A Framework to Optimize the Energy Cost of Securing Neural Network Inference

Tanjina Islam
University of Amsterdam
Amsterdam, The Netherlands

Ana Oprescu
University of Amsterdam
Amsterdam, The Netherlands

Zoltán Ádám Mann
University of Amsterdam
Amsterdam, The Netherlands

Sander Klous
University of Amsterdam
Amsterdam, The Netherlands

***Abstract*—With the rise of deep neural networks (NNs), Machine Learning as a Service (MLaaS) gained significant attention. In MLaaS, a service provider offers inference with a pre-trained NN to clients, allowing them to obtain the inference output without the need for computationally intensive training. However, MLaaS introduces data privacy and security concerns for both clients and service providers. Clients' sensitive data in the input and output must be kept private, and the NN representing the service provider's intellectual property should not be shared with clients. Secure Neural Network Inference (SNNI) addresses these concerns by ensuring that the client learns only the output, and the service provider remains oblivious to the input and output.**

Many SNNI approaches proposed in recent years are mainly based on advanced cryptographic techniques such as Secure Multiparty Computation (MPC) and Homomorphic Encryption (HE). These approaches incur an overhead due to the underlying cryptographic primitives, making them significantly more compute-intensive and thus more energy-hungry than conventional inference. Optimization approaches for SNNI mainly focused on maximizing accuracy and minimizing execution time. However, amidst growing climate concerns, energy consumption becomes a crucial aspect when determining optimal deployments of SNNI. Thus, conducting a comprehensive investigation into energy-friendly SNNI approaches remains an open challenge.

We design and develop a framework for determining the optimal deployment of SNNI. Given a machine learning (ML) inference task, the framework selects the SNNI approach and NN that minimize energy consumption while considering additional constraints, such as the desired level of accuracy and execution time. This knowledge-based framework distills information from experiments involving combinations of SNNIs and NNs, subsequently identifying the best deployment option, i.e., the (near-) optimal choice of SNNI and NN, based on the client's request.

***Index Terms*—energy consumption, secure neural network inference, neural network, privacy-preserving machine learning, secure multi-party computation**

I. Introduction

In recent years, the field of machine learning (ML) has gained enormous attention. Deep neural networks (NNs) are especially widely used in various applications, such as image classification, natural language processing (NLP), pattern recognition, and predictive analytics, among others [1], [25].

ML encompasses two main phases: training and inference. The training phase involves substantial volumes of data to refine and optimize the parameter values in a NN model. The inference phase uses a pre-trained NN model to process and analyze new input data. To master the often tedious and computationally intensive training phase, Machine Learning as a Service (MLaaS) has gained significant attention [5]. In MLaaS, a service provider offers inference with a pre-trained NN to clients, allowing them to obtain the output of the NN without the need for computationally intensive training.

However, the adoption of MLaaS introduces data privacy and security concerns for both clients and service providers [19]. The input and output may contain sensitive information about the client, which must be kept private. Meanwhile, the service provider, as the proprietor of the NN models, is concerned that the parameters of the NN could be stolen in the inference phase by the client or other adversaries.

Secure Neural Network Inference (SNNI) addresses these concerns by computing the output of the NN based on client inputs, ensuring that the client learns only the output and nothing about the service provider's NN beyond what the output reveals, while the service provider remains oblivious to the client's input and output.

In recent years, numerous approaches have been proposed for SNNI [13]. To achieve security and privacy goals, SNNI relies on sophisticated cryptographic techniques such as Secure Multiparty Computation (MPC) and Homomorphic Encryption (HE). While these approaches preserve privacy and safeguard sensitive data, they often incur computational overhead due to their reliance on complex cryptographic primitives [2]. Thus, SNNI protocols are significantly more compute-intensive and energy-hungry than conventional inference methods.

Minimizing energy consumption is crucial to mitigate carbon emissions in response to the escalating climate concerns. Compared to training, ML inference consumes the majority of computing resources, resulting in high costs and a significant environmental impact, particularly in terms of carbon footprint [22], [23]. For instance, the carbon footprint of Meta's Transformer-based Universal Language Model for text translation is predominantly generated during the inference phase rather than the training phase, with inference requiring significantly more resources (65%) compared to training [23].

However, optimization approaches proposed for SNNI have primarily focused on (i) maximizing accuracy (i.e., measures for the ratio of correct inferences made by the ML model) and (ii) minimizing execution time (i.e., the time needed to perform inference) [13]. Amidst growing climate concerns,

This paper was published in the *Proceedings of the 20th IEEE International Conference on Green Computing and Communications (GreenCom)*, pp. 339-346, 2024

optimizing energy consumption when determining optimal deployments of SNNI becomes a crucial aspect due to the high computational overhead. Therefore, making SNNI more energy-friendly remains an important research challenge.

