TRUSTWORTHY MACHINE LEARNING

AVAILABILITY

MOHAMMAD ALOMRANI, QUINLAN SYKORA, JIAQI WANG,
with help from DHIRAJ TAWANI and SIPHELELE DANISA

OCTOBER 5TH, 2021

UNIVERSITY OF TORONTO

Threat model : A Brief Recap

UNIVERSITY OF TORONTO

Basics of Security - CIA TRIAD

UNIVERSITY OF TORONTO

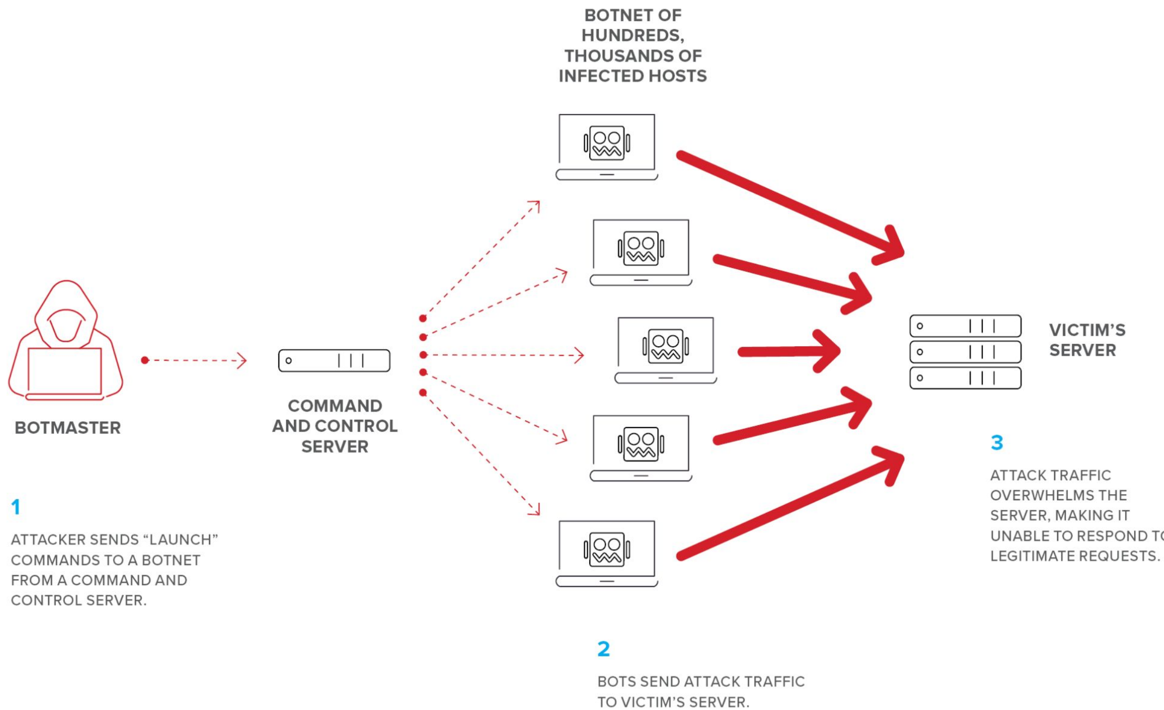
Availability

- Availability, commonly defined on a high level, guarantees that systems, applications and data are available to users when they need them.

- Disruption of system availability for even a short time can lead to loss of revenue, customer dissatisfaction and reputation damage.

- Some availability attacks can directly affect people's lives e.g. disabling pilot system of a self-driving car, attacking an autonomous public transportation system or a critical healthcare system.

