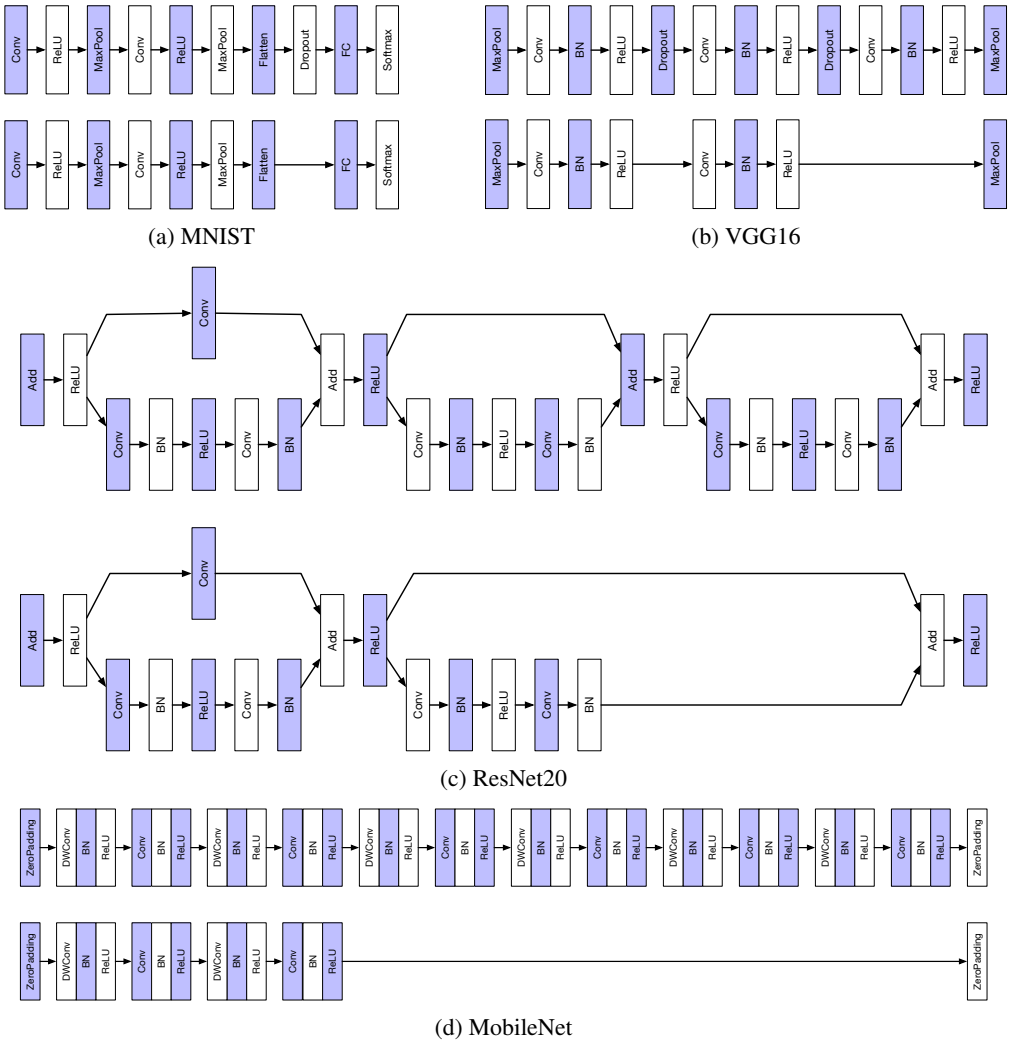
Table 1: Victim Models Information.

	Datasets	Input Shape	# of Parameters	# of layers	# of layer types
MNIST	MNIST	(28,28,1)	34,826	11	7
VGG16	CIFAR-10	(32,32,3)	150,001,418	60	7
ResNet20	CIFAR-10	(32,32,3)	19 274,442	72	8
MobileNet	CIFAR-10	(32,32,3)	3,239,114	91	9



(a) MNIST (b) VGG16

(c) ResNet20

(d) MobileNet

Figure 6: Architecture Comparison. This figure partially shows the difference of the network architecture between original models and extracted models. MNIST

ResNet20, and MobileNet. The detailed information of all victim models is shown in Table 1. All the pre-trained models are designed in the Keras framework (Team, 2022a; Team, 2022b) with Tensorflow as the backbone and compiled by TVM (Chen et al., 2018) to generate the binaries. We use LLVM (Lattner and Adve, 2004) as our host platform. As mentioned in Section 2, we assume that we only have access to the DNN binary library during the attack.

