Optimizing Machine Learning Inference and Training Workloads in Fully Homomorphic Encryption

Trevor Leong

CMU-CS-24-138

August 2024

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Wenting Zheng, Chair
Lujo Bauer

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science.

Keywords: Fully Homomorphic Encryption, Privacy-Preserving Machine Learning, Transformers, Private Information Retrieval

For the professors, friends, and family that have supported me thus far.

Abstract

FHE (Fully Homomorphic Encryption) enables computation over encrypted data without revealing plaintext inputs. This property allows clients to outsource computation to servers without revealing their inputs. A notable application of FHE is in Privacy-Preserving Machine Learning as a Service (MLaaS), which enables clients to submit data to a server-hosted machine learning model and receive processed results while maintaining data confidentiality.

However, the practical implementation of FHE in evaluating machine learning models remains challenging. The restricted set of operations permissible under FHE presents a significant hurdle to implentation. This is further compounded by the significant performance overhead of each FHE operation compared to its plaintext counterpart. Computing nonlinear functions like softmax requires complex polynomial approximations. Additionally, even FHE-compatible operations like matrix multiplication take considerable time.

This thesis addresses the performance and security constraints associated with using FHE to evaluate machine learning models. First, I propose a novel application of a softmax approximation for evaluation in FHE that leads to a $4\times$ reduction in latency. Then, I describe a procedure for evaluating the embedding layer on the server without the client learning the model's embedding matrix, achieving a $5\times$ speedup over the naive approach. Lastly, I optimize the HELR algorithm for an in-house hardware accelerator by modifying rescale and bootstrap placement, significantly reducing the number of bootstraps.

Acknowledgments

I would like to start my acknowledgments by thanking my thesis committee. Both Professor Bauer and Professor Zheng were instrumental throughout my journey CMU. It is no exaggeration to say that I would not be where I am today without Professor Zheng. From introducing me to research at the end of my junior year to working with me to find an appropriate thesis topic, she has made my journey challenging, yet engaging. She taught me that while research requires a bit of elbow grease, it can provide fulfillment unlike any other activity. Additionally, I would like to thank Professor Bauer for introducing me to the field of computer security, providing advice, and answering questions as my undergraduate advisor.

I would now like to thank the Cryptosystems research group, Sid, Edward, Qi, Andrew, and Will for assisting all across the spectrum. From taking the time to onboard me onto their research projects to giving me feedback on presentations and simply giving me someone to relate to about research-related issues, they have been a crucial part of this research.

To wrap up these acknowledgments, I would like to thank my mother, my father, and my sister for raising me and supporting me. They were always there to help me, whether it be by instilling values of discipline and perseverance or simply being there to listen to my problems when I felt overwhelmed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The unprecedented performance of AlexNet in 2012 showcased the capabilities of ML (machine learning) models [28] by outperforming the best-known existing solutions in the ImageNet competition. Since then, cutting-edge ML techniques have revolutionized numerous fields, such as translation [22] and game-playing [17, 46], consistently surpassing state-of-the-art solutions and even humans with remarkable ease. More recently, transformer-based Large Language Models (LLMs) have enabled an even more impressive set of use cases such as answering medical questions [3], generating code [41], and even financial modeling [47].

Many companies have found success in training and hosting proprietary ML models. They offer Machine Learning as a Service(MLaaS) where users can send their inputs to be evaluated by a proprietary ML model hosted on their servers. ChatGPT is a popular example of MLaaS, exposing an API that allows users to evaluate text inputs on its proprietary models [4].

From a privacy perspective, both MLaaS providers and the users of these services face a significant dilemma. If a service refuses to give up its model, users must upload their queries to the service in plain text, risking the exposure of confidential information. Conversely, if users are unwilling to share their queries, services must disclose their models, risking financial losses due to potential unauthorized copying and distribution.

MLaaS providers such as ChatGPT are often not willing to shoulder the risks associated with model leakage and require users to send queries to their servers. In these cases, the service's model is protected, but the user's privacy remains at risk [4]. For this reason, countries like Italy and companies like Samsung are going so far as to outright ban the service [6, 7]. Protecting user privacy is not a simple task. Any solution that aims to protect user privacy necessitates that computations be performed over the user's input in a manner that does not reveal the input.

We can use a cryptographic primitive known as FHE(Fully Homomorphic Encryption) to perform computations subject to the above constraints [16]. Using FHE, any party with a user's public key can compute using ciphertexts encrypted by the user. Put simply, if we have two ciphertexts $c_1 = Enc(k, a), c_2 = Enc(k, b)$ and a public key, FHE allows us to apply an operation $\oplus$ on $c1, c2$ such that the result is a ciphertext encrypting a similar operation, like addition, over the plaintext inputs: $c_1 \oplus c_2 = Enc(k, a) \oplus Enc(k, b) = Enc(k, a + b)$. Using similar operations, clients can outsource ML model evaluations to the server by sending a public key with the encrypted inputs.

Despite fulfilling all of our security requirements, FHE is not without its drawbacks. The

limited computations FHE is able be perform serve as a barrier to adoption. Even worse, the operations that are computable in FHE incur a large performance overhead over their plaintext counterparts. As such, naive implementations attempting to apply FHE to evaluate ML models can lead to runtime overheads that render them impractical. Therefore, recent works have developed solutions to optimize computations that can be computed in FHE and efficiently approximate those that cannot.

Many existing solutions address issues with older neural net architectures such as Convolutional Neural Networks (CNNs) [29]. Unfortunately, these techniques do not apply to recent transformer-based models, which require approximations for non-linear functions not found in CNNs, and efficient Matrix Multiplication operations for the larger matrices involved. The majority of the works that propose non-linear approximations suitable for transformers substitute "FHE-friendly" activation functions, which require an expensive model retraining or distillation process [10, 49]. Finally, to the best of our knowledge, the only existing work that fully evaluates transformers in FHE without any changes, NEXUS, suffers from a large runtime overhead [48].

1.1 Our contributions

This thesis aims to create performant and secure solutions for executing ML training and inference workloads in FHE. It addresses the challenges in privacy-preserving MLaaS, focusing on the computational overhead associated with FHE. It applies our techniques to the HELR [20] and NEXUS [48] algorithms to illustrate how our techniques reduce evaluation time for ML workloads in FHE. The first contribution of this thesis applies a novel softmax approximation that can achieve a significantly reduced runtime for transformer-based ML models. Next, it leverages techniques used by Private Information Retrieval (PIR) schemes to hide the embedding layer in transformer-based models at almost no cost to the end-to-end runtime. Finally, it ends by showing how existing algorithms can be adapted for custom accelerators by manually placing bootstrap and rescale operations to drastically cut the number of bootstraps required over the naive placement.

Reducing latency incurred by evaluation of the softmax operation in FHE

The first problem we address is the excessive runtime caused by softmax evaluation. In every FHE scheme, expensive bootstrap operations must be performed for every few multiplication operations. Softmax evaluation in FHE requires many multiplications, leading to numerous expensive bootstrap operations. We introduce a relaxation to the evaluation of softmax, which requires much fewer multiplication operations and fewer bootstraps, ultimately leading to a $1.3\times$ reduction in runtime when applied to the NEXUS framework.

Applying PIR techniques for faster server-side evaluation of the embedding layer

The next challenge we address is to efficiently evaluate the embedding layer of transformer-based models in FHE. Inputs to transformer-based models evaluated in FHE are often assumed to be pre-embedded. This means that the client learns one part of the server model. While not directly

threatening, this makes it easier for the client to potentially perform a model stealing attack, as it no longer has to steal the embedding layer. While many existing protocols provide a means of performing this operation, it is not practical, with NEXUS's solution taking nearly $4\times$ the total runtime of FHE evaluation. We find that the embedding layer computations are similar to the problem of problems faced by PIR. Accordingly, the embedding process requires the client query to be hidden from the server just as we require the embedding index to be hidden from the server. By applying techniques introduced by PIR to the evaluation of the embedding layer in the NEXUS framework, we achieve a $5\times$ speedup.

