

A. Appendix/Supplemental material

A.1. Detailed Experiments Setting

We implemented the Bit-flip Latency Attack using PyTorch 1.9.1 [21] on 10 NVIDIA Tesla V100-SXM3 GPUs [20]. To ensure a comprehensive evaluation, we tested several YOLO models, including YOLOv3, YOLOv4, YOLOv5, and YOLOv7 [1, 15, 24, 25, 31]. These models were initially pre-trained on the MS-COCO 2017 dataset (118K training images, 80 classes) [18] and subsequently fine-tuned on the BDD100k (100K training images, 10 classes) [34] and Pascal VOC (11K training images, 20 classes) [11] datasets using the PhantomSponges framework [26]. This fine-tuning aimed to adapt the models to diverse object detection contexts and validate the attack’s effectiveness across various scenarios.

Specifically, the MS-COCO 2017 dataset comprises 118K training and 5K validation images across 80 object classes. BDD100k, used for autonomous driving, includes 100K images across 10 classes and various environmental conditions. The Pascal VOC dataset provides 11K training images across 20 categories. For the attack experiments, we trained the models on a subset of 2000 images (640 x 640) for 70 epochs, with a learning rate of 0.1 and the Adam Optimizer [17]. To maintain consistent settings across all the experiments, we enabled inference with augmentation, which yielded up to 45,147 detections per image.

For baselines, we benchmarked our attack against a few state-of-the-art latency attacks, such as PhantomSponges (PS) [26] and Overload [3]. Since our work is the first to explore bit-flip attacks on object detection models, direct comparisons with other methods are either inappropriate or infeasible. This is because latency-based attacks are typically designed to exploit specific vulnerabilities and optimization strategies unique to individual models or applications, which limits overlap between different attacks and makes cross-application comparisons impractical. For example, although PandaSloth [13] and NICGSlowdown [6] also address latency attacks, they target applications other than object detection. Although SlowTrack [19] shares some similarities in attacking object detection, it mainly focuses on latency in tracking, which involves following objects over time rather than detecting them in individual frames. We will continue to monitor advancements in this field to update our comparisons with emerging research.

A.2. Additional Evaluations

In addition to the experiments presented in the paper, we conducted further evaluations to analyze specific aspects of our attack. Specifically, we tested the attack on CPUs and GPUs to evaluate how our method performs across different hardware configurations used for model execution. Additionally, we examined the effects of the attack on model

recall to demonstrate its ability to minimize undesired side effects while maintaining its effectiveness.

Latency increase on CPU and GPU platforms: To evaluate our attack’s performance across different platforms, we conducted tests on both CPUs and GPUs to understand how hardware differences affect the latency induced by our attack. We used YOLO models v3, v4, v5, and v7, and three datasets: COCO, VOC, and BDD100k. The results, presented in Fig. 1, are normalized against the model’s baseline performance (i.e., without the attack) to provide a clear comparison. We specifically analyzed the latency introduced by the NMS filter, as it is a critical component in object detection that can amplify latency changes. Our findings reveal that the attack significantly impacts latency on both types of platforms. On GPUs, latency increases by an average factor of $5\times$ to $20\times$, while on CPUs, it jumps by $100\times$ to $200\times$. Notably, in several instances, the attack causes latency spikes exceeding $20\times$ on GPUs and $200\times$ on CPUs. Even in the worst-case scenarios, our attack still induces a substantial latency increase of nearly $5\times$ on GPUs and $50\times$ on CPUs, demonstrating its effectiveness in degrading performance across different hardware platforms.

Ability to preserve recall: The model’s recall value reflects the percentage of correctly detected objects in the image. To evaluate the impact of our attack on the model’s performance, we assessed how the recall decreases due to the bit-flips induced by the attack. As illustrated in Fig. 2, our attack effectively preserves 50% to 95% of the model’s original recall, demonstrating its robustness in maintaining detection performance despite the introduced bit-flips. This performance surpasses previous methods, which achieved only 20% to 70% retention of the original recall while also inducing latency. Our results demonstrate that our attack effectively balances preserving a high percentage of detected objects from the original model while significantly increasing latency, thus enhancing its impact with minimal unintended consequences.

