Delphi: A Cryptographic Inference Service for Neural Networks

Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan
Wenting Zheng, Raluca Ada Popa
UC Berkeley

Abstract

Many companies provide neural network prediction services to users for a wide range of applications. However, current prediction systems compromise one party's privacy: either the user has to send sensitive inputs to the service provider for classification, or the service provider must store its proprietary neural networks on the user's device. The former harms the personal privacy of the user, while the latter reveals the service provider's proprietary model.

We design, implement, and evaluate Delphi, a secure prediction system that allows two parties to execute neural network inference without revealing either party's data. Delphi approaches the problem by simultaneously co-designing cryptography and machine learning. We first design a hybrid cryptographic protocol that improves upon the communication and computation costs over prior work. Second, we develop a planner that automatically generates neural network architecture configurations that navigate the performance-accuracy trade-offs of our hybrid protocol. Together, these techniques allow us to achieve a 22× improvement in online prediction latency compared to the state-of-the-art prior work.

ACM Reference Format:
Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan and Wenting Zheng, Raluca Ada Popa. 2020. Delphi: A Cryptographic Inference Service for Neural Networks. In *2020 Workshop on Privacy-Preserving Machine Learning in Practice (PPMLP'20), November 9, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3411501.3419418

1 Introduction

Recent advances in machine learning have driven increasing deployment of neural network inference in popular applications like voice assistants [3] and image classification [19]. However, the use of inference in many such applications raises privacy concerns. For example, home monitoring systems (HMS) such as Kuna [17] and Wyze [27] use proprietary neural networks to classify objects in video streams of users' homes such as cars parked near the user's house, or faces of visitors to the house. These models are core to these companies' business and are expensive to train.

To make use of these models, either the user has to upload their streams to the servers of the HMS (which then evaluate the model over the stream), or the HMS has to store its model on the user's monitoring device (which then performs the classification). Both of these approaches are unsatisfactory: the first requires users to upload video streams containing sensitive information about their daily activities to another party, while the second requires the HMS to store its model on every device, thus allowing users and competitors to steal the proprietary model.

This work is licensed under a Creative Commons Attribution International 4.0 License.

PPMLP'20, November 9, 2020, Virtual Event, USA
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8088-1/20/11.
https://doi.org/10.1145/3411501.3419418

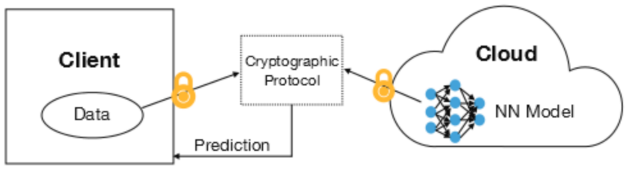

Figure 1: Cryptographic neural network inference. The lock indicates data provided in encrypted form.

To alleviate these privacy concerns, a number of recent works have proposed protocols for *cryptographic prediction* over (convolutional) neural networks [12, 20, 18, 16] by utilizing specialized secure multi-party computation (MPC) [28, 13]. At a high level, these protocols proceed by encrypting the user's input and the service provider's neural network, and then tailor techniques for computing over encrypted data (like homomorphic encryption or secret sharing) to run inference over the user's input. At the end of the protocol execution, the intended party(-ies) learn the inference result; neither party learns anything else about the other's input. Fig. 1 illustrates this protocol flow.

Unfortunately, these cryptographic prediction protocols are still unsuitable for deployment in real world applications as they require the use of heavy cryptographic tools during the online execution. These tools are computationally intensive and often require a large amount of communication between the user and the service provider. Furthermore, this cost grows with the complexity of the model, making these protocols unsuitable for use with state-of-the-art neural network architectures used in practice today. For example, using a state-of-the-art protocol like Gazelle [16] to perform inference for state-of-the-art deep neural networks like ResNet-32 [15] requires ~ 82 seconds and results in over 560 MB communication.

