Secure Neural Network Inference for Edge Intelligence: Implications of Bandwidth and Energy Constraints[1]

Jorit Prins, Zoltán Ádám Mann

University of Amsterdam,
Amsterdam, The Netherlands

Abstract. Recently, there has been growing interest in Machine Learning as a Service (MLaaS). In MLaaS, an operator provides pre-trained neural networks, with which inference on clients' inputs can be performed. MLaaS is attractive in providing edge intelligence in an Internet of Things (IoT) setup for multiple reasons, for example because it relieves clients with limited capacity from the computationally heavy training process. However, MLaaS may lead to privacy threats for both the client and the provider. In particular, the input of the client may be sensitive information that the provider is not allowed to learn. The provider, on the other hand, may not want to reveal the parameters of the neural network to the client, because these parameters are the provider's intellectual property. Besides, the output of the neural network might also reveal sensitive properties about the input. Lastly, traditional security solutions might fail in an IoT setup. In recent years, several cryptographic protocols have been devised for secure neural network inference (SNNI). Secure neural network inference entails the problem of computing the output of a neural network on the client's input without revealing the input to the provider, nor the parameters of the neural network to the client. So far, SNNI approaches were optimized for efficiency and accuracy, mainly in cloud settings. The goal of this chapter is to investigate the applicability of SNNI approaches in an edge computing setup. In particular, with power-constrained edge and IoT devices in mind, we investigate power consumption and energy consumption characteristics of SNNI approaches. Taking into account the typical bandwidth of access networks relevant to edge and IoT, we also investigate the effect of bandwidth limitations on the duration and energy consumption of the SNNI process. Our results indicate that the power consumption of SNNI depends significantly on both the used

[1] This paper was published in: S. Pal, C. Savaglio, R. Minerva, F. C. Delicato (editors): *IoT Edge Intelligence*, Springer, pp. 265-288, 2024

SNNI protocol and the available bandwidth.

Keywords: Privacy-preserving machine learning, Neural network, Edge intelligence, Energy consumption, Power consumption, Network bandwidth, Edge computing, Internet of Things

1 Introduction

With the rise in the availability of large amounts of data and computing resources, machine learning (ML) has gained huge importance. ML techniques like Neural Networks (NN) are promising ways to analyze data, to make decisions, and to predict future developments. ML has a wide variety of applications domains such as water quality evaluation [ZWY+2022], mental health prediction [CT2022], fake news detection [CA2023] and more [LYZ+2022] [GDL+2016].The use of ML typically consists of two phases: training and inference. In the training phase, a NN is trained by feeding an extensive dataset to find the best parameter values for the NN. In the inference phase, the NN is applied to new inputs. The training phase is often a tedious and time-consuming process. Because not everyone has the time, resources, data, and know-how to train a NN, Machine Learning as a Service (MLaaS) became popular [HLL+2021]. In MLaaS, a company or other party offers a pre-trained NN to its clients. This way, clients can benefit from inference with the NN, without needing to worry about the training phase. This is an incredibly useful application in the Internet of Things (IoT) [TKC+2020], for example patient monitoring by medical sensors [SRR+2021], or using IoT sensors to analyze real-time performance in automotive manufacturing [SAF+2018].

A typical MLaaS situation (Fig. 1) consists of two parties: the client holding an input x and a service provider holding a pre-trained neural network realizing a function f. This research focuses on the inference part: the client wants to know the output of the NN, available on a server owned by the service provider, applied to input x held by the client. This could be easily achieved: the client sends x to the server, the server calculates $f(x)$ and sends back the result to the client.

However, such a naïve implementation of MLaaS would lead to significant threats to security and privacy [TM2021]. The client's input may be confidential and the client may therefore be reluctant to send x to the server. Furthermore, the output $f(x)$ of the NN on the given input could also be confidential, resulting in the need to retain this information from unauthorized parties. Besides, traditional security and privacy approaches may fail on IoT devices [SKA+2023], for example because these devices heavily rely on other nodes in the network [LXZ2015].

An alternative implementation could consist in downloading the NN to the client and performing the inference there. However, this would also be problematic. For the service provider to train the model as accurately as possible, access to a large amount of precise data is needed, which may consist of sensitive information. This data, or properties of this data, could be stolen in the inference phase by the client. In addition, the service provider could be worried that the client or another adversary could steal the parameters of the NN, thus interfering with the business model of the service provider [QIU+2020]. Lastly, this may not work for IoT devices, since they often are power and resource constrained [LXZ2015].

Therefore, the aim is that the client receives $f(x)$, without learning anything about

the parameters of the NN (beyond what $f(x)$ might reveal about them), and the server does not learn anything about x or $f(x)$. This is the Secure Neural Network Inference (SNNI) problem. In recent years, many different approaches have been proposed to solve the SNNI problem [MWC+2023].

The SNNI approaches proposed so far were mainly optimized for and evaluated in a setup where both the client and server computers are powerful machines, often in the cloud, connected by a network with high bandwidth. In addition, evaluation mostly focused on latency (i.e., the duration of the SNNI process).

However, there is an increasing tendency to employ ML inference in an edge computing setup and/or in connection with the Internet of Things (IoT), often involving sensitive data, like medical data [LMD2021, SRR+2021]. In such a setup, security and privacy requirements have to be considered in conjunction with requirements stemming from the constrained resources of the involved devices and of the network [Man2022].

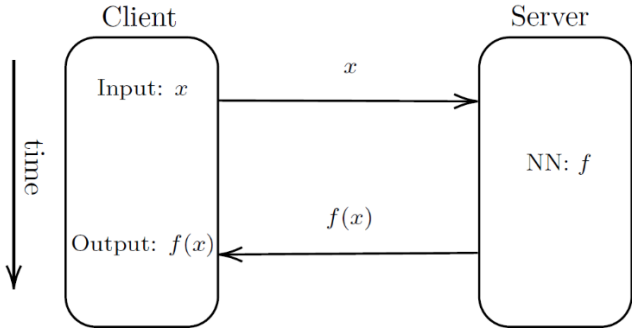


Fig. 1. Secure Neural Network Inference (SNNI) process in a Machine Learning as a Service (MLaaS) context. The neural network (NN) is already trained to realize a function f. The client provides an input x to the NN and wants to know the output $f(x)$ of the NN for this input.

In edge computing and IoT settings, energy consumption also plays a crucially important role [AAM+2021]. Client devices are often battery-powered, where the usefulness is heavily influenced by the battery life. Besides, client devices are often very constrained in terms of computing power. Companies, on the other hand, also want to keep energy consumption as low as possible because of energy costs and budget limitations, and should therefore try to avoid big energy overhead. Moreover, a server is often connected with many clients in a deeply connected IoT network. Metrics like energy and power consumption are therefore also of importance, and should be investigated. Another reason to limit energy consumption is the desire to reduce carbon emission in the fight against climate change. For example, an estimation made in the SMARTer 2030 report is that all ICT systems worldwide will make up for 2% of the global carbon emission in 2030 [Glo2015]. The growing number of IoT devices also contribute to the increase in global power consumption. Researchers also state that ICT programmers will have the potential to avoid 20% of the global greenhouse gas emissions with smart programming. However, the energy consumption of SNNI approaches has not been investigated yet.

The aim of this paper is to provide insight into the applicability of proposed SNNI approaches for edge intelligence in both edge computing and IoT setups. Our goal is to achieve a better understanding of the factors that could potentially limit the deployment of SNNI for edge intelligence in edge computing and IoT setups. We focus on two

properties of such setups: (1) the importance of power and energy consumption, and (2) limited network bandwidth. We perform a set of experiments to investigate the power and energy consumption of two state-of-the-art SNNI approaches. In particular, we make the following contributions:

- We empirically compare the two SNNI approaches on two different NNs in terms of energy and power consumption on both the client and the server side.
- We investigate how power consumption changes over time during the SNNI process.
- We investigate the impact of the network bandwidth between client and server on SNNI latency and on power and energy consumption on both the client and the server side.

