV.

Ultimately, OWC introduces a novel design paradigm for integrating cryptographic techniques with modern PEFT architectures. Instead of retrofitting existing models—often burdened by bidirectional data flows—OWC serves as a guiding principle that informs model construction to inherently protect intermediate computations. As elaborated in the following sections, OWC is foundational to building secure and parameter-efficient pipelines, enabling the adaptation of large pre-trained models to new tasks under strict privacy constraints without incurring excessive overhead.

IV. CryptPEFT Design Details

In Sec. III-B, we established that enforcing OWC policy is essential for enabling efficient PEFT-based private inference. However, traditional adapter architectures experience a notable degradation in accuracy when subject to OWC constraints. To overcome this challenge, we begin with key empirical observations and design principles in Sec. IV-A, which serve as the foundation for our approach. Based on these principles, we describe the architecture and placement strategies for the adapters in Sec. IV-B and Sec. IV-C, respectively. In Sec. IV-D, we introduce an algorithm that automates the search for optimal adapter configurations.

A. Understanding OWC-Compliant Adapters

In compliance with the OWC policy, data flows unidirectionally from the backbone to the adapter. This fundamental constraint necessitates a careful reconsideration of the adapter architecture to maintain both high accuracy and private inference efficiency. We first examine the impact of several critical factors in adapter design with empirical studies, ultimately leading to the formulation of the key design constraints. Specifically, we aim to address the following questions.

(Q1): *Are traditional adapters effective for* CryptPEFT*?*

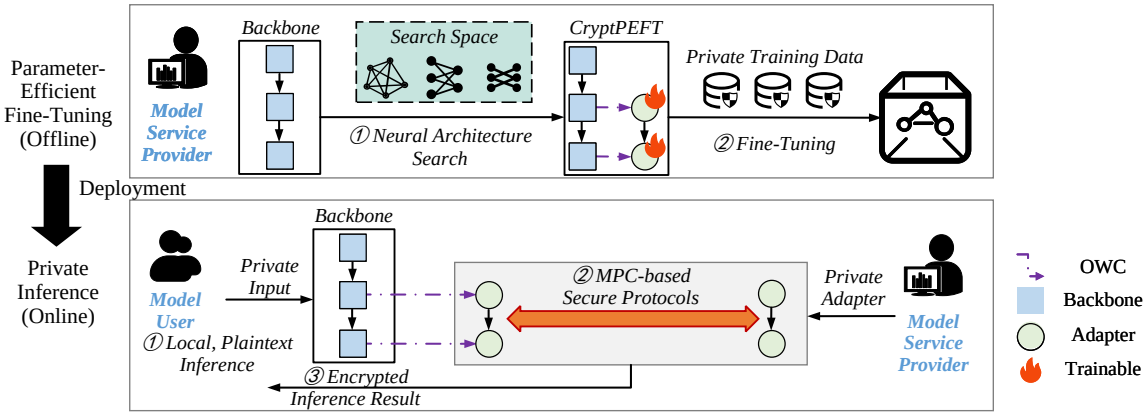


Fig. 2: Workflow of CRYPTPEFT.

TABLE I: Classification accuracy (%) of traditional adapters across different datasets under the TWC and OWC settings.

Datasets	LoRA		AdaptFormer	
	TWC	OWC	TWC	OWC
CIFAR-10	97.31	95.84 (↓1.47)	97.34	95.91 (↓1.43)
CIFAR-100	87.41	83.73 (↓3.68)	87.23	83.72 (↓3.51)
Food-101	83.95	80.13 (↓3.82)	83.91	80.02 (↓3.89)
SVHN	91.81	63.17 (↓28.64)	91.72	63.11 (↓28.61)
Flowers-102	84.37	80.76 (↓3.61)	84.42	80.89 (↓3.53)

To evaluate how the OWC constraint affects model performance, we conducted a series of experiments using traditional PEFT methods such as LoRA and AdaptFormer, both originally designed under the Two-Way Communication (TWC) paradigm. In these experiments, we held all training hyperparameters and downstream tasks (i.e., dataset for specific tasks) unchanged, modifying only the communication policy—from TWC to OWC. As shown in Table I, this shift leads to considerable fluctuations in model utility. In most downstream tasks, we observe a clear performance drop. Specifically, on the SVHN dataset—a street-view digit recognition task with only 10 labels and a data distribution that deviates significantly from standard object classification—the model's utility decreases by an average of 28.63%.

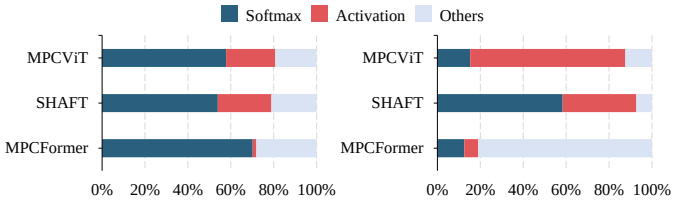
This degradation highlights a core limitation of conventional adapter architectures under the OWC constraint: their *inability to effectively capture inter-token dependencies*. In TWC-based PEFT methods, task-specific features extracted by the adapter are re-integrated into the backbone, allowing the backbone's attention mechanisms to recompute and propagate information across tokens. However, OWC restricts such bidirectional interaction, preventing the features extracted by adapters from propagating across the tokens—including the `[CLS]` token, which is crucial for downstream prediction. As a result, the model loses a key pathway for token-level information exchange, which is especially detrimental for tasks with complex or non-standard input distributions. To address this challenge, our adapter design must incorporate an internal attention block within the adapter that explicitly models inter-token relationships. This allows the adapter to refine the representation of the `[CLS]` token and recover utility.

(Q2): *Are existing attention mechanisms efficient for* CRYPTPEFT*?*

In standard Transformer architectures, attention and MLP layers play a central role in modeling long-range dependencies within input sequences (Sec. II-A). However, in private inference settings, the Softmax operation in attention and the GELU activation in MLP introduce significant computational and communication overhead. To address this, recent works have proposed MPC-friendly alternatives. MPCFormer [34] approximates both Softmax and GELU using low-degree polynomials. MPCViT [52] adopts a hybrid attention mechanism—combining ReLU Softmax Attention [40] and Scaling Attention [50]—and replaces GELU with the more MPC-efficient ReLU. SHAFT [29] further improves communication efficiency by introducing a constant-round Softmax protocol and an MPC-tailored GELU approximation. Since MPC-based private inference is highly sensitive to communication—both in rounds and bandwidth—we evaluate the communication overhead of these methods within an adapter block using CrypTen [30]. As shown in Fig. 3, the Softmax and activation functions in MPCViT and SHAFT account for over 80% of total communication overhead with the private inference of an adapter block. While MPCFormer reduces bandwidth usage, it still incurs a high number of communication rounds. These results highlight that directly applying existing attention and activation designs in adapter modules can severely impact private inference efficiency, underscoring the need for more communication-efficient alternatives.

(Q3): *How does the placement of adapters affect model utility?*

Conventional PEFT methods typically insert adapters into every layer of the backbone, resulting in a number of adapters that scales with model depth. While this design improves model expressiveness, it substantially compromises the ef-



(a) Communication Round (b) Communication Cost

Fig. 3: Communication overhead breakdown: Softmax and activations are dominant contributors in existing methods.

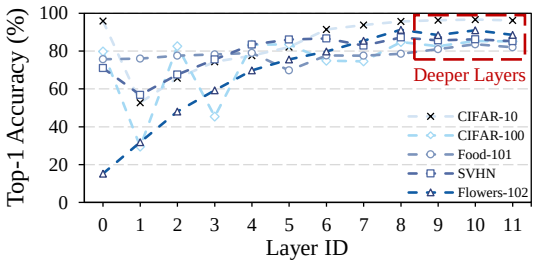


Fig. 4: Impact of adapter placement on model utility.

ficiency of private inference. In this work, we revisit the adapter placement strategy within the CryptPEFT framework, aiming to preserve utility while minimizing the number of adapters. A central question we explore is whether positioning adapters in shallower or deeper layers significantly affects model performance. This sparse adapter setting not only improves computational efficiency but also reduces the search space for neural architecture search (NAS).

