On the Practical Uses of Experimental Adversarial Neural Cryptography

Korn Sooksatra * and Pablo Rivas †, *Senior, IEEE*
School of Engineering and Computer Science
Department of Computer Science, Baylor University
Email: {*Korn_Sooksatra1,†Pablo_Rivas}@Baylor.edu

***Abstract*—With the rise of generative adversarial networks (GANs), many areas have seen remarkable improvements, e.g., computer vision, natural language processing, and the medical field. Notably, cryptography has been fueled by GANs producing adversarial neural cryptography (ANC). However, in these five years, ANC has little documented experimentation and applications that can be used in the real world. This paper aims to perform experiments on ANC to verify if the current status of ANC is ready for practical implementations of symmetric-key encryption. In our investigation, we assess several entities in ANC during training, encryption, and decryption of an ANC model, including decryption accuracy analysis. Furthermore, we study the resources required for deployment using different quantization techniques to reduce the size of an ANC model and its impact on performance and decryption accuracy. Our study provides enough data for offering practical advice for using and implementing ANC models.**

***Index Terms*—adversarial generative network, adversarial neural cryptography, cryptography, privacy**

I. Introduction

Generative adversarial networks (GANs) [7] have been in attention and widely applied in several applications (e.g. computer vision [8], [10], [12], natural language processing [11], [14] and medical field [4], [9]. Essentially, Abadi and Anderson [2] proposed adversarial neural cryptography (ANC), applying GANs to create secure communication and provided privacy between two entities. Later, Coutinho *et al.* [6] improved ANC to make encryption and decryption algorithms stronger, and Sooksatra and Rivas [15] reviewed these works and the relationship between machine learning and cryptography. Further, Sooksatra and Rivas [16] adapted ANC to create a learning scheme that could perform a cryptographic integrity check against a man-in-the-middle attack. As discussed, ANC has been extensively applied in the field of cryptography.

Because the work in [2] is the main contribution of ANC, we extended its experiments to evaluate if it could be used in secure communication in general and the one in a mobile device that had a small memory. This work shows a) how the loss values of the sender, receiver and adversary during their training in several input sizes; b) how much encryption and decryption time vary in several input sizes and the numbers of test samples; c) how decryption accuracy changes in many input sizes; d) the experiment on the quantized versions of the sender and receiver in terms of loss values during training, encryption and decryption time and decryption accuracy in many settings.

This paper is organized as follows: Section II provides a review of adversarial neural cryptography and quantization techniques for mobile devices; Section III shows extensive experiments and their results; Section IV addresses the results from the experiments and compare ANC to AES in terms of encryption and decryption time; finally, Section V draws our concluding remarks.

II. Background

This section discusses preliminary knowledge before going through the extended experiments in Section III. First, we explain what ANC is and how it works. Then, second, we briefly demonstrate what post-training quantization is and its variants to understand how this minimizes the size of a learning model.

A. Adversarial neural cryptography (ANC)

This scheme was first created in [2], and there are the following works [6], [16] that extended and utilized this scheme. ANC in [2] consists of a sender (named Alice), a receiver (named Bob) and an adversary (named Eve). The scheme works as follows. First, Alice has plaintext P and wants to send it to Bob. However, Eve can eavesdrop on a message between Alice and Bob, and Alice does not want Eve to understand the message. Hence, Alice uses a shared key K to encrypt P, obtains ciphertext C and sends it to Bob. Then, after Bob receives C, it can use K to decrypt C and hopes to fully obtain P where Eve does not have K. It is worth noting that the sizes of P, C and K are the same, which is N. This scheme can be illustrated in Figure 1, and its goals can be summarized as follows:

- Alice needs to find a way to encrypt P, and Bob needs to find a way to decrypt C so that Bob can fully recover P from C and K.
- Alice needs to determine an encryption method so that Eve cannot recover P from C.

Next, we briefly review the loss function of Alice, Bob and Eve defined in [2] to understand how to train them and the experiments in Section III. Bob's loss function was defined as

$$L_B(P, P_{Bob}) = d(P, P_{Bob})$$

where $d(P, P_{Bob}) = \sum_{i=0}^{N-1} |P^i - P^i_{Bob}|$ and P^i is bit i of P. This loss is high when Bob only correctly recover a few

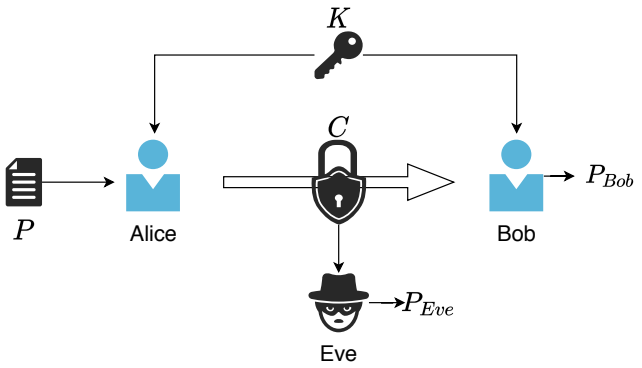


Fig. 1: Training scheme of adversarial neural cryptography which includes Alice as the sender, Bob as the receiver and Eve as the adversary.

