PrivSpike: A Privacy-Preserving Inference Framework for Deep Spiking Neural Networks using Homomorphic Encryption

Nges Brian Njungle[1], Erich Jahns[1], Milan Stojkov[2], and Michel A. Kinsy[1]

[1] STAM Center, Arizona State University, 85281 - USA
[2] Faculty of Technical Sciences, University of Novi Sad - Serbia
{nnjungle, jjahns, mkinsy}@asu.edu, stojkovm@uns.ac.rs

Abstract. Deep learning has become a cornerstone of modern machine learning. It relies heavily on vast datasets and significant computational resources for high performance. This data often contains sensitive information, making privacy a major concern in deep learning. Spiking Neural Networks (SNNs) have emerged as an energy-efficient alternative to conventional deep learning approaches. Nevertheless, SNNs still depend on large volumes of data, inheriting all the privacy challenges of deep learning. Homomorphic encryption addresses this challenge by allowing computations to be performed on encrypted data, ensuring data confidentiality throughout the entire deep learning pipeline.
In this paper, we introduce PrivSpike, a privacy-preserving inference framework for SNNs using the CKKS homomorphic encryption scheme. PrivSpike supports deep SNNs and introduces two key algorithms for evaluating the Leaky Integrate-and-Fire activation function: (1) a polynomial approximation algorithm designed for high-performance SNN inference, and (2) a novel scheme-switching algorithm that optimizes precision at a higher computational cost. We evaluate PrivSpike on MNIST, CIFAR-10, Neuromorphic MNIST, and CIFAR-10 DVS using LeNet-5 and ResNet-19 architectures, achieving encrypted inference accuracies of 98.10%, 79.3%, 98.1%, and 66.0%, respectively. On a consumer-grade CPU, LeNet-5 achieved inference times of 28 seconds on MNIST and 212 seconds on Neuromorphic MNIST. For ResNet-19, inference took 784 seconds on CIFAR-10 and 1,846 seconds on CIFAR-10 DVS. These results establish PrivSpike as a viable and efficient solution for secure SNN inference, bridging the gap between energy-efficient deep neural networks and strong cryptographic privacy guarantees while outperforming prior encrypted SNN solutions.

Keywords: Privacy-preserving Machine Learning · Spiking Neural Networks · Fully Homomorphic Encryption · CKKS · OpenFHE

1 Introduction

In recent years, machine learning has revolutionized various industries by driving significant technological advancements. Its applications span diverse fields,

including medical diagnosis, traffic prediction, image recognition, speech recognition, product recommendation, self-driving cars, virtual personal assistants, and natural language processing [1]. Central to these advancements are deep neural networks, particularly convolutional neural networks (CNNs), which leverage large-scale datasets to achieve high precision, especially in domain-specific tasks [2]. Despite their effectiveness, CNNs require substantial energy in both training and inference. These limitations have sparked great interest in Spiking Neural Networks (SNNs) as a promising alternative [3].

Unlike conventional deep neural network approaches that synchronously process static frames, SNNs operate on asynchronous, event-driven principles that more closely mimic biological neural systems. This key distinction enables SNNs to process spatiotemporal information with remarkable energy efficiency and responsiveness [4]. Utilizing these dynamics, Lenz et al. [5] demonstrated that SNNs deployed on Intel's Loihi 2 neuromorphic hardware [6] can reduce the energy consumption of standard CNN inference by up to 246× on MNIST. These results underscore the potential of SNNs in low-power computing scenarios where traditional deep learning models are impractical. Another major application domain of SNNs is in event-based data processing. SNNs have shown considerable promise in processing time-sensitive and event-based biomedical and sensory data, where traditional machine learning approaches fall short. This is because the temporal dynamics of these datasets naturally fit into the event-driven data processing mechanisms of SNNs. For instance, Xu et al. [7] demonstrated superior electromyographic pattern recognition ability using SNNs, outperforming cyclic CNNs of the same architecture by up to 50%. Sola et al. [8] also reported state-of-the-art results on gesture recognition from an event-based dataset using SNNs with up to 25% better performance over recurrent neural networks. These examples highlight the superiority of SNNs in handling data with intrinsic temporal structure, making them ideal for applications in neuromorphic computing and energy-constrained settings. However, similar to other deep neural network approaches, SNNs also require significant data to train, thus inheriting the critical data and model privacy issues of deep neural networks. These concerns are particularly apparent when models are inferred on highly sensitive private information like biomedical and sensory data.

Although privacy-preserving techniques such as homomorphic encryption [9], multi-party computation [10], differential privacy [11], and trusted execution environments [12] have been extensively explored for privacy-preserving computations in CNNs, little attention has been paid to data and model privacy issues in SNNs [13]. Among these techniques, homomorphic encryption (HE) has emerged as the most suitable for privacy-preserving neural networks due to its strong cryptographic security guarantees and efficient communication properties despite the higher computational overhead it brings. It enables computations directly on encrypted data, thus ensuring data confidentiality throughout the computational pipeline. Among HE schemes, the Cheon-Kim-Kim-Song (CKKS) scheme is particularly well suited for neural network applications due to its support for approximate arithmetic and parallel data processing [14]. Despite

its advantages, utilizing CKKS for SNNs presents distinct challenges. Firstly, the efficient evaluation of non-linear functions, such as the leaky integrate-and-fire (LIF) activation function, is difficult because the scheme does not inherently support non-linear operations. Secondly, SNNs encode data using temporal and spatial dynamics into multiple time-steps, further increasing the computational and memory requirements of encrypted inference. Previous works applying HE to SNNs, such as [15], have attempted to address these issues by proposing distinct privacy-preserving SNN inference solutions; however, all previous solutions have fallen short in terms of performance and scope of evaluation.

In this work, we introduce PrivSpike, a comprehensive and efficient framework for privacy-preserving SNN inference using the CKKS scheme. PrivSpike tackles critical challenges in privacy-preserving SNN inference by introducing two novel algorithms for the widely-used LIF activation function, alongside optimized implementations of all SNN layers. These two LIF algorithms offer a trade-off between performance and accuracy of models. The first algorithm employs polynomial approximation to handle the non-linear components of the LIF, while the second utilizes a scheme-switching technique to evaluate spikes of encrypted inputs under the TFHE scheme securely [16]. To the best of our knowledge, this is the first work to propose such algorithms for encrypted-domain LIF evaluation. Additionally, we incorporate architectural optimizations that significantly reduce the computational overhead and memory requirements of evaluating encrypted data on SNNs. We evaluated our work using the MNIST, CIFAR-10, Neuromorphic MNIST (N-MNIST), and CIFAR-10 DVS datasets. While MNIST and CIFAR-10 are popular deep neural network datasets, N-MNIST and CIFAR-10 DVS are popular neuromorphic datasets mainly used with SNNs. We use SNN versions of the widely adopted LeNet-5 and ResNet-19 architectures to analyze these datasets. Our evaluation demonstrates that PrivSpike scales efficiently, with models of identical architecture using almost the same amount of memory, independent of the number of time steps. This consistent resource utilization simplifies deployment and enhances scalability across different model types and datasets. In terms of inference latency, we observe that the inference time of SNNs grows linearly with the increase in the number of time steps, under the same HE security parameters. This increase in latency is attributed to the temporal dynamics inherent to SNNs, which involve iterative updates over multiple time steps. Nonetheless, PrivSpike 's efficient scaling makes this additional cost manageable. Furthermore, the accuracy of privacy-preserving SNN models closely matches that of their plaintext counterparts, demonstrating the practicality of our implementations and methods. Notably, our scheme-switching LIF algorithm maintained a higher precision, with encrypted inference results deviating by 0.8% from plaintext outputs in the LeNet-5 model using MNIST. On the LeNet-5 architecture, shows about 34× and 50× better inference time compared to related works. In this work, we make the following concrete contributions:

– We propose PrivSpike, an efficient, open-source, privacy-preserving SNN inference framework using the CKKS homomorphic encryption scheme.

- We introduce and implement two algorithms for the LIF activation function tailored for high performance and precision in encrypted SNN computations. The first algorithm utilizes Chebyshev polynomials to approximate the non-linear component of the LIF activation function, while the second employs the scheme-switching technique to compute high-precision results of this component using the TFHE scheme.
- We evaluate PrivSpike using four widely studied datasets and two well-established SNN architectures using eight SNN models. Specifically, we conduct experiments on the MNIST and CIFAR-10 datasets, as well as on the N-MNIST and CIFAR-10 DVS datasets. The N-MNIST and CIFAR-10 DVS ensure the relevance of our work on neuromorphic and event-based data processing, while the MNIST and CIFAR-10 datasets show the compatibility of PrivSpike with traditional ML workloads.

2 Related Works

In 2022 and 2023, Casaburi and Nikfam et al. introduced the first HE framework for SNNs [15], [17]. Their approach employed the Brakerski/Fan-Vercauteren (BFV) HE scheme [18] to perform encrypted inference on SNNs. Experiments were conducted on the FashionMNIST dataset using LeNet-5 and AlexNet architectures. The framework required approximately 930 seconds to infer a single encrypted image using the SNN LeNet-5 model. For the SNN AlexNet model, the authors estimated an inference time of 901,800 seconds per encrypted image. Despite the high latency, the approach achieved a notable accuracy of 96.5% for encrypted inference on the LeNet-5 model. The framework also introduced a significant communication overhead due to the computation of the LIF activation functions on the client side. Their setup requires the server to transfer encrypted data to the client for the evaluation of activation functions. The client decrypts the data, computes the activation function in plaintext, re-encrypts the results, and sends them back to the server to continue the inference process. The authors employed a Tesla P100-PCIE GPU, an Intel(R) Xeon(R) Gold 6134 @ 3.20GHz CPU, and 100GB of RAM. Although they showcased the feasibility of privacy-preserving SNNs, the framework remains inadequate for real-world adoption due to the substantial computational resources required, the communication overhead it introduces, and the high inference latency of the models.