The results that we report in this paper offer many important insights into the applicability of existing SNNI approaches in edge or IoT setups, the factors for selecting one or the other SNNI approach, as well as pointers for future research, thus making progress towards secure edge intelligence.

The rest of the paper is organized as follows. Section 2 introduces the necessary background knowledge. NNs, SNNI and the two approaches are discussed here. The logical design and technical setup, as well as the evaluation metrics of the experiments are discussed in Section 3. In Section 4, the initial measurements of both NNs is described and compared. In the second part, the relation between the bandwidth and the power consumption is explained. Section 5 presents the lessons learned, while Section 6 discusses related work, and Section 7 concludes the paper.

2 Preliminaries

This section summarizes the necessary background in neural networks, secure neural network inference, the CrypTFlow2 and Cheetah SNNI approaches, and the benchmark NNs that will be used later in this paper.

2.1 Neural Networks

A feed-forward neural network (NN) computes a function, mapping an input vector to an output vector. The NN consists of a sequence of *layers*, numbered from 1 to L. The input of the first layer is the input of the NN, and the output of the last layer is the output of the NN. For $1 \leq i \leq L-1$, the output of layer i serves as input of layer $i+1$. Each layer computes a function, and the function computed by the NN is the composition of the functions computed by the individual layers.

Layers are often categorized as linear and non-linear. Typical *linear layers* are:

- Fully-connected (FC): computes the output vector by multiplying the input vector with a given weight matrix.
- Convolution (CONV): follows a more sophisticated definition, but can also be cast as multiplication of the input vector with a given matrix.
- Batch normalization (BN): scales each element of the input vector by multiplying it with a given number and adding another given number to the result.

Typical *non-linear layers*:

- Activation functions: the same real function is applied to each element of the input vector. An often-used example is the ReLU function, which maps x to

$\max(x, 0)$.

- Pooling functions: each element of the output vector is computed by applying a given function to a subset of the input numbers. An often-used example is max-pooling, in which the output number is the maximum of the considered input numbers.

A NN typically contains many parameters, such as the matrices in FC and CONV layers. The aim of the *training phase* is to find appropriate values for these parameters, so that the NN computes the desired function, or at least an approximation of it. For this purpose, the NN is evaluated on inputs with a known desired output. Comparing the actual output of the NN with the desired output, the parameters can be tuned such that the actual output of the NN becomes closer to the desired output.

During the *inference phase*, the NN is applied to new inputs for which no desired output is known.

In many cases, the NN is used for classification, i.e., to determine which of a given set of classes the input belongs to. In such cases, the quality of the trained NN can be quantified using its *accuracy*: the ratio of inputs in a validation dataset for which the NN outputs the correct class.

2.2 Secure Neural Network Inference

In Machine Learning as a Service (MLaaS), a service provider has trained a NN and offers inference with this NN as a service. A client can use the service to obtain the output of the NN on a given input provided by the client.

Such a scenario may involve some secrecy constraints. The input and the output may constitute sensitive information that the client wants to keep secret. The parameters of the NN constitute the intellectual property of the service provider which the service provider may not be willing to disclose. Thus, the secure neural network inference (SNNI) problem is to compute the output of the NN in such a way that the client only learns the output, but nothing about the parameters of the NN (beyond what the output might reveal), while the service provider learns nothing about the input nor the output.

In recent years, several SNNI approaches have been proposed [MWC+2023]. They use sophisticated cryptographic protocols to solve the SNNI problem.

In particular, SNNI can be cast as a secure 2-party computation (2PC) problem. 2PC means that two parties, typically denoted as Alice and Bob, compute $f(x, y)$, where f is a publicly known function, x is Alice's secret input and y is Bob's secret input. A 2PC protocol guarantees that neither party learns anything about the other party's input, beyond what the output reveals. In the case of SNNI, Alice is the client, Bob is the service provider, x is the input to the NN, y is the set of NN parameters (such as the elements of the weight matrices), and f is the evaluation of the NN with parameters y on input x.[2]

Several 2PC techniques exist that can be leveraged for SNNI. One such technique is *additive secret-sharing*. A number x is secret-shared between two parties by generating two numbers x_1 (given to one party) and x_2 (given to the other party) in such a way that

[2] In this formulation of the SNNI problem, the structure of the NN is publicly known. A possible other formulation would entail that the structure of the NN is part of Bob's secret input. However, we will stick to the formulation given above because it allows more efficient implementations and is in line with the assumptions underlying most of the state-of-the-art SNNI approaches.

$x_1 + x_2 = x$ but otherwise x_1 and x_2 are random. It is possible to perform simple operations, such as addition and multiplication, on secret-shared numbers using appropriate protocols. Such protocols can be composed to compute more complicated functions on secret-shared numbers.

Another useful 2PC primitive is *Oblivious Transfer* (OT). In OT, Alice has two messages m_0 and m_1, while Bob has a selection bit $b \in \{0,1\}$. The aim is that Bob gets the message m_b, without learning anything about the other message, while Alice learns nothing. OT and its various extensions (e.g., for more than two messages) can be used as building blocks in many 2PC protocols. For example, the most efficient known protocols for computing some non-polynomial functions, such as ReLU, on secret-shared numbers, use OT.

Homomorphic encryption (HE) can also be used in the context of 2PC. An encryption scheme is called partially homomorphic if at least one operation on plaintexts (e.g., addition or multiplication) can be evaluated homomorphically, i.e., by appropriate manipulation of the corresponding ciphertexts. Fully homomorphic encryption (FHE) schemes support the homomorphic evaluation of both addition and multiplication. In a 2PC setting, FHE allows Alice to encrypt her input and send it to Bob, who can evaluate any polynomial function on Alice's encrypted input, without learning Alice's secret input. After sending the encrypted result to Alice, she can decrypt it to obtain the result in plaintext.

Different 2PC protocols have different advantages and disadvantages. For example, homomorphic encryption is appropriate for evaluating linear layers of a NN, while for non-linear layers, OT-based protocols are more appropriate [HLH+2022]. Hence, many SNNI approaches combine different 2PC protocols on a per-layer basis. They use additive secret-sharing as an overarching scheme: as an invariant, the inputs to each layer are secret-shared between the parties, and the protocol for the layer yields its output again secret-shared between the parties. This way, different protocols can be composed, paving the way for a variety of SNNI approaches. Under appropriate conditions, it can be proven that such protocols satisfy the secrecy requirements [HLH+2022].

2.3 CrypTFlow2 / SCI$_{HE}$ / SCI$_{OT}$

CrypTFlow2 [RRK+2020] is a typical representative of SNNI approaches combining different types of protocols for different types of NN layers, using additive secret-sharing as the overarching method. The major contribution of CrypTFlow2 is a set of sophisticated and highly optimized protocols for non-linear layers (ReLU, MaxPool, ArgMax) and division, using OT. (Division is used to maintain a fixed bitlength and in MeanPool layers.)

For linear layers, CrypTFlow2 implements two different protocols, one based on HE and another based on OT.

CrypTFlow2 ensures that its output is bitwise equal to the output of cleartext inference, i.e., the used security protocols do not distort the output in any way. For this reason, the SNNI approach of CrypTFlow2 is also called SCI, an abbreviation for Secure and Correct Inference. More precisely, CrypTFlow2 provides two SNNI approaches, denoted as SCI$_{HE}$ and SCI$_{OT}$. SCI$_{HE}$ and SCI$_{OT}$ only differ in whether they handle linear layers with HE or with OT.