A.3. Understanding Latency Attacks

In this section, we revisit latency attacks to explore their growing appeal, focusing on key aspects such as attack objectives, incentives and potential attackers, and the complexities of detection.

Objectives and impacts: Latency attacks aim at increasing the execution times of neural network models. In critical real-time applications like autonomous driving, such delays can have severe consequences. For example, if latency causes a delay in object detection during obstacle avoidance, it could lead to slower reaction times, increasing the risk of collisions and endangering passenger safety [19]. In addition to execution delays, these attacks can significantly affect energy consumption, leading to excessive bat-

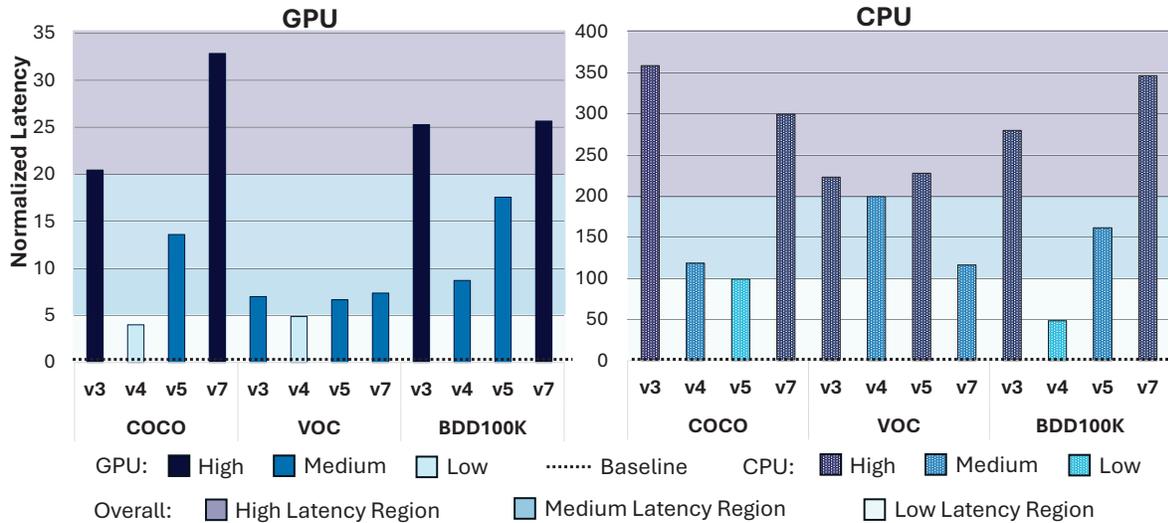


Figure 1. Evaluation of our attack performance on different hardware platforms.

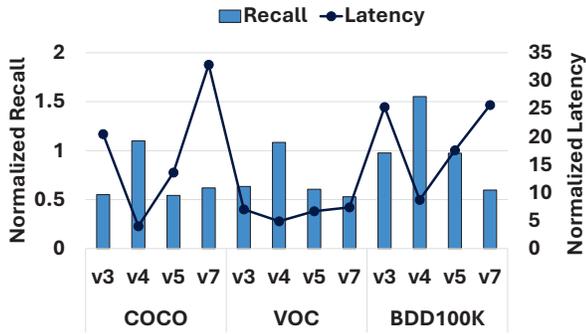


Figure 2. Normalized Recall and Normalized Latency Evaluation under the proposed attack.

tery drain, overheating, and potential device failure [27]. This disruption in quality of service impairs system performance and potentially causes damage to the hardware, highlighting the broader implications of latency attacks on both safety and operational efficiency.

Incentives for attackers: Modern machine learning algorithms are highly energy-intensive, making it crucial to optimize hardware and software for efficient task completion. However, this reliance on optimization has introduced vulnerabilities, allowing malicious actors such as competitors or cybercriminals to exploit latency attacks. These attacks can severely degrade the service quality and cause system failures in real-time applications by increasing energy consumption [4, 27]. For example, competitors might intentionally degrade a rival’s system performance to harm user experience and tarnish the product’s reputation, potentially gaining a competitive edge. Cybercriminals, on the other hand, might use latency attacks to disrupt critical services,

such as financial systems or emergency response platforms, leading to operational delays, financial losses, or compromised safety. By inducing latency, attackers can create confusion, diverting resources away from addressing other security threats and weakening the overall security posture of the targeted organization.

