BACHELOR INFORMATICA

UNIVERSITY OF AMSTERDAM

INFORMATICA — UNIVERSITEIT VAN AMSTERDAM

Title

Jorit Prins

December 15, 2022

Supervisor(s): Zoltan Mann

Signed: Signees

Abstract

Abstract

Contents

CHAPTER 1

Introduction

1.1 Relevance

The recent rise in Big Data increased the data exchange on the internet. With more and more computer resources available, researchers quickly started to utilise the possibilities of machine learning (ML) to analyse the data. Techniques like neural networks (NN) are promising ways to scientific breakthroughs. Machine learning has a wide variety of applications for classification such as traffic analysis, image recognition, intrusion detection, spam detection, medical or genomics predictions, financial predictions and face recognition (Dowlin et al. 2017; Gilad-Bachrach et al. 2016; He et al. 2015; Islam et al. 2011)

The use of ML typically consists of two phases: training and inference. In the first phase a NN is trained by feeding an extensive dataset to find the best parameters. NNs used for machine learning have to be maintained, evaluated and the training phase is often a tedious and time exhausting process. In the inference phase an input is applied to the trained NN. Because of the time consuming process of creating a NN, machine learning as a service (MLaaS) became popular (Ribeiro, Grolinger, and Capretz 2015). In MLaaS, a company offers a pre-trained NN to the clients. Now, clients only need to worry about the inference phase.

A typical MLaaS situation (figure 1.1) consist of two parties: the client holding an input x and a company holding neural network *f*. This research focuses on the inference part: the client wants to know the neural network (held by the server) applied to input x (held by the client): *f(x)*.

However, MLaaS offers great threats to privacy. For the server to train the model as accurately as possible a NN needs access to a large amount of precise data, which may consist of sensitive information. Data providers may be reluctant to provide the server with this data because of the sensitive properties of the data. Besides, other features, irrelevant to the prediction task, could also be derived from this data (Nasr, Shokri, and Houmansadr 2019). Moreover, input from the client can also be confidential and the client can therefore be reluctant to send x to the server.

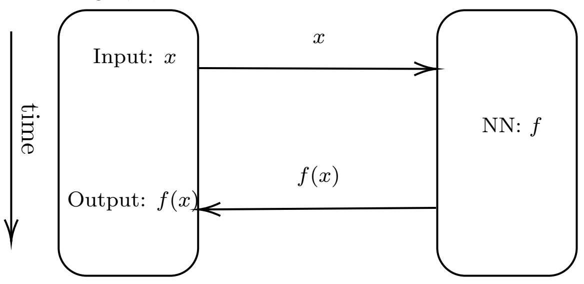

Figure 1.1: Inference process in Machine Learning as a Service (MLaaS). The NN is already trained by the server and the client wants to know the output of the NN: $f(x)$

Benchmark	System	E2E Time	Commu.
MiniONN	DELPHI	$\approx$ 110s	$\approx$ 3.6GB
	Cheetah	3.55s	0.03GB
		$\approx$ 30x	$\approx$ 117x
ResNet32	DELPHI	$\approx$ 200s	$\approx$ 6.5GB
	Cheetah	15.95s	0.11GB
		$\approx$ 12x	$\approx$ 58x

1GB = 2^{30} bytes; E2E Time = End2End Time

Benchmark	System	End2End Time		Commu.
		LAN	WAN	
SqNet	SCI_{HE}	41.1s	147.2s	5.9GB
	Cheetah	16.0s	29.1s	0.5GB
		$\approx$ 2.6x	$\approx$ 5.1x	$\approx$ 11.8
RN50	SCI_{HE}	295.7s	759.1s	29.2GB
	Cheetah	80.3s	134.7s	2.3GB
		$\approx$ 3.7x	$\approx$ 5.6x	$\approx$ 12.7x
DNet	SCI_{HE}	296.2s	929.0s	35.4GB
	Cheetah	79.3s	177.7s	2.4GB
		$\approx$ 3.7x	$\approx$ 5.2x	$\approx$ 14.8x

SqNet=SqueezeNet; RN50=ResNet50; DNet=DenseNet121

Table 1.1: Table 7 (left) and 8 (right) from (Huang et al. 2022, p. 821): running time and communication costs of Cheetah compared to Delphi and CryptFlow2 (SCI_{HE})

On the other hand, owners of a NN could be worried that an adversary could steal (parameters of) their (often costly) NN. Furthermore, the result ($f(x)$) of the NN could also be confidential resulting in the need to retain this information from unauthorized parties. The secure neural network inference (SNNI) problem entails calculating the applied input *f(x)* while still holding all the above security requirements.