To investigate this, we conduct a series of experiments in which a single adapter—comprising a low-rank decomposition module, an attention mechanism, and a MLP module—is integrated into the backbone architecture under the constraints of OWC. The adapter is inserted at varying depths across different layers of the network to evaluate its influence on model utility while adhering to OWC. As shown in Fig. 4, the results indicate that, across many downstream tasks, placing the adapter in deeper layers consistently yields superior performance—highlighted by the deep red dashed boxes in the figure. These findings suggest that, under tight resource or efficiency constraints, prioritizing adapter placement in deeper layers is a more effective strategy for preserving model utility.

Summary of design constraints. In summary, to fully harness the utility benefits of PEFT within CRYPTPEFT's workflow, CRYPTPEFT enforces 4 key constraints and incorporates a neural architecture search (NAS) mechanism.

• *Constraint 1: Ensuring One-Way Communication (OWC).* This constraint enforces a unidirectional flow of encrypted data, preventing any back-and-forth exchange that might require decryption mid-inference. By adhering to OWC, CRYPTPEFT effectively reduces potential leakage points.

• *Constraint 2: Integrating attention mechanisms with the adapters.* Adapters augmented with attention mechanisms demonstrate a strong capacity to model inter-token dependencies and capture global contextual representations, thereby facilitating effective updates to the `[CLS]` token. When adhering to OWC, models incorporating such attention-based adapters are still capable of preserving a high level of utility.

• *Constraint 3: Using MPC-friendly attention and activation functions.* The fundamental operations supported by MPC are addition and multiplication, while other operations require the use of approximation algorithms or protocols, which inevitably increase the communication rounds and communication cost. Therefore, adapters should aim to minimize the use of complex approximation algorithms and protocols.

• *Constraint 4: Focusing on deeper layers.* Rather than inserting adapters throughout the entire backbone, CRYPTPEFT targets deeper layers, which typically carry high-level features essential for final predictions. This selective adjustment yields a favorable trade-off between model accuracy and efficiency.

• *Neural Architecture Search (NAS).* To identify the optimal configurations of adapter structures and placements, CRYPTPEFT employs a NAS approach. By exploring different architecture candidates under the above constraints, it systematically discovers a model architecture that maximizes model accuracy while minimizing private inference latency.

B. CRYPTPEFT *Adapter Architecture*

Fig. 5 shows the core design of our adapter. The fundamental goal is to *update the `[CLS]` token exclusively within the adapter*, ensuring that no information is propagated back into the backbone. Since the backbone's `[CLS]` token cannot be updated by an external module under OWC, our adapter incorporates an attention block to extract and refine features specifically for the `[CLS]` token.

To reduce the cost of private SoftMax in existing attention mechanisms, we draw inspiration from *non-local neural networks* [50] and propose a novel attention mechanism: *learnable linear attention (`LinAtten`)*. Non-local neural networks introduce a general non-local operation to capture long-range dependencies, defined as:

$$\text{Non-local}(x) = \frac{1}{C(x)} \sum_i f(x_i, x) g(x_i).$$

Here, $f(\cdot)$ is a similarity function, $g(\cdot)$ is a linear transformation, and $\frac{1}{C(\cdot)}$ is a normalization factor. In `LinAtten`, we adopt the standard dot-product operations from classical attention for both $f(\cdot)$ and $g(\cdot)$, i.e., matrix multiplication. Moreover, to enhance private inference efficiency, we replace the normalization term $\frac{1}{C(\cdot)}$ with a learnable linear transformation. As a result, `LinAtten` is formally defined as:

$$\texttt{LinAtten}(Q, K, V) = L(QK^T)V,$$

where $L(\cdot)$ denotes a learnable linear transformation, parameterized as $L(\mathbf{x}) = W_L\mathbf{x}$. `LinAtten` is a highly efficient alternative that relies solely on linear operations, i.e., multiplication and addition, making it particularly well-suited for private inference while maintaining effective attention modeling. Our

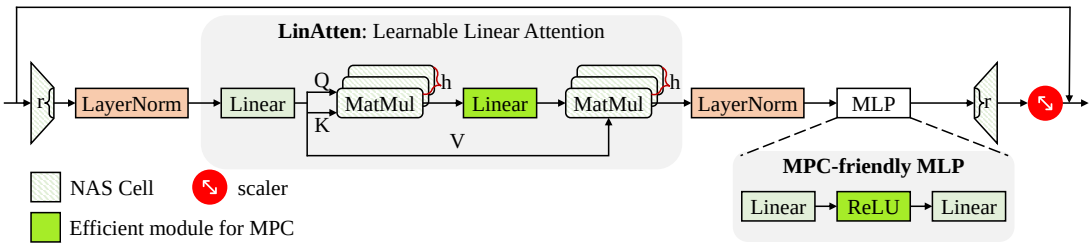


Fig. 5: The proposed adapter structure in CRYPTPEFT.

evaluation shows that `LinAtten` achieves a good utility-efficiency tradeoff (Table III and Table IV in Sec. VI).

Although `LinAtten` eliminates expensive operations such as SoftMax, directly applying it at the full feature dimensionality can remain computationally burdensome in private inference settings due to the scale of the involved matrix operations. To alleviate this issue, we adopt a low-rank adaptation strategy [26] to strike a balance between model expressiveness and computational efficiency. By constraining the learnable parameters to low-rank matrices, CRYPTPEFT significantly reduces both computational and communication overhead in secure MPC environments. Specifically, adapter weight matrices are decomposed into low-rank components parameterized by a tunable rank r, enabling fine-grained control over the model's parameter count and the private inference efficiency.

To further optimize the efficiency of the MLP module under MPC, we replace the GELU activation with ReLU, a more computation-friendly alternative in secure MPC environments, thereby yielding an MPC-friendly MLP design.

We also introduce a *scaler* factor—a tunable multiplicative constant—to modulate the contribution of the adapter output. While this factor is often fixed to small values (e.g., 0.1) in plaintext settings, our OWC-constrained architecture may benefit from alternative configurations to better navigate the accuracy-efficiency tradeoff. The effects of varying this *scaler* factor are empirically explored in Sec. IV-D.

Finally, the number of attention heads h in `LinAtten` affects both the model's representational capacity and the cost of core operations such as matrix multiplication. We allow h to be adjusted based on the requirements of downstream tasks, retaining only the minimal number of heads necessary to sustain competitive accuracy while minimizing overhead.

C. Organizational Strategy of the Adapter

As shown in Fig. 6, the backbone remains frozen, and we selectively insert adapters and a classifier in the deeper layers. Let s denote the number of final layers chosen for parameter-efficient fine-tuning. By restricting adapter placement to the last s Transformer layers, we can capture high-level semantic features with fewer trainable parameters—an essential consideration for preserving low-overhead private inference.

D. Automated Search for Adapters

In a typical NAS workflow, four components are essential: the search space, the performance-cost estimator, the sampling

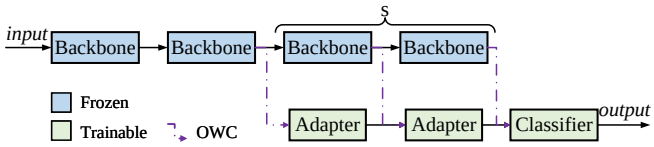


Fig. 6: Adapter placement: trainable adapters and classifier are selectively inserted into deeper layers of a frozen backbone.

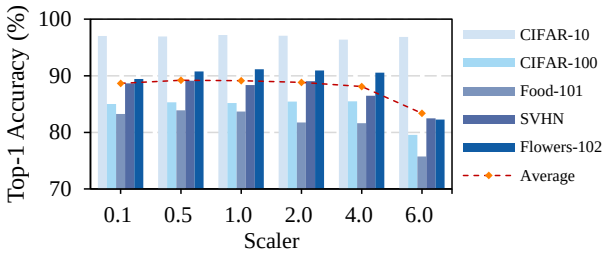


Fig. 7: Impact of *scaler* on model utility.

controller, and the search strategy. The search strategy iteratively selects candidate architectures from the search space via the sampling controller, evaluates them using the estimator, and leverages the results as reward to optimize the controller. In our work, we tailor all these components to address the specific requirements of PEFT-based private inference.