To address this challenge, we design and develop a framework that determines the optimal deployment option for SNNI in terms of energy consumption. When referring to energy consumption, we mean the overall energy consumption encompassing both client-side and server-side energy. Given an ML inference task, this framework selects the SNNI approach and NN that minimize energy consumption while considering additional constraints, such as the desired level of accuracy (i.e., the minimum acceptable accuracy for the client) and execution time (i.e., the maximum acceptable execution time for the client). The framework is based on a knowledge base, which distills information from experiments involving combinations of SNNIs and NNs. The framework uses the knowledge base to identify the (near-) optimal choice of SNNI and NN based on the client's request. This paper outlines the conceptual framework and showcases the main ideas. Specifically, the contributions of this paper are as follows:

- An overview of our framework, presenting a novel approach designed to support any SNNI protocol.
- An empirical experiment using two state-of-the-art SNNI protocols with three different NNs.
- An investigation into the implications of combining NNs and SNNIs in terms of execution time, power, and energy consumption on both the client and server sides.

The paper is organized as follows: Section II presents relevant background information, Section III provides an overview of our framework, while Section IV exemplifies it; we discuss our insights in Section V, and review related work in Section VI. Section VII concludes the paper.

II. Background

A. Secure Neural Network Inference

Fig. 1 shows a typical setup of SNNI in the context of MLaaS [13]. In this scenario, there are two parties: the client and the service provider. The service provider has a pre-trained NN on a server owned by them, with which it offers inference as a service to clients. The client owns an input, x, and seeks to acquire the neural network's output, f(x), for this input. Both input and output may contain sensitive information about the client and must be kept private. The parameters of the NN may also be sensitive and the service provider may not want to reveal them to the client. An SNNI protocol allows the client to obtain the output of the NN while ensuring that x, f(x), and the parameters of NN remain private to their respective owners. We now briefly describe two such protocols.

1) CrypTFlow2: CrypTFlow2 [18] is an SNNI approach that evaluates an NN layer by layer, applying different cryptographic primitives for different layers. In the used version of CrypTFlow2, called SCI_HE[1], linear layers (e.g., fully-connected or convolutional layers) are evaluated using HE. CrypTFlow2 introduces special MPC protocols for securely and efficiently evaluating non-linear layers, such as ReLU and Argmax. CrypTFlow2 ensures that the results of secure inference are bitwise equivalent to the corresponding fixed-point cleartext computations.

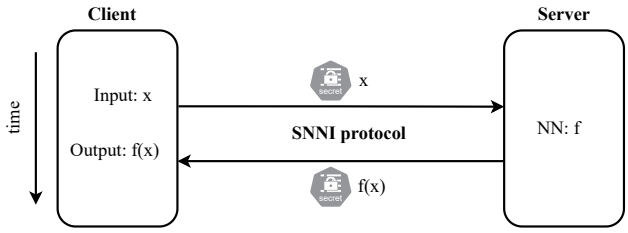

Fig. 1. Typical SNNI setup in a MLaaS context, based on prior work [13]

CrypTFlow2 can securely evaluate SqueezeNet [8] in under a minute. It can securely evaluate ImageNet-scale practical deep NNs, such as ResNet50 [4] and DenseNet121 [6], which are significantly larger than those considered in prior work. CrypTFlow2 claims to be more efficient than the previous state-of-the-art protocol, Delphi [15], particularly in evaluating non-linear layers of smaller CIFAR-scale deep NNs considered in previous studies. CrypTFlow2 achieves this efficiency by requiring an order of magnitude less communication and reducing execution time significantly by 20×-30× [18].

2) Cheetah: Cheetah [7] is a recently introduced SNNI approach. It performs secure inference on large NN models like ResNet50, with significantly reduced computation and communication overheads compared to other SNNI approaches [7].

Similarly to CrypTFlow2, Cheetah evaluates the NN layer by layer, applying HE for linear layers and special MPC primitives for nonlinear layers. However, Cheetah uses a different HE scheme from the one used in CrypTFlow2, and Cheetah also further optimizes some of the MPC protocols used for nonlinear layers compared to CrypTFlow2 [7].

B. Benchmark Neural Networks

We use three ImageNet-scale benchmark NNs: SqueezeNet [8], ResNet50 [4], and DenseNet121 [6].

1) SqueezeNet: a convolutional neural network (CNN) designed to have fewer parameters while maintaining competitive accuracy. It achieves AlexNet-level accuracy on ImageNet with a 50X reduction in model size compared to AlexNet [8]. It comprises 26 convolution layers, 26 ReLU layers, 3 Maxpool layers, and 1 Avgpool layer.

2) ResNet50: a CNN based on residual network architecture. It comprises 53 convolutional layers, each followed by batch normalization and ReLU activation layers. Additionally, the network features 1 Maxpool layer, 1 Avgpool layer, and 1 fully connected layer.

3) DenseNet121: a densely connected CNN that connects each layer to every other layer. It comprises 121 convolutional layers, each followed by batch normalization and ReLU activation layers. Additionally, it includes 1 Maxpool layer, 4 Avgpool layers, and 1 fully connected layer.

[1]SCI means Secure and Correct Inference. In this paper, we use SCI_HE and CrypTFlow2 interchangeably.

