Characterizing AI Model Inference Applications Running in the SGX Environment

Shixiong Jing*, Qinkun Bao*, Pei Wang‡, Xulong Tang†, and Dinghao Wu*
*The Pennsylvania State University
†University of Pittsburgh
‡Independent

***Abstract*—Intel Software Guard Extensions (SGX) is a set of extensions built into Intel CPUs for the trusted computation. It creates a hardware-assisted secure container, within which programs are protected from data leakage and data manipulations by privileged software and hypervisors. With the trend that more and more machine learning based programs are moving to cloud computing, SGX can be used in cloud-based Machine Learning applications to protect user data from malicious privileged programs.**

However, applications running in SGX suffer from several overheads, including frequent context switching, memory page encryption/decryption, and memory page swapping, which significantly degrade the execution efficiency. In this paper, we aim to i) comprehensively explore the execution of general AI applications running on SGX, ii) systematically characterize the data reuses at both page granularity and cacheline granularity, and iii) provide optimization insights for efficient deployment of machine learning based applications on SGX. To the best of our knowledge, our work is the first to study machine learning applications on SGX and explore the potential of data reuses to reduce the runtime overheads in SGX.

***Index Terms*—Cloud Security, Intel SGX, Performance Evaluation, Machine Learning**

I. Introduction

Intel Software Guard Extensions (SGX) is a trusted computing design to solve the secure remote computation problem. SGX provides integrity and confidentiality protection against privileged software (kernel, hypervisor, etc.) by adding additional protection mechanisms in the CPU while maintaining a secure memory region called enclaves. However, the confidentiality and integrity guarantee come at a price. Significant overheads severely degrade some applications' performances. Those overheads include: (i) context switch for OS when EPC misses, (ii) encryption and decryption when pages are moved across EPC and non-EPC, (iii) The updates of integrity tree when EPC hits, etc. According to Meysam's evaluation, the encryption and decryption can lead to a slowdown of more than 20 times than baseline non-secure program on page faults, and context switch will cause a slowdown of about 15 times than non-secure program on page faults [1].

In this research, we aim to explore the performance of various machine learning algorithms inside SGX, analyze the memory access patterns of different machine learning modes, and discuss the possibility to leverage Depp Neural Networks (DNNs) execution characteristics and alleviate the performance degradation.

978-1-7281-7744-1/21/$31.00 ©2021 IEEE

Our Contributions. 1) We characterize the data reuse distance distribution in common deep learning models and some other machine learning models. 2) We discuss the relationship between model structures and memory reuse patterns. 3) We provide insights for future optimizations for DNNs to run inside SGX enclaves with better performance.

II. Background

A. Intel SGX

Intel SGX is an instruction set extension that offers protection for the application and its data. SGX leverages trusted hardware to establish secure containers called enclaves. Computations can be performed inside SGX enclaves while code and data inside the enclaves are isolated and protected from the outside environment. SGX has the potential to be applied in many different services, and researchers are creating new designs in more and more fields to take advantage of this architecture. Such exploration is happening in secure cloud microservices, private web searches, credential protection for synchronization applications, and enclave-based databases [2].

Although SGX is powerful in protecting the integrity and confidentiality of the program, it suffers from efficiency overhead due to the structure of enclaves. Arnautov's work showed that there is a large gap between EPC (Enclave Page Cache) hit and miss latencies, 200 cycles vs. 40,000 cycles [3]. Various other researches, such as Zhao's work [4] and Gjerdrum's work [5], have also corroborated the conclusion that limited memory space inside the enclave causes great performance drop when the number of page swap, amount of memory accessed, or frequency of memory access increases.