Optimizing HELR for a custom hardware accelerator

The final challenge we face is determining how to adapt existing FHE-based ML model evaluation for a custom hardware accelerator. While there have been several works introducing compilers that aim to solve this very issue, there is often no backend support for custom FHE hardware accelerators. As a result, bootstraps and rescales must be manually placed. Even worse, the bootstrap placement used by algorithms introduced in prior works cannot be used, as the accelerator runtime uses a different bootstrap algorithm. We use HELR as an example, noting that naive placement results in a large runtime blowup over the results reported in the paper. To rectify this, we modify the bootstrap and rescale placement to reduce the total number of bootstraps and bring down the runtime to a level suitable for benchmarking.

Chapter 2

Background

2.1 Privacy Risks in Machine Learning

Confidentiality is crucial for ML applications concerning areas such as healthcare and finance. Queries regarding these areas can reveal sensitive information. As an example, consider what happens when a client sends a query such as: "What are the symptoms of COVID?". Although not directly stated, the server can infer that a client, or someone they know, has COVID-like symptoms. Further, there is evidence that services like ChatGPT deal with sensitive information, with some studies showing that people use ML applications such as ChatGPT for writing assistance of personal documents such as emails [23]. To make matters worse, ML is increasingly being used for developing personal assistants, some of which use user-uploaded documents when answering questions [2]. Privacy policies can mitigate potential misuse of sensitive information such as those revealed in personal essays or documents uploaded to cloud-based storage, but ultimately only serve as a deterrent and cannot guarantee that information users send to MLaaS services is not exploited.

A tangential, but widely studied concern is whether training data can be extracted using only model weights or access to a query-based API. Many attacks and defenses have been developed around this issue, with model inversion attacks being used to access training data and membership inference attacks aiming to determine whether a given sample is a part of the training dataset [37]. Unfortunately, FHE does not protect against these classes of attacks.

2.2 RNS-CKKS encryption scheme

The FHE scheme that we primarily deal with in this thesis is the RNS-CKKS scheme, which supports arithmetic operations on encrypted data over real or complex numbers [11]. RNS-CKKS supports Single-Instruction Multiple-Data (SIMD) operations over vectors of real or complex numbers. The available SIMD operations are addition, multiplication, and conjugation. Rotation is also supported, but it is not element-wise like the other operations. Ciphertexts in RNS-CKKS have a multiplicative depth, more commonly referred to as a "level", set to some number L, meaning that up to L multiplications can be performed without consequence. The workflow of computation in the FHE scheme can be seen in figure 2.1.

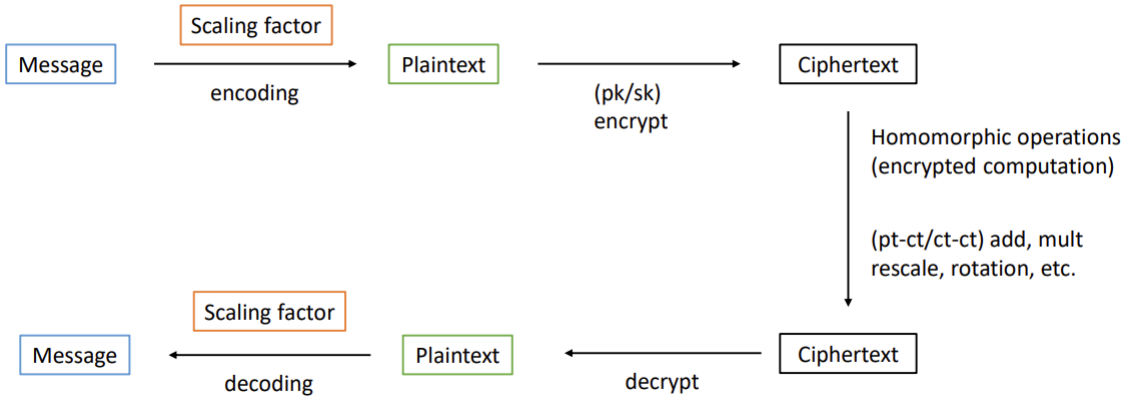

Figure 2.1: A diagram depicting the RNS-CKKS workflow from [5]

The security of the RNS-CKKS scheme is based on the Ring Learning With Errors (RLWE) hardness assumption, which is based on problems involving arithmetic over polynomials. To encrypt a given plaintext μ with secret key polynomial s into a ciphertext, RNS-CKKS first samples random polynomials a, e and returns the following pair of polynomials $(-a \cdot s + \mu + e, a)$, where e is a small noise polynomial. Note that real and complex number messages must first be encoded into a polynomial μ before being encrypted in the RNS-CKKS scheme.

Central to the security of RNS-CKKS are the "noise" polynomials of the ciphertext. Decryption in RLWE schemes relies on keeping this noise small enough such that it can be removed without adversely affecting the original value. Performing homomorphic operations increases the noise in a ciphertext. If the noise grows too large, decryption becomes impossible. To reduce noise, rescale operations can be performed at the cost of a level.

The amount of rescales performed determines what level a ciphertext is at. In RNS-CKKS, up to L rescales can be performed. Once a ciphertext's levels decrease below a certain threshold, an operation known as bootstrapping is required. Intuitively, bootstrapping performs decryption and encryption using homomorphic operations, resulting in a fresh ciphertext with L levels remaining. It is important to note that bootstrapping itself consumes some number of levels K, which means only $L - K$ levels are available for computation. Thus, the bootstrapping threshold mentioned above is K.

2.3 Transformer architecture

Most models based on the transformer architecture share the same basic structure. The transformer architecture is most well known for its usage of the attention mechanism, which enables it to capture long-range dependencies between tokens without significant computation overhead. This ability to capture long-range dependencies has proved to be invaluable in many settings such as Natural Language Processing, enabling it to match and even surpass humans in some scenarios [14, 40].

As shown in Figure 2.2, the input to a transformer is a vector of words. This vector is first

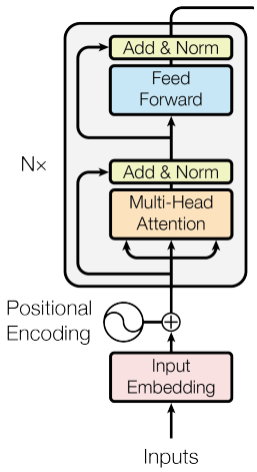

Figure 2.2: A depiction of a transformer as shown in [45]

tokenized using vocabulary and position information into a vector of n tokens. Then, each token is mapped to a vector of size d_k. From there, it is passed to an attention layer and feed-forward network. The output of each layer is fed into a layer norm operation. The attention layer, feed-forward and LayerNorm operations constitute a transformer block. Transformer-based models consist of multiple blocks stacked together, where the output of each feed-forward layer is fed into the attention layer of the next block. The final block outputs a value d_k for each of the n tokens fed into it.

Attention Layer

The attention layer tasks as input an embedding matrix $A \in \mathbb{R}^{n \times d_k}$. It Multiplies it by three learned matrices of weights $W_K, W_Q, W_V \in \mathbb{R}^{d_k \times d_k}$ to form $Q, K, V \in \mathbb{R}^{n \times d_k}$ respectively. From there, the following is evaluated:

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{d_k})V$$

Layer Normalization

LayerNorm is performed after the attention and feed-forward layers. It takes as input a sample: $X \in \mathbb{R}^m$ and is defined as follows:

$$LayerNorm(X)_i = \gamma \times \frac{X_i - \mu}{\sqrt{\sigma + \epsilon}} + \beta$$

Where $\mu = \frac{1}{m}\sum_{i=0}^{m-1} X_i$, $\sigma = \sqrt{\frac{1}{m}\sum_{i=0}^{m-1}(X_i - \mu)^2}$ and β, γ are hyperparameters.

Feed-Forward Layer

The feed-forward layer takes as input the result of the attention layer after it has been passed through a layer norm operation $A \in \mathbb{R}^{n \times d_k}$ consisting of matrix multiplication by a learned weight matrix $W_1 \in \mathbb{R}^{d_k \times d_{hidden}}$, then passed through a GELU activation function, and finally another matrix multiplication by a learned weight matrix $W_2 \in \mathbb{R}^{d_{hidden} \times d_k}$. It is formulated as follows:

$$FeedForward(X) = GELU(XW_1)W_2$$

2.4 Private Information Retrieval

In this thesis, we use PIR techniques to ensure that client embeddings can be safely retrieved without the server learning which word the client wants to embed. Private Information Retrieval (PIR) is a cryptographic method that allows users to query a database and retrieve specific items without disclosing which items they are accessing to the database server. PIR schemes involve a client and a server, or multiple servers which host a database, or parts of a database. The client sends a query to the server for a given item. The server or servers will process the query and return a result.