Still in 2023, Li et al. introduced FHE-DiCSNN [19]. Their work leveraged the discrete nature of the Fully Homomorphic Encryption over the Torus (TFHE) scheme [16] to perform privacy-preserving inference on SNNs. They chose the TFHE scheme because it naturally aligns with the discrete spike characteristic of SNNs. Their work was evaluated on the MNIST dataset and achieved an accuracy of 95.10% on encrypted data. FHE-DiCSNN took 0.75 seconds to infer a single encrypted MNIST image. The primary limitation of FHE-DiCSNN is that its evaluation is done on a small network comprising only 30 neurons. In practical SNN research, LeNet-5 is the most commonly used shallow architecture, consisting of roughly 60,000 neurons. Hence, approximating the total inference

time of the framework on an SNN LeNet-5 model gives an inference time of 1,500 seconds per encrypted image. The TFHE scheme aligns naturally with SNNs, but it is significantly slower in deep learning tasks compared to schemes like CKKS. This limitation arises from its lack of support for parallel data processing, a capability widely leveraged to accelerate computations on encrypted data [14].

In contrast to the aforementioned works, our approach utilizes the CKKS scheme to provide privacy-preserving inference in SNNs. We leverage optimizations, such as parallel data computation for high performance evaluation of all linear layers. We propose two distinct algorithms for evaluating the LIF activation function in the encrypted domain. The first algorithm employs the Chebyshev polynomial approximation of the LIF function. The second algorithm utilizes scheme switching, allowing for high precision evaluation of the non-linear component of the LIF in TFHE. This hybrid approach leverages the strengths of both schemes, thus using CKKS for efficient linear computation and TFHE for high-precision non-linear evaluation. We infer our models on encrypted data, and notably, our SNN LeNet-5 model, which employs the approximation method for LIF, takes just 28 seconds to infer an encrypted MNIST image on a consumer CPU. Additionally, we evaluated SNN models based on the ResNet-19 architecture, further demonstrating great performance, scalability, and efficient resource utilization for encrypted data processing within PrivSpike. This comprehensive evaluation with models from widely used SNN architectures and diverse datasets (standard and event-based) highlights the practical applications of PrivSpike.

3 Background

3.1 Homomorphic Encryption

Homomorphic Encryption (HE) is an advanced cryptographic protocol that allows computation on encrypted data. The term "homomorphic" also means "same form". In mathematical terms, objects have the same form in plaintext as in ciphertext if there exist equivalent mathematical operations in both the plaintext and ciphertext domains that produce corresponding outputs in their respective domains. Since the demonstration of Fully Homomorphic Encryption (FHE) by Gentry [20], the field has seen significant research breakthroughs and the introduction of numerous FHE schemes [21]. CKKS [14], BFV [18], BGV [22], and TFHE [16] make up state-of-the-art HE schemes, each optimized for different application scenarios. For instance, TFHE processes bitwise circuits, while BFV and BGV handle integer computations. In contrast, CKKS is designed for floating-point arithmetic computations, making it well-suited for this work.

Cheon-Kim-Kim-Song (CKKS) scheme This scheme was released in 2016 by Cheon, Kim, Kim, and Song, and it is the only mainstream HE scheme today that operates on Approximate Arithmetic Numbers [14]. The scheme anchors its security on the security assumptions of the non-probabilistic hard problem of the ring version of the Learning with Errors problem [23]. Mathematically, let $\mathbb{Z}$,

$\mathbb{R}$, and $\mathbb{C}$ represent Integers, Real numbers, and Complex numbers, respectively. If given a vector $b \in \mathbb{Z}_q^m$ and matrix $A \in \mathbb{Z}_q^{m \times n}$, the standard LWE problem is looking for an unknown vector $s \in \mathbb{Z}_q^n$ such that:

$$As + e = b \; mod \; q \tag{1}$$

where e is an error vector introduced from random samples through an error distribution, and q is a large prime number.

The Ring Learning with Errors (RLWE) problem extends the LWE problem to polynomial rings over finite fields. This ring structure enables CKKS efficient parallel data representation and processing. The primary homomorphic operations defined in the CKKS scheme include addition, multiplication, rotation, and bootstrapping. Bootstrapping allows for the refreshing of noise in the ciphertext. The Rotation operation shifts the data encoded in the ciphertext to the right or left, while multiplication and addition allow for the multiplication and addition of ciphertext messages, respectively. In 2018, Cheon et al. released an optimized version of CKKS on the residue number system [24] that transforms the CKKS ciphertext to Double Chinese-Remainder-Transform polynomials, allowing for fast multiplication using the Number Theoretical Transform (NTT), which reduces the complexity of multiplications from $O(n^2)$ in basic modular polynomial multiplication to $O(n \log n)$.

A key feature of CKKS is its ability to perform ciphertext slot packing, wherein multiple pieces of data are encoded into the coefficients of a single large polynomial. This packing mechanism facilitates a Single Instruction Multiple Data (SIMD) computational paradigm, where a single homomorphic operation can process multiple values in parallel. This approach significantly accelerates the runtime of CKKS by enabling the simultaneous evaluation of multiple data points, making the scheme highly efficient for applications that require large amounts of data through batching. Let $R = \mathbb{Z}[X]/(X^N + 1)$ represent a ring of polynomials modulo $X^N + 1$, where N is a power of 2. The CKKS scheme converts a message $v \in \mathbb{C}^{N/2}$ to R. Using the SIMD approach, homomorphic operations can be performed in parallel across all encoded slots, effectively processing multiple data elements in the ciphertexts simultaneously, thus reducing the cost of operations in such settings. However, at most half the slots in a ciphertext can be independently used due to the need to balance complex conjugate pairs in the ciphertext encoding process. This constraint arises because CKKS encodes plaintext values in the complex domain, and thus, decryption requires the conjugate symmetry of the encoded coefficients. The use of approximate arithmetic in CKKS, combined with its support for the operations above and SIMD parallel processing capabilities, makes it particularly well-suited for large-scale data processing tasks, such as those involved in SNNs.

Open-source HE Libraries, such as Microsoft SEAL [25], HElib [26], TFHE-rs [27], and OpenFHE [28], provide diverse implementations of HE schemes [29]. These libraries facilitate the adoption of HE by abstracting the complexities of developing HE schemes. Among the available CKKS implementations, OpenFHE offers the most advanced support for the scheme. Its implementation supports

optimizations such as SIMD, bootstrapping, scheme switching to TFHE, and efficient rotation strategies, which is why we chose it for this work.

3.2 Spiking Neural Networks (SNNs)

SNNs are a biologically inspired class of artificial neural networks that process information through discrete spike events, closely mimicking the communication mechanisms of neurons in the brain [30]. Unlike traditional neural networks, which utilize continuous-valued activations, SNNs operate over multiple discrete time steps, dynamically and sparsely processing input data [31]. This temporal and event-driven paradigm allows SNNs to inherently encode and process time-varying information, making them particularly energy-efficient and well-suited for neuromorphic hardware such as Intel Loihi, IBM TrueNorth, and BrainScaleS [32]. The core computational units of SNNs are spiking neurons, which simulate the dynamics of biological neurons. For a neuron i, the membrane potential $V_i(t)$ evolves (t) in response to incoming spikes, and a spike is emitted when $V_i(t)$ crosses a predefined threshold. This spike resets the membrane potential, introducing temporal dynamics to the computation. These dynamics enable SNNs to process and encode spatiotemporal patterns natively [33]. At each time step, only a subset of neurons and synapses are active, resulting in a significant reduction in computational overhead compared to traditional networks that process all features simultaneously. This property aligns well with hardware constraints in low-power and real-time scenarios such as event-based cameras.

Figure 1 illustrates the structure and temporal operation of an SNN. Input data is encoded as spike events and processed over multiple discrete timesteps. As spikes propagate through the network, linear layers perform spatial transformations, and spiking neurons integrate inputs over time to extract temporal features, maintaining a membrane potential that enables them to retain information between time steps. The network's output layer aggregates spiking activity across all time-steps to decode a final prediction, effectively capturing the dynamic nature of time-dependent data.

4 Secure Spiking Neural Network Layers

This section presents a detailed overview of the layers and processes involved in the design and development of SNNs. For each layer, we describe how it is adapted for encrypted computation within the PrivSpike framework.