B. SGX Memory Management

Intel SGX-enable CPUs reserve a special memory area called Processor Reserved Memory (PRM) during system boot. This region of memory is a subset of DRAM (Dynamic Random Access Memory), but it is protected from other software including the program running in the kernel space [6]. Enclave Page Cache (EPC) is a subset of PRM where SGX Enclave data is stored. The EPC is divided into 4KB pages and a variant of Merkle tree called SGX integrity tree is applied to ensure the integrity of EPC pages. The maximum size of PRM is set in the BIOS settings, with the maximum size of

128 MB in most cases. In those 128 MB, the available size of EPC for enclave programs is around 92 MB.

Although current SGX provides protection for pages in EPC, it still relies on the untrusted operating system to manage the memory pages. Intel SGX Driver for Linux uses the Least Recently Used (LRU) policy for EPC page swapping, though the real implementation is not strictly following the LRU manner. The program inside the enclave still shares the same cache with the outside program to fill the access gap between the processors and the DRAM.

In this paper, we study the performance in terms of memory reuse distance. Memory reuse distance is defined by the number of distinct references that have happened between two references to the same memory location. In particular, a page reuse distance between two accesses of the same page determines whether this page has been swapped out of EPC due to LRU policy, while cacheline reuse distances can reflect the performance of cache and determine whether EPC will be visited.

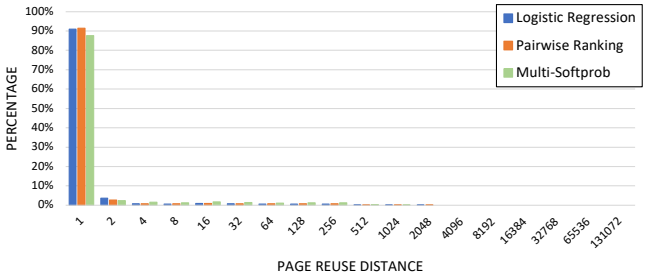

Figure 1: Page reuse distance frequency of GBDT with X-axis showing page reuses with reuse distance x categorized by the value of $\lfloor \log_2 x \rfloor$, and Y-axis showing the percentage among all page reuses.

III. Characterization and Observation

We run 4 machine learning algorithms to evaluate the paging performance and cacheline reuse characteristics for different families of machine learning tools. The models we tested are NIN (A micro-neural network) [7] , Resnet50 (residual network with 50 layers), R-CNN (CNN-based network for object detection and semantic segmentation), and GBDT (gradient boosting decision trees).

These models cover many types of widely applied neural networks. The results from these experiments can reflect other neural networks with similar structures.

A. Experiment Setup

To characterize and quantify the data reuse and reuse distance, we first gather the memory access trace of machine learning algorithms on SGX. we use the simulation mode in SGX to collect the trace because SGX is implemented with protection against all outside software including all dynamic analysis tools. We use Pin toolset to collect the memory trace. It is important to note that, the memory trace is not collected for the whole program, because the overall size of the trace will be too large (100GB for NIN, and over 200 GB for Resnet50). Instead, our Pin tool neglects the memory access trace for early library loading and model preparation.

The experiments were conducted on an Intel Core (TM) i7-7700T CPU at 2.90GHz frequency with 32KB L1 D-cache, 32KB L1 I-cache, 256 KB L2 cache, and 8192 KB L3 cache. We used the Anakin framework and gbdt-rs framework for machine learning applications since they are two of the few frameworks that support SGX.

Anakin is a cross-platform inference engine for neural network architectures. It supports models originally built for Caffe and Tensorflow. Gbdt-rs is a lightweight implementation of the gradient boosting decision tree (GBDT) algorithm. Gbdt-rs is written in Rust and has been optimized to be SGX friendly [8]. In the memory trace collection, Anakin models (Resnet50, NIN, R-CNN) are run with input data filled with constant values for 3 rounds with batch size 50; GBDT with logistic regression is tested with agaricus testing dataset; GBDT with multi-softprob is tested with default dermatology testing data set; GBDT pairwise ranking with the mq2008 testing data set [9].

B. Data Collection

We build a simulator for EPC, simulating the paging behavior under the LRU policy when memory is accessed in the order of the recorded memory trace. We assume that the machine learning application will be the only program that requires EPC memory, thus there will be only one enclave alive throughout the experiments. After the memory page is loaded into the EPC, the CPU accesses the required data to the cache at the cacheline granularity.

