Practical and Private Hybrid ML Inference with Fully Homomorphic Encryption

Anonymous authors
Paper under double-blind review

Abstract

In contemporary cloud-based services, protecting users' sensitive data and ensuring the confidentiality of the server's model are critical. Fully homomorphic encryption (FHE) enables inference directly on encrypted inputs, but its practicality is hindered by expensive bootstrapping and inefficient approximations of non-linear activations. We introduce Safhire, a hybrid inference framework that executes linear layers under encryption on the server while offloading non-linearities to the client in plaintext. This design eliminates bootstrapping, supports exact activations, and significantly reduces computation cost. To safeguard model confidentiality despite client access to intermediate outputs, Safhire applies randomized shuffling, which obfuscates intermediate values and makes it practically impossible for a client to reconstruct the model. To further reduce inference latency, Safhire incorporates advanced optimizations such as fast ciphertext packing and partial extraction. Evaluations on multiple standard models and datasets show that Safhire achieves up to 10.7× and 7.4× lower inference latency than Orion and Concrete-ML, respectively, which are two state-of-the-art FHE baselines, with manageable communication overhead and comparable accuracy. These results underline the practicality of our hybrid FHE inference scheme.

1 Introduction

Machine learning (ML) has profoundly impacted industrial domains such as healthcare (Habehh & Gohel, 2021), finance (Dixon et al., 2020), and the Internet of Things (Hussain et al., 2020). In many practical deployments, ML models are hosted on cloud servers and accessed by clients who submit private and privileged data for inference (Zhang et al., 2019). However, this paradigm, also known as ML-as-a-Service (MLaaS), raises serious privacy concerns as users must share their inference inputs, such as medical or financial data, with cloud servers who can directly use or share the data with other parties (Mireshghallah et al., 2020). As users increasingly rely on online inference services, ensuring the privacy of user data during inference is paramount (Hesamifard et al., 2018).

Fully homomorphic encryption (FHE) offers a compelling solution to this privacy concern by enabling computations directly on encrypted user data, see Fig. 2 (top) (Gentry, 2009; Marcolla et al., 2022). FHE provides strong privacy guarantees as the server learns nothing beyond ciphertext sizes and protocol metadata. An ML inference request with FHE works as follows: a user first encrypts its inference inputs and sends the encrypted data to the server (step 1–2). The server then performs the computations related to the neural network forward pass in the encrypted domain (step 3) and sends back the encrypted output (step 4), after which the user decrypts the output and obtains the inference result (step 5). This way, the server is unable to infer any information from the input data since it remains encrypted during the inference request (Gilad-Bachrach et al., 2016).

Existing FHE inference schemes face two main obstacles that make E2E encrypted inference impractical at scale (Ebel et al., 2025; Nam et al., 2022). First, encrypted vectors can only undergo so many operations before becoming too noisy to decrypt correctly. To reset ciphertext noise, FHE schemes rely on an operation called *bootstrapping*. Bootstrapping enables further computations on these vectors by reducing this noise, but comes with a high computational cost (Bossuat et al., 2021). Fig. 1 shows the duration of bootstrapping

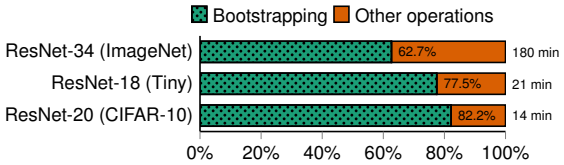

Figure 1: Latency breakdown for bootstrapping and other operations during a single sample inference request using Orion, a state-of-the-art FHE inference framework. We evaluate three models and datasets and show the E2E inference duration in minutes on the right.

and other operations in Orion, a state-of-the-art FHE inference framework (Ebel et al., 2025). We evaluate the latency required for bootstrapping and other operations for three convolutional neural network (CNN) models, ResNet-20, ResNet-18 and ResNet-34, and three datasets, CIFAR-10, Tiny and ImageNet. Across these configurations, bootstrapping takes between 62% to 85% of total inference time, highlighting the significant cost of this operation. Second, some non-linear operations commonly included in neural networks, such as ReLU, are not natively supported by FHE and must be replaced by, for example, high-order polynomial approximations or softmax functions (Daubechies et al., 2022; Zhang et al., 2024; Ishiyama et al., 2020). Unfortunately, these approximations require many more computations than their native counterparts and quickly accumulate ciphertext noise, necessitating additional bootstrapping steps and thus increasing latency.

This work introduces Safhire, a novel and practical hybrid FHE scheme for ML inference that overcomes the above limitations. Safhire leverages a simple yet powerful idea: we keep the computation of linear, parameterized layers (*e.g.*, convolutions and fully connected layers) on the server under encryption and push non-linear operations (*e.g.*, ReLU activations) to the client which evaluates them in plaintext. Operationally, Safhire proceeds in rounds that correspond one-for-one to linear blocks bounded by non-linearities.[1] We show the E2E workflow of Safhire in Fig. 2 (bottom). In each round, the client encrypts and sends the current layer input (steps 1–2), the server applies the encrypted linear transform (step 3) and returns the resulting ciphertexts (step 4). Upon receipt, the client decrypts and applies the exact non-linear operation (*e.g.*, ReLU), then re-encrypts the outputs for the next round (step 5). This process repeats until the forward pass completes after which the client obtains the final result (step 6). The intermediate decryption and re-encryption at the client helps avoid bootstrapping since we decrypt before the noise becomes unmanageable. Moreover, because all non-linearities are executed in plaintext on the client, Safhire avoids any server side non-linear approximations, preventing additional bootstrapping and amplified inference latency.

However, revealing intermediate outputs to the client with Safhire poses the risk of model reconstruction wherein the client may try to reverse engineer the model weights. This undermines *model confidentiality* as the server model often represents proprietary intellectual property and investment due to high training and deployment costs. To prevent this, Safhire randomly shuffles the outputs of the linear operations on the server before sending them to the client. The shuffling function and the associated unshuffling function is derived by the server from a per-session secret seed and is unknown to the client. We provide formal differential privacy guarantees resulting from the shuffling operation in combination with the default noise introduced by server side operations. Thus, Safhire makes model reconstruction practically as hard as full (or non-hybrid) server-side inference, in a black box setting. At the same time, we show that the correctness of the client-side decryption remains unaffected.

Under the hood, Safhire leverages the fully homomorphic encryption over the torus (TFHE)-based ring learning with errors (RLWE) cryptographic scheme which is widely used for practical homomorphic computation (Chillotti et al., 2018; Chartier et al., 2025). A key advantage of the TFHE scheme is that it allows us to apply fine-grained advanced optimizations, such as fast ciphertext packing and partial trace extraction, resulting in substantial performance improvements. Our optimizations reduce both computation and communication costs, yielding practical E2E latency under standard security parameters. We imple-

[1] While we primarily evaluate on CNNs (in particular ResNet models), our approach applies to any architecture that alternates linear operations with non-linear ones (*e.g.*, blocks that are linear up to an activation).

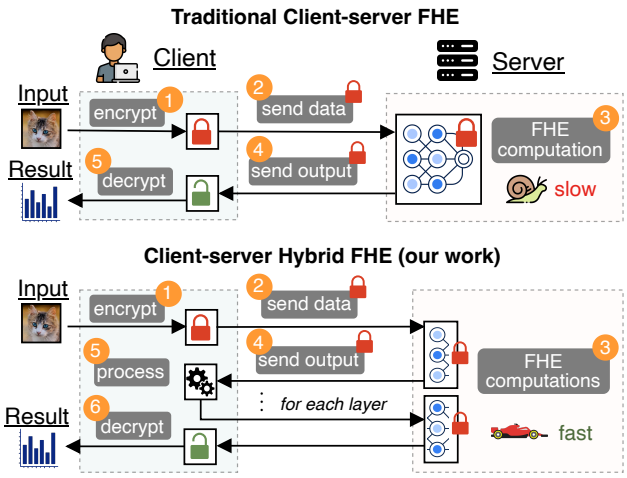

Figure 2: Existing FHE-based ML inference (top) performs all computation on the server in the encrypted domain; our hybrid scheme (bottom) offloads computationally expensive operations to the client to significantly improve efficiency.

ment Safhire and publish its code as a library on GitHub.[2] Our implementation supports inference using multi-core CPUs and GPUs.

We evaluate the efficiency of our scheme using widely adopted CNNs architectures and different datasets. Under realistic network conditions, Safhire using a single thread reduces E2E latency of inference requests by up to 10.7× compared to Orion, a state-of-the-art single-threaded baseline. When running Safhire with 32 threads, we obtain up to 7.4× reduction in E2E inference latency compared to Concrete-ML, a multi-threaded baseline that also offloads activations to the client. This comes at a manageable ciphertext communication cost of at most 499 MB for ResNet-18 and 170 MB for ResNet-20. Moreover, we show that through the use of multithreading Safhire reduces server-side execution time up to 54× compared to Orion. Using GPU acceleration, our inference duration on CIFAR-10 using model ResNet-20 can be as low as 13.65 seconds even with a modest 1.25MB/s internet connection. In summary, Safhire offers a practical step toward efficient FHE inference with high model accuracy.

Contributions. Our key contributions are as follows:

- We introduce Safhire, a hybrid and efficient FHE inference framework that executes linear, parameterized layers on the server under encryption while offloading non-linearities to the client in plaintext, eliminating bootstrapping while supporting non-linearities without approximations (Sec. 4).
- We show that, with randomized shuffling, Safhire helps preserve server-side model confidentiality, in addition to enabling clients to perform model inference in a private and secure manner. We derive the amplified differential privacy guarantees, jointly emerging from shuffling with inherent ciphertext noise (Sec. 5).
- We implement Safhire using the TFHE-based RLWE scheme and develop efficiency optimizations, including high-throughput ciphertext packing and partial trace extraction, greatly reducing computational cost.
- We evaluate Safhire on standard CNN model architectures and datasets, demonstrating up to 10.7× lower E2E latency and up to 63× less server compute than Orion depending on the dataset and configuration per inference with comparable accuracy (Sec. 6).

[2]SAFHIRE code: `https://anonymous.4open.science/r/SAFHIRE-fhe`

2 Background and Preliminaries

In this section, we present a high-level overview of the crucial concepts underlying the scheme, including the encryption–decryption process and essential operations such as key-switching, fast trace, and packing, which are integral to the design and optimal functionality of Safhire. A formal and more detailed explanation of these concepts is provided by Chartier et al. (2025). A summary of the key notations used in this work is provided in Appendix A.

