THOR: A Non-Speculative Value Dependent Timing Side Channel Attack Exploiting Intel AMX

Farshad Dizani, Azam Ghanbari, Joshua Kalyanapu, Darsh Asher, and Samira Mirbagher Ajorpaz

***Abstract*—The rise of on-chip accelerators signifies a major shift in computing, driven by the growing demands of artificial intelligence (AI) and specialized applications. These accelerators have gained popularity due to their ability to substantially boost performance, cut energy usage, lower total cost of ownership (TCO), and promote sustainability. Intel's Advanced Matrix Extensions (AMX) is one such on-chip accelerator, specifically designed for handling tasks involving large matrix multiplications commonly used in machine learning (ML) models, image processing, and other computational-heavy operations. In this paper, we introduce a novel value-dependent timing side-channel vulnerability in Intel AMX. By exploiting this weakness, we demonstrate a software-based, value-dependent timing side-channel attack capable of inferring the sparsity of neural network weights without requiring any knowledge of the confidence score, privileged access or physical proximity. Our attack method can fully recover the sparsity of weights assigned to 64 input elements within 50 minutes, which is 631% faster than the maximum leakage rate achieved in the Hertzbleed attack.**

***Index Terms*—Intel advanced matrix extensions (AMX), value-dependent timing side-channel, ML privacy, NN sparsity.**

I. INTRODUCTION

PRIVACY of machine learning models deployed on Machine Learning as a Service (MLaaS) platforms can be compromised by adversaries to extract sensitive details, including model architecture and hyperparameters [15]. However, this paper shows that the tight integration of on-core AI accelerators like Intel AMX [5] causes AMX performance to depend upon operand value which works as proxy, enabling timer attacks on ML models thus introducing a timing side channel that bypasses conventional defenses like speculative execution mitigations or cache isolation and challenges the architectural and data privacy of ML.

Cache attacks exploit timing variations between accessing or flushing data in cache levels or DRAM [4]. Spectre [8] and Meltdown [11] exploit speculative execution and exception handling. MDS Attacks [1], [14] target microarchitectural buffers. Defenses include cache partitioning [7], Invisispec [19], privileged access controls (e.g., RAPL restrictions) [10], SMT disabling and other patches with performance overhead. However, no work has been done on the security evaluation of newly built Intel AMX for AI applications.

Black-box attacks extract model functionality or hyperparameters with minimal access but require confidence scores and are mitigated by various methods like limiting the query rate, obfuscating the input data by masking, rounding confidence, or using homomorphic encryption. Early physical attacks, such as IKWYS [18] used power or EM side channels to infer neural network (NN) parameters but required physical access. [20] leverages cache timing to leak NN architecture but is reliant on particular ML libraries and co-location. We confirmed that vulnerabilities on NNs using floating-point timing [3] have been completely mitigated in the Intel AMX design and thus are no longer available to the attacker.

In this paper, we demonstrate the feasibility of exploiting Intel AMX to leak sensitive information in NN, such as sparsity of it's weights. Inferring sparsity could inadvertently reveal sensitive data patterns. Understanding the sparsity pattern might provide insights into the underlying data distributions, potentially breaching data privacy. Additionally, knowledge of sparsity patterns could enable attackers to craft more effective adversarial attacks. By understanding the structure of a model, an attacker might design inputs that strategically exploit sparsity, leading to degraded model performance or incorrect predictions. Our approach is notable for not requiring physical access or a shared cache, unlike previous works that often necessitating physical access for power and electromagnetic measurements or shared cache manipulation via known cache side channels [20], lowering the bar for potential attackers. We discover that the execution times of the Intel AMX multiplications vary depending on the operand's value. This phenomenon is different from the observed Hertzbleed [12], [17], which is a side-channel attack that leverages dynamic voltage and frequency scaling (DVFS) on modern x86 processors to transform power fluctuations into timing attacks, extractable without direct power measurement. Hertzbleed exploits CPU frequency variations, linked to data-dependent power consumption. Our attack continues to function even with the Intel Turbo Boost (DVFS) feature turned off, suggesting a novel root cause specific to the Intel AMX design compared to Hertzbleed, which relies on the DVFS feature being enabled and obsolete cryptographic libraries. Our attack also bypasses TEE based defenses in contrary to Rowhammer-based attacks like DeepSteal [13] that leak partial weights but can be protected by TEE-based defenses.