4.1 Input Layer

The input layer of an SNN is the initial stage where raw input signals are processed and converted into spike-based representations suitable for subsequent computations [34]. The layer forms the foundation, ensuring that the temporal and spatial characteristics of inputs are preserved and appropriately formatted for processing through the network. As illustrated in Figure 1, the input layer

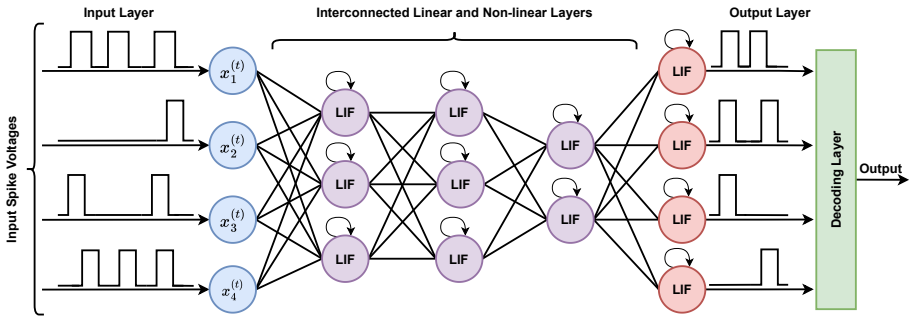

Fig. 1: Overview of a spiking neural network. Spikes are input across discrete time steps and propagated through linear and non-linear layers. Linear layers extract spatial features, while non-linear spiking dynamics capture temporal dependencies via recurrent processing. The output layer accumulates spikes over time and decodes them into a prediction.

accepts voltage signals corresponding to input spike voltages, which encode the temporal dynamics of the data. These input spike voltages are represented across multiple dimensions, denoted by indices i, which define the spatial and temporal resolution of the input [35], [36]. The layer converts these input signals into spiking activity, a process that generally involves SNN input encoding schemes such as rate coding [37] and temporal coding [38]. Its primary function is to bridge the gap between conventional input data and the event-driven nature of SNNs.

To avoid the high latency, computational cost, and performance degradation introduced by encoders, the first convolutional layer typically replaces a dedicated encoder in deep SNNs when processing static data, as it can perform feature extraction and spike transformation simultaneously. Unlike traditional encoders, which are explicitly designed to preprocess and convert input signals into spike trains, a convolutional layer can learn spatial and temporal patterns from raw input data while generating spike-based activations. This dual functionality eliminates the need for a separate encoding mechanism, streamlining the architecture and reducing computational overhead [39]. This approach is adopted in PrivSpike as it offers several advantages in SNN computations, especially when combined with HE and high-dimensional inputs. It simplifies the data preprocessing pipeline while maintaining flexibility and robustness by enabling the networks to adaptively encode and process data without relying on hand-crafted encoding rules. Additionally, this design reduces the number of time steps required to achieve the desired accuracy thus improve temporal efficiency.

4.2 Convolution Layer

In deep learning, the convolution operation is a transformation to extract features from input data [40]. The convolution layer applies a set of filters, called kernels, to the input data by sliding the filters over the data, applying element-wise multiplication, and summing the results to produce a single output [41], [42]. This process results in a feature map that highlights specific features the

filter detects. Mathematically, the convolution operation is defined as follows: Let the input tensor be denoted by X, with a shape of (d, h, w), where: d is the depth (or number of channels), h is the height, and w is the width of the input matrix. Let the kernel K have a shape of $(c_{\text{out}}, c_{\text{in}}, k_h, k_w)$, where: c_{out} is the number of output channels, c_{in} is the number of input channels, k_h is the height of the kernel, and k_w is the width of the kernel. The convolution operation computes the output feature map Y with a shape of $(c_{\text{out}}, h_{\text{out}}, w_{\text{out}})$, where h_{out} and w_{out} are the height and width of the output, respectively. The convolution operation can be expressed mathematically as:

$$Y_{c_{\text{out}},h_{\text{out}},w_{\text{out}}} = \sum_{c_{\text{in}}=1}^{c_{\text{in}}} \sum_{i=1}^{k_h} \sum_{j=1}^{k_w} X_{c_{\text{in}},h_{\text{out}}+i-1,w_{\text{out}}+j-1} \cdot K_{c_{\text{out}},c_{\text{in}},i,j} \tag{2}$$

$X_{c_{\text{in}},h_{\text{out}}+i-1,w_{\text{out}}+j-1}$ represents the element of the input tensor at the corresponding depth, height, and width location and $K_{c_{\text{out}},c_{\text{in}},i,j}$ represents the value of the kernel at the corresponding output channel, input channel, and kernel location. The sum is taken over all input channels c_{in} and over the height and width of the kernel k_h and k_w. Employing the convolution operation naïvely with HE requires considerable storage and computational resources. A more efficient approach to the convolution operation for HE was introduced by Juvekar et al. [43], called vector encoding, which utilizes SIMD optimization in HE to perform multiple convolution operations in parallel. This approach has also been adopted for use in many privacy-preserving CNN works, such as [44], [45], [46], [47].

To use this approach, we generalize it and implement a configurable convolutional layer reusable across all forms of networks. This layer accepts an encrypted input along with metadata specifying the input width, number of input and output channels, and kernel size. The input to the first convolutional layer is a flattened and encrypted spatial tensor, represented as a SIMD ciphertext. This ciphertext is formed by packing the input values channel-by-channel into a single vector, which is then encrypted. Each subsequent layer in the network processes an encrypted input, which is the ciphertext output from the preceding layer.

To perform the encrypted convolution, we generate $k^2 - 1$ rotations of the input SIMD ciphertext (where k is the kernel size). Each rotated ciphertext corresponds to a shift in the input and is multiplied by a precomputed SIMD vector containing repeated copies of the corresponding kernel weight. The unrotated input is similarly multiplied by a SIMD vector containing the repeated first kernel element. All resulting ciphertexts from these multiplications are summed to produce a final ciphertext encoding the full convolution output. Finally, to obtain a compact convolution, we apply w_{out} ciphertext rotations to drop intermediate error values introduced during by the encrypted computation. Figures 3 and 4 in the Appendix illustrate this convolution process for a 2×2 kernel.

4.3 Average Pooling

The pooling layer in deep learning is primarily used for dimensionality reduction [48]. It is conceptually similar to a convolution layer, but differs because its filters

are uniform. The two widely used pooling layers in deep learning are *Maximum Pooling* and *Average Pooling*. Maximum Pooling selects the maximum value from each region covered by the pooling kernel. At the same time, Average Pooling computes the average of all values within the region covered by the pooling kernel [49]. Pooling layers often use striding, allowing the filter to skip certain input regions. This effectively reduces the data size used in subsequent network layers. In the context of CKKS, *Maximum Pooling* is computationally expensive as it involves evaluating a non-linear greater-than function. In contrast, *Average Pooling* is more efficient and well-suited for encrypted computations as it can be performed using a single homomorphic multiplication and $k^2 - 1$ homomorphic additions. Average pooling is mathematically defined as follows:

$$\text{Output}(i, j) = \frac{1}{k^2} \sum_{x=0}^{k-1} \sum_{y=0}^{k-1} \text{Input}(i + x, j + y), \tag{3}$$

This operation involves summing all kernel-mapped elements and multiplying by a pre-computed $\frac{1}{k^2}$. Our average pooling implementation uses the same algorithm as in the convolution layer to generate the different rotations of the input tensor. We then sum the rotated ciphertexts and multiply with a precomputed value, $\frac{1}{k^2}$. Figure 5 in the Appendix show average pooling for $k = 2$.

4.4 Fully Connected Layer

Fully connected layers map all input data to an output space by learning a set of weights that define the relationships between all input and output neurons in the neural networks [50]. In this layer, every input neuron is connected to every output neuron, thus allowing the network to learn complex global patterns. Mathematically, if given an input vector x of size n, weights W of size $m \times n$, biases b of size m, and output vector y of size m, the computation for the fully connected layer can be expressed as:

$$y_k = \sum_{i=1}^{n} W_{ki} x_i + b_k \quad \text{for each } k = 1, 2, \ldots, m \tag{4}$$

With SIMD ciphertext packing in CKKS, operations in the fully connected layer become highly efficient, requiring only one multiplication and n additions for each output neuron, making the fully connected layer the most computationally efficient component in PrivSpike. For each output neuron in any fully connected layer, a single SIMD multiplication is performed for $W_i \times x_i$, where W is the weights vector of the output neuron, x represents the input neurons vector, and i denotes the index of the corresponding input neuron. Following this, n homomorphic additions are carried out to compute the result for the output neuron. These computations are repeated n_{out} times for all output neurons in the layer. To aggregate the results, n_{out} rotations are applied to combine the output neuron values into a single ciphertext representing the layer's outputs. n_{out} rotations also means n_{out} rotation keys for all corresponding indices.

4.5 Leaky-Integrate and Fire (LIF) Layer

Activation Functions are essential components of deep neural networks, enabling them to learn complex, non-linear relationships within data [51]. In SNNs, they play a unique and critical role by determining how neurons process and transmit information through discrete spikes, mimicking the behavior of biological neurons. Unlike traditional neural networks, which use activation functions that output continuous values, SNNs employ spike-based mechanisms to encode information temporally, making them ideal for tasks that require dynamic and time-sensitive processing. Several activation mechanisms are commonly used in SNNs, including but not limited to the Integrate-and-Fire (IF) model, Leaky Integrate-and-Fire (LIF) model, and the Hodgkin-Huxley model [52]. These activation functions enable temporal information processing by incorporating dynamics such as integrating input signals over time, voltage decay, and spike generation. Among these functions, the LIF model is the most widely adopted due to its balance between computational efficiency and biological realism.