No general implementation of an SNNI has been widely accepted. Rapid progress in this area has made it hard to get a good overview of technological advances. Mann et al. (2022) has summarized several proposed approaches for SNNI. However, these approaches are often proof-of-concept and are not thoroughly tested. Moreover, the performance is often only tested on basic measures like efficiency or accuracy. Other metrics like energy consumption, that could be of relevance, are not researched. This could be of importance because of limitations on the client side. For example when a device is battery powered. One other example is in the case of IoT devices that have limited power resources following that the overall energy consumption should be low. Companies, on the other hand, also want to keep energy consumption as low as possible because of budget limitations and thus should not encounter big energy overhead. Another reason to limit the energy consumption is the desire to reduce carbon emission in the fight against climate change.

1.2 Research question(s)

To contribute to the prior research in this area, I will discuss the energy implications of a suggested, open source implementation of an approach to SNNI. A few of these implementations are ABY2.0 (Patra et al. 2020), Chameleon (Riazi et al. 2018), Cheetah (Huang et al. 2022), CrypTFlow2 (Kumar et al. 2019) and Delphi (Mishra et al. 2020).

For this research I will test the approach called Cheetah (Huang et al. 2022). This has several reasons. Firstly it is one of the most recent papers in this area (research also included in the 2022 Proceedings of the 31st USENIX Security Symposium). Second, they provided a working proof-of-concept on Github[1] and the last commit on GitHub was in July 2022. Third, the proof-of-concept implementation has been tested on a WAN and LAN network and on two different devices. This is good, because it represents the most realistic MLaaS scenario (Ribeiro, Grolinger, and Capretz 2015) and it is not necessary anymore to adapt the code. Fourth, it claims to achieve better speedup than the other recent approaches (e.g. Delphi and CryptFlow2, see Table 1.1). Last, one author (Dong et al. 2022) has already used aforementioned approaches as building blocks and also saw significant performance improvements when using Cheetah over others.

The main research question is of this project is:

> *RQ: What are the energy implications of Cheetah compared to its counterpart CryptFlow2?*

[1]Their implementation is available from https://github.com/ Alibaba-Gemini-Lab/OpenCheetah. I will use the commit 09fe195 for this research.

To help research the implications of the SNNI, I have defined a subset of research questions:

> *RQa: How do we best measure the energy consumption of a SNNI?*

Once I have established a way to measure energy consumption of SNNIs, I can start with the experiments of measuring the energy consumption. I will be calculating the overhead of Cheetah. With these results, the difference between the overhead on the client side and the overhead on the server side can be calculated. If there is a difference I shortly want to look at the implications of this difference. This implications could for example have impact on the aforementioned IoT devices or carbon emission, thus resulting in research question *RQb*:

> *RQb: If there is a difference between energy overhead on the client side and on the server side: what are the implications of the overhead on the client side and what are implications the overhead the server side?*

1.3 Method

To answer these aforementioned questions first I will have to select one implementation of a SNNI to start with. Once the implementation is chosen I will need to get it working on my own setup. My setup will probably be a desktop and a laptop (on running server side and the other the client side). Simultaneous to this I will answer question *RQa* with a literature search on how other authors measure energy consumption. There is a possibility that other authors have already researched the energy consumption of an approach to SNNI mentioned in the preceding section, or researched the energy implications of other programs and I can use their methods if proven successful.

Once the implementation is set up and *RQa* has been answered, I can start testing the energy consumption of a NN with and without the chosen SNNI. With the results the overhead can be calculated. If there is time to set up another implementation I will compare the overhead between these implementations. With the results of this experiment I can answer the first part of *RQb*. The second part of *RQb* can be answered with a small literature search on the implications of energy consumption. w

CHAPTER 2

Theoretical background

2.1 Neural Networks

The goal of ML is to imitate the human brain to let a computer 'learn' a task without programming task-specific rules. One way to achieve ML is through a NN. A NN is a network that is given an input and evaluates this input to calculate an output without general knowledge of the input. The input of a NN is typically a vector containing numerical data, and the output is a vector that gives insight on the input data. The input vector can represent different types of data. How the output vector should be interpreted is dependent of the task of the NN. For classification tasks, the output vector contains the likelihood of the input being a class. For example in the MNIST benchmark (Lecun et al. 1998), the dataset contains a matrix containing 28×28 grayscale pictures of the handwritten digits $0, 1, ...9$. The input vector for the NN is then has length $28 \cdot 28$ and the output vector is of length 10: for each digit the likelihood of the picture being that digit. The classification of the digit on the picture is done by choosing the digit that correlates to the highest likelihood in the output vector.