2.1 Encryption and Decryption with LWE and RLWE

Learning with errors (LWE) is a cryptographic problem widely used in post-quantum cryptography due to its resilience against quantum attacks (Regev, 2009). The LWE problem essentially focuses on solving systems of noisy linear equations. Let us consider a discrete torus $T_p = \frac{1}{p}\mathbb{Z}_p$ and $T_q = \frac{1}{q}\mathbb{Z}_q$ where p is an integer representing the size of the message space and q is the size of the ciphertext space. In this work, values for p range from 2^8 to 2^{16} while the value of q is taken to be 2^{53} such that T_q is representable by double floating point precision. Besides, let $\mathbb{S}$ be a finite subset of $\mathbb{Z}$. Given a message $\mu \in T_p$ and a secret key $s \in \mathbb{S}^n$, we sample a n-dimensional mask a uniformly at random from T_q^n and a noise $e \sim \mathcal{N}(0, \Delta^2)$ where $\mathcal{N}(0, \Delta^2)$ denotes Gaussian distribution with mean 0 and standard deviation Δ. An LWE-encryption of μ using the secret key s is given by $\mathrm{LWE}_s(\mu) = (a, b)$, where $b = \sum_{i=1}^{n} s_i a_i + \mu + e \mod 1$. The corresponding LWE-decryption of (a, b) with a secret key s is given by $\pi_p(b - \sum_{i=1}^{n} s_i a_i)$ where π_p is a projection on the discrete torus T_p defined as $\pi_p(\xi) = \frac{\lfloor p\xi \rceil}{p}$ for any $\xi \in \mathbb{R}$. The decryption is correct as long as $|e| < 1/2p$.

The above encryption-decryption scheme is canonically extended to polynomials in the cyclotomic quotient ring $T_q[X]/\Phi_M[X]$ where Φ_M is a prime-power cyclotomic polynomial (*i.e.*, $M = t^\alpha$ for a prime t and a non-negative integer α). This gives rise to *ring learning with errors (RLWE)*. Given a polynomial $\mu[X] \in T_p[X]/\Phi_M[X]$ and a secret polynomial key $s(X) \in \mathbb{S}[X]$ of degree $N = \phi(M)$ (here ϕ denotes the Euler's totient function), we form $a^*(X) = \sum_{i=0}^{N-1} a_i^* \Omega_i^*(X)$, where each $a_0^*, \ldots a_{N-1}^*$ is sampled uniformly from T_q, and $e^*(X) = \sum_{i=0}^{N-1} e_i^* \Omega_i^*(X)$ where $e_i^* \sim \mathcal{N}(0, \Delta^2)$ for all $i = 1, \ldots, N-1$. Here, the family $(\Omega_i^*)_{0 \le i \le N-1}$ is meant to be the dual basis of $(X^i)_{0 \le i \le N-1}$ with respect to an appropriate scalar-product. An RLWE-encryption of $\mu(X)$ using the secret key polynomial $s(X)$ is given by $\mathrm{RLWE}_{s(X)}(\mu(X)) = (a(X), b(X))$ where $a(X) = (\Omega_0^*)^{-1}(X) a^*(X)$ and $b(X) = s(X)a(X) + \mu(X) + (\Omega_0^*)^{-1}(X) e^*(X)$. Similar to LWE, the decryption is performed by applying π_p coefficient-wise to $b(X) - s(X) \cdot a(X)$ and it is correct if $\left\|(\Omega_0^*)^{-1}(X) e^*(X)\right\|_\infty < 1/2p$.

2.2 Key-switching

Key-switching is a technique that changes an RLWE ciphertext of a given message encrypted under one key to another RLWE ciphertext of the same message encrypted under another key. Given an RLWE encryption of $\mu(X)$ with a secret key $s(X)$, and a key-switching key $\mathrm{KSK}_{s(X) \to s'(X)}$, the key-switching operation outputs $\mathrm{RLWE}_{s'(X)}(\mu(X))$, an RLWE encryption of $\mu(X)$ with secret key $s'(X)$. In RLWE schemes, ring automorphisms are applied to ciphertexts, which effectively change the secret key from $s(X)$ to $s(X^d)$ for some d coprime with M. With key-switching, $\mathrm{RLWE}_{s(X^d)}\left(\mu(X^d)\right)$ is converted back to $\mathrm{RLWE}_{s(X)}\left(\mu(X^d)\right)$.

2.3 Fast Computation of the Trace Operator

The *trace operator* is a fundamental concept in algebraic fields and structures that allows for the mapping of elements from a field extension back to the base field. Trace operators play a crucial role in the analysis and manipulation of algebraic structures effectively, and they need to be computed efficiently to be used in optimization of packing algorithms (*cf.*, Sec. 4.4). The trace operator of a polynomial $P(X)$ is given by $\mathrm{Tr}\left(P(X)\right) = \sum_{\substack{1 \le d \le M \\ \gcd(d,M)=1}} P\left(X^d\right) \mod \Phi_M(X)$. This can be homomorphically evaluated through N key-switches. This is not only slow but also induces a lot of noise. To address this issue, one can use the factorization of the trace through partial traces. Recall that polynomials in $\mathcal{K} = \mathbb{Q}[X] \mod \mathbb{Z}[X] \mod \Phi_M$ form a Galois field extension of $\mathbb{Q}$ and, that there exists a towering field extension $\mathbb{Q} = \mathcal{K}_0 \subset \mathcal{K}_1 \subset \ldots \subset \mathcal{K}_\alpha = \mathcal{K}$. Therefore, denoting $\mathcal{K}_{i+1}$ as the Galois field extension of $\mathcal{K}_i$, and letting $\mathrm{Tr}_{\mathcal{K}_i/\mathcal{K}_j}$ be the partial trace

from $\mathcal{K}_i$ down to $\mathcal{K}_j$ for $i > j$, and using the fact that $\mathrm{Tr}_{\mathcal{K}_1/\mathcal{K}_0}$ can be further factorized, Chartier et al. (2025) conceived an algorithm that homomorphically computes the complete trace with $(\alpha-1)(t-1)+\sum_{\ell\in\omega_{t-1}}(\ell-1)$ key-switches, where M is as defined in Sec. 2.1, and ω_s is the set of all prime factors of any natural number s counted with their multiplicity.

2.4 Packing

The *packing* operation combines several LWE ciphertexts encrypting messages $\mu_i \in T_p$ into a single RLWE ciphertext that encrypts a polynomial whose coefficients are the μ_is. We use RLWE to reduce communications overhead when exchanging ciphertexts. This technique was first introduced in the setting of power-of-two cyclotomic polynomials and later extended to prime-power cyclotomic fields (Chartier et al., 2025). *Fast packing* algorithms transform the input LWE ciphertexts into an RLWE ciphertext such that the first coefficient of the encrypted polynomial corresponds to the coefficient of interest. However, in this process, all other coefficients are randomized. The homomorphic trace operator is then applied to zero out all but the relevant coefficient. The resulting ciphertexts are subsequently rotated and summed to obtain a single ciphertext encrypting all the μ_is. By exploiting the decomposition property over towering Galois expansions, Chartier et al. (2025) showed how partial trace operations can be shared among different ciphertexts. This optimization reduces the number of required key-switching operations to $\sum_{\ell\in\omega_{t-1}}(\ell-1) + \sum_{1\le i\le\alpha}\frac{1}{t^{i-1}}$ per packed coefficient, where t, α, and ω_{t-1} are as defined in Sec. 2.1 and 2.3.

3 System and Threat Model

We now describe our system and threat model, and list the assumptions made in this work.

ML Model. The server stores an ML model parameterized with weights θ. Each model weight is stored in plaintext and only known by the server. Since we are using a TFHE scheme which supports only integer arithmetic, we consider the case where the model weights and the activations (or outputs) are quantized. Crucially, the accumulators, which store intermediate sums of multiply–accumulate (MAC) operations also operate under predefined integer bit-widths. To achieve this, the server could apply state-of-the-art quantization-aware training (QAT) techniques that enforce fixed-point integer representations for the weights, the activations and the accumulator (Colbert et al., 2024; 2023; Ni et al., 2021). We refer to the bit-width of the accumulator as b (*e.g.*, 12-bit precision). The bit-widths of weights and activations are typically lower than the accumulator (*e.g.*, 4-bit precision) (Colbert et al., 2024). All the bit precisions are public information and known to clients. Training a quantized model with 4-bit weights, 4-bit activations and 14-bit accumulator can maintain accuracy within 1% of the original floating-point model (Colbert et al., 2024).

Clients. The client has some input data *e.g.*, an image that it wants to evaluate using the server's model θ. The client generates a unique session identifier for each inference request. All server-client messages carry this session identifier and a monotonic round counter; messages from stale or mismatched rounds are rejected. We assume that the client remains online during the inference request in order to receive and process the intermediate outputs by the server, and to send back the processed outputs to the server. Thus, Safhire requires increased client participation across multiple short communication rounds compared to traditional client-server FHE where the client only needs to be available to send the encrypted input data and collect the inference results from the server. In practice, we find that each round typically completes within seconds, making transient disconnections easily mitigated by simple session resumption. Furthermore, this assumption is consistent with prior secure inference frameworks (Juvekar et al., 2018; Mishra et al., 2020; Smajlović et al., 2025).

Threat Model. Our security objective is two-fold. On the one hand, we want to preserve *data privacy*, preventing the server from inferring information from the client input data that is used for inference. On the other hand, we want to preserve *model confidentiality*, preventing the client from learning about the model weights θ. Model confidentiality is important in client-server deployments because the model often represents proprietary intellectual property and investment due to high development and training costs. Exposing model weights to clients could allow them to replicate or reverse-engineer the service, undermining monetization and intellectual property protections.

We assume that the client and server follow the FHE protocol. However, we assume that the server is honest-but-curious, *i.e.*, it faithfully runs the protocol yet attempts to learn about the client inputs from all messages it sees. On the other hand, we assume that the clients can be adversarial *i.e.*, they try to reverse engineer θ. To achieve this, they may send arbitrary input to the server in every round. Network traffic is authenticated and encrypted (*e.g.*, using TLS) and we do not consider side-channel leakage outside the protocol, such as timing or power measurements on client hardware. Our explicit leakage is limited to model metadata disclosed during the setup phase (*cf.*, Sec. 4.2) and the size and number of messages sent between the server and client.

4 Design of Safhire

We now describe the design of Safhire. We first present the high-level workflow in Sec. 4.1 and then explain each step in detail in the remaining sections.

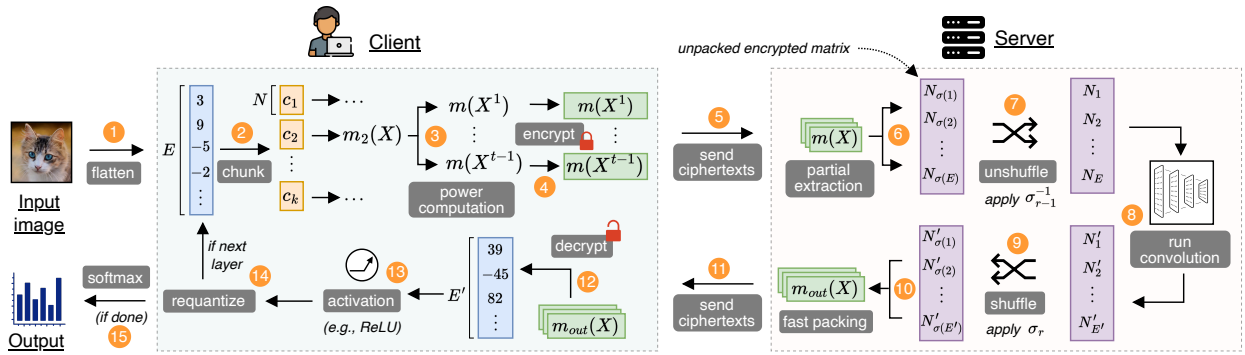

Figure 3: The workflow of the Safhire during an inference request using a CNN model. The box on the left and right shows the operations performed on the client and server, respectively.