In the LIF model, neurons accumulate input currents over time $I(t)$ into a state variable known as the membrane potential, denoted as $V(t)$ [53]. The neuron emits a spike when the membrane potential crosses a predefined threshold Th. After firing, the neuron undergoes a reset, returning $V(t)$ to a resting state V_{rest}. The membrane potential dynamics are governed by a differential equation that incorporates a decay term, reflecting potential dissipation over time. This decay is controlled by the membrane time constant τ_m, which influences the rate at which the voltage returns to its resting state without input. The behavior of the LIF neuron is expressed as shown in Equation 5.

$$\tau_m \frac{dV}{dt} = -(V - V_{\text{rest}}) + I \tag{5}$$

A neuron emits a spike when $V \geq Th$, where Th is the threshold voltage. Upon firing, the membrane potential is reset to V_{rest}, and the neuron is optionally subjected to a refractory period, during which it temporarily becomes inactive. To further simplify this equation for implementation purposes, the discretized version of the equation is expressed as shown in Equation 6.

$$V(t+1) = \left(1 - \frac{\Delta t}{\tau_m}\right) V(t) + I(t) \tag{6}$$

In Equation (6), Δt is the discretization step size, and we assume that $V_{rest} = 0$. Algorithmically, we define the LIF model as shown in Algorithm 1. It contains a non-linear comparison operation, used to determine whether the membrane potential exceeds the threshold. However, this operation cannot be directly evaluated in the CKKS scheme since the scheme does not support the evaluation of non-linear functions. To address this limitation, we propose two distinct versions of this algorithm. We employ the Chebyshev polynomials to approximate the comparison function in a computationally feasible manner. We also employ the scheme-switching technique for high precision evaluation of the LIF within the TFHE scheme.

Algorithm 1 The Leaky Integrate-and-Fire Algorithm

Input:, $V(t)$, $I(t)$, t
Initialization:
$V_{rest} \leftarrow 0$
$\tau_m \leftarrow 0.25$
Th $\leftarrow 0.05$
spikeValue $\leftarrow 0$
for all time steps $t = 1, 2, \ldots, T$ **do**
  If initial timestep:
  if $t = 1$ **then**
    $V(t) \leftarrow I(t)$
  else
    $V(t) \leftarrow (1 - \frac{\Delta t}{\tau_m})V(t-1) + I(t)$
  end if
  Spike Decision:
  if $V(t) >$ Th **then**
    spikeValue $\leftarrow 1$
    $V(t) \leftarrow V_{\text{rest}}$
  else
    spikeValue $\leftarrow 0$
  end if
  $V(t) \leftarrow (1 - \text{spikeValue}) \cdot V(t)$
end for

Polynomial Approximation of the LIF Function Chebyshev polynomials are a set of orthogonal polynomials defined over the interval $[-1, 1]$, and they are particularly useful for approximating non-linear functions with high precision [54]. This approach has been extensively used in privacy-preserving CNN works with HE as the state-of-the-art approach for evaluating the ReLU function [46].

Using this approach, we then convert the comparison function present in LIF and express it as a series of polynomials, which becomes CKKS-friendly and computationally feasible. In the context of the LIF neuron model, we experimentally determine efficient values to scale $V(t)$ to $[-1, 1]$ for each dataset and each layer and set them as an input parameter for the function since the values of $V(t)$ are the encrypted results of the previous convolution operation and are not within the desired range for evaluation. We replace the comparison operation $V(t) > Th$ with an approximation function based on Chebyshev polynomial evaluation of the $>$ operation. With Chebyshev polynomial approximation, the degree of the polynomial expansion N determines the accuracy of the approximation of the spiking activation function. The higher the degree, the better the approximation to the non-linear function. On the downside, the higher the polynomial degree, the higher the computational complexity of the approximation function. With further application of SIMD, we design an FHE-friendly algorithm for the LIF function based on Chebyshev approximation as presented in Algorithm 2.

This algorithm takes an input membrane potential and accumulated voltage ciphertexts, and scales their sum using a predetermined scaling value to fit all values of the ciphertext within [-1, 1]. We then apply a SIMD-friendly approx-

Algorithm 2 Secure LIF Algorithm Using Chebyshev Approximation

Input:, $V(t) \leftarrow$ **vector**, $I(t) \leftarrow$ **vector**, t, $scaleValue$
Output:, $Spikes(t) \leftarrow$ **vector**, $V(t) \leftarrow$ **vector**
Initialization:
$\tau_m \leftarrow 0.25$
Th $\leftarrow 0.5/scaleValue$
N $\leftarrow 50$
If initial timestep:
if $t = 1$ **then**
    $V(t) \leftarrow I(t)$
else
    $V(t) \leftarrow (1 - \frac{\Delta t}{\tau_m})V(t-1) + I(t)$
end if
$V(t) \leftarrow$ ScalingFunction$(V(t), scaleValue, [-1, 1])$
Spike Decision:
$spikes(t) \leftarrow$ ApproximateGreaterThan$(V(t)$, Th,N$)$
$V(t) \leftarrow (1 - Spikes(t)) \cdot V(t)$

imation of the comparison function over a scaled threshold. To calculate the accumulated voltage of the next timestep, we subtract the result of the spikes multiplied by the input voltage from one and return both the spikes and the accumulated voltage. The spikes are then propagated to the next layer of the network, while the accumulated voltage is added to the input current of the next time step.

We experimentally evaluated the Chebyshev Polynomial Approximation function with varying values of N: 10, 20, 30, 40, 50, 60, 70, 80, and 90. For $N = 10$ and $N = 20$, the required multiplicative depths of 6 and 7, respectively, are relatively low. However, despite these low depths, the resulting approximations caused significant accuracy degradation, particularly when applied to the ResNet-20 model, making them less effective. Between $N = 30$ and $N = 59$, the multiplicative depth remained constant at 7. Through our experiments, we found that $N = 50$ provided the best trade-off, offering minimal accuracy degradation while maintaining reasonable computational efficiency. Higher values of N (ranging from 60 to 90) did not yield significant improvements in accuracy over $N = 50$, but they substantially increased the computational cost of the circuits. Based on these findings, we adopted $N = 50$ as the default value for our implementation. This degree of the Chebyshev polynomial requires seven multiplications for evaluation in the CKKS scheme, which is considered reasonable given the parameters needed for secure deep SNN models. Additionally, we parameterize N in our implementation, as different values of N can slightly impact the precision of the approximation.

Scheme-Switching Evaluation of LIF Function HE schemes offer different trade-offs in terms of efficiency and computational capabilities. These differences make certain schemes more suitable for specific types of encrypted operations. The CKKS scheme is particularly effective for approximate arithmetic on real-

valued data. It supports SIMD computations, which allows for parallel evaluation of linear operations over large encrypted datasets.

In contrast, schemes such as FHEW [55] and TFHE [16] are designed for high-precision computation based on Boolean logic. These schemes are well-suited for evaluating non-linear functions like comparisons, sign determination, and the floor function. To evaluate a function in these schemes, the function must be expressed as a Boolean circuit, which is built from basic logic gates such as AND, OR, XOR, and NOR gates. This gate-level representation enables accurate computation of discrete functions within the constraints of HE and is defined as the basic operations of these schemes. Although FHEW and TFHE achieve high accuracy through Boolean computations, they do not support SIMD parallelism. As a result, they are computationally expensive when used for large-scale data processing as well as evaluating complex deep circuits like those found in SNNs.

To leverage the strengths of TFHE and CKKS in PrivSpike, we also propose a hybrid LIF algorithm for high-precision evaluation of the LIF function based on scheme-switching. This algorithm combines both schemes for data processing in SNNs. All linear components in privacy-preserving SNNs, such as convolutional and fully connected layers, are computed using CKKS to leverage its parallel processing capabilities. For high precision in the non-linear components of the LIF, the TFHE scheme is employed for the comparison operation, while CKKS is maintained for all other linear operations found in the algorithm. The idea of switching between encryption schemes was introduced by Christina et al. in the CHIMERA framework [56]. Scheme switching is possible between CKKS and TFHE because both schemes are based on the LWE problem. This shared mathematical foundation enables ciphertexts to be converted between these schemes while preserving the encrypted values and semantic meaning, without any compromise in the security and privacy of the encrypted data.

We used the implementation of scheme-switching offered by OpenFHE in this work. It embeds CKKS and TFHE plaintexts into the common torus module T_R and realizes the conversion as homomorphic evaluations (external products, functional key-switching, and bootstrapping). Correctness follows because the homomorphic maps compute the intended algebraic transforms in T_R, and the analyses in [56] show that the converted ciphertexts decrypt to the desired mapped messages within the prescribed precision and rounding guarantees. Noise propagation is controlled through explicit variance bounds for the external product and functional key-switching operations: by selecting the precision α, decomposition length ℓ, and ring dimension N equal to the next power of 2, number of elements in the CKKS ciphertext. The resulting ciphertexts remain within the noise budgets required by both CKKS and TFHE. Finally, publishing the standard switching material (KS/BK/RK) does not introduce new cryptanalytic weaknesses beyond the Ring-LWE assumptions already required by TFHE/CKKS; the security of the hybrid construction therefore reduces to the assumed hardness of the underlying Ring-LWE instances. For formal statements, proofs, and parameter guidance, the OpenFHE implementation relies on the constructions and bounds in Chimera [56].

Our LIF scheme-switching algorithm proposed and implemented in PrivSpike is shown in Algorithm 3. It accepts three ciphertexts: one for the encrypted membrane potential, the encrypted accumulated voltage, and the encrypted threshold vector. A threshold vector, filled with the LIF model's standard threshold value of 0.5, is pre-computed and encrypted using the CKKS scheme for secure use in SNNs. In the LIF model, both the encrypted membrane potential and the threshold are converted into TFHE ciphertexts. A Boolean comparison circuit is then used to determine whether the neuron should spike by comparing the membrane potential and the threshold value. The result is a binary ciphertext that contains 0s and 1s, indicating the inverse spike states of all neurons. The spikes are calculated by subtracting all resulting SIMD binary ciphertext from 1, ensuring that all spiked neurons are 1s while 0s are non-spike neurons.

