DeSparsify: Adversarial Attack Against Token Sparsification Mechanisms

Oryan Yehezkel[*], Alon Zolfi[*], Amit Baras, Yuval Elovici, Asaf Shabtai

Ben-Gurion University of the Negev, Israel

[*]Equal Contribution

Motivation

- Vision transformers have demonstrating state-of-the-art performance in diverse tasks (e.g., image classification, object detection).
- However, their high computational requirements grow quadratically with the number of tokens used.
- Token sparsification techniques have been proposed to address this issue. These techniques employ an input-dependent strategy, in which uninformative tokens are discarded from the computation pipeline, improving the model’s efficiency.
- Despite their benefits, the robustness of these techniques against availability attacks remains underexplored.

DeSparsify

Attack Goal

- In this work we present DeSparsify attack
- The attack increases the model's computational cost by adding a small perturbation to the input that is invisible to the human eye.
- Increase the number of chosen tokens within the model. It can be achieved by comprehending the selection mechanism of the attacked module, we can manipulate the model's decisions, compelling it to elevate the number of tokens utilized.
- thereby potentially compromising its performance due to the associated computational overhead. Additionally, is likely to increase energy, memory, throughput & GFLOPS

DeSparsify

Methodology

- We utilize the PGD attack and propose a novel loss function used to iteratively update the perturbation pixels.

$$\delta^{t+1} = \prod_{||\delta||_p < \epsilon} (\delta^t + \alpha \cdot \text{sign}(\nabla_\delta \sum_{(x,y)\in\mathcal{D}'} \mathcal{L}(x,y)))$$

- The loss function consists of two components:
  - $\mathcal{L}_{\text{atk}}$ - the attacking component aimed at maintaining as many tokens as possible.
  - $\mathcal{L}_{\text{cls}}$ - maintain the model's original classification for a stealthier attack.

$$\mathcal{L} = \mathcal{L}_{\text{atk}} + \lambda \cdot \mathcal{L}_{\text{cls}}$$

DeSparsify

Methodology

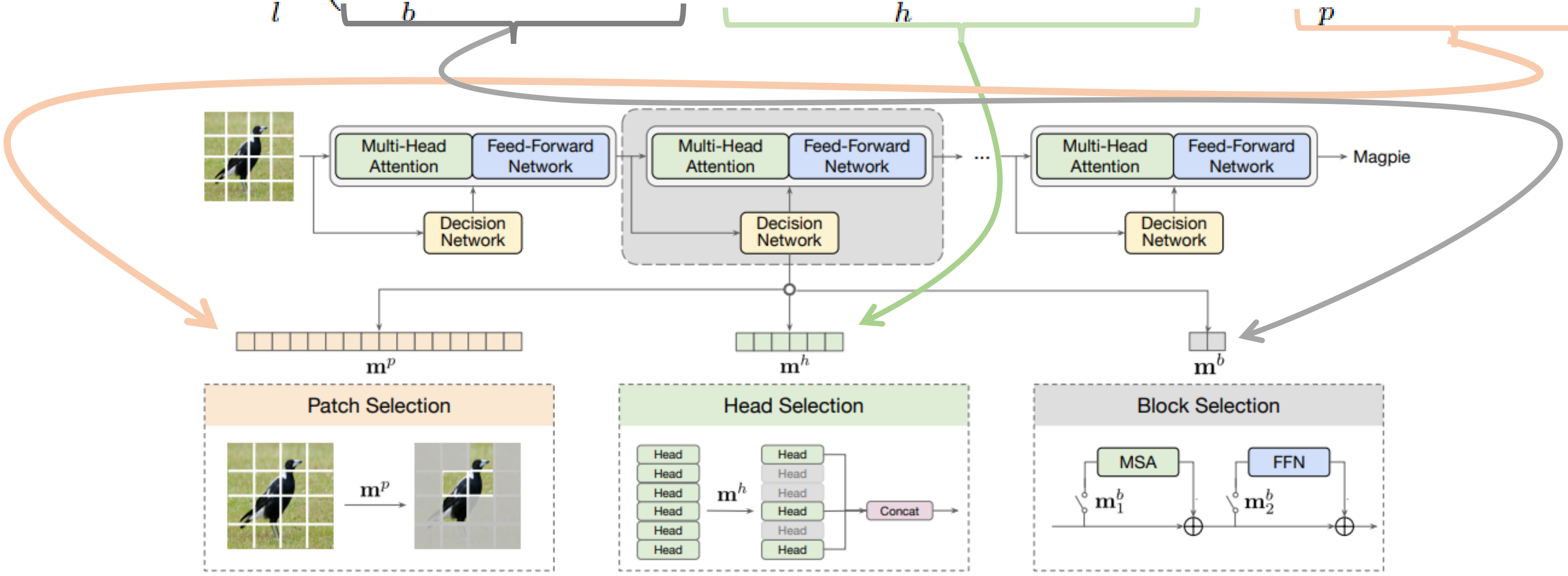
- In this work, we target three primary TS mechanisms:
  - ATS [15]- differentiable parameter-free module that adaptively down-samples input tokens by utilizing the significance scores from the attention matrix.
  - AdaViT [16]- adaptively determines the use of tokens, self-attention heads, and transformer blocks using a lightweight decision network.
  - A-ViT [17]- halts the computation of different tokens at different depths using a halting score-based module that is incorporated into the model's network.