A NN consists of one or several neurons. A neuron typically consist of a list of weights $w_0, ..., w_n \in \mathbb{R}$ and a bias $b \in \mathbb{R}$ and a non-linear activation function $f : \mathbb{R} \to \mathbb{R}$. The neuron receives the input vector $x_0, ..., x_n \in \mathbb{R}$ and computes $y = x_0 w_0, ...x_n w_n + b$. The output of the neuron is the activation function applied on y. Another representation of a neuron is the dot product between the input and the weight vector with b being one of the elements in the weight vector and constant value 1 being the corresponding element in the input vector. The activation function f often is the sigmoid function $\sigma(y) = 1/(1 + e^{-y})$, the tanh fucntion or the ReLU function $\text{ReLU}(y) = \max(0, y)$.

NNs have sequences of layers containing one or more neurons. The first layer contains the input of the NN and the last layer is the output. Layers in between are one or more 'hidden layers'. The neurons in a layer are conneected to neurons on other layers: the output of a neuron can be connected to the input of other neurons in other layers. The network created is a directed weighted graph. Each link in the graph has a weight, it determines the strength of a neurons connection to another. The neurons can be connected in different ways. In a fully connected NN all nodes in a layer are connected to all the neurons in the next layers. When the output of the neurons in layer i are connected to layer $i + 1$ the network is called a feed-forward NN. Networks that also allow connections to previous layers are called recurrent networks. Some of the typically used layers are fully-connected (FC) layers, convolutional (Conv) layers[1], activation layers and pooling layers. FC and Conv layers apply a linear function on their input and hence are called linear layers. Most other layers are non-linear.

To create a NN one must first determine its architecture: what number and types of layers should it contain, in what order should the layers be and what are the sizes of the layers. Some layers change the size of their input and can thus create a degree of freedom.

[1]Convolutional layers play an important role in convolutional neural networks (CNNs) that are widely used in networks used for image processing tasks

Next, the network has to be trained. During the training the parameters of the layers (for example the weights in the FC layer) are iteratively tuned, typically by calculating the difference between the calculated output (that often is a prediction) and the target output. This is done by applying data on the NN where the output is already known (e.g. manually labeled).

After training the network it is useed for the inference phase where new input is applied to the NN. The accuracy of the NN is tested on new data where the output is also known, typically a dedicated subset of the training data. The accuracy is then defined as the ratio between the expected output of the NN and mislabeled output.

2.2 Secure Neural Networks Inference

The hidden layers contain all the information of the NN. After the tedious process of training the model one often wants to keep the parameters of the hidden layer secret. Besides, the model can reveal information about the training data (Qayyum et al. 2020) which in turn can also contain privacy sensitive information. This and the aforementioned reasons result in the SNNI problem in the case of MLaaS. Solving this problem is quite challenging. Some approaches have been suggested with cryptographic techniques like homomorphic encryption (Rivest, Adleman, and Dertouzos 1978) and secure multi party encryption (A. C. Yao 1982; A. C.-C. Yao 1986). Other hardware based approaches have also been suggested (Rouhani, Riazi, and Koushanfar 2017). All approaches achieve different trade-offs in terms of accuracy, security, efficiency and applicability.

Several authors have tried to make an overview on privacy preserving ML. Tanuwidjaja et. al. (Tanuwidjaja2020) have done a comprehensive survey on MLaaS in general and have given a chronological overview of the works. Mann et. al. (Mann22) have summarized the work on the inference part of ML. The approach that I will test has been chosen from this paper, since it is the most recent overview on the field of SNNI. It also provides substantial information about the approaches, and I could therefore make an informed choice on the approach to test.

2.3 Cheetah

Only draft text or notes from me from this point
Is hybrid model i.e. it uses different primitives for linear function and other for non-linear functions. cheetah achieves performmance by codesing of dnn, lattice-based homomorphic encryption, bolivious transfer and secret-sharing

Here i will give a short overview on the techniques used in cheetah. For a more in depth explanation and design choices i refer the reader to the original paper.