We successfully leaked zeros in 64 hidden weights with 100% accuracy within 50 minutes of observation without access to the NN output value, confidence score or shared cache bypassing state of the art defenses such as obfuscation, masking or rounding confidence values of NN and cache defenses. Our proof of concept (PoC) attack (THOR) represents the first step toward a new direction in ML privacy attacks. It is an attack on a single-layer NN but its significance is that, unlike prior approaches that rely on confidence scores, logits, physical access, or higher privilege, THOR leverages timing information alone to infer critical and private parameters like NN's weight sparsity patterns and distributions.

This paper presents the following contributions:

- It characterized and reverse-engineered the Intel AMX, uncovering a novel type of data-dependent timing side-channel vulnerability within the Intel AMX accelerator.
- It demonstrates the techniques necessary to exploit this newly discovered timing side-channel for launching attacks on NNs. Specifically, it illustrates how to determine the sparsity of NN weights by measuring execution time. We show that the THOR leakage rate is 631% and 1,493% higher than the maximum

Received 15 January 2025; revised 19 February 2025; accepted 20 February 2025. Date of publication 27 February 2025; date of current version 19 March 2025. *(Corresponding author: Farshad Dizani.)*

The authors are with the Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC 27606 USA (e-mail: fdizani@ncsu.edu; aghanba2@ncsu.edu; smirbag@ncsu.edu).

Digital Object Identifier 10.1109/LCA.2025.3544989

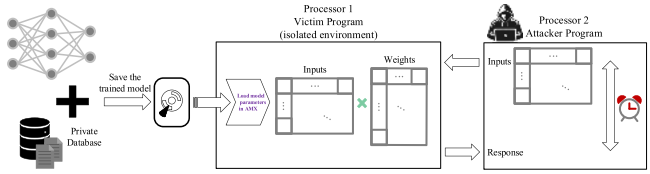

Fig. 1. THOR threat model.

leakage rates in the Hertzbleed and Collide+Power [9] attacks, respectively.

- We discuss defenses, propose a counter-measurement and measure its overhead on power consumption (2.59% to 12.33%).

Responsible disclosure: We reported our attack and findings to Intel in May 2024 and Intel confirmed our findings.

II. THREAT MODEL

The attacker and victim are two distinct processes running on the same server. These processes are entirely isolated in terms of address space and can be executed on different CPU cores. As shown in Fig. 1, the victim process is an API that runs a NN designed for specific tasks and utilizes Intel AMX for matrix computations. The attacker process, which operates as a non-privileged user, does not have direct access to the model's parameters but interacts with the model via the inference API. The attacker sends crafted inputs to the inference API and measures the timing of the victim process's responses. By examining timing variations, the attacker can gain insights into the NN's internal parameters by taking advantage of the timing differences caused by Intel AMX computations. We assume cache defenses are deployed or there is no shared cache, thus channels such as [20] is not available to the adversary. The ML model is a single layer NN protected by obfuscated the confidence score and the output by methods such as adding noise or rounding confidence scores [2]. The victim may leverage TEEs, Intel SGX, to execute the NN computations within isolated enclaves.

III. THOR

Our investigations utilized a server running Red Hat Enterprise Linux release 9.4 with Linux kernel version 5.14, powered by an Intel Xeon Gold 5420+ from the Sapphire Rapids microarchitecture. The tests were carried out with a range of configurations, including the autonomous (hardware-only) and cooperative P-state control modes, under minimal power, maximum performance, and efficiency-focused platform power settings. Turbo Boost was tested in both enabled and disabled states. Prefetchers were toggled between enabled and disabled states across experiments to assess their impact on noise and timing variations. The server configurations included the UEFI firmware versions SRV650-v3-3.14 (May 2024) and SRV650-v3-3.20 (June 2024), with the latter mitigating certain Spectre variants but all above settings remains vulnerable to our attack.