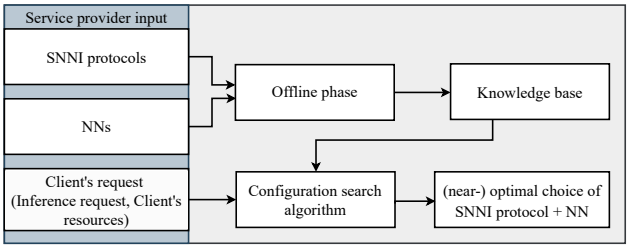

Fig. 2. Overview of the framework, left pane is the service provider input

III. Our approach

In our approach, the service provider interacts with our framework to find an optimal combination of SNNI and NN. To do that, the framework relies on a knowledge base (KB). We proceed to introduce the framework, the KB, and the interaction between client, service provider, and our framework.

A. Overview of the framework

Fig. 2 provides an overview of the framework, which is used by the service provider. In an initial offline phase, the KB is populated through experiments involving different combinations of SNNI protocols and NNs. This is followed by an online phase, wherein the populated KB is utilized by our framework and extended with results from subsequent protocol execution. The service provider offers the following deployment options for the offline phase: (i) a list of available SNNI protocols; (ii) a list of available pre-trained NNs.

The client's request typically comprises (i) an inference request, which includes the input size, inference request type (e.g., image classification), minimum acceptable accuracy, and maximum acceptable execution time; and (ii) information regarding the client's resources, including CPU, memory, and network bandwidth.

Our framework aims to determine the optimal deployment option, i.e., the optimal choice of SNNI approach and NN. In its current version, our framework focuses on a single optimization objective function: *minimizing energy consumption*, while respecting the client's constraints. For an ML inference task, the framework selects the SNNI approach and NN that minimize energy consumption while considering additional constraints, such as the desired level of accuracy and execution time. Once the optimal choice is identified, the framework forwards this solution to the service provider, who then recommends it to the client. The service provider uses our approach as a service. This self-learning service employs existing SNNI approaches and NNs, and dynamically decides what is optimal at run time. It autonomously selects the best solution without requiring human intervention.

To find the optimal deployment option, our framework uses a *configuration search algorithm*. This algorithm is driven by the client's specific request. The search space encompasses existing deployment options, each comprising a possible combination of SNNI approach and NN. The algorithm evaluates various deployment options stored in the KB to identify the best one, i.e., the (near-) optimal choice of SNNI approach and NN for the client's needs. (This paper focuses on a simple proof-of-concept, where an explicit KB is used as the search space for the configuration search algorithm. Future iterations of the configuration search algorithm may incorporate more advanced methods. However, for larger-scale knowledge bases, the configuration search algorithm may require further optimization in the future to handle increased size and complexity.) Below we describe how the algorithm works. First, we select the KB entries that are compatible with the client's request, by performing the following checks for each entry in the KB:

- Step 1: if the Inference Type of the client request does not match the Inference Type in the entry, return NO,
- Step 2: if the Inference Size of the client request does not match the Inference Size in the entry, return NO,
- Step 3: if the recorded accuracy in the entry is less than the requested accuracy, return NO,
- Step 4: if the recorded execution time in the entry is larger than the requested execution time, return NO,
- Step 5: check if the client's resources match the resources recorded in the entry.

To scope the work presented in this paper, we assume that step 5 is always fulfilled. In future work, we plan to investigate a more nuanced set of checks, such as:

- Hard NO: if the Inference Type is not a match, the deployment option in the respective KB entry is not suitable for the client request,
- Soft NO: if the client's resources do not match those recorded in the entry, then a prediction mechanism is used to determine the execution time and energy consumption for the client using the deployment option in the entry.

Once we identify the compatible KB entries, we select the one with the smallest recorded energy consumption. To break ties, we select the one with the smallest recorded execution time.

B. Knowledge base

During the offline phase, data is collected from experiments involving SNNIs and NNs to populate the KB. Subsequently, the online phase utilizes this KB to predict the optimal deployment option in various scenarios. The KB continually enhances its performance as a self-learning model through online learning. We describe the experiments in Section IV-A.

C. Interaction between client, service provider, and framework

The sequence diagram presented in Fig. 3 illustrates the execution of operations and the interaction between the client, the service provider, and our framework, outlining the order of events. For brevity, the diagram solely focuses on the online phase, omitting the offline phase wherein the KB is populated.

The process begins with the client sending a request to the service provider, who then forwards it to the framework for identifying the optimal deployment option. The framework uses its configuration search algorithm and determines the near-optimal choice for the client. Once the best combination of SNNI and NN is determined, the framework passes this to the service provider, who then recommends it to the client.

