Practical Homomorphic LSTM via Programmable Bootstrapping

Thomas Crasson
AMIAD Pôle recherche
Palaiseau, France
thomas.crasson@polytechnique.edu

Nathan Cassereau
AMIAD Pôle recherche
Palaiseau, France
nathan.cassereau@polytechnique.edu

Florian Méhats
Sopra Steria Defense & Security
and CS Research Lab
Courbevoie and Palaiseau, France
florian.mehats@soprasteria.com

***Abstract*—While deep learning is ubiquitous, centralized processing exposes sensitive sequential data—such as natural language—to untrusted servers, forcing an unacceptable privacy-utility trade-off. Fully Homomorphic Encryption (FHE) resolves this by computing directly on encrypted data. However, standard neural networks ported to FHE suffer from severe latency bottlenecks, particularly because continuous non-linear activations dominate the computational budget.**

To overcome this, we introduce the Blind Spiking LSTM (BSLSTM), a TFHE-optimized recurrent architecture for privacy-preserving sequential inference. By co-designing the network with the cryptographic framework, we replace expensive continuous non-linearities with an efficient multi-threshold programmable bootstrapping paradigm. Evaluated on standard NLP tasks, BSLSTM achieves an inference latency of 3.8 seconds for a 128-token sequence, significantly outperforming traditional homomorphic approaches while maintaining competitive accuracy. Operating at an amortized cost of 155 microseconds per bootstrapping operation, our work demonstrates the practical viability of low-latency, fully homomorphic inference for real-world applications.

1. Introduction

Despite the unprecedented capabilities of deep learning, its reliance on centralized processing introduces critical data privacy vulnerabilities. Indeed, standard model inference requires access to raw data to perform computations, posing significant risks of data leakage. Privacy-Preserving Machine Learning (PPML) techniques have been developed to address these concerns. Although early approaches like Differential Privacy [1] effectively protect training data anonymity, they do not ensure input data privacy during the inference phase itself, leaving user queries exposed to untrusted servers.

Fully Homomorphic Encryption (FHE) stands out as a promising technique for secure neural network inference, as it enables computations directly on encrypted data without decryption. FHE provides strong mathematical guarantees of security, particularly regarding the time required to decrypt data without the secret key.

Four primary FHE schemes have been developed in past years: BFV [2], [3], BGV [4], CKKS [5], and TFHE [6], [7]. Among these, CKKS and TFHE have emerged as the most viable options for deep learning inference due to their specific operational capabilities over encrypted data.

Natural language is one of the many data modalities requiring privacy protection, as it often encodes rich contextual information. Recurrent Neural Networks (RNNs), and in particular Long Short-Term Memory (LSTM) networks [8], are an architecture of choice for modeling such sequential data, owing to their internal memory mechanism which captures long-range temporal dependencies. This memory mechanism, however, introduces a strict sequential dependency: each timestep's hidden state depends on the output of the previous one, precluding any parallelization across the temporal dimension.

Spiking Neural Networks (SNNs) [9], [10], [11] constitute an alternative recurrent paradigm in which the continuous activations of standard RNNs are replaced by discrete, event-driven spikes. This design yields highly sparse, binary internal representations while preserving the temporal, recurrent structure inherent to RNNs, making SNNs a natural, lightweight alternative for sequential processing under strict computational constraints.

In this work, we present a new spiking recurrent architecture, Blind Spiking LSTM (BSLSTM), based on TFHE's constraints. We design each gating and update mechanism of the BSLSTM to align as closely as possible with the native operations offered by TFHE rather than adapting an existing architecture post hoc. This tight co-design allows us to minimize the number and cost of the cryptographic operations required at each timestep, thereby drastically reducing the overall inference latency.

1.1. Related work

Recent research on FHE computation has focused on adapting existing deep learning models to perform inference on encrypted data. However, a major bottleneck of FHE is the significant latency required to compute non-polynomial functions. Some works bypass this issue by relying on client-server interactions, either through garbled circuits [12] or by requiring the client device to compute non-linearities

[13]. These hybrid setups lack scalability, as they necessitate continuous client connectivity and bandwidth during inference. To ensure complete server-side autonomy, we focus exclusively on approaches that execute all computations on the server, which can be categorized into two groups based on the underlying FHE scheme: those using CKKS [14], [15], [16], [17], [18], [19], [20], and those relying on TFHE [21], [22], [23], [24], [25], [26]. Most of this prior work, however, focuses on processing tabular data or low-resolution images such as MNIST and FashionMNIST.

Natural Language Processing (NLP) remains comparatively unexplored for FHE inference, owing to the high dimensionality of sequences and the large number of activation functions that must be evaluated. This gap is further compounded by the strict recurrent dependencies discussed above: because each timestep depends on the previous one, the sequential nature of RNN and LSTM architectures limits the effectiveness of SIMD batching-based schemes like CKKS, despite their frequent adoption in large NLP models [14], [15], [19], [20], [27]. TFHE, in contrast, offers native scalar-level granularity, enabling low-latency evaluation of temporal feedback loops; it further provides a smaller ciphertext memory footprint, better noise management, and the unique ability to compute arbitrary activation functions via Programmable Bootstrapping (PBS) instead of relying on precision-degrading polynomial approximations.

1.2. Contributions

Existing literature merely ports standard deep learning methods onto encrypted frameworks, inheriting severe performance penalties. We propose a different approach based on architectural co-design: we adapt deep learning layers to be intrinsically FHE-friendly while optimizing the FHE framework for deep learning. Specifically, we present a new TFHE-optimized recurrent architecture designed for secure and efficient sequential inference. Our contributions include:

1) **Fused Operator Computation:** We merge non-linear activation functions with element-wise weight multiplications, completely avoiding expensive ciphertext-ciphertext multiplications in TFHE.

2) **Quantization Relaxation:** We remove the mandatory rounding function before encryption, enabling floating-point representation within TFHE-based inference and relaxing the strict quantization constraints on weights found in prior works [21], [22], [28].

3) **Multi-Function Look-Up Tables:** We leverage multi-value Programmable Bootstrapping to evaluate k distinct functions simultaneously, increasing the model's expressiveness without incurring additional bootstrapping overhead.

4) **Cryptographically Optimized Recurrent Architecture:** We design a novel recurrent architecture tailored for TFHE. This co-design reduces the number of functional bootstrappings per timestep from $11d$ in standard TFHE-based LSTMs to only $3d/k$, where d denotes the hidden dimension.

2. Background

This section introduces the foundational concepts and architectures leveraged in our proposed approach.

2.1. Recurrent Neural Networks

2.1.1. LSTM. The Long Short-Term Memory (LSTM) network is a standard architecture for recurrent neural networks [8]. Its memory mechanism improves the learning capacity of the network by separating the long-term context of the sequence from the local, short-term information at each timestep. This architecture effectively mitigates the vanishing and exploding gradient problems inherent to standard RNNs [29].