DeSparsify

Methodology

- To attack AdaViT's TS mechanism we aim to push its decision network to never discard any tokens.

$$\mathcal{L}_{\text{AdaViT}} = \frac{1}{L}\sum_{l}^{L}\left(\frac{1}{2}\sum_{b}^{2}\ell_{\text{MSE}}(M_l^b, \hat{M}_l^b) + \frac{\mathbb{1}_{M_l^0=1}}{H}\sum_{h}^{H}\ell_{\text{MSE}}(M_l^h, \hat{M}_l^h) + \frac{1}{N}\sum_{p}^{N}\ell_{\text{MSE}}(M_l^p, \hat{M}_l^p)\right)$$



Classification loss

- To increase the stealthiness of our attack by preserving the original classification of the input image, we use the output of the attacked image and bring it closer to the output of the clean image.

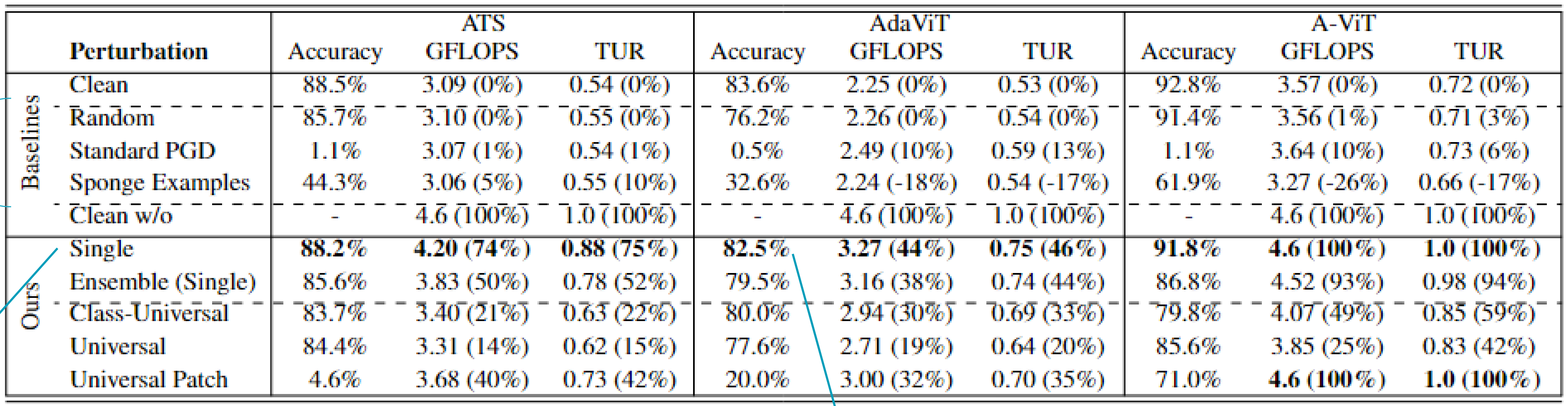
- By keeping the adversarial output similar to the clean output, we minimize any discrepancies that could be flagged by anomaly detection systems.