Any PIR scheme must satisfy correctness and privacy guarantees. In other words, the client must receive the correct result and the server or servers must not be able to figure out which database entry the client was querying for. A visualization of this can be seen in figure 2.3.

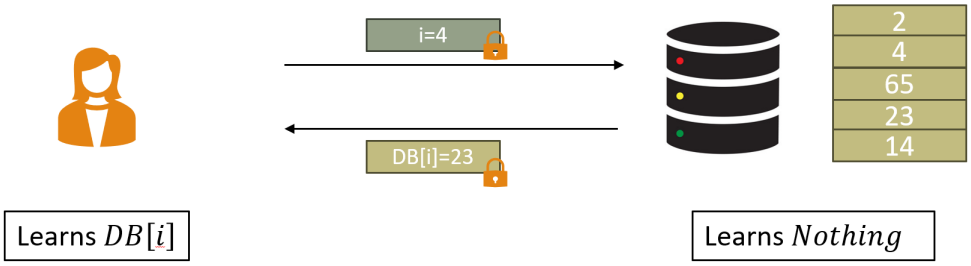

Figure 2.3: A diagram showing how PIR schemes work

There have been two main approaches to developing PIR solutions. Single-server PIR schemes assume there is only one server that is trusted and thus must handle the entirety of the client's request. Multi-server schemes are generally faster and require less storage but require assumptions about collusion and can be bottlenecked by servers that are slow to respond. Thus, Single-server schemes remain an area of interest. Single-server schemes, however, are subject to the fundamental limitation of PIR in that the entire database must be scanned. The state-of-the-art schemes using lattice-based schemes have made great strides in amortizing this cost across multiple inputs, using the properties of lattice-based schemes to compress queries as well as perform multiple queries at once using batch PIR techniques alongside FHE schemes like BFV [36].

Chapter 3

Related Work

3.1 Privacy-Preserving Machine Learning

The field of PPML (Privacy-Preserving Machine Learning) marries existing cryptographic primitives for secure computation with the computations required to train and evaluate models in a privacy-preserving manner. Although this work primarily focuses on FHE-based schemes, we compare them with alternative approaches to better understand the trade-offs involved.

3.1.1 Secure Multi-Party Computation approaches

Secure Multi-Party Computation (SMPC), more commonly referred to as MPC, approaches enable n parties with n inputs $x_1 \cdots x_n$ to compute a function $f(x_1, \cdots x_n)$ without any individual party finding about the others' inputs. Early attempts such as SecureML and MiniONN, used MPC techniques to train and perform inference to evaluate neural networks [33, 35]. [24] observed that HE could be used alongside MPC to evaluate communication-heavy vector-matrix multiplications. This hybrid approach turned out to be rather fruitful, with a $40\times$ speedup over MiniONN when evaluated on the CIFAR-10 dataset. The current state-of-the-art, BOLT builds on these approaches to evaluate more complex transformers with more efficient packing schemes and faster non-linear function approximations [38]. Other approaches such as MPCFormer and Puma opt to assume large bandwidth settings and use pure MPC approaches[15, 32]. Although MPC and hybrid approaches achieve the best latencies of all PPML solutions, they require fast network conditions to work well.

3.1.2 Fully Homomorphic Encryption approaches

Existing approaches primarily leverage the ability of RNS-CKKS to compute over real numbers to evaluate ML models in FHE. This is not a trivial task due to the limitations of the available operations in the RNS-CKKS scheme. Firstly, the operations that are available in FHE: addition, multiplication, and rotation can be used to evaluate dot products and matrix multiplications, but there are many non-linear functions such as max that it cannot. Secondly, matrix multiplication, which is an essential operation, must be done with care. The reasoning behind this is that some operations incur large runtimes when the input matrix is not packed correctly. Often, this requires

repacking the matrix, which is difficult because only SIMD operations are supported, so one cannot directly index into individual vector slots. Further, any data movement requires expensive rotation operations. Finally, bootstrapping is inevitable for deep computations such as those involved in large transformer models and can result in a large runtime blowup if placed naively.

Now that we have explained the challenges involved with FHE implementations, we can now introduce how existing solutions address each of these issues.

HELR: Efficient Logistic Regression on Large Encrypted Data

The authors of HELR observed that many prior schemes did not scale very well, and were either trained with extremely small datasets or only used for inference purposes [20]. The main reason for the poor performance is the unsuitability of the HE schemes being used. Before the HELR, no other works used the CKKS scheme.

However, implementing logistic regression using CKKS remains challenging due to the restricted set of permissible operations and the significant performance overhead of FHE operations compared to their plaintext counterparts. Firstly, there is the issue of finding an appropriate polynomial to approximate the Sigmoid activation function so that it can be evaluated in FHE. To solve this, the authors use a least squares fitting polynomial with degree 3, which can achieve a much smaller error than the Taylor expansion polynomial. Secondly, there is the issue of minimizing the multiplicative depth to also reduce the number of bootstraps. The authors were able to cut down the multiplicative depth of each gradient descent iteration from 8 to 5, decreasing the number of bootstraps from 67 to 40.

Privacy Preserving Machine Learning with Fully Homomorphic Encryption for Deep Neural Net

However, for deeper neural nets such as ResNet, evaluating RELU and convolution operations using existing methods is difficult to perform in FHE and consumes much more multiplicative depth than previously implemented ML models. Additionally, bootstrap placement presents another issue, as a bad placement is compounded over multiple layers. [30] was able to address these issues by developing solutions for efficiently evaluating all three of these troublesome functions. For RELU, they use a composite polynomial developed incite optimal comparison in FHE paper. For evaluating convolutions, they modify the strided convolution scheme in [24] to pack the values in such a way that is natural to the RNS-CKKS scheme. Finally, they were able to reduce runtime by $1.38\times$ by placing bootstraps after convolution operations instead of after RELU operations. However, even with all of these optimizations, it still takes 3 hours to infer a single image [30].

Transformer-based models

The most recent FHE-based ML schemes aim to evaluate transformer-based models. However, this requires approximations for non-linear functions such as softmax, inverse square root, and GELU. Prior works either don't have approximations or don't result in reasonable accuracy.

Many existing solutions address these issue by substituting FHE-friendly polynomial approximations for non-linear functions in the original model and retraining or distilling the resulting model to achieve the same accuracy as the original.

THE-X: Privacy-preserving transformer inference with homomorphic encryption

THE-X [10] is one such solution using fine-tuning and distillation. The two core ideas behind THE-X are to find friendly approximations for activation functions and to offload compute to the user device for non-linear functions that cannot be easily computed like max. The FHE-friendly approximations for the non-linear functions require a distillation step and a fine-tuning step before they can be used for inference in FHE. The resulting model after fine-tuning and distillation is fairly accurate, with only a 1.9% accuracy drop in F1 score compared to the original plaintext model. However, distillation is not desirable due to the large amount of training data and computing involved. Additionally, offloading computations to the user may leak information about the model.

NEXUS(Secure Transformer Inference made non-interactive)

The state-of-the-art transformer evaluation in FHE that requires neither retraining nor distilling is NEXUS [48]. As mentioned in the introduction, the NEXUS framework introduced the first non-interactive protocol for transformer inference. Unlike previous schemes for Transformer inference, it did not use any MPC techniques, opting to use purely FHE for evaluation. The NEXUS framework begins with the client embedding the input. It then encrypts this result to hide it from the server and sends it to the server. The server then evaluates the transformer model using FHE operations. Once complete, the server sends back the response to the client. The client then decrypts the result and learns the result of their input when evaluated on the server's model. Note that the only times the client and server communicate are when the client initially sends their input and when the server sends back a response making it non-interactive.