Complexities in detection: Unlike integrity attacks that might trigger sudden crashes, latency attacks subtly degrade model performance over time, making them more challenging to detect immediately [5, 7]. These attacks are designed to impair performance gradually rather than cause abrupt failures, which can be insidious. The impact of such degradation may only become evident after extended periods of operation, potentially affecting system reliability and user experience in subtle, yet significant ways.

Moreover, our method employs bit-flips induced by row-hammer to carry out latency attacks on object detection models during runtime. These bit-flips cause temporary changes since they occur while the model is still in memory [28, 33]. These modifications do not affect the on-disk model unless explicitly saved, which is uncommon, making detection more challenging. In contrast, bit-flip attacks on inputs or data require direct changes to files, making them more detectable. By targeting the model in memory, our attack introduces transient changes that are less likely to be immediately noticed.

Potential mitigation: While limiting the number of detected objects may reduce the impact of latency attacks, it does not eliminate the threat. Strictly setting thresholds can result in the exclusion of legitimate objects, thereby diminishing model accuracy [4]. Moreover, even with stringent thresholds, latency attacks can still cause significant increases in latency. For instance, [19] has demonstrated

that these attacks can remain effective even with stringent limits, increasing latency significantly and ultimately raising vehicle crash rates due to impaired object tracking.

A.4. Practical Feasibility of Bit-Flips

Utilizing row-hammer bit-flips to perform latency attacks requires the attacker to execute a specific program that repeatedly accesses memory while knowing targeted memory locations within the victim model. This knowledge enables the attacker to induce bit-flips at desired locations within the model’s allocated memory areas. In practice, while the memory characteristics of a model can be inferred through the analysis of open-source models, profiling, and other techniques [10, 12], actual bit-flip attacks necessitate custom programs executed on the target machine. Fortunately, direct hardware access or maintaining remote control is not required. Instead, attackers can leverage methods such as social engineering or exploiting vulnerabilities to persuade the victim into downloading a malicious application. Once installed, this application can automatically trigger bit-flips during the model’s runtime, manipulating the model’s performance without needing ongoing access or physical contact with the victim’s hardware. For example, [9] demonstrates a browser-based attack where bit-flips are induced remotely via JavaScript executed in the victim’s browser, illustrating the effectiveness of such attacks.

Obtaining the exact base address of the model remains challenging. However, the malicious applications introduced can dynamically locate the victim model’s memory address using several established techniques. Memory or pointer scanning allows the application to find specific values or patterns indicative of the model’s data, such as weights or activations. Attackers can also use operating system APIs to retrieve information about loaded modules, which assists in locating the model’s address. Additionally, when debugging symbols are available, they provide further context for pinpointing the base address and other critical memory locations. By using these methods, attackers can navigate through multiple layers of address translation to determine the base address of the model, enabling precise identification of desired parameters and effective execution of attacks.

Recent methods have explored active control over memory. For example, [33] describes multi-page memory masaging, where attackers exploit the CPU’s cache to strategically place victim pages between attacker-controlled pages. This involves requesting memory to ensure three consecutive rows in a DRAM bank are occupied and verifying the placement through reverse engineering and side-channel analysis [22]. Once exploitable pages are identified, they are freed and quickly replaced with the victim’s pages, facilitating targeted bit-flips.

By inducing users to download and run specialized ma-

licious programs and employing techniques to locate memory addresses, attackers can feasibly execute bit-flip attacks on the model. This demonstrates that bit-flip attacks are not only theoretically sound but also practical and achievable in real-world scenarios.

