CHET: Compiler and Runtime for Homomorphic Evaluation of Tensor Programs

Roshan Dathathri[*], Olli Saarikivi[†], Hao Chen[†], Kim Laine[†], Kristin Lauter[†], Saeed Maleki[†], Madanlal Musuvathi[†], and Todd Mytkowicz[†]

[*]Department of Computer Science, University of Texas at Austin, USA
roshan@cs.utexas.edu
[†]Microsoft Research, USA
{olsaarik,haoche,kilai,klauter,saemal,madanm,toddm}@microsoft.com

Abstract

Fully Homomorphic Encryption (FHE) provides the promise of performing arbitrary computations on encrypted data without requiring the decryption key. FHE can enable novel privacy-sensitive machine learning scenarios. However, programming FHE applications today is hard for non-FHE experts due to two challenges. First, achieving practical performance requires performing FHE-specific optimizations, including maximizing the vectorizing/batching capabilities of the underlying FHE scheme. Second, FHE schemes involve a careful choice of encryption parameters that tradeoff security for correctness, performance, and message bloat.

This paper proposes CHET, a compiler and runtime for homomorphically evaluating tensor programs. Given a neural network specified as a high-level tensor circuit, CHET optimizes and compiles this circuit to an interface called the Homomorphic Instruction Set Architecture (HISA), which can then be targetted to different encryption libraries. CHET automatically chooses the encryption parameters and layouts that maximize performance for a given security and precision requirements. As a result, CHET generated code is faster than expert-optimized implementations.

1 Introduction

Privacy-preserving machine learning extends the scope of neural networks to fields such as healthcare, banking, and finance. As an example, a hospital might desire to use the massive compute capabilities of a public cloud to analyze its patient data to improve its healthcare, but is unable to do so today due to privacy concerns.

Fully Homomorphic Encryption (FHE) provides the promise of performing neural network inference and training tasks on encrypted data without requiring the decryption key. FHE provides a simple trust model: the data owner neither needs to trust the cloud software provider nor the vendor of the hardware doing the computation. In contrast, other cryptographic solutions like Secure Multi-Party Computation (MPC) [13, 14] require non-collusion assumptions and typically have larger communication costs. Non-cryptographic technologies such as *secure enclaves* [11] (such as Intel SGX [8]) requires one to trust the hardware vendor, which could be a non-starter for many privacy-sensitive applications.

FHE is usually considered impractical due to its performance overhead. However, recent advances in FHE schemes and implementations have dramatically improved the performance to make many applications practical. Morover, modern FHE schemes [5, 9, 7] organically support integer and fixed-precision arithmetic dramatically reducing the size of the circuits required when compared to schemes that only provide Boolean operations. Nevertheless, achieving acceptable performance remains a huge challenge.

Another concern is that building efficient FHE applications is manual and error-prone. Specifically, FHE schemes require setting of encryption parameters that involve complex trade offs among correctness, performance, security, and message bloat. Thus, many applications of FHE today require an active involvement of an FHE expert.

In this paper, we address these challenges with CHET, a compiler and runtime for fully-homomorphic evaluation of tensor programs. Given a neural network specified as a high-level tensor program, CHET compiles it to an intermediate representation called as Homomorphic Instruction Set Architecture (HISA). The HISA representation can then be targetted to an appropriate FHE library, such as HEAAN [7], that provides the desired encryption schemes implementing HISA. We assume that the activations functions are replaced with polynomial approximations [10, 6].

CHET guarantees sound semantics by selecting the right encryption parameters that maximize performance while guaranteeing security and application-intended precision of the computed results. In addition, CHET explores a large space of optimization choices, such as the multiple ways to batch data into ciphertexts as well as multiple ways to implement computational kernels for each layout.

We evaluate CHET with a set of real-world CNN models and show how compiler optimizations can significantly improve inference latencies. We further demonstrate how the optimized kernel implementations in CHET allow neural networks of large depths to be homomorphically evaluated. Specifically, we show that different neural networks require subtly different layout and optimization parameters to maximize performance, suggesting that a compiler is better suited to make these choices than a human. In particular, for a neural network used by an industry partner for medical imaging, the inference using the straightforward implementation on HEAAN took more than 18 hours. A hand-tuned implementation by an FHE expert reduced this to 40 minutes, while CHET generated an implementation that only takes 5 minutes.

2 Software Stack for Homomorphic Evaluation of Tensor Programs

This section presents an overview of an end-to-end software stack for evaluating *tensor programs* on homomorphically encrypted data. The target architecture for this stack is a Fully Homomorphic Encryption (FHE) scheme and our tensor compiler, CHET, interacts with this scheme using the Homomorphic Instruction Set Architecture (HISA). At the top of stack rests the user-provided tensor program or circuit. CHET consists of a compiler and runtime to bridge the gap in-between them.

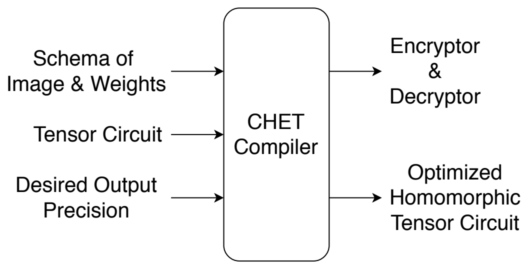

Figure 1: Overview of the CHET system at compile-time.