Our search space is explicitly designed around the adapter structure described in Sec. IV-B. In this paper, we focus on micro-searching [57], i.e., searching for NAS cells that can be repeatedly stacked to construct the network, which has proven effective and significantly reduces the search space. Specifically, each adapter contains two NAS cells representing distinct design choices: (1) the `LinAtten` module with a configurable number of attention heads, and (2) the low-rank compression module characterized by its rank. In addition to the internal structure of each adapter, we also treat the total number of adapters as a separate search dimension. In practice, we fix the *scaler* parameter based on preliminary tuning to simplify the overall search space. To identify an appropriate fixed value, we conducted a series of controlled experiments in which all other parameters were held constant while varying only *scaler*. We evaluated the model utility across multiple downstream tasks, and as shown in Fig. 7, a value of 0.5 consistently yielded the best average performance. As such, we fix *scaler* at 0.5 for all subsequent evaluations. Accordingly, the

full search space is defined by three key hyperparameters: h (the number of heads), r (the rank for low-rank compression), and s (the number of adapters), which jointly affect both model utility and the cost of private inference.

Unlike conventional NAS in plaintext setting, which typically assumes computational cost scales proportionally with model size, private inference introduces fundamentally different cost dynamics. Here, the cost of each candidate architecture depends not only on parameter size, but also on communication cost, the number of communication rounds, and local computation time. These factors are influenced by system characteristics such as bandwidth, latency, and the ability to batch operations into a single communication round. To enable cost-aware NAS, we profiled all key operations, including linear layers, matrix multiplications, normalization, and activation functions, under the target network environment, and derived cost estimation models based on the profiling results. Specifically, the communication cost is modeled as $(0.001153\,h + 0.000187\,r + 0.000578)\,s + 0.005692$, while the number of communication rounds is calculated as $26\,s + 3$.[2] Under a fixed network environment, both communication cost and the number of communication rounds contribute linearly to the overall communication time, which can therefore be modeled as $(c_1 h + c_2 r + c_3)s + c_4$. For example, in a typical WAN setting (400 Mbps bandwidth, 4 ms latency), the communication time is:

$$\textit{comm\_time} = (0.02117\,h + 0.00344\,r + 0.35828)\,s + 0.15541,$$

with a coefficient of determination $R^2 = 0.9975$. Similarly, the computation time is modeled as (with $R^2 = 0.9873$):

$$\textit{comp\_time} = (0.01711\,h + 0.00121\,r + 0.12311)\,s + 0.16581.$$

Consequently, the total private inference latency can be modeled as the sum of communication time and computation time.

Guided by the cost models, our NAS objective is to identify architectures that achieve a target utility while minimizing the latency of private inference. Since latency is primarily influenced by the number of adapters (s), our customized NAS strategy (outlined in Algorithm 1) prioritizes minimizing s, while tuning h and r to improve utility within the latency constraint. The value of s is increased only when necessary, i.e., when the desired utility cannot be met by adjusting h and r, or when increasing h or r incurs a higher cost than adding an additional adapter. Specifically, we first use a sampling controller to sample values for h and r (line 6), and estimate the latency using our latency model (line 7). If the sampled adapter meets the latency constraint, we evaluate the utility (line 11) and use both utility and latency to compute the reward (line 12), which guides the controller toward configurations with higher utility and lower latency (line 20). If repeated samples fail to reach the target utility, the current search round is terminated (line 21), and the number of adapters is incremented before restarting the search (line 23). The process returns either the adapter configuration that meets the target utility (line 18), or the one closest to it (line 25). Overall, these adaptations make our NAS particularly well-suited for private inference, where both latency and communication cost are critical constraints.

[2]Each adapter involves 26 communication rounds: 9 for linear operations, 2 for matrix multiplication, 6 for normalization, and 9 for activation. The linear layer contributes 1 round, and the normalization layer contributes 2 rounds.

Algorithm 1: NAS-based search algorithm.

Input: Targets $\mathcal{U}_{\text{target}}, \mathcal{L}_{\text{target}}, \mathcal{T}_{\text{target}}$; Latency model $\text{Latency}(*,*,*)$; Search spaces $\mathcal{H}, \mathcal{R}, \mathcal{S}$
Output: Best config (h^*, r^*, s^*); Best utility U^*

1 $(s, \Delta) \leftarrow (1, 1)$;
2 $(h^*, r^*, s^*, \mathcal{U}^*) \leftarrow (\perp, \perp, \perp, -\infty)$;
3 **while** $s \leq max(\mathcal{S})$ **do**
4 $\tau \leftarrow 0$;
5 **while** *True* **do**
6 $(h, r) \leftarrow \mathcal{C}(\theta)$;
7 **if** *Latency*$(h, r, s) >$ *Latency*$(h_{init}, r_{init}, s{+}\Delta)$ *or* *Latency*$(h, r, s) > \mathcal{L}_{target}$ **then**
8 $(Reward, \tau) \leftarrow (1/\text{Latency}(h, r, s), \tau + 1)$;
9 **end**
10 **else**
11 $\mathcal{U} \leftarrow \text{Eval}(h, r, s)$;
12 $Reward \leftarrow \mathcal{U} + 1/\text{Latency}(h, r, s)$;
13 $\tau \leftarrow \tau + 1$;
14 **if** $\mathcal{U} > \mathcal{U}^*$ **then**
15 $(h^*, r^*, s^*, \mathcal{U}^*) \leftarrow (h, r, s, \mathcal{U})$;
16 $\tau \leftarrow 0$;
17 **end**
18 **if** $\mathcal{U}^* \geq \mathcal{U}_{target}$ **then return** (h^*, r^*, s^*, U^*);
19 **end**
20 $\theta \leftarrow \mathcal{OPT}(\mathcal{C}, \theta, Reward)$;
21 **if** $\tau \geq \mathcal{T}_{target}$ **then break**;
22 **end**
23 $s \leftarrow s + 1$;
24 **end**
25 **return** (h^*, r^*, s^*, U^*);

V. Security Analysis

The security of CryptPEFT is established under the Universal Composability (UC) framework [7], which guarantees that the protocol remains secure even when composed with arbitrary other protocol instances. Specifically, the UC framework relies on a simulation-based definition, where a protocol is considered secure if for every real-world adversary A, there exists a polynomial-time simulator S such that no environment $\mathcal{Z}$ can distinguish whether it is interacting with the real protocol execution or with the ideal functionality emulated by S. In CryptPEFT, all computation is performed using CrypTen [1], which provides a suite of cryptographic primitives including arithmetic and binary secret sharing, as well as secure conversions between them (Sec. II-C). These primitives have been rigorously proven secure under standard simulation-based techniques, and their security holds under the UC

composition theorem [31, Appendix B]. Since all higher-level operations in CRYPTPEFT are constructed as compositions of these UC-secure primitives, the overall protocol inherits their composability guarantees. Consequently, CRYPTPEFT is UC-secure against semi-honest adversaries, and its security holds even in complex, concurrent environments where multiple protocol instances may be executed simultaneously.

VI. EVALUATIONS

A. Experiment Methodology

Hardware and software configurations. Our experimental platform consists of an Intel Xeon Silver 4310 CPU, 64 GB of RAM, and two NVIDIA GeForce RTX 4090 GPUs. The GPU was employed for automated architecture search and fine-tuning. Since most existing private inference frameworks are optimized for CPU-based implementation, our primary evaluations were conducted on the CPU. We used PyTorch version 2.3.1 for plaintext computations and CRYPTEN [1] for private inference. Specifically, plaintext computations for the backbone and private inference for the adapter were executed on the CPU with CRYPTEN's default configurations. Additionally, we examined the benefits of GPU acceleration in the context of batched task processing, utilizing the GPU to evaluate its impact on private inference efficiency.