Algorithm 3 LIF Algorithm using the Scheme-switching Technique

Input:, $V(t) \leftarrow$ **vector**, $I(t) \leftarrow$ **vector**, $T(t) \leftarrow$ **vector** t, vectorSize
Output:, $Spikes(t) \leftarrow$ **vector**, $V(t) \leftarrow$ **vector**
Initialization:
$\tau_m \leftarrow 0.25$
If initial timestep:
if $t = 1$ **then**
    $V(t) \leftarrow I(t)$
else
    $V(t) \leftarrow (1 - \frac{\Delta t}{\tau_m})V(t-1) + I(t)$
end if
Spike Decision:
Set slots of c_{enc} to vectorSize
$tc_{\text{fst}} \leftarrow$ CKKSSwitchToTFHEW($V(t)$, vectorSize)
$tc_{\text{sec}} \leftarrow$ CKKSSwitchToTFHEW($T(t)$, vectorSize)
$cResult(t) \leftarrow$ TFHECompare($tc_{\text{fst}}, tc_{\text{sec}}$, vectorSize)
$spikes(t) \leftarrow 1 -$ cResults(t)
$V(t) \leftarrow cResults(t) \cdot V(t)$

4.6 Output Decoding Layer

The Output Decoding Layer in SNNs is responsible for interpreting the spiking activity of neurons and mapping them into meaningful outputs suitable for specific tasks, such as classification or regression [36]. The decoding layer translates these spike-based representations into a usable format by aggregating spike counts, analyzing spike timings, or employing temporal coding strategies to extract relevant information from the spike trains generated by the network.

In PrivSpike, the Output Decoding Layer aggregates the network's outputs over multiple steps to produce a final prediction. At each time step, the spike-based outputs are summed, effectively integrating temporal information and smoothing fluctuations in spike activity. This decoding approach combines the

advantages of temporal dynamics with a straightforward summation mechanism, enabling robust and simple handling of the sparse and discrete nature of spike-based representations. The network can refine its predictions by integrating information across time, compensating for noise or variability in individual time steps. This method aligns well with our approach, as summing multiple output ciphertexts is a computationally efficient operation. The aggregated results can be decrypted to obtain the inference results. The predicted label is determined as the class corresponding to the maximum value in the aggregated ciphertext.

5 Threat Model

Our threat model follows the conventions used in most prior HE-based works. It is designed for a setting in which a client outsources the evaluation of an SNN to a remote server. The client seeks to maintain the confidentiality of all sensitive information, including inputs, intermediate activations, and final outputs, while enabling the server to perform inference without learning anything about the underlying plaintext values. We consider two parties: a client, who holds the private key, and a server, which hosts the deployed PrivSpike model and performs all computations on encrypted data. The server receives encrypted inputs from the client and returns encrypted inference results. The server is treated as semi-honest (honest-but-curious), meaning that it faithfully follows the prescribed protocol during SNN evaluation but may attempt to extract information about the client's data by analyzing any encrypted values or auxiliary public information it receives. The client is considered fully trusted and is assumed to be the sole holder of the secret key.

The adversary we consider includes the computation server itself, the model owner if distinct from the server, and any external observer who can intercept communication between the client and the server. This adversary is allowed full access to all ciphertexts, public parameters, and auxiliary evaluation keys such as rotation keys, relinearization keys, and key-switching or bootstrapping keys. The adversary may perform arbitrary computations on these values, including attempts to infer relationships between ciphertexts or to launch chosen-input attacks, so long as it does not deviate from the protocol's execution path. Because the system uses CKKS, and because all evaluation keys are Ring-LWE ciphertexts that reveal no information about the secret key, the adversary cannot derive plaintext information beyond what is allowed by semantic security. All encrypted inputs, internal representations, and outputs remain indistinguishable from random under the Ring-LWE assumption. Security in PrivSpike therefore stems from the semantic security of CKKS and from the fact that the server never gains access to any plaintext intermediate values during computation. Only the client can decrypt inference results. The confidentiality of inputs, spike representations, synaptic currents, and final SNN outputs is preserved throughout the entire inference pipeline.

The threat model does not consider malicious servers that intentionally deviate from the protocol or fabricate incorrect ciphertexts, nor does it address

adversaries capable of mounting side-channel attacks such as timing, power, cache, or electromagnetic analysis. We also exclude attacks in which the client's private key or local device is compromised, as well as active network adversaries that attempt to modify ciphertexts in transit or inject malformed messages.

6 Experiment

This work was evaluated on two systems: the Ryzen 5900x CPU with 64GB of RAM (consumer-grade CPU) and 64-core AMD EPYC Milan 7713 CPUs with 256GB of RAM. All models that used the LIF approximation were evaluated on the Ryzen 5900X CPU, along with the LeNet-5 model that employed the scheme-switching approach for evaluating LIF. The ResNet-19 models, which employ the scheme-switching approach, were evaluated on the AMD EPYC system due to the higher memory requirements of this approach.

6.1 Spiking Neural Network Architectures

We validate PrivSpike using two well-known architectures within deep SNN literature: LeNet-5 and ResNet-19. These architectures serve as established benchmarks in SNN research, showcasing the effectiveness of our methods and their ability to scale across multiple architectures. Table 4 in the Appendix shows the details of the neural network layers, along with their input and output channels, kernel sizes, strides, and padding configurations, for the models from these architectures used in this study. The input channels of the first convolution layer are 2 for both N-MNIST and CIFAR-10 DVS. The first fully connected layer of our LeNet-5 model has 256 or 576 input features for the MNIST and N-MNIST datasets, respectively. LeNet-5, with its relatively simple design, serves as a baseline, while ResNet-19, with its deeper, more complex architecture, allows us to test PrivSpike on more challenging tasks. This contrast ensures comprehensive validation of our methods across varying architectural complexities, demonstrating the versatility of PrivSpike for broader applications.

Architectural Optimization of SNNs for HE Achieving efficient inference of SNNs under HE requires careful architectural design for optimal performance. Optimizations must reduce latency and computational overhead while maintaining accuracy and security guarantees. In this work, we adopt several key SNN architectural innovations tailored for efficiency under HE constraints.

Preprocessing Rotation Keys. We introduce an offline stage for precomputing and generating evaluation keys for HE-friendly SNNs. Here, we analyze the input tensor shapes, kernel dimensions, and channel configurations across all layers to determine the full set of ciphertext rotation indices required during inference. We precompute and generate the rotation indices and keys for the entire network. These keys are imported into the network and reused throughout inference, avoiding the redundancy and memory overhead associated with regenerating or

reloading duplicate keys. This ensures consistent memory usage across inferences, regardless of the number of time steps processed.

Layer-wise Iterative Evaluation. Unlike typical plaintext SNN implementations, which process all layers sequentially across each time step, we instead evaluate each layer iteratively across all time steps before proceeding to the next layer. This approach is motivated by the temporal structure of SNNs, where the output at a given time step depends on the current input and the accumulated membrane potential from previous steps. Iterative layer-wise evaluation enhances data locality and throughput, aligning more closely with the resource limitations of encrypted computation. This optimization enables the dynamic loading and unloading of model weights for individual layers during inference, significantly reducing memory requirements. However, when the model is used to infer multiple images in sequence, frequent loading and offloading of weights becomes a trade-off, as it slightly increases inference latency.

Bootstrapping in Deep Networks. To enable deep networks with LIF approximation, PrivSpike performs bootstrapping on the membrane potential ciphertext $V(t)$ before computing the spiking function approximation, particularly when the timestep is greater than one. This step is essential for maintaining ciphertext integrity because the Chebyshev polynomial used for approximation, such as a degree-50 polynomial, requires a multiplicative depth of at least seven. Furthermore, when an average pooling layer follows immediately after a bootstrapped non-linear layer, an additional bootstrapping is applied before pooling to ensure the correctness of encrypted computations. In contrast, fewer bootstrapping operations are required in networks that use scheme switching to evaluate the LIF. In these networks, bootstrapping is applied at carefully calculated intervals to minimize computational overhead.

Training All models were trained in PyTorch, and their weights were exported as CSV files and imported into the C++ implementations of our models. All models were trained on 80% of the dataset, with the remaining 20% used as the test set to validate model accuracy. PrivSpike is available on GitHub at `https://github.com/stamcenter/securespike`. To the best of our knowledge, PrivSpike is the first open-source privacy-preserving SNN inference framework.

6.2 Datasets

The MNIST dataset is a widely used benchmark in machine learning, consisting of 60,000 grayscale images of handwritten digits, each with dimensions 28×28 pixels [57]. To evaluate the inference accuracy of the LeNet-5 models, we use a sample of 1,000 images from the MNIST validation set. The N-MNIST dataset extends the original MNIST into an event-based format designed explicitly for SNNs and neuromorphic computing applications [58]. In this conversion, static handwritten digit images are transformed into sequences of asynchronous spike

events, which are stored in AER (Address-Event Representation) format. To prepare this data for use in our models, we considered the structure of our ciphertext for the most efficient transformation into the encrypted domain. Each digit sample is converted into a sequence of 5 time steps, where each time step represents a frame composed of accumulated spike events. Every frame contains two channels for ON and OFF polarities, formatted as a tensor of dimensions $2 \times 36 \times 36$, which captures both the spatial and temporal dynamics of the event. Unlike the default $2 \times 34 \times 34$ representation with 10 time steps commonly used in most SNN studies, our input transformation offers a more efficient alternative for HE-friendly models. This transformation reduced inference time in the encrypted domain by half while preserving plaintext accuracy. Specifically, training a LeNet-5 with the default N-MNIST transformation yielded a plaintext accuracy of 98.8%, whereas the same dataset using our transformed inputs achieved a slightly higher accuracy of 99.02% in the plaintext domain. To evaluate PrivSpike, we use 150 encrypted samples from the validation set of the N-MNIST dataset to test our privacy-preserving LeNet-5.