$$\mathcal{L}_{cls} = \frac{1}{M} \sum_{m=1}^{M} \mathcal{L}_{CE}\big(f_m\,(x+\delta), f_m\,(x)\big)$$

DeSparsify

Main Results

Baselines are incapable of compromising the sparsification mechanism



		ATS			AdaViT			A-ViT		
	Perturbation	Accuracy	GFLOPS	TUR	Accuracy	GFLOPS	TUR	Accuracy	GFLOPS	TUR
Baselines	Clean	88.5%	3.09 (0%)	0.54 (0%)	83.6%	2.25 (0%)	0.53 (0%)	92.8%	3.57 (0%)	0.72 (0%)
	Random	85.7%	3.10 (0%)	0.55 (0%)	76.2%	2.26 (0%)	0.54 (0%)	91.4%	3.56 (1%)	0.71 (3%)
	Standard PGD	1.1%	3.07 (1%)	0.54 (1%)	0.5%	2.49 (10%)	0.59 (13%)	1.1%	3.64 (10%)	0.73 (6%)
	Sponge Examples	44.3%	3.06 (5%)	0.55 (10%)	32.6%	2.24 (-18%)	0.54 (-17%)	61.9%	3.27 (-26%)	0.66 (-17%)
	Clean w/o	-	4.6 (100%)	1.0 (100%)	-	4.6 (100%)	1.0 (100%)	-	4.6 (100%)	1.0 (100%)
Ours	Single	**88.2%**	**4.20 (74%)**	**0.88 (75%)**	**82.5%**	**3.27 (44%)**	**0.75 (46%)**	**91.8%**	**4.6 (100%)**	**1.0 (100%)**
	Ensemble (Single)	85.6%	3.83 (50%)	0.78 (52%)	79.5%	3.16 (38%)	0.74 (44%)	86.8%	4.52 (93%)	0.98 (94%)
	Class-Universal	83.7%	3.40 (21%)	0.63 (22%)	80.0%	2.94 (30%)	0.69 (33%)	79.8%	4.07 (49%)	0.85 (59%)
	Universal	84.4%	3.31 (14%)	0.62 (15%)	77.6%	2.71 (19%)	0.64 (20%)	85.6%	3.85 (25%)	0.83 (42%)
	Universal Patch	4.6%	3.68 (40%)	0.73 (42%)	20.0%	3.00 (32%)	0.70 (35%)	71.0%	**4.6 (100%)**	**1.0 (100%)**