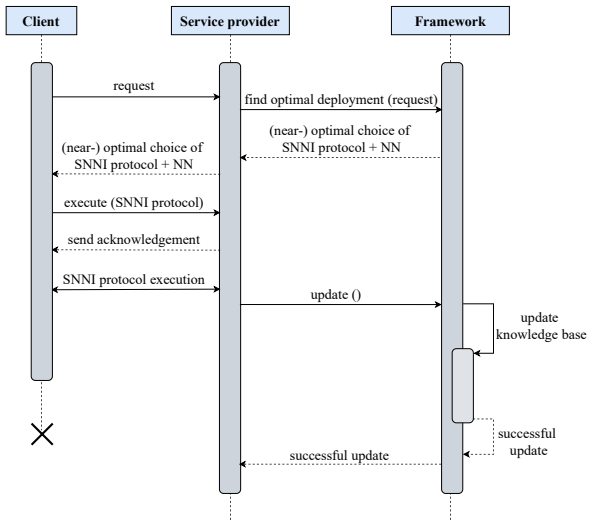

Fig. 3. Interaction between client, service provider, and framework

Following the initial handshaking process between the client and the service provider, the client sends a protocol execution request for the recommended SNNI protocol to the service provider, which is duly acknowledged. Subsequently, the agreed SNNI protocol is executed jointly by the service provider and the client, adhering to the protocol specifications.

Afterward, the service provider sends an update request to the framework to incorporate the new measurements into the KB. This update includes the new energy consumption and execution time values obtained from the executed protocol.

IV. Exemplifying Our Framework: A Proof-of-Concept Approach

We describe here the experiments and the resulting data used to populate the KB during the *offline phase*. We also explain how the framework functions in the *online phase*.

A. Exemplification of the Offline Phase

Our experimental methodology and metric definitions are built on prior work [17].

1) Experiment design: We conduct experiments using two state-of-the-art SNNI approaches: Cheetah and CrypTFlow2 (SCI_HE), and three NNs: SqueezeNet, ResNet50, and DenseNet121. We measure the execution time, average power consumption, and subsequently, the total energy consumption for one secure inference. We obtain these measurements independently for both the client and server machines. Currently, we focus on one objective: *minimize energy consumption*.

We run each combination of SNNI and NN 30 times. Based on the experimental results, we compute the overall energy consumption (encompassing both client-side and server-side energy) and execution time. We also collect data on the client's resources, including CPU, memory, and network bandwidth, as well as specifics of the client's inference request such as type, input size, and requested levels of accuracy and execution time. These experimental findings are used to populate our KB.

In our experiments, we use a simple system model consisting of one server and one client machine. To represent these, we use two identical Intel(R) Xeon(R) E-2378 computers, both running Ubuntu 22.04.2 LTS with a 64-bit Linux kernel version 5.15.0-83-generic x86_64. Each computer is equipped with a 2.60 GHz, 8-core processor and has 64 GB of RAM each. To measure power and energy consumption, we utilize the Linux Hardware Monitoring (hwmon)[2] interface on both the client and server machines. Through this interface, we collect the average power consumption of both client and server machines every second. From these measurements, we calculate the average power consumption and estimate the total energy consumption of each computer during the entire duration of the secure inference process. Regarding the SNNI approaches, we utilize the latest version of Cheetah (*commit number: 0b63d6f*) released on 2nd March, 2023. The codebase of Cheetah includes an implementation of CrypTFlow2 (SCI_HE), which we employ in our experiments as an alternative SNNI approach alongside Cheetah.

We take the following precautions in our experiments: (i) ensuring no other workload is running on both computers besides our experiment, (ii) subtracting the mean idle power consumption of the system when measuring the actual power consumption of running an SNNI program on both machines, and (iii) repeating the measurements of each SNNI and NN combination 30 times.

We collect the following metrics in our experiment:

- Execution time (s): Execution time is the duration of the secure inference process execution for one inference request. It is measured by the time difference between when the client starts and finishes its part in the secure inference process. Since the actual execution of the secure inference protocol starts when the client joins, we disregard the waiting time of the server. In the end, both the server and the client finish the protocol execution at about the same time. It is measured in seconds (s).
- Average power consumption of the server (W): The additional power consumed by the server machine during the execution of the SNNI program, averaged over the duration of the secure inference process. It is measured in Watts (W).
- Average power consumption of the client (W): The additional power consumed by the client machine during the execution of the SNNI program, averaged over the duration of the secure inference process. It is measured in Watts (W).
- Total energy consumption of the server (J): The integral of the instantaneous power consumption on the server machine accumulates over the duration of the secure inference process, calculated as the product of the average power consumption of the server and the execution time of the SNNI program. It is measured in Joules (J).
- Total energy consumption of the client (J): The integral of the instantaneous power consumption on the client machine accumulates over the duration of the secure inference process, calculated as the product of the average power

[2]https://www.kernel.org/doc/Documentation/hwmon/sysfs-interface

consumption of the client and the execution time of the SNNI program. It is measured in Joules (J).