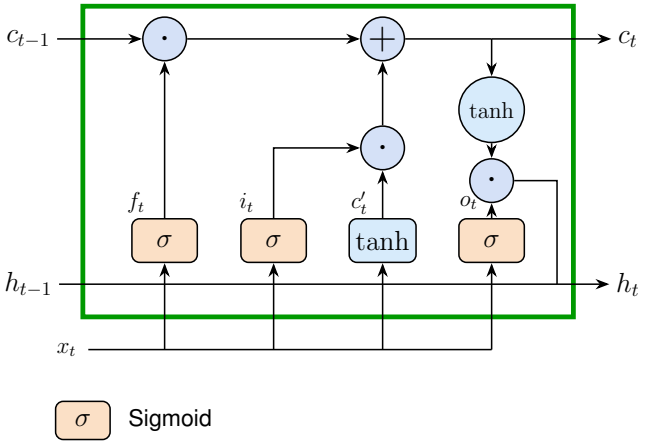

Figure 1: Canonical LSTM cell architecture

Figure 1 illustrates the canonical LSTM cell architecture. The LSTM cell employs a forget/input mechanism: it deletes obsolete information from the memory cell using the forget gate f_t, and integrates new knowledge via the input gate i_t and candidate cell c'_t. It then generates the new latent vector using the output gate o_t.

However, this canonical architecture is not well-suited for TFHE due to its reliance on multiple non-linear operations and element-wise multiplications, which introduce prohibitive computational overhead in FHE.

2.1.2. Spiking LSTM. Spiking Neural Networks (SNNs) [9], [10] have emerged as a computationally efficient alternative to standard floating-point neural networks. They mimic biological neuronal mechanisms, where neurons transmit only one bit of information to their neighbors. SNN-based recurrent architectures [30] have gained traction because they exhibit high sparsity—driven by the Heaviside activation function—and maintain a low memory footprint due to binary activations. These characteristics make them inherently well-suited for FHE inference [31], [32], [33], as the discrete hidden space and simplified activation functions are significantly cheaper to evaluate than continuous functions like ReLU or GeLU. We leverage this paradigm, specifically

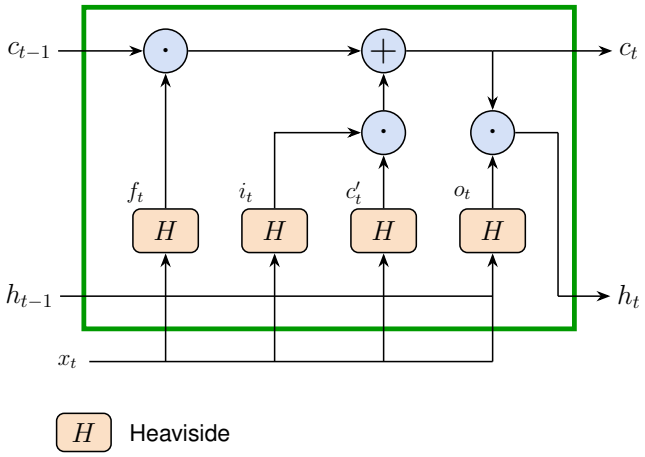

Figure 2: Spiking LSTM cell architecture

the Spiking LSTM architecture as defined in [11], to perform NLP tasks over encrypted data.

Figure 2 presents a Spiking LSTM cell. It adapts the Leaky Integrate-and-Fire (LIF) model [9], [30], [34] by using the Heaviside function as the firing mechanism. Compared to the canonical LSTM described in Section 2.1.1, all sigmoid activation functions are replaced by the Heaviside step function:

$$H_\gamma(x) = \mathbb{1}_{x>\gamma} \tag{1}$$

where γ is a hyperparameter of the SNN.

Because the candidate cell c'_t is restricted to binary values, the memory cell c_t is strictly bounded by t, promoting numerical stability. However, training SNNs presents challenges due to the non-differentiable nature of the Heaviside function. To overcome this, we employ surrogate gradient learning, utilizing arctangent-like surrogate gradients [35] during backpropagation.

2.2. System Architecture and Threat Model

Similarly to previous works [14], [16], [20], [26], [27], we consider a privacy-preserving inference scenario involving two parties: a client $\mathcal{C}$ and a server $\mathcal{S}$. We operate under an honest-but-curious (semi-honest) threat model, wherein both parties strictly adhere to the protocol specifications but may attempt to infer additional sensitive information from the data they receive. In this asymmetric setup, the client holds the sensitive input data but lacks the computational resources to execute the model locally. Conversely, the server provides the computational infrastructure and the proprietary pre-trained model. To achieve mutual privacy, the protocol must guarantee the confidentiality of the client's inputs against the server, while safeguarding the server's model weights from the client.

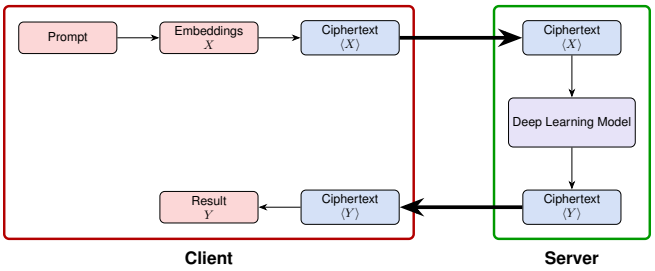

Figure 3: Inference Model Setup

The proposed secure inference protocol is illustrated in Figure 3 and proceeds as follows:

1) **Embedding Generation:** The client computes the embeddings of the input prompt locally. Because this operation consists primarily of deterministic index-based lookups from a vocabulary, it incurs negligible computational overhead for the client.
2) **Encryption:** The client encrypts the resulting embeddings under the TFHE scheme and transmits the ciphertexts to the server.
3) **Homomorphic Evaluation:** The server evaluates the recurrent neural network homomorphically over the encrypted inputs using its plaintext model weights. Upon completion, the server transmits the encrypted results back to the client.
4) **Decryption:** The client decrypts the received ciphertexts using their secret key to recover the final plaintext predictions.

Crucially, to enable the homomorphic evaluation of the non-linear operations required by the LSTM architecture, the client must provision the server with a comprehensive set of evaluation keys. This provisioning occurs during an offline setup phase prior to the inference. Because these keys are generated and transmitted only once, their communication overhead is fully amortized across all subsequent inference requests.

2.3. TFHE

2.3.1. Motivation. Most FHE approaches for deep learning inference rely on the CKKS scheme due to its fast ciphertext/ciphertext multiplication capabilities. CKKS is a leveled FHE scheme. It allows the user to perform a limited number of multiplications, determined by its multiplicative depth (or levels), before needing to carry out a bootstrapping operation to replenish this multiplicative budget. State-of-the-art CKKS parameters for deep learning [14], [19] allow between 9 and 21 levels, depending on the cryptographic parameters, for a 128-bit security level. However, polynomial approximations used in CKKS are only as effective as the smoothness of the underlying function. Because SNN architectures rely on highly discontinuous activation functions (such as the Heaviside step function), CKKS is not a suitable scheme for this type of encrypted inference. In contrast, the TFHE scheme, with its ability to evaluate arbitrary functions

in constant time, proves to be fundamentally better suited for this task.