In most network settings, the performance of NEXUS falls behind the current state-of-the-art solution, BOLT [38]. However, NEUXS shines in poor network conditions, as it removes almost all of the communication costs incurred by the operations in BOLT performed using MPC. This allows it to exceed the performance of BOLT under poor network conditions such as over a wide area network (WAN).

The main optimization introduced by NEXUS is a novel batching technique, which enables a preprocessing phase that allows a matrix multiplication to occur without expensive rotation operations. For the non-linear layers, the authors largely use approximations from prior works such as a set of composite polynomials for the sign function, Newton iteration for the inverse square root, and the Goldschmidt division algorithm for computing the inverse. Additionally, the authors set the ciphertext levels $L = 35$ and bootstrapping level consumption to $K = 15$, meaning they use $L - K = 21$ levels for computation. This gives NEXUS enough levels to be able to evaluate all layers except for attention without bootstrapping within the layer.

Chapter 4

Methods

We design two cryptographic protocols for both ML inference and training workloads. The first protocol builds on the NEXUS protocol introduced in section 3. The second protocol adapts the HELR algorithm for an in-house FHE runtime. Initially, we illustrate how our approach reduces latency by proposing an alternative softmax function. Subsequently, we detail the evaluation of the embedding layer on the server, achieving this with minimal overhead. Finally, we shift focus to the second protocol, outlining how we optimize bootstrap and rescale operations to better align with the specific requirements of our environment.

4.1 Reducing softmax level consumption in NEXUS

This section describes our approach to reducing the computational overhead associated with evaluating the softmax function in the NEXUS framework, focusing on minimizing level consumption. Reducing the levels required for softmax evaluation is crucial for improving performance, as it directly impacts the speed and efficiency of the NEXUS framework. This is made clear in figure 4.1, which illustrates that the majority of the runtime is consumed by bootstrapping and softmax evaluation.

The state-of-art bootstrapping procedure is highly optimized, and more speedup will likely come from designing specialized hardware accelerators and finding optimal bootstrap placement [19, 25, 42]. Knowing this, we choose to improve the softmax evaluation, which currently consumes 143 levels for each transformer block. To put this into perspective, we can compare its runtime to that of LayerNorm. The level consumption of an individual LayerNorm operation is 20. Since there are two of them for each transformer block, they consume a total of 40 levels per transformer block. However, as can be seen in figure 4.1, the percentage of runtime LayerNorm occupied is less than the amount we would expect if level consumption were the only determining factor. The runtime disparity is primarily explained by the bootstrap operations that must be performed within the layer. Recall from section 3 that NEXUS has $L - K = 21$ levels available for computation, so bootstrapping is not required within LayerNorm, but is required multiple times within softmax.

The equation for softmax is:

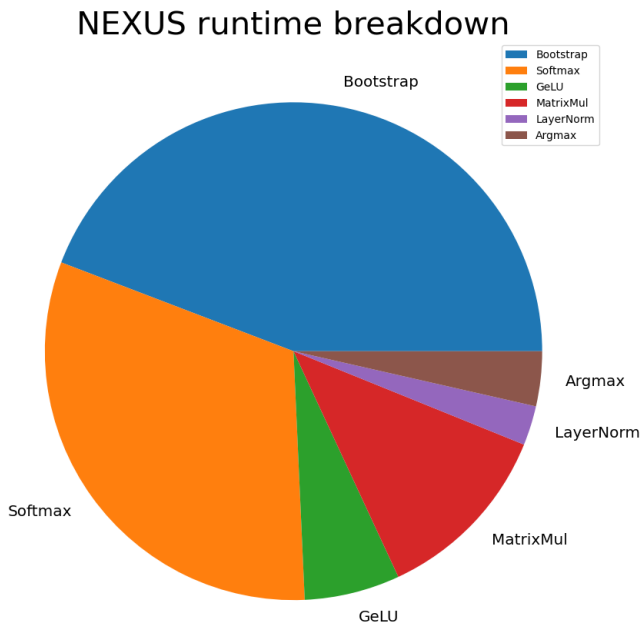

Figure 4.1: Pie chart displaying the percentage of time the evaluation of each layer contributes to the end-to-end runtime

$$Softmax(X)_i = \frac{e^{x_i}}{\sum_i e^{x_i}}$$

However, it is more commonly computed as follows:

$$Softmax(X)_i = \frac{e^{x_i - max(X)}}{\sum_i e^{x_i - max(X)}}$$

The reasoning behind subtracting our the max is that it ensures that the result does not grow too large or too small. It prevents overflow by ensuring that the maximum value in the numerator is 1, and the denominator is at least 1. Therefore, any solution that replaces the max in softmax must also be careful to ensure that overflow and underflow are managed properly.

Softmax's computation overhead mainly comes from the approximation of max:

$$max(a, b) = \frac{a + b - (sign(a - b) * (a - b)}{2}$$

From the evaluation of the function, we see that the number of levels consumed is the number of levels required for the sign approximation plus two. The total level consumption of max is 18 since the approximation for sign consumes 16 levels. Recall from section 3 that the number of levels before bootstrap was 21. This means that for a single element-wise max, a bootstrap is required. Even using a folding method to reduce the number of element-wise maxes required to compute an array-wise max from n(128) to (7) still results in 7 total bootstraps.

The max operation is generally expensive to compute, as it requires comparison with other array elements. In many settings, this requires moving elements, which is often more expensive than direct computation. With this in mind, there have been several attempts at alternate softmax formulations that substitute the full max operation with a term that is less expensive to compute.One such recent work observed that the max in the numerator and the sum in the denominator could be replaced by a sufficiently high constant [34]. Their final approximation was:

$$Softmax(X)_i \approx Consmax(X)_i = \frac{e^{x_i - \beta}}{\gamma}$$

Here, the max is completely removed, potentially reducing the total level consumption down to 4 when computed in CKKS. However, to achieve acceptable performance with this modified softmax, the authors trained a GPT-2 model from scratch.

However, training from scratch is a very expensive operation, as it requires a large amount of training data, computing a gradient over the results, and propagating the changes through the entire model. The original Consmax work required 20000 iterations to achieve comparable performance to the original, unmodified model. Additionally, one of the main advantages of NEXUS is that it can evaluate models without modification. So, adding a training requirement would eliminate this benefit.

As a logical next step, we used fine-tuning to determine if a suitable γ and β could be found without complete retraining. Fine-tuning is generally less expensive than training, as its weight initialization is informed by prior training steps, and therefore takes less time to achieve acceptable performance. Although this solution would require some modification to the model, it would still be one step in the right direction regarding the goal of being able to adapt models into FHE with minimal modification.

To accomplish this, we followed a similar methodology to the Consmax authors, randomly initializing the β and γ parameters at each end to the parameters at each layer and running a few warm-up iterations to determine which gave the best performance. Unfortunately, the overflow/underflow issues were still present, causing gradients to overflow to infinity, or simply become 0. This was primarily because the constants could not adequately protect against overflow and underflow as the max function did. The reason is that when $\gamma >> e^{x-\beta}$, then the result would overflow, while on the flip side, if $\gamma << e^{x-\beta}$, the result would underflow.

The main issue with the prior approach lay in the inability of the γ and β parameters to perform the normalization functionality of max properly. During the training attempts, it was observed that this was primarily due to the value of γ not properly adjusting when β made the numerator too large or too small. Combining this with the fact that the Goldschmidt division algorithm used for computing the inverse was only a fraction of the contributions of max led us to consider finding a middle ground between approximations.

With this reasoning, our insight was to keep the sum in the denominator while replacing the max with β. In the new formulation, the sum serves as a safeguard to ensure that even if the β parameter was too large or too small, the sum would be able to dynamically adjust itself and ensure proper normalization. This also accomplishes our original goal of reducing level consumption, as it can eliminate our dependency on evaluating the max function. If the β had been replaced instead, it is unclear whether it could have accounted for cases where the max is

much larger than the rest of the elements, nor does it save many levels. Proper initialization is still required to ensure that the numerator doesn't overflow, but accuracy results show that it is a lot less dependent than the original formulation using γ in the denominator.

