Exploiting Intel AMX Power Gating

Joshua Kalyanapu, Farshad Dizani, Azam Ghanbari, Darsh Asher, *Graduate Student Member, IEEE*, and Samira Mirbagher Ajorpaz

***Abstract*—We identify a novel vulnerability in Intel AMX's dynamic power performance scaling, enabling NETLOKI, a stealthy and high-performance remote speculative attack that bypasses traditional cache defenses and leaks arbitrary addresses over a realistic network where other attacks fail. NETLOKI shows a 34,900% improvement in leakage rate over NetSpectre. We show that NETLOKI evades detection by three state-of-the-art microarchitectural attack detectors (EVAX, PerSpectron, RHMD) and requires a 20,000x reduction in the system's timer resolution (10 us) than the standard 0.5 ns hardware timer to be mitigated via timer coarsening. Finally, we analyze the root cause of the leakage and propose an effective defense. We show that the mitigation increases CPU power consumption by 12.33%.**

***Index Terms*—Intel advanced matrix extensions (AMX), side channel, covert channel, machine learning accelerator.**

I. INTRODUCTION

SINCE the discoveries of Spectre [1] and Meltdown [2], many available microarchitectural covert channels have been mitigated by some hardware or software improvement such as cache partitioning [3] which defeats cache-set based attacks like Prime+Probe [4] and Collide+Power [5], Invisispec [6] which defeats shared-memory based attacks like Flush+Reload [7] and Flush+Flush [8], privileging access to RAPL and adding white noise to power consumption measurements which mitigates Platypus [9] disabling simultaneous multithreading (SMT) which mitigates Binoculars [10], TLBLeed [11] and PortSmash [12]; and disabling TurboBoost which mitigates Hertzbleed [13]. Others [14] are vastly mitigated by advanced microarchitectural attack malware detectors [15], [16], [17] and many are mitigated by reducing the resolution of timers available in CPUs [18] or are only applicable to isolated and de-noised environment [13], [19]. We introduce a novel technique that overcomes these limitations.

We investigated the performance of Intel Advanced Matrix Extensions (AMX) and discovered distinct performance stages based on whether the AMX is in a warmed-up or cooled-down state. Intel AMX is an on-core AI accelerator introduced in the 4th Generation Intel Xeon Scalable Processors tailored for matrix computations. It resides within the execution pipeline itself, eliminating the significant offloading cost incurred in CPU-GPU setups [20]. We measured how long it takes to execute a single AMX matrix multiplication instruction while varying the intervals between consecutive executions. In Fig. 1, we observe that the latency of a `TDPBSSD` (signed-signed 8-bit integer matrix multiplication) differs depending on the time since the AMX unit was last used, i.e. AMX *utilization*. By changing the length of these intervals, we observed five distinct *performance stages* which correspond to a 50, 600, 6000, 9000, and 20,000 cycle latency to perform a `TDPBSSD`. We

Received 28 February 2025; accepted 20 March 2025. Date of publication 26 March 2025; date of current version 17 April 2025. *(Corresponding author: Samira Mirbagher Ajorpaz.)*

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

Digital Object Identifier 10.1109/LCA.2025.3555183

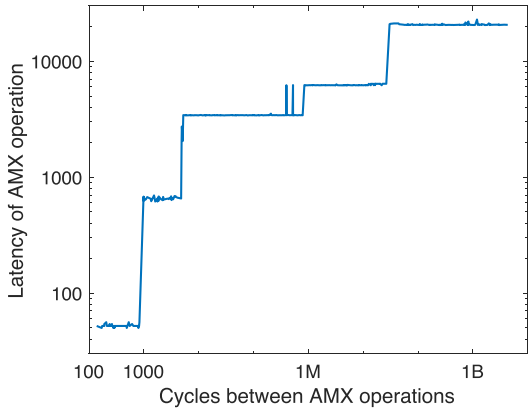

Fig. 1. The five sleep stages of Intel AMX.

classified these into performance states, with the shortest execution time labeled as the "Warm State", the longer execution times are referred to as "Cold States 1–4" with Cold State 4 having the longest execution time. A single execution of any AMX matrix multiplication instruction is sufficient to bring AMX into the warm state. Fig. 1 visually represents these performance states. Five distinct performance stages implies a total of $\binom{5}{2} = 10$ possible covert channels, or *variants*, that can be exploited as microarchitectural covert or transmission channels.