2.3.2. TFHE Capabilities.

TFHE encrypts a plaintext polynomial $m \in \mathcal{R}_{n,t} = \mathbb{Z}_t[X]/(X^n+1)$, where t is a known integer, using a secret key $S = (S_0, ..., S_{\ell-1})$, into a GLWE($m$) ciphertext [6], [7]. This ciphertext, denoted $\langle m \rangle_S$, is defined as follows:

$$\text{GLWE}(m) = \left(A, \sum_{i=0}^{\ell-1} A_i \cdot S_i + \Delta m + e \pmod q \right) \quad (2)$$

where $\cdot$ is the product operator in the polynomial ring $\mathcal{R}_{n,q}$, A is sampled uniformly from $\mathcal{R}_{n,q}^{\ell}$, S is sampled from $\mathcal{R}_{n,q}^{\ell}$ with binary or ternary coefficients, Δ is a scaling factor[1], and e is Gaussian noise. When $n = 1$, the ciphertext is denoted LWE(m), and when $\ell = 1$, it is denoted RLWE(m).

The noise term grows as operations are computed over the encrypted message. TFHE supports several operations on ciphertexts, as defined in [6], [7], [36]:

1) Ciphertext/Ciphertext addition.
2) Ciphertext/Plaintext addition.
3) Ciphertext/Plaintext multiplication.
4) Key switching, which enables the transition from a secret key sk to sk'.
5) Coefficient extraction, which takes a ciphertext RLWE($\sum_{i=0}^{N-1} m_i X^i$) and an index $k \in [0, N-1]$, and returns LWE(m_k).
6) Coefficient packing, which takes a set of ciphertexts $\{\text{LWE}(m_i)\}_{i\in[0,d]}$ where $0 \leq d < N$, and generates RLWE(M) = RLWE($\sum_{i=0}^{d} m_i X^i$).
7) Modulus switching, which changes the ciphertext modulus q to a new modulus q' in Equation (2).
8) Functional Bootstrapping (FB), also known as Programmable Bootstrapping , which evaluates arbitrary non-linear functions while simultaneously reducing the noise of the ciphertext.

To multiply two ciphertexts, denoted $\langle x \rangle$ and $\langle y \rangle$, TFHE requires computing the difference [23]:

$$\langle xy \rangle = \frac{(\langle x+y \rangle)^2}{4} - \frac{(\langle x-y \rangle)^2}{4} \quad (3)$$

where the square operations are evaluated using functional bootstrapping. This method introduces a significant computational overhead as it requires two separate bootstrappings. Consequently, TFHE-based algorithms strive to avoid ciphertext/ciphertext multiplications entirely.

2.3.3. Functional Bootstrapping and Optimizations.

In every FHE scheme, bootstrapping [36] remains a critical operation as it represents the primary bottleneck of homomorphic computation. Unlike the CKKS approach—which approximates activation functions via Taylor or minimax expansions and uses bootstrapping solely to restore multiplication levels—TFHE requires a bootstrapping operation to compute each non-linear activation function. This offers the structural advantage of producing ciphertexts with a fresh, predefined noise level after every activation. However, it also demands a significant number of operations since activations must be evaluated individually for each neuron; there is no native batched algorithm for computing arbitrary functions over a set of TFHE ciphertexts. This computational overhead explains why recent FHE deep learning frameworks [15], [18], [37] have predominantly favored CKKS. Despite this limitation, TFHE is the optimal choice for recurrent architectures (such as SNNs) requiring exact non-linearities, due to its ability to evaluate arbitrary functions with a fixed time complexity.

The core of TFHE bootstrapping relies on the blind rotation technique, which acts as a sequential homomorphic decryption. Prior to this step, a key switching and a modulus switching are applied to the input ciphertext.

It is worth noting that the complexity of the bootstrapping procedure scales linearly with the secret key size. Thus, a straightforward optimization is to perform a key switch on the input LWE ciphertext prior to bootstrapping, converting it to a smaller secret key. However, maintaining a constant security level with a smaller key requires accommodating a higher noise variance, resulting in noisier ciphertexts. Following this key switch, a modulus switch from q to $2N$ is applied. Finally, a blind rotation is performed over the Look-Up Table (LUT) polynomial defined as:

$$\text{LUT}(X) = \sum_{i=-\frac{N}{2}}^{\frac{N}{2}-1} f(i) X^i \quad (4)$$

The output of the blind rotation is the constant coefficient of RLWE(LUT$(X)X^{b-a\cdot s+\epsilon}$), where (a, b) is the LWE input of the blind rotation, and ϵ represents the original noise of the ciphertext aggregated with the noise introduced by the key switching and modulus switching operations.

A fundamental constraint for the correctness of the bootstrapping procedure lies in bounding the message derived from the modulus switch. Because the blind rotation evaluates the LUT using a cyclotomic polynomial product, the operation is subject to the negacyclic property of the ring. To correctly map the phase to the intended LUT value without triggering an unintended wrap-around at the boundaries, the scaled plaintext Δm must be strictly bounded within the interval:

$$\Delta m \in \left]-\frac{q}{4}, \frac{q}{4}\right[\quad (5)$$

where q is the ciphertext modulus. Exceeding these limits leads to an incorrect phase evaluation, causing the blind rotation to fail.

To counter the noise, redundancy is added to the LUT by repeating the same value multiple times[2]. The effective

1. Traditional TFHE encoding sets Δ as q/t. Nevertheless, in Section 3.1, we introduce an alternative scaling factor β_x to relax the weight quantization constraints.

2. Because our neural network architecture relies on the Heaviside step function, the corresponding LUTs are inherently highly redundant, containing only two distinct output values.

LUT is thus defined as:

$$\text{LUT}(X) = \sum_{i=-\frac{N}{2}}^{\frac{N}{2}-1} f(x_i)X^i, \quad x_i = \lfloor it/N \rceil. \tag{6}$$

To further accelerate inference time, we integrate the collapse technique introduced in [38], [39] to significantly reduce the bootstrapping duration. This technique simultaneously decreases the variance of the noise induced by the modulus switch, thereby improving the overall numerical robustness of the TFHE inference.

Finally, another major optimization is the many-LUTs FB [40], which enables the evaluation of multiple functions over the same input LWE(x). Given a set of functions $f_1, \ldots, f_k$, we define the following composite LUT:

$$\text{LUT}(X) = \sum_{j=1}^{k} \sum_{i=-\frac{N}{2k}}^{\frac{N}{2k}-1} f_j(x_i)X^{ik+j-1}, \quad x_i = \lfloor ikt/N \rceil. \tag{7}$$

This structure permits the simultaneous computation of k functions by reducing the quantization precision over the input space. Consequently, instead of subdividing the interval $[-t, t]$ into N quantization levels, it is partitioned into N/k levels.

3. TFHE-Friendly Architecture

This section presents the core technical mechanisms introduced to optimize TFHE for deep learning inference.

3.1. Relaxing Quantization: Floating-Point Adapted Encoding

We propose a novel quantization strategy that enables higher precision for real-valued messages.