Our contribution. In this paper, we present Delphi, a cryptographic prediction system for realistic neural network architectures. Delphi achieves its performance via a careful co-design of cryptography and machine learning. Delphi contributes a novel hybrid cryptographic prediction protocol, as well as a planner that can adjust the machine learning algorithm to take advantage of the performance-accuracy trade-offs of our protocol. Our techniques enable us to perform cryptographic prediction on more realistic network architectures than those considered in prior work. For example, using Delphi for cryptographic prediction on ResNet-32 requires just 3.8 seconds and 60 MB communication in the online phase, improving upon Gazelle by 22× and 9× respectively.

1.1 Techniques

We now describe at a high level the techniques underlying Delphi's excellent performance.

Performance goals. Modern convolutional neural networks consist of a number of layers, each of which contains one sub-layer for *linear* operations, and one sub-layer for *non-linear* operations. Common linear operations include convolutions, matrix multiplication, and average pooling. Non-linear operations include activation functions such as the popular ReLU (Rectified Linear Unit) function.

Achieving cryptographic prediction for realistic neural networks thus entails (a) constructing efficient subprotocols for evaluating linear and non-linear layers, and (b) linking the results of these subprotocols with each other.

Prior work. Almost all prior protocols for cryptographic prediction utilize heavyweight cryptographic tools to implement these subprotocols, which results in computation and communication costs that are much higher than the equivalent plaintext costs. Even worse, many protocols utilize these tools during the latency-sensitive *online phase* of the protocol, i.e., when the user acquires their input and wishes to obtain a classification for it. (This is opposed to the less latency-sensitive *preprocessing phase* that occurs before the user's input becomes available).

For example, the online phase of the state-of-the-art Gazelle protocol uses heavy cryptography like linearly homomorphic encryption and garbled circuits. This results in heavy preprocessing *and* online costs: for the popular network architecture ResNet-32 trained over CIFAR-100, Gazelle requires $\sim$ 158 seconds and 8 GB of communication during the preprocessing phase, and $\sim$ 50 seconds and 5 GB of communication during the preprocessing phase, and $\sim$ 82 seconds and 600 MB of communication during the online phase.

Delphi's protocol. To achieve good performance on realistic neural networks, Delphi builds upon techniques from Gazelle to develop new protocols for evaluating linear and non-linear layers that minimize the use of heavy cryptographic tools, and thus minimizes communication and computation costs in the preprocessing and online phases. We begin with a short overview of Gazelle's protocol as it is the basis for Delphi's protocols.

Starting point: Gazelle. Gazelle [16] is a state-of-the-art cryptographic prediction system for convolutional neural networks. Gazelle computes linear layers using an optimized *linearly-homomorphic encryption* (LHE) scheme [6, 21, 22, 8] that enables one to perform linear operations directly on ciphertexts. To compute non-linear layers, Gazelle uses *garbled circuits* [28] to compute the bitwise operations required by ReLU. Finally, because each layer in a neural network consists of alternating linear and non-linear layers, Gazelle also describes how to efficiently switch back-and-forth between the two aforementioned primitives via a technique based on additive secret sharing.

As noted above, Gazelle's use of heavy cryptography in the online phase leads to efficiency and communication overheads. To reduce these overheads, we proceed as follows.

Reducing the cost of linear operations. To reduce the online cost of computing the linear operations, we adapt Gazelle to move the heavy cryptographic operations over LHE ciphertexts to the preprocessing phase. Our key insight is that the service provider's input $\mathbf{M}$ to the linear layer (i.e. the model weights for that layer) is known before user's input is available, and so we can use LHE to create secret shares of $\mathbf{M}$ during preprocessing. Later, when the user's input becomes available in the online phase, all linear operations can be performed directly over secret-shared data without invoking heavy cryptographic tools like LHE, and without requiring interactions to perform matrix-vector multiplications.