In this work, we use SCI$_{HE}$ because it incurs less communication and is thus more efficient in the case of limited bandwidth than SCI$_{OT}$ [RRK+2020].

2.4 Cheetah

Cheetah [HLH+2022] is one of the most recent SNNI approaches. It provides a highly optimized solution for SNNI. Cheetah was shown to be significantly faster than previous approaches, such as CrypTFlow2 [RRK+2020], and able to perform secure inference with large neural networks such as ResNet50 in less than 2.5 minutes in WAN settings [HLH+2022].

In line with most other recent SNNI approaches, Cheetah (1) uses additive secret-sharing as the overall 2PC scheme; (2) uses signed fixed-point arithmetic; (3) assumes that the NN architecture is known to both parties.

The main novelties of Cheetah are:

- For linear layers (fully-connected, convolution, batch normalization), Cheetah provides a new technique using homomorphic encryption. The used homomorphic encryption scheme is based on polynomials encoded as vectors. By arranging parameters carefully into the list of coefficients, homomorphic operations can be efficiently implemented as polynomial multiplication. Additionally, in contrast to previous approaches, the used arithmetic is not in the ring Z_p (for a prime p), but in the ring Z_{2^k} (for a positive integer k), which makes implementation more efficient and also makes conversions between different rings unnecessary when switching between layers.
- For non-linear layers, Cheetah uses improved versions of CrypTFlow2's protocols. For truncation (i.e., division by a power of 2), Cheetah allows a small error, enabling a significant speedup of the protocol. A new OT extension protocol (silent OT) is used in the protocols of both truncation and comparison, with the latter being the basis for multiple further protocols, such as for ReLU. The truncation protocol can be further accelerated if the most significant bit of the input is known, which is the case for example after a ReLU.
- Some further, smaller optimizations are introduced, for the special case of a convolution layer followed by a batch normalization layer, and for decreasing the amount of data transfer in the protocols based on homomorphic encryption.

Cheetah was experimentally evaluated using cloud servers with 2.70 GHz CPU and 16 GB RAM [HLH+2022]. Two network setups were used in the evaluation: LAN with 384 MBps bandwidth and 0.3 ms latency, and WAN with 44 MBps bandwidth and 40 ms latency. Using multiple NNs (including SqueezeNet, ResNet32, ResNet50, and DenseNet121), Cheetah was compared to and found superior to CrypTFlow2. (For a fair comparison, the code in SCI_{HE} was modified to adopt the latest versions of the used libraries.) In addition, Cheetah was also compared to SecureQ8, a recent 3-party protocol. Cheetah proved faster than SecureQ8 in the WAN setting; however, in the LAN setting, SecureQ8 was faster than Cheetah.

2.5 Benchmark neural networks

In our experiments, we use two benchmark neural networks that were used in the past to evaluate CrypTFlow2 and Cheetah. Both of them are convolutional neural networks for image classification, but they significantly differ in terms of their size and their types of layers. The two networks are:

- SqueezeNet [IHM+2016]: a network of moderate size that was specifically created to achieve relatively high accuracy with a strictly limited size (less than 500 thousand trainable parameters). It consists of
  - 26 convolutional layers
  - 26 ReLU layers
  - 3 MaxPool layers
  - 1 AvgPool layer
- ResNet50 [HZR+2016]: with over 23 million trainable parameters, ResNet50 is significantly larger than SqueezeNet. Moreover, ResNet50 features a richer set of layer types:
  - 53 convolutional layers
  - 49 batch normalization layers
  - 98 truncation layers
  - 49 ReLU layers
  - 1 fully-connected layer
  - 1 MaxPool layer
  - 1 AvgPool layer
  - 1 ArgMax layer

The protocols implementing secure inference with these two neural networks are readily available in both CrypTFlow2 and Cheetah.

3 Design of experiments

In this section, we describe what we want to experimentally investigate and how we are going to do that. We first describe our experiments on the logical level and define the evaluation metrics, followed by the description of the technical setup used in the experiments.

3.1 Logical design of the experiments

Existing SNNI approaches like CrypTFlow2 and Cheetah were evaluated in a cloud environment, using powerful computers on both server and client side, and assuming a wired connection (either LAN or WAN) between the two computers. As evaluation metrics, accuracy, latency (i.e., the time needed to perform an inference), and the amount of transferred data were used.

In contrast, we are interested in the applicability of SNNI approaches in an edge computing or IoT environment. Applying SNNI in such an environment entails several challenges, from which we focus on two in this paper:

- In edge computing, energy consumption and power consumption are very important. Many edge devices are battery powered, which may limit both the momentarily power draw of the device and the available energy budget for the inference process.
- In edge computing, client and server machines may communicate over legacy wireless connections (e.g., 3G/4G), offering significantly lower bandwidth than what is available in typical wired LAN or WAN environments.

To investigate these aspects, we perform two sets of experiments. In the first set of experiments, we compare two SNNI approaches (CrypTFlow2 and Cheetah), using two NNs (SqueezeNet and ResNet50), in terms of energy consumption and power consumption. We measure energy consumption and power consumption separately for

the client computer and the server computer, because in many edge computing scenarios, energy and power constraints are more stringent for the client than for the server. We also look at how power consumption changes over time during the inference process, and how it varies between different executions of the inference process.

The second set of experiments focuses on the effect of the bandwidth available between client and server. For this purpose, we vary the available bandwidth, and compare again the two SNNI approaches with the two NNs. We measure the duration of the inference process as well as total energy consumption and average power consumption on the client and on the server computer.

According to the experimental results of [HLH+2022], Cheetah was clearly superior to CrypTFlow2, in every tested situation and according to all considered metrics. An interesting question is whether this holds true in the extended set of situations considered in our experiments and considering our extended set of metrics. The results might change the preference for choosing a specific SNNI approach in a given situation.

3.2 Evaluation metrics

During our experiments, we collect the following metrics:

- Latency: the duration of the secure inference process for one input. This is measured as the difference in wall-clock time between the time when the client starts and when the client finishes its part in the inference process. Note that the server can start earlier than the client but we disregard the time when the server is just waiting for the client, since the actual protocol execution starts only when the client joined. At the end of the protocol server and client finish at about the same time. Unit: second (s).
- Average power consumption on the server side: the additional power consumption of the server computer caused by the SNNI program, averaged over the whole duration of the secure inference process. Note that only the duration of the protocol execution is taken into account, i.e., the setup time of the server before the client joins is disregarded. Unit: Watt (W).
- Average power consumption on the client side: analogously to the server-side power consumption, but measured on the client computer.
- Total energy consumption of the server side: the integral of the instantaneous power consumption over the whole duration of the secure inference process. In line with the above, only the duration of the protocol execution is taken into account, i.e., the setup time of the server before the client joins is disregarded. Unit: Joule (J).
- Total energy consumption on the client side: analogously to the server-side energy consumption, but measured on the client computer.

3.3 Technical setup

For performing the experiments, we use two identical computers, one as server and one as client. Both computers have the following specification:

- CPU: Intel Xeon E-2378 @ 2.60 GHz, 8 cores
- RAM: 64 GB
- Network controller: Intel I350
- Operating system: Ubuntu 22.04.2 LTS (GNU/Linux 5.15.0-71-generic x86_64)

During the execution of our experiments, the two computers are used for no other purpose; thus, no other workload is running on them.