Existing literature on TFHE-based deep learning [21], [22], [23], [24], [26], [28] typically quantizes inputs strictly before encryption. Specifically, for an input $x \in \mathbb{R}$, a scaling factor Δ_x is defined such that the quantized integer $X = \lfloor \Delta_x x \rceil$ is computed and subsequently encrypted as $\langle X \rangle = (A, A \cdot S + \Delta X + e \pmod q)$. This approach does not fully exploit the continuous torus structure of TFHE, treating encryption merely as a wrapper around a pre-quantized integer. To avoid overflow during linear operations involving a weight matrix W, this standard approach must satisfy the following stringent constraints to correctly perform a Functional Bootstrapping (FB):

$$\begin{aligned} X &= \text{Quantize}(x) = \lfloor \Delta_x x \rceil, \\ W &= \text{Quantize}(w) = \lfloor \Delta_w w \rceil, \\ &\max_{x \in \mathcal{X}}(|XW|) < t/4 \end{aligned} \tag{8}$$

where $\mathcal{X}$ denotes the input space. In typical TFHE-based deep learning implementations, the plaintext modulus t is severely restricted, often limited to 2^8 or 2^{10} [22], [23], [26]. Because this limited bit budget is usually distributed equally between weights and activations, both are often restricted to a mere 2^4 (4-bit) distinct values.

To address this limitation, we propose a modified TFHE encoding that integrates quantization directly into the encryption process. We encrypt a real value $x \in \mathbb{R}$ as follows:

$$\langle x \rangle = (A, A \cdot S + \lfloor \beta_x x \rceil + e \pmod q), \tag{9}$$

where $\beta_x \in \mathbb{R}$ serves as the real-valued scaling factor. Under this encoding, the overflow constraint for linear computations shifts from the restrictive plaintext modulus t to the much larger ciphertext modulus q:

$$\begin{aligned} X &= \text{Quantize}(x) = \lfloor \beta_x x \rceil, \\ W &= \text{Quantize}(w) = \lfloor \beta_w w \rceil, \\ &\max_{x \in \mathcal{X}}(|XW|) < q/4 \end{aligned} \tag{10}$$

Since $t \ll q$, these new constraints offer significantly more flexibility than those defined in Equation (8). By decoupling the quantization from the plaintext modulus t, we can substantially increase the precision of the weights W without degrading the resolution of the activations X.

Furthermore, this paradigm allows each ciphertext $\langle x \rangle$ to be associated with its own scaling factor β_x, effectively introducing a dynamic scaling system reminiscent of the CKKS scheme. To multiply a ciphertext $\langle c \rangle$ (with scaling factor β_c) by a plaintext scalar f, the scalar is quantized using a constant $\beta_f \in \mathbb{R}$, resulting in an output ciphertext $\langle f \times c \rangle$ with a combined scaling factor $\beta_c \times \beta_f$. To prevent the scaling factor from growing indefinitely and causing an overflow over $q/4$, a bootstrapping operation is periodically performed to reset the scaling factor back to a nominal β_x.

Additionally, to homomorphically sum two ciphertexts $\langle x \rangle$ and $\langle h \rangle$, their respective scaling factors β_x and β_h must be perfectly aligned ($\beta_x = \beta_h$), consistent with standard TFHE addition protocols [6], [7].

Finally, to fully leverage this encoding and maximize the available precision, we employ Quantization-Aware Training (QAT) to ensure that model weights are strictly bounded by 1. We define the normalized weights $\widetilde{W}$ as follows:

$$\widetilde{w}_{i,j} = \frac{w_{i,j}}{\max_i(|w_{i,j}|)}. \tag{11}$$

Linear operations are then performed between the ciphertext $\langle x \rangle$ and the plaintext normalized weights $\widetilde{W}$. Given a fixed input scaling factor β_x, to prevent overflow during the dot product, the quantization constant for $\widetilde{W}$ is optimally set to $\lfloor q/(4\beta_x M) \rceil$, where $M = \max_{x \in \text{Valset}}(|x\widetilde{w}|)$. Consequently, the homomorphic computation of $x\widetilde{w}$ is executed as the product of the scaled encrypted input $\langle \beta_x x \rangle$ and the highly precise quantized weights $\widetilde{W}$.

3.2. Multiplication-Oriented LUT

Many deep learning architectures require element-wise multiplication of latent representations, particularly following the application of non-linear activation functions (e.g., the softmax operation in attention mechanisms [41] or gating

mechanisms in LSTM networks [8]). To homomorphically multiply ciphertexts, standard TFHE inference frameworks [23] typically rely on the difference-of-squares approach described in Equation (3). This method requires two distinct bootstrapping operations per multiplication, incurring significant computational overhead and latency.

To mitigate this bottleneck, we introduce a multiplication-oriented bootstrapping paradigm. Rather than treating the LUT as a static plaintext polynomial, we define a dynamic, encrypted LUT that absorbs one multiplication operand directly into its coefficients, allowing us to simultaneously evaluate the non-linear activation function and the ciphertext-ciphertext multiplication within a single bootstrapping operation — at the cost of a single PBS instead of three.

To illustrate this mechanism, consider the forget gate in an LSTM cell. Let A be the encrypted pre-activation scalar, $F = \mathbb{1}_{A>\alpha}$ be the binary output of the Heaviside step function, and C be the latent memory state. The goal is to homomorphically compute the product $F \cdot C$. The proposed algorithm for this operation is detailed in Algorithm 1. By packing the ciphertext $\text{LWE}(C)$[3] into a monomial RLWE ciphertext, we can perform a plaintext-ciphertext polynomial multiplication between the theoretical LUT and $\text{RLWE}(C)$. The resulting scaled, encrypted LUT is then fed as the test vector into the blind rotation of the bootstrapping process.

Algorithm 1 Ciphertext-Integrated LUT Bootstrapping

Input: Ciphertexts $\text{LWE}(C),\ \text{LWE}(H)$
Input: Bootstrapping key BSK
Input: $\text{LUT}(X) = \sum_{i=-N/2}^{N/2-1} f\left(\frac{i}{N}\right) X^i$
1: $\text{RLWE}(C \cdot X^0) \leftarrow \text{Packing}(\text{LWE}(C))$
2: $\text{RLWE}(\text{LUT}_{\text{scaled}}) \leftarrow \text{LUT}(X) \cdot \text{RLWE}(C \cdot X^0)$
Output: $\text{FB}(\ \text{BSK},\ \text{RLWE}(\text{LUT}_{\text{scaled}}),\ \text{LWE}(H))$

Using this method, a ciphertext-ciphertext multiplication is executed with the latency of a single bootstrapping operation, provided that one of the operands is the immediate output of an activation function. This optimization is critical for unlocking the performance of our proposed Spiking LSTM architecture, specifically within its forget, input, and output gating mechanisms.

3.3. TFHE Spiking LSTM

Standard LSTM architectures are computationally prohibitive for homomorphic inference due to their reliance on multiple non-polynomial activation functions (e.g., sigmoid and tanh). In TFHE, these continuous activations must be evaluated via Functional Bootstrapping, an operation whose latency is strictly independent of the underlying function's mathematical complexity. As noted in Equation (6), the Heaviside step function—central to Spiking Neural Networks (SNNs)—is natively compatible with this mechanism, mapping to a simple binary transition within the LUT. This inherent synergy makes SNNs an ideal backbone for TFHE-based recurrent architectures.

Despite the relative efficiency of SNNs, the primary computational bottleneck remains the raw number of bootstrapping operations required per timestep. To optimize this budget, we propose a streamlined Spiking LSTM cell that merges the traditional candidate and input gates into a unified activation step. This architectural optimization reduces the bootstrapping overhead per cell by 25% without significantly compromising the model's representational capacity.