The benefits of this technique are two-fold. First, the online phase only requires transmitting secret shares instead of ciphertexts, which immediately results in an $8\times$ reduction in online communication for linear layers. Second, since the online phase only performs computations over elements of prime fields, and since our system uses concretely small 32-bit primes for this purpose, our system can take advantage of state-of-the-art CPU and GPU libraries for computing linear layers; see the full version for details.

Reducing the cost of non-linear operations. While the above technique already significantly reduces computation time and communication cost, the primary bottleneck for both remains the cost of evaluating garbled circuits for the ReLU activation function. To minimize this cost, we use an alternate approach [12, 18, 20, 5] that is better suited to our setting of computing over finite field elements: computing polynomials. In more detail, Delphi replaces ReLU activations with polynomial (specifically, quadratic) approximations. These can be computed securely and efficiently via standard protocols [4].

Because these protocols only require communicating a small constant number of field elements per multiplication, using quadratic approximations significantly reduces the communication overhead per activation, without introducing additional rounds of communication. Similarly, since the underlying multiplication protocol only requires a few cheap finite field operations, the computation cost is also reduced by several orders of magnitude. Concretely, the online communication and computation costs of securely computing quadratic approximations are $192\times$ and $10000\times$ smaller (respectively) than the corresponding costs for garbled circuits.

However, this performance improvement comes at the cost of accuracy and trainability of the underlying neural network. Prior work has already established that quadratic approximations provide good accuracy in some settings [20, 18, 11, 5]. At the same time, both prior work [20] and our own experiments indicate that in many settings simply replacing ReLU activations with quadratic approximations results in severely degraded accuracy, and can increase training time by orders of magnitude (if training converges at all). To overcome this, we develop a *hybrid cryptographic protocol* that uses ReLUs *and* quadratic approximations to achieve good accuracy *and* good efficiency.

Planning an efficient usage of the hybrid cryptographic protocol. It turns out that it is not straightforward to determine *which* ReLU activations should be replaced with quadratic approximations. Simply replacing arbitrary ReLU activations with quadratic approximations can degrade the accuracy of the resulting network, and can even cause the network to fail to train.

So, to find an appropriate placement or *network configuration*, we design a planner that automatically discovers which ReLUs

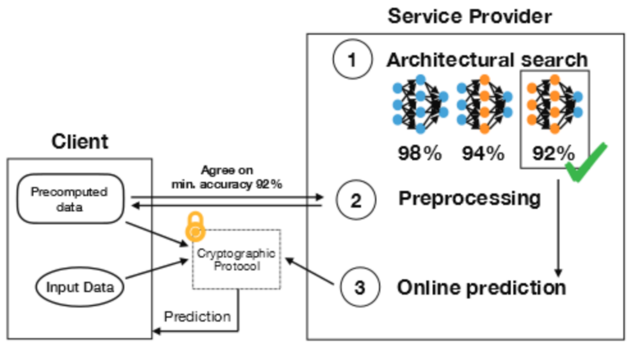

Figure 2: Delphi's architecture. Orange layers represent quadratic approximations while blue ones represent ReLUs.

to replace with quadratic approximations so as to maximize the number of approximations used while still ensuring that accuracy remains above a specified threshold.

The insight behind our planner is to adapt techniques for *neural architecture search* (NAS) and *hyperparameter optimization* (see [7, 25] for in-depth surveys of these areas) to our setting. Namely, we adapt these techniques to discover which layers to approximate within a given neural network architecture, and to optimize the hyperparameters for the discovered network.

The overall system. Delphi combines the above insights into a cohesive system that service providers can use to automatically generate cryptographic prediction protocols meeting performance and accuracy criteria specified by the provider. In more detail, the service provider invokes Delphi's planner with acceptable accuracy and performance thresholds. The planner outputs an optimized architecture that meets this goal, which Delphi then uses to instantiate a concrete cryptographic prediction protocol that utilizes our cryptographic techniques from above.

This co-design of cryptography and machine learning enables Delphi to efficiently provide cryptographic prediction for networks deeper than any considered in prior work. For example, using Delphi to provide inference for the popular ResNet-32 architecture requires only 60 MB communication and 3.8 seconds.