A. Reverse Engineering

Intel AMX is an on-core accelerator introduced with the 4th Gen Intel Xeon Scalable processors, specifically engineered to boost matrix computation efficiency. Tile Matrix Multiply unit (TMUL) is the accelerator engine for executing multiplication calculations. Intel AMX can perform 512 and 1024 multiplications/additions per cycle for BF16 and INT8, respectively [6].

We studied the impact of operand sparsity on TMUL operations. By varying the number of zero values in the Tile operands during

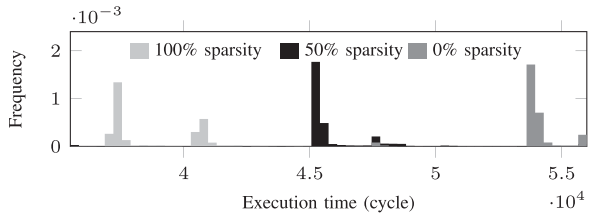

Fig. 2. Execution time with varying operand sparsity.

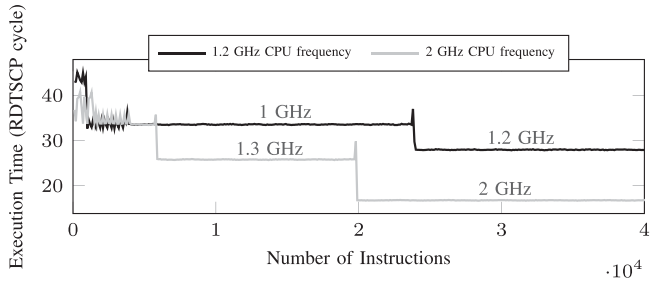

Fig. 3. Illustration of AMX frequency adjustment.

multiplication, we observed that increasing the number of zeros accelerates the AMX execution. This effect is illustrated in Fig. 2 , which presents three distinct distributions showing the sparsity levels of the operand: 100% sparsity, 50% sparsity, and 0% sparsity. The results indicate that a greater sparsity of operands in the tile operands leads to faster execution of the TMUL. The program with 1,000 AMX multiplications averages 54,005 cycles when operands have 0% sparsity. Increasing sparsity to 50% and 100% reduces the average execution time to 45,953 and 38,747 cycles, respectively. In this case, 50% and 100% more sparsity make the execution 17.5% and 39.4% faster, respectively.

This discovery uncovers a new value-dependent timing side channel that reveals the sparsity of Tile operands in Intel AMX. Our results suggest the presence of a zero-skipping-like mechanism for both BF16 and INT8 operands in Intel AMX design for acceleration and power saving purposes.

Since the operand value timing dependence persists even with Turbo Boost disabled and the CPU frequency fixed, we hypothesized that AMX operates under a separate clock domain. To estimate its internal independent frequency (assuming there is one), we analyzed its performance and observed that the execution times initially fluctuate through intermediate levels before stabilizing. We measured the stabilized execution time of AMX multiplications at each available CPU frequency. By comparing these times, we found that interestingly the intermediate levels correspond precisely to some frequencies in our dataset. This suggests that AMX transitions through intermediate frequencies before stabilizing to match the CPU frequency. For example, at a fixed CPU frequency of 1.2 GHz, AMX stabilizes at 1 GHz before aligning with 1.2 GHz. Similarly, at a fixed CPU frequency of 2 GHz, AMX progresses through 1 GHz and 1.3 GHz before reaching 2 GHz (see Fig. 3). These consistent transitions across tests strongly suggest that AMX may operate under an independent clock, autonomously adjusting its frequency to match the CPU frequency. Interestingly, the latency of this adjustment also depends on operand values.