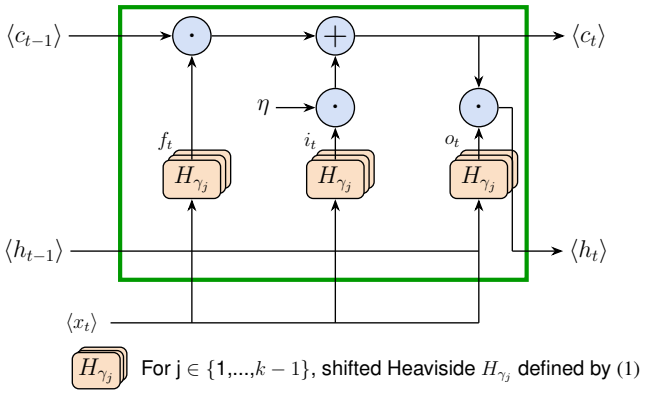

Figure 4: Proposed TFHE Spiking Cell

Figure 4 illustrates the atomic cell of our proposed TFHE-friendly architecture. Compared to the vanilla spiking LSTM (Fig. 2), the merged gating mechanism is followed by an element-wise multiplication with a learnable weight vector η. By employing the floating-point TFHE encoding introduced in Section 3.1, η is treated as a high-precision vector. The forward pass equations of the TFHE spiking cell are defined as follows:

$$a_t = [h_{t-1} \parallel x_t] \tag{12}$$
$$f_t = [H_{\gamma_1}(\widetilde{W_f} a_t) \parallel \cdots \parallel H_{\gamma_k}(\widetilde{W_f} a_t)] \tag{13}$$
$$i_t = [H_{\gamma_1}(\widetilde{W_i} a_t) \parallel \cdots \parallel H_{\gamma_k}(\widetilde{W_i} a_t)] \tag{14}$$
$$o_t = [H_{\gamma_1}(\widetilde{W_o} a_t) \parallel \cdots \parallel H_{\gamma_k}(\widetilde{W_o} a_t)] \tag{15}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \eta \tag{16}$$
$$h_t = c_t \odot o_t \tag{17}$$

where $\odot$ denotes the element-wise Hadamard product and $\parallel$ denotes vector concatenation. Let n be the dimension of the concatenated input a_t, and d be the dimension of the state vectors h_{t-1} and c_t. The learnable parameters W_f, W_i, W_o belong to $\mathbb{R}^{\frac{d}{k} \times n}$, and $\eta \in \mathbb{R}^d$.

To compute the ciphertext-ciphertext multiplications efficiently, we apply the ciphertext-integrated LUT method detailed in Section 3.2. Furthermore, to ensure the algebraic correctness of the concatenation, addition, and multiplica-

3. To pack a single LWE ciphertext into a monomial RLWE ciphertext, we adopt the trace-based packing technique introduced in [42], which is significantly more efficient for a small number of ciphertexts than standard key-switching packing methods.

tion operations in Equations (12), (16), and (17), we strictly align the scaling factors such that:

$$\beta_x = \beta_h = \beta_c \tag{18}$$

where $\beta_x, \beta_h, \beta_c$ are the scaling factors of the ciphertexts $\langle X_t \rangle, \langle H_t \rangle$, and $\langle C_t \rangle$, respectively.

Table 1 compares the computational complexity of our proposed architecture against standard techniques used to adapt existing models for TFHE inference.

TABLE 1: Comparison of the number of functional bootstrapping operations per timestep, where d is the hidden dimension.

Standard LSTM	Spiking LSTM	Proposed Spiking LSTM
$11d$	$10d$	$3d/k$

Our multiplication-oriented LUT paradigm drastically reduces the bootstrapping burden by eliminating $6d$ operations per timestep (corresponding to 3 multiplications). Additionally, omitting the traditional candidate vector c'_t removes another d operations.

To compensate for the simplified gating and enhance the network's expressiveness, we replace the standard binary activation with a multi-threshold mechanism, as detailed below.

3.4. Multi-Threshold Many-LUTs Activation

As described in Section 2.3.3, TFHE bootstrapping takes a single LWE ciphertext as input. Consequently, the computational complexity of homomorphic inference scales linearly with the hidden dimension of the network. To mitigate this overhead, we leverage the Many-LUTs bootstrapping technique (Eq. (7)) to evaluate multiple functions over the same ciphertext $\langle x \rangle$ within a single FB operation.

The standard Heaviside function, $H(x) = \mathbb{1}_{x>0}$, collapses all positive inputs to a single binary value, aggressively discarding information regarding the magnitude of the activation. To recover this loss of precision, we propose a multi-threshold activation strategy that evaluates several shifted indicator functions for a single input. We define a set of k thresholds $(\gamma_j)_{j\in[0,k-1]}$ as:

$$\gamma_j = -\alpha + \frac{2j\alpha}{k-1}, \quad \text{for } j \in \{0, \ldots, k-1\}. \tag{19}$$

Defining these thresholds accurately requires estimating the pre-activation input range for the Heaviside functions H_{γ_j}. Because our Quantization-Aware Training utilizes normalized weights $\widetilde{w} \in \mathbb{R}^{n\times d}$ strictly bounded by 1, we can derive a conservative bound for the variance of the linear output $x \cdot \widetilde{w}_\ell$, given the variance $\mathrm{Var}(x)$ of the input vector $x \in \mathbb{R}^n$:

$$\mathrm{Var}(x \cdot \widetilde{w}_\ell) < n \times \mathrm{Var}(x)$$

where $\widetilde{w}_\ell$ denotes the ℓ-th column of $\widetilde{W}$.

In summary, the state vectors $f_t \odot c_{t-1}$, $i_t \odot \eta$, and h_t are evaluated by seamlessly combining both optimizations: the Many-LUTs functional bootstrapping and the multiplication-oriented LUT paradigm.

4. Experiments

4.1. Implementation Details

To ensure precise benchmarking and efficient prototyping, our implementation was deployed across two distinct hardware configurations. To measure computational latency and benchmark execution times—enabling a direct comparison with the baseline reported in [14]—we utilized a server equipped with an NVIDIA H100NVL GPU (94 GB VRAM). Please refer to Section A.2 for further hardware specifications. Conversely, to facilitate rapid prototyping and architectural iterations, we employed a server featuring an NVIDIA GB200 GPU (192 GB VRAM), 8 CPU cores, and 80 GB of system RAM. The expanded memory capacity of the GB200 was critical for accommodating the massive memory footprint of encrypted tensors during the development phase.

Regarding the model architecture, we utilized pre-trained BERT embeddings [43]. The embedding layer remained frozen, while the recurrent backbone and the classification head were trained from scratch. Training was conducted entirely on plaintext data. We then evaluated the system's performance using a held-out test set processed through the homomorphic inference pipeline described in Figure 3.

Consistent with standard FHE literature, we assume that network communication latency is negligible relative to the substantial computational overhead of TFHE bootstrapping. Consequently, both the client-side encryption and the server-side inference were simulated on the same physical machine.

Our implementation is available in the following repository: https://gitfront.io/r/tc/KtHbzxHVYTBr/Blind-Spiking-LSTM/ .