2 System overview

2.1 System setup

There are two parties in the system setup: the client and the service provider (or server). In the plaintext version of our system, the service provider provides prediction as a service using its internal models via an API. The client uses this API to run prediction on its own data by transferring its data to the service provider. The service provider runs prediction using the appropriate neural network, then sends the prediction result back to the client. In Delphi, the two parties execute a secure prediction together by providing their own inputs. The service provider's input is the neural network, while the client's input is its private input used for prediction.

2.2 Threat model

Delphi's threat model is similar to that of prior secure prediction works such as Gazelle [16] and MiniONN [18]. More specifically, Delphi is designed for the two-party semi-honest setting, where only one of the parties is corrupted by an adversary. Furthermore, this adversary never deviates from the protocol, but it will try to learn information about the other parties' private inputs from the messages it receives.

2.3 Privacy goals

Delphi's goal is to enable the client to learn only two pieces of information: the architecture of the neural network, and the result of the inference; all other information about the client's private inputs and the parameters of the server's neural network model should be hidden. Concretely, we aim to achieve a strong simulation-based definition of security.

Like all prior work, Delphi does not hide information about the architecture of the network, such as the dimensions and type of each layer in the network. For prior work, this is usually not an issue because the architecture is independent of the training data. However, because Delphi's planner uses training data to optimally place quadratic approximations, revealing the network architecture reveals some information about the data. Concretely, in optimizing an ℓ-layer network, the planner makes ℓ binary choices, thus reveals at most ℓ bits of information about the training data. Because ℓ is concretely small for actual networks (for example, $\ell = 32$ for ResNet-32), this leakage is negligible. This leakage can be further mitigated by using differentially private training algorithms [23, 1]

Delphi, like most prior systems for cryptographic prediction, does not hide information that is revealed by the result of the prediction. In our opinion, protecting against attacks that exploit this leakage is a complementary problem to that solved by Delphi. Indeed, such attacks have been successfully carried out even against systems that "perfectly" hide the model parameters by requiring the client to upload its input to the server [10, 2, 9, 26, 24]. Furthermore, popular mitigations for these attacks, such as differential privacy, can be combined with Delphi's protocol. We discuss these attacks and possible mitigations in more detail in the full version.

2.4 System architecture and workflow

Delphi's architecture consists of two components: a hybrid cryptographic protocol for evaluating neural networks, and a neural network configuration planner that optimizes a given neural network for use with our protocol. Below we provide an overview of these components, and then demonstrate how one would use these in practice by describing an end-to-end workflow for cryptographic prediction in home monitoring systems (HMS).

Hybrid cryptographic protocol. Delphi's protocol for cryptographic prediction consists of two phases: an offline preprocessing phase, and an online inference phase. The offline preprocessing phase is independent of the client's input (which regularly changes), but assumes that the server's model is static; if this model changes, then both parties would have to re-run the preprocessing phase. After preprocesing, during the online inference phase, the client provides its input to our specialized secure two-party computation protocol, and eventually learns the inference result. We note that our protocol provides two different methods of evaluating non-linear layers: the first offers better accuracy at the cost of worse offline and online efficiency, while the other degrades accuracy, but offers much improved offline and online efficiency.

Planner. To help service providers navigate the trade off between performance and accuracy offered by these two complementary methods to evaluate non-linear layers, Delphi adopts a principled approach by designing a *planner* that generates neural networks that mix these two methods to maximize efficiency while still achieving the accuracy desired by the service provider. Our planner applies neural architecture search (NAS) to the cryptographic setting in a novel way in order to automatically discover the right architectures.

Example 2.1 (HMS workflow). As explained in Section 1, a home monitoring system (HMS) enables users to surveil activity inside and outside their houses. Recent HMSes [17, 27] use neural networks to decide whether a given activity is malicious or not. If it is, they alert the user. In this setting privacy is important for both the user and the HMS provider, which makes Delphi an ideal fit. To use Delphi to provide strong privacy, the HMS provider proceeds as follows.