4.2 A faster and more secure embedding procedure

NEXUS, like many other PPML schemes, assumes that the model architecture is public and that the client can embed their inputs easily. Both assumptions make model stealing attacks easier to conduct for malicious clients. While it is true that there are many high-performing open-source models whose architecture is readily available[8, 21, 44], some companies believe that the architecture of their models is the main aspect driving performance and thus seek to hide their model architectures completely [1, 4]. While many MPC solutions require the architecture to be known, FHE-based solutions do not, as they are non-interactive, and thus the user should only have to send input once and receive output without the end-user knowing the computations required to compute the model.

Unfortunately, the proposed NEXUS solution is not compatible with performing the embedding on the server because it assumes that the inputs are pre-embedded. For this to happen, the client has to perform the embedding step themselves, which would require access to the embedding matrix. We propose hiding the underlying architecture in the NEXUS framework by moving the embedding step to the server. The embedding layer multiplies one-hot vectors, which encodes input words, by an embedding matrix containing the word embeddings. This can be seen in 4.2, where the one-hot vector encoding word i serves to select the contents of the i^{th} word embedding.

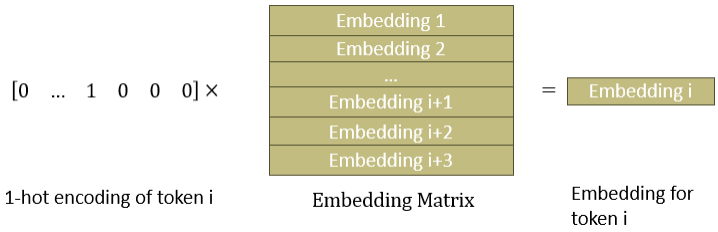

Figure 4.2: A diagram of the embedding process for the i^{th} token

These operations can be evaluated in RNS-CCKS using the batched version of the attention projection matrix multiplications proposed by NEXUS. However, the embedding matrix can grow very large since it scales with model vocabulary size. For instance, the vocabulary size of GPT-2 is 50257, meaning that the embedding process becomes a $[128 \times 50257] \times [50257 \times 768]$ matrix multiplication. The original batch matrix multiplication for matrices of size $[128 \times 768] \times [768 \times 768]$ already took 65 seconds. Performing this on a matrix $65\times$ as large results in a large performance blowup even when amortized over multiple inputs.

In ML libraries such Pytorch [39] the embedding layer is implemented as a lookup table, where instead of 1-hot encoding each of the tokens, a simple index into the transpose of the embedding matrix is performed. This does not require performing many vector operations, only

a memory lookup, and is thus much faster. Unfortunately, as mentioned in section 2, this cannot be done directly in FHE, since one cannot directly index into vector slots. Therefore, the large matrix multiplication with the one-hot vectors appears to be a problem to optimize. While it appears we are back at square one, we now have an alternate view of the problem, namely that of hiding a client query from the server.

We observe that this problem can be addressed using PIR techniques. As was explained in section 2, PIR schemes are generally classified as single-server or multi-server schemes. In our non-interactive scenario, it does not make sense to add additional servers simply for an embedding layer, so single-server solutions are the closest to our scenario. However, even within the class of single server schemes, there are many decisions that we must consider when determining applicability to performing the embedding layer.

An ideal PIR solution for this scenario should not require additional functionality and as an additional constraint, should not add too much computational or communication complexity to the end-to-end runtime of NEXUS. The first constraint limits our search to FHE-based schemes, as the existing NEXUS framework only uses FHE operations. Within this space, we ideally want a scheme that can take advantage of batching similar to the way NEXUS batches inputs. In regards to the second constraint, the target scheme should have a limited communication complexity. With minimal added communication, NEXUS would continue to have an advantage in settings where communication is the limiting factor.

These schemes lead us to use techniques from Vectorized Batch PIR [36]. As a Batch PIR scheme, Vectorized Batch PIR aims to process multiple queries efficiently, effectively amortizing the large linear scan costs across multiple inputs. We observe that the techniques used in Vectorized Batch PIR to reduce the large matrix multiplication costs incurred by 1-hot vector encodings can be used to also improve the performance of the embedding process within the NEXUS framework.

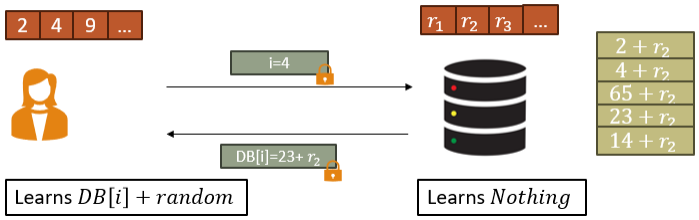

Figure 4.3: A diagram of the proposed embedding process

While Vectorized Batch PIR is perfect for protecting client queries, it does not protect the server's embeddings, leaving us almost where we started. However, this problem can be solved by adding a vector of random numbers to the server's response to the client. This effectively serves as a one-time pad and is very lightweight in performance since the only operation required is addition. When the client has all the embeddings, and sends the packed embedded inputs to the server, the server can then subtract the random masks from the client's inputs.

Figure 4.3 details what happens when a client wants to retrieve an embedding for their second element. The client first sends over a query using techniques from Vectorized Batch PIR. In response, the server samples a random vector and adds it to all database entries. Then, it runs

the Vectorized Batch PIR response protocol and sends the result back to the client. The server learns nothing by the security of the underlying PIR protocol, and the client cannot distinguish the values of the embeddings from random values.

4.3 Modifications to bootstrap and rescale placement to HELR when running on an accelerator

Implementing FHE-based programs poses significant challenges, particularly regarding the precise placement of bootstraps and rescaling operations, crucial for maintaining computational efficiency. While there have been several works introducing compilers that aim to solve this very issue, there was no backend support for the custom accelerator HELR was being used as a benchmark. Unfortunately, this means that prior works cannot easily be applied to the custom accelerator. It is important to note that this is not a fundamental limitation, but one that is born from a lack of backend support for the accelerator. With time and engineering effort, backend support will improve to the point where compilers can be used to automatically adapt prior works to different settings. Until then, prior works must be adapted on a case-by-case basis.

In our case, the new accelerator backend had a different maximum number of levels before bootstrapping, rendering it incompatible with the algorithm described in HELR. Our initial attempts to implement HELR using the new backend caused a significant runtime increase compared to the results reported in the paper. We describe how we reduce the runtime increase by reconciling the difference in bootstrap placement caused by the different maximum number of levels before bootstrapping.

As stated in the background section, the HELR algorithm introduced a method for evaluating Logistic Regression in CKKS [20]. Its two main contributions were the novel sigmoid approximation and evaluation strategy for Nesterov Gradient Descent, the equation of which is shown in figure 4.4. By intelligently reusing intermediate terms, the HELR authors were able to reduce the level consumption of each iteration of Gradient Descent from 8 to 5. Accordingly, they set their bootstrap to refresh 25 levels, meaning that 5 iterations of gradient descent could be performed before bootstrapping.

$$
\begin{aligned}
w_{i+1} &= v_i - \gamma \cdot \Delta_w l(v_i) \\
v_{i+1} &= (1-\eta) \cdot w_{i+1} + \eta \cdot w_i
\end{aligned}
$$

Figure 4.4: The Nesterov gradient descent equations which HELR evaluates in CKKS to train a Logistic Regression Model. w_{i+1} updates the weights at iteration $i+1$. v_{i+1} stores a moving average of previous weight updates to help the algorithm converge faster.

Development of benchmarks like HELR for the custom accelerator was done using an in-house FHE runtime known as Butterscotch. Butterscotch effectively emulated the custom accelerator, enabling easier program debugging and preliminary performance measures before being

run on the accelerator. The environment of the Butterscotch runtime differed from that of the HELR setting in several key aspects.

First, there was the issue of using 28-bit primes instead of the assumed 30-bit primes used by HELR. For most cases, the difference in precision that 30-bit primes provide over 28-bit primes is not necessary. However, precision loss accumulates over repeated computations, and results in subpar model performance. Second, the maximum level before bootstrap in Butterscotch is 12, unlike the 25 levels assumed in the original. Normally, this wouldn't be an issue since a compiler would do the work of adapting the program.