bits. Further, Eve's loss function was similar to Bob's one and defined as

$$L_E(P, P_{Eve}) = d(P, P_{Eve}). \tag{1}$$

The last loss function is for both Alice and Bob. Since they had to cooperate with each other to create a strong C against Eve and fully recover P from C in Bob's side, their cooperative loss function was defined as

$$L_{AB}(P, P_{Bob}, P_{Eve}) = L_B(P, P_{Bob}) + \left(\frac{\frac{N}{2} - L_E(P, P_{Eve})}{\frac{N}{2}} \right)^2. \tag{2}$$

The first part of (2) is simply Bob's loss function because Alice's and Bob's cooperative loss should be proportional to Bob's loss. Then, the second part is to make Eve's decryption not different from a random guess which is $\frac{N}{2}$ error.

Furthermore, the algorithm for training this scheme was proposed in [2] which followed the instruction of GANs in [7]. In short, it starts with freezing Eve to train Alice and Bob one time by using the gradient of L_{AB} in (2), Then, it changes to train Eve twice by freezing Alice and Bob and using the gradient of L_E in (1). It repeats these training for 20 epochs.

Despite the experiments in [2], we need to know the results in several settings of N (i.e., 16, 32, 64 and 128) to use this in the real world in which the number of bits is not only 16. Also, we need to experiment regarding encryption and decryption time because we expect these values not to be very different from the ones of widely used symmetric encryption algorithms (e.g., advanced encryption standard (AES) [13]).

B. Post-training quantization

We use this library in Tensorflow [1]. Specifically, this is a technique after training to reduce the size of Alice (i.e., encryption method) and Bob (i.e., decryption method) for a small device (e.g., a mobile device) that has a small memory. However, this sacrifices a little accuracy of the models. We investigate two quantization methods: dynamic range quantization and float16 quantization.

1) Dynamic range quantization: This quantization is the simplest method that converts the weights of a learning model from floating-point to 8-bit integer. Therefore, this method can reduce the size of the learning model up to 4 times.

2) Float16 quantization: This method reduces the size of a learning model by quantizing all the weights to 16-bit floating-point. Hence, this can reduce the size of the learning model up to 2 times and cause a minimal loss of accuracy.

III. Experiments and Results

This section describes how we performed our experiments and shows the particular results. Further, we explain the implication of the results. We demonstrate the results in terms of the following aspects:

- Alice's, Bob's and Eve's losses during their training.
- Alice's encryption time and Bob's decryption time.
- Bob's and Eve's decryption accuracy.
- The quantized versions of Alice and Bob.

A. Experimental setup

We have experimented with Tensorflow version 2 [1] on Google Colab Pro [5], and the sizes of plaintext, key and ciphertext are equally N bits. We trained all the entities for 20 epochs and generated training samples (i.e., plaintexts and keys) whose sizes depended on experiments; however, mainly, their sizes are 1000000. Further, the training time with 1000000 training samples is around 6000 to 6500 seconds.

B. Alice, Bob and Eve during training

During the training with 1000000 generated training samples, Alice and Bob successfully found a way to make Eve poorly guess the plaintext from its particular ciphertext and also allow Bob to recover the plaintext as seen in Figure 2. However, the difference between Bob's and Eve's losses is inversely proportional to N. Plus, the losses in Figure 2d are very close to each other. Therefore, the result after $N = 64$ is not acceptable with 1000000 training samples because the architecture is larger and increases the number of parameters. Hence, it needs more training samples to improve its performance.

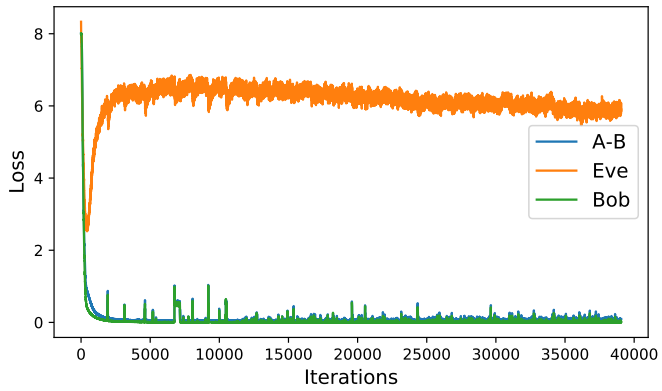
C. Encryption and decryption time

We have also experimented with the scheme's encryption time and decryption time and found that no matter N is, those values are almost the same. Further, both encryption and decryption time linearly increase with the size of test samples, as demonstrated in Figure 3.