To emulate different network conditions, we utilized the Traffic Control (TC) tool on Linux. Following the setup in BumbleBee [36] and SHAFT [29], we reproduced inference latency scenarios under two representative network environments: a wide-area network (WAN) with 400 Mbps bandwidth and 4 ms latency, and a local-area network (LAN) with 1 Gbps bandwidth and 0.5 ms latency. We evaluated the impact of varying bandwidth on private inference efficiency in Sec. VI-C.

Evaluation metrics. To evaluate the effectiveness of CRYPTPEFT, we focus on two key aspects: model utility, measured by classification accuracy, and private inference efficiency, evaluated in terms of communication time and total time during private inference, averaged over 10 runs.

Baselines. We selected two baseline approaches:

• *PEFT*—the traditional PEFT architecture, where we applied two typical methods—LoRA [26] and AdaptFormer [9]—to the backbone by implementing their proposed adapters. When evaluating inference efficiency, the encryption-then-decryption approach may lead to privacy leaks (Fig. 1b). To align with the privacy guarantees of CRYPTPEFT, we followed the inference process depicted in Fig. 1c for the PEFT baselines.

• *Simple Fine-tuning (SFT)*. A straightforward approach to achieving OWC involves fine-tuning only the final few layers of the backbone while keeping the majority of the layers frozen [46]. In private inference, this strategy enables plaintext execution for the frozen layers, while the fine-tuned final layers are executed with secure computation protocols, thereby reducing computational and communication overhead.

Datasets. We used ViT-B-16 as the backbone model for evaluation and evaluated the effectiveness of CRYPTPEFT on five representative datasets: CIFAR-10 [28], CIFAR-100 [3], Food-101 [6], SVHN [22] and Flowers-102 [41]. Prior studies, such as MPCViT [52] and AdaptFormer [9], conducted evaluations on subsets of these datasets. TinyImageNet was intentionally excluded due to its strong visual and distributional similarity to ImageNet, the dataset used to pretrain our ViT-B-16 backbone. This overlap can lead to inflated performance, reducing its effectiveness for evaluating domain generalization. Instead, we included Flowers-102 to introduce greater style diversity. Its relatively small size and large number of classes also present a challenging low-resource setting, enabling a more comprehensive evaluation of model adaptability under realistic, distribution-shifted conditions relevant to private inference.

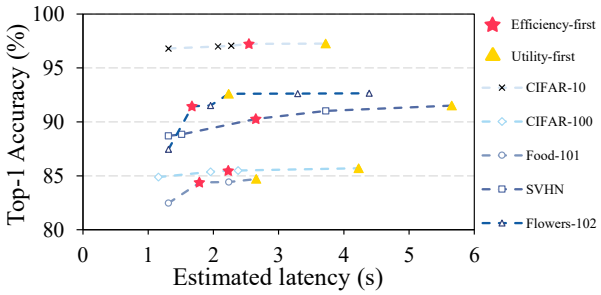


Fig. 8: The relationship between model utility and estimated private inference latency targeting the WAN environment.

B. Utility

NAS. CRYPTPEFT performs NAS by leveraging a cost model tailored to the target network environment to estimate the end-to-end private inference latency under each candidate adapter. Given a target utility, the search strategy selects the adapter with the lowest estimated latency that meets the utility constraint. We first use the WAN setting as a case study to examine the relationship between target utility and resulting private inference latency (Fig. 8). The results show that, across various downstream tasks, higher utility generally leads to increased latency. However, beyond a certain point, further utility gains are marginal despite significantly increased latency. Based on this observation, we define two trade-off strategies: *efficiency-first*, which sacrifices some utility for lower latency, and *utility-first*, which accepts higher latency to maximize utility.

The adapter structures (in the format of $\{h, r, s\}$) discovered under both WAN and LAN settings are summarized in Table II. Our results indicate that the optimal adapter design varies across downstream tasks and depends on the performance objective. For example, under the utility-first setting targeting the WAN environment, the Food-101 dataset requires 12 heads in the `LinAtten` module, while only 4 are needed under the efficiency-first setting. In the following evaluations, we primarily adopt the efficiency-first setting, as it offers strong utility with significantly improved efficiency. Under this setting, CRYPTPEFT completed the search in 4.15, 1.77, 3.87,

TABLE II: Optimized adapter structures obtained by NAS.

Datasets	Utility-first		Efficiency-first	
	LAN	WAN	LAN	WAN
CIFAR-10	{1, 180, 2}	{10, 180, 2}	{4, 120, 1}	{2, 120, 2}
CIFAR-100	{1, 300, 2}	{2, 300, 2}	{1, 240, 1}	{1, 300, 1}
Food-101	{10, 300, 1}	{12, 300, 1}	{6, 180, 1}	{4, 180, 1}
SVHN	{12, 120, 3}	{12, 180, 3}	{12, 60, 1}	{12, 300, 1}
Flowers-102	{1, 300, 1}	{1, 300, 1}	{1, 120, 1}	{1, 180, 1}

4.73, and 0.62 hours on CIFAR-10, CIFAR-100, Food-101, SVHN, and Flowers-102, respectively.

Utility comparison. We began by evaluating the model utility under various configurations of CRYPTPEFT. In the baseline approach, the neural architecture search component is absent, resulting in the use of the same adapter structure for all downstream tasks. In contrast, our approach customizes CRYPTPEFT to align with the specific characteristics of each downstream task. As shown in Table III, the utility-first configuration of CRYPTPEFT outperforms the traditional PEFT baseline by 1.45%. This improvement is particularly significant considering that the PEFT baseline requires the majority of computations on both the backbone and adapter to be performed with secure protocols (Fig. 1c), which introduces substantial communication and computational overhead (Table IV). Furthermore, even the efficiency-first configuration of CRYPTPEFT, which sacrifices some utility for inference efficiency, outperforms the baseline that simply fine-tuning the last layer by an average of 0.80%.

We also report the average number of parameters involved in encrypted computation across the downstream tasks. CRYPTPEFT achieves a significant reduction, lowering the parameter count by 88.81% in the efficiency-first setting for the WAN environment, compared to the last-layer fine-tuning baseline (0.80 million vs. 7.15 million). This reduction translates to lower computational and communicational overhead in private inference, which is further analyzed in the following.

C. Private inference efficiency

We implemented an end-to-end private inference system for CRYPTPEFT using the privacy-preserving machine learning framework CRYPTEN [1]. To ensure a fair comparison, we adopted the efficiency-first configuration and applied the same approximation techniques—specifically, the state-of-the-art Softmax and GELU approximations introduced by SHAFT—to all baseline methods, as described in Sec. IV-A. We evaluated the inference efficiency of CRYPTPEFT and the baselines under both simulated LAN and WAN settings.

As shown in Table IV, CRYPTPEFT significantly reduces private inference latency in both LAN and WAN environments. Taking CIFAR-100 as an example, CRYPTPEFT achieves a $20.85\times$ speedup over the last-layer fine-tuning baseline and a $238.76\times$ speedup over AdaptFormer in the LAN setting. In the WAN setting, it yields a $20.08\times$ and $243.84\times$ speedup over the same baselines, respectively. Notably, CRYPTPEFT demonstrates an inference time of 2.26 seconds with an accuracy of 85.47% on CIFAR-100 under the WAN environment.

Performance breakdown. Table V provides a detailed performance breakdown of CRYPTPEFT on the CIFAR-100 dataset, including communication cost, communication round, communication time, and total inference time. For comparison, we used the most efficient baseline among the SFT configurations—fine-tuning only the last layer. Compared to this baseline, CRYPTPEFT reduces the number of communication rounds by 62.37% and the communication cost by an average of 96.45% in both LAN and WAN environments. Since communication is the dominant contributor to latency in MPC-based inference, these reductions result in significant performance gains: CRYPTPEFT achieves speedups of $20.85\times$ in LAN settings and $20.08\times$ in WAN settings.

Reliability of the cost model. The cost model plays a critical role in our NAS framework. To evaluate its accuracy, we compare the estimated end-to-end latency with actual measurements under both LAN and WAN settings (Table IV). The results show that the estimation error remains within a $\pm 4.3\%$ margin, indicating that the model provides reliable latency predictions to guide the search process.