4.1 Safhire in a Nutshell

The core idea behind Safhire is to split inference across the client and server: the server performs all linear operations (*e.g.*, convolutions, fully connected layers) under RLWE encryption, while the client applies the nonlinearities (*e.g.*, ReLU) in plaintext. By letting the client periodically decrypt intermediate results, our hybrid design prevents the noise accumulation that would otherwise require costly bootstrapping on the server. Moreover, evaluating nonlinearities in the clear avoids the overhead of polynomial approximations used in conventional FHE-based inference. Together, these choices yield substantial speedups over prior schemes. This design, however, introduces a new challenge: intermediate plaintext outputs become visible to the client, creating the risk of reverse-engineering the model weights. To protect model confidentiality, Safhire applies a secret, random permutation to the server's outputs before sending them back to the client. This obfuscation makes it practically infeasible for the client to reconstruct model parameters.

Fig. 3 illustrates the E2E workflow for an inference request in Safhire. The inference proceeds for L rounds, corresponding to each model layer. At the start of round r, the client has a flattened representation of the input image (when $r = 1$, step 1) or the output of the previous layer (when $r > 1$). The client encrypts these polynomials into multiple RLWE ciphertexts and sends them to the server (steps 2-5). The server then prepares the input matrix for the convolution by unpacking the coefficients in the RLWE ciphertexts using an optimized *partial extraction* scheme (step 6). Before running the linear operations, the server unshuffles ciphertexts to undo a secret permutation from the previous round, which is required for model confidentiality (step 7). The server then runs the convolution, shuffles the columns of the convolution output, and executes a fast packing algorithm to convert this output into multiple RLWE ciphertexts (steps 8-10). These ciphertexts are sent back to the client (step 11). The client decrypts these ciphertexts, applies the activation in clear

and requantizes the outputs for the next layer (steps 12-14), and repeats. Finally, when $r = L$, the client optionally runs softmax in clear to obtain the final output (step 15).

4.2 Protocol Setup

The protocol starts by the client generating an RLWE encryption key $s(X)$ and a key-switching key KSK. The key-switching key for automorphism $X \to X^d$ is given by $\mathrm{RLWE}_{s(X)}\left(\frac{s(X^d)}{D^i}\right)$ for $1 \leq i \leq l$, where l and D, representing *depth of decomposition* and *base*, respectively, are the parameters widely used in the context of homomorphic modular products (Chartier et al., 2025). The client computes the key-switching key for at most $\alpha(t-1)$ indexes. The client then requests various elements it requires to process the client-side operations, including the required input shape for the first layer, all other layers input and output size, the specifications of the activation functions used (*e.g.*, ReLU or sigmoid), and the per layer re-quantization scale η. This exchange only has to be done once for each unique model θ at the server and its communication overhead is negligible.

4.3 Client Operations (pre-server)

Here, we outline the operations performed by the client in a given round before communicating with the server.

Flatten and chunk. For an input image $I \in \mathbb{Z}^{H \times W \times C_{\text{in}}}$, where H, W, and C_{in} represent the kernel height, kernel width, and number of input channels, respectively, is first flattened into a fixed order (*e.g.*, row-major with channels last) to obtain a one-dimensional vector $V \in \mathbb{Z}^E$ (step 1 in Fig. 3). The client then splits V into k chunks $c_0, \ldots, c_{k-1}$ each of length N, and applies zero-padding to the final chunk c_{k-1} (step 2). Each chunk $c_j = (c_{j,0}, \ldots, c_{j,N-1})$ is hence embedded as the coefficients of a polynomial $m_j(X)$, *i.e.*, $m_j(X) = \sum_{i=0}^{N-1} c_{j,i} X^i$.

Power computations. Next, the client computes, for each polynomial $m(X)$, evaluations at specific powers of X (step 3). The range of these powers is determined by the *trace extraction level* parameter γ. After receiving the encrypted powers from the client, the server performs *fast packing* (*cf.*, Sec. 4.4) at trace level γ, thereby significantly reducing the number of key switches per packed element. For example, if $\gamma = 0$, the client does not compute any powers and simply sends an encrypted version of each $m(X)$ to the server. If $\gamma = 1$, the client additionally computes and sends $m(X^j)$ for all $j \in \{1, \ldots, t-1\}$. For $\gamma = 2$, the client computes $\nu(X) = m(X^{j(kt+1)})$ for all $j \in \{1, \ldots, t-1\}, k \in \{0, \ldots, t-1\}$. Each $\nu(X)$ is then encrypted using a private $s(X)$ as $\mathrm{RLWE}_{s(X)}(\nu(X))$ (step 4) and sent to the server.

4.4 Server Operations

We next describe the operations performed by the server in a given round r.

Partial trace extraction. If $\gamma > 0$, for each encrypted polynomial chunk $\mathrm{RLWE}_{s(X)}(m_j(X))$, using its associated encrypted powers computed by the client, the server extracts an RLWE encryption of $\mathrm{Tr}_{\mathcal{K}_\gamma/\mathcal{K}_0}\left((m_j(X)\overline{\Omega}_i^*(X)\right)$ for $0 \leq i \leq N-1$, where $\mathrm{Tr}_{\mathcal{K}_0/\mathcal{K}_0}$ is the identity mapping and $\overline{P}(X) = P\left(X^{M-1}\right)$ for any $P \in \mathcal{K}$ (step 6). It is worth noting that, as $c_{j,i} = \langle m_j(X), \Omega_i^*(X)\rangle = \mathrm{Tr}\left(m_j(X)\overline{\Omega}_i^*(X)\right)$, where $c_{j,i}$ is as defined in Sec. 4.3, each of the extracted RLWE can be mapped to an encryption of one of the input coefficients using the homomorphic evaluation of the partial trace $\mathrm{Tr}_{\mathcal{K}/\mathcal{K}_\gamma}$. For a given γ, the partial trace is extracted using the expressions derived in the following lemma. In the interest of space, the proof is postponed to Appendix B.

Lemma 1. *For all $P \in \mathcal{K}$ and $i = 0, \ldots N-1$, we have:*

$$\mathrm{Tr}_{\mathcal{K}_\gamma/\mathcal{K}_0}\left((P(X)\overline{\Omega}_i^*(X)\right) = \begin{cases} \sum_{k=1}^{t-1} P(X^k)\overline{\Omega}_i^*(X^k) \text{ for } \gamma = 1 \\ \sum_{j=0}^{t-1}\sum_{k=1}^{t-1} P(X^{k(jt+1)})\overline{\Omega}_i^*(X^{k(jt+1)}) \text{ for } \gamma = 2 \end{cases}$$

Key size $M = t^\alpha$	**Trace extraction level (γ)** $\gamma = 0$ (Base)	$\gamma = 1$	$\gamma = 2$
3^7	2.50	1.50	0.50
5^5	3.25	1.25	0.25
7^4	4.16	1.16	0.16

Table 1: Key switches per output coefficient for different start partial trace extraction level (γ).

As addition is homomorphic and the powers of the message sent by the client are encrypted, the only part that needs to be addressed now is the multiplication by the clear polynomial. Recalling the expression of dual basis, one can derive an expression for $\overline{\Omega}_i^*(X^d)$. However, these polynomials do not have integer coefficients, and multiplication by a clear-text polynomial $P(X)$ is homomorphic if and only if $P(X) \in \mathbb{Z}[X]$. To alleviate this problem, the server instead multiplies by $M^{-1}\left(M\overline{\Omega}_i^*(X^d)\right)$ where $M^{-1}M = 1 \mod p$, where p is as defined in Sec. 2. Moreover, as these polynomials have only two non-zero coefficients, the multiplication can be cheaply performed by rotating and summing the clear-text, taking $4M$ elementary operations for one RLWE encryption.

Unshuffle. Before the server runs each encrypted linear layer, it applies an unshuffle operation σ_{r-1}^{-1} to the incoming unpacked encrypted matrices $N_{\sigma(1)}, ..., N_{\sigma(E)}$ to undo the secret permutation σ_{r-1} that was applied to the previous layer's outputs when they were packed and returned to the client (step 7). As we will discuss in Sec. 5, this shuffling and unshuffling is necessary for model confidentiality. For the first linear layer ($r = 1$) there is no preceding permutation, so we set $\sigma_0^{-1} = \text{id}$; consequently, the first unshuffle is a no-op. This results in the rows $N_1, ..., N_E$ of the unpacked, unshuffled encrypted matrix N with shape $E \times 2M$.

Convolution. The server reshapes matrix N to obtain N_r with shape $H \times W \times C_{\text{in}} \times 2M$. It then runs the convolution operation where $2M$ can be considered as batch size. After reshaping, this results in matrix N' with dimension $E' \times 2M$.

Shuffle. Next, the server applies the permutation σ_r to matrix N', which shuffles its rows (step 9). The permutation σ_r is generated uniformly at random from a secret seed derived from the session identifier and the current round number, ensuring it is unique for each round and client. The resulting matrix has rows $N'_{\sigma(1)}, \ldots, N'_{\sigma(E)}$.

Fast Packing. The *fast packing* operation takes as input $t^{\beta-1}(t-1)$ RLWE ciphertext encryptions, for $1 \leq \beta \leq \alpha - 1$, of $\text{Tr}_{\mathcal{K}_\gamma/\mathcal{K}_0}\left(\overline{\Omega}_0^*(X)m'_i + B(X)\right)$ where $B \in \ker(\text{Tr}_{\mathcal{K}/\mathcal{K}_\gamma})$ and outputs $\text{RLWE}_{s(X)}(m'(x))$, where $m'(X) = \sum_{i=0}^{t^{\beta-1}(t-1)-1} m'_i X^{i\frac{M}{t^\beta}}$ (step 10). Thus, the packing operation reduces the number of ciphertexts by a factor $t^{\beta-1}(t-1)$. Finally, the resulting chunk-polynomials $m'(X)$ are sent to the client (step 11).

Packing is a computationally expensive operation, dominating runtime (*cf.*, Fig. 5) due to the key-switching operations needed for automorphisms. Table 1 shows the number of key switches per packed element, which grows with the size of the prime t and is lower bounded by 2.5. As a result, packing becomes a key factor in runtime and provides an important lever in selecting appropriate key sizes. Applications must balance packing cost (favoring smaller t) against precision and security, as larger keys improve security but increase memory usage and slow basic operations.