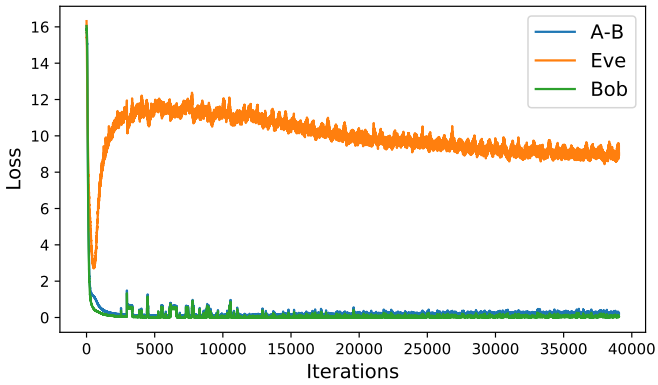
D. Accuracy

Furthermore, we have investigated Bob's and Eve's accuracy with the variation of N and the fixed number of training samples (i.e., 1000000 samples). According to Figure 4, the performance of the scheme was worse when we increased N. In other words, Bob's accuracy decreased, and Eve's accuracy increased when N increased.

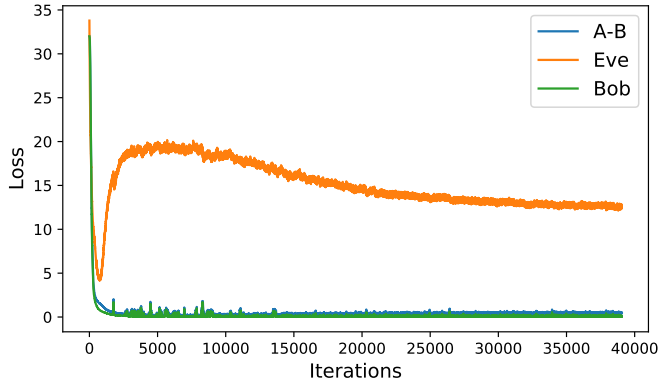
Additionally, we also discovered that when we fixed N, we could increase Bob's accuracy and decrease Eve's accuracy by increasing the number of training samples. We empirically guarantee this claim with the illustration in Figure 5.



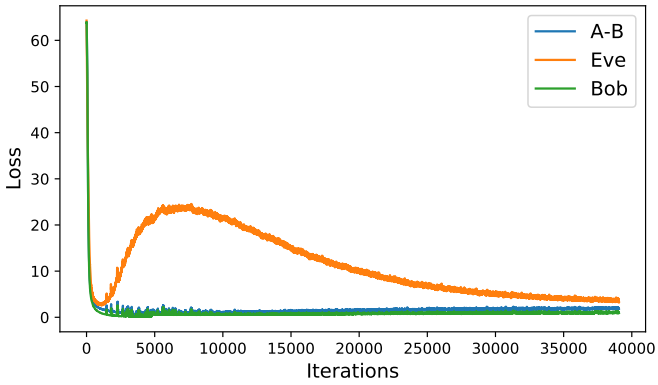
(a) $N = 16$.



(b) $N = 32$.

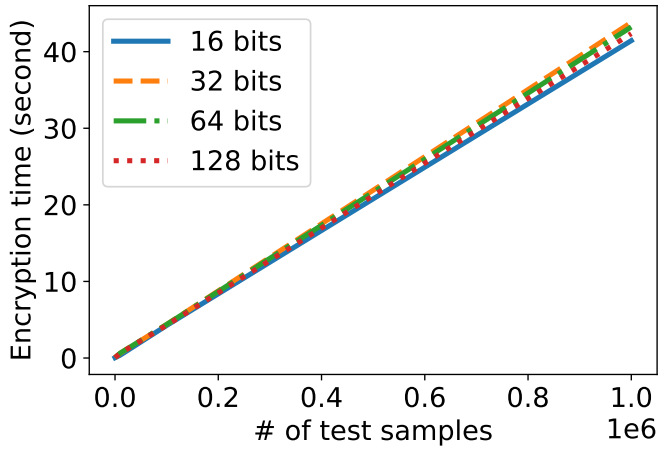


(c) $N = 64$.

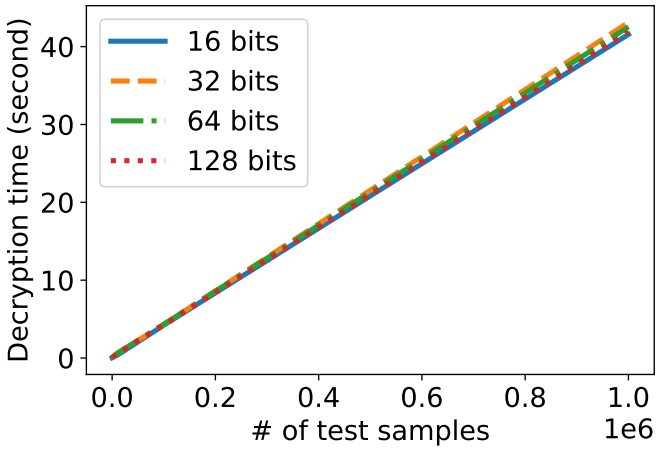


(d) $N = 128$.