This brings us to the third difference, mainly that Butterscotch used a modified version of the EVA CKKS compiler [13] from Microsoft. The modified version lacked essential features such as bootstrap and rescale placement, which as mentioned earlier, can have a dramatic effect on performance.

Direct implementation of the HELR algorithm in Butterscotch results in a significant accuracy decrease from the original paper. This was the result of the precision loss described earlier. To amend this, a higher scale is required to address the accuracy decreases. Compilers such as Hecate [31] make this explicit with the concept of a waterline, which is a floor on the possible scale values, meaning that if the scale falls below the waterline, precision is lost, and the ciphertext may be corrupted little by little. With this constraint, Hecate [31] and other compilers can ensure correctness while optimizing rescales. Since the modified EVA compiler for Butterscotch lacked this feature, these had to be placed manually.

The original EVA compiler employed a greedy rescale approach, ensuring that there would always be a rescale as long as the ciphertext remained above the waterline. This worked fairly well and was easy to roll out manually. However, as a tradeoff, more levels were consumed when restoring ciphertexts with double, or even triple scale to the singular scale the rest of the ciphertexts were using.

As a result of adding extra rescale operations, each iteration of gradient descent consumed more levels than the original evaluation strategy evaluated by HELR. Combining this with the fact that there were fewer levels before bootstrap in Butterscotch meant that bootstraps could no longer be placed after each iteration as was done in the original evaluation strategy. Therefore, we now must address the challenge of placing bootstraps in a manner that is comparable to the original scheme, but now with the level constraints of the bootstrap operation provided by Butterscotch.

Now, instead of bootstrapping at the end of an iteration, bootstraps could potentially happen within the loop. This resulted in a serious runtime increase, as there were points in the middle of the loop, where multiple ciphertexts had to be bootstrapped. These "fan-out" points are places where bootstrapping is to be avoided since multiple bootstraps must be performed instead of just one. For instance, we can see this happening in figure 4.5, which illustrates what happens when we choose to bootstrap at a "fan-out" point.

To fix this, we moved the bootstrapping step to the aggregation step below it, which was a "fan-in" point or where the number of ciphertexts that would have had to be bootstrapped at that step decreased from the previous step. This is illustrated in figure 4.6, where the output of a function produces fewer ciphertexts than inputs that don't necessarily have to be bootstrapped. Although this wasted some levels during some iterations, it ensures that only one bootstrap had to be performed, resulting in a net decrease in runtime.

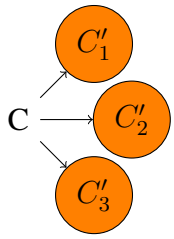

Figure 4.5: A diagram depicting a fan-out scenario where orange filling indicates a ciphertext that must be bootstrapped

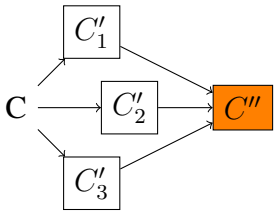

Figure 4.6: A diagram depicting a fan-in scenario where orange filling indicates a ciphertext that must be bootstrapped

Chapter 5

Results

5.1 Implementation

Similar to the original NEXUS paper, we implement our system in C++, utilizing the SEAL [43] library for RNS-CKKS homomorphic encryption and FHE-MP-CNN [30] for bootstrapping. Further, we also use HEXL [9] to accelerate SEAL on Intel CPUs. We set the multiplicative depth to $L = 35$ and the depth for bootstrapping to $K = 14$, indicating that the available multiplicative depth is $L - K = 21$.

5.2 Experimental Setup

We primarily compare these results with the original NEXUS paper. However, since the implementation was not publicly available at the time of writing, we implemented the NEXUS scheme ourselves. To enable as direct of a comparison as possible, we conducted benchmarks using similar settings:

- Experiments were run on c6i.16x large instances on EC2, with 128 GB of memory and 3rd Generation Intel Xeon Scalable processors.
- The communication bandwidth between instances was controlled using the Linux Traffic Control (tc) command.
- To evaluate the end-to-end results of our softmax relaxation, we set the bandwidth to 3 Gbps and the round-trip latency to 0.8 ms to simulate the communication in LAN.
- To evaluate the end-to-end results of our embedding procedure, we set the bandwidth to 100 Mbps and the round-trip latency to 0.8 ms to simulate WAN communication.
- The model parameters were taken from a pre-trained BERT-base transformer model [14].
- The number of threads was set to 32

5.2.1 Evaluation metrics

- Total runtime for a single softmax operation

- Total runtime for end-to-end BERT inference
- Accuracy on benchmarks with softmax relaxation
- Total runtime for end-to-end BERT inference with embedding done server-side

5.3 Softmax relaxation

5.3.1 Runtime comparison

The first evaluation here is meant to determine how much of a performance improvement over the originally proposed relaxation of Softmax can be achieved. It measured the time taken to evaluate each variation of the softmax function in seconds. As shown in figure 5.1, we can see that our proposed softmax function is about $5\times$ faster than the original formulation for softmax. We believe that this difference is largely due to the significant reduction of bootstrapping operations required as a result of the level reduction in the new softmax.

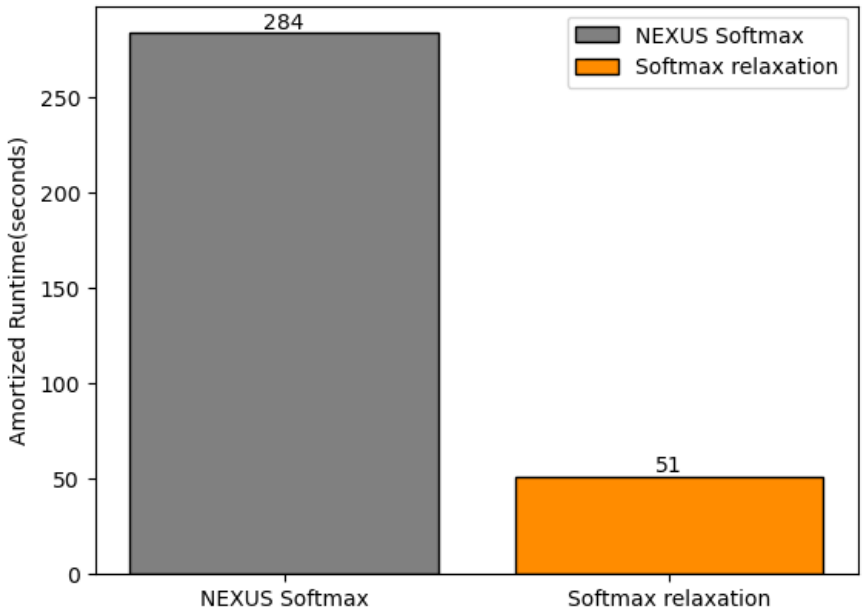

Figure 5.1: Bar graph showing the runtime of the original softmax alongside the softmax relaxation

5.4 End-to-End

5.4.1 Runtime comparison

To see how much this affected the end-to-end latency of the scheme, we evaluated a BERT-base model in the NEXUS framework with and without the new softmax approximation. As shown in

figure 5.2, the results are as expected, and our improvements resulted in an approximately $1.3\times$ speedup over the original end-to-end runtime.