For measuring power and energy consumption, we use the Linux Hardware Monitoring (hwmon) interface[3]. Through this interface, we can obtain a new reading every second, giving the average power consumption of the system in the last second with a resolution of 1.0 W. By periodically measuring this value, we can calculate an approximation of the total energy consumption and the average power consumption of the system for a longer period of time.

The two computers are connected by a direct cable connection. For simulating different network bandwidths, we use the Linux tc (traffic control) utility to throttle the data rate of the client computer. For measuring bandwidth, we use the iperf tool (version 2.1.5)[4].

We use the latest version of Cheetah, which is commit 0b63d6f from 02 March 2023[5]. This codebase also includes a version of CrypTFlow2, which is also used in our experiments. We do not change any parameters of Cheetah or CrypTFlow2.

4 Experimental Results

In this section, we describe the results of our experiments. We start with some initial measurements to characterize the experimental environment in terms of bandwidth and idle power consumption. Then we conduct experiments on the power and energy consumption of Cheetah and CrypTFlow2 and report our findings in terms of both aggregated numbers (over the course of an inference) and the development of power consumption over time during inference. Finally, we analyze the impact of throttling the network bandwidth on power and energy consumption.

4.1 Initial measurements

We start our experiments by first measuring some basic properties of our experimental environment.

Without using any bandwidth throttling, the network connection between the client and server computers is characterized by the following parameters:

- Bandwidth between client and server: 941 Mbps
- Round-trip time between client and server: 0.6 ms

In addition, we measure the idle power consumption of both computers. For this purpose, we measure power consumption with a frequency of 1 sec, for a period of 10 minutes. From these 600 measurements, we compute the mean and standard deviation. The results are shown in Table 1.

[3] https://www.kernel.org/doc/Documentation/hwmon/sysfs-interface
[4] https://sourceforge.net/projects/iperf2/
[5] https://github.com/Alibaba-Gemini-Lab/OpenCheetah/commit/0b63d6f2cfe979a446a7999ee78d705b6ef5ab81

Table 1. Idle power consumption of the test machines

	Mean	Standard deviation
Server	29.62 W	0.54 W
Client	29.73 W	0.68 W

From the low standard deviation, we conclude that the idle power consumption of both server and client is fairly stable. Therefore, we can measure the power consumption of running a program by measuring the power consumption of the system while the program is running and subtracting from this value the mean idle power consumption of the system. We report this difference as power consumption, and also compute energy consumption on the basis of this difference in the experiments described next.

4.2 Power and energy comparison

In our first experiment, we use the experimental setup as described above, without any modification of the bandwidth. We measure the average power consumption and the total energy consumption of performing one secure inference with different SNNI solutions and different neural networks.

Table 2 presents an overview of the results. Each reported number is the result of averaging the metrics from 10 runs. The numbers under “Average power” are the result of additionally also averaging over the duration of the inference process.

Table 2. Comparison of performing one secure neural inference with Cheetah and SCI_{HE} on two neural networks in terms of power and energy consumption. The shown numbers are the average of 10 measurements

Neural network	SNNI approach	Participant	Latency	Average power	Total energy
SqueezeNet	Cheetah	Client	25.5 s	29.7 W	772.1 J
		Server		27.91 W	725.7 J
	SCI_{HE}	Client	79.9 s	9.96 W	796.9 J
		Server		25.33 W	2,026.57 J
ResNet50	Cheetah	Client	107.9 s	25.53 W	2,757.61 J
		Server		42.31 W	4,569.9 J
	SCI_{HE}	Client	371 s	8.83 W	3,276.33 J
		Server		45.12 W	16,741.35 J

In line with the results reported in [HLH+2022], we can see that Cheetah is significantly faster than SCI_{HE}, on both neural networks. In light of this, it is also no surprise that the overall energy consumption (i.e., server and client together) of Cheetah is significantly less than that of SCI_{HE}.

However, investigating the server's and the client's energy consumption separately, we can see an interesting difference between Cheetah and SCI_{HE}. In the case of Cheetah, the distribution of energy consumption between client and server is balanced, whereas in SCI_{HE}, the energy consumption of the server is much larger than that of the client.

Investigating the average power consumption leads to even more interesting findings. In terms of server-side power consumption, Cheetah and SCI_{HE} have roughly equal results. However, as a consequence of the large power imbalance of SCI_{HE} between client and server, the client-side power consumption of SCI_{HE} is significantly lower than that of Cheetah.

This leads to an interesting trade-off between client-side power consumption on the one hand and latency and overall energy consumption on the other hand. In a setup where client-side power consumption is not a major concern, Cheetah is more appropriate because of its lower latency and lower overall energy consumption. However, if client-side power consumption is a major limiting factor (especially for battery-powered client devices), then CrypTFlow2 is more appropriate. This is a new insight that was not visible in the experimental evaluation in existing work.

4.3 Timeseries analysis

To obtain a more detailed understanding of the power consumption characteristics of SNNI, we now look at how power consumption evolves over time during the secure inference process.

The aim is to present the distribution of the power consumption at given points in time obtained from $k = 10$ measurements. A difficulty lies in the fact that the latency of the entire inference process is not constant across different runs. Therefore, we use the following methodology for determining the average power consumption at given points in time (see also Fig. 2):

1. We perform k runs, storing the power readings for every second in every run, leading to k time series. Let t_i denote the duration of the ith run.

2. We determine the average latency $t^* = \frac{1}{k}\sum_{i=1}^{k} t_i$.
3. We modify the time scale of each time series in such a way that their duration equals the average latency. That is, the time stamp of every power reading in time series i is multiplied by t^*/t_i.
4. We split the interval $[0, \lceil t^* \rceil]$ into $\lceil t^* \rceil$ time slots with a size of 1 second.
5. For each of these time slots, we determine the average and the standard deviation of the measured power consumption values whose time stamp (after the modification of step 3) falls into the given time slot. This is the resulting time series that we visualize.

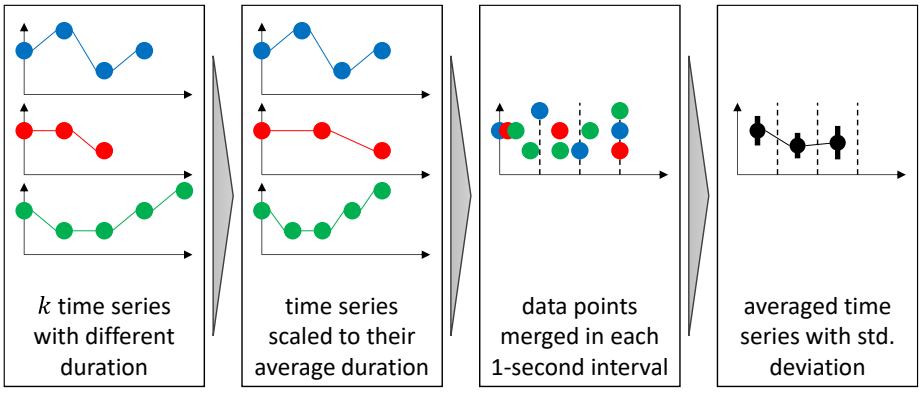
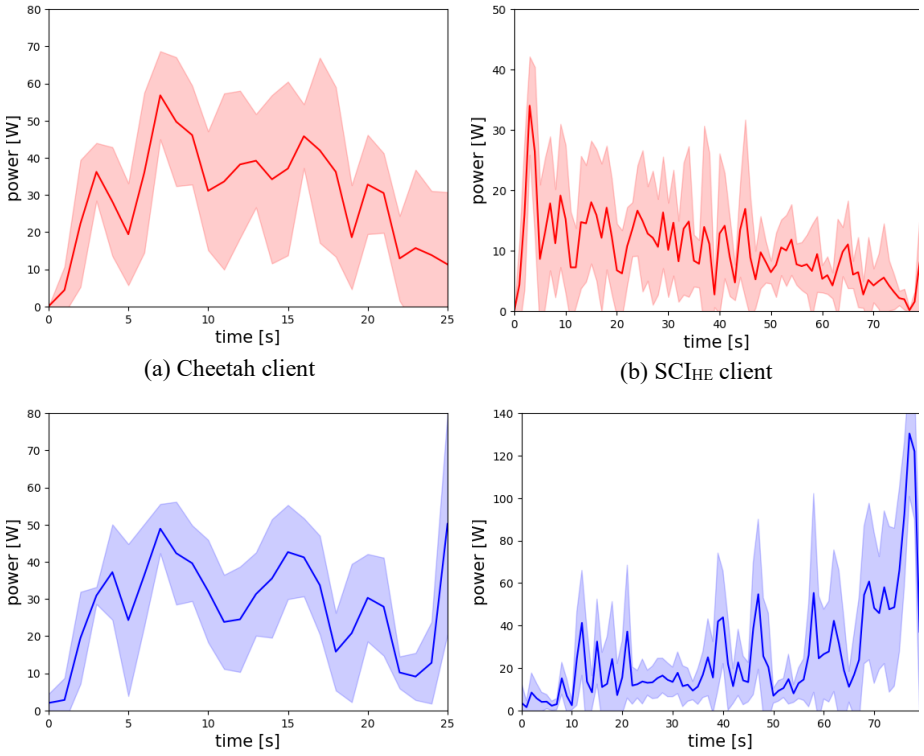