2) Experimental results: We conduct an empirical experiment, obtaining the idle power measurements initially. Subsequently, we measure power and energy consumption while executing various combinations of SNNIs and NNs. To measure the power consumption of running an SNNI program, we first measure the idle power consumption of both the client and server machines. Then, we measure the system's power consumption while the SNNI program is running. Finally, we subtract the mean idle power consumption of the system from this value and report this difference as power consumption. We also compute the energy consumption metric based on this difference later in our experiments.

a) Idle power measurements: We measure the idle power consumption for 10 minutes and collect power measurements every second, yielding 600 measurements. Subsequently, we compute the mean and standard deviation of idle power consumption for both client and server machines from these measurements. The idle power consumption for the client is 29.86 ± 0.48 watt, and for the server, it is 30.68 ± 0.75 watt. The standard deviation is small compared to the idle power consumption, indicating it will also be small relative to the additional power consumption incurred during the experiment. This gives us a reliable basis for accurately measuring the power consumption of running an SNNI program. Given the relatively small standard deviation of idle power consumption, we disregard it in further calculations.

b) Energy measurements: We measure the average power consumption and the total energy consumption of executing one secure inference using two state-of-the-art SNNI protocols on three different NNs. The measurement of average power and energy consumption accounts for idle power consumption. Table I shows the results. For each metric, we report an average of 30 runs.

For each combination of SNNI and NN, we obtain measurements for the overall energy consumption (encompassing both client-side and server-side energy) and execution time from our experiments. We use accuracy information from the literature: "Cheetah produces predictions almost the same as CrypTFlow2 (SCI_HE), which are bit-wise equivalent to plaintext fixed-point computation" [7]. The authors of CrypTFlow2 demonstrate that the accuracy achieved by the fixed-point code is equivalent to that of floating-point numbers [18]. Therefore, we use the top-1 fixed point accuracy from the CrypTFlow2 paper (cf. Table 10) as our accuracy indicator for the respective SNNI and NN combination. Table II provides a snapshot of the populated KB at the time the client places a request.

B. Exemplification of the Online Phase

The framework uses its KB to recommend the optimal combination of SNNI and NN online. We now use an example to illustrate our process of identifying and recommending a deployment option for SNNI protocols and NNs.

Table III presents sample data regarding the client's request-related information, with each row representing a separate request. We assume that the client's inference requests come with client resources similar to those in our KB.

For the first request, a client holds an input image of size 224x224 pixels and seeks secure inference for an image classification task. The client's resources consist of a 2.60 GHz processor with 64 GB of RAM and a network bandwidth of 1000 Mbps. The client's objective is to determine the SNNI + NN combination that achieves secure inference with minimal energy consumption, ensures at least 75% accuracy, and completes the execution within 120 seconds.

The service provider forwards the client's request to the framework, which uses it as input for its configuration search algorithm to determine the optimal combination of SNNI and NN from the KB that best satisfies the client's request.

The search algorithm traverses the KB and evaluates deployment options stored therein (as shown in Table II). Table IV summarizes the evaluation results. The first option is Cheetah + SqueezeNet. Despite its low execution time and energy consumption, this option is discarded because the accuracy of Cheetah + SqueezeNet (55.90%) falls short of the client's minimum threshold (75%) and due to a mismatched input size. Similarly, SCI_HE + SqueezeNet is discarded due to accuracy and input size mismatches. Cheetah + ResNet50 satisfies all the client's requirements and thus is considered as a possible recommended option. The remaining options are discarded due to various shortcomings: SCI_HE + ResNet50 exceeds the maximum acceptable execution time, Cheetah + DenseNet121 fails to meet the minimum accuracy requirement, and SCI_HE + DenseNet121 fails on both criteria. Ultimately, the configuration search algorithm selects Cheetah + ResNet50 as the only feasible deployment choice, resulting in an energy consumption of 7749.71 J, an accuracy level of 76.45%, and an execution time of 108.70 seconds.

The second client request is more flexible in terms of accuracy. They seek a combination of SNNI and NN that achieves inference with minimal energy consumption while ensuring an accuracy of at least 70% and completing the execution within 120 seconds.

After analyzing the options in the KB (see Table V), the configuration search algorithm eliminates Cheetah + SqueezeNet and SCI_HE + SqueezeNet due to the same reasons identified in the prior scenario. Likewise, SCI_HE + ResNet50, and SCI_HE + DenseNet121 are discarded for exceeding the execution time limit. Both Cheetah + ResNet50 and Cheetah + DenseNet121 meet all the client's requirements in this instance. Comparing their overall energy consumption, the search algorithm selects Cheetah + DenseNet121 (6750.58 J) over Cheetah + ResNet50 (7749.71 J), prioritizing lower energy consumption (6750.58 J) with an accuracy of 74.35% and an execution time of 115.27 seconds.

Our framework selects the best deployment option for the client and suggests it to the service provider, who then recommends the client proceed with this choice for execution. Upon successful protocol execution, new measurements obtained in the online phase replace old values stored in the KB entry.