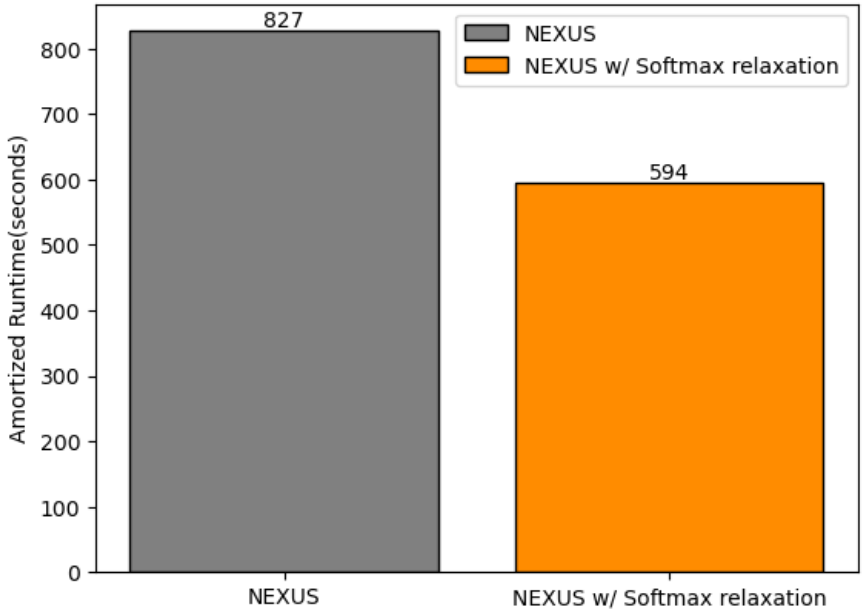

Figure 5.2: Bar graph showing the end-to-end runtime of the original NEXUS alongside the proposed version where softmax relaxation is used

5.4.2 Accuracy

To evaluate the accuracy, we used a plaintext BERT-base model implemented in Pytorch [39], substituting our softmax approximation for the softmax provided by Pytorch. An essential property for any approximation used in evaluating models in FHE is that it does not significantly degrade the performance of the model. To ensure that our softmax approximation was able to maintain accuracy while also reducing the evaluation time, we also ran benchmarks used to evaluate Language models such as BERT-base to ensure there was no significant decrease in accuracy. As shown in table 5.1, we can see that there is a $< 0.60\%$ accuracy drop for BERT-base from the NEXUS scheme. Further, there is a $< 1\%$ drop from the original plaintext evaluation of each of the models.

Model	Dataset	Plaintext	NEXUS	Modified
BERT-base	RTE	70.04 %	69.88%	69.21%

Table 5.1: Table of results showcasing the minimal drop in accuracy when evaluated on the RTE dataset

5.5 Secure Embedding

5.5.1 Runtime Comparison

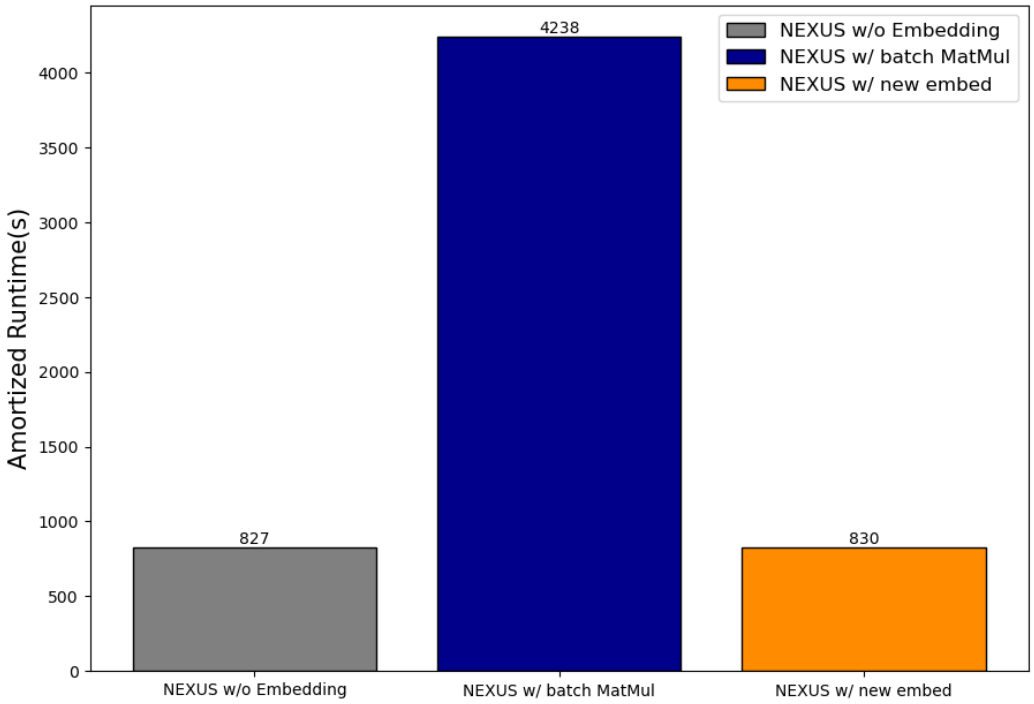

Figure 5.3: Bar graph comparing the runtimes of the NEXUS protocol in three scenarios: When the embedding is done on the client side, when the proposed technique for matrix multiplication is used for Embedding, and finally our proposed embedding procedure

To evaluate our embedding layer, we compared it to the performance of the naive scheme using the matrix multiplication techniques proposed by the original NEXUS authors to evaluate the matrix multiplication for the embedding layer. Both the modified and unmodified versions were run end-to-end. The results of these experiments are shown in figure 5.3. It can be seen that our proposed changes for matrix multiplication are much closer to the original, adding only 3 seconds to the overall evaluation. On the other hand, when using the batch matrix multiplication proposed by NEXUS, the result is much worse, with almost $5\times$ performance overhead.

Chapter 6

Conclusion

In summary, we enhance the evaluation of ML models in FHE by introducing two techniques to supplement the NEXUS framework. These techniques lower end-to-end latency and adapt the HELR benchmark to the Butterscotch runtime. For applications utilizing transformers such as medical diagnostics, these improvements will go a long way in making both service providers benefit and clients more willing to invest and use the services respectively. Service providers benefit from the option to use a faster and more secure version of NEXUS, while clients remain content that their information remains private.

Future work should explore whether our proposed softmax approximation generalizes well to other models and datasets. While the approximation worked relatively well for BERT-base, it is unclear whether it will produce reliable outputs for other models. Since it was only tested with two models and a dataset, the chosen parameter may very well be overfitting to the specific dataset and may not generalize well. If this is the case, then there is a potential research direction of whether it is possible at all to get a β value that works well across multiple datasets, or whether it can be interleaved with normal softmax to achieve reasonable results.

Another important takeaway from this thesis is the pressing need for better collaboration between compiler and accelerator development. Much of the struggles with adapting HELR for Butterscotch were since rescales and Bootstraps were not automatically placed. Although I was able to fix these issues with domain knowledge of the space, it remains an issue for wider FHE adoption and benchmark development.

These issues stem from the lack of a unified framework for representing FHE programs. Many compilers have proposed a variety of optimizations from tensor packing [27] to rescale and bootstrap placement [12]. Additionally, many works have made strides in developing expressive IR representations for different nuances of FHE [8, 18]. Unfortunately, there is no widely adopted or agreed-upon standard. As a result, recent accelerator designs [26, 42] must implement custom compilers, requiring re-implementation of existing compiler optimizations, or simply not including them at all. As a result, FHE accelerator development often takes longer than necessary. Therefore, future work should be put into designing accelerators around existing frameworks or building out compiler support for more backends to help reduce the work of re-implementing benchmarks and accelerate accelerator development.

Bibliography

[1] Cohere | The leading AI platform for enterprise, . URL https://cohere.com/. 4.2

[2] Gemini Apps Privacy Hub - Gemini Apps Help, . URL https://support.google.com/gemini/answer/13594961?hl=en. 2.1

[3] Meditron: An LLM suite for low-resource medical settings leveraging Meta Llama, . URL https://ai.meta.com/blog/llama-2-3-meditron-yale-medicine-epfl-open-source-llm/. 1

[4] OpenAI, . URL https://openai.com/. 1, 4.2

[5] Presentations, . URL https://yongsoosong.github.io/presentations/. (document), 2.1

[6] ChatGPT banned in Italy over privacy concerns. March 2023. URL https://www.bbc.com/news/technology-65139406. 1

[7] Samsung Bans Generative AI Use by Staff After ChatGPT Data Leak. *Bloomberg.com*, May 2023. URL https://www.bloomberg.com/news/articles/2023-05-02/samsung-bans-chatgpt-and-other-generative-ai-use-by-staff-after-leak. 1

[8] Song Bian, Zian Zhao, Zhou Zhang, Ran Mao, Kohei Suenaga, Yier Jin, Zhenyu Guan, and Jianwei Liu. HEIR: A Unified Representation for Cross-Scheme Compilation of Fully Homomorphic Computation, 2023. URL https://eprint.iacr.org/2023/1445. Publication info: Published elsewhere. Minor revision. NDSS 2024. 4.2, 6