Fig. 2. Methodology for aggregating k time series with different duration

The results are shown in Fig. 3 for SqueezeNet and in Fig. 4 for ResNet50. A striking property of these plots is the large variance of the power consumption values over time. Short periods of low and high power consumption alternate quickly. In addition, when taking a closer look, it becomes apparent that in some periods the power consumption of the server and that of the client are closely correlated, whereas in other periods they are not.

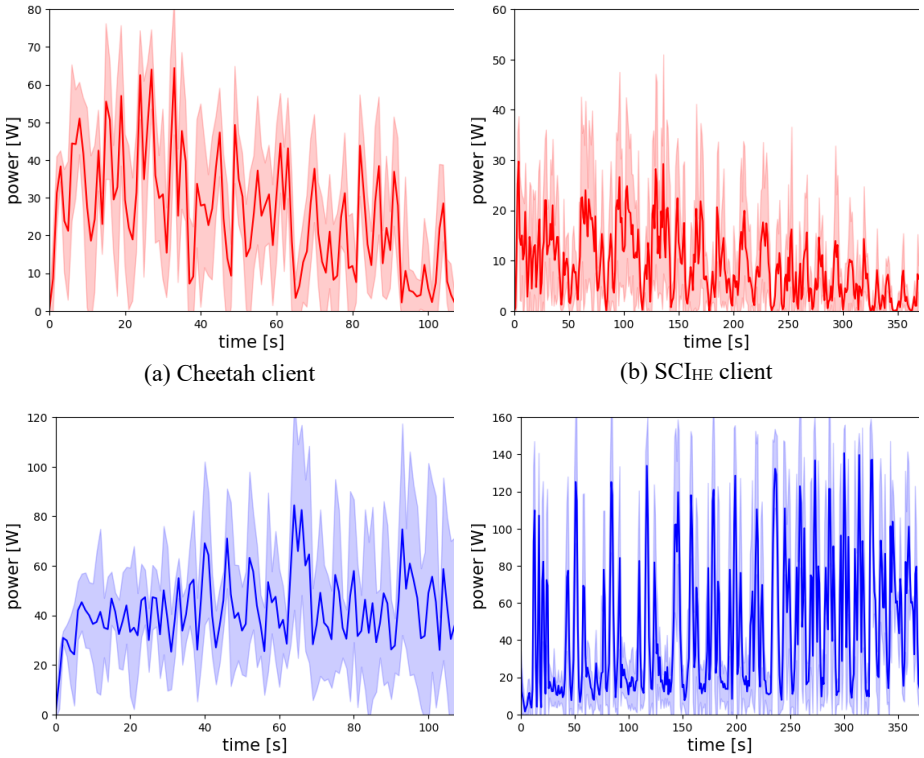


(a) Cheetah client (b) SCI$_{HE}$ client

(c) Cheetah server (d) SCI$_{HE}$ server

Fig. 3. Power consumption over time while performing secure inference with the SqueezeNet neural network. Each plot shows results distilled from 10 runs: the line shows the mean, while the shaded area shows the standard deviation around the mean. The plots on the left-hand side show results of Cheetah, while the plots on the right-hand side show results of SCI$_{HE}$, in both cases separately for the client (upper plot) and the server (lower plot). It should be noted that the scale of the axes is different in the different plots

This seemingly strange phenomenon is actually not at all surprising. Remember that both CrypTFlow2 and Cheetah use different protocols for the different types of layers. The different protocols differ significantly in how much computation they require from the client and the server. Typically, linear and non-linear layers alternate, and the corresponding protocols are completely different in terms of being computation intensive or communication intensive. In some protocols, the load on server and client is similar, whereas in other protocols, the load distribution is strongly asymmetric. Also, layers of the same type can have different size, which also can cause significant differences in power consumption.



(a) Cheetah client (b) SCI$_{HE}$ client

(c) Cheetah server (d) SCI$_{HE}$ server

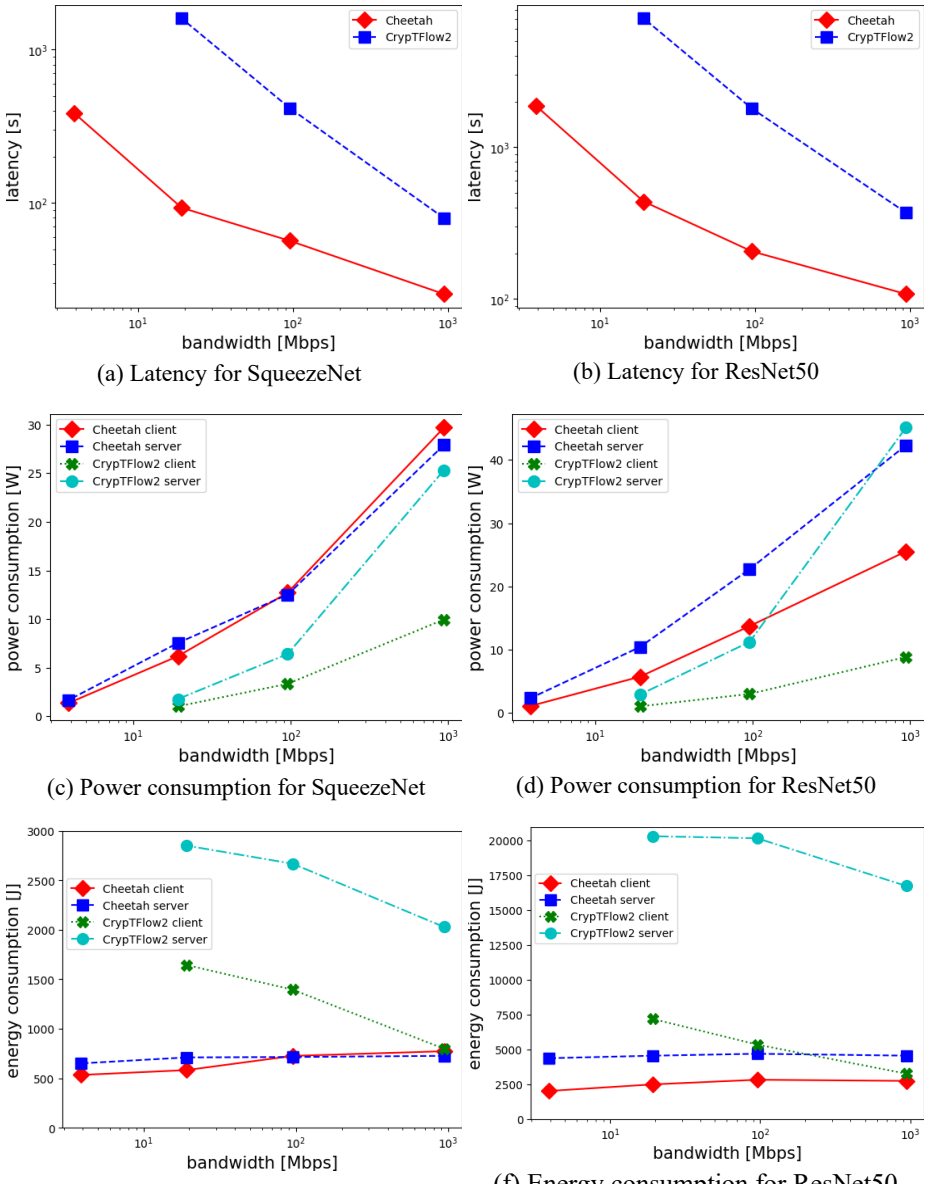
Fig. 4. Power consumption over time while performing secure inference with the ResNet50 neural network. The same remarks apply as in the case of Fig. 3