5.2 Architectural Completeness

This section compares the architectures of the original models and the extracted models. Figure 6 visualizes part of the architecture of original models and extracted models for MNIST, VGG16, ResNet20, and MobileNet. Each block in the figure represents a DNN layer. As shown in the figure, the basic architecture remains the same in extracted models. However, several layers are missing. First, the dropout layer is

Table 2: Statistics of the comparison between original model and extracted model.

	Test accuracy		Layer number			Layer type number	
	Original (%)	Extracted (%)	Original	Extracted	Percentage (%)	Original	Extracted
MNIST	99.17	99.04	11	10	90.91	7	6
VGG16	93.16	90.59	60	38	63.33	7	6
ResNet20	91.65	83.92	72	49	68.06	8	8
MobileNet	83.16	75.00	91	66	72.53	9	8

used to avoid over-fitting during the training phase by setting randomly selected input units to 0, so it will be invalid during the inference.

Regarding other missing layers, as we mentioned in Section 4.3, the same type of layer with the same I/O dimensions and attributes will produce the same layer function. This rule of thumb allows us to identify the layer type and its related attributes but also leads to the situation that the layer function can be reused in the DNN executable. As shown in Figure 6, the recovery of MobileNet is least satisfying. The reason for missing a larger chunk in this model is because that the original DNN model is basically the composition of the same pattern repeated for four times. Since our attack targets the DNN library, it is difficult to infer the number of repetitions solely from the code, as the same code can be executed for arbitrary times during inference. On the other hand, Figure 6a shows that as long as each layer of the DNN model holds an unique layer function, we are able to fully recover the whole network architecture.

5.3 Accuracy of Extracted Models

We compared other statistics information between original DNN models with the extracted ones as shown in Table 2. In order to evaluate the functionality of the extracted model, we re-trained the extracted models and compared their accuracy with the original models. As for the training settings, we use the same hyper-parameter as the original one for a more precise comparison. For the record, we re-train the model to demonstrate the accuracy of extracted model architecture which does not assume that our framework requires the training dataset and the hyper-parameters. As shown in the Table 2, with the missing layers, it is not surprising that the accuracy of VGG16, ResNet20, and MobileNet is worse than the original ones. However, we can see that the accuracy drop of ResNet20 and MobileNet is more significant than that of VGG16, although we recover more layers for ResNet20 and MobileNet. We guess that the importance of layers varies inside the neural network architecture. Although VGG16 missed more layers, the key skeleton still remains. As for the layer types, so far, the only layer we are not able to recover is the Dropout layer, which will disappear during the inference process.

6 DISCUSSION

The limitation of our attack framework is due to the connection between layers. The shared library compiled from DNN models only contains the distinct layer functions, so we need to know the exact used number of each layer function. One solution is that we can enlarge our search space and use meta-learning to make us closer to the original network architecture. For example, we do not limit the usage of any function layers and allow the search engine to explore the possible combination. We train and test each explored architecture's accuracy and leave the best result. However, time-consuming will be out of imagination. On the other hand, if we can access the JSON file or even the parameter file, we can recover the precise DNN models, which makes our work more meaningful.

7 RELATED WORK

The basic logic of the model extraction attack is leveraging the information gathered from a different source to leak the vital features of the machine learning model. The most commonly used source is the side channel (Hua et al., 2018; Yan et al., 2020; Wei et al., 2018; Xiang et al., 2020; Duddu et al., 2018; Hunt et al., 2020; Batina et al., 2019). For example, by exploiting the memory and timing side-channel, (Hua et al., 2018) presented a model extraction attack to infer the network architecture and identify the value of parameters of the convolutional neural network (CNN) running on a hardware accelerator. Bus traffic is also an important information source (Zhu et al., 2021; Hu et al., 2019). (Zhu et al., 2021) identified a new attack surface based on encrypted PCIe traffic and fully extracted the DNN model with the exact model characteristics, and achieved the same inference accuracy as the target model. Some exciting work also relied on the query-prediction pairs from the target model (Tramèr et al., 2016; Oh et al.,

2019; Orekondy et al., 2019; Kariyappa et al., 2021). (Tramèr et al., 2016) relied on the information carried by the output from the ML prediction APIs to generate a similar or the same model and successfully applied the attack against the online services.