The CIFAR-10 dataset is another widely used dataset benchmark and contains 60,000 color images categorized into 10 distinct classes [59]. Each image has a resolution of $3 \times 32 \times 32$, corresponding to RGB color channels. Compared to MNIST, CIFAR-10 is more challenging due to the diversity and complexity of the objects it represents. In our evaluation, we used 150 images from the CIFAR-10 validation set to evaluate ResNet-19. Just like N-MNIST, the CIFAR10-DVS dataset is a neuromorphic version of the standard CIFAR-10 dataset, adapted for event-based processing in SNNs [60]. It consists of recordings from a Dynamic Vision Sensor (DVS) that captures changes in pixel intensity over time, resulting in sparse, asynchronous spike events. Each of the original CIFAR-10 images is converted into a short spatiotemporal event stream that reflects temporal dynamics similar to those observed in biological vision. The event streams are stored in AER format, where each event encodes the pixel location, timestamp, and polarity of the change. To prepare this dataset for efficient encrypted SNN inference, we converted each CIFAR10-DVS sample into 5 time steps, with each time step represented as a frame of dimensions $2 \times 32 \times 32$. This transformation enables the data to be efficiently fitted into SIMD ciphertexts under the CKKS scheme. In comparison to the common transformation approach used in SNN literature, which involves 10 time steps with frame dimensions of $2 \times 48 \times 48$, our reduced representation results in only a modest accuracy drop of approximately 5 percent in the ResNet-19 models from 73.6% to 68.10. We believe this is an acceptable trade-off given the significant reduction in computational overhead that would have been incurred to process models with inputs of size $2 \times 48 \times 48$ under HE. We evaluated our methods on 150 encrypted preprocessed samples from the validation set of CIFAR10-DVS.

6.3 HE Security Paramaters

We pre-determined the CKKS encryption parameter set to enable the most efficient computations within each neural network architecture. These parameters

include the polynomial degree for the CKKS encryption, the number of slots available in each ciphertext, the multiplicative depth, and the scaling modulus of bootstrapping. Selecting appropriate encryption parameters is critical, as they directly impact the number of ciphertext slots and the computation depth that can be achieved before and after bootstrapping. However, there is a trade-off as larger encryption parameters result in slower performance, increased key sizes, and higher memory requirements for ciphertext evaluations. Balancing these factors is essential to optimize both efficiency and security. Table 1 shows the parameter sets we used for the different architectures. We selected these parameters to serve as a general configuration for all models within the same architecture, thus giving us a matrix for comparative analysis.

Table 1: FHE parameter sets for encrypted inference.

Parameter	**LeNet-5**	**ResNet-19**
Polynomial Degree	16,384	32,768
Number of SIMD Slots	8,192	16,384
Multiplicative Depth	12	12
Scaling Factor	56	56

7 Results

We measure the accuracy, latency, and memory used by models built on PrivSpike. Table 2 compares the baseline accuracies of models with their encrypted accuracies and also shows the latency and memory profiles of all our models.

Table 2: Accuracy, Latency, and Memory Usage of Models Using Approximation and Scheme-Switch LIF Neurons Compared to Baseline Plaintext SNNs

Arch.	**Dataset**	**TS**	**LIF Type**	**Acc. (%)**	**Lat. (s)**	**Mem. (GB)**
LeNet-5	MNIST	2	Baseline	98.90	–	–
			Approx.	95.70	28	4.3
			Scheme-Sw.	98.10	110	9.4
	N-MNIST	5	Baseline	99.02	–	–
			Approx.	95.3	212	11.4
			Scheme-Sw.	97.3	714	25.4
ResNet-19	CIFAR-10	2	Baseline	83.19	–	–
			Approx.	76.0	784	18.7
			Scheme-Sw.	79.3	3264	85.9
	CIFAR-10 DVS	5	Baseline	68.10	–	–
			Approx.	64.71	1846	21.2
			Scheme-Sw.	66.00	8167	93.7

Accuracy On the LeNet-5 model using the MNIST dataset, the plaintext model achieved an accuracy of 98.9%. The encrypted counterpart using the LIF approximation achieved a 95.70%, reflecting a 3.2% accuracy drop. The version using scheme switching for LIF evaluation achieved a 98.10% accuracy, reducing the performance gap to only 0.8% relative to the plaintext baseline. These results indicate that both methods maintain high accuracy in the encrypted domain, with the scheme-switch approach almost matching the plaintext model. On the N-MNIST dataset, the plaintext LeNet-5 model achieved an accuracy of 99.02%. The encrypted model using LIF approximation achieves 95.3%, while the scheme-switching approach also matches the plaintext accuracy at 97.3% further supporting the findings we got from the MNIST dataset.

Using ResNet-19 on both the standard CIFAR-10 and the CIFAR-10 DVS datasets, the plaintext model achieved an accuracy of 83.19% while the encrypted variant using the LIF approximation achieved 76.0%. In contrast, the scheme-switching LIF approach improves accuracy in the encrypted domain to 79.3%, reducing the performance gap to 3.89%. On the CIFAR-10 DVS dataset, the plaintext model achieved an accuracy of 68.10% while the encrypted variant using LIF approximation obtained 64.71%. The ResNet-19 model using the scheme-switching reached a 66.0% accuracy.

Memory Usage We compared the memory consumption of the privacy-preserving LeNet-5 and ResNet-19 models by measuring the memory used per inference image. Models that utilize scheme switching for LIF evaluation showed significantly higher memory requirements compared to those using the polynomial approximation approach. Our results also indicated very minimal additional memory overhead across time steps, demonstrating efficient temporal scaling of encrypted inference. This efficiency is primarily due to the rotation reuse strategy employed in PrivSpike, where the most memory-intensive components (the rotation keys) remain unchanged regardless of the number of time steps. As a result, memory usage stays relatively constant over time.

Latency Although the scheme-switching evaluation of LIF results in higher-precision models, the latency requirements also significantly increase compared to models that use the approximation approach. This is understandable, as the evaluation of the LIF under TFHE requires unique keys for switching schemes, as well as slower computation. These results illustrate the trade-off one must consider when selecting an appropriate LIF evaluation approach.

Our evaluation of accuracy, memory usage, and latency demonstrates that the LIF scheme-switching technique faithfully preserves model accuracy. This result is expected, as the numerical errors introduced by scheme-switching are substantially smaller than those incurred by the approximation-based method, regardless of the polynomial degree. In contrast, the approximation approach requires significantly less memory and achieves lower runtime when evaluating each circuit. Consequently, the choice between the two LIF evaluation strategies

should be guided by the specific requirements of the target application; thus, developers must determine their design priority.

8 Conclusion

In this work, we introduced PrivSpike, an open-source novel framework for privacy-preserving inference in SNNs using the CKKS HE scheme. To enable efficient and high-precision evaluation of the LIF activation function under HE constraints, we proposed and implemented two novel HE-friendly LIF algorithms. Built on top of PrivSpike, we implemented the LeNet-5 and ResNet-19 architectures and proposed multiple architectural optimizations for HE-friendly SNNs. We infer the models on encrypted data across MNIST, Neuromorphic MNIST, CIFAR-10, and CIFAR-10 DVS datasets. Our findings show the potential and application of SNNs in privacy-preserving machine learning, especially in energy-constrained environments and event-driven data processing scenarios.

Unlike prior works, which are constrained to shallow SNNs and small-scale datasets, PrivSpike enables the development of deep privacy-preserving SNN models. The models built on PrivSpike consistently deliver superior accuracy while achieving significantly lower inference latency than prior work. On the LeNet-5 architecture, our LIF approximation-based model outperforms the work of Farzad et al. [15] by approximately 34×, and FHE-DiCNN [19] by approximately 50× in terms of inference speed while also showing better accuracies in both cases. Future work will focus on exploring end-to-end encrypted training and inference of SNNs, as well as developing more architecture-level optimizations to further reduce memory overhead and improve inference latency, particularly in the scheme-switching approach of evaluating the LIF.