B. Attack Building Blocks

The THOR attack begins with strategically crafted input vectors, where the adversary measures `TMUL` execution time under different

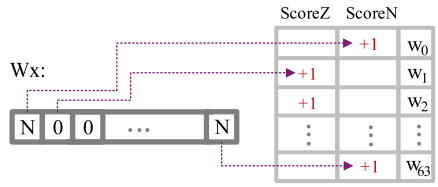

Fig. 4. Scoring update process diagram: $ScoreZ$ and $ScoreN$ increment based on W_x values.

operand conditions. Multiple measurements mitigate inherent noise, yielding a composite timing value.

(1) Setting Threshold: By comparing execution times with inputs of full zeros and full non-zeros, we establish a differential threshold (Thr). This threshold enables the selective acceptance of randomly generated input vectors for scoring. The essence of the attack involves two key steps: (1) continuously generating random input vectors throughout the attack and (2) using the execution time associated with each vector to accumulate evidence that helps refine the estimates of the weight scores. Suppose a weight element is non-zero; then, the execution time will be longer if the corresponding input element is non-zero compared to when it is zero. Conversely, if a weight element is zero, the execution time remains unchanged whether the corresponding input element is zero or not. Based on these assumptions, if a generated input vector has a higher number of correct non-zeros according to the unknown weights, it is more likely to be accepted by this algorithm, thereby impacting the scoring tables.

(2) Execution Time Analysis and Vector Selection: The attacker measures execution times for two scenarios: one for the randomly generated vector (W_s) and another for its inverted version (W_r) (in W_r, all zeros in W_s become non-zeros, and all non-zeros become zeros). We predict that the vector holding more correct non-zeros according to the unknown weights will require a longer execution time. If the difference in execution times exceeds the threshold, either W_s or W_r—whichever records the longer time—will be selected (W_x) for updating score vectors ($ScoreZ$ and $ScoreN$). After selecting W_x, a scoring method updates the score vectors according to W_x entries. This involves iterating through each element of W_x from 0 to 63, incrementing the corresponding entry in $ScoreZ$ by one if the element is zero and in $ScoreN$ if the element is non-zero (see Fig. 4).

(3) Inferring Weights from Scoring Vectors: Following the attack period, the attacker analyzes the ScoreZ and ScoreN vectors. A weighted element marked as zero would be expected to yield similar scores in both $ScoreZ$ and $ScoreN$ vectors. However, for a weight element marked as non-zero, a higher score is expected in the $ScoreN$ vector compared to the $ScoreZ$. By dividing each element of $ScoreN$ by $ScoreZ$, the attacker can infer the zero weights. If the ratio is close to 1, it suggests that the weight scores are equivalent; thus, the weight is likely zero. A ratio significantly above 1 indicates that the weight is likely non-zero.

IV. Security Evaluation

A. Leakage Rate

Fig. 5 shows success rates increasing with longer attack durations, from 60% at 5 minutes to a complete 100% success rate at 50 minutes (0.02 bit/s) and beyond.

Fig. 6 shows THOR leakage rate of 76.8 bits/hour compared to other related side channels. While Platypus has a higher leakage rate, it is patched and unavailable to non-privileged users. While THOR does not require access to RAPL and is not limited to SGX, it exhibits a higher leakage rate compared to Hertzbleed and Collide+Power attacks. THOR's leakage rate is 1,493% faster than Collide+Power (MDS) (76.8 versus 4.82 bits/hour), and 631% faster than Hertzbleed (10.5 bits/hour), which exploits outdated libraries now replaced by more secure cryptographic implementations. Collide+Power attacks, which leveraged Microarchitectural Data Sampling (MDS) and Meltdown, have been patched on modern Intel Xeon servers. Similarly, Platypus, which depends on the RAPL interface, is mitigated by restricting access to energy and power reporting. With these mitigations, THOR remains one of the few viable microarchitectural attack methods available to low-privilege attackers, posing a unique and unpatched threat to AI applications, even in environments where traditional microarchitecture side channels have been neutralized.