4.4 Impact of the bandwidth

By throttling the data rate of the client, we emulate computer networks with smaller and smaller bandwidth, and measure the impact on latency, average power consumption, and total energy consumption. Specifically, we perform measurements with the following bandwidth values:

- 941 Mbps (no throttling)
- 95.6 Mbps (result of specifying a throttling target of 100 Mbps)
- 19.2 Mbps (result of specifying a throttling target of 20 Mbps)
- 3.89 Mbps (result of specifying a throttling target of 4 Mbps)

We performed the experiments with both Cheetah and CrypTFlow2. However, based on the trend observed for the first three bandwidth values, we decided to not run CrypTFlow2 for the last (most constrained) bandwidth value. We had two reasons for this. First, the latency of CrypTFlow2 was increasing rapidly as the bandwidth was being reduced, which not only made our experiments last very long, but also made it clear that CrypTFlow2 was not practical in a system with this low bandwidth between client and server computer. Second, the average power consumption of CrypTFlow2 was decreasing quickly as the bandwidth was reduced, and it approached the range of measurement errors in our power readings (see the standard deviation in idle power measurement in Section 4.1), which made further power measurement unreliable.



(a) Latency for SqueezeNet

(b) Latency for ResNet50

(c) Power consumption for SqueezeNet

(d) Power consumption for ResNet50

(e) Energy consumption for SqueezeNet

(f) Energy consumption for ResNet50

Fig. 5. Impact of the bandwidth between client and server on the latency, average power consumption, and total energy consumption of the secure inference process. Note the logarithmic scale of the horizontal axis in all plots; additionally, in the latency plots, also the vertical axes have a logarithmic scale. All reported numbers are the average from 10 runs. CrypTFlow2 was not run for the smallest considered bandwidth.

The results are shown in Fig. 5. From subfigures (a) and (b) we can establish that with decreasing bandwidth between client and server, the latency of the secure inference process grows quickly, for both SNNI solution approaches and for both NNs. This is not surprising, given the large amount of data transfer between client and server during

the secure inference protocols. We can also observe that the latency difference between Cheetah and CrypTFlow2 (to Cheetah's advantage) grows with decreasing bandwidth. This is again no surprise since Cheetah causes less data transfer between client and server than CrypTFlow2, as shown in [HLH+2022].

The results on average power consumption (subfigures (c) and (d)) are more interesting. We can observe consistently across all measurements that reducing the bandwidth leads to a reduction of average power consumption. This is probably due to the decreased bandwidth slowing down the whole secure inference protocol: as the bandwidth decreases, both client and server spend more time waiting for I/O, effectively reducing the rate with which they can perform computation.

For similar reasons, since CrypTFlow2 has to transfer more data between client and server than Cheetah, CrypTFlow2 has lower average power consumption than Cheetah. This is superposed by a higher asymmetry between client and server in terms of computation, and thus average power consumption, for CrypTFlow2 than for Cheetah. As a result, the advantage of CrypTFlow2 over Cheetah in terms of average power consumption is much more significant on the client side than on the server side.

The results for total energy consumption (subfigures (e) and (f)) are less conclusive. For CrypTFlow2, reducing the bandwidth between client and server seems to lead to higher total energy consumption, whereas for Cheetah, reducing the bandwidth seems to lead to no significant change or even to a slight reduction of total energy consumption. The reasons for this may require further, more in-depth analysis of the involved protocols. We can also observe that CrypTFlow2 is competitive with Cheetah in terms of client energy consumption, especially for high bandwidth values; however, in terms of server energy consumption, Cheetah has a significant advantage.

5 Discussion

In this section, we discuss the lessons learned from our experiments, the consequences for future research, and potential threats to the validity of our findings.

5.1 Lessons learned

In the following, we distill our key findings from the experiments.

Energy and power consumption of SNNI do matter. As we have seen, secure inference can lead to significant energy consumption and significant instantaneous power consumption. In an edge computing or IoT setup, this can be problematic. Different SNNI approaches have different characteristics in terms of total energy consumption and average power consumption. Thus, depending on the specific constraints of the edge / IoT environment, one or the other SNNI approach may be more appropriate. While research in the field of SNNI so far focused mainly on reducing latency, constraints on power or energy consumption may be more stringent in some environments than constraints on latency.

Energy consumption and power consumption need to be considered separately. Although there is a clear connection between power consumption and energy consumption, low power consumption does not guarantee low energy consumption, and also low energy consumption does not guarantee low power consumption. For example, we have seen cases where CrypTFlow2 leads to lower average power consumption, but Cheetah leads to lower total energy consumption. Therefore, it is important to consider if, in a given target environment, average power consumption or total energy consumption is more important, and this may necessitate different design choices in the

used SNNI protocols.

Client and server power consumption can differ significantly. We have seen that, in terms of the balance between client-side and server-side power consumption, CrypTFlow2 is much more asymmetric than Cheetah, meaning that the CrypTFlow2 client consumes less power than the server, and also less than the Cheetah client. This can give an advantage to CrypTFlow2 in environments where the power draw on the client side is strictly constrained. This consideration yields a more nuanced picture about the advantages and disadvantages of different SNNI approaches, something that was not considered in previous evaluations such as in [HLH+2022].

Power consumption varies significantly over time. The instantaneous power consumption fluctuates widely over time. This was consistently observed for both SNNI approaches, for both NNs, and for both client and server. The fluctuation is logical, given the varying types and sizes of layers that make up a NN, and the different protocols used for different layers. However, the strong fluctuations make it challenging to schedule such processes.

Network bandwidth has major impact. We have found that the bandwidth between client and server significantly influences all considered metrics. Reducing the bandwidth leads to an increase in latency, a decrease in average power consumption, and can have different influence on total energy consumption. Bandwidth may also influence which SNNI approach performs best according to a given metric. In existing work, typically only a LAN and a WAN setup were tested, where even the WAN setup has relatively high bandwidth. Wireless networks used in many IoT and edge computing environments have lower bandwidth, which was typically not considered in the evaluation of existing works, thus potentially missing important insights into the suitability of different SNNI approaches for low-bandwidth environments.