TABLE I
COMPARISON OF CHEETAH AND CRYPTFLOW2 (SCI_HE) ON SQUEEZENET, RESNET50, AND DENSENET121 REGARDING EXECUTION TIME, POWER CONSUMPTION, AND ENERGY CONSUMPTION. THE RESULTS ARE THE AVERAGE OF 30 RUNS, CORRECTED FOR AVERAGE IDLE POWER CONSUMPTION

Neural network	SNNI approach	Participant	Execution time (s)	Average Power consumption (W)	Total Energy consumption (J)
SqueezeNet	Cheetah	Client Server	24.50 ± 0.51	28.44 ± 2.92 31.63 ± 2.99	696.79 ± 73.23 775.36 ± 77.91
	SCI_HE	Client Server	77.53 ± 0.86	9.36 ± 0.82 29.58 ± 2.58	725.94 ± 66.61 2293.61 ± 200.62
ResNet50	Cheetah	Client Server	108.70 ± 0.47	24.22 ± 1.69 47.07 ± 3.14	2633.05 ± 182.51 5116.66 ± 347.83
	SCI_HE	Client Server	366.90 ± 4.09	8.66 ± 0.65 47.00 ± 1.35	3178.43 ± 235.43 17242.15 ± 512.64
DenseNet121	Cheetah	Client Server	115.27 ± 0.52	24.99 ± 1.17 33.58 ± 1.55	2,880.30 ± 132.23 3,870.28 ± 181.26
	SCI_HE	Client Server	409.73 ± 3.75	8.79 ± 0.53 37.70 ± 1.24	3,600.76 ± 216.57 15,443.01 ± 460.93

TABLE II
SAMPLE KNOWLEDGE BASE WITH DATA COLLECTED FROM ACTUAL EXPERIMENTS. SNET = SQUEEZENET, RNET = RESNET50, DNET = DENSENET121, IM = IMAGE CLASSIFICATION

Service provider's deployment options			Client's inference requests		Client's resources			Measured outcome	
SNNI	NN	Accuracy (%)	Type	Input size	CPU (GHz)	Memory (GB)	Network bandwidth (Mbps)	Execution time (s)	Overall Energy consumption (J)
Cheetah	SNet	55.90	IM	227x227	2.60	64	1000	24.50	1472.15
SCI_HE	SNet	55.90	IM	227x227	2.60	64	1000	77.53	3019.55
Cheetah	RNet	76.45	IM	224x224	2.60	64	1000	108.70	7749.71
SCI_HE	RNet	76.45	IM	224x224	2.60	64	1000	366.90	20420.58
Cheetah	DNet	74.35	IM	224x224	2.60	64	1000	115.27	6750.58
SCI_HE	DNet	74.35	IM	224x224	2.60	64	1000	409.73	19043.77

TABLE III
EXAMPLE CLIENT REQUESTS

Client's inference requests				Client's resources		
Input size	Type	Min. accuracy (%)	Max. execution time (s)	CPU (GHz)	Memory (GB)	Network bandwidth (Mbps)
224x224	IM	75	120	2.60	64	1000
224x224	IM	70	120	2.60	64	1000
…	…	…	…	…	…	…

TABLE IV
EVALUATION RESULTS OF POSSIBLE DEPLOYMENT OPTIONS FOR THE FIRST CLIENT REQUEST

SNNI	NN	Appropriate for request
Cheetah	SNet	NO: wrong input size, accuracy too low
SCI_HE	SNet	NO: wrong input size, accuracy too low
Cheetah	RNet	YES
SCI_HE	RNet	NO: execution time too high
Cheetah	DNet	NO: accuracy too low
SCI_HE	DNet	NO: accuracy too low, execution time too high

TABLE V
EVALUATION RESULTS OF POSSIBLE DEPLOYMENT OPTIONS FOR THE SECOND CLIENT REQUEST

SNNI	NN	Appropriate for request
Cheetah	SNet	NO: wrong input size, accuracy too low
SCI_HE	SNet	NO: wrong input size, accuracy too low
Cheetah	RNet	YES
SCI_HE	RNet	NO: execution time too high
Cheetah	DNet	YES
SCI_HE	DNet	NO: execution time too high

V. DISCUSSION

We discuss the insights from this study, the limitations of our research, and potential threats to validity.

A. Lessons learned

Our experiments in Section IV-A demonstrate that different combinations of NN and SNNI approaches can lead to very different execution times and energy consumption values. Our measurement results are in line with previous research [17], but we are the first to go one step further and leverage these measurement results for determining the best combination of NN and SNNI approach for a given client request.

For a future SNNI service that is used by many clients, it is fair to assume that different clients can have different requirements in terms of accuracy and execution time. As Section IV-B shows, our framework can accommodate these different requirements, by proposing different NNs and/or SNNI approaches depending on the requirements. This dynamic adaptivity is a special feature of our framework that differentiates it clearly from previous approaches.

Section IV-B has also shown that for certain client requests, multiple NN / SNNI approach combinations can be suitable. Our framework automatically leverages such cases for saving energy, by choosing from the suitable deployment options the one with the least energy consumption. Again, this is a property that no existing approach exhibits.

Section IV has also demonstrated how the offline and online

phases are tied together by the KB. The KB captures in a concise form all the relevant results of all experiments of the offline phase. During the online phase, the KB provides a solid basis for responding to client queries.