In order to take cache influence into consideration, we build a cache simulator on top of the EPC memory simulator. The CPU cache simulator is based on intel-i7-7700 architecture (3-level cache with LRU write-back replacement policy). In the experiments, the simulation of EPC is based on the assumption that about only 96MB of the EPC memory is actually available to the user program, which has been corroborated by various research [10].

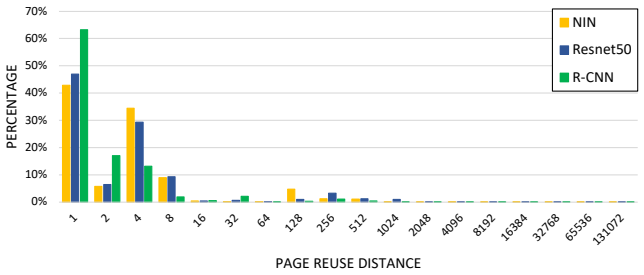

Figure 2: Page reuse distance frequency of NIN, Resnet50, and R-CNN on Anakin with X-axis showing page reuses with reuse distance x categorized by the value of $\lfloor \log_2 x \rfloor$, and Y-axis showing the percentage among all page reuses.

C. Observations and Analysis I: Page Reuse Distance

We first study the reuse distances at the page granularity. Our goal is to investigate whether the limited EPC capacity

is able to capture the massive page reuses without incurring substantial page swapping. The result is shown in terms of frequencies of different range of reuse distance. In figure 1 and 2, page reuses with reuse distance x will be categorized by the value of $\lfloor \log_2 x \rfloor$.

We make the following observations on the result of page reuse distance frequency:

1) Observation 1: Both NIN and Resnet have the second most frequent reuse distance between 2^1 and 2^2. The top 5 frequent page reuse distance categories of NIN and Resnet50 are the same.

2) Analysis 1: Benchmarks with similar structures tend to end up with similar memory access patterns. Both NIN and Resnet are neural networks designed for image classification, and both of them involve the general structure with many convolutional layers at the bottom for feature extraction and some fully connected layers (NIN uses MLPconv layers, which are composed of several small fully connected layers) for scoring. We also observe that the rest of the reuse mainly gathered around 2^6 to 2^{11}. These page reuse would be the reuse of weights of the neurons each time a new piece of data is fed in. The fact that those reuses are skewed to larger reuse distances for Resnet50 than for NIN also corroborates this explanation, because Resnet50 has more layers with more parameters, which naturally leads to a larger reuse distance due to more memory access between the reuse of weights when each time a new piece of data is loaded.

This answers question #3 that similar networks do end up with similar memory access patterns.

3) Observation 2: For all neural networks, page access with extremely low reuse distance (less or equals to 1) occupied over 40% of the page reuse. In terms of portions occupied by low page reuse distance, R-CNN is higher than Resnet, Resnet is higher than NIN.

4) Analysis 2: Convolution layers would lead to high frequencies of low page reuse distance, especially when the channel number is low. Consider the size of one EPC page (4K Byte), for the extreme case of a convolution layer with 3×3 filters on 3 channels, input data used for more than 100 columns in the *im2col* operation can be on the same page. Besides, the weights of the filters are also going to be reused as many times as the size of input approximately.

As for fully connected layers, theoretically, the input data will be used in computation for as many time as the number of output neurons during the matrix multiplication (unless the matrix is very sparse such that the math kernel library skip some computation). However, the weights will be used for computation only once during a single pass. Therefore, the reuse distance for weights is more likely to be large. The size of weights is proportional to the number of input neurons times the number of output neurons, and fully connected layers usually have more than 1000 neurons. As a result, the weight of fully connected layers would contribute a considerable amount of page reuse with large reuse distances.

NIN uses MLPConv layer, which indeed includes several smaller fully connected layers inside the convolution layer; while Resnet uses mostly convolution layers. Given the difference in memory access patterns between convolution layers and fully connected layers, it is not surprising to see that the portion of low page reuse distance of Resnet is slightly higher than NIN.