Apart from exploiting the very large timing difference created by the AMX power-performance scaling, another interesting feature of this covert channel is that instead of actively running instructions to flush or trash the caches or TLB, we can bring the microarchitecture to a known state by simply issuing no AMX-related instructions for a period of time. This significantly reduces active attack footprints from a detectability perspective. We show that state-of-the-art ML-based microarchitectural malware detectors [15], [16], [17] that successfully detect other powerful covert channels [10], [14], [19] and the state-of-the-art magnifiers [18], [21], fail to detect NETLOKI despite being trained on NETLOKI samples. This paper focuses on exploiting this characterization as a remote side channel attack to showcase the unique threat landscape that power optimizations can introduce under the most challenging environments for microarchitectural side-channel attacks.

Our attack's threat model is similar to NetSpectre [19] but is performed on a realistic network. A *real network* is a production environment with uncontrolled traffic from users, services, and devices, introducing congestion, jitter, and delays. Examples include university campuses, corporate infrastructures, and public clouds. Unlike lab setups, real networks have non-deterministic packet delivery due to router congestion, multi-hop paths, and traffic and firewalls, which further disrupt microarchitectural side-channel timing measurements.

NETLOKI is a remote attack that allows an attacker to steal data from a computer over a network without an adversary process running on the victim's machine. It builds on Spectre [1], but unlike traditional Spectre attacks that require running attacker-controlled code on the target

```
if (x < length)
    if (array[x] > y)
        TDPBSSD %tmm0, %tmm1, %tmm2
```

Fig. 2. Gadget used in AMX Remote Spectre.

system, NETLOKI works entirely through network requests assuming that a NETLOKI gadget is available in a network-exposed software on the victim machine similar to Netspectre [19] and Hertzbleed [13]. The attacker sends a sequence of network packets to trick the victim's CPU into speculatively accessing sensitive data. By measuring how long the victim takes to respond, the attacker can infer secret values. NETLOKI exploits novel power states in Intel AMX without relying on the typical requirements of other power-based software microarchitectural attacks. Unlike Hertzbleed [13], it does not require DVFS. It does not utilize low power limits, unlike [22], nor does it require privileged access to RAPL as in Platypus [9]. Unlike cache-, TLB-, or MMU-based attacks, NETLOKI operates independently of shared memory resources. Unlike Binoculars [10] it does not rely on SMT, rendering SMT disabling ineffective against it. NETLOKI functions across all possible frequencies and remains viable even if these features are disabled or fixed/locked. NETLOKI does not require any physical access to the device.

Contributions. This paper makes the following contributions:

- We present a timing side-channel attack that exploits a novel vulnerability in Intel AMX to leak arbitrary addresses over a real network with 34,900% improvement in leakage rate over NetSpectre [19].
- We show that NetLoki evades three of the state-of-the-art microarchitectural detectors [15], [16], [17] by leaking a full character before any positive flag and bypasses timer-coarsening defenses [23], requiring a 20,000× coarser timer (instead of 200× for NetSpectre) than the current hardware timer to mitigate the attack.
- We analyze the root cause of the leakage, propose a new defense, and show that mitigation increases CPU power consumption up to 12.33%.

Responsible Disclosure: We reported our attacks and findings to Intel in multiple stages from May 2023 to May 2024. Intel confirmed our findings and completed a thorough security investigation. In June 2024, Lenovo released a UEFI update, Version 3.20, Build ID ESE126H [Critical], which inadvertently mitigates the most critical NetLoki variants particularly those exploiting AMX performance stages 3, 4, or 5.