General Example of Availability Attack

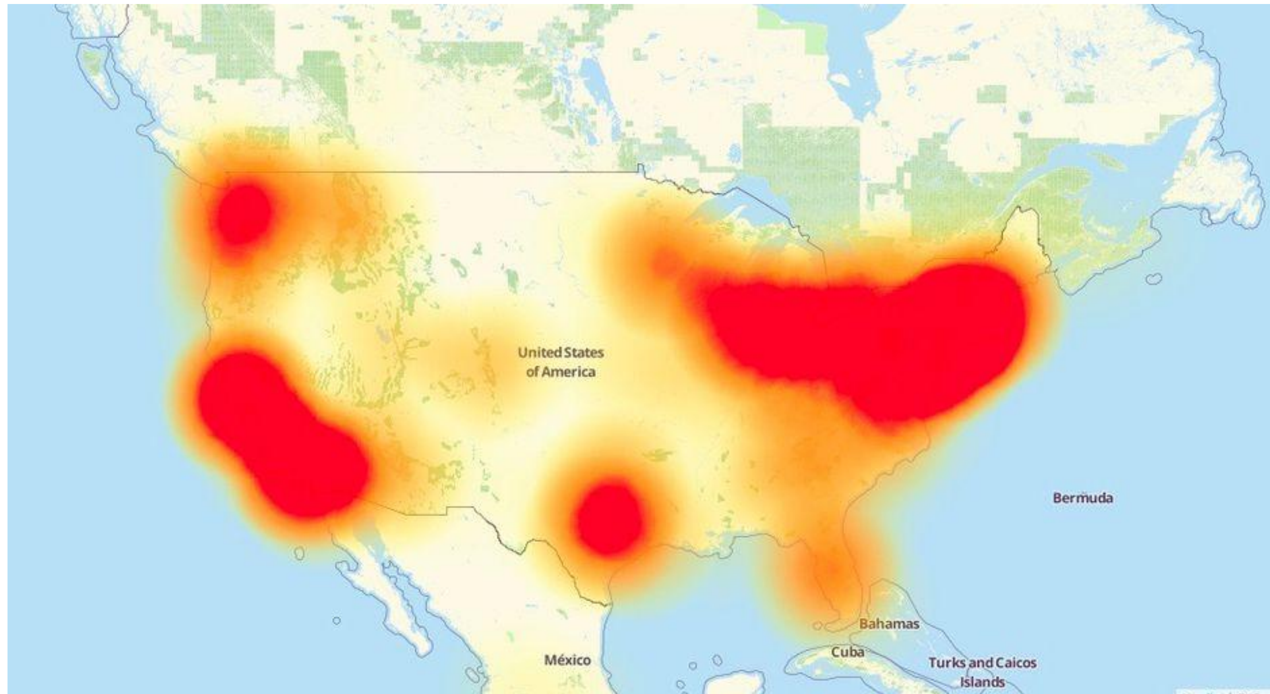


UNIVERSITY OF TORONTO

DDoS

UNIVERSITY OF TORONTO

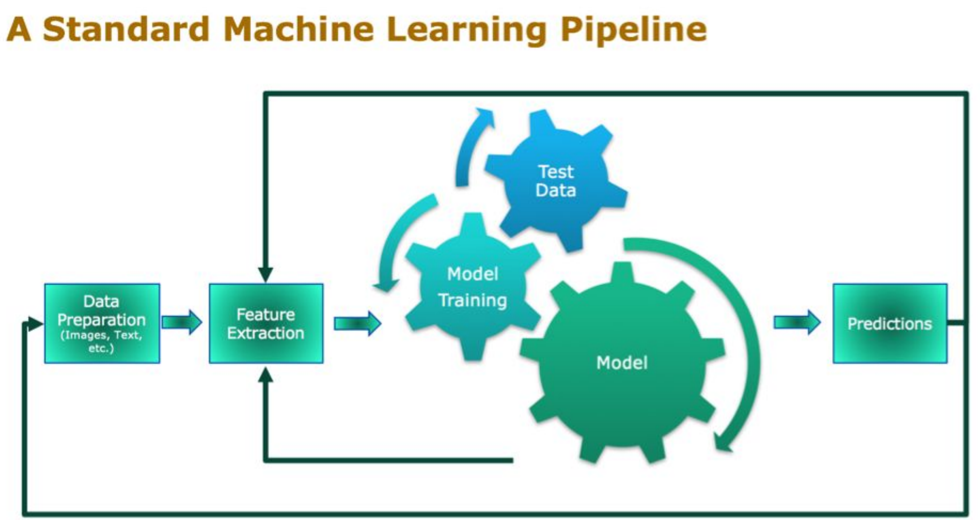
The Mirai Dyn DDoS Attack in 2016



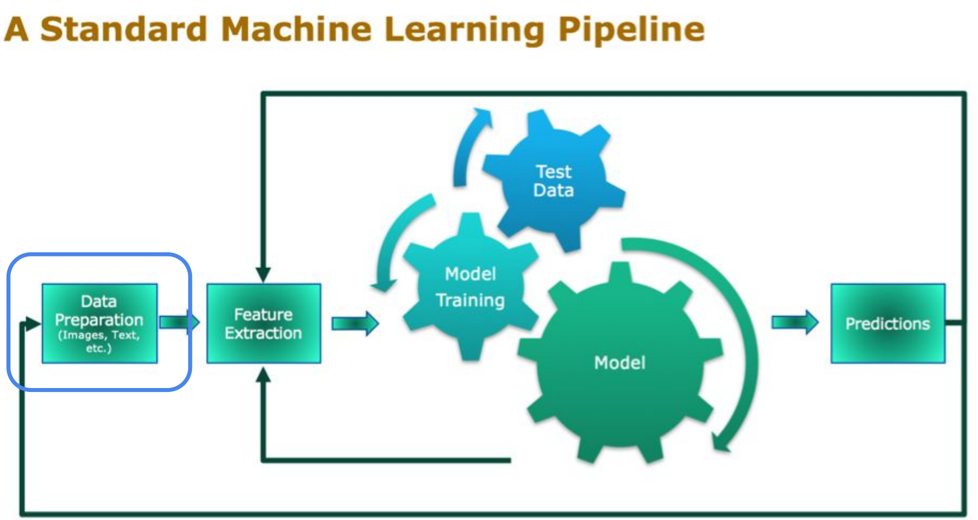
Availability of ML Systems

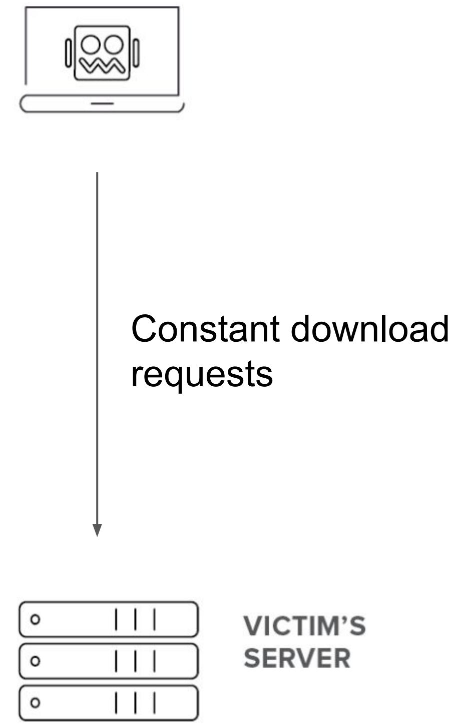
- Modern ML models have many threat vectors.