Fig. 2: Progression of the loss functions during training iterations for different N values.



(a) Encryption time over the number of test samples.

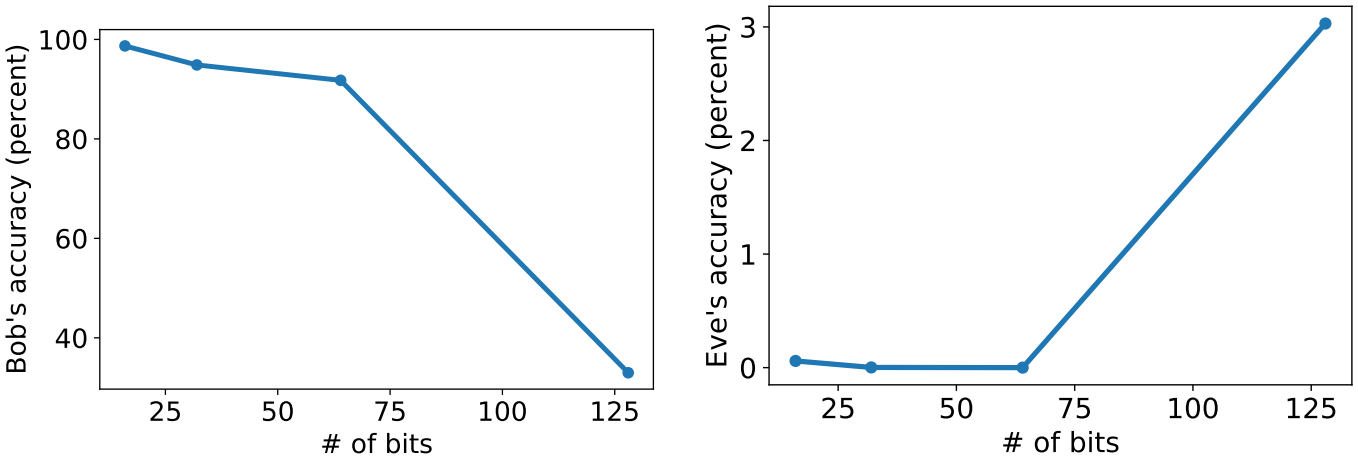


(b) Decryption time over the number of test samples.

Fig. 3: Encryption time of Alice and decryption time of Bob over the numbers of test samples for each N.

E. Quantization for mobile devices

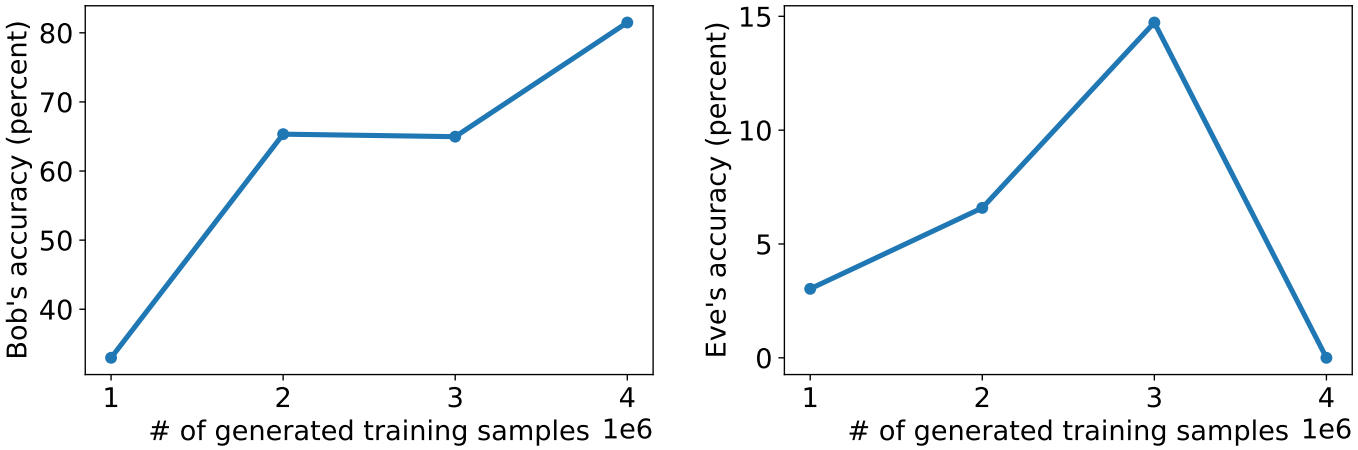
In this section, we have modified the trained models to reduce their sizes so that they could fit mobile devices which had tiny memories. We have used dynamic range quantization and float16 quantization for the quantized models. As seen in Figure 6, 7, 8 and 9, the encryption and decryption time of all the trained models linearly increased with the number of test samples. Also, the encryption and decryption time of the quantized models were lower than the ones of the standard model. Further, these gaps were getting smaller when N is larger. Next, when we considered only the quantized models, the encryption and decryption time of dynamic quantization



(a) Bob's accuracy.