A.5. Extending Attacks to Various Architectures

Our attack method primarily focuses on Non-Maximum Suppression (NMS) because using NMS is a mainstream approach in many state-of-the-art object detection (OD) models, where it is crucial for post-processing to refine detection results. NMS is widely used for its effectiveness in eliminating redundant bounding boxes and improving detection accuracy. Aside from the direct transferability we have tested on similar YOLO models with NMS, our approach can also be extended to other object detection models that incorporate NMS, such as Faster-RCNN [8]. For transformer-based designs, it is important to examine whether the integrated detector head includes an NMS process. For example, DETA [32] reintroduces NMS on top of the DETR architecture to enhance detection accuracy. Our method can be extended to such models with specific modifications to accommodate the specific implementations.

Recently, new object detection models have emerged that aim to eliminate the traditional NMS filtering step, such as YOLOv10 [30], DETR [2], and RT-DETR [35]. Although our attack cannot be directly applied to these models, the underlying attack logic remains relevant by targeting their optimization strategies. For example, YOLOv10 employs a Rank-guided block design to reduce computational redundancy by replacing less critical blocks with compact inverted blocks (CIBs) that use simpler convolutions. Thus, we can design a latency attack to exploit this optimization, aiming to force the model to operate at full capacity while maintaining high accuracy. In the future, we will further investigate how to adapt our attack to other NMS-free architectures.

A.6. Row-Hammer Deployment

Row-Hammer is a technique for inducing bit flips in memory by repeatedly accessing specific memory rows. This approach exploits the physical properties of DRAM to cause unintended changes in adjacent memory cells. Row-hammer is attractive for performing the designed latency attack due to its efficiency and its recognition as a critical security vulnerability affecting DRAM.

First introduced in a landmark ISCA 2014 study [16], row-hammer induces bit flips by repeatedly accessing specific memory rows, exploiting DRAM’s physical properties. This technique is typically executed through malicious applications that use designed memory access patterns, often involving cache-flush instructions to bypass caching mechanisms and directly target vulnerable DRAM cells. Recent

advancements have significantly improved the effectiveness, reducing the required memory accesses and increasing precision. For instance, [29] demonstrated its use for escalating privileges on mobile systems, while [23] showed how to target specific memory locations by manipulating access patterns. Further refinements by [14] optimized data patterns, achieving a 72.4% success rate in targeted attacks and reducing unintended bit flips by 99.7%. The severity of the row-hammer vulnerability has worsened with DRAM density increases over the past decade, resulting in a tenfold reduction in activation requirements and a 500-fold increase in bit-flip frequency for the same number of activations.

However, designing an effective row-hammer application to induce bit flips requires specialized expertise in memory manipulation and a deep understanding of hardware systems. Our primary goal is to propose an attack model that effectively leverages row-hammer, rather than developing a specific row-hammer program. In practice, we can utilize established row-hammer techniques, such as those described in [23,33], and adapt them to target the specific memory models of object detection systems to achieve optimal attack results.