The first question is waht domain should be used for additive sharing to switch back-and-forth between different types of cryptographic primitives. OT-based protocols based on the ring Z_{2^l} perform better than protocols based on Z_p. State-of-the-art HE-based protocols force to export to Z_p because these protocols heavily utilize the homomorphic Single-Instruction-Multiple-Data (SIMD) technique to amortize the cost. One can use techniques to accept Z_{2^l} at the cost of increasing overhead and ruining the gains of the non-linear part. Authors tried to get the best of two worlds: amortized homomorphic operations while keeping the efficient non-linear protocols without extra overhead

They did this by improving CrypTFlow2 using smart techniques.

2.3.1 Fast and SIMD free linear protocols

Due to spatial property of the convolution and matrix-vector multiplicaation it is inevitable for the prior he-based and simd protocols to rotate the operands many times. Because it is expensive(?). THus the authors of Cheetah designed their linear layers (conv, bn, fc) via polynomial arithmetic circuits which maps the values of the input to the proper coefficients of the output polynomial(s).

> *"By careful design of the coefficient mappings, we not only eliminate the expensive rotations but are also are able to accept secret shares from Z_{2^l} for free"* (Huang et al. 2022, p. 810)

This also helps to reduce the cost of other homomorphic operations (e.g. encryption and decryption). It can also use large lattice dimension and also smaller.

2.3.2 Leaner protocols for the non linear functions

with advent of silent ot extension many communication ot extensions are proposed. But it does not always achieve best performance, for example with the millionaire protocol.

Improvements to trunctation protocol which is rquired after each multiplication so that the fixed point values will not overflow. First by not touching one of the two probability errors because it barely harms the inference results. Secondly because sometimes the Most significant bit is somtimes already known.

Polynomial multiplication can be viewed as a batch of inner products if we arrange the coeficients properly. They introduce natural mappings $\pi_{\pi}^{i}{}^{w}$ to properly place the values in the input and weight to polynomial coefficients for each of the $\in CONV, BN, FC$ [[[[[[[[[[dit nog ff checken of het wel e symbool is]]]]]]]].

2.3.3 lattice based homomorphic encryption

HE that is based on learning with errors (LWE) and its ring variant (ring-LWE).

CHAPTER 3

My work

3.1 testing code

CHAPTER 4

Experiments

4.1 RQa, how do we measure the energy consumption

Also notes[Solution?] Scaphandre https://github.com/hubblo-org/scaphandre/ measure power consumptoin of tech services and get data in a convenient form examples can be found here: https://metrics.hubblo.org/d/GOHnbBO7z/scaphandre?orgId=1refresh=15m

[Solution?] pcm-monitor utility, dont know how to get it working ==

[Hardware] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=9663203 Leia: A Lightweight Cryptographic Neural Network Inference System at the Edge 10.1109/TIFS.2021.3138611

We implement Leia and deploy it on Raspberry Pi Usw COOWOO popwer meter COOWOO USB Digital Power Meter Tester. [Online]. Available: http://www.coowootech.com/tools.html [[HARDWARE]]

[Hardware] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=9120239 The voltage and current are measured using a Powerjive USB multimeter and the power is calculated by the multiplication of voltage and current

---

[Java] Chiyoung Seo, Sam Malek, and Nenad Medvidovic. "Component-level energy consumption estimation for distributed java-based software systems". In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 5282 LNCS (2008). ISSN: 16113349. DOI: 10.1007/978-3-540-87891-9_7.

////////// GOOGLE ///////////////// [No open source implementation found] https://ceur-ws.org/Vol-1938/paper-san.pdf introduce crapl a c rapl tool, no open soure implementation found

[Solution?] https://github.com/hubblo-org/scaphandre/ measure power consumptoin of tech services and get data in a convenient form examples can be found here: https://metrics.hubblo.org/d/GOHnbBO7z/scapha

[Theoretical calculation based on data movement, not actual overhead of program] https://arxiv.org/pdf/1611.05128.pc With the idea of exploiting data reuse in a multi-level memory hierarchy, Chen et al. [21] have presented a framework that can estimate the energy consumption of a CNN for inference [21] Y. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in ISCA, 2016. [22] Y. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in ISSCC, 2016.

[Hardware] https://dl.acm.org/doi/pdf/10.1145/3310273.3323070

////////// Survey ///////////////// https://dl.acm.org/doi/pdf/10.1145/3141842.3141846 Survey of Approaches for Assessing Software Energy Consumption

[Hardware] Silicon Labs [19] provide an Eclipse-based IDE and development boards for ARM Cortex microcontrollers that allow measuring and profiling the energy consumption of embedded software at runtime. Energy consumption measurements for each method call are provided in a table. This system is commercially available and uses a low-cost development board with fast current and voltage sensors. The processor's program counter is read periodically and its value is sent to the host computer along with voltage and current measurements. The program running on the microcontroller is statically linked, uploaded by the host computer and resides at a previously known and constant address in memory. Hence, the address of the program counter is sufficient to infer the currently executed method.