(b) Eve's accuracy.

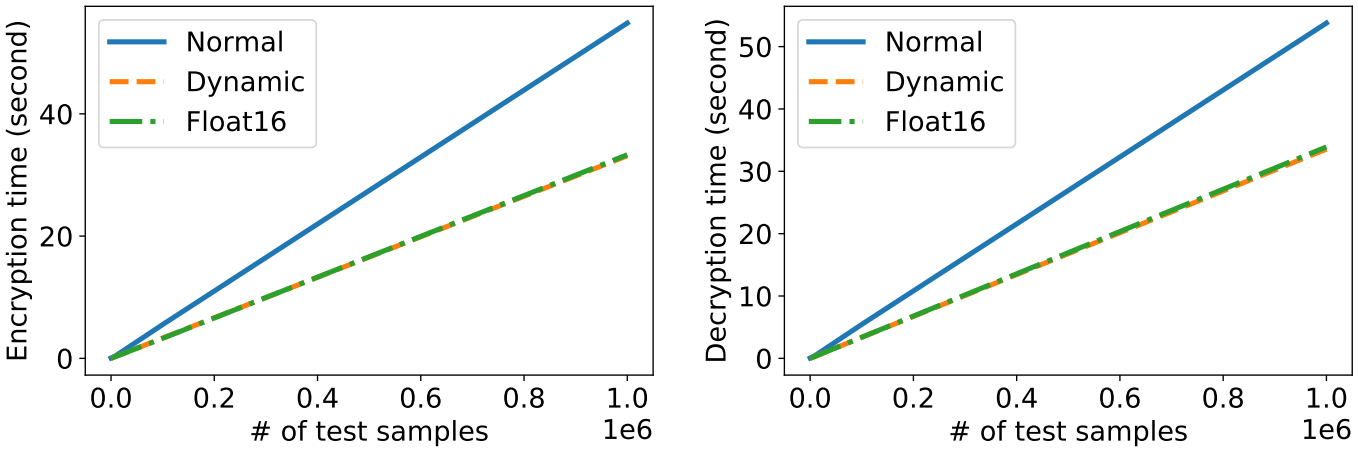
Fig. 4: Accuracy of recovering plaintext over N.



(a) Bob's accuracy.

(b) Eve's accuracy.

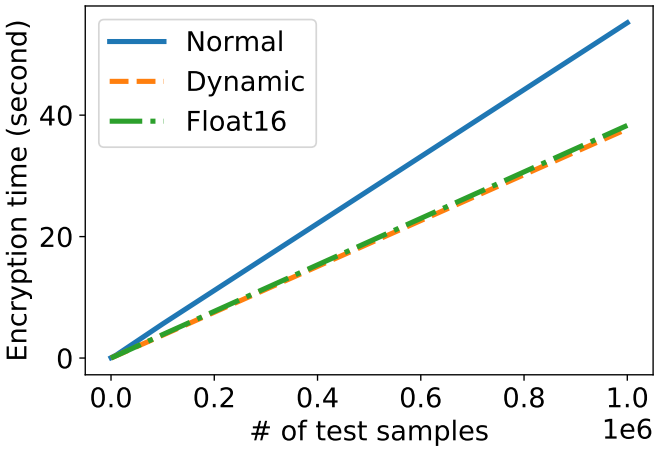
Fig. 5: Accuracy of recovering plaintext with $N = 128$ over the numbers of training samples.



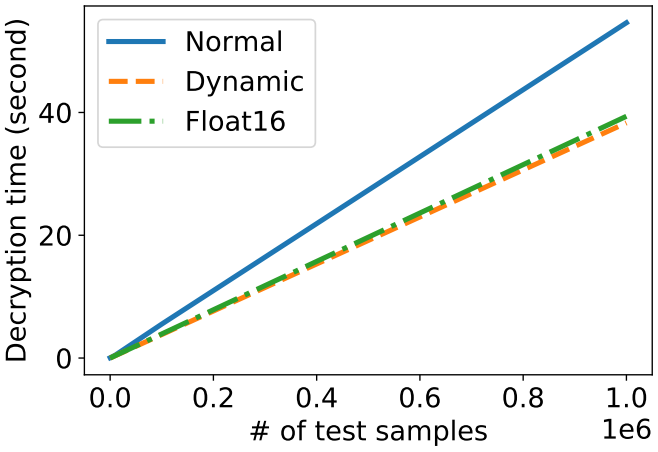
(a) Encryption time.

(b) Decryption time.