References

1. N. Duggal, "Top 10 machine learning applications and examples in 2024," Sep 3, 2024. [Online]. Available: https://www.simplilearn.com/tutorials/machine-learning-tutorial/machine-learning-applications
2. U. Z. . A. S. Q. Asifullah Khan, Anabia Sohail, "A survey of the recent architectures of deep convolutional neural networks," *Artificial Intelligence Review*, 2020.
3. P. Plagwitz, F. Hannig, J. Teich, and O. Keszocze, "Snn vs. cnn implementations on fpgas: An empirical evaluation," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, I. Skliarova, P. Brox Jiménez, M. Véstias, and P. C. Diniz, Eds. Cham: Springer Nature Switzerland, 2024, pp. 3–18.
4. C. Cimarelli, J. A. Millan-Romera, H. Voos, and J. L. Sanchez-Lopez, "Hardware, algorithms, and applications of the neuromorphic vision sensor: a review," 2025. [Online]. Available: https://arxiv.org/abs/2504.08588
5. G. Lenz, G. Orchard, and S. Sheik, "Ultra-low-power image classification on neuromorphic hardware," 2024. [Online]. Available: https://arxiv.org/abs/2309.16795
6. Intel, "Taking neuromorphic computing to the next level with loihi 2 technology brief," https://www.intel.com/content/www/us/en/research/

neuromorphic-computing-loihi-2-technology-brief.html, 2021, accessed: 03-19-2024.
7. M. Xu, X. Chen, A. Sun, X. Zhang, and X. Chen, "A novel event-driven spiking convolutional neural network for electromyography pattern recognition," *IEEE Transactions on Biomedical Engineering*, vol. 70, no. 9, pp. 2604–2615, 2023.
8. ——, "A novel event-driven spiking convolutional neural network for electromyography pattern recognition," *IEEE Transactions on Biomedical Engineering*, vol. 70, no. 9, pp. 2604–2615, 2023.
9. R. Podschwadt, D. Takabi, P. Hu, M. H. Rafiei, and Z. Cai, "A survey of deep learning architectures for privacy-preserving machine learning with fully homomorphic encryption," *IEEE Access*, vol. 10, pp. 117 477–117 500, 2022.
10. Q. Zhang, C. Xin, and H. Wu, "Privacy-preserving deep learning based on multiparty secure computation: A survey," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 412–10 429, 2021.
11. M. Gong, Y. Xie, K. Pan, K. Feng, and A. Qin, "A survey on differentially private machine learning [review article]," *IEEE Computational Intelligence Magazine*, vol. 15, no. 2, pp. 49–64, 2020.
12. X. Li, B. Zhao, G. Yang, T. Xiang, J. Weng, and R. H. Deng, "A survey of secure computation using trusted execution environments," *ArXiv*, vol. abs/2302.12150, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:257102679
13. N. B. Njungle, E. Jahns, Z. Wu, L. Mastromauro, M. Stojkov, and M. Kinsy, "Guardianml: Anatomy of privacy-preserving machine learning techniques and frameworks," *IEEE Access*, 2025.
14. J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," Cryptology ePrint Archive, Paper 2016/421, 2016. [Online]. Available: https://eprint.iacr.org/2016/421
15. F. Nikfam, R. Casaburi, A. Marchisio, M. Martina, and M. Shafique, "A homomorphic encryption framework for privacy-preserving spiking neural networks," *Information*, vol. 14, no. 10, 2023. [Online]. Available: https://www.mdpi.com/2078-2489/14/10/537
16. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Tfhe: Fast fully homomorphic encryption over the torus," Cryptology ePrint Archive, Paper 2018/421, 2018, https://eprint.iacr.org/2018/421. [Online]. Available: https://eprint.iacr.org/2018/421
17. R. Casaburi, "Homomorphic encryption for spiking neural networks," Ph.D. dissertation, POLITECNICO DI TORINO, 2022.
18. J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:1467571
19. P. Li, H. Huang, T. Gao, J. Guo, and J. Duan, "Efficient privacy-preserving convolutional spiking neural networks with fhe," *ArXiv*, vol. abs/2309.09025, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:262046057
20. C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009, crypto.stanford.edu/craig.
21. C. Marcolla, V. Sucasas, M. Manzano, R. Bassoli, F. H. Fitzek, and N. Aaraj, "Survey on fully homomorphic encryption, theory, and applications," Cryptology ePrint Archive, Paper 2022/1602, 2022, https://eprint.iacr.org/2022/1602. [Online]. Available: https://eprint.iacr.org/2022/1602
22. N. Aggarwal, C. Gupta, and I. Sharma, "Fully homomorphic symmetric scheme without bootstrapping," pp. 14–17, 2014.

23. V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," Cryptology ePrint Archive, Paper 2012/230, 2012. [Online]. Available: https://eprint.iacr.org/2012/230
24. J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," Cryptology ePrint Archive, Paper 2018/931, 2018. [Online]. Available: https://eprint.iacr.org/2018/931
25. "Microsoft SEAL (release 4.1)," https://github.com/Microsoft/SEAL, Jan. 2023, microsoft Research, Redmond, WA., Accessed: 2024-02-27.
26. S. Halevi and V. Shoup, "Design and implementation of helib: a homomorphic encryption library," Cryptology ePrint Archive, Paper 2020/1481, 2020, https://eprint.iacr.org/2020/1481. [Online]. Available: https://eprint.iacr.org/2020/1481
27. Zama, "TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data," 2022, https://github.com/zama-ai/tfhe-rs.
28. A. A. Badawi, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee, Z. Liu, D. Micciancio, I. Quah, Y. Polyakov, S. R.V., K. Rohloff, J. Saylor, D. Suponitsky, M. Triplett, V. Vaikuntanathan, and V. Zucca, "Openfhe: Open-source fully homomorphic encryption library," Cryptology ePrint Archive, Paper 2022/915, 2022, https://eprint.iacr.org/2022/915. [Online]. Available: https://eprint.iacr.org/2022/915
29. N. B. Njungle, M. Stojkov, and M. A. Kinsy, "A safety-centric analysis and benchmarks of modern open-source homomorphic encryption libraries," 2025.
30. S. Ghosh-Dastidar and H. Adeli, "Spiking neural networks," *International journal of neural systems*, vol. 19, no. 04, pp. 295–308, 2009.
31. A. Marchisio, G. Nanfa, F. Khalid, M. A. Hanif, M. Martina, and M. Shafique, "Is spiking secure? a comparative study on the security vulnerabilities of spiking and deep neural networks," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–8.
32. A. Shrestha, H. Fang, Z. Mei, D. P. Rider, Q. Wu, and Q. Qiu, "A survey on neuromorphic computing: Models and hardware," *IEEE Circuits and Systems Magazine*, vol. 22, no. 2, pp. 6–35, 2022.
33. C. S. Han and K. M. Lee, "A survey on spiking neural networks," *International Journal of Fuzzy Logic and Intelligent Systems*, vol. 21, no. 4, pp. 317–337, 2021.
34. N.-D. Ho and I.-J. Chang, "Tcl: an ann-to-snn conversion with trainable clipping layers," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 793–798.
35. A. Almomani, M. Alauthman, M. Alweshah, O. Dorgham, and F. Albalas, "A comparative study on spiking neural network encoding schema: implemented with cloud computing," *Cluster Computing*, vol. 22, pp. 419–433, 2019.
36. Y. Yang, J. Ren, and F. Duan, "The spiking rates inspired encoder and decoder for spiking neural networks: an illustration of hand gesture recognition," *Cognitive Computation*, vol. 15, no. 4, pp. 1257–1272, 2023.
37. Y. Kim, H. Park, A. Moitra, A. Bhattacharjee, Y. Venkatesha, and P. Panda, "Rate coding or direct coding: Which one is better for accurate, robust, and energy-efficient spiking neural networks?" in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 71–75.
38. H. Mostafa, "Supervised learning based on temporal coding in spiking neural networks," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 7, pp. 3227–3235, 2017.

39. E. Forno, V. Fra, R. Pignari, E. Macii, and G. Urgese, "Spike encoding techniques for iot time-varying signals benchmarked on a neuromorphic classification task," *Frontiers in Neuroscience*, vol. 16, 2022. [Online]. Available: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2022.999029
40. I. Namatēvs, "Deep convolutional neural networks: Structure, feature extraction and training," *Information Technology and Management Science*, vol. 20, no. 1, pp. 40–47, 2017.
41. U. Kamath, J. Liu, and J. Whitaker, *Convolutional Neural Networks*. Cham: Springer International Publishing, 2019, pp. 263–314.
42. J. Jiang, D. Huang, J. Du, Y. Lu, and X. Liao, "Optimizing small channel 3d convolution on gpu with tensor core," *Parallel Computing*, vol. 113, p. 102954, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167819122000473
43. C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 1651–1669. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar
44. D. Kim and C. Guyot, "Optimized privacy-preserving cnn inference with fully homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 2175–2187, 2023.
45. E. Lee, J.-W. Lee, J. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and W. Choi, "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," in *Proceedings of the 39th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 17–23 Jul 2022, pp. 12 403–12 422. [Online]. Available: https://proceedings.mlr.press/v162/lee22e.html
46. L. Rovida and A. Leporati, "Encrypted image classification with low memory footprint using fully homomorphic encryption," Cryptology ePrint Archive, Paper 2024/460, 2024. [Online]. Available: https://eprint.iacr.org/2024/460
47. N. B. Njungle, E. Jahns, and M. A. Kinsy, "Fheon: A configurable framework for developing privacy-preserving neural networks using homomorphic encryption," 2025. [Online]. Available: https://arxiv.org/abs/2510.03996
48. H. Gholamalinezhad and H. Khosravi, "Pooling methods in deep neural networks, a review," 2020. [Online]. Available: https://arxiv.org/abs/2009.07485
49. V. Passricha and R. K. Aggarwal, "Chapter 2 - end-to-end acoustic modeling using convolutional neural networks," in *Intelligent Speech Signal Processing*, N. Dey, Ed. Academic Press, 2019, pp. 5–37. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128181300000027
50. geeksforgeeks, "Introduction to convolution neural network," 2018. [Online]. Available: https://www.geeksforgeeks.org/introduction-convolution-neural-network/
51. S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, vol. 503, pp. 92–108, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231222008426
52. J. D. Nunes, M. Carvalho, D. Carneiro, and J. S. Cardoso, "Spiking neural networks: A survey," *IEEE Access*, vol. 10, pp. 60 738–60 764, 2022.
53. C. Teeter, R. Iyer, V. Menon, N. Gouwens, D. Feng, J. Berg, A. Szafer, N. Cain, H. Zeng, M. Hawrylycz *et al.*, "Generalized leaky integrate-and-fire models classify multiple neuron types," *Nature communications*, vol. 9, no. 1, p. 709, 2018.