However, thanks to the partial trace extraction operation, the ciphertexts that our fast packing algorithm receives as inputs in step 3 already encrypt the image of the desired polynomial by the partial trace $\text{Tr}_{\mathcal{K}_\gamma/\mathcal{K}_0}$. This means that the fast packing procedure does not have to homomorphically compute the trace $\text{Tr}_{\mathcal{K}_\gamma/\mathcal{K}_0}$, thus resulting in noticeable computational gains. From Table 1, we see that setting $\gamma = 1$, *i.e.*, skipping the $\text{Tr}_{\mathcal{K}_1/\mathcal{K}_0}$ operation, makes the number of key switches per packed element bounded between 1 and 1.5 for any choice of t, thus relaxing the aforementioned trade-off. Setting $\gamma = 2$, *i.e.*, skipping both $\text{Tr}_{\mathcal{K}_1/\mathcal{K}_0}$ and $\text{Tr}_{\mathcal{K}_2/\mathcal{K}_1}$ allows to further improve this result. At the cost of a bit more computation from the client and with more data transfers, the server can significantly reduce the cost of packing. We have investigated this trade-off in our experiments (*cf.*, Sec. 6.4).

4.5 Client Operations (post-server)

Upon receiving the packed ciphertexts $\mathrm{RLWE}_{s(X)}(m'(X))$, the client decrypts each chunk to recover a polynomial $m'(X)$ whose coefficients contain a block of output entries. Reading the designated packed positions and concatenating across all returned ciphertexts (and dropping any padding) yields the layer output vector $V' \in \mathbb{Z}^{E'}$. The client then applies the layer's activation function f in plaintext element-wise (*e.g.*, ReLU). However, even when the weights and the inputs are quantized to low-bit values (*e.g.*, 4-bit), the output derived from several multiply-accumulate operations of a convolution can exceed this bit precision. Therefore, the client requantizes the output to the predefined input precision for next layer: using the per-layer scale η, we map $y \mapsto \mathrm{clip}\big(\lfloor y/\eta \rceil\big)$. If this is the final layer of a classifier, the client directly computes the softmax over V' to obtain probabilities and returns them to the user (step 15). Otherwise, V' becomes the input to the next round $r+1$ (step 14).

4.6 Trade-offs

At the core of Safhire lies a deliberate architectural trade-off: we avoid costly bootstrapping altogether by performing intermediate decryptions on the client and executing non-linear operations in plaintext. This significantly reduces the computational burden on the server, eliminating one of the main performance bottlenecks in FHE inference. However, this comes at the cost of additional network communication, as intermediate ciphertexts must be transmitted back and forth between the client and server after each linear block. Thus, Safhire exchanges the high compute latency of bootstrapping for higher communication volume, a trade-off that is particularly advantageous when network bandwidth is abundant but compute resources are at a premium.

5 Model Confidentiality

We now analyze the model confidentiality guarantees of Safhire. As outlined in Sec. 3, we assume that the model held by the server must remain private from the clients (model confidentiality). Clients may send arbitrary inputs and use all server responses to infer the model weights.

To contextualize our contribution, we first establish a concrete attack vector. We present a novel method sufficient to demonstrate the system's practical vulnerabilities, thereby highlighting the crucial role server-side shuffling plays in Safhire to uphold model confidentiality. The key outcome of our presented attack is that it grants the adversary only partial information: the set of columns or rows of the target matrix, but not their canonical ordering. Consequently, the adversary faces the challenging reconstruction task of assembling the matrix from its components, which are permuted by an unknown function π. Crucially, to the best of our knowledge, there are no existing model-reconstruction attacks that function solely by exploiting a random ordering of the model weights. Furthermore, a brute-force search for the correct permutation is computationally infeasible given the size of our target matrix, *e.g.*, 512 columns. Therefore, *even if an adversary successfully executes this attack, they cannot efficiently recover the original matrix.*

We start by showing how, in the absence of shuffling, the participating clients can deliberately craft and send inputs to the server that could compromise the confidentiality of the server-side model. Consequently, we illustrate how Safhire remains robust against such adversarial model-inference attempts due to its integrated shuffling mechanism. We highlight that, to the best of our knowledge, we mount the most effective attack on Safhire where the client aims to recover the server-side model.

Attack Formulation. Recalling that Safhire focuses on linear server-side computations, fully connected and convolutional blocks can be written as $f(x) = Ax + b$ with $A \in \mathbb{R}^{m\times d}$ and $b \in \mathbb{R}^m$. A convolution is a matrix–vector product with the corresponding Toeplitz matrix (Gray, 2006). We compare three interface variants, and write σ_{in} and σ_{out} for unknown input and output permutations, which may be freshly sampled per query. ***(i) No shuffling.*** A client querying $x = 0$ reveals b. For each basis vector e_i, the response is $f(e_i) = Ae_i + b = a_i + b$, where a_i is the i-th column of A; subtracting b gives a_i. Repeating over every i, the client exactly recovers A and b, thus compromising model confidentiality. ***(ii) Output shuffling only*** *(first round of Safhire).* The server returns $\sigma_{\text{out}}(Ax + b)$. The probe $x = 0$ reveals b up to a permutation

σ_{out} unknown to the client. To recover a column a_i up to the same permutation, the client queries $x_1 = e_i$ and $x_2 = 2e_i$, obtaining $y_1 = \sigma'_{\text{out}}(b + a_i)$ and $y_2 = \sigma''_{\text{out}}(b + 2a_i)$. For any entry $\tilde{b}_k$ in the shuffled bias, there exists an index j such that $(\tilde{b}_k + y_{2,j})/2$ appears in y_1, where $y_{2,j}$ is the j-th element of the vector y_2. Then $(y_{2,j} - \tilde{b}_k)/2$ equals the corresponding entry of a_i at index $\sigma^{-1}_{\text{out}}(k)$. Scanning all $k \in [m]$ reconstructs the multiset of entries of a_i and iterating over $i \in [m]$ recovers all columns, but only *up to a common unknown permutation*. If ties are present, they can be disambiguated with extra probes such as $3e_i$. Thus, even though output shuffling may reveal the values of the model weights, their ordering would remain private from the clients without the knowledge of the permutation function σ_{out}. Therefore, in order to infer the exact indices of the model weights, the clients need to know σ_{out}. To the best of our knowledge, no existing model-inversion-based attack in the literature can work just by exploiting an arbitrary ordering of the values of the model weights. In practice with $m = 512$ in ResNet-18, even with a brute force approach, the likelihood of guessing σ_{out} is $1/m! \approx 0$, thus making it practically impossible for the client to retrieve the exact model held by the server. ***(iii) Input and output shuffling*** *(subsequent rounds of* Safhire*).* The server computes $\sigma_{\text{out}}(A\sigma_{\text{in}}x + b)$, *i.e.*, an input $x = e_i$ is mapped to $e_{\sigma(i)}$ for some $\sigma(i) \in [m]$. Thus, the client cannot target a fixed column across queries without knowing σ_{in} and σ_{out}. Client-side observations have the form $\sigma_{\text{out}}\big(b + \sum_i x_{\sigma_{\text{in}}(i)} a_{\sigma_{\text{in}}(i)}\big)$, which reveal only unordered mixtures; the individual columns a_i cannot be isolated even up to permutation, and only the histogram of the weights of A is revealed. Hence, the model-inversion-based reconstruction attacks are not applicable once input shuffling is applied.

It is important here that the server does not shuffle the outputs from the last layer, as the client is expected to perform the `arg max` operation on the logits it receives from the server to conclude the classification-based inference. However, not shuffling the last layer's output still preserves the privacy of the server's model up to the unshuffling function, analogous to the input permutation in our analysis, of the preceding round. In other words, by observing the server's outputs in the last round, the client can learn the individual rows of A but not their order up to the aforementioned input permutation of the previous round.

In addition to making it practically impossible for a client to exploit intermediate layer information to reconstruct the model stored on the server, we extend the advantage of shuffling the convolution outputs in every round to derive formal guarantees of *differential privacy (DP)* (Dwork & Roth, 2014). To this, we explore the *shuffle model* of DP (Bittau et al., 2017; Cheu et al., 2019) and use the privacy amplification results of shuffling. For an input space $\mathcal{X}$ and output space $\mathcal{Y}$, the shuffle model of DP involves a *local randomizer* $\mathcal{R} : \mathcal{X} \mapsto \mathcal{Y}$ and a *shuffler* $\psi : \mathcal{Y}^n \mapsto \mathcal{Y}^n$ for some $n \in \mathbb{N}$. $\mathcal{R}$ is responsible for obfuscating any $x \in \mathcal{X}$ by mapping it to some $y \in \mathcal{Y}$. For any set of messages $x_1, \ldots, x_n$ in $\mathcal{X}$ that has been point-wise obfuscated by $\mathcal{R}$, ψ applies a random permutation σ to the locally obfuscated messages $\mathcal{R}(x_1), \ldots, \mathcal{R}(x_n)$ to obtain $\mathcal{R}(x_{\sigma(1)}), \ldots, \mathcal{R}(x_{\sigma(n)})$. This essentially ensures that, for any $i = 1, \ldots, n$, an observer cannot link a certain message $\mathcal{R}(x_{\sigma(i)})$ to its corresponding sender i without knowing the permutation function σ used by the shuffler.

Recent studies (Feldman et al., 2022; Erlingsson et al., 2019; Feldman et al., 2023; Biswas et al., 2023; Koskela et al., 2023; Balle et al., 2019) on shuffle model of DP have shown that if $\mathcal{R}$ satisfies *local* DP, then the shuffling results in an amplification of the local DP guarantees and $\psi \circ \mathcal{R}^n$ satisfies central DP from the perspective of the observer with the corresponding amplified privacy bound. Aligned with this, in the following theorem, we formally derive the amplified DP guarantees of the server's model under Safhire. In the interest of space, the proof has been postponed to Appendix C.

Theorem 2. *In each round of* Safhire*, if the convolution outputs satisfy $(\varepsilon_0, \delta_0)$-local DP, then for any $\delta \in [0, 1]$ s.t. $\varepsilon_0 \leq \ln\left(\frac{n}{16\ln(2/\delta)}\right)$ the convolution outputs as a whole in each round, as observed by the client, satisfies $(\varepsilon, \delta + (e^{\varepsilon} + 1)(e^{-\varepsilon_0}/2 + 1)n\delta_0)$-DP, where*

$$\varepsilon \leq \ln\left(1 + \frac{e^{\varepsilon_0} - 1}{e^{\varepsilon_0} + 1}\left(\frac{8\sqrt{e^{\varepsilon_0}\ln(4/\delta)}}{\sqrt{n}} + \frac{8e^{\varepsilon_0}}{n}\right)\right).$$

Remark 3. *We empirically observe that the noise introduced by the fast packing operation into each output of the convolutional layers follows a Gaussian distribution (see discussion below and Fig. 10). Therefore, we empirically verify that the outputs of the convolution sent to the client by the server satisfy $(\varepsilon_0, \delta_0)$-local DP.*

Precision (b)	**Key size** ($M = t^\alpha$)	**RLWE Noise** (Δ)	$\gamma = 0$		$\gamma \in [1, 2]$	
			D	l	D	l
8 bits	3^7	2^{-36}	512	3	512	3
12 bits	7^4	2^{-51}	2^{12}	3	2^{16}	2
16 bits	5^5	2^{-51}	310	5	310	4

Table 2: Parameterization of SAFHIRE for different precision levels b and levels of partial trace extraction level γ.