II. NETLOKI ATTACK

The NetLoki attack operates entirely through network requests, requiring no direct attacker-owned/controlled code execution on the victim machine. It relies on two key gadgets which are assumed to exist in the source code (as in the Spectre and NetSpectre threat models): (1) Leak Gadget: A flaw in the victim's network-exposed software that allows speculative execution to access out-of-bounds memory. (2) Covert Channel: Running Intel AMX instructions to transmit stolen data back to the attacker by subtly changing the CPU's internal AMX state depending on the secret bit's value which affects response times over the network. The main attack steps are:

(I) RESET:The attacker sends nothing to the server for a period of about 20 ms, allowing the AMX unit to fully revert to power state 4.

(II) Speculative Execution and Input-dependent AMX Utilization: The attacker sends a packet whose contents will induce the server software to speculatively access an out-of-bounds values. When the victim processes the packet, a Spectre-like gadget shown in Listing 2 present in a network-exposed function speculatively executes an AMX instruction conditionally on a secret value. The code fragment begins with a bounds check on `x`, a best practice for secure software. A packet with `x` out of bounds is then sent where `array[x]` contains a secret value in the target's memory. The branch predictor, still assuming the bounds check to be true, speculatively executes the memory access. If the referenced element `x` is larger than the attacker-controlled value `y`, the instruction is executed, causing the AMX unit to power on. This power-up also occurs if the branch predictor mispredicts the bounds check, leading to speculative execution of the AMX instruction. Only the fact that an AMX instruction was executed is used to transmit the secret bit of information through the covert channel.

(III) Response Transmission and Network Timing Measurement: The attacker sends another packet which will induce execution of a nonspeculative AMX instruction. The victim then sends a response packet back to the attacker. Since prior (non)execution of AMX in step 2 will affect the execution time of this nonspeculative AMX instruction, the attacker can infer secret bits remotely by measuring latency of the victim's response.

With these building blocks, we build the first pure-AMX covert channel and the first AMX-based remote covert channel. Our proof-of-concept NETLOKI implementation leaks arbitrary bits from the victim by specifying an out-of-bounds index and comparing it with an input value. NETLOKI leaks arbitrary process memory, including cryptographic keys, passwords, and other sensitive data. If the Spectre gadget accesses privileged memory, it extends to OS-level secrets, exposing kernel and system data. Backdoored TensorFlow, PyTorch, Intel MKL, or OpenBLAS could introduce AMX-based speculative covert channels. Supply chain attacks may embed AMX leakage into ML frameworks or network-facing software. NETLOKI also leaks within Intel SGX, raising concerns as TEEs increasingly secure ML models [24]. Unlike traditional side channels, its passive reset phase allows AMX to self-reset, blending into AI/ML workloads. Malware can exploit AMX timing variations in AI pipelines. Cloud or HPC insiders can inject AMX operations for remote data exfiltration. If AMX executes speculatively in kernel modules, NETLOKI may leak kernel memory remotely. Our PoC targets process memory but highlights broader risks. NETLOKI's impact grows with the discovery of malicious AMX gadgets, similar to NetSpectre's reliance on AVX speculative access. While no AMX leakage gadgets has been found in the wild yet, AMX is new (two years old). As AI and HPC workloads increasingly rely on AMX-optimized libraries, the attack surface will expand.

Our investigations utilized a server as a victim running Red Hat Enterprise Linux 9.4 with kernel 5.14, powered by an Intel Xeon Gold 5420+ CPU from the Sapphire Rapids microarchitecture. The network we used is a production network with average daily traffic of tens of terabytes, employing no network isolation. The attacker/client is a Skylake desktop.

Our PoC NETLOKI leaks arbitrary data from the victim's memory through a 1-hop Ethernet connection at 0.07 bps (1 b/15 seconds), which is 34,900% higher than our observed speed of 0.0002 bps (1 b/5000 seconds) on the same network for Netspectre, making it a vastly more realistic attack.