Fig. 6: Encryption and decryption time of standard and tiny models with $N = 16$ over the number of test samples.

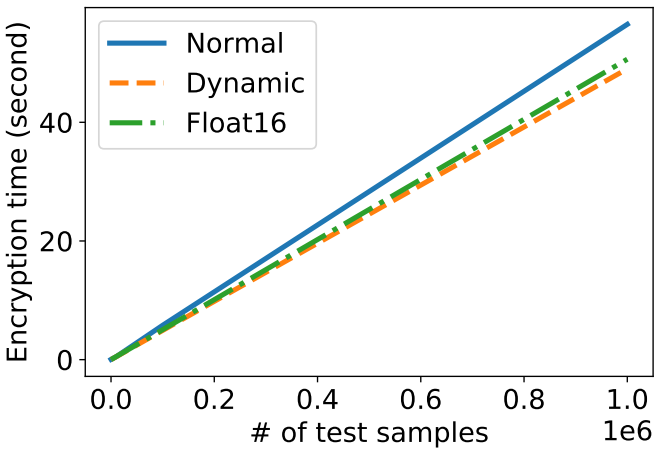


(a) Encryption time.

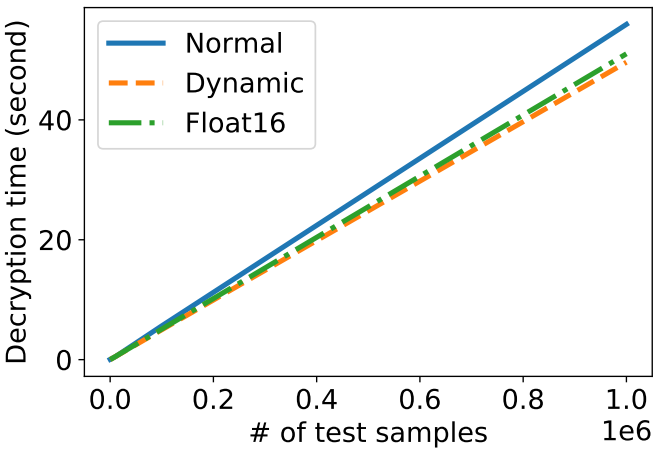


(b) Decryption time.

Fig. 7: Encryption and decryption time of standard and tiny models with $N = 32$ over the number of test samples.

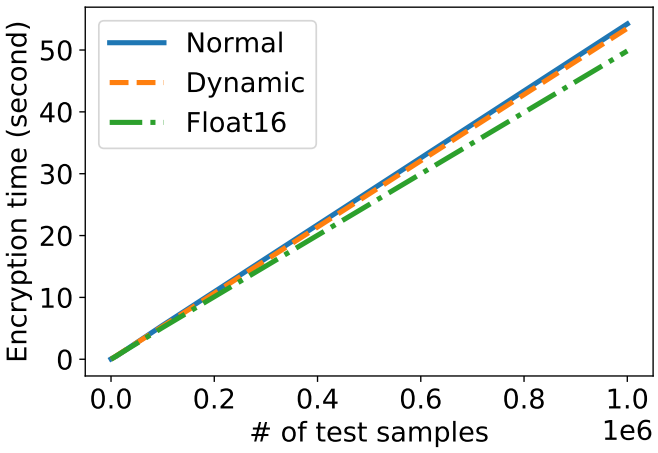


(a) Encryption time.

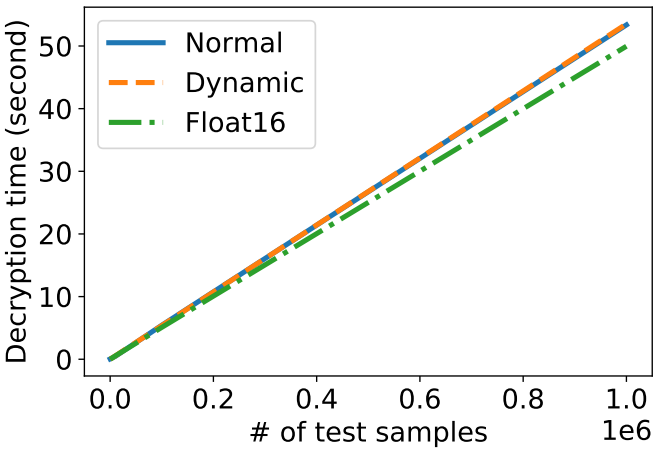


(b) Decryption time.

Fig. 8: Encryption and decryption time of standard and tiny models with $N = 64$ over the number of test samples.