This, in turn, makes Th. 2 applicable to SAFHIRE *and, hence, the outputs received by the client in any given round satisfy* (ε, δ)*-DP with an amplification in the privacy level as given by Th. 2.*

To support the applicability of Th. 2 to SAFHIRE as outlined in Rem. 3, Fig. 10 in Appendix D.1 provides an illustrative example showing that the noise induced by fast packing on the first layer outputs of ResNet-20 applied to CIFAR-10 indeed follows a Gaussian distribution. For subsequent convolutional layers, convolution-induced noise depends on kernel weights and size, while fast packing noise depends only on its parameters, independent of input noise. These parameters remain consistent across all layers and models, and convolutional noise is negligible compared to the noise induced by fast packing. Hence, the noise distribution observed in the first convolutional layer (Fig. 10) is representative of the noise distribution across all convolutional layers in the network.

Remark 4. *A noteworthy observation from Fig. 10 is that the noise added to the outputs of the convolutional layers lies within the range* $[-1/2p, 1/2p]$*, which ensures correct decryption on the client side, in accordance with the TFHE decryption procedure detailed in Sec. 2.1. Thus, the DP amplification bound comes at no extra loss in the correctness of the decryption. For certain sensitive applications, if the server seeks stronger formal privacy guarantees to better protect its model weights, it may choose to inject a higher level of noise into the convolutional outputs. However, doing so increases the risk of decryption errors on the client side, potentially degrading the utility. This trade-off needs to be evaluated for context-specific applications and the corresponding requirements of privacy and utility.*

In summary, this section demonstrates how the shuffle operation in SAFHIRE prevents clients from inferring the server-side model. We show that it is practically impossible for clients to reconstruct the server's model, even with sending adversarially crafted inputs intended to compromise model confidentiality. Moreover, we show that the noise introduced by fast packing operations is further amplified through shuffling, thereby deriving an amplified DP guarantee without affecting the correctness of client-side decryption. In certain scenarios requiring a stronger formal DP guarantee, the server may inject additional noise into the outputs shared with clients, albeit at the potential cost of reduced utility.

6 Evaluation

We conduct an experimental evaluation of SAFHIRE and answer the following questions:

1) How does single-threaded SAFHIRE compare in terms of runtime against ORION, a state-of-the-art FHE framework (Sec. 6.3)?

2) What is the effect of different trace extraction and precision levels on the latency of individual SAFHIRE operations (Sec. 6.4)?

3) How does the runtime of SAFHIRE evolve when increasing the number of utilized CPU threads for different trace extraction and precision levels (Sec. 6.5)?

4) What is the runtime of SAFHIRE on a GPU for different trace extraction and precision levels (Sec. 6.6)?

Model	Orion	Extraction level (γ)	Precision (b)					
			8 bit		12 bit		16 bit	
			Safhire	Speedup	Safhire	Speedup	Safhire	Speedup
ResNet-20 (CIFAR-10)	866.5 s	0	160.8 s	5.4×	255.3 s	3.4×	407.5 s	2.1×
		1	117.4 s	7.4×	95.7 s	**9.1**×	219.1 s	4.0×
		2	80.7 s	**10.7**×	116.9 s	7.4×	107.1 s	**8.1**×
ResNet-18 (Tiny)	1253.9 s	0	591.6 s	2.1×	889.1 s	1.4×	1257.9 s	1.0×
		1	466.0 s	2.7×	416.4 s	3.0×	666.0 s	1.9×
		2	357.3 s	**3.5**×	377.3 s	**3.3**×	512.4 s	**2.4**×
ResNet-34 (ImageNet)	10819.6 s	0	4056.1 s	2.7×	6077.4 s	1.8×	8656.0 s	1.2×
		1	3095.8 s	3.5×	3024.8 s	3.6×	4901.4 s	2.2×
		2	2523.9 s	**4.3**×	2802.6 s	**3.9**×	3791.6 s	**2.9**×

Table 3: The E2E latencies of Safhire over Orion and associated speedups, for different trace extraction levels and precisions. To compute the duration of network communication in Safhire, we assume a conservative network speed of 1.25 MB/s. For each model and precision, we mark the largest speedup in bold.

6.1 Implementation

We implement Safhire, including its core cryptographic primitives, in the Julia programming language. The core machine learning architecture relies on the `NNlib.jl` library and we leverage `FFTW.jl` for efficient Fast Fourier Transform computations during keyswitches. We use the `Permutations.jl` library for handling permutation-based operations. Our implementation supports multi-threaded and GPU execution.

To verify the correctness of our implementation, we conduct inferences across multiple precision levels (b), partial trace extraction levels (γ), and model–dataset pairs. For each configuration, we generate random input vectors and perform inference using both a quantized cleartext model and Safhire. We then compare the resulting outputs element-wise. Across all tested configurations, we observe no discrepancies between the cleartext and encrypted inference results, giving us high confidence in the correctness of our implementation. This aligns with our empirical observations of the FHE noise in Appendix D.1.

6.2 Experimental Setup

Models and Datasets. We evaluate three representative CNN models commonly used in prior FHE studies (Ebel et al., 2025; Lee et al., 2022): ResNet-20 (0.27M parameters), ResNet-18 (11.2M), and ResNet-34 (21.8M) (He et al., 2016). We use three vision datasets, namely CIFAR-10 (Krizhevsky et al., 2009) (with image size 32×32), Tiny (Le & Yang, 2015) (with image size 64×64), and ImageNet (Deng et al., 2009) (with image size 224×224). Images across all datasets have three input channels. For efficiency evaluation in Safhire, the semantic content of the images is immaterial; however, the input shape and the number of output classes impact inference time, as they determine the sizes of the first convolutional and final classification layers. We use the standard ReLU function as activation function for Safhire. For ResNet-34 on the ImageNet dataset, we replace the max pooling with average pooling, as the latter is linear and therefore significantly more efficient under FHE.

FHE Parameterization. Table 2 presents the different parameters chosen in the Safhire implementation, for different precision levels b. The noise, D, and l are chosen to guarantee correct decryption of the ciphertext with high probability after each convolution and packing operation. The key size is then chosen to compromise between size, which directly affect memory usage and runtime, and the number of key switches required for fast packing. All parameter choices yield at least 128-bits of security when tested with M. Albrecht's security estimator (Albrecht et al., 2015).

Baselines. We compare Safhire against Orion (Ebel et al., 2025), a state-of-the-art FHE baseline that is based on the CKKS scheme, as well as Concrete-ML, a high-performance open-source FHE library based on the TFHE scheme, which provides a hybrid model inference implementation similar to our paper (Zama AI, 2024). Orion is a single-threaded CPU approach that optimizes inference by pairing single-shot multiplex packing with automated bootstrap placement (Cheon et al., 2017). However, unlike our hybrid RLWE design,

Dataset	Model	Orion (SiLU)	Safhire	
			16 bit	12 bit
CIFAR-10	ResNet-20	91.70%	90.00%	89.63%
Tiny	ResNet-18	57.00%	53.19%	53.16%

Table 4: Accuracy of SAFHIRE across configurations.

ORION keeps all layers encrypted and therefore still requires costly bootstrapping, as shown in Fig. 1. We compare ORION and SAFHIRE only under single-threaded CPU execution since ORION does not support multi-threaded CPU or GPU execution, while CONCRETE-ML is used as the baseline for multithreaded execution. Meanwhile, we use CONCRETE-ML in the hybrid inference mode where activations are run on the client side in plaintext, while linear layers are run encrypted on the server side. When using CONCRETE-ML we quantize the served models to 6-bit precision in order to ensure optimal performance, as we observe significant degradation in performance of CONCRETE-ML for higher precision.

Compute Platform. We conduct our experiments on three distinct hardware platforms to evaluate CPU and GPU performance separately. CPU-only evaluations are performed on two servers. The first is equipped with two AMD EPYC 7K62 CPUs with 2.60GHz base frequency, providing 192 logical cores total; 2003GB of RAM; and runs Ubuntu 20.04. Due to the hardware requirements of Orion, a more powerful server was needed for experiments involving the ResNet-34 model. This second machine, used exclusively for these tests, uses two Intel® Xeon® Platinum 8272L CPUs with a 2.60GHz base frequency and 104 logical cores; 2975GB of RAM; and also runs Ubuntu 20.04. GPU-accelerated experiments are run on an on-demand compute cluster. Each experiment utilizes a node with 24 cores of an AMD EPYC 7543 CPU with a 2.80GHz base frequency and a single NVIDIA A100-SXM4 GPU with 80GB of VRAM. These nodes run Ubuntu 22.04.

Metrics. Our evaluation focuses on two metrics: (1) the wall-clock time for client- and server-side operations, and (2) the total communication volume between the client and server. The E2E latency of SAFHIRE consists of computations on both the client and the server, and the ciphertext transfer time. Network latency is highly dependent on the conditions in the deployment setting and we separately measure the overhead of communication under different network conditions. Client-side operations include encryption, decryption, activations, and requantization, while server-side operations consist of packing, linear operations, and extraction. As the client-side operations are orders of magnitude faster, we mainly focus on optimizing the overhead of server-side operations. We report their individual times, as well as the total sum, which we refer to as *server-side execution time*. We also report the accuracy achieved by SAFHIRE under different precision levels.

6.3 Performance of Safhire Compared to Orion

First, we compare SAFHIRE against ORION on the same hardware for single-threaded execution. Table 3 compares the E2E latency against ORION under a conservative network speed of 1.25 MB/s (10 Mbit/s), for different extraction levels γ and precisions b. In this setting SAFHIRE offers significant speedups ranging from 1.0× for $\gamma = 0$ and 1.9× for $\gamma > 0$ up to 10.7×. Larger b increases runtime due to larger computational demands. For a fixed model and b, speedups increase when increasing γ in majority of cases. We recall that $\gamma = 0$ represents the baseline which does not use our extraction optimization which is a vital contribution of this work. Additionally, Table 4 reports the accuracy achieved by SAFHIRE and ORION. We use A2Q+ (Colbert et al., 2024) for training quantized models under the specified precision of the accumulator. For CIFAR-10, SAFHIRE is within 2.1% accuracy of ORION, while it is within 3.9% for Tiny. While we use the default hyperparameters of A2Q+, this gap can be further reduced by their careful tuning (Colbert et al., 2024).