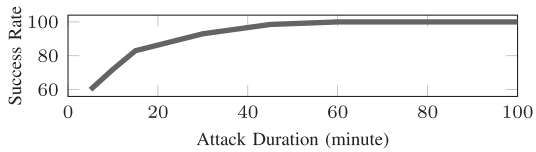

Fig. 5. Impact of attack duration on the success rate of weight determination.

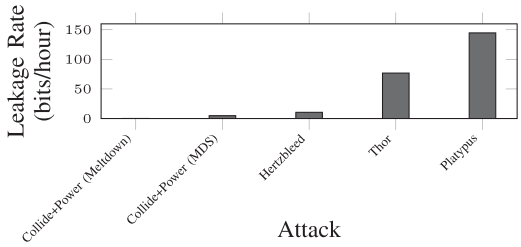

Fig. 6. THOR leakage rate comparison.

B. Counter-Measurement

THOR relies only on precise *timing* measurements. Thus, it cannot be mitigated by defenses that alter NN classification outputs, such as adding noise or rounding confidence scores [2]. Trusted Execution Environment (TEE)-based defenses, like those performing non-linear computations inside the TEE while offloading linear computations to untrusted sources [16], are also inadequate, as the value dependencies of Intel AMX timing that THOR relies on are unaffected by TEE environments like Intel SGX. AI workloads demand high speed and efficiency, prompting AI libraries to prioritize performance optimizations. As a result, known constant-time programming techniques are often unsuitable as a defense. For example, masking can help protect weights from leaking but introduces extra computational overhead, increasing both power consumption and execution time.

A more effective strategy involves introducing response randomness by delaying execution, which disrupts the timing signals relied upon by THOR. However, this approach adds latency to AI applications. Limiting the query rate of the model could also slow these attacks, but at the cost of reducing system responsiveness. Extending detection mechanisms, to identify malicious patterns in power and performance characteristics offers a promising defense against THOR. Lastly, employing homomorphic encryption could provide robust protection but comes with substantial computational overhead and performance costs, making it less practical for high-speed AI applications.

One mitigation which can be applied through a micro-code update or a software patch is to keep the AMX unit moderately in the Warm State at all times or at least during Intel SGX execution to protect TEEs computation against THOR. This approach is effective because

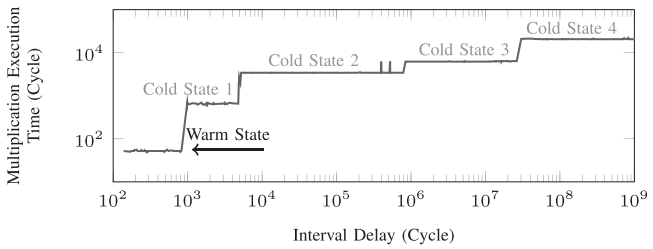

Fig. 7. Performance states of TMUL and the secure recommended Intel AMX operational state (Warm).

we observed that timing differences dependent on zero values are only significantly measurable when the Intel AMX is in a Cold State. Warm and Cold States introduced here come from an interesting observation in which we measured the time to execute a single AMX multiplication instruction while varying the intervals between consecutive executions. By adjusting the length of these intervals, we identified five distinct execution times, classifying them into performance states. The shortest execution time with the lowest intervals was labeled as the Warm State, while the longer execution times with higher intervals were classified as Cold States. These performance states are shown in Fig. 7. However, this mitigation comes with trade-offs in power management and execution speed, as system power limits could be more easily reached, leading to unnecessary throttling of the AMX unit. We measured the power overhead of such defense for THOR and found that depending on the cold versus warm stage, the overhead ranges from 2.59% to 12.33%. Although this secure design requires more power consumption, it is faster as it keeps the Intel AMX in the highest performance state at all time; this is in contrary to other secure designs for different microarchitectural attacks which almost always incurred a high performance overhead.