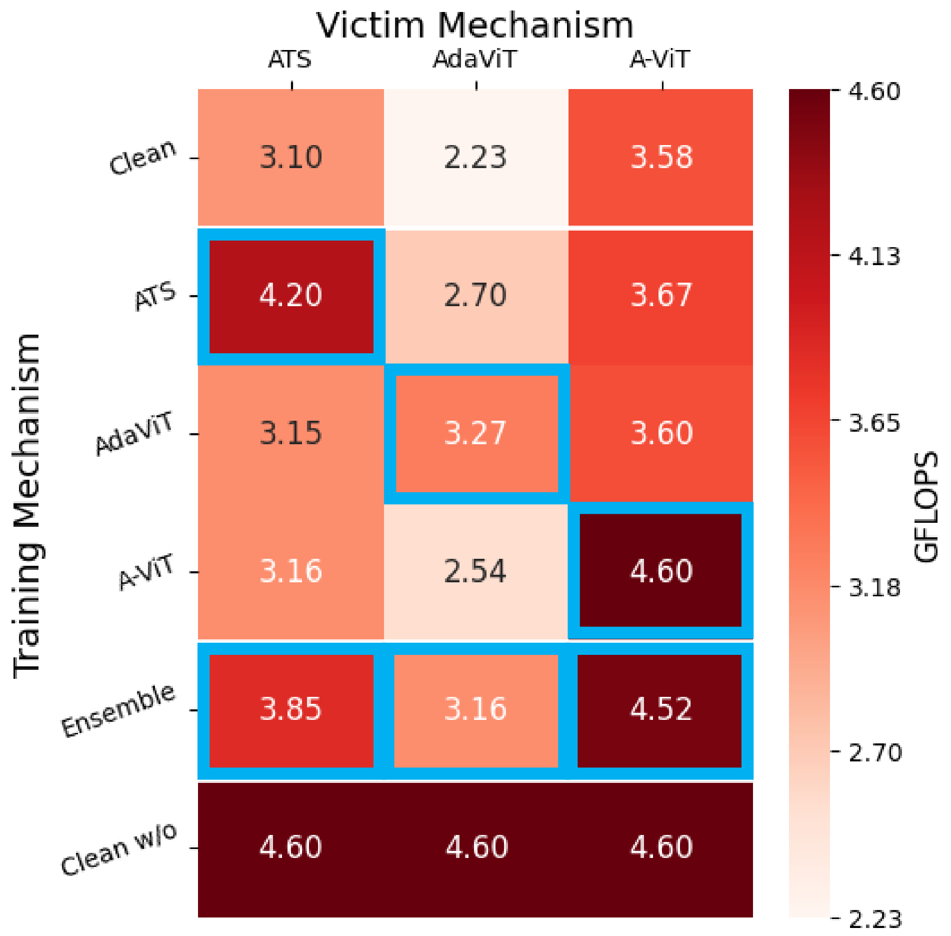
Single-Image perturbations increase GFLOPS by 74%, 44%, and 100% on the ATS, AdaViT, and A-ViT, respectively.

Note that the crafted perturbations have just a minor effect on the model's classification accuracy.

DeSparsify

Transferability & Ensemble

- We examine the effect of perturbations trained on a model with one sparsification mechanism are tested on the same model with a different sparsification mechanism.
  - Transferability works to some extent, however, not as much as white-box attacks.
- Another strategy we evaluate is the ensemble training strategy, in which the adversarial example is trained concurrently on all of the sparsification techniques.
  - Here, the perturbation can affect all TS mechanisms, achieving nearly the same performance as white-box attacks.

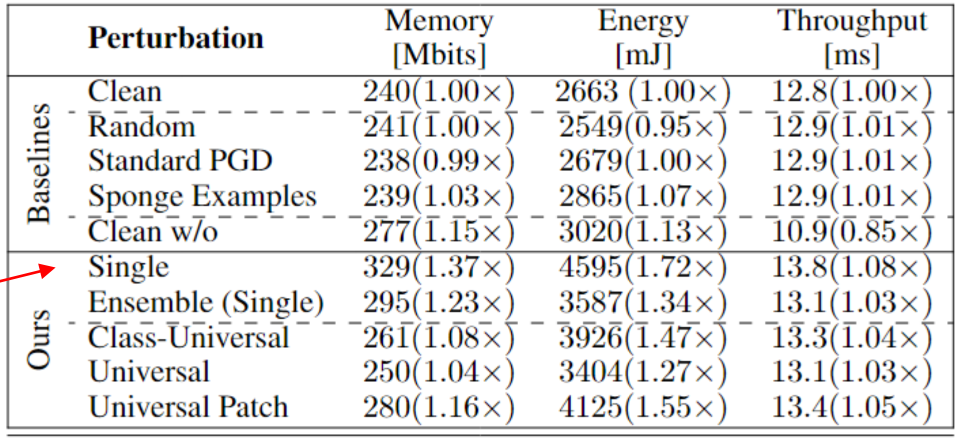


Training Mechanism	ATS	AdaViT	A-ViT
Clean	3.10	2.23	3.58
ATS	4.20	2.70	3.67
AdaViT	3.15	3.27	3.60
A-ViT	3.16	2.54	4.60
Ensemble	3.85	3.16	4.52
Clean w/o	4.60	4.60	4.60

DeSparsify

Effect on Hardware

- We also assess the effect of our attack on hardware, based on several GPU metrics.
- As opposed to the baselines that have no effect, the attack significantly affects the GPU metrics.
    - For example, the single-image attack variant increases the memory usage by 37%, the energy consumption by 72%, and the throughput by 8% compared to the clean images.