Figure 1 shows the overview of the CHET compiler. In addition to the tensor circuit, CHET requires the schema of its input and weights. The schema specifies the dimensions of the tensors as well as the floating-point precision required of the values in those tensors. CHET also requires the desired floating-point precision of the output of the circuit. Using these constraints, CHET generates an equivalent, optimized homomorphic tensor circuit as well as an encryptor and decryptor. These executables encode the choices made by the compiler to make the homomorphic computation efficient.

To evaluate the tensor circuit on an image, the client first generates a private key and encrypts the image using the encryptor (which can also generate private keys) provided by the compiler, as shown in Figure 2. The encrypted image is then sent to the server along with unencrypted weights and public keys required for evaluating homomorphic operations (i.e., multiplication and rotation). The server executes the optimized homomorphic tensor circuit generated by the CHET compiler. The homomorphic tensor operations in the circuit are executed using the CHET runtime, which uses an underlying FHE scheme to execute homomorphic computations on encrypted data. The circuit produces an encrypted prediction, which it then sends to the client. The client decrypts the encrypted

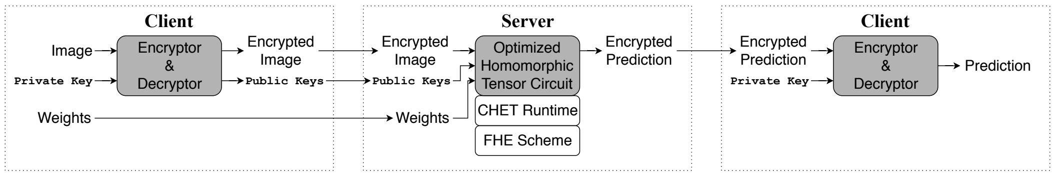

Figure 2: Overview of the CHET system at runtime.

prediction with its private keys using the compiler generated decryptor. In this way, the client runs tensor programs like neural networks on the server without the server being privy to the data, the output (prediction), or any intermediate state.

The FHE scheme exposes several policies or heuristics such as the encryption parameters. Similarly, the CHET runtime also exposes several policies that could be specific to or independent of the FHE scheme. This is analogous to Intel MKL libraries having different implementations of the same operation, where the most performant implementation depends on the size of the input or the target architecture. In the case of homomorphic computation, these policies not only affect performance but can also affect security and accuracy. A key design principle of CHET is the separation of concern between the policies of choosing the secure, accurate, and most efficient homomorphic operation from the mechanisms of executing those policies. Some of these policies are independent of the input or weights schema, so they are entrusted to the CHET runtime. On the other hand, most policies may require either the input or weights schema or global analysis of the program. In such cases, the CHET compiler chooses the appropriate policy and the CHET runtime implements the mechanisms for that policy.

3 Evaluation

Our evaluation targets a set of neural network architectures for image classification tasks.

LeNet-5-like is a series of networks for the MNIST [2] dataset. We use three versions with different number of neurons: LeNet-5-small, LeNet-5-medium, and LeNet-5-large. The largest one matches the one used in the TensorFlow's tutorials [4] (21385674 floating-point operations). These networks have two convolutional layers, each followed by ReLU activation and max pooling, and two fully connected layers with a ReLU in between.

SqueezeNet-CIFAR is a network for the CIFAR-10 dataset [1] that follows the SqueezeNet [12] architecture. This version has 4 Fire-modules [3] for a total of 10 convolutional layers (37759754 floating-point operations).

Industrial is a pretrained HE-compatible network from an industry partner for a privacy-sensitive image classification task. We are unable to reveal the details of the network other than the fact that it has 5 convolutional layers and 2 fully connected layers.

All networks other than Industrial use ReLUs and max-pooling, which are not compatible with homomorphic evaluation. We modified the activation functions from ReLU to a second-degree polynomial [10, 6]. The key difference with prior work is that our activation functions are $f(x) = ax^2 + bx$ with learnable parameters a and b. During the training phase, the DNN adjusts these parameters automatically to appropriately approximate the ReLU function. To avoid exploding the gradients during training (which usually happens during the initial parts of training), we initialized a to zero and clipped the gradients when large. We also replaced max-pooling with average-pooling. To the best of our knowledge, SqueezeNet-CIFAR is the deepest neural network that has been homomorphically evaluated.

All experiments were run on a dual socket Intel Xeon E5-2667v3@3.2GHz with 224 GB of memory. Hyperthreading was off for a total of 16 hardware threads. All runtimes are reported as averages over 20 different images. We present the average latency of image inference with a batch size of 1.

Model	CHET	Hand-written
LeNet-5-small	8	14
LeNet-5-medium	51	140
LeNet-5-large	265	-
Industrial	312	2413
SqueezeNet-CIFAR10	1342	-

Table 1: Average latency (in seconds) of CHET and hand-written versions.