8 CONCLUSIONS

The rising of DL compilers and the privatization situation introduce a new threat to the ML community. Several novel attack framework has been proposed. However, all of them assume they have full access to the DNN binaries. In this paper, we narrow down the threat model and demonstrate that with only the DNN binary library, we can leak the network architecture information of DNN models. We propose a framework, namely LibSteal, using only the DNN library to get the layer types, attributes, dimensions, and even compatible topology connections. We implemented a prototype of LibSteal and evaluated it on four DNN models compiled from TVM. The evaluation results indicate that our framework can reconstruct a similar or even equivalent model architecture compared to the original one, achieving comparable accuracy after training with the public datasets with only the library files.

ACKNOWLEDGEMENTS

This research was supported in part by the National Science Foundation (NSF) grant CNS-1652790 and the Office of Naval Research (ONR) grant N00014-17-1-2894.

REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *OSDI 16*, pages 265–283.

Batina, L., Bhasin, S., Jap, D., and Picek, S. (2019). CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel. In *USENIX Security 19*, pages 515–532.

Brumley, D., Jager, I., Avgerinos, T., and Schwartz, E. J. (2011). Bap: A binary analysis platform. In *CAV 2011*, pages 463–469. Springer.

Chen, S., Khanpour, H., Liu, C., and Yang, W. (2022). Learning to reverse dnns from ai programs automatically. *arXiv preprint arXiv:2205.10364*.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. (2015). Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*.

Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L., et al. (2018). TVM: An automated end-to-end optimizing compiler for deep learning. In *OSDI 18*, pages 578–594.

Chollet, F. et al. (2015). Keras. https://keras.io.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML 2008*, pages 160–167.

Cyphers, S., Bansal, A. K., Bhiwandiwalla, A., Bobba, J., Brookhart, M., Chakraborty, A., Constable, W., Convey, C., Cook, L., Kanawi, O., et al. (2018). Intel ngraph: An intermediate representation, compiler, and executor for deep learning. *arXiv preprint arXiv:1801.08058*.

Ding, S. H., Fung, B. C., and Charland, P. (2019). Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In *IEEE S&P 2019*, pages 472–489. IEEE.

Duddu, V., Samanta, D., Rao, D. V., and Balas, V. E. (2018). Stealing neural networks via timing side channels. *arXiv preprint arXiv:1812.11720*.

Eagle, C. (2011). *The IDA pro book*. no starch press.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *ICASSP 2013*, pages 6645–6649. Ieee.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *CVPR 2016*, pages 770–778.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Hu, X., Liang, L., Deng, L., Li, S., Xie, X., Ji, Y., Ding, Y., Liu, C., Sherwood, T., and Xie, Y. (2019). Neural network model extraction attacks in edge devices by hearing architectural hints. *arXiv preprint arXiv:1903.03916*.

Hua, W., Zhang, Z., and Suh, G. E. (2018). Reverse engineering convolutional neural networks through side-channel information leaks. In *DAC 2018*, pages 1–6. IEEE.

Hunt, T., Jia, Z., Miller, V., Szekely, A., Hu, Y., Rossbach, C. J., and Witchel, E. (2020). Telekine: Secure computing with cloud GPUs. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 817–833.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. (2017). In-datacenter performance analysis of a tensor processing unit. In *ISCA '17*, pages 1–12.

Kariyappa, S., Prakash, A., and Qureshi, M. K. (2021). Maze: Data-free model stealing attack using zeroth-order gradient estimation. In *CVPR 2021*, pages 13814–13823.

Kato, S., Takeuchi, E., Ishiguro, Y., Ninomiya, Y., Takeda, K., and Hamada, T. (2015). An open approach to autonomous vehicles. *IEEE Micro*, 35(6):60–68.

Ketkar, N. (2017). Introduction to keras. In *Deep learning with Python*, pages 97–111. Springer.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.