[9] Fabian Boemer, Sejun Kim, Gelila Seifu, Fillipe DM de Souza, Vinodh Gopal, et al. Intel HEXL (release 1.2). https://github.com/intel/hexl, 2021. 5.1

[10] Tianyu Chen, Hangbo Bao, Shaohan Huang, Li Dong, Binxing Jiao, Daxin Jiang, Haoyi Zhou, Jianxin Li, and Furu Wei. THE-X: Privacy-Preserving Transformer Inference with Homomorphic Encryption, June 2022. URL http://arxiv.org/abs/2206.00216. arXiv:2206.00216 [cs]. 1, 3.1.2

[11] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 409–437. Springer, 2017. 2.2

[12] Seonyoung Cheon, Yongwoo Lee, Dongkwan Kim, Sunchul Jung, Taekyung Kim, Dongyoon Lee, Ju Min Lee, and Hanjun Kim. DaCapo: Automatic Bootstrapping Management for Efficient Fully Homomorphic Encryption. 6

[13] Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine, and Madan Musuvathi. Eva: An encrypted vector arithmetic language and compiler for efficient homomorphic computation. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 546–561, 2020. 4.3

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, May 2019. URL `http://arxiv.org/abs/1810.04805`. arXiv:1810.04805 [cs]. 2.3, 5.2

[15] Ye Dong, Wen-jie Lu, Yancheng Zheng, Haoqi Wu, Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, and Wenguang Chen. PUMA: Secure Inference of LLaMA-7B in Five Minutes, September 2023. URL `http://arxiv.org/abs/2307.12533`. arXiv:2307.12533 [cs]. 3.1.1

[16] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009. 1

[17] Scott R Granter, Andrew H Beck, and David J Papke Jr. Alphago, deep learning, and the future of the human microscopist. *Archives of pathology & laboratory medicine*, 141(5): 619–621, 2017. 1

[18] Mirko Günther, Lars Schütze, Kilian Becher, Thorsten Strufe, and Jeronimo Castrillon. HElium: A Language and Compiler for Fully Homomorphic Encryption with Support for Proxy Re-Encryption, December 2023. URL `http://arxiv.org/abs/2312.14250`. arXiv:2312.14250 [cs]. 6

[19] Kyoohyung Han and Dohyeong Ki. Better bootstrapping for approximate homomorphic encryption. In *Cryptographers' Track at the RSA Conference*, pages 364–390. Springer, 2020. 4.1

[20] Kyoohyung Han, Seungwan Hong, Jung Hee Cheon, and Daejun Park. Logistic regression on homomorphic encrypted data at scale. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 9466–9471, 2019. 1.1, 3.1.2, 4.3

[21] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024. 4.2

[22] Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351, 2017. 1

[23] Yongnam Jung, Cheng Chen, Eunchae Jang, and S Shyam Sundar. Do we trust chatgpt as much as google search and wikipedia? In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, pages 1–9, 2024. 2.1

[24] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A low

latency framework for secure neural network inference. In *27th USENIX security symposium (USENIX security 18)*, pages 1651–1669, 2018. 3.1.1, 3.1.2

[25] Jongmin Kim, Gwangho Lee, Sangpyo Kim, Gina Sohn, Minsoo Rhu, John Kim, and Jung Ho Ahn. Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1237–1254. IEEE, 2022. 4.1

[26] Jongmin Kim, Gwangho Lee, Sangpyo Kim, Gina Sohn, Minsoo Rhu, John Kim, and Jung Ho Ahn. ARK: Fully Homomorphic Encryption Accelerator with Runtime Data Generation and Inter-Operation Key Reuse. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1237–1254, October 2022. doi: 10.1109/MICRO56248.2022.00086. URL https://ieeexplore.ieee.org/document/9923889/?arnumber=9923889. 6

[27] Aleksandar Krastev, Nikola Samardzic, Simon Langowski, Srinivas Devadas, and Daniel Sanchez. A Tensor Compiler with Automatic Data Packing for Simple and Efficient Fully Homomorphic Encryption. *Proc. ACM Program. Lang.*, 8(PLDI):152:126–152:150, June 2024. doi: 10.1145/3656382. URL https://dl.acm.org/doi/10.1145/3656382. 6

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://papers.nips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html. 1

[29] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. 1

[30] Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, and Jong-Seon No. Privacy-Preserving Machine Learning with Fully Homomorphic Encryption for Deep Neural Network, June 2021. URL http://arxiv.org/abs/2106.07229. arXiv:2106.07229 [cs]. 3.1.2, 5.1

[31] Yongwoo Lee, Seonyeong Heo, Seonyoung Cheon, Shinnung Jeong, Changsu Kim, Eunkyung Kim, Dongyoon Lee, and Hanjun Kim. HECATE: Performance-Aware Scale Optimization for Homomorphic Encryption Compiler. In *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 193–204, April 2022. doi: 10.1109/CGO53902.2022.9741265. URL https://ieeexplore.ieee.org/abstract/document/9741265. 4.3

[32] Dacheng Li, Rulin Shao, Hongyi Wang, Han Guo, Eric P. Xing, and Hao Zhang. MPCFormer: fast, performant and private Transformer inference with MPC, March 2023. URL http://arxiv.org/abs/2211.01452. arXiv:2211.01452 [cs]. 3.1.1

[33] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC conference on*

computer and communications security, pages 619–631, 2017. 3.1.1

[34] Shiwei Liu, Guanchen Tao, Yifei Zou, Derek Chow, Zichen Fan, Kauna Lei, Bangfei Pan, Dennis Sylvester, Gregory Kielian, and Mehdi Saligane. ConSmax: Hardware-Friendly Alternative Softmax with Learnable Parameters, February 2024. URL http://arxiv.org/abs/2402.10930. arXiv:2402.10930 [cs]. 4.1

[35] Payman Mohassel and Yupeng Zhang. SecureML: A System for Scalable Privacy-Preserving Machine Learning, 2017. URL https://eprint.iacr.org/2017/396. Publication info: Published elsewhere. Minor revision. IEEE Symposium on Security and Privacy 2017. 3.1.1

[36] Muhammad Haris Mughees and Ling Ren. Vectorized batch private information retrieval. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 437–452. IEEE, 2023. 2.4, 4.2

[37] Daryna Oliynyk, Rudolf Mayer, and Andreas Rauber. I Know What You Trained Last Summer: A Survey on Stealing Machine Learning Models and Defences. *ACM Computing Surveys*, 55(14s):1–41, December 2023. ISSN 0360-0300, 1557-7341. doi: 10.1145/3595292. URL https://dl.acm.org/doi/10.1145/3595292. 2.1

[38] Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. Bolt: Privacy-preserving, accurate and efficient inference for transformers. *Cryptology ePrint Archive*, 2023. 3.1.1, 3.1.2

[39] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 4.2, 5.4.2

[40] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. 2.3

[41] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023. 1

[42] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sanchez. CraterLake: a hardware accelerator for efficient unbounded computation on encrypted data. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 173–187, New York New York, June 2022. ACM. ISBN 978-1-4503-8610-4. doi: 10.1145/3470496.3527393. URL https://dl.acm.org/doi/10.1145/3470496.3527393. 4.1, 6

[43] SEAL. Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL, January 2023. Microsoft Research, Redmond, WA. 5.1

[44] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*,

2023. 4.2

[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html. (document), 2.2

[46] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *nature*, 575 (7782):350–354, 2019. 1

[47] Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*, 2023. 1

[48] Jiawen Zhang, Jian Liu, Xinpeng Yang, Yinghao Wang, Kejia Chen, Xiaoyang Hou, Kui Ren, and Xiaohu Yang. Secure Transformer Inference Made Non-interactive, 2024. URL https://eprint.iacr.org/2024/136. Publication info: Preprint. 1, 1.1, 3.1.2

[49] Itamar Zimerman, Moran Baruch, Nir Drucker, Gilad Ezov, Omri Soceanu, and Lior Wolf. Converting transformers to polynomial form for secure inference over homomorphic encryption. *arXiv preprint arXiv:2311.08610*, 2023. 1