4.2. Experimental Results

We evaluated our proposed architecture on three datasets from the GLUE benchmark [44]: SST-2, CoLA, and MRPC, each consisting of a training, validation, and test set. We trained the model on the training set and calibrated the quantization scale M (defined in Section 3.1) using the validation set. The recurrent backbone was configured with a hidden dimension of $d = 256$, followed by a two-layer MLP serving as the classification head. The multi-threshold Heaviside activation evaluates 4 LUTs ($k = 4$) with distinct thresholds as detailed in Section 3.4, while the MLP utilizes a ReLU activation evaluated via a single LUT. We encrypted the test set data, performed homomorphic inference using the TFHE Spiking LSTM and the classification head, and decrypted the final outputs for evaluation. The specific cryptographic parameters used are provided in Section A.1.

As shown in Table 2, we observe competitive metrics between our plaintext inference and the GLUE baseline, which is a bidirectional LSTM. Moreover, we notice only

TABLE 2: Performance comparison between plaintext inference, ciphertext inference, and the GLUE baseline.

Dataset	Plaintext	Ciphertext	GLUE Baseline
SST-2 *(Acc)*	85.6	85.7	85.9
CoLA *(Matthews Corr)*	10.8	11.1	15.7
MRPC *(F1/Acc)*	80.5/68.0	80.5/68.0	79.4/69.3

negligible performance deviations between plaintext and encrypted inference. These variations are inherently tied to the noise introduced by the TFHE cryptographic mechanisms, specifically:

- As discussed in Section 3.1, the quantization constant is directly linked to the scaling factor β_x. This quantization truncates low-precision information from the network's weights.
- The bootstrapping operation requires a modulus switch on the input, which increases the noise variance. This noise scales linearly with the number of LUTs (k) evaluated during the blind rotation, as formalized in Equation (7).
- The output of the bootstrapping operation contains a fresh, predefined noise variance dependent on the chosen cryptographic parameters. This noise is then propagated to the subsequent timestep through linear operations.
- The initialization of the ciphertext-integrated LUT requires packing an LWE ciphertext into an RLWE ciphertext, which incrementally raises the input noise.

Due to the complex interaction of these noise sources, analytically deriving the cumulative noise at the network's final output is challenging. To rigorously assess this compounding effect, we conducted an empirical study simulating the noise introduced by each cryptographic operation and compared the results against a non-quantized plaintext baseline. Furthermore, we measured the strict prediction agreement (fidelity) between the plaintext baseline and the encrypted inference.

TABLE 3: Prediction agreement between baseline inference and noisy output. Each value represents the percentage of data points for which the encrypted model's prediction strictly matches the plaintext baseline.

Dataset	Real TFHE	Simulated Theoretical Noise
SST-2	92.7%	92.9%
CoLA	89.1%	88.5%
MRPC	99.5%	99.5%

Table 3 demonstrates the high fidelity of ciphertext inference relative to the plaintext model on the test set. The injected cryptographic noise only marginally alters the final classification. Detailed analysis reveals that divergent predictions typically occur only when the plaintext model's confidence is inherently low (e.g., pre-activation logits close to zero), where even minor noise fluctuations can flip the decision boundary. Please refer to Section A.3 for a comprehensive ablation study on the impact of each noise mechanism.

TABLE 4: Comparison of amortized inference latency on an NVIDIA H100 GPU.

Model	Inference Time (s)
THOR	584.4
EFLLM	25.1
TFHE-LSTM	26.7
BSLSTM	3.8

Table 4 shows inference latency for several deep learning models. We compare the inference time of two transformer models using CKKS: THOR [16] performs full BERT inference, while EFLLM [14] evaluates two transformer layers. In addition to these two state-of-the-art CKKS-based models, we implemented a TFHE-based LSTM as an additional baseline. Finally, we report the inference time of our proposed BSLSTM architecture. We observe that our architecture achieves substantially faster inference than all other models.

Our implementation executes $3dT/k$ bootstrappings per inference, where T denotes the maximum sequence length. In practice, the network performs 24,576 bootstrappings in 3.8 seconds, yielding an amortized latency of 155 microseconds per operation. It is worth noting that in an isolated benchmark utilizing a perfectly parallelizable function—which fully saturates the GPU—our TFHE implementation achieves a peak amortized bootstrapping latency of merely 92 microseconds per operation.

4.3. Microbenchmark

In this section, we perform a microbenchmark of the baseline LSTM cell and a BSLSTM cell in order to compare the behavior of each architecture. This microbenchmark also allows us to assess the relevance of the architectural choices made for the BSLSTM.

Table 5 compares the ciphertext inference time of an LSTM cell and a BSLSTM cell. We observe that the experimental ratio of PBS runtime, computed as the sum of the first eight rows of Table 5 for each architecture, equals 14.4, which is very close to the theoretical value of 14.7 predicted by Table 1. However, the total latency ratio is different from the theoretical as the packing mechanism used for the multiplication-oriented LUT consumes a substantial amount of time. At the end of each cell, we pack the hidden state h_t and the memory cell c_t in order to reduce the memory footprint transmitted between successive cells. Table 5 shows that replacing standard TFHE multiplications with the Multiplication-Oriented LUT drastically reduces inference time. Moreover, replacing the candidate c'_t with η weights further reduces latency.

TABLE 5: Breakdown of the computation time (ms) per operation for a single LSTM cell and a single BSLSTM cell, both with hidden dimension $d = 256$ and $k = 4$. A dash (-) indicates that the operation is not part of the corresponding architecture.

Operation	LSTM (ms)	BSLSTM (ms)
f_t	20.5	5.4
i_t	20.7	5.4
o_t	20.9	5.5
c'_t	20.8	-
$f_t \cdot c_{t-1}$	43.2	-
$i_t \cdot c'_t$	42.8	-
$\tanh(c_t)$	21.8	-
$o_t \cdot \tanh(c_t)$	44.0	-
LUT packing f_t	-	4.1
LUT packing i_t	-	4.1
Packing (h_t, c_t)	3.7	3.4
Total	**238.3**	**27.8**

4.4. Ablation Study on Multi-Threshold Activation

To isolate the impact of the Many-LUTs activation mechanism, we conducted an ablation study on the parameter k (the number of simultaneously evaluated functions). To ensure a fair comparison, all evaluated configurations were constrained to an identical ciphertext inference latency budget. Given the strict predictive fidelity established in Table 2, these ablation experiments were conducted using the plaintext model to accelerate validation.

TABLE 6: Plaintext performance comparison relative to the number of LUTs (k) under a strictly fixed ciphertext inference time budget.

k	SST-2 (Acc)	CoLA (Matth. Corr)	MRPC (F1 / Acc)
1	83.5	5.0	80.1 / 67.9
2	85.1	7.4	80.5 / 68.5
4	85.6	10.6	80.5 / 68.0
8	85.3	15.7	80.2 / 68.6

Under a fixed latency constraint, increasing k allows the architecture to scale its hidden dimension proportionally, thereby increasing the total parameter count and the model's representational expressiveness. Table 6 corroborates this theoretical advantage, demonstrating a general positive trend in downstream task performance as k increases.