- To name a few: adversarial examples, data poisoning, membership inference, and fault injection attacks.
- These attacks target the **confidentiality** and **integrity** of ML systems.
- Can one target the **availability** of ML systems at the inference/training stage?

Availability of ML Systems



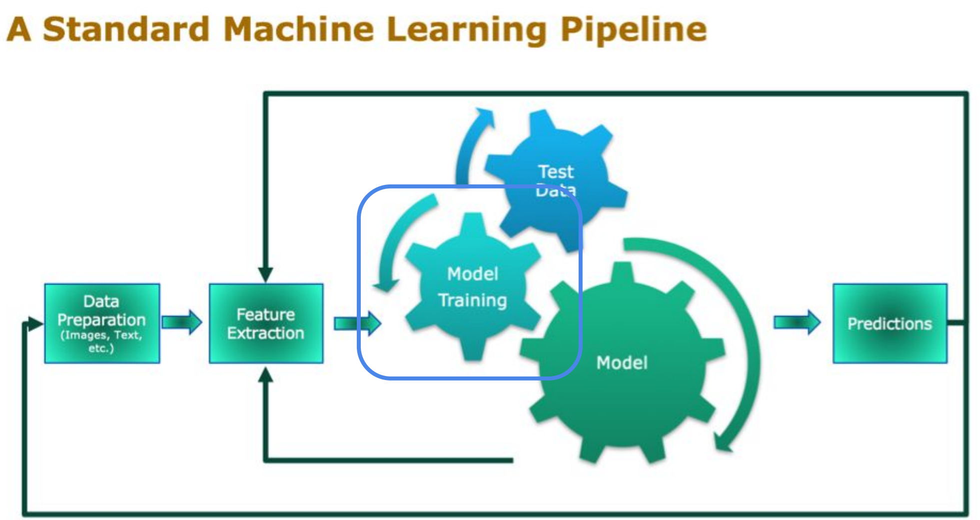
Availability of ML Systems