References

- [1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020. 1
- [2] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020. 3
- [3] Erh-Chung Chen, Pin-Yu Chen, I Chung, Che-rung Lee, et al. Overload: Latency attacks on object detection for edge devices. *arXiv preprint arXiv:2304.05370*, 2023. 1
- [4] Erh-Chung Chen, Pin-Yu Chen, I Chung, Che-Rung Lee, et al. Overload: Latency attacks on object detection for edge devices. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 24716–24725, 2024. 2
- [5] Simin Chen, Hanlin Chen, Mirazul Haque, Cong Liu, and Wei Yang. The dark side of dynamic routing neural networks: Towards efficiency backdoor injection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 24585–24594, 2023. 2
- [6] Simin Chen, Zihe Song, Mirazul Haque, Cong Liu, and Wei Yang. Nicgslowdown: Evaluating the efficiency robustness of neural image caption generation models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15365–15374, 2022. 1
- [7] Wencheng Chen and Hongyu Li. Stealthy energy consumption-oriented attacks on training stage in deep learning. *Journal of Signal Processing Systems*, 95(12):1425–1437, 2023. 2
- [8] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. 3
- [9] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer.js: A remote software-induced fault attack in javascript. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*, pages 300–321. Springer, 2016. 3
- [10] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Bad-nets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017. 3
- [11] Derek Hoiem, Santosh K Divvala, and James H Hays. Pascal voc 2008 challenge. *World Literature Today*, 24(1):1–4, 2009. 1
- [12] Sanghyun Hong, Nicholas Carlini, and Alexey Kurakin. Handcrafted backdoors in deep neural networks. *Advances in Neural Information Processing Systems*, 35:8068–8080, 2022. 3
- [13] Sanghyun Hong, Yiğitcan Kaya, Ionuț-Vlad Modoranu, and Tudor Dumitraș. A panda? no, it’s a sloth: Slowdown attacks on adaptive multi-exit neural network inference. *arXiv preprint arXiv:2010.02432*, 2020. 1
- [14] Sangwoo Ji, Youngjoo Ko, Saeyoung Oh, and Jong Kim. Pinpoint rowhammer: Suppressing unwanted bit flips on rowhammer attacks. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pages 549–560, 2019. 4
- [15] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, Yonghye Kwon, Kalen Michael, Jiacong Fang, Colin Wong, Zeng Yifu, Diego Montes, et al. ultralytics/yolov5: v6. 2-yolov5 classification models, apple m1, reproducibility, clearml and deci. ai integrations. *Zenodo*, 2022. 1
- [16] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. *ACM SIGARCH Computer Architecture News*, 42(3):361–372, 2014. 3
- [17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1
- [18] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014. 1
- [19] Chen Ma, Ningfei Wang, Qi Alfred Chen, and Chao Shen. Slowtrack: Increasing the latency of camera-based perception in autonomous driving using adversarial examples. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 4062–4070, 2024. 1, 2
- [20] NVIDIA. Nvidia tesla v100. <https://www.nvidia.com/en-us/data-center/v100/>. 1
- [21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming

- Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 1
- [22] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. {DRAMA}: Exploiting {DRAM} addressing for {Cross-CPU} attacks. In *25th USENIX security symposium (USENIX security 16)*, pages 565–581, 2016. 3
- [23] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip feng shui: Hammering a needle in the software stack. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1–18, 2016. 4
- [24] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 1
- [25] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 1
- [26] Avishag Shapira, Alon Zolfi, Luca Demetrio, Battista Biggio, and Asaf Shabtai. Phantom sponges: Exploiting non-maximum suppression to attack deep object detectors. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 4571–4580, 2023. 1
- [27] Iliia Shumailov, Yiren Zhao, Daniel Bates, Nicolas Papernot, Robert Mullins, and Ross Anderson. Sponge examples: Energy-latency attacks on neural networks. In *2021 IEEE European symposium on security and privacy (EuroS&P)*, pages 212–231. IEEE, 2021. 2
- [28] M Caner Tol, Saad Islam, Andrew J Adiletta, Berk Sunar, and Ziming Zhang. Don’t knock! rowhammer at the backdoor of dnn models. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 109–122. IEEE, 2023. 2
- [29] Victor Van Der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic rowhammer attacks on mobile platforms. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1675–1689, 2016. 4
- [30] Ao Wang, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, and Guiguang Ding. Yolov10: Real-time end-to-end object detection. *arXiv preprint arXiv:2405.14458*, 2024. 3
- [31] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7464–7475, 2023. 1
- [32] Kai Yang, Haijun Zhang, Feng Gao, Jianyang Shi, Yanfeng Zhang, and QM Jonathan Wu. Deta: A point-based tracker with deformable transformer and task-aligned learning. *IEEE Transactions on Multimedia*, 25:7545–7558, 2022. 3
- [33] Fan Yao, Adnan Siraj Rakin, and Deliang Fan. {DeepHammer}: Depleting the intelligence of deep neural networks through targeted chain of bit flips. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1463–1480, 2020. 2, 3, 4
- [34] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2636–2645, 2020. 1
- [35] Yian Zhao, Wenyu Lv, Shangliang Xu, Jinman Wei, Guanzhong Wang, Qingqing Dang, Yi Liu, and Jie Chen. Detsr beat yolos on real-time object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16965–16974, 2024. 3