However, this architectural scaling is strictly bounded by underlying cryptographic constraints. As formalized in Equation (7), increasing k reduces the quantization precision available during the modulus switch operation. This precision loss inherently amplifies the ciphertext noise variance and elevates the failure probability of the blind rotation. Through empirical evaluation, we determined that $k = 4$ provides the optimal, safe trade-off between model expressiveness and cryptographic stability when using a polynomial ring degree of $N = 2048$.

5. Conclusion

In this work, we propose BSLSTM, a compact and efficient TFHE-friendly architecture designed for privacy-preserving sequential inference. We demonstrate a practical inference latency of 3.8 seconds over a 128-token sequence, maintaining competitive accuracy compared to standard plaintext recurrent baselines. Our work highlights how the inherent flexibility of TFHE can be harnessed to drastically reduce computational overhead. Specifically, by introducing a multiplication-oriented LUT paradigm and a multi-threshold Many-LUTs activation strategy, we enhance the model's representational capacity without incurring additional latency penalties.

Furthermore, we establish that Spiking Neural Networks are natively suited for TFHE, as their step-based activation functions translate seamlessly into exact binary Look-Up Tables. Conversely, standard leveled FHE approaches like CKKS would suffer from severe precision degradation on such architectures, as highly discontinuous functions—such as the Heaviside step—are notoriously difficult to approximate accurately via polynomial expansions. Ultimately, our work paves the way for the practical deployment of secure, compact NLP models, delivering low-latency inference while maintaining stringent cryptographic security guarantees.

References

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," 2016.

[2] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Advances in Cryptology – CRYPTO 2012*, 2012.

[3] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," 2012. [Online]. Available: https://eprint.iacr.org/2012/144

[4] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, 2014.

[5] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2017.

[6] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption over the torus," *J. Cryptol.*, 2018.

[7] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *Advances in Cryptology – ASIACRYPT 2016*, 2016.

[8] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, 1997.

[9] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural networks*, 2019.

[10] C. Zhou, H. Zhang, L. Yu, Y. Ye, Z. Zhou, L. Huang, Z. Ma, X. Fan, H. Zhou, and Y. Tian, "Direct training high-performance deep spiking neural networks: a review of theories and methods," *Frontiers in Neuroscience*, 2024.

[11] A. Lotfi Rezaabad and S. Vishwanath, "Long short-term memory spiking networks and their applications," in *International Conference on Neuromorphic Systems 2020*, 2020, pp. 1–9.

[12] B. Feng, Q. Lou, L. Jiang, and G. Fox, "Cryptogru: Low latency privacy-preserving text analysis with gru," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

[13] T. Chen, H. Bao, S. Huang, L. Dong, B. Jiao, D. Jiang, H. Zhou, J. Li, and F. Wei, "The-x: Privacy-preserving transformer inference with homomorphic encryption," in *Findings of the Association for Computational Linguistics: ACL 2022*, 2022.

[14] D. Rho, T. Kim, M. Park, J. W. Kim, H. Chae, E. K. Ryu, and J. H. Cheon, "Encryption-friendly llm architecture," in *The Thirteenth International Conference on Learning Representations*, 2024.

[15] J. Park, S. Park, J. H. Park, J. H. Ahn, J. H. Cheon, G. Hanrot, J. W. Kim, M. Park, and D. Stehlé, "Scaling up privacy-preserving ml: A ckks implementation of llama-2-7b," 2026. [Online]. Available: https://arxiv.org/abs/2601.18511

[16] J. Moon, D. Yoo, X. Jiang, and M. Kim, "THOR: Secure transformer inference with homomorphic encryption," in *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, 2024.

[17] C.-H. Nguyen, T. H. Dinh, D. N. Nguyen, K. Lauter, and M. Kim, "Empowering artificial intelligence with homomorphic encryption for secure deep reinforcement learning," 2025.

[18] A. Ebel, K. Garimella, and B. 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*, 2025, pp. 734–749.

[19] L. Zhang, X. Wang, J. J. Sim, Z. Huang, J. Zhong, H. WANG, P. Duan, and K.-Y. Lam, "MOAI: Module-optimizing architecture for non-interactive secure transformer inference," in *The Fourteenth International Conference on Learning Representations*, 2026.

[20] J. Zhang, J. Liu, X. Yang, Y. Wang, K. Chen, X. Hou, K. Ren, and X. Yang, "Secure transformer inference made non-interactive," *IACR Cryptol. ePrint Arch.*

[21] A. Stoian, J. Frery, R. Bredehoft, L. Montero, C. Kherfallah, and B. Chevallier-Mames, "Deep neural networks for encrypted inference with TFHE," in *International Symposium on Cyber Security, Cryptology, and Machine Learning*, 2023.

[22] L. Montero, J. Frery, C. Kherfallah, R. Bredehoft, and A. Stoian, "Neural network training on encrypted data with tfhe," 2024.

[23] R. Brännvall, "The inhibitor: Relu and addition-based attention for efficient transformers," *arXiv preprint arXiv:2310.02041*, 2023.

[24] D. Trama, P.-E. Clet, A. Boudguiga, and R. Sirdey, "Building blocks for LSTM homomorphic evaluation with TFHE," 2023, p. 117. [Online]. Available: https://cea.hal.science/cea-04186110

[25] I. Chillotti, M. Joye, and P. Paillier, "Programmable bootstrapping enables efficient homomorphic inference of deep neural networks," 2021.

[26] S. Selvakumar and B. Senthilkumar, "A privacy preserving machine learning framework for medical image analysis using quantized fully connected neural networks with tfhe based inference," 2025.

[27] D. Park, E. Lee, and J.-W. Lee, "Powerformer: Efficient and high-accuracy privacy-preserving language model with homomorphic encryption," in *Annual Meeting of the Association for Computational Linguistics*, 2024.

[28] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," in *Advances in Cryptology – CRYPTO 2018*, 2018.

[29] K. Cho, B. Van Merriënboer, Ç. Gulçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[30] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long short-term memory and learning-to-learn in networks of spiking neurons," *Advances in neural information processing systems*, 2018.

[31] L. Guo, P. Li, T. Gao, Y. Zhong, H. Fan, and J. Duan, "Efficient encrypted computation in convolutional spiking neural networks with tfhe," *arXiv preprint arXiv:2603.26781*, 2026.

[32] P. Li, H. Huang, T. Gao, J. Guo, and J. Duan, "Efficient privacy-preserving convolutional spiking neural networks with fhe," *arXiv preprint arXiv:2309.09025*, 2023.

[33] N. B. Njungle, E. Jahns, M. Stojkov, and M. A. Kinsy, "Privspike: Employing homomorphic encryption for private inference of deep spiking neural networks," *arXiv preprint arXiv:2510.03995*, 2025.

[34] W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, and Y. Tian, "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021.

[35] C. Lv, T. Li, W. Qiao, X. Wang, M. Wu, W. Liu, S. Dou, X. Zheng, and X. Huang, "Spikebert: A language spikformer learned from bert with knowledge distillation," *Neural Networks*, 2025.

[36] L. Ducas and D. Micciancio, "Fhew: Bootstrapping homomorphic encryption in less than a second," in *Advances in Cryptology – EUROCRYPT 2015*, 2015.