As for R-CNN, although it does not apply as many convolution layers as Resnet, it generates more than 1000 regional proposals for each input image and feeds all of them to CNN for feature extraction, which leads to a large amount of convolution layer computation. Therefore it is also reasonable for it to have a very large portion of low page reuse distance.

This answers Question #2 about how the structure of neural networks influences memory access patterns.

5) Observation 3: We noticed that for GDBT models, page accesses with extremely low reuse distance (less or equals to 1) occupy around 90% of all page reuses. Page reuses with large reuse distances are very rare.

6) Analysis 3: GBDT model inferences turn out to have very high localities. This might be the effect of computing patterns of random forests. GBDT prediction is achieved by combining the results of many weak decision tree models. The computation of inference can be broken up as multiple rounds of decision tree computations, which are mostly in a linear fashion.

7) Takeaway: Page access rarely comes with large reuse distances, which shows that in most cases, the page processed a long time ago is not likely to be reused.

Page reuse distance between two access of the same page will determine whether this page has been swapped out between the two access. In the case of EPC, it can hold about 25000 pages at the same time, which means that a page must have been swapped out once between two access if the reuse distance of the two access is larger than 25000. By counting the frequencies of all page reuse distances in the memory trace, we can get some idea about the performance of LRU mechanism.

In the chart, most page reuses come with small page reuse distances less than 2^{14}, therefore they will be kept inside the enclave page cache between the two visits and would not cause any page swap. There are few page accesses with reuse distance between 2^{14} and 2^{15}, which map to the situation where pages were swapped out of the memory right before bringing them back in. Very few pages reuse with reuse distances larger than 2^{15} shows that for the given models, there are no frequent page swaps during the inference.

This answers Question #1 on data reuse of various machine learning models.

D. Observations and Analysis II: cacheline Reuse Distance

Memory traces for machine learning models were also analyzed in terms of cacheline reuse distance. We cut out the cache simulation portion in our simulator to generate the result. In figure 3, cacheline reuse with reuse distance x will be categorized by the value of $\lfloor \log_2 x \rfloor$.

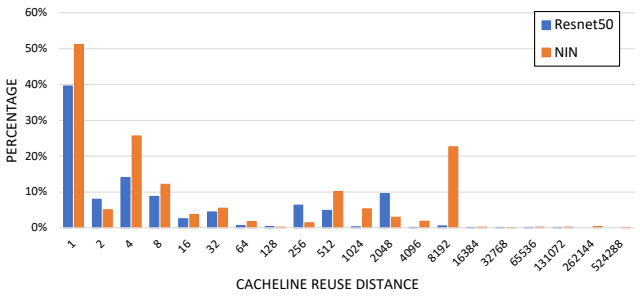

Figure 3: Cacheline reuse distance frequency of NIN and Resnet50 on Anakin, with X-axis showing cacheline reuses with reuse distance x categorized by the value of $\lfloor \log_2 x \rfloor$, and Y-axis showing the percentage among all cacheline reuse.

1) Observation 1: We can see from the graph that at the cacheline level, there are more reuses with medium to large reuse distance($2^5 - 2^{10}$) compared with memory reuse on the page level. In the previous page reuse distance chart, less than 5% of the reuses occur in any category with reuse distance larger than 2^4. In cacheline reuse distance chart, around 10% of cacheline reuse in NIN has distance larger than 2^{12} and more than 20% of cacheline reuses in Resnet50 have reuse distance larger than 2^{14}.

2) Analysis 1: The difference between cacheline reuse distance and page reuse distance matches our expectations for caches. Many of the cacheline visits with short or medium reuse distance might end up as a cache hit, thus will not reflect on the page reuse distance analysis. Besides, some of the cachelines belong to the same page, therefore the distinct cacheline visit when calculating the cacheline reuse distance might not correspond to distinct page visits.

As a result of all the factors mentioned above, some cache blocks visited with medium or large reuse distance will not enjoy a large page reuse distance because of cache hits.

3) Observation 2: Despite more memory reuse with medium distances at cacheline level, the overall trend from page reuse distance chart still exists in the cacheline reuse distance chart. We still have the largest portion of memory reuse with reuse distance of 0 and 1, with the second largest portion of memory reuse around 2^3 and 2^4.