Thus, future research must prioritize the development of effective mechanisms to mitigate the proposed threat vector introduced by AMX and similar technologies while addressing the secure design's performance and power consumption.

REFERENCES

[1] C. Canella et al., "Fallout: Leaking data on meltdown-resistant CPUs," in *Proc. 2019 ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 769–784.

[2] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 1322–1333.

[3] C. Gongye, Y. Fei, and T. Wahl, "Reverse-engineering deep neural networks using floating-point timing side-channels," in *Proc. 57th ACM/IEEE Des. Automat. Conf.*, 2020, pp. 1–6.

[4] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+flush: A fast and stealthy cache attack," in *Proc. Int. Conf. Detection Intrusions Malware Vulnerability Assessment*, Springer, 2016, pp. 279–299.

[5] Intel, "Advanced matrix extensions (AMX) for AI acceleration," Intel Corporation, 2023. [Online]. Available: https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/advanced-matrix-extensions/ai-solution-brief.html

[6] Intel, "Intel 64 and IA-32 architectures optimization reference manual," 2024. [Online]. Available: https://www.intel.com/content/www/us/en/content-details/814198/intel-64-and-ia-32-architectures-optimization-reference-manual-volume-1.html

[7] V. Kiriansky, I. A. Lebedev, S. P. Amarasinghe, S. Devadas, and J. S. Emer, "DAWG: A defense against cache timing attacks in speculative execution processors," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2018, pp. 974–987.

[8] P. Kocher et al., "Spectre attacks: Exploiting speculative execution," in *Proc. 2019 IEEE Symp. Secur. Privacy*, 2019, pp. 1–19.

[9] A. Kogler et al., "Collide+Power: Leaking inaccessible data with software-based power side channels," in *Proc. 32nd USENIX Secur. Symp.*, 2023, pp. 7285–7302.

[10] M. Lipp et al., "PLATYPUS: Software-based power side-channel attacks on x86," in *Proc. 2021 IEEE Symp. Secur. Privacy*, 2021, pp. 355–371.

[11] M. Lipp et al., "Meltdown: Reading kernel memory from user space," *Commun. ACM*, vol. 63, no. 6, pp. 46–56, 2020.

[12] C. Liu, A. Chakraborty, N. Chawla, and N. Roggel, "Frequency throttling side-channel attack," in *Proc. 2022 ACM SIGSAC Conf. Comput. Commun. Secur.*, 2022, pp. 1977–1991.

[13] A. S. Rakin, M. H. I. Chowdhuryy, F. Yao, and D. Fan, "DeepSteal: Advanced model extractions leveraging efficient weight stealing in memories," in *Proc. 2022 IEEE Symp. Secur. Privacy*, 2022, pp. 1157–1174.

[14] M. Schwarz et al., "ZombieLoad: Cross-privilege-boundary data sampling," in *Proc. 2019 ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 753–768.

[15] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *Proc. 25th USENIX Conf. Secur. Symp.*, 2016, pp. 601–618.

[16] F. Tramèr and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," 2019. [Online]. Available: https://arxiv.org/abs/1806.03287

[17] Y. Wang, R. Paccagnella, E. T. He, H. Shacham, C. W. Fletcher, and D. Kohlbrenner, "Hertzbleed: Turning power side-channel attacks into remote timing attacks on x86," in *Proc. 31st USENIX Secur. Symp.*, 2022, pp. 679–697.

[18] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. Xu, "I know what you see: Power side-channel attack on convolutional neural network accelerators," in *Proc. 34th Annu. Comput. Secur. Appl. Conf.*, 2018, pp. 393–406.

[19] M. Yan, J. Choi, D. Skarlatos, A. Morrison, C. Fletcher, and J. Torrellas, "InvisiSpec: Making speculative execution invisible in the cache hierarchy," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2018, pp. 428–441.

[20] M. Yan, C. W. Fletcher, and J. Torrellas, "Cache telepathy: Leveraging shared resource attacks to learn DNN architectures," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 2003–2020.