We further explore the effect of network latency on the E2E inference time in Fig. 4. The points on this plot are calculated by summing the computational times for the client and server with an approximated network transfer time. This transfer time is derived by dividing the total exchanged data by a hypothetical bandwidth. We select hypothetical bandwidths ranging from 0.1–300 MB/s such that they are roughly logarithmically spaced and representative of consumer connections. For simplicity of presentation, we assume that the client and server have symmetrical upload and download speeds.

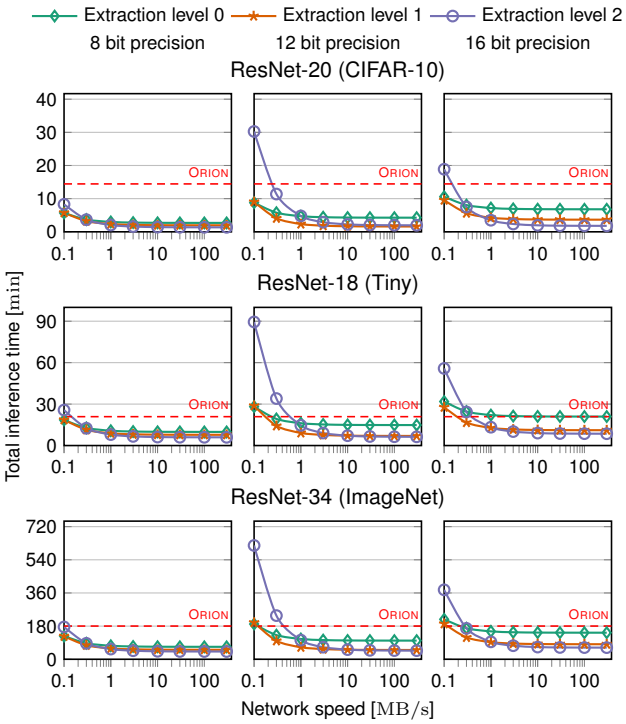

Figure 4: Total inference time for SAFHIRE under varying the network speed between the client and server. We consider networks speeds of 0.1, 0.3, 1, 3, 10, 30, 100, and 300 MB/s.

			Extraction level (γ)		
			0	**1**	**2**
Precision (b)	**8 bits**	**Client**	0.3 s	0.4 s	0.8 s
		Server	591.4 s	465.6 s	356.5 s
	12 bits	**Client**	0.3 s	1.2 s	5.7 s
		Server	888.8 s	415.2 s	371.6 s
	16 bits	**Client**	0.3 s	1.8 s	3.8 s
		Server	1257.5 s	664.3 s	508.6 s

Table 5: Breakdown of total inference time between client and server on Tiny with ResNet-18.

Fig. 4 shows that even for a network speed as low as 0.3 MB/s, the inference time of SAFHIRE is comparable or faster than ORION in most settings. For all higher bandwidths, SAFHIRE with extraction optimization ($\gamma > 0$) outperforms ORION, even at modest network speeds. These results also highlight that even at a network speed as low as 3 MB/s, the inference time for SAFHIRE is close to its performance at much higher speeds, emphasizing that computation, rather than communication, is the primary bottleneck beyond that point. At high speeds, $\gamma = 2$ yields little benefit over $\gamma = 1$, while at low speeds, $\gamma = 1$ clearly outperforms $\gamma = 0$. Overall, $\gamma = 1$ emerges as a robust choice across network regimes.

6.4 Single-threaded Performance of Safhire

Next, we analyze how the different operations in SAFHIRE contribute to its performance for single-threaded execution for inference with a single sample.

Table 5 reports the split of total computational time between the client and the server for Tiny inference with the ResNet-18 model across different values of b and γ. This table shows that the client-side execution time increases with the extraction level, yet it remains negligible compared to the server-side execution time, which is between 65× and 3682× greater across all tested settings. Therefore, for the rest of this section, we focus specifically on the server-side operations as they are the dominating factor in total inference latency.

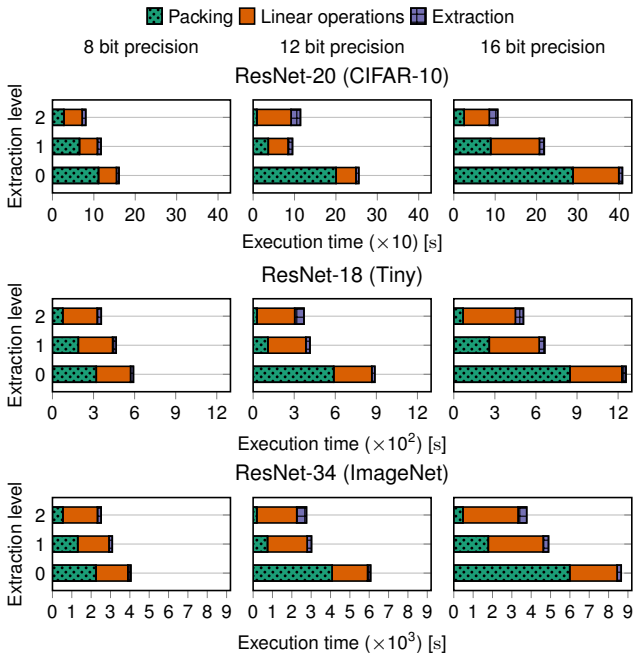

Figure 5: Breakdown of the server-side execution time for a single sample inference in Safhire across various settings.

Fig. 5 shows a time breakdown of server-side operations. We make two key observations. First, increasing γ beyond 0 significantly reduces the time taken for packing, for all models and values of b but slightly increases the time required for extraction. Second, linear operations require more time as b increases, particularly for ResNet-18 and ResNet-34, but remain constant as γ varies and b is fixed.

To further analyze the communication-computation trade-offs, we show the data transfer size to and from the server in Fig. 6, for different models and values of b and γ. For a fixed model and value of b, the size of data sent by the server to the client remains constant across different values of γ, but the size of uploaded data varies. This is because for higher γ, the client sends additional higher-degree polynomials to the server. In exchange for this higher upload cost, increasing γ allows the server to spend significantly less time on packing and slightly more on extraction (see Fig. 5).

Both Fig. 5 and Fig. 6 highlight the computation-communication trade-offs. As discussed in Sec. 4.3, this trade-off stems from the client providing the server with higher-degree polynomials for higher extraction level. This results in fewer key switches for the server, as shown in Table 1, which significantly reduces packing time. However, as can be seen, this also significantly increases the amount of data the client must upload. For example, with 16-bit quantization on CIFAR-10, increasing the extraction level from 0 to 2 reduces the server time spent on packing from 71% to 24% of the server-side execution time. In the same setting, however, the data uploaded by the client increases from 4.2 to 83.9 MB.

6.5 Multi-threaded Performance of Safhire

Next, we quantify the benefits of multithreading on the server-side performance of Safhire. For these experiments, we measure the total server-side execution time while utilizing 1, 4, 8, 16 and 32 threads. Fig. 7 shows the results of these measurements across various precisions b and extraction levels γ. Notably, for $b = 8$ server-side execution time stays close across different values of γ. Meanwhile, for $b = 12$ and $b = 16$ the gap between $\gamma = 0$ is significantly higher than for $\gamma = 1$ and $\gamma = 2$, which are relatively comparable to each other.

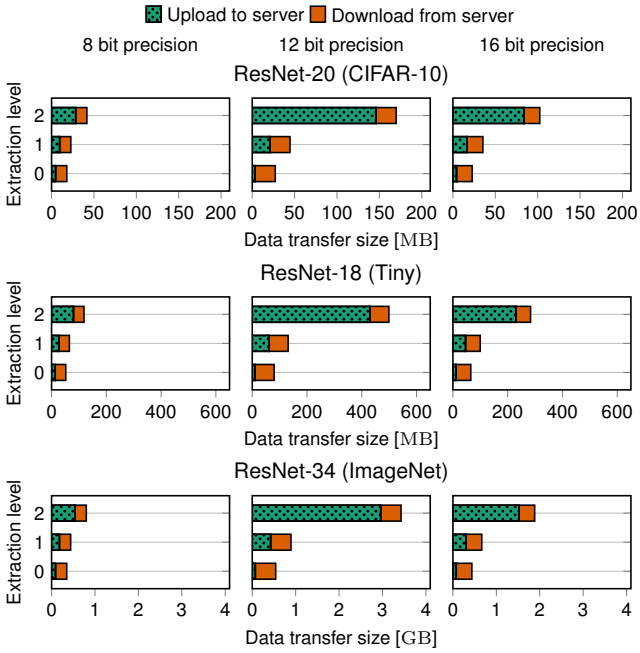

Figure 6: Breakdown of the client-server communication for a single sample inference in Safhire across various settings.

Model	Concrete-ML	Extraction level (γ)	Precision (b)					
			8 bit		12 bit		16 bit	
			Safhire	Speedup	Safhire	Speedup	Safhire	Speedup
ResNet-20 (CIFAR-10)	86.8 s	0	60.9 s	1.4×	109.6 s	0.8×	147.4 s	0.6×
		1	35.6 s	2.4×	22.2 s	3.9×	36.9 s	2.4×
		2	19.7 s	**4.4**×	16.0 s	**5.4**×	20.5 s	**4.2**×
ResNet-18 (Tiny)	357.5 s	0	175.7 s	2.0×	325.1 s	1.1×	400.0 s	0.9×
		1	84.6 s	4.2×	63.6 s	5.6×	109.7 s	3.3×
		2	55.3 s	**6.5**×	48.2 s	**7.4**×	59.7 s	**6.0**×
ResNet-34 (ImageNet)	2116.9 s	0	1180.9 s	1.9×	2145.8 s	1.0×	2840.2 s	0.8×
		1	638.9 s	3.5×	443.9 s	5.0×	632.4 s	3.5×
		2	337.4 s	**6.6**×	307.7 s	**7.2**×	378.9 s	**5.9**×

Table 6: The E2E latencies of Safhire over Concrete-ML on multi-threaded execution and associated speedups, for different trace extraction levels and precisions. Both solutions are run using 32 threads. To compute the duration of network communication in Safhire, we assume a conservative network speed of 1.25 MB/s. For each model and precision, we mark the largest speedup in bold.

To put 32-thread results into perspective, we provide a direct comparison against hybrid model inference with Concrete-ML in Table 6. We focus this comparison on 32-thread execution, as Concrete-ML is a natively multithreaded implementation and is neither designed nor optimized for single-threaded operation. Furthermore, due to the implementation of Concrete-ML, we were unable to obtain a credible and accurate estimate of its server-client communication overhead without making a substantial changes to their library. Therefore, for Concrete-ML we report total inference including computations of server and client together with communication in-between them done over very fast loopback network interface. On other hand, for Safhire we report results with simulated network speed of modest 1.25 MB/s giving Concrete-ML substantial advantage in communication speed. Despite this, we observe that Safhire with extraction optimization ($y > 0$) achieves between 2.4× and 7.4× speedup over Concrete-ML in the same settings.