[37] K. Garimella, A. Ebel, G. De Micheli, and B. Reagen, "He-lrm: Encrypted deep learning recommendation models using fully homomorphic encryption," *arXiv preprint arXiv:2506.18150*, 2025.

[38] P. Chartier, M. Koskas, M. Lemou, and F. Mehats, "Fully Homomorphic Encryption on large integers," 2025. [Online]. Available: https://hal.science/hal-05388210

[39] T. Zhou, X. Yang, L. Liu, W. Zhang, and N. Li, "Faster bootstrapping with multiple addends," *IEEE Access*, 2018.

[40] I. Chillotti, D. Ligier, J.-B. Orfila, and S. Tap, "Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE," 2021.

[41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[42] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient homomorphic conversion between (ring) lwe ciphertexts," in *Applied Cryptography and Network Security: 19th International Conference, ACNS 2021, Kamakura, Japan, June 21–24, 2021, Proceedings, Part I*, 2021.

[43] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *North American Chapter of the Association for Computational Linguistics*, 2019.

[44] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," in *Proceedings of the 2018 EMNLP workshop BlackboxNLP: Analyzing and interpreting neural networks for NLP*, 2018.

[45] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *Journal of Mathematical Cryptology*, 2015.

[46] CrypoLab, "Heaan." [Online]. Available: https://www.cryptolab.co.kr/en/products-en/heaan-he/

Appendix A.

A.1. Cryptographic Parameters

We selected our cryptographic parameters using the Lattice Estimator [45], which is the standard tool for assessing the security level of homomorphic encryption schemes. Table 7 summarizes the parameters used for our ciphertext inference experiments.

TABLE 7: Cryptographic parameters for TFHE-based inference.

Parameter	Value
λ (Security Level)	128
$\log_2(q)$	64
N	2^{11}
$\log_2(\sigma_N)$	22.6
n	840
$\log_2(\sigma_n)$	48.6
$\log_2(\beta_x)$	56
β_{PBS}	2^{13}
ℓ_{PBS}	2
Collapse factor	3
$\log_2(\beta_{KS})$	1
ℓ_{KS}	20
$\log_2(\beta_{\text{packing}})$	2
ℓ_{packing}	20

The pairs $(\beta_{PBS}, \ell_{PBS})$, (β_{KS}, ℓ_{KS}), $(\beta_{packing}, \ell_{packing})$ correspond to the parameters of the so-called gadget decomposition for the following operations: programmable bootstrapping, key switching, and packing. Please refer e.g. to [6] for a detailed description of these standard TFHE operations.

A.2. Hardware Setup

We compared our execution times against the baseline established by [14] using their provided codebase. Their implementation relies on the HEaaN library [46], which is explicitly compiled for NVIDIA GPUs and x86 CPUs. Because our primary prototyping server (hosting the GB200) operates on an ARM-based CPU architecture, running their x86-optimized code natively was not viable. Therefore, to ensure a fair and hardware-aligned runtime evaluation, we conducted all comparative latency benchmarks on a separate server equipped with an NVIDIA H100 NVL GPU and x86 CPUs.

A.3. Noise Study

To complement the results presented in Table 3, we conducted a detailed empirical ablation study of the noise mechanisms by cumulatively injecting them into the inference pipeline.

TABLE 8: Prediction agreement between plaintext inference and various noise models. Each value indicates the percentage of matching predictions relative to the pristine baseline.

Dataset	Quantization	Quantization + Input Noise	Total Error
SST-2	94.2%	93.0%	92.9%
CoLA	93.3%	88.7%	88.5%
MRPC	99.7%	99.6%	99.5%

We distinguish the error induced by the weight quantization process from the noise introduced by the key-switching and modulus-switching operations (grouped here under "Input Noise," as these operations inject noise directly at the input of the bootstrapping process). The "Total Error" column reflects the accumulation of all error sources, namely: the quantization error, the bootstrapping input noise, and the predefined bootstrapping output noise.

Table 8 displays the results of simulating these different scenarios sequentially to isolate the impact of each mechanism. The results clearly demonstrate that quantization acts as the primary source of prediction divergence from the baseline model. The key-switching and modulus-switching noises only marginally increase this divergence, with the notable exception of the CoLA dataset. Finally, we observe that the intrinsic output noise generated by the blind rotation (bootstrapping) has a negligible impact on the network's final predictions.

A.4. Memory footprint

Although the main goal of this work is to achieve the lowest running time possible, we also study the memory footprint of this setup. To do so, we evaluate the memory size of an encrypted sequence and auxiliary keys.

A.4.1. Encrypted sequence. We estimate the memory footprint of a sequence of length T. We pack multiple timestep embeddings in a single ciphertext as the dimension of the embeddings d_{emb} is smaller than the degree N. Under this assumption the memory footprint of an encrypted sequence is :

$$\left\lceil \frac{T d_{emb}}{N} \right\rceil 2N \log_2(q)$$

with d_{emb} the dimension of the embeddings, which is 768 in this case. Under our cryptographic parameters, an encrypted sequence weights 1.6MB. For comparison, we compute the memory footprint using the CKKS parameters from [14], which gives 19.1MB. The main differences between both schemes are the degree N and the modulus q. In standard CKKS parameter set, N equals 2^{16} and $\log_2(q)$ is around 780. This comparison emphasizes one of the advantages of TFHE as the ciphertexts are much smaller than in CKKS.

A.4.2. Bootstrapping key. In addition to the encrypted sequence, the server requires a bootstrapping key in order to perform PBS. The memory size of the bootstrapping key is as follows:

$$\frac{4nN\ell_{PBS}(2^m - 1)\log_2(q)}{m}$$

where m is the collapse factor [38], [39]. According to cryptographic parameters of Table 7, the size of the bootstrapping key is about 257 MB.

A.4.3. Packing keys. As mentioned above, we perform the trace-based packing technique. Thus, the size of the packing key is:

$$\log_2(N) 2N\ell_{packing} \log_2(q)$$

According to this formula, the packing key size is 7.2MB.

A.4.4. Key switching key. A key switching is performed previously to PBS in order to reduce the secret key size. This operation uses a key switching key. The memory cost of this auxiliary key is as follows:

$$nN\ell_{KS} \log_2(q)$$

When applying this to parameters of Table 7, the key switching key size is 275.3MB.

A.5. Inference latency scaling

We investigate in this part how our architecture inference time scales with respect to the hidden dimension and the sequence length.

TABLE 9: Running time as a function of the sequence length T, with hidden dimension $d = 256$.

$T = 64$	$T = 128$	$T = 256$	$T = 512$
2.0 s	3.8 s	7.5 s	15.0 s

TABLE 10: Running time as a function of the hidden dimension d, with sequence length $T = 128$.

$d = 64$	$d = 128$	$d = 256$	$d = 512$
1.3 s	2.2 s	3.8 s	7.0 s

Tables 9 and 10 present how the running time evolves. We notice that the inference latency increases linearly with the sequence length. This scaling is the result of the recurrent architecture. One should expect a similar result for the hidden dimension scaling. However, we performed our evaluation on GPU. Thus, the latency growth with respect to the hidden dimension is slightly sub-linear as GPU performs PBS in batch, achieving a better scaling.