Comparison with MPCViT [52]. We also compared CRYPTPEFT with MPCViT, the current state-of-the-art that leverages NAS and approximation techniques for efficient private inference for ViT. MPCViT provides pre-trained models for various downstream tasks, including CIFAR-10 and CIFAR-100, which align with our evaluation benchmarks. As shown in Table VI, CRYPTPEFT outperforms MPCViT on both datasets, achieving higher utility and a $9.19\times$ to $13.14\times$ speedup in private inference. These gains result from CRYPTPEFT's PEFT-specific design, which leverages a public backbone and adopts the OWC policy to minimize encrypted computation, thereby enhancing both model utility and efficiency.

D. Ablation Study and Sensitivity Analysis

Sensitivity to network conditions. In addition to the simulated LAN and WAN environments, we further evaluated the private inference efficiency of CRYPTPEFT under a range of network bandwidths using the CIFAR-100 dataset. The network latency was fixed at 4 ms, and the SFT configuration was adopted as the baseline. As shown in Table VII, CRYPTPEFT consistently outperforms the baseline across all bandwidth settings. Notably, CRYPTPEFT reduces the communication volume to only 3.87% of that required by SFT (Table V). This efficiency gain becomes increasingly significant under limited bandwidth conditions. Specifically, CRYPTPEFT achieves speedups of $9.56\times$ under optimal network conditions and up to $21.72\times$ under the most constrained setting. Given that MPC-based private inference is inherently communication-intensive and highly sensitive to network conditions, CRYPTPEFT exhibits strong robustness to degraded network environments, thereby enhancing its viability for real-world applications.

Effectiveness of `LinAtten`. We evaluated `LinAtten` by comparing it with several attention mechanisms specifically designed for MPC-friendly computation. Using the CIFAR-100 dataset, we integrated three representative approaches—

TABLE III: Comparison of model utility (i.e., accuracy). CRYPTPEFT achieves classification accuracy comparable to the baseline methods, while significantly reducing the number of parameters involved in encrypted computation.

Methods		Avg. private params. (M)	CIFAR-10	CIFAR-100	Food-101	SVHN	Flowers-102	Avg. utility
PEFT (baseline)	LoRA	86.01	97.31%	**87.41%**	83.95%	91.81%	84.37%	88.97%
	AdaptFormer	87.05	97.34%	87.23%	83.91%	91.72%	84.42%	88.92%
SFT (baseline)	Last Layer	7.15	97.28%	86.24%	84.99%	87.35%	88.81%	88.93%
	Last 2 layers	14.23	**97.51%**	86.57%	**85.56%**	90.10%	90.63%	90.07%
CRYPTPEFT	Utility-first (LAN)	1.28	97.29%	85.63%	84.84%	**91.82%**	**92.60%**	**90.43%**
	Efficiency-first (LAN)	**0.46**	97.19%	85.37%	84.56%	90.03%	91.32%	89.69%
	Utility-first (WAN)	1.53	97.26%	85.70%	84.70%	91.51%	**92.60%**	90.35%
	Efficiency-first (WAN)	0.80	97.23%	85.47%	84.38%	90.27%	91.45%	89.76%

TABLE IV: Comparisons of private inference latency (unit: second) within typical LAN and WAN environments. We also report the latency estimated by our cost model for comparison (with wavy underline).

	Methods		CIFAR-10	CIFAR-100	Food-101	SVHN	Flowers-102
LAN	**PEFT** (baseline)	LoRA	271.34	270.39	266.42	266.59	267.82
		AdaptFormer	269.02	269.80	270.75	268.31	271.65
	SFT (baseline)	Last layer	23.30	23.56	23.44	23.17	23.72
		Last 2 layers	46.89	46.96	44.95	45.28	47.24
	CRYPTPEFT	Efficiency-first	**0.81** (0.83)	**1.13** (1.10)	**1.04** (1.06)	**0.90** (0.87)	**0.75** (0.75)
WAN	**PEFT** (baseline)	LoRA	548.27	545.91	547.08	546.43	545.56
		AdaptFormer	549.95	551.08	550.64	549.66	548.96
	SFT (baseline)	Last layer	45.32	45.38	45.25	45.56	44.26
		Last 2 layers	90.70	90.80	90.07	88.08	90.54
	CRYPTPEFT	Efficiency-first	**2.45** (2.55)	**2.26** (2.24)	**1.85** (1.79)	**2.78** (2.66)	**1.61** (1.68)

TABLE V: Performance breakdown analysis.

	Metrics	SFT (baseline)	CRYPTPEFT	Improvements
LAN	Comm. (GB)	1.55	0.05	31.00×
	Comm. round	77	29	2.66×
	Comm. time (s)	14.40	0.55	26.18×
	Total time (s)	23.56	1.13	20.85×
WAN	Comm. (GB)	1.55	0.06	25.83×
	Comm. round	77	29	2.66×
	Comm. time (s)	34.42	1.51	22.79×
	Total time (s)	45.38	2.26	20.08×

TABLE VI: Comparison of utility and private inference latency (unit: second) between MPCViT and CRYPTPEFT.

Settings		Metrics	MPCViT	CRYPTPEFT	Improvements
CIFAR-10	LAN	Utility	94.27%	97.19%	↑2.92%
		Total time	10.64	0.81	13.14×
	WAN	Utility	94.27%	97.23%	↑2.96%
		Total time	24.10	2.45	9.84×
CIFAR-100	LAN	Utility	77.46%	85.37%	↑7.91%
		Total time	10.38	1.13	9.19×
	WAN	Utility	77.46%	85.47%	↑8.01%
		Total time	22.97	2.26	10.16×

MPCFormer, SHAFT, and MPCViT—into the CRYPTPEFT framework and conducted end-to-end private inference under various network conditions. As shown in Fig. 9, we

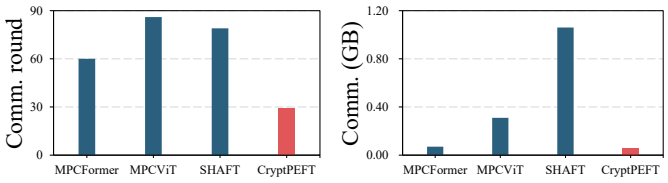
TABLE VII: Total private inference latency (unit: second) under varying network bandwidths.

Bandwidths	SFT (baseline)	CRYPTPEFT	Improvements
100 Mbps	145.77	6.71	21.72×
500 Mbps	38.65	2.11	18.32×
1 Gbps	23.71	1.71	13.87×
5 Gbps	12.95	1.35	9.56×

report both communication rounds and communication cost for each method. Compared to existing attention mechanisms, CRYPTPEFT achieves significant reductions in both metrics. This performance gain is attributed to the design of `LinAtten`, which provides a more communication-efficient attention mechanism. Specifically, `LinAtten` relies predominantly on MPC-friendly operations (additions and multiplications), thereby reducing both communication and computation overheads (see Fig. 5).

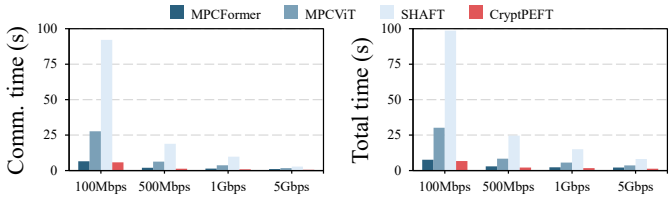
Under different network conditions, Fig. 10b presents the total inference latency, where CRYPTPEFT consistently outperforms all baselines. In particular, compared to MPCFormer, MPCViT, and SHAFT, CRYPTPEFT achieves speedups of 1.13× to 1.56×, 2.73× to 4.49×, and 6.02× to 14.70×, respectively, highlighting the effectiveness of `LinAtten` in improving private inference efficiency.