Moreover, multithreading provides substantial speedup over ORION: up to 54.0× on CIFAR-10, up to 26.0× on Tiny, and up to 35.2× on ImageNet. Higher extraction levels benefit from the highest relative speedup. For example, on CIFAR-10, the multithreading speedup grows from 2.63× ($\gamma = 0$) to 4.10× ($\gamma = 2$), on Tiny 2.33× ($\gamma = 0$) to 7.29× ($\gamma = 2$), and finally on ImageNet 2.91× ($\gamma = 0$) to 5.21× ($\gamma = 2$). Precision plays a lesser role, though higher b yields slightly larger gains (*e.g.*, 4.09× for $b = 8$ vs. 5.93× for $b = 16$ on CIFAR-10).

Scaling up to 16 threads delivers the largest improvements; beyond that, benefits taper off (<5% difference between 16 and 32 threads), with occasional slowdowns arising likely from the overheads of thread management. Furthermore, Fig. 8 breaks down how the execution time of individual operations scales with an increasing number of threads, using ResNet-18 with $b = 16$ on the Tiny dataset. These measurements reveal that every operation benefits from multithreading. In this configuration, we observe up to 14.10× speedup for linear operations and between 2.31× and 4.09× speedup for packing, compared to a single-threaded execution. Overall, SAFHIRE benefits strongly from multithreading, especially at $\gamma = 1$ and $\gamma = 2$, while gains beyond 16 threads are marginal.

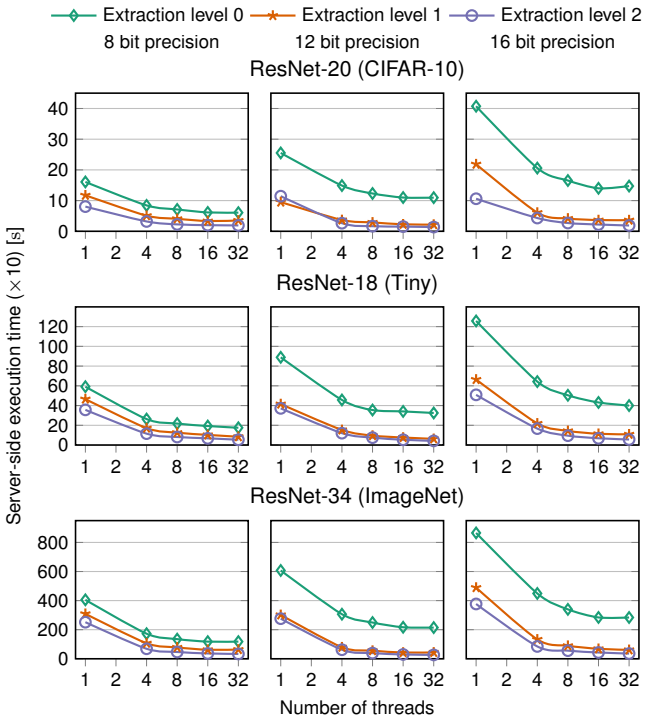

Figure 7: The breakdown of server-side execution time of SAFHIRE in function of the number of utilized threads.

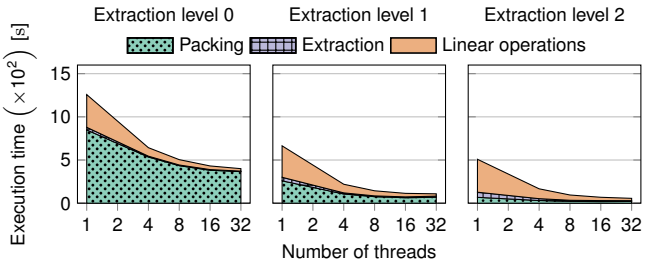

Figure 8: Breakdown of the duration of individual server operations when increasing the number of threads, for ResNet-18 with the Tiny dataset, while fixing $b = 16$.

6.6 Accelerating Safhire with a GPU

Finally, we enhance the Safhire implementation and offload compute-intensive operations such as key switches, packing and partial extraction to the GPU. Fig. 9 breaks down the execution time for different extraction levels γ, quantization levels b and when using the CIFAR-10 dataset with a ResNet-20 model, when running Safhire on an A100 GPU. We remark that the communication volume overhead is similar to the one shown in Fig. 6 (top). For $\gamma = 0$ and $b = 8$, a single inference request takes 11.29 s of compute time using a GPU which is a speedup of 14.4× and compared to a single-threaded CPU setting which takes 162.57 s. In line with other experiments, we observe that the execution time reduces as γ and b increase, at the cost of additional communication volume. For all configurations of γ and b, we notice that the packing operation requires the most server-side compute time, *e.g.*, for $\gamma = 0$ and $b = 16$ the packing operation takes 96.1% of all compute time. Additional GPU-specific optimizations, such as custom kernels, could further reduce this cost. In summary, Safhire benefits substantially from GPU acceleration, making GPUs a promising path for practical FHE inference.

7 Related Work

FHE based neural-network inference has progressed from early proofs of concept to GPU-accelerated systems at ImageNet scale. We group prior work into five strands and position our RLWE-based hybrid within this landscape.

Leveled FHE and Compiler Tool-Chains. CryptoNets demonstrated end-to-end neural network inference on encrypted MNIST using leveled homomorphic encryption, replacing non-linearities with the square function as the activation (Gilad-Bachrach et al., 2016). However, CryptoNets incurred high latency. Follow-up work addressed both accuracy and efficiency: CryptoDL achieved higher accuracy by retraining networks with carefully chosen low-degree polynomial approximations of common activations (Hesamifard et al., 2016), while LoLa reduced latency substantially through optimized data layout and alternating ciphertext representations (Brutzkus et al., 2019). Compiler stacks such as CHET and EVA further automated parameter selection, layout, and (where applicable) bootstrapping for CKKS, improving developer productivity while matching or exceeding hand-tuned baselines (Dathathri et al., 2019; 2020). These systems remain fully encrypted and therefore pay for polynomial activations and/or bootstrapping.

Interactive and Hybrid Cryptographic Protocols. To reduce online latency, several works combine HE with secure two-party computation. Gazelle evaluates linear layers under CKKS and non-linearities via garbled circuits, achieving sub-second online times on small CNNs but requiring multiple interactive rounds and a semi-honest two-party model (Juvekar et al., 2018). Delphi refines this MPC/HE split by front-loading rotation/packing costs in a preprocessing phase to further shrink the online path (Mishra et al., 2020). More recently, Shechi introduced the first multiparty homomorphic encryption (MHE) compiler, automatically translating Python code into secure distributed computation that combines homomorphic encryption with multiparty computation (Smajlović et al., 2025). Shechi focuses on distributed analytics such as PCA and genomic workloads, showing up to 15× runtime improvements over prior secure frameworks and highlighting the promise of compiler-based optimization for hybrid cryptographic protocols. Our approach also splits work across cryptographic boundaries, but is *non-interactive*: non-linearities execute locally on the client, avoiding online MPC while still protecting server-side parameters.

Scalable Fully Encrypted Frameworks. Recent frameworks push fully encrypted inference to deeper networks. HyPHEN introduces GPU-friendly kernels (RAConv/CAConv) and weight-reuse techniques to bring single-GPU ResNet-18/ImageNet latency to the tens-of-seconds regime (Kim et al., 2024). Orion adds single-shot multiplex packing and automated bootstrap placement, outperforming earlier CKKS baselines on ResNet-20 and demonstrating the first FHE-based YOLO inference under CKKS (Ebel et al., 2025). EncryptedLLM shows that small GPT-style models can run end-to-end under FHE with large GPU/CPU speedups (de Castro et al., 2025). All these systems remain fully homomorphic and thus still pay for bootstrapping or high-degree polynomial activations.

Client–Server Hybrid Execution. Zama's Concrete-ML library exposes hybrid models as a developer feature, allowing certain layers to run on the client in plaintext and the rest under encryption on the

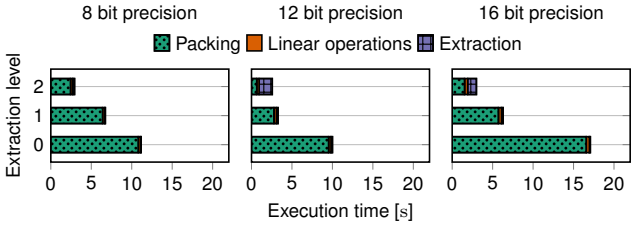

Figure 9: The breakdown of server-side execution time on A100 GPU for a single sample inference using a ResNet-20 model on CIFAR-10 dataset.

server (Zama AI, 2024). However, it remains at the level of developer tooling rather than a systematically studied design point. In contrast, our work formalizes the hybrid setting on an RLWE backend, introduces partial-trace packing and other optimizations to reduce key-switch pressure, and provides a systematic evaluation against state-of-the-art FHE framework on diverse neural models and datasets. Moreover, our work explicitly analyzes and mitigates model leakage via randomized shuffling with differential-privacy amplification, dimensions which are unexplored in existing hybrid toolkits.

Quantization and Low-Precision Accumulators. Quantization is orthogonal yet complementary to HE. WrapNet adds overflow-penalty regularization and cyclic activations to maintain accuracy with very low-precision arithmetic (Ni et al., 2021). A2Q/A2Q+ constrain weight norms during training to avoid accumulator overflow, recovering near-baseline accuracy with 14–16-bit accumulators on ImageNet-scale networks (Colbert et al., 2023; 2024). We leverage these insights by selecting 8/12/16-bit accumulator widths compatible with our discrete-torus plaintext space.

Bootstrapping Advances in TFHE. Programmable bootstrapping over torus has seen large constant-factor improvements and enabled early CNN experiments under gate-by-gate evaluation (Chillotti et al., 2018; 2016). Our design circumvents bootstrapping entirely, leveraging client-side computation and trading extra network round-trip for substantial speed-ups.

8 Conclusion

We presented Safhire, a practical hybrid FHE inference framework that eliminates bootstrapping by offloading non-linear operations to the client while keeping linear layers encrypted on the server. This design drastically reduces computation cost, avoids approximation of non-linearities, and ensures model confidentiality through shuffling. Our evaluation on standard CNNs shows that Safhire significantly lowers E2E latency compared to Orion, a state-of-the-art FHE inference system. Overall, Safhire demonstrates that our hybrid approach can make privacy-preserving ML inference both efficient and practical.

References

Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. Cryptology ePrint Archive, Paper 2015/046, 2015. URL `https://eprint.iacr.org/2015/046`.

Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim. The privacy blanket of the shuffle model. In *Annual International Cryptology Conference*, pp. 638–667. Springer, 2019.

Sayan Biswas, Kangsoo Jung, and Catuscia Palamidessi. Tight differential privacy guarantees for the shuffle model with k-randomized response. In *International Symposium on Foundations and Practice of Security*, pp. 440–458. Springer, 2023.

Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th symposium on operating systems principles*, pp. 441–459, 2017.

Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 587–617. Springer, 2021.

Alon Brutzkus, Ran Gilad-Bachrach, and Oren Elisha. Low latency privacy preserving inference. In *International Conference on Machine Learning*, pp. 812–821. PMLR, 2019.