[Android] Green Droid [3] instruments Android programs and uses PowerTutor [24] to measure their energy consumption at runtime. A trace of the instrumented program and the measured energy consumption are then used to create visualizations in the form of diagrams. These visualizations can help programmers to identify parts of their applications with problematic energy behaviour. To measure energy consumption, PowerTutor relies on a component-based model of the device's energy usage that can be automatically generated by running a calibration program on the device under test

[Java] ELens [10] annotates Java source code for Android with energy consumption estimates. An IDE visualizes this by coloring code according to its estimated energy consumption. The estimates are derived from an instruction-level energy consumption model with path dependencies that has been created by the authors for some specific device. Evaluating this approach with actual energy measurements gives an accuracy error of about 10% for methods that run longer than 10ms. Methods with a runtime of less than 10ms could not be measured by the setup used by the authors (https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=6606555 eLens for android applications)

Hardly any implementations found, a lot of approaches. While literature and too lsearch was not performed in a very systematic way However, we can report that there is a peak of literature on the energy consumption in Android systems.

//////////// Thesis ////////////////// https://ul.qucosa.de/api/qucosaThe software scope is chosen because it does not require extra hardware, e. g. an ampere meter, to take measurements

Measurements can be taken in Joule or Watt. These units can be used interchangeably when the time frame is known. It is generally advised to use Joule for small and timelimited measurements like the energy usage of a function. Watt is better suited for longer or indefinitely running program

Power = energy/time

Instant power measurements require extra hardware Time measurement is based on the drainage of the energy stored in a battery. Laptops are ideally suited for this technique because they are equiped with integrated batery. Creating model can remove biggest disadvantage: namely the dependency on physical instrumentation. Isolate energy consumption of single application.

[Java and linux only] https://github.com/kliu20/jRAPL Can only measure energy consumption on linux

[Python and not program specific] https://github.com/powerapi-ng/pyRAPL calculates global energy consumption of all the processes running on the machine.

[Linux only] https://github.com/kentcz/rapl-tools ./AppPowerMeter sleep 5 will measure the total energy and average power of the CPU while it runs "sleep 5". The CPU power is sampled every 100ms then written to rapl.csv.