The HMS provider first invokes Delphi's planner to optimize its baseline all-ReLU neural network model. Then, during the HMS device's idle periods, the device and the HMS server run the preprocessing phase for this model. If the device detects suspicious activity locally, it can run the online inference phase to obtain a classification. On the basis of this result, it can decide whether to alert the user or not.

Remark 2.2 (applications suitable for use with Delphi). Example 2.1 indicates that Delphi is best suited for applications where there is ample computational power available for preprocessing, and where inference is latency-sensitive, but is not performed frequently enough to deplete the reserve of preprocessed material. Other examples of such applications include image classification in systems like Google Lens [14].

References

[1] M. Abadi, A. Chu, I. J. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. "Deep Learning with Differential Privacy". In: CCS '16.

[2] G. Ateniese, L. V. Mancini, A. Spognardi, A. Villani, D. Vitali, and G. Felici. "Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers". In: *IJSN* (2015).

[3] B. Barrett. "The year Alexa grew up". https://www.wired.com/story/amazon-alexa-2018-machine-learning/.

[4] D. Beaver. "Precomputing Oblivious Transfer". In: CRYPTO '95.

[5] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei. "Faster CryptoNets: Leveraging Sparsity for Real-World Encrypted Inference". ArXiV, cs.CR 1811.09953.

[6] T. Elgamal. "A public key cryptosystem and a signature scheme based on discrete logarithms". In: *IEEE Trans. on Inf. Theory* (1985).

[7] T. Elsken, J. H. Metzen, and F. Hutter. "Neural Architecture Search: A Survey". In: *JMLR* (2019).

[8] J. Fan and F. Vercauteren. "Somewhat Practical Fully Homomorphic Encryption". ePrint Report 2012/144.

[9] M. Fredrikson, S. Jha, and T. Ristenpart. "Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures". In: CCS '15.

[10] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart. "Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing". In: USENIX Security '14.

[11] Z. Ghodsi, T. Gu, and S. Garg. "SafetyNets: Verifiable Execution of Deep Neural Networks on an Untrusted Cloud". In: NIPS '17.

[12] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing. "CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy". In: ICML '16.

[13] O. Goldreich, S. Micali, and A. Wigderson. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority". In: STOC '87.

[14] Google. "Google Lens". https://lens.google.com/.

[15] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: CVPR '16.

[16] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. "GAZELLE: A Low Latency Framework for Secure Neural Network Inference". In: USENIX '18.

[17] Kuna. "Kuna AI". https://getkuna.com/blogs/news/2017-05-24-introducing-kuna-ai.

[18] J. Liu, M. Juuti, Y. Lu, and N. Asokan. "Oblivious Neural Network Predictions via MiniONN Transformations". In: CCS '17.

[19] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi. "A survey of deep neural network architectures and their applications". In: *Neurocomputing* (2017).

[20] P. Mohassel and Y. Zhang. "SecureML: A System for Scalable Privacy-Preserving Machine Learning". In: IEEE S&P '17.

[21] P. Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: EUROCRYPT '99.

[22] O. Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: *JACM* (2009).

[23] R. Shokri and V. Shmatikov. "Privacy-Preserving Deep Learning". In: CCS '15.

[24] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. "Stealing Machine Learning Models via Prediction APIs". In: USENIX Security '16.

[25] M. Wistuba, A. Rawat, and T. Pedapati. "A Survey on Neural Architecture Search". ArXiV, cs.LG 1905.01392.

[26] X. Wu, M. Fredrikson, S. Jha, and J. F. Naughton. "A Methodology for Formalizing Model-Inversion Attacks". In: CSF '16.

[27] "Wyze: Contact and Motion Sensors for Your Home". https://www.wyze.com/.

[28] A. C. Yao. "How to Generate and Exchange Secrets (Extended Abstract)". In: FOCS '86.