4) Analysis 2: One cacheline has 64 bytes, therefore it is still very likely for several pieces of data accessed sequentially to reside in the same cacheline. Similar to what we have mentioned in observation # 1, while cache hits are not counted in page reuse distance chart, all cacheline reuse will be calculated in the cacheline reuse distance frequency chart, therefore low distance reuse of cacheline can still occupy the largest portion of cacheline reuse.

Matrix multiplication operations also play an important role. In most math kernel libraries, including libraries like MKL and OpenBLAS, the matrix multiplication has been optimized for hardware and cache. In order to reduce the effect of memory access latency, the library usually reorders the computation to increase temporal locality.

5) Observation 3: In previous page reuse distance charts, we observe that there is a short spike at long page reuse distance for Resnet at 2^8 and NIN at 2^9. In the cacheline reuse distance chart, the spike still exists, but at a larger reuse distance and occupying a larger portion.

6) Analysis 3: In the earlier analysis on page reuse distance of NIN and Resnet, we mentioned that the last short spike of page reuse with large reuse distance would be the reuse of model weights when each time a new package of data passed through. The spike in cacheline reuse distance chart is likely to be the same case. The difference between page reuse distance and cacheline reuse distance (4–5 magnitudes for base 2) also matches the difference between the size of cachelines (64 Bytes for the machine we are using) and the size of pages (4k Bytes).

IV. Conclusion

In this paper, we have tested the performance of several machine learning applications inside SGX and explored the factors that have influenced their behaviors in memory access patterns. The observations from the page reuse distance and cacheline reuse distance have shown that the tested machine learning models have high memory locality in general and the current paging policy in EPC performs well.

We would conclude that further optimization research on SGX for machine learning models might want to focus on caches. Although the performance of those tested networks is not likely to be boosted by changing EPC paging policy, larger networks with more frequent large-distance page reuses might benefit from a customized paging policy.

Acknowledgement

The authors sincerely thank all the reviewers for their constructive feedback and suggestions. This work is supported in part by NSF grant #2011146 and startup funding from the University of Pittsburgh.

References

[1] M. Taassori, A. Shafiee, and R. Balasubramonian, "VAULT: Reducing paging overheads in SGX with efficient integrity verification structures," in *ASPLOS*, 2018, pp. 665–678.

[2] C. Priebe, K. Vaswani, and M. Costa, "EnclaveDB: A secure database using SGX," in *IEEE S&P*, 2018.

[3] S. Arnautov, B. Trach *et al.*, "SCONE: Secure Linux containers with Intel SGX," in *OSDI*, 2016.

[4] C. Zhao, D. Saifuding *et al.*, "On the performance of Intel SGX," in *WISA*. IEEE, 2016, pp. 184–187.

[5] A. T. Gjerdrum, R. Pettersen *et al.*, "Performance of trusted computing in cloud infrastructures with Intel SGX," in *CLOSER 2017 - Proceedings of the 7th International Conference on Cloud Computing and Services Science*, 2017, pp. 668–675.

[6] V. Costan and S. Devadas, "Intel SGX explained," 2016, IACR Cryptology ePrint Archive. [Online]. Available: http://eprint.iacr.org/2016/086

[7] M. Lin, Q. Chen, and S. Yan, "Network in network," in *ICLR*, 2014, pp. 1–10.

[8] T. T. Li, T. T. Li *et al.*, "Poster: gbdt-rs: Fast and trustworthy gradient boosting decision tree," in *Posters In 2019 IEEE S&P*, 2019, pp. 2–3.

[9] J. Schlimmer, "Uci repository of machine learning databases: Agaricus-lepiota dataset," 1987. [Online]. Available: http://www.grappa.univ-lille3.fr/ torre/Recherche/Experiments/Datasets/#agaricus-lepiota

[10] A. Mandal, J. C. Mitchell *et al.*, "Data oblivious genome variants search on Intel SGX," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, 2018, pp. 296–310.