When tested on a real production network, we noticed that the state-of-the-art remote speculative attack NetSpectre [19] fails in more than half of the bits while trying to leak a single byte as shown in Fig. 3. Each row is a distribution of response times from the server for the attacked bit. Dots indicate the means of each timing measurement distribution observed by each attack. The vertical line is the dividing line between a bit classified as 0 and 1 by each attack. If the dot is on the left it means the attack assumes the value is 1, or 0 otherwise. NETLOKI

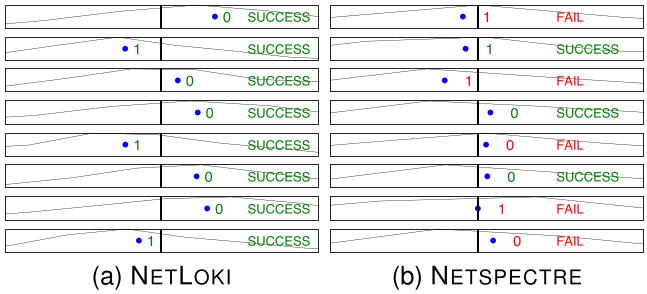

Fig. 3. Comparison of leaking the byte `I` = <01001001>.

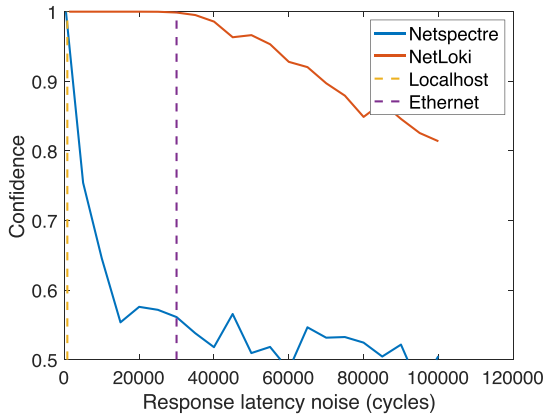

Fig. 4. Noise resilience.

successfully leaks the target secret character. Our results show that NETLOKI achieves a particularly low repetition requirement for 99% confidence, needing only 500 trials representing a 1400x improvement over Netspectre (700,000 trials) in stealthiness.

Fig. 4 illustrates the noise resilience of NETLOKI versus Netspectre for increasing network response time variance. In order to increase the noise in a controlled manner, we applied additive white Gaussian noise of increasing level to latency measurements to simulate noisier networks and plotted attack confidence versus the effective noise of the network. The dashed vertical lines indicate the noise levels in a localhost setting (yellow) and a 1-hop network in our production environment (purple), showing that NETLOKI remains effective under real-world conditions. As network noise increases, Netspectre experiences a sharp decline in accuracy, approaching 0% almost immediately. In contrast, NETLOKI maintains full resilience up to our measured 1-hop connection variance, significantly outperforming the state of the art. Note that our measured 1-hop environment had $\sigma \approx 30,000$ cycles.

One commonly implemented defense is timer coarsening [23]. We make the observation that setting the timer period to the timing differences we exploit will render the states indistinguishable mitigating the covert channel. Therefore, setting the timer period to 10 us (20,000 cycles) representing a coarsening factor of 20,000x (instead of 200x for NetSpectre) over the Sapphire Rapids Time Stamp Counter i.e., TSC (0.5 ns) is needed for mitigation of NETLOKI.

Modern ML based attack detectors such as [15], [16], [17] exploit repetitive attack patterns such excessive cache evictions, high branch mispredictions, or irregular memory accesses to identify attacks. Prior work overlooks the impact of these advanced malware detection mechanisms, assuming an outdated threat model that allows attackers to execute malicious payloads repeatedly without being flagged. To evaluate the detectability of NETLOKI, we compared the detection accuracy using the software versions of state-of-the-art microarchitectural attack

TABLE I
DETECTION ACCURACY

Attack	EVAX	PerSpectron	RHMD
NETLOKI	10%	9%	6%
Microscope	80%	78%	63%
Flush+Flush	99%	87%	72%
Binoculars	98%	97%	85%
NetSpectre	97%	95%	94%
HackyRacers	100%	98%	90%

detectors: EVAX [17], RHMD [16] and PerSpectron [15]. We evaluated the magnifiers Microscope [21] and HackyRacers [18] and the attacks Binoculars [10], Flush+Flush [8] and NetSpectre. The results are shown in Table I.