5.2 Consequences for future research

Our findings have shown the importance of energy and power consumption in SNNI, as well as the large impact of the network bandwidth between client and server. These findings have important consequences on the design, implementation, and evaluation of SNNI approaches that should be taken into account in future work.

First of all, more research is needed to understand the exact **requirements** on energy and power consumption of secure inference in typical SNNI use cases. In particular, it is important to understand the relative importance of requirements concerning latency, power consumption on the client respectively the server side, and energy consumption on the client respectively the server side. Beside the relative importance of these metrics, it is also important to understand the acceptable ranges for these metrics, e.g., the maximum acceptable client-side power consumption in typical SNNI use cases in edge computing.

Second, more research is needed to **devise SNNI methods** optimized for low power consumption and/or low energy consumption and for limited bandwidth. This may include optimizing existing techniques for such environments, for example by shifting some of the load from the client to the server to better support battery-powered clients. On the other hand, completely new methods may be needed to achieve improvements on all considered metrics at the same time [CSM2023].

Finally, we need **experimental evaluation and comparison** of proposed SNNI approaches in realistic settings – for example, with low bandwidth between client and server – paying attention to the metrics relevant in IoT or edge computing settings, like

client-side energy consumption. That is, in contrast to the current practice of latency-focused evaluation in LAN and WAN environments, also power consumption and energy consumption should be considered as additional metrics, and bandwidth limitations of typical wireless networks should also be tested. More realistic evaluation and comparison of SNNI approaches may lead to a preference for other methods than the one that performs best in terms of latency in a LAN or WAN environment.

In addition to these direct consequences of our findings, it is also worth investigating whether an **artificial cap on the bandwidth** between client and server can help reduce power consumption if needed. Our experimental results suggest that this is possible. Thus, if there is a strict limit on power consumption, artificially throttling the network bandwidth could help keep the power consumption low. This way, a trade-off between power consumption and latency could be controlled directly via the bandwidth.

5.3 Threats to validity

We tried to perform our study carefully, but some threats to the validity of our study still remain. In the following, we review the most important threats to internal and external validity.

Internal validity. Our measurements of power consumption and energy consumption assumed that the used machines have a stable level of idle power consumption, and all additional power consumption can be attributed to the secure inference process. It is conceivable that some other programs may also create additional power consumption at some points in time, thus influencing the power consumption that we attribute to the secure inference process. To mitigate this threat, we observed the level of idle power consumption over a longer period of time, without experiencing significant deviations (see Section 4.1). In addition, all of the reported numbers are the average of 10 measurements, thus decreasing the implication of random effects.

Similarly, the way we measure latency also assumes that no other processes take a significant amount of time, compared to the latency of SNNI. Again, random effects like the occasional activation of some system services could impact the results. We mitigated this threat by averaging latency measurements over 10 runs. We did not observe major fluctuations in latency among the 10 runs in any of our experiments.

External validity. It is not clear to what extent our findings transfer to other setups. To improve generalization possibilities from our findings, we performed experiments with two SNNI approaches and with two NNs. However, it is possible that repeating our experiments with other SNNI approaches, other NNs, on other computers, and using other computer networks, would yield significantly different results. Further experimental research could help establish an improved coverage of relevant setups. For example, it would be interesting to investigate the effects of a higher round-trip time between server and client, as well as scenarios in which multiple clients connect to the same server.

6 Related work

In recent years, the SNNI problem has received considerable research attention and several SNNI approaches have been proposed [MWC+2023]. CryptoNets was probably the first approach to offer end-to-end protection in the NN inference process, although in a very limited setting and with huge overhead [GDL+2016]. Subsequent work aimed mainly at reducing the overhead of SNNI. While the CryptoNets approach was based on homomorphic encryption, other work like DeepSecure used secure multi-party

computation protocols, such as garbled circuits [RRK2018]. Gazelle combined homomorphic encryption and secure multi-party computation, so that each layer of the NN can be evaluated with the most efficient protocol [JVC2018]. SecureML also used secure multi-party computation, but proposed a different setup, in which the actual protocol is carried out by two non-colluding servers, and the client only provides the input and collects the output [MZ2017]. Falcon used even three servers to further improve efficiency [WTB+2021]. XONN used binary neural networks (i.e., NNs in which each weight, bias, and activation value is either 1 or -1) to enable more efficient secure multi-party protocols [RSC+2019]. Delphi improved the protocols of Gazelle with several optimizations [MLS+2020]. CrypTFlow created a framework for automatically turning TensorFlow code into a secure multi-party computation protocol implementation [KRC+2020]. CrypTFlow2 extended the CrypTFlow framework with an efficient SNNI approach based on a combination of homomorphic encryption and different secure multi-party protocols [RRK+2020]. Cheetah further optimized the protocols of CrypTFlow2 to yield one of the most efficient SNNI implementations to date [HLH+2022].

All of the above works assumed a powerful client device with sufficient network bandwidth to the server. The proposed approaches were typically evaluated in a setting where both client and server were powerful cloud instances connected by a LAN or WAN. The constraints of typical edge computing or IoT setups were not considered. Also, power and energy consumption were not considered.

Some researchers investigated the SNNI problem in an edge computing, mobile computing, or IoT context. Instead of using compute-intensive cryptographic protocols, a possible approach that was suggested is to split the evaluation of the NN between client and server in such a way that the client evaluates the first couple of layers and sends the resulting features to the server which then completes the inference [OSS+2020]. However, this approach comes with no security guarantee, and the empirical experience showed that many layers have to be processed by the client in order to sufficiently constrain the leakage of information about the input to the server. Another proposed approach consisted of using secure multi-party computation using two edge servers [HLF+2021]. In addition to assuming that the two edge servers do not collude, this approach also required a trusted third party to perform certain preprocessing tasks. A completely different approach is to perform the inference entirely on the client machine, in a trusted execution environment [HLL+2021]. However, this requires a relatively powerful client machine, with support for the appropriate technology on the hardware level.

To summarize, the efforts culminating in CrypTFlow2 and Cheetah have created secure and efficient SNNI approaches, but failed to address the constraints of edge computing and IoT setups. On the other hand, existing work on SNNI for edge and IoT setups has serious limitations. Thus, there is a need to investigate paths for transferring Cheetah-type SNNI approaches to edge and IoT setups. Our work is a step into this direction.

7 Conclusion

In this paper, we focused on some important aspects of edge intelligence by deploying SNNI in the context of edge computing or IoT applications: energy and power consumption, and the impact of the bandwidth between client and server. We used CrypTFlow2 and Cheetah, two state-of-the-art SNNI approaches, and performed

controlled experiments with them in a dedicated client-server environment.

Our experimental results revealed many interesting details. In particular:

- SNNI leads to significant energy consumption, with considerable differences depending on the used SNNI approach and NN.
- High energy consumption of SNNI does not necessarily mean high power consumption, and vice versa.
- Different SNNI approaches can differ significantly in terms of the balance between server-side power consumption versus client-side power consumption.
- Power consumption of SNNI may vary considerably over time.
- Network bandwidth has major impact on all considered metrics.

These insights can help choose the right SNNI approach in a given situation, considering the NN, power consumption constraints, network bandwidth etc.

The gained insights also help inform future research into edge intelligence using SNNI in edge computing and IoT environments. In particular, we need more research to better understand the relevant requirements (in terms of power consumption, energy consumption, bandwidth) in such environments, to devise SNNI methods specifically optimized for such environments, and more experimental evaluation and comparison of SNNI methods in such environments.