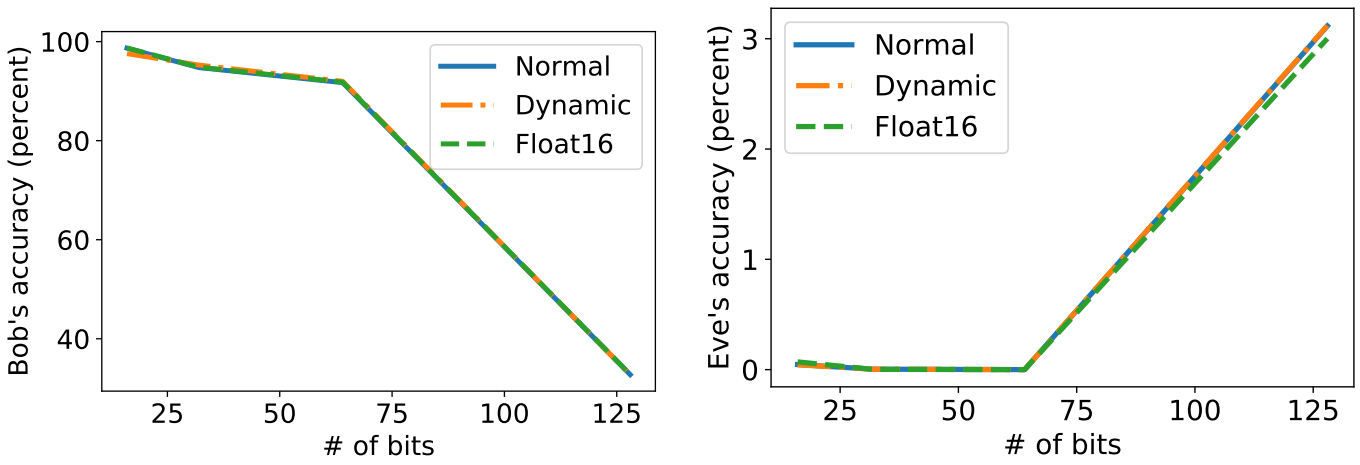


(a) Encryption time.



(b) Decryption time.

Fig. 9: Encryption and decryption time of standard and tiny models with $N = 128$ over the number of test samples.



(a) Bob's accuracy.

(b) Eve's accuracy.

Fig. 10: Accuracy of recovering plaintext over N in several models.

and float16 quantization were very close to each other when $N = 16$ and $N = 32$ as seen in Figure 6 and 7; nonetheless, float16 quantization took longer time to encrypt and decrypt messages than dynamic quantization when N increased as demonstrated in Figure 8 and 9.

Moreover, as expected, the accuracy of Bob and Eve in the quantized models was not significantly different from the standard one as demonstrated in Figure 10.

In addition, we have investigated the sizes of Alice and Bob in dynamic range quantization, float16 quantization and the standard one and found that both quantizations could significantly reduce the sizes according to Table I. Further, dynamic range quantization was more effective than float16 quantization since it had a constant ratio that was less than float16 quantization. Plus, the ratio of float16 quantization is proportional to N; this implies that it is not good when N is large (e.g., 128).

IV. Discussion

According to the training time of 6000 to 6500 seconds for creating one cipher, we can utilize a parallel system to reduce this number since we need to create as many ciphers as possible to often alter ciphers for users to prevent them from attackers. However, for large N, ANC takes much time to create one cipher because it needs much more training samples to decrease Eve's accuracy as seen in Figure 4b and increase Bob's accuracy as seen in Figure 4a. Further, we compare the encryption and decryption time of ANC which are bound to N^2, to the ones of advanced encryption standard (AES) [3] which is bounded to N and generally used for symmetric encryption. Nevertheless, we can compare only in the setting of $N = 128$, the minimum size of an input in AES. Hence, we used *pycrypto* module in Python to implement AES and obtained its encryption and decryption time with several numbers of test samples. Then, we used the information in Figure 3 to compare to the result of AES and summarized it in Table II and III. Noticeably, AES is around $50-60$ times faster than ANC in terms of both encryption time and decryption time. If we expect to replace AES with ANC, this needs to be improved.

Additionally, we applied quantization on ANC to reduce its size for practical secure communication in a mobile device. As discussed in the previous section, we could encrypt and decrypt a message in a quantized model faster than in a standard model. Furthermore, as noticed, dynamic range quantization takes significantly less time to encrypt and decrypt messages than float16 quantization in large N, and the quantized ANC of dynamic range quantization is much smaller than the one of float16 quantization. Moreover, the accuracy of Bob and Eve in float16 quantization is slightly worse than the ones of dynamic range quantization; however, the accuracy of those methods are almost the same as the regular ANC as seen in Figure 10. Thus, we recommend using dynamic range quantization rather than using float16 quantization.

V. Conclusion

In this paper, we briefly explain what ANC is [2], its loss function and the training algorithm. Further, we propose using a quantization library (i.e., dynamic range quantization and float16 quantization) in Tensorflow version 2 to reduce ANC's size for mobile devices. According to all the results, those quantized models worked very well despite sacrificing a little accuracy. Then, we show the result of our experiment that can be concluded as follows:

- ANC with a large N needs more training samples than ANC with a smaller N.
- Encryption and decryption time of Alice and Bob are almost constant with several settings of N.
- Among the quantized versions, dynamic range quantization outperforms float16 quantization in terms of encryption and decryption time and Bob's accuracy in our experiments.