Comparison with hand-written: Table 1 compares hand-written implementations and CHET with all optimizations. CHET clearly outperforms hand-written implementations. The hand-written implementations lack some of the optimizations in the CHET compiler and runtime. Moreover, it is difficult to scale these hand-written implementations to large networks like LeNet-5-large and SqueezeNet-CIFAR10, so we do not have hand-written implementations for these to compare against.

Parameter Selection: In Table 2, the last columns show precision (the number of decimal digits) required for the image or ciphertext (P_c) and the weights or the plaintext (P_p), that are provided by the user. The precision provided is used by the CHET compiler to select the encryption parameters N and Q. The values of these parameters grow with the depth of the circuit, as shown in the figure. With these parameters, CHET generated homomorphic tensor circuits achieve the same accuracy as the unencrypted circuits while providing strong security guaranty for the input, intermediate states, and the output. In addition, the difference between the output values of these circuits is within the desired precision of the output.

Model	$\log(N)$	$\log(Q)$	$\log(P_c)$	$\log(P_p)$
LeNet-5-small	14	240	30	16
LeNet-5-medium	14	240	30	16
LeNet-5-large	15	400	40	20
Industrial	16	705	35	25
SqueezeNet-CIFAR10	16	940	30	20

Table 2: Encryption parameters (N and Q) selected by CHET and the user-provided precisions (P_c and P_p) for each model.

Data Layout Selection: We evaluate four different data tiling layouts choices: (i) HW: each ciphertext has all height and width elements of a single channel, (ii) CHW: each ciphertext has multiple channels (all height and width elements of each), (iii) HW-conv and CHW-rest: same has CHW, but move to HW before each convolution and back to CHW after each convolution, and (iv) CHW-fc and HW-before: same as HW, but switch to CHW during the first fully connected layer and CHW thereafter. Table 3 presents the average latency of each network for each layout. We can see that for each network, a different layout provides the lowest latency. It is very difficult for the user to determine which is the best data layout and more importantly, it is difficult to implement each network manually using a different data layout. This highlights how the compiler should search the space of possible layouts and kernel implementations for each program separately, while entrusting the runtime to implement it efficiently. In this case, the compiler chooses the best performing data layout for each network based on the cost model of HEAAN.

Model	HW	CHW	HW-conv CHW-rest	CHW-fc HW-before
LeNet-5-small	8	12	8	**8**
LeNet-5-medium	82	91	52	**51**
LeNet-5-large	325	423	270	**265**
Industrial	330	**312**	379	381
SqueezeNet-CIFAR	**1342**	1620	1550	1342

Table 3: Average latency (in seconds) with different layouts.

Rotation Keys Selection: We evaluate the efficacy of our rotation keys optimization on the three largest networks. Figure 4 presents the average latencies with the optimization on or off. The optimization provides significantly improved performance for all networks and should be always used. Having this optimization implemented as an automatic compiler pass removes the burden of adding proper rotation keys in each program separately.

Model	Unoptimized	Optimized
LeNet-5-small	14	8
LeNet-5-medium	73	51
LeNet-5-large	426	265
Industrial	645	312
SqueezeNet-CIFAR	2648	1342

Table 4: Average latency (in seconds) with and without rotation keys optimization.

4 Conclusion

Good abstractions separate concerns to either side of that abstraction. For example, `x86` lets hardware manufacturers innovate independently of the software developers and compiler writers that target that abstraction. This paper introduces such an abstraction for FHE applications that cleanly abstracts and exposes features of FHE implementations. We demonstrate a compiler and runtime that targets FHE tensor programs, evaluate that compiler and runtime on real-world CNN models, and demonstrate our compiler is able to significantly optimize the performance of FHE tensor programs. Because of these optimizations, this paper demonstrates the deepest FHE based CNNs to date.

References

[1] The CIFAR-10 dataset. `https://www.cs.toronto.edu/~kriz/cifar.html`.

[2] The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`.

[3] Squeezenet for CIFAR-10. `https://github.com/kaizouman/tensorsandbox/tree/master/cifar10/models/squeeze`.

[4] TensorFlow's LeNet-5-like convolutional MNIST model example. `https://github.com/tensorflow/models/blob/v1.9.0/tutorials/image/mnist/convolutional.py`.

[5] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):13, 2014.

[6] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. Privacy-preserving classification on deep neural network. *IACR Cryptology ePrint Archive*, 2017:35, 2017.

[7] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, LNCS 10624, pages 409–437. Springer, 2017. `https://doi.org/10.1007/978-3-319-70694-8_15`.

[8] Intel Corp. Intel software guard extensions (Intel SGX), ref. 332680-002. `https://software.intel.com/sites/default/files/332680-002.pdf`, jun 2015.

[9] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[10] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 201–210, 2016.

[11] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. Using innovative instructions to create trustworthy software solutions. In *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, pages 11:1–11:1, New York, NY, USA, 2013. ACM.

[12] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016. `https://arxiv.org/abs/1602.07360`.

[13] P. Mohassel and Y. Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38, May 2017.

[14] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: Scalable provably-secure deep learning. *CoRR*, abs/1705.08963, 2017.