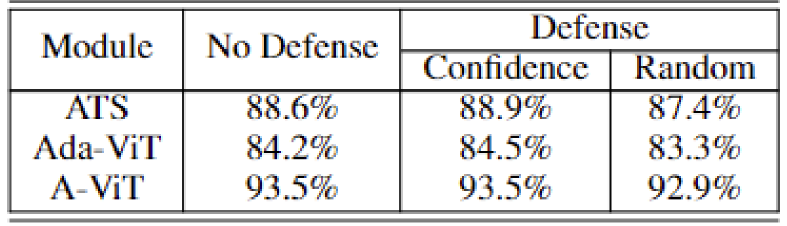
	Perturbation	Memory [Mbits]	Energy [mJ]	Throughput [ms]
Baselines	Clean	240(1.00×)	2663 (1.00×)	12.8(1.00×)
	Random	241(1.00×)	2549(0.95×)	12.9(1.01×)
	Standard PGD	238(0.99×)	2679(1.00×)	12.9(1.01×)
	Sponge Examples	239(1.03×)	2865(1.07×)	12.9(1.01×)
	Clean w/o	277(1.15×)	3020(1.13×)	10.9(0.85×)
Ours	Single	329(1.37×)	4595(1.72×)	13.8(1.08×)
	Ensemble (Single)	295(1.23×)	3587(1.34×)	13.1(1.03×)
	Class-Universal	261(1.08×)	3926(1.47×)	13.3(1.04×)
	Universal	250(1.04×)	3404(1.27×)	13.1(1.03×)
	Universal Patch	280(1.16×)	4125(1.55×)	13.4(1.05×)

DeSparsify

Countermeasures

- To actively mitigate the presented threats, an upper bound can be set to the number of tokens used in each transformer block.
    - Can be determined by computing the average number of active tokens in each block on a holdout set.

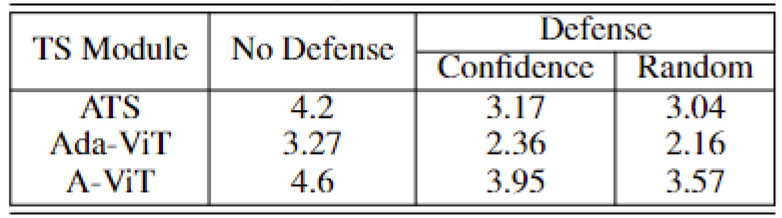
Table 10: Accuracy results for clean images on DeiT-s with and without the proposed defense.



Module	No Defense	Defense	
		Confidence	Random
ATS	88.6%	88.9%	87.4%
Ada-ViT	84.2%	84.5%	83.3%
A-ViT	93.5%	93.5%	92.9%

The defense does not compromise model accuracy on clean (benign) images.

Table 11: GFLOPS results for adversarial images on DeiT-s with and without the proposed defense.



TS Module	No Defense	Defense	
		Confidence	Random
ATS	4.2	3.17	3.04
Ada-ViT	3.27	2.36	2.16
A-ViT	4.6	3.95	3.57

The defense successfully mitigates the attack's effectiveness.

Summary & Discussion

- **<u>Vulnerabilities</u>**: Our research exposes key weaknesses in token sparsification mechanisms in vision transformers, enabling an attack that raises computational cost and energy use.

- **<u>DeSparsify Attack</u>**: Demonstrated effectiveness across various TS mechanisms, with thorough evaluation on transferability and hardware impact.

- **<u>Future Outlook</u>**: Emphasizes the need for secure deployment in resource-sensitive environments. Suggests future work on robust defenses and applications in other domains.

Thank You