B. Opportunities for future research

We implemented a first version of our framework as a proof of concept (PoC) to show the fundamental viability of our idea. We plan to extend our framework in several aspects.

We currently assume that the client provides hard requirements on accuracy and execution time, and we optimize energy consumption. In reality, several other metrics could also be relevant (e.g., energy consumption of only the client, power draw, memory demand), and there may not be hard requirements on most of the metrics. Multi-objective optimization methods [14] can be applied to find good deployment options in such a case.

For now we used the same client computer in the offline and the online phase. In reality, clients in the online phase can differ from what was tested in the offline phase. This requires predicting how long inference would take and how much energy it would consume on the new client, given measurements with other clients [24]. We plan to apply prediction methods, such as machine learning, to solve this problem.

We currently employ three conventional NNs in the experiment: SqueezeNet, ResNet50, and DenseNet121. Future work aims to enrich the framework by incorporating additional state-of-the-art NNs, such as transformers.

Our PoC only considers a single client and a single server at a time. In reality, a service provider may possess several servers with different capabilities and may serve multiple clients simultaneously. Our framework could be extended with resource management aspects, allowing it to select not only the best NN and SNNI approach but also the most appropriate server for a given client request, considering the load on the servers and their energy efficiency.

C. Threats to validity

External validity. The PoC incorporates two SNNI approaches and three NNs. To address external validity concerns, we select two state-of-the-art SNNI approaches. We also utilize three often-used neural networks that were previously employed to evaluate the chosen SNNI approaches. We use two robust identical computers to simulate the server and client machines, both equipped with relatively powerful hardware configurations. Real clients may not always possess such powerful computing devices. Therefore, as part of our future work, we aim to accommodate varied hardware configurations to represent less powerful client computers. However, running the SNNIs on different computers and employing diverse SNNI approaches alongside other NNs may result in varying energy measurements. Further replication of the experiment can help mitigate this potential threat.

Internal validity. System processes may contribute to additional power consumption, potentially impacting the energy usage of the SNNI approaches. To address this concern, we conduct idle power consumption measurements for 10 minutes on both client and server machines. We observe a relatively small standard deviation in the idle power consumption of both machines over extended periods, indicating that any additional power consumption of the system during the experiment can be attributed to the SNNI process. We also ensure that no other workloads run on either computer during our experiments to maintain experimental integrity. To mitigate the potential impact of random effects, we repeat the measurements of each SNNI and NN combination 30 times, and calculate the average of the collected metrics. Finally, we measure power and estimate energy consumption using the Linux Hardware Monitoring (hwmon) interface. We partially mitigate potential bias by subtracting the mean idle power consumption of the system when measuring the actual power consumption of an SNNI process. However, future replications of experiments using alternative power monitoring tools may yield more precise and reliable measurements, further mitigating this potential threat to internal validity.

VI. Related work

In recent years, the field of SNNI has witnessed significant progress, with numerous approaches proposed, many of which leverage advanced cryptographic techniques [13]. Current research has mostly focused on making secure inference faster.

Among the earliest protocols to explore secure two-party NN inference, CryptoNets [3] demonstrated the application over encrypted data, using HE. At the cost of high overhead in terms of execution time, CryptoNets achieved both high accuracy and security, with high throughput. GAZELLE [9] combined multiple protocols, incorporating a blend of HE and two-party computation techniques such as garbled circuits.

Some approaches [15], [16], [11], [21] use offline pre-computation to expedite the online phase. For instance, in Delphi, heavy computational operations are shifted to the offline pre-processing phase, thus significantly reducing execution time and communication costs in the online phase compared to prior work [15]. SecureML uses online/offline phases similar to Delphi, but assumes the presence of two non-colluding servers and leverages parallelization in its offline phase, leading to increased efficiency [16]. Falcon, on the other hand, differs from SecureML by using three non-colluding servers instead of two, leading to improved efficiency compared to prior works [21]. Also, MiniONN uses an offline pre-computation phase to reduce the overhead during the online phase, achieving comparable online performance to prior works and significantly improved offline performance [11].

CrypTFlow2 [18] enhances CrypTFlow [10] by combining HE and various MPC protocols, leading to reduced execution time. Cheetah [7] refines CrypTFlow2, yielding one of the most efficient SNNI implementations to date.

Some approaches leverage the edge computing paradigm, where most compute-intensive tasks are offloaded to edge devices. Tian et al. [20] devised an IoT-based edge-assisted scheme for efficiently executing complex CNN inference tasks, resulting in significant energy savings for IoT devices. However, it requires powerful devices to handle high-frequency

CNN requests, due to storage overhead and substantial energy consumption during offline key pre-computation [20]. Liu et al. [12] introduced Leia, a lightweight cryptographic NN inference system at the edge, delegating the entire secure inference to the edge. Additionally, the authors measured the energy consumption of Leia across various medical application datasets [12]. By outsourcing compute-intensive tasks to the edge, these approaches focus on reducing computational cost and energy consumption of IoT devices, and not on reducing the overall energy consumption of the entire system, although this would be more important to improve sustainability.