[not program specific] https://github.com/sosy-lab/cpu-energy-meter

[Only battery devices] PowerTOP

[Possible but no documentation] https://gitlab.com/MarcoCouto/c-lem

4.2 Design of experiments

4.3 Explaining testing environment

4.4 Results

these are the first results from the experiments, they need a lot of finetuning

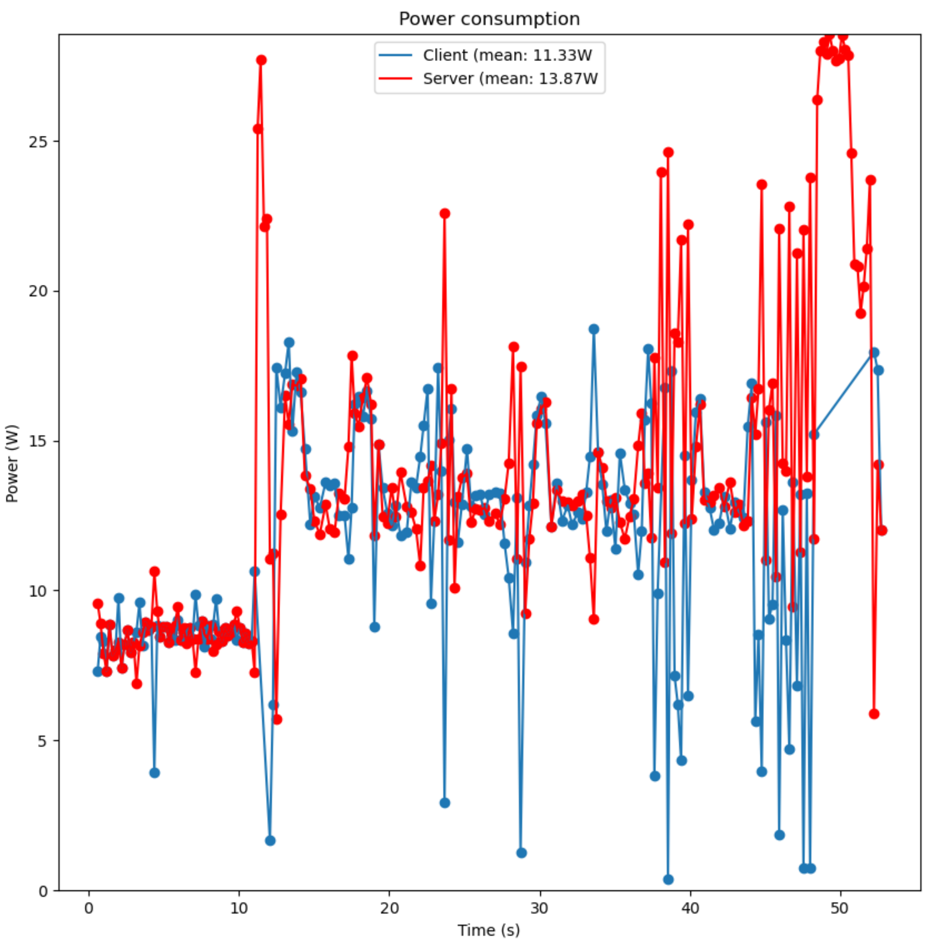

Figure 4.1: Running the SNNI cheetah on sqnet

4.5 RQb, what are the differences on server and client side and what are the implications

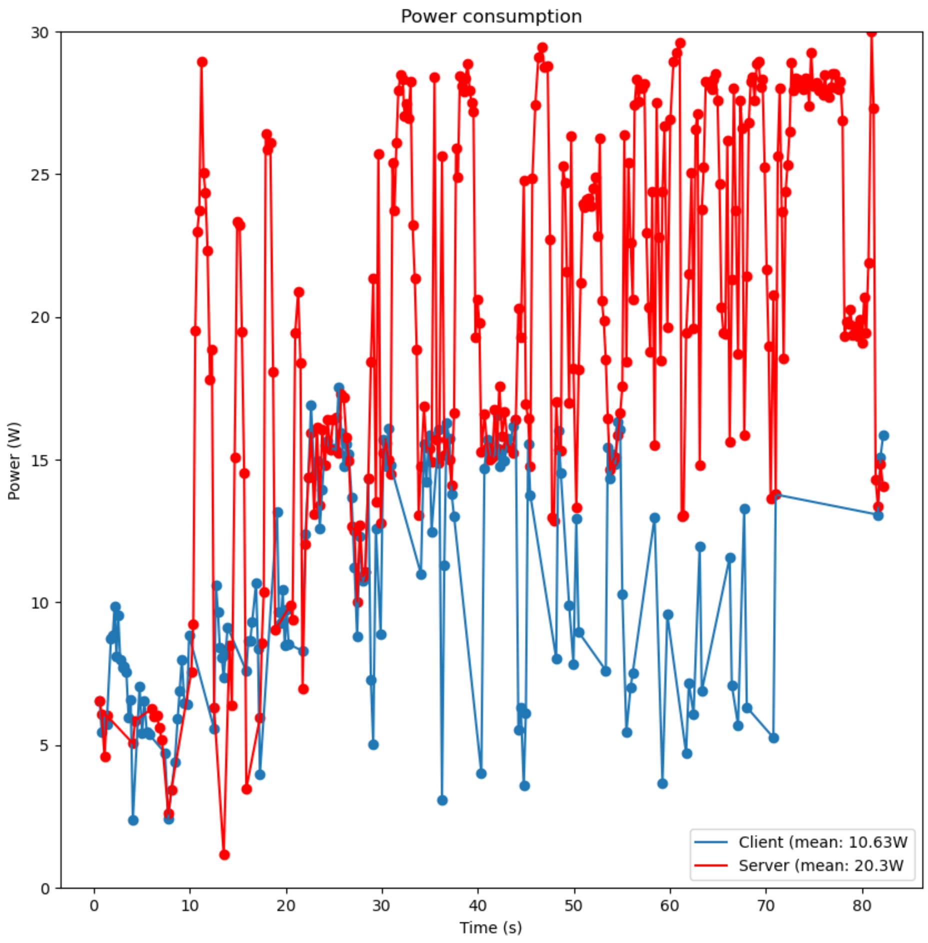

Figure 4.2: Running the SNNI CryptFlow2 on sqnet

CHAPTER 5

Conclusion

5.1 Conclusion

5.2 Discussion

5.3 Ethics