Despite retraining EVAX, RHMD and PerSpectron on 100 million collected hardware performance counters (HPCs) from NETLOKI samples, all three detectors fail to detect the attack before leakage. This level of stealth provided by NETLOKI is due to the very few repetitions required to achieve high confidence and more importantly, NETLOKI does not require explicit cache flushing (in contrast to [7]) or TLB thrashing (in contrast to [10]) to maintain its microarchitectural state and thus behaves significantly less maliciously in contrast to [18] that chain custom instructions to create artificially low ILP or [21] that artificially invalidates TLB entries. Finally, EVAX, PerSpectron and RHMD do not include measurement of any HPCs for directly monitoring on core accelerators such as Intel AMX and thus should update their monitoring counters or techniques to mitigate NETLOKI.

III. COUNTERMEASURE

We investigated the root cause of Intel AMX's performance stages and confirmed that NETLOKI exploits power-state transitions driven by AMX's independent power management, rather than frequency scaling, operand dependencies, or core power-saving settings. Below is a summary of the six main findings:

Disabling Turbo Boost had no impact on AMX's performance stages, confirming that CPU frequency scaling (DVFS) is not responsible for the observed timing variations. ❶ Thus, disabling Turbo Boost does not mitigate NETLOKI. Fixing the CPU frequency (800 MHz–2 GHz) showed that while timing differences changed with frequency, even at 800 MHz, a 9,000-cycle gap remained exploitable. ❷ Therefore, frequency locking cannot fully mitigate NETLOKI, unlike Hertzbleed [13], which relies on frequency scaling. A slight timing variation depending on the operand is observed only when AMX is in deep sleep states but disappears as it warms up, ❸ ruling out operand values as the root cause of five AMX's performance modes. We hypothesized AMX power states are controlled by CPU C-states. On Sapphire Rapids, the measurable C-state residencies are C1 (light sleep), C6 (deep sleep), and C1E (Enhanced Halt State) which is an optional, enhanced version of the C-state between C1 and C6. AMX stages shown in Fig. 1 appear despite disabling both C-states and C1E. If we disable C1E while C-states is enabled and `usleep` is used to wait (which explicitly puts the CPU into a low C-state), AMX 20,000 cycle latency stage appears, leaving the attack viable. With C-states and C1E both enabled, while using usleep to wait, we see that the 20,000 cycle stage disappears. However, AMX exhibits new stages corresponding to a 3,000 and 9,000 cycle latency, indicating that ❹ while conventional C-states do not affect AMX power gating, C1E definitely does: C1E prevents AMX from entering the deepest sleep but it does not remove all exploitable stages therefore disabling C1E does not mitigate NETLOKI. Interestingly, when executing non-AMX instructions instead of using `usleep` to wait and therefore ensuring the CPU does not go to a lower C-state, all five AMX

performance stages are visible regardless of power settings (C-state or C1E). ❺ This indicates AMX undergoes staged power gating when the core remains active. We tested low-power and high-performance operating modes and NETLOKI works under both configurations when core remains active. Finally, measuring AMX power consumption at different stages confirmed a discrete (not continuous) decline in power usage, supporting a power-gated design hypothesis. ❻ The gradual decrease from Stage 0 (142.08 W) to Stage 4 (138.49 W) aligns with staged power gating, where AMX transitions through intermediate power states before full gating. Unlike cache-based or frequency-based side channels [13], [25], NETLOKI exploits power gating dynamics, making it fundamentally distinct from memory-access or frequency-throttling attacks.

Thus, to mitigate NETLOKI, we propose maintaining Intel AMX in a moderately warm state, preventing exploitable power-state transitions by eliminating timing variations. This requires minimal AMX utilization (at least 6% execution versus 94% busy-waiting) or pre-warming before SGX execution, as AMX timing leaks persist within enclaves due to unenclaved power gating. We verified that keeping AMX warm forces a stable execution state, closing the transmission channel. RAPL measurements show a power overhead of 2.59% to 12.33% for this mitigation, depending on the maintained AMX stage. Unlike traditional mitigations that degrade performance, this approach enhances security while increasing execution speed, though at a higher power cost, which contrasts with defenses for other microarchitectural vulnerabilities that slow execution [3], [6]. A microcode update or software patch could enforce this via `EENTER/EEXIT` hooks for SGX protection or periodic AMX activation. Future work should refine Intel AMX power management to balance security, power, and performance considering NETLOKI type attacks.