Sensitivity to batch sizes. In real-world inference scenarios, batching and GPU parallelism are commonly employed to



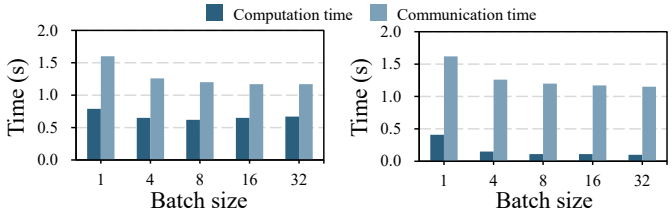
(a) Communication round (b) Communication cost

Fig. 9: CRYPTPEFT outperforms existing attention mechanisms in terms of both communication round and cost.



(a) Communication time (b) Total time

Fig. 10: Inference efficiency across attention mechanisms.



(a) CPU (b) GPU

Fig. 11: Private inference efficiency under different batch sizes.

improve inference efficiency. To evaluate the impact of batch size on private inference efficiency, we evaluated CRYPTPEFT across a range of batch sizes using both CPU and GPU backends. All experiments were conducted on the CIFAR-100 dataset under a WAN network setting, and we report the amortized overhead per sample. As illustrated in Fig. 11, increasing the batch size leads to substantial improvements in private inference efficiency. For example, when using a CPU, scaling the batch size from 1 to 32 results in a 1.30× speedup in computation time and a 1.37× speedup in communication time per sample. However, the marginal gains decrease with larger batch sizes due to hardware and network bandwidth limitations. GPU-based computation yields even greater benefits: increasing the batch size from 1 to 32 achieves an 4.10× speedup in computation time.

Effectiveness of NAS. Our NAS approach incorporates a cost model tailored for private inference, prioritizing architectures with higher efficiency in this setting. To evaluate its effectiveness, we adopted ENAS [44], a representative reinforcement learning based NAS method, as the baseline. We conducted experiments under the WAN network environment across 5 downstream tasks. The end-to-end private inference latency was evaluated using the adapters discovered by both methods. As shown in Fig. 12, our method achieved a 4.07× average

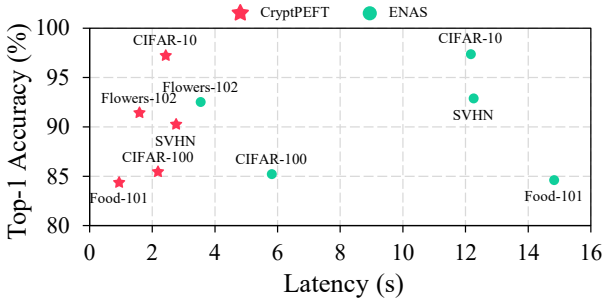


Fig. 12: Comparison of accuracy and private inference latency for models discovered by CRYPTPEFT and ENAS.

speedup in private inference latency, with only a marginal drop in classification accuracy (89.73% vs. 90.51%). In addition, it identified target models 1.85× faster on average compared to ENAS (3.03 hours vs. 5.60 hours on average).

VII. LIMITATIONS AND DISCUSSIONS

Backbone model overhead. Under the OWC policy adopted by CRYPTPEFT, a substantial portion of the computation on the backbone model is offloaded to the **MU** and executed in plaintext. This design assumes the **MU** has moderate computational capability—a reasonable assumption given that, in traditional MPC-based approaches, the **MU** already engages in costly encrypted computations. Since encrypted computation typically dominates the **MU**'s workload, reducing its reliance through OWC offers a significant advantage.

In CRYPTPEFT, both **MS** and **MU** must download the backbone model (e.g., from public repositories like Hugging Face), incurring a one-time offline cost that does not impact private inference efficiency. Both parties also store a local copy of the backbone, introducing certain storage overhead. However, similar to existing PEFT methods, CRYPTPEFT benefits from model reuse across multiple downstream tasks, significantly reducing overall storage demands.

Scalability to LLMs. A promising avenue is to extend CRYPTPEFT's applicability to a wider range of architectures and application domains. While our current focus is on vision-based tasks, adapting the proposed approach to large language models (LLMs) or multi-modal networks could yield novel insights and present new challenges. Specifically, addressing the unique functional blocks and data flow patterns inherent in language transformers may necessitate specialized adapter designs, enhanced OWC strategies, and innovative approximation methods tailored to natural language processing.

Trusted Execution Environment (TEE). While our primary focus is on the MPC setting, our proposed method is compatible with TEE-based deployments with minimal modification. Specifically, the OWC constraint and the resulting architecture search procedure are agnostic to the underlying secure execution mechanism and can be readily applied in TEE environments. TEEs offer advantages in terms of lower latency and simpler deployment, especially in single-device or edge

scenarios. However, they rely on trusting the hardware vendor and are potentially susceptible to side-channel attacks, whereas MPC provides stronger security guarantees under standard cryptographic assumptions.

VIII. Related Works

Privacy-preserving computation friendly neural network architectures. Traditional neural networks were not originally designed to accommodate private inference. To bridge this gap, recent research has explored the optimization of neural architectures for privacy-preserving settings. Early works primarily focused on adapting CNNs for private inference under HE and MPC frameworks [32], [14], [15], [33]. Recently, Zeng et al. introduced MPCViT [52], a ViT architecture tailored for MPC settings. By reducing the scale of the ViT and implementing heterogeneous attention mechanisms, MPCViT achieves efficient and accurate inference under MPC frameworks.

Efficient approximation of neural networks. To accelerate private inference, researchers have explored polynomial approximations of nonlinear functions [51], [34], [4], [21], [10], [54]. While low-precision approximations can enhance computational efficiency, they often lead to utility degradation. MPCFormer employs knowledge distillation to improve model utility compromised by low-precision approximations [34]. AutoFHE introduces layerwise mixed-degree polynomial approximations, assigning different polynomial degrees to various layers based on their sensitivity to approximation errors [4]. Recent studies have proposed more efficient approximation methods to mitigate utility degradation caused by low-precision approximations [56], [43].

Private transformer inference systems. The widespread adoption of Transformer architectures has introduced notable challenges for private inference. In response, several studies have developed efficient private inference systems tailored for Transformers [23], [42], [36], [25], [29]. Specifically, Iron addresses the intensive computational demands of large-scale matrix multiplications and complex nonlinear functions like Softmax and GELU inherent in Transformer models [23]. BumbleBee [36] and Bolt [42] implement advanced protocols for matrix multiplication and activation functions. CipherGPT extends these advancements to generative large language models [25]. SHAFT [29] extends CrypTen [1] by introducing efficient protocols for constant-round Softmax and GELU computations. In comparison, CryptPEFT introduces a novel architecture tailored for PEFT methods, which can be seamlessly integrated into existing backbone models.

IX. Conclusions

In this paper, we present CryptPEFT, the first PEFT architecture designed specifically for private inference. CryptPEFT introduces an OWC paradigm that confines encrypted computation to the adapter, thereby eliminating the need for expensive two-way interactions with the backbone. To ensure strong model utility under the OWC constraint, we explore the design space of OWC-compliant adapters and incorporate an automated search mechanism to identify optimal configurations. Our evaluations demonstrate that CryptPEFT achieves significant improvements in private inference efficiency.

Acknowledgments

We would like to express our sincere gratitude to the anonymous reviewers and our shepherd for their insightful and valuable feedback. The work of the authors from the Institute of Information Engineering was supported in part by the Strategic Priority Research Program of the Chinese Academy of Sciences (Grant No. XDB0690100), the National Natural Science Foundation of China (Grant Nos. 92270204 and 62272452), and a research grant from Huawei Technologies.

References

[1] Crypten - a framework for privacy preserving machine learning. https://github.com/facebookresearch/CrypTen, 2025.

[2] Aymen M Al-Hejri, Riyadh M Al-Tam, Muneer Fazea, Archana Harsing Sable, Soojeong Lee, and Mugahed A Al-Antari. Etecadx: Ensemble self-attention transformer encoder for breast cancer diagnosis using full-field digital x-ray breast images. *Diagnostics*, 13(1):89, 2022.