Lattner, C. and Adve, V. (2004). Llvm: A compilation framework for lifelong program analysis & transformation. In *CGO 2004.*, pages 75–86. IEEE.

Leary, C. and Wang, T. (2017). Xla: Tensorflow, compiled. *TensorFlow Dev Summit*.

LeCun, Y. (1998). The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

Li, M., Liu, Y., Liu, X., Sun, Q., You, X., Yang, H., Luan, Z., and Qian, D. (2020). The deep learning compiler: A comprehensive survey. *arXiv preprint arXiv:2002.03794*.

Liang, Y., Cai, Z., Yu, J., Han, Q., and Li, Y. (2018). Deep learning based inference of private information using embedded sensors in smart devices. *IEEE Network*, 32(4):8–14.

Liu, Z., Yuan, Y., Wang, S., Xie, X., and Ma, L. (2022). Decompiling x86 deep neural network executables. *arXiv preprint arXiv:2210.01075*.

Markham, A. and Jia, Y. (2017). Caffe2: Portable high-performance deep learning framework from facebook. *NVIDIA Corporation*.

Oh, S. J., Schiele, B., and Fritz, M. (2019). Towards reverse-engineering black-box neural networks. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 121–144. Springer.

Orekondy, T., Schiele, B., and Fritz, M. (2019). Knockoff nets: Stealing functionality of black-box models. In *CVPR 2019*, pages 4954–4963.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. In *NIPS-W*.

Rotem, N., Fix, J., Abdulrasool, S., Catron, G., Deng, S., Dzhabarov, R., Gibson, N., Hegeman, J., Lele, M., Levenstein, R., et al. (2018). Glow: Graph lowering compiler techniques for neural networks. *arXiv preprint arXiv:1805.00907*.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61:85–117.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Song, D., Brumley, D., Yin, H., Caballero, J., Jager, I., Kang, M. G., Liang, Z., Newsome, J., Poosankam, P., and Saxena, P. (2008). Bitblaze: A new approach to computer security via binary analysis. In *ICISSP 2008*, pages 1–25. Springer.

Team, K. (2022a). Keras applications.

Team, K. (2022b). Keras examples.

Tramèr, F., Zhang, F., Juels, A., Reiter, M. K., and Ristenpart, T. (2016). Stealing machine learning models via prediction apis. In *USENIX Security 16*, pages 601–618.

Vasilache, N., Zinenko, O., Theodoridis, T., Goyal, P., DeVito, Z., Moses, W. S., Verdoolaege, S., Adams, A., and Cohen, A. (2018). Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions. *arXiv preprint arXiv:1802.04730*.

Wang, F. and Shoshitaishvili, Y. (2017). Angr-the next generation of binary analysis. In *2017 IEEE Cybersecurity Development (SecDev)*, pages 8–9. IEEE.

Wang, S., Wang, P., and Wu, D. (2015). Reassembleable disassembling. In *USENIX Security 15*, pages 627–642.

Wei, L., Luo, B., Li, Y., Liu, Y., and Xu, Q. (2018). I know what you see: Power side-channel attack on convolutional neural network accelerators. In *ACSAC '18*, pages 393–406.

Wu, C.-J., Brooks, D., Chen, K., Chen, D., Choudhury, S., Dukhan, M., Hazelwood, K., Isaac, E., Jia, Y., Jia, B., et al. (2019). Machine learning at facebook: Understanding inference at the edge. In *HPCA 2019*, pages 331–344. IEEE.

Wu, R., Kim, T., Tian, D. J., Bianchi, A., and Xu, D. (2022). DnD: A cross-architecture deep neural network decompiler. In *USENIX Security 22*, pages 2135–2152.

Xiang, Y., Chen, Z., Chen, Z., Fang, Z., Hao, H., Chen, J., Liu, Y., Wu, Z., Xuan, Q., and Yang, X. (2020). Open dnn box by power side-channel attack. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(11):2717–2721.

Yan, M., Fletcher, C. W., and Torrellas, J. (2020). Cache telepathy: Leveraging shared resource attacks to learn DNN architectures. In *USENIX Security 20*, pages 2003–2020.

Zhu, Y., Cheng, Y., Zhou, H., and Lu, Y. (2021). Hermes attack: Steal DNN models with lossless inference accuracy. In *USENIX Security 21*.