References

[AAM+2021] Ahvar, E., Ahvar, S., Mann, Z. Á., Crespi, N., Glitho, R., Garcia-Alfaro, J. DECA: A dynamic energy cost and carbon emission-efficient application placement method for edge clouds. *IEEE Access*, 9:70192-70213, 2021

[CA2023] Choudhury, D., Acharjee, T. A novel approach to fake news detection in social networks using genetic algorithm applying machine learning classifiers. *Multimedia Tools and Applications*, 82(6):9029-9045, 2023

[CSM2023] Chabal, D., Sapra, D., Mann, Z. Á. On Achieving Privacy-Preserving State-of-the-Art Edge Intelligence. *4th AAAI Workshop on Privacy-Preserving Artificial Intelligence (PPAI-23)*, 2023

[CT2022] Chung, J., Teo, J. Mental health prediction using machine learning: taxonomy, applications, and challenges. *Applied Computational Intelligence and Soft Computing*, Article ID 9970363, 2022

[GDL+2016] Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., Wernsing, J. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. *33rd International Conference on Machine Learning*, PMLR 48:201-210, 2016

[Glo2015] Global e-Sustainability Initiative (GeSI). #SMARTer2030 – ICT Solutions for 21st Century Challenges. Report, https://smarter2030.gesi.org/downloads/Full_report.pdf, 2015

[HLF+2021] Huang, K., Liu, X., Fu, S., Guo, D., Xu, M. A lightweight privacy-preserving CNN feature extraction framework for mobile sensing. *IEEE Transactions on Dependable and Secure Computing*, 18(3):1441-1455, 2021

[HLH+2022] Huang, Z., Lu, W., Hong, C., Ding, J. Cheetah: Lean and fast secure

two-party deep neural network inference. *31st USENIX Security Symposium (USENIX Security 22)*, pp. 809–826, 2022

[HLL+2021] Hou, J., Liu, H., Liu, Y., Wang, Y., Wan, P. J., Li, X. Y. Model Protection: Real-time privacy-preserving inference service for model privacy at the edge. *IEEE Transactions on Dependable and Secure Computing*, 19(6):4270-4284, 2021

[HZR+2016] He, K., Zhang, X., Ren, S., Sun, J. Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770-778, 2016

[IHM+2016] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., Keutzer, K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size. *arXiv preprint,* arXiv:1602.07360, 2016

[JVC2018] Juvekar, C., Vaikuntanathan, V., Chandrakasan, A. GAZELLE: A low latency framework for secure neural network inference. *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1651-1669, 2018

[KRC+2020] Kumar, N., Rathee, M., Chandran, N., Gupta, D., Rastogi, A., Sharma, R. CrypTFlow: Secure TensorFlow inference. *IEEE Symposium on Security and Privacy (SP)*, pp. 336-353, 2020

[LMD2021] Lachner, C., Mann, Z. Á., Dustdar, S. Towards understanding the adaptation space of AI-assisted data protection for video analytics at the edge. *IEEE 41st International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp. 7-12, 2021

[LXZ2015] Li, S., Xu, L. D., Zhao, S. The Internet of Things: a survey. *Information Systems Frontiers*, 17:243-259, 2015

[LYZ+2022] Li, Z., Yoon, J., Zhang, R., Rajabipour, F., Srubar III, W. V., Dabo, I., Radlińska, A. Machine learning in concrete science: Applications, challenges, and best practices. *npj Computational Materials*, Article 127, 2022

[Man2022] Mann, Z. Á. Security- and privacy-aware IoT application placement and user assignment. *Computer Security – ESORICS 2021 International Workshops*, pp. 296-316, Springer, 2022

[MLS+2020] Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., Popa, R. A. Delphi: a cryptographic inference service for neural networks. *29th USENIX Security Symposium*, pp. 2505-2522, 2020

[MWC+2023] Mann, Z. Á., Weinert, C., Chabal, D., Bos, J. W. Towards practical secure neural network inference: The journey so far and the road ahead. *ACM Computing Surveys*, https://dl.acm.org/doi/10.1145/3628446, 2023

[MZ2017] Mohassel, P., Zhang, Y. SecureML: A system for scalable privacy-preserving machine learning. *IEEE Symposium on Security and Privacy (SP)*, pp. 19-38, 2017

[OSS+2020] Osia, S. A., Shamsabadi, A. S., Sajadmanesh, S., Taheri, A., Katevas, K., Rabiee, H. R., Lane, N. D., Haddadi, H. A hybrid deep learning architecture for privacy-preserving mobile analytics. *IEEE Internet of*

Things Journal, 7(5):4505-4518, 2020

[QIU+2020] Qayyum, A., Ijaz, A., Usama, M., Iqbal, W., Qadir, J., Elkhatib, Y., Al-Fuqaha, A. Securing machine learning in the cloud: A systematic review of cloud machine learning security. *Frontiers in Big Data*, 3:587139, 2020

[RRK2018] Rouhani, B. D., Riazi, M. S., Koushanfar, F. DeepSecure: Scalable provably-secure deep learning. *Proceedings of the 55th Annual Design Automation Conference (DAC'18)*, art. 2, 2018

[RRK+2020] Rathee, D., Rathee, M., Kumar, N., Chandran, N., Gupta, D., Rastogi, A., Sharma, R. CrypTFlow2: Practical 2-party secure inference. *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 325–342, 2020

[RSC+2019] Riazi, M. S., Samragh, M., Chen, H., Laine, K., Lauter, K. E., Koushanfar, F. XONN: XNOR-based Oblivious Deep Neural Network Inference. *USENIX Security Symposium*, pp. 1501-1518, 2019

[SAF+2018] Syafrudin, M., Alfian, G., Fitriyani, N. L., Rhee, J. Performance analysis of IoT-based sensor, big data processing, and machine learning model for real-time monitoring system in automotive manufacturing. *Sensors*, 18(9):2946, 2018

[SKA+2023] Sarker, I. H., Khan, A. I., Abushark, Y. B., Alsolami, F. Internet of Things (IoT) security intelligence: a comprehensive overview, machine learning solutions and research directions. *Mobile Networks and Applications*, 28(1):296-312, 2023

[SRR+2021] Somani, S., Russak, A. J., Richter, F., Zhao, S., Vaid, A., Chaudhry, F., De Freitas, J. K., Naik, N., Miotto, R., Nadkarni, G. N., Narula, J., Argulian, E., Glicksberg, B. S. Deep learning and the electrocardiogram: review of the current state-of-the-art. *EP Europace*, 23(8):1179-1191, 2021

[TKC+2020] Tran-Dang, H., Krommenacker, N., Charpentier, P., Kim, D. S. Toward the Internet of Things for physical internet: Perspectives and challenges. *IEEE Internet of Things Journal*, 7(6):4711-4736, 2020

[TM2021] Timan, T., Mann, Z. Data protection in the era of artificial intelligence: trends, existing solutions and recommendations for privacy-preserving technologies. *The Elements of Big Data Value: Foundations of the Research and Innovation Ecosystem*, pp. 153-175, Springer, 2021

[WTB+2021] Wagh, S., Tople, S., Benhamouda, F., Kushilevitz, E., Mittal, P., Rabin, T. Falcon: Honest-majority maliciously secure framework for private deep learning. *Proceedings on Privacy Enhancing Technologies*, 2021(1):188-208, 2021

[ZWY+2022] Zhu, M., Wang, J., Yang, X., Zhang, Y., Zhang, L., Ren, H., Wu, B., Ye, L. A review of the application of machine learning in water quality evaluation. *Eco-Environment & Health*, 1(2):107-116, 2022