54. L. A. Gil-Alana and J. C. Cuestas, "A non-linear approach with long range dependence based on chebyshev polynomials," 2012.
55. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," Cryptology ePrint Archive, Paper 2016/870, 2016. [Online]. Available: https://eprint.iacr.org/2016/870
56. C. Boura, N. Gama, M. Georgieva, and D. Jetchev, "Chimera: Combining ring-lwe-based fully homomorphic encryption schemes," *Journal of Mathematical Cryptology*, vol. 14, no. 1, pp. 316–338, 2020.
57. L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.
58. G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in Neuroscience*, vol. 9, 2015. [Online]. Available: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2015.00437
59. A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.
60. H. Li, H. Liu, X. Ji, G. Li, and L. Shi, "Cifar10-dvs: An event-stream dataset for object classification," *Frontiers in Neuroscience*, vol. Volume 11 - 2017, 2017. [Online]. Available: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2017.00309

9 Appendix

9.1 Chebyshev polynomial Approximation

Chebyshev polynomials are denoted as $T_n(x)$, and defined recursively as:

$$T_0(x) = 1 \tag{7}$$

$$T_1(x) = x \tag{8}$$

$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x), \quad \text{for } n \geq 2 \tag{9}$$

These polynomials exhibit several important properties. Mainly, a polynomial of degree n has exactly n roots in the interval $[-1, 1]$, which are distributed as:

$$x_k = \cos\left(\frac{k\pi}{n}\right), \quad \text{where } 0 \leq k < n. \tag{10}$$

This distribution of roots ensures that Chebyshev polynomials achieve the best uniform approximation to a smooth function over $[-1, 1]$. This makes them highly efficient for approximating non-linear functions. The approximation of a non-linear function $f(x)$ over the interval $[-1, 1]$ can be achieved by expressing it as a series of Chebyshev polynomials:

$$f(x) \approx \sum_{n=0}^{N} c_n T_n(x), \tag{11}$$

where c_n are the coefficients determined by the function $f(x)$ and N is the degree of the polynomial.

9.2 Illustration of HE in outsourced cloud scenario

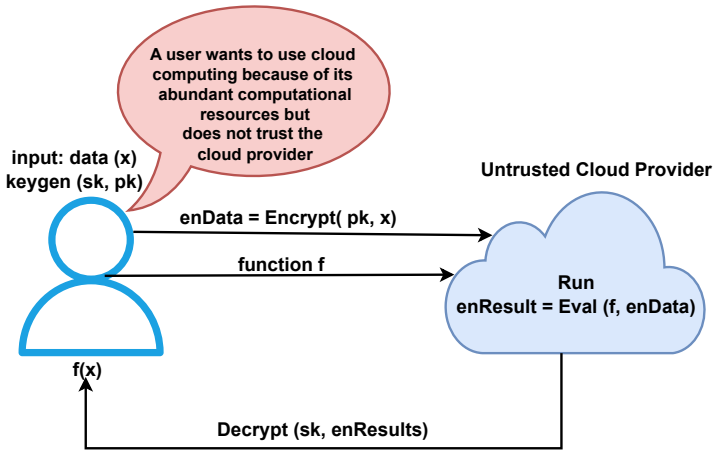

Fig. 2: An HE outsourced cloud computation scenario showing a user who encrypts data and evaluates a function in the cloud.

9.3 Comparative Studies compared with Related Works

We compare the results of our work with those presented in Farzad et al. [15] and FHE-DiCNN [19]. Table 3 highlights the superior efficiency and performance of PrivSpike in encrypted inference. Our baseline SNN LeNet-5 model, which uses the approximation approach for LIF, achieved 95.70% accuracy on MNIST using only two time steps, with an inference time of 28 seconds per image. This represents a substantial reduction in computational and latency requirements compared to the work by Farzad et al. [15], which required 40 time steps and 930 seconds to evaluate a single image from the Fashion-MNIST dataset using the same model architecture. Similarly, when compared to FHE-DiCNN [19], our SNN LeNet-5 model with LIF approximation computes 60,000 neurons on a consumer-grade CPU in just 28 seconds. In contrast, FHE-DiCNN requires approximately 1,500 seconds on a high-performance CPU to perform the same computation. For CIFAR-10, our ResNet-19 model achieved 76.0% using the approximation of the LIF in only two time steps, with a latency of 784 seconds. While no existing works utilize this complex architecture or dataset, Farzad et al. [15] estimate an inference latency of 901,800 seconds on a secure SNN AlexNet model with 60 time steps, on Fashion MNIST. This performance and capability difference between PrivSpike and related works shows our distinguished benefits.

Table 3: Comparison of PrivSpike LeNet-5 Model to Related Works

Model	Data	Time Steps	Latency	Accuracy (%)
PrivSpike LeNet-5	**MNIST**	**2**	**28**	**95.7**
PrivSpike ResNet-19	**CIFAR-10**	**2**	**784**	**76.0**
Farzad LeNet-5 [15]	F-MNIST	40	930	96.5
Farzad AlexNet [15]	F-MNIST	60	901,800	–
FHE-DiCNN [19]	MNIST	1	1,500	95.2

9.4 Model Architectures

Table 4: Layer configurations for LeNet-5 and ResNet-19. Conv: Convolution, FC: Fully Connected, RB: Residual Block. Config shows (Kernel Size, Padding, Stride).

Architecture	Layer	In Ch.	Out Ch.	Kernel, Padding, Stride
LeNet-5	Conv + LIF	1 (or 2)	6	5×5, 1, 0
	AvgPool	6	6	2×2, 2, 0
	Conv + LIF	6	16	5×5, 1, 0
	AvgPool	16	16	2×2, 2, 0
	FC + LIF	256 (or 576)	120	–
	FC + LIF	120	84	–
	FC	84	10	–
ResNet-19	Conv + LIF	3 (or 2)	16	3×3, 1, 1
	3 RBs	16	16	3×3, 1 (1,1,1)
	3 RBs	16	32	3×3, 1 (2,1,1)
	2 RBs	32	64	3×3, 1 (2,1)
	AvgPool	64	64	8×8, 1, 0
	FC	64	10	–

9.5 Spiking Neural Network Visual Representations

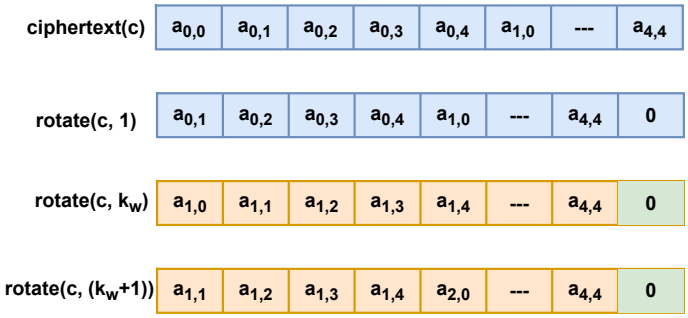

Fig. 3: Rotated Ciphertexts for convolution. The number of rotations is a total of $k^2 - 1$ where k is the kernel width.

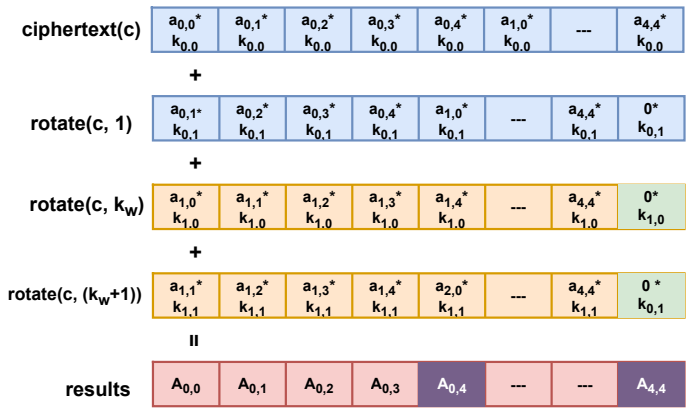

Fig. 4: Vector Encoding Convolution: The repeated kernel values are multiplied with equivalent rotated ciphertexts and summed to produce results.

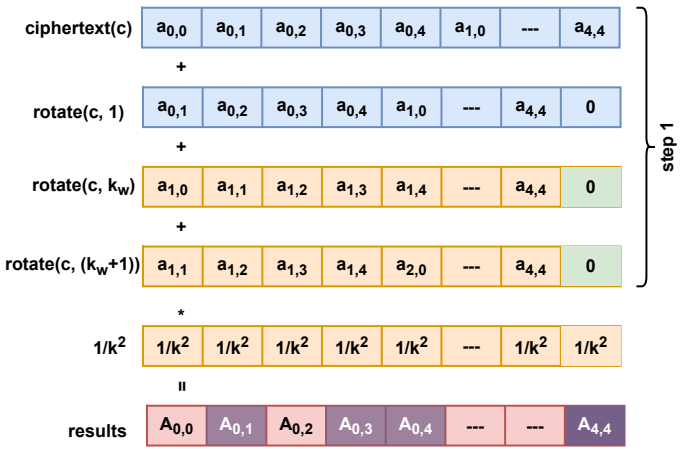

Fig. 5: Vector Encoding Secure Average Pooling: These rotations are then summed and multiplied by $1/k^2$