REFERENCES

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

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

[3] V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. 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.

[4] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *Proc. IEEE Symp. Secur. Privacy*, 2015, pp. 605–622.

[5] 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.

[6] 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.

[7] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack," in *Proc. 23rd USENIX Conf. Secur. Symp.*, 2014, pp. 719–732.

[8] 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.

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

[10] Z. N. Zhao, A. Morrison, C. W. Fletcher, and J. Torrellas, "Binoculars: {Contention-based}{side-channel} attacks exploiting the page walker," in *Proc. 31st USENIX Secur. Symp.*, 2022, pp. 699–716.

[11] B. Gras, K. Razavi, H. Bos, and C. Giuffrida, "Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks," in *Proc. USENIX Secur.*, 2018, pp. 955–972. [Online]. Available: https://www.vusec.net/download/?t=papers/tlbleed_sec18.pdf

[12] A. C. Aldaya, B. B. Brumley, S. ul Hassan, C. P. García, and N. Tuveri, "Port contention for fun and profit," 2018, [Online]. Available: https://eprint.iacr.org/2018/1060.pdf

[13] 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.

[14] D. Weber, A. Ibrahim, H. Nemati, M. Schwarz, and C. Rossow, "Osiris: Automated discovery of microarchitectural side channels," in *USENIX Secur. Symp.*, 2021, pp. 1415–1432.

[15] S. Mirbagher-Ajorpaz, G. Pokam, E. Mohammadian-Koruyeh, E. Garza, N. Abu-Ghazaleh, and D. A. Jiménez, "PerSpectron: Detecting invariant footprints of microarchitectural attacks with perceptron," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2020, pp. 1124–1137.

[16] K. N. Khasawneh, N. Abu-Ghazaleh, D. Ponomarev, and L. Yu, "RHMD: Evasion-resilient hardware malware detectors," in *Proc. Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2017, pp. 315–327.

[17] S. M. Ajorpaz, D. Moghimi, J. N. Collins, G. Pokam, N. Abu-Ghazaleh, and D. Tullsen, "EVAX: Towards a practical, pro-active & adaptive architecture for high performance & security," in *Proc. 55th IEEE/ACM Int. Symp. Microarchitecture*, 2022, pp. 1218–1236.

[18] H. Xiao and S. Ainsworth, "Hacky racers: Exploiting instruction-level parallelism to generate stealthy fine-grained timers," in *Proc. 28th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, New York, NY, USA, 2023, pp. 354–369, doi: 10.1145/3575693.3575700.

[19] M. Schwarz, M. Schwarzl, M. Lipp, and D. Gruss, "NetSpectre: Read arbitrary memory over network," 2018, *arXiv: 1807.10535*.

[20] H. Kim, G. Ye, N. Wang, A. Yazdanbakhsh, and N. S. Kim, "Exploiting intel advanced matrix extensions (AMX) for large language model inference," *IEEE Comput. Archit. Lett.*, vol. 23, no. 1, pp. 117–120, Jan./Jun. 2024.

[21] D. Skarlatos, M. Yan, B. Gopireddy, R. Sprabery, J. Torrellas, and C. W. Fletcher, "Microscope: Enabling microarchitectural replay attacks," *IEEE Micro*, vol. 40, no. 3, pp. 91–98, May/Jun. 2020.

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

[23] "Feature: Align performance API timer resolution to cross-origin isolated capability," 2022, Accessed: Sep. 11, 2024. [Online]. Available: https://chromestatus.com/feature/6497206758539264

[24] D. Li, Z. Zhang, M. Yao, Y. Cai, Y. Guo, and X. Chen, "TEESlice: Protecting sensitive neural network models in trusted execution environments when attackers have pre-trained models," 2024. [Online]. Available: https://arxiv.org/abs/2411.09945

[25] A. Baniasadi and A. Moshovos, "Instruction flow-based front-end throttling for power-aware high-performance processors," in *Proc. Int. Symp. Low Power Electron. Des.*, 2001, pp. 16–21.