To sum up, the primary focus of existing SNNI research lies in making execution faster, without explicitly optimizing overall energy consumption. In our work, we try to fill this gap in existing research by providing a framework for minimizing the overall energy consumption of SNNI.

VII. Conclusion

MLaaS allows clients to obtain inference output from pre-trained neural networks offered by service providers. The data privacy and security concerns introduced by MLaaS are addressed by SNNI approaches. SNNI leverages cryptographic techniques to shield the client's input and output from the service provider, and the service provider's NN from the client.

The high computational overhead associated with state-of-the-art SNNI approaches makes finding energy-friendly solutions an ongoing challenge. To tackle this challenge, we proposed a novel framework designed to determine an optimal deployment option for the SNNI and NN based on client requests. This initial version focuses solely on optimizing the overall energy consumption, while accuracy and execution time metrics are treated as client constraints. By leveraging a knowledge base derived from experimental data, we implemented the first version of our framework as a proof-of-concept, demonstrating the feasibility of providing tailored deployment options for clients.

We plan to extend our framework by incorporating prediction methods and multi-objective optimization to enhance its practicality. Additionally, we aim to accommodate resource-constrained clients and support multiple servers serving multiple clients within the framework. Furthermore, we intend to apply our framework and evaluate its overall performance in real-world scenarios to validate and refine its practical utility.

Acknowledgments

This research was supported by GreenDIGIT, an European Union project funded under the grant agreement 101131207.

References

[1] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J. Santamaría, J. Santamaría, Mohammed A. Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, 8(53), 2021.

[2] Rik de Vries and Zoltán Ádám Mann. Secure neural network inference as a service with resource-constrained clients. In *IEEE/ACM 16th International Conference on Utility and Cloud Computing (UCC)*, 2023.

[3] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 201–210, 2016.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[5] Jiahui Hou, Huiqi Liu, Yunxin Liu, Yu Wang, Peng-Jun Wan, and Xiang-Yang Li. 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, 2022.

[6] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[7] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure two-party deep neural network inference. In *31st USENIX Security Symposium*, pages 809–826, 2022.

[8] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. arXiv preprint arXiv:1602.07360, 2016.

[9] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium*, pages 1651–1669, 2018.

[10] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CrypTFlow: Secure TensorFlow inference. In *IEEE Symposium on Security and Privacy (SP)*, pages 336–353, 2020.

[11] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via MiniONN transformations. In *ACM SIGSAC Conference on Computer and Communications Security*, page 619–631, 2017.

[12] Xiaoning Liu, Bang Wu, Xingliang Yuan, and Xun Yi. Leia: A lightweight cryptographic neural network inference system at the edge. *IEEE Transactions on Information Forensics and Security*, 17:237–252, 2022.

[13] Zoltán Ádám Mann, Christian Weinert, Daphnee Chabal, and Joppe W. Bos. Towards practical secure neural network inference: The journey so far and the road ahead. *ACM Comput. Surv.*, 56(5), nov 2023.

[14] R Timothy Marler and Jasbir S Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26:369–395, 2004.

[15] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference system for neural networks. In *Workshop on Privacy-Preserving Machine Learning in Practice*, PPMLP'20, page 27–30, 2020.

[16] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *IEEE Symposium on Security and Privacy (SP)*, pages 19–38, 2017.

[17] Jorit Prins and Zoltán Ádám Mann. *Secure Neural Network Inference for Edge Intelligence: Implications of Bandwidth and Energy Constraints*, pages 265–288. Springer Nature Switzerland, Cham, 2024.

[18] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CrypTFlow2: Practical 2-party secure inference. In *ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 325–342, 2020.

[19] Harry Chandra Tanuwidjaja, Rakyong Choi, Seunggeun Baek, and Kwangjo Kim. Privacy-preserving deep learning on machine learning as a service—a comprehensive survey. *IEEE Access*, 8:167425–167447, 2020.

[20] Yifan Tian, Laurent Njilla, Jiawei Yuan, and Shucheng Yu. Low-latency privacy-preserving outsourcing of deep neural network inference. *IEEE Internet of Things Journal*, 8(5):3300–3309, 2021.

[21] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. Falcon: Honest-majority maliciously secure framework for private deep learning. *Proceedings on Privacy Enhancing Technologies*, 2021(1):188–208, 2021.

[22] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 945–960, 2022.

[23] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, et al. Sustainable AI: Environmental implications, challenges and opportunities. In D. Mar-

culescu, Y. Chi, and C. Wu, editors, *Proceedings of Machine Learning and Systems*, volume 4, pages 795–813, 2022.

[24] Eloise Zhang and Zoltán Ádám Mann. Predicting the execution time of secure neural network inference. In *39th International Conference on ICT Systems Security and Privacy Protection (IFIP SEC)*, 2024.

[25] Qiao Zhang, Chunsheng Xin, and Hongyi Wu. Privacy-preserving deep learning based on multiparty secure computation: A survey. *IEEE Internet of Things Journal*, 8(13):10412–10429, 2021.