Philippe Chartier, Michel Koskas, and Mohammed Lemou. Exploring general cyclotomic rings in torus-based fully homomorphic encryption: Part i - prime power instances. Cryptology ePrint Archive, Paper 2025/488, 2025. URL `https://eprint.iacr.org/2025/488`.

Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International conference on the theory and application of cryptology and information security*, pp. 409–437. Springer, 2017.

Albert Cheu, Adam Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. Distributed differential privacy via shuffling. In *Annual international conference on the theory and applications of cryptographic techniques*, pp. 375–403. Springer, 2019.

Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology – ASIACRYPT 2016*, volume 10031 of *Lecture Notes in Computer Science*, pp. 3–33. Springer, 2016. doi: 10.1007/978-3-662-53887-6_1.

Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. Cryptology ePrint Archive, Paper 2018/421, 2018. URL `https://eprint.iacr.org/2018/421`.

Ian Colbert, Alessandro Pappalardo, and Jakoba Petri-Koenig. A2q: Accumulator-aware quantization with guaranteed overflow avoidance. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 16989–16998, 2023.

Ian Colbert, Alessandro Pappalardo, Jakoba Petri-Koenig, and Yaman Umuroglu. A2q+: Improving accumulator-aware weight quantization. In *Forty-first International Conference on Machine Learning*, 2024. URL `https://openreview.net/forum?id=mbx2pLK5Eq`.

Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. Chet: An optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pp. 142–156. ACM, 2019. doi: 10.1145/3314221.3314628. URL `https://dl.acm.org/doi/10.1145/3314221.3314628`.

Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine, and Madanlal Musuvathi. Eva: An encrypted vector arithmetic language and compiler for efficient homomorphic computation. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pp. 546–561. ACM, 2020. doi: 10.1145/3385412.3386023. URL `https://dl.acm.org/doi/10.1145/3385412.3386023`.

Ingrid Daubechies, Ronald DeVore, Simon Foucart, Boris Hanin, and Guergana Petrova. Nonlinear approximation and (deep) relu networks. *Constructive Approximation*, 55(1):127–172, 2022.

Leo de Castro, Daniel Escudero, Adya Agrawal, Antigoni Polychroniadou, and Manuela Veloso. EncryptedLLM: Privacy-preserving large language model inference via GPU-accelerated fully homomorphic encryption. In *Forty-second International Conference on Machine Learning*, 2025. URL `https://openreview.net/forum?id=PGNff6H1TV`.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Matthew F Dixon, Igor Halperin, Paul Bilokon, et al. *Machine learning in finance*, volume 1170. Springer, 2020.

Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and trends® in theoretical computer science*, 9(3-4):211–487, 2014.

Austin Ebel, Karthik Garimella, and Brandon Reagen. Orion: A fully homomorphic encryption framework for deep learning. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '25, pp. 734–749, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400710797. doi: 10.1145/3676641.3716008. URL https://doi.org/10.1145/3676641.3716008.

Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 2468–2479. SIAM, 2019.

Vitaly Feldman, Audra McMillan, and Kunal Talwar. Hiding among the clones: A simple and nearly optimal analysis of privacy amplification by shuffling. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 954–964. IEEE, 2022.

Vitaly Feldman, Audra McMillan, and Kunal Talwar. Stronger privacy amplification by shuffling for rényi and approximate differential privacy. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 4966–4981. SIAM, 2023.

Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pp. 169–178, 2009.

Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 201–210, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/gilad-bachrach16.html.

Robert M Gray. Toeplitz and circulant matrices: A review. *Foundations and Trends® in Communications and Information Theory*, 2(3):155–239, 2006.

Hafsa Habehh and Suril Gohel. Machine learning in healthcare. *Current genomics*, 22(4):291–300, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Cryptodl: towards deep learning over encrypted data. In *Annual Computer Security Applications Conference (ACSAC 2016), Los Angeles, California, USA*, volume 11, pp. 1–2, 2016.

Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca N Wright. Privacy-preserving machine learning as a service. *Proceedings on Privacy Enhancing Technologies*, 2018.

Fatima Hussain, Rasheed Hussain, Syed Ali Hassan, and Ekram Hossain. Machine learning in iot security: Current solutions and future challenges. *IEEE Communications Surveys & Tutorials*, 22(3):1686–1721, 2020.

Takumi Ishiyama, Takuya Suzuki, and Hayato Yamana. Highly accurate cnn inference using approximate activation functions over homomorphic encryption. In *2020 IEEE International Conference on Big Data (Big Data)*, pp. 3989–3995. IEEE, 2020.

Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha P. Chandrakasan. Gazelle: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1651–1669. USENIX Association, 2018. URL https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar.

Donghwan Kim, Jaiyoung Park, Jongmin Kim, Sangpyo Kim, and Jung Ho Ahn. Hyphen: A hybrid packing method and its optimizations for homomorphic encryption-based neural networks, 2024. ISSN 2169-3536. URL http://dx.doi.org/10.1109/ACCESS.2023.3348170.

Antti Koskela, Mikko A Heikkilä, and Antti Honkela. Numerical accounting in the shuffle model of differential privacy. *Transactions on machine learning research*, 2023.

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

Yann Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, et al. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *iEEE Access*, 10:30039–30054, 2022.

Chiara Marcolla, Victor Sucasas, Marc Manzano, Riccardo Bassoli, Frank HP Fitzek, and Najwa Aaraj. Survey on fully homomorphic encryption, theory, and applications. *Proceedings of the IEEE*, 110(10): 1572–1609, 2022.

Fatemehsadat Mireshghallah, Mohammadkazem Taram, Prakash Ramrakhyani, Ali Jalali, Dean Tullsen, and Hadi Esmaeilzadeh. Shredder: Learning noise distributions to protect inference privacy. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 3–18, 2020.

Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 2505–2522. USENIX Association, 2020. URL https://www.usenix.org/conference/usenixsecurity20/presentation/mishra.

Kevin Nam, Hyunyoung Oh, Hyungon Moon, and Yunheung Paek. Accelerating n-bit operations over tfhe on commodity cpu-fpga. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, 2022.

Renkun Ni, Hong min Chu, Oscar Castaneda, Ping yeh Chiang, Christoph Studer, and Tom Goldstein. Wrapnet: Neural net inference with ultra-low-precision arithmetic. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=3SqrRe8FWQ-.

Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

Haris Smajlović, David Froelicher, Ariya Shajii, Bonnie Berger, Hyunghoon Cho, and Ibrahim Numanagić. Shechi: A secure distributed computation compiler based on multiparty homomorphic encryption. In *Proceedings of the 34th USENIX Security Symposium (USENIX Security 25)*. USENIX Association, 2025. URL https://www.usenix.org/conference/usenixsecurity25/presentation/smajlovic.

Zama AI. Concrete-ml documentation: Hybrid models (client/server). https://docs.zama.ai/concrete-ml/guides/hybrid-models, 2024. Accessed 2025-08-24.

Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. {MArk}: Exploiting cloud services for {Cost-Effective},{SLO-Aware} machine learning inference serving. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pp. 1049–1062, 2019.

Peng Zhang, Dongyan Qiu, Ao Duan, and Hongwei Liu. Efficient homomorphic approximation of max pooling for privacy-preserving deep learning. In *International Conference on Machine Learning for Cyber Security*, pp. 70–80. Springer, 2024.

A Table of Notation

We list in Table 7 the key notations used in this paper.

Param.	**Description**
μ	A cleartext message
$\mathrm{LWE}_s(\mu)$	LWE encryption of μ with key s
$\mathrm{RLWE}_{s(X)}(\mu(X))$	RLWE encryption of polynomial $\mu(X)$ with key s(X)
M	Cyclotomic polynomial index; size of RLWE key
N	$\phi(M)$ s.t. ϕ is Euler's totient function
α	Exponent in $M = t^\alpha$
t	Prime base in $M = t^\alpha$
Δ	Std. dev. of Gaussian encryption noise
p	Size of cleartext message
D	Decomposition base
l	Depth of the decomposition
r	Round number in Safhire
β	Packing level
γ	Partial trace extraction level
b	The bit width of model accumulator
θ	Model held by the server
I	Input image by the client
H	Height of input image I
W	Width of input image I
C_{in}	Channels of input image I
L	Number of layers in θ
η	Re-quantization scale

Table 7: Relevant RLWE parameters

B Proof of Lem. 1

Let $P \in \mathcal{K}$ and $0 \leq i \leq N-1$. Using Equations (23) and (24) from Chartier et al. (2025), for all $Q \in \mathcal{K}$, we obtain:

$$\mathrm{Tr}_{\mathcal{K}_1/\mathcal{K}_0}(Q(X)) = \sum_{i=1}^{t-1} Q(X^i) \tag{1}$$

$$\mathrm{Tr}_{\mathcal{K}_2/\mathcal{K}_1}(Q(X)) = \sum_{i=0}^{t-1} Q(X^{it+1}) \tag{2}$$

Substituting $Q(X)$ for $\overline{\Omega}_i^*(X)P(X)$ in equation 1 yields the first result, and composing this with equation 2 yields the second. □

C Proof of Th. 2

Let, in any given round, the convolution outputs satisfy $(\varepsilon_0, \delta_0)$-local DP. Then, by the very design of Safhire, the convolution outputs of each layer are shuffled by the server, independently in each round, before being sent to the client. Therefore, the result follows from Theorem 3.8 of Feldman et al. (2022). □

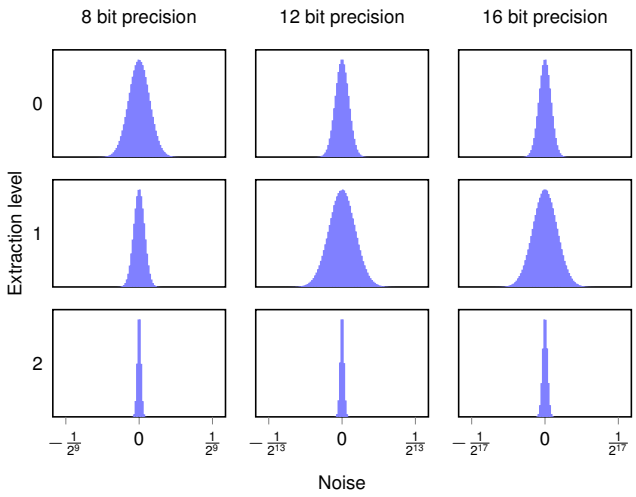

Figure 10: The distribution of noise from the fast packing operation for varying precision and extraction levels.

D Implementation

D.1 FHE Noise in Safhire

Fig. 10 shows the noise distribution induced by fast packing on the outputs of the first layer of ResNet-20 when applied to the CIFAR-10 dataset, for different extraction levels γ and precision levels b. As long as the noise falls within the indicated boundaries, the decryption is correct. For the given combinations of γ and b, we observe that the noise level remains within the allotted boundaries. While we cannot formally exclude pathological cases, our empirical evidence strongly suggests that decryption failure does not occur in practice for our parameter choices.