TABLE I: Size of Alice or Bob in adversarial neural cryptography and its quantized versions and the ratio between the sizes of its quantized versions and the one of its standard one.

Type	$N = 16$	$N = 32$	$N = 64$	$N = 128$
Standard (KB)	41	53	102	299
Dynamic (KB)	9	13	25	75
Dynamic (ratio)	0.22	0.25	0.25	0.25
Float16 (KB)	12	18	43	141
Float16 (ratio)	0.29	0.34	0.42	0.47

TABLE II: Encryption time of ANC [2] and AES with $N = 128$ and several numbers of test samples. Note that Ratio is computed by dividing encryption time of AES to encryption time of ANC.

Method	100 samples	1000 samples	10000 samples	100000 samples	1000000 samples
ANC	0.049	0.12	0.61	4.28	42.64
AES	0.00095	0.0019	0.0081	0.075	0.75
Ratio	51.58	63.16	75.31	57.07	56.85

TABLE III: Decryption time of ANC [2] and AES with $N = 128$ and several numbers of test samples. Note that Ratio is computed by dividing decryption time of AES to decryption time of ANC.

Method	100 samples	1000 samples	10000 samples	100000 samples	1000000 samples
ANC	0.048	0.1	0.6	4.22	42.18
AES	0.0015	0.0018	0.0076	0.079	0.75
Ratio	32	55.56	78.94	53.42	56.24

Furthermore, we suggest that solutions for reducing training, encryption, and decryption time are required because we need more training samples for ANC with a big N to use ANC in real-world secure communication scenarios.

From the perspective discussed here, ANC can be beneficial. While AES is widely used for symmetric encryption and performs very well, one can produce as many ciphers as needed with ANC. Thus, it can be difficult for attackers to find a vulnerability on a cipher created with ANC before another one replaces the cipher. Therefore, if ANC-based ciphers can be popularized for symmetric-key encryption, it will positively impact cryptography.

ACKNOWLEDGMENT

The work presented here was supported in part by the Baylor AI lab in Baylor University's Department of Computer Science.

REFERENCES

[1] Martín Abadi. Tensorflow: learning functions at scale. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, pages 1–1, 2016.

[2] Martín Abadi and David G Andersen. Learning to protect communications with adversarial neural cryptography. *arXiv preprint arXiv:1610.06918*, 2016.

[3] Mehdi-Laurent Akkar and Christophe Giraud. An implementation of des and aes, secure against some attacks. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 309–318. Springer, 2001.

[4] Mostapha Benhenda. Chemgan challenge for drug discovery: can ai reproduce natural chemical diversity? *arXiv preprint arXiv:1708.08227*, 2017.

[5] Ekaba Bisong. *Building machine learning and deep learning models on Google Cloud Platform*. Springer, 2019.

[6] Murilo Coutinho, Robson de Oliveira Albuquerque, Fábio Borges, Luis Javier Garcia Villalba, and Tai-Hoon Kim. Learning perfectly secure cryptography to protect communications with adversarial neural cryptography. *Sensors*, 18(5):1306, 2018.

[7] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[8] Nikolay Jetchev, Urs Bergmann, and Roland Vollgraf. Texture synthesis with spatial generative adversarial networks. *arXiv preprint arXiv:1611.08207*, 2016.

[9] Nathan Killoran, Leo J Lee, Andrew Delong, David Duvenaud, and Brendan J Frey. Generating and designing dna with deep generative models. *arXiv preprint arXiv:1712.06148*, 2017.

[10] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[11] Jiwei Li, Will Monroe, Tianlin Shi, Sébastien Jean, Alan Ritter, and Dan Jurafsky. Adversarial learning for neural dialogue generation. *arXiv preprint arXiv:1701.06547*, 2017.

[12] Liqian Ma, Xu Jia, Qianru Sun, Bernt Schiele, Tinne Tuytelaars, and Luc Van Gool. Pose guided person image generation. *arXiv preprint arXiv:1705.09368*, 2017.

[13] Prerna Mahajan and Abhishek Sachdeva. A study of encryption algorithms aes, des and rsa for security. *Global Journal of Computer Science and Technology*, 2013.

[14] Tingting Qiao, Jing Zhang, Duanqing Xu, and Dacheng Tao. Mirrorgan: Learning text-to-image generation by redescription. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1505–1514, 2019.

[15] Korn Sooksatra and Pablo Rivas. A review of machine learning and cryptography applications. In *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 591–597. IEEE, 2020.

[16] Korn Sooksatra and Pablo Rivas. Learning on integrity check for secure communications with adversarial neural cryptography. *International Joint Conference on Neural Networks (IJCNN)*, 2021.