[3] Krizhevsky Alex. Learning multiple layers of features from tiny images. *https://www. cs. toronto. edu/kriz/learning-features-2009-TR. pdf*, 2009.

[4] Wei Ao and Vishnu Naresh Boddeti. {AutoFHE}: Automated adaption of {CNNs} for efficient evaluation over {FHE}. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 2173–2190, 2024.

[5] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology-CRYPTO'91: Proceedings 11*, pages 420–432. Springer, 1992.

[6] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101–mining discriminative components with random forests. In *Computer vision–ECCV 2014: 13th European conference, zurich, Switzerland, September 6-12, 2014, proceedings, part VI 13*, pages 446–461. Springer, 2014.

[7] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[8] Octavian Catrina and Sebastiaan De Hoogh. Improved primitives for secure multiparty integer computation. In *Security and Cryptography for Networks: 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings 7*, pages 182–199. Springer, 2010.

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

[10] Tianyu Chen, Hangbo Bao, Shaohan Huang, Li Dong, Binxing Jiao, Daxin Jiang, Haoyi Zhou, Jianxin Li, and Furu Wei. The-x: Privacy-preserving transformer inference with homomorphic encryption. *arXiv preprint arXiv:2206.00216*, 2022.

[11] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR, 2020.

[12] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E. Hinton. Big self-supervised models are strong semi-supervised learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[13] Zhe Chen, Yuchen Duan, Wenhai Wang, Junjun He, Tong Lu, Jifeng Dai, and Yu Qiao. Vision transformer adapter for dense predictions. In *The Eleventh International Conference on Learning Representations*, 2022.

[14] Minsu Cho, Zahra Ghodsi, Brandon Reagen, Siddharth Garg, and Chinmay Hegde. Sphynx: A deep neural network design for private inference. *IEEE Security & Privacy*, 20(5):22–34, 2022.

[15] Minsu Cho, Ameya Joshi, Brandon Reagen, Siddharth Garg, and Chinmay Hegde. Selective network linearization for efficient private inference. In *International Conference on Machine Learning*, pages 3947–3961. PMLR, 2022.

[16] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings 2*, pages 342–362. Springer, 2005.

[17] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography Conference*, pages 285–304. Springer, 2006.

[18] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual cryptology conference*, pages 643–662. Springer, 2012.

[19] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[20] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20:55:1–55:21, 2019.

[21] Zahra Ghodsi, Akshaj Kumar Veldanda, Brandon Reagen, and Siddharth Garg. Cryptonas: Private inference on a relu budget. *Advances in Neural Information Processing Systems*, 33:16961–16971, 2020.

[22] Ian J Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*, 2013.

[23] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private inference on transformers. *Advances in neural information processing systems*, 35:15718–15731, 2022.

[24] Haoyu He, Jianfei Cai, Jing Zhang, Dacheng Tao, and Bohan Zhuang. Sensitivity-aware visual parameter-efficient fine-tuning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11825–11835, 2023.

[25] Xiaoyang Hou, Jian Liu, Jingyu Li, Yuhan Li, Wen-jie Lu, Cheng Hong, and Kui Ren. Ciphergpt: Secure two-party gpt inference. *Cryptology ePrint Archive*, 2023.

[26] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2021.

[27] Mali Jin, Daniel Preoţiuc-Pietro, A Seza Doğruöz, and Nikolaos Aletras. Automatic identification and classification of bragging in social media. *arXiv preprint arXiv:2203.05840*, 2022.

[28] Andrej Karpathy. Lessons learned from manually classifying cifar-10. *Published online at http://karpathy. github. io/2011/04/27/manually-classifying-cifar10*, 2011.

[29] Andes Y. L. Kei and Sherman S. M. Chow. SHAFT: Secure, handy, accurate, and fast transformer inference. In *NDSS*, 2025.

[30] B. Knott, S. Venkataraman, A.Y. Hannun, S. Sengupta, M. Ibrahim, and L.J.P. van der Maaten. Crypten: Secure multi-party computation meets machine learning. In *arXiv 2109.00984*, 2021.

[31] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems*, 34:4961–4973, 2021.

[32] Souvik Kundu, Shunlin Lu, Yuke Zhang, Jacqueline Liu, and Peter A Beerel. Learning to linearize deep neural networks for secure and efficient private inference. *arXiv preprint arXiv:2301.09254*, 2023.

[33] Souvik Kundu, Yuke Zhang, Dake Chen, and Peter A Beerel. Making models shallow again: Jointly learning to reduce non-linearity and depth for latency-efficient private inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4685–4689, 2023.

[34] Dacheng Li, Rulin Shao, Hongyi Wang, Han Guo, Eric P Xing, and Hao Zhang. Mpcformer: fast, performant and private transformer inference with mpc. *arXiv preprint arXiv:2211.01452*, 2022.

[35] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

[36] Wen-jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Cheng Hong, Kui Ren, Tao Wei, and WenGuang Chen. Bumblebee: Secure two-party inference framework for large transformers. *Cryptology ePrint Archive*, 2023.

[37] Ella Mahoro and Moulay A Akhloufi. Breast cancer classification on thermograms using deep cnn and transformers. *Quantitative InfraRed Thermography Journal*, 21(1):30–49, 2024.

[38] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 2505–2522. USENIX Association, 2020.

[39] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 19–38. IEEE Computer Society, 2017.

[40] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*, pages 19–38. IEEE, 2017.

[41] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

[42] Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. Bolt: Privacy-preserving, accurate and efficient inference for transformers. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 4753–4771. IEEE, 2024.

[43] Dongjin Park, Eunsang Lee, and Joon-Woo Lee. Powerformer: Efficient privacy-preserving transformer with batch rectifier-power max function and optimized homomorphic attention. *Cryptology ePrint Archive*, 2024.

[44] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018.

[45] Aditya Prakash, Kashyap Chitta, and Andreas Geiger. Multi-modal fusion transformer for end-to-end autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7077–7087, 2021.

[46] Deevashwer Rathee, Dacheng Li, Ion Stoica, Hao Zhang, and Raluca A. Popa. Mpc-minimized secure LLM inference. *CoRR*, abs/2408.03561, 2024.

[47] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 325–342. ACM, 2020.

[48] Sijun Tan, Brian Knott, Yuan Tian, and David J. Wu. Cryptgpu: Fast privacy-preserving machine learning on the GPU. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1021–1038. IEEE, 2021.

[49] Xuan-Son Vu, Duc-Trong Le, Christoffer Edlund, Lili Jiang, and Hoang D Nguyen. Privacy-preserving visual content tagging using graph transformer networks. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 2299–2307, 2020.

[50] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7794–7803, 2018.

[51] Joonsang Yu, Junki Park, Seongmin Park, Minsoo Kim, Sihwa Lee, Dong Hyun Lee, and Jungwook Choi. Nn-lut: neural approximation of non-linear operations for efficient transformer inference. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 577–582, 2022.

[52] Wenxuan Zeng, Meng Li, Wenjie Xiong, Tong Tong, Wen-jie Lu, Jin Tan, Runsheng Wang, and Ru Huang. Mpcvit: Searching for accurate and efficient mpc-friendly vision transformer with heterogeneous attention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5052–5063, 2023.

[53] Yuanhan Zhang, Kaiyang Zhou, and Ziwei Liu. Neural prompt search. *arXiv preprint arXiv:2206.04673*, 2022.

[54] Yuke Zhang, Dake Chen, Souvik Kundu, Chenghao Li, and Peter A Beerel. Sal-vit: Towards latency efficient private inference on vit using selective attention search with a learnable softmax approximation. In

Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 5116–5125, 2023.
[55] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proc. IEEE*, 109(1):43–76, 2021.
[56] Itamar Zimerman, Allon Adir, Ehud Aharoni, Matan Avitan, Moran Baruch, Nir Drucker, Jenny Lerner, Ramy Masalha, Reut Meiri, and Omri Soceanu. Power-softmax: Towards secure llm inference over encrypted data. *arXiv preprint arXiv:2410.09457*, 2024.
[57] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

APPENDIX A
ARTIFACT APPENDIX

A. Description & Requirements

CRYPTPEFT is a parameter-efficient fine-tuning (PEFT) solution specifically designed for private inference scenarios. It minimizes the cost of encrypted computation by introducing a novel one-way communication (OWC) architecture, significantly reducing both computational and communication overhead. To maintain strong model utility under these constraints, CRYPTPEFT incorporates OWC-compatible adapters and employs an automated neural architecture search (NAS) algorithm. Our artifact includes the code and scripts necessary to evaluate both the model utility and private inference efficiency of the proposed method.

1) How to access: The source code of CRYPTPEFT is publicly available at https://doi.org/10.5281/zenodo.17036866.

2) Hardware dependencies: Our artifact can be executed on a standard Linux server without requiring specialized hardware. The original experiments were conducted on a machine equipped with two NVIDIA GeForce RTX 4090 GPUs, used primarily for neural architecture search (NAS) and for assessing the benefits of GPU acceleration in batched task processing. To reduce hardware dependency and enhance accessibility, GPU-related components have been excluded from the artifact. This modification does not impact the paper's core claims: CRYPTPEFT achieves high model utility and efficient private inference through the introduction of the OWC policy and an OWC-compliant adapter design.

3) Software dependencies: We implemented CRYPTPEFT using Python 3.10.14. The dependencies for CRYPTPEFT (and the following experiments) are detailed in `requirements.txt`.

4) Benchmarks: (1) Datasets: CIFAR-10 [28], CIFAR-100 [3], Food-101 [6], SVHN [22] and Flowers-102 [41]. (2) Models: ViT-B-16, a set of adapters searched using Algorithm 1.

B. Artifact Installation & Configuration

Download the source code locally following Sec. A-A1. In the AE folder under the project root directory (default: CRYPTPEFT), you will find a `README.md` file with detailed installation and configuration instructions.

C. Major Claims

- (C1): CRYPTPEFT achieves classification accuracy comparable to the state-of-the-art methods, while significantly reducing the number of parameters involved in encrypted computation. This is proven by the experiment (E1), with results reported in Table III.
- (C2): CRYPTPEFT significantly reduces private inference latency in both LAN and WAN environments. This is proven by the experiment (E2), with results reported in Table IV and Table V.
- (C3): CRYPTPEFT significantly outperforms MPCViT [52] by providing higher utility and achieving $9.19\times$ to $13.14\times$ speedup in private inference on CIFAR-10 and CIFAR-100. This is proven by the experiment (E1), experiment (E2) and experiment (E3), with results reported in Table VI.
- (C4): The `LinAtten` proposed in CRYPTPEFT significantly outperforms the attention mechanisms used in MPCFormer [34], MPCViT [52], and SHAFT [29] in terms of private inference efficiency. This is proven by the experiment (E4), with results reported in Fig. 10.

D. Evaluation

1) Experiment (E1): [CRYPTPEFT utility] [5 human-minutes + 20 compute-minutes]: This experiment is designed to evaluate the utility of CRYPTPEFT under both Utility-first and Efficiency-first strategies.

[Preparation] Go to the project root directory (default name: CRYPTPEFT).

[Execution] Run the command below.

```
$: bash AE/eval_model_utility.sh
```

[Results] All results are saved in the directory `AE/eval_result`. Open the file named like `WAN_CRYPTPEFT_Efficiency_first_cifar100` to find results similar to the example below (note that results may vary by about 1% depending on the experimental environment and hardware).

```
========= eval result ...... =========
acc:85.47
n_param:1.12141
```

2) Experiment (E2): [CRYPTPEFT private inference latency] [5 human-minutes + 90 compute-minutes]: This experiment is designed to evaluate the private inference latency of CRYPTPEFT under LAN and WAN environments.

[Preparation] Go to the project root directory (default name: CRYPTPEFT).

[Execution]

(1): Execute the following command in the Linux terminal to simulate a LAN network environment (avoid using VSCode, as it may cause the execution to fail).

```
$: sudo tc qdisc add dev lo root \
netem rate 1gbit delay 0.5ms
```

(2): Open two terminals and run the following commands separately (you can use either VSCode or a Linux terminal).

```
$0: bash AE/eval_CRYPTPEFT_PI.sh 0 LAN
$1: bash AE/eval_CRYPTPEFT_PI.sh 1 LAN

$0: bash AE/eval_SFT_Last_PI.sh 0 LAN
$1: bash AE/eval_SFT_Last_PI.sh 1 LAN
```

(3): Execute the following command in the Linux terminal to simulate a WAN network environment (avoid using VSCode, as it may cause the execution to fail).

```
$: sudo tc qdisc del dev lo root
$: sudo tc qdisc add dev lo root \
netem rate 400mbit delay 4ms
```

(4): Open two terminals and run the following commands separately (you can use either VSCode or a Linux terminal).

```
$0: bash AE/eval_CRYPTPEFT_PI.sh 0 WAN
$1: bash AE/eval_CRYPTPEFT_PI.sh 1 WAN

$0: bash AE/eval_SFT_Last_PI.sh 0 WAN
$1: bash AE/eval_SFT_Last_PI.sh 1 WAN
```

(5): Restore the default network environment (avoid using VSCode, as it may cause the execution to fail).

```
$: sudo tc qdisc del dev lo root
```

[Results] All results are saved in the directory `AE/eval_private_inference_result`. Open the file named like {e.g., `eval_CryptPEFT_LAN_cifar100`} to find results similar to the example below (the total_time and comm_time may vary due to differences in CPU specifications; however, the improvements compared to the baseline, comm_cost and comm_round remain consistent).

```
total_time: 1.1336361408233642
comm_round: 29.0
comm_cost: 0.052045270800590515
comm_time: 0.49003771375864746
```

3) Experiment (E3): [Comparation with MPCViT] [5 human-minutes + 20 compute-minutes]: This experiment is designed to compare the model utility and private inference latency between CryptPEFT and MPCViT (the utility results of MPCViT are taken from the results reported in their paper [52]).

[Preparation]

(1): Go to the project root directory (default name: CryptPEFT).

(2): Simulate WAN and LAN network environments following similar steps as in Experiment (E2).

[Execution]

(1): Make sure you are in the simulated LAN network environment. Open two terminals and run the following commands separately (you can use either VSCode or a Linux terminal).

```
$0: bash AE/eval_MPCViT_PI.sh 0 LAN
$1: bash AE/eval_MPCViT_PI.sh 1 LAN
```

(2): Make sure you are in the simulated WAN network environment. Open two terminals and run the following commands separately (you can use either VSCode or a Linux terminal).

```
$0: bash AE/eval_MPCViT_PI.sh 0 WAN
$1: bash AE/eval_MPCViT_PI.sh 1 WAN
```

[Results] All results are saved in the directory `AE/eval_private_inference_result`. Open the file named like `eval_MPCViT_WAN_cifar100` to find results similar to the example below.

```
total_time: 22.374131655693056
comm_round: 513.0
comm_cost: 0.5615268349647522
comm_time: 18.25921990380448
```

4) Experiment (E4): [The private inference efficiency of `LinAtten`] [5 human-minutes + 50 compute-minutes]: This experiment is designed to evaluate the efficiency of private inference with various attention mechanisms.

[Preparation]

(1): Go to the project root directory (default name: CryptPEFT).

(2): Simulate different network bandwidths following similar steps as in Experiment (E2). Note that the network latency should be fixed at 4ms, and the network environment must be reset to default before changing to a different bandwidth.

[Execution] We assume you are in a network environment with 1Gbps bandwidth and 4ms latency. Open two terminals and run the following commands separately.

```
$0: bash AE/ablation_LinAtten_PI.sh 0 1G
$1: bash AE/ablation_LinAtten_PI.sh 1 1G
```

[Results] All results are saved in the directory `AE/eval_private_inference_result`. Open the file named like `ablation_LinAtten_1G_CryptPEFT_cifar100` to find results similar to the example below.

```
total_time: 1.5270877838134767
comm_round: 29.0
comm_cost: 0.06424663960933685
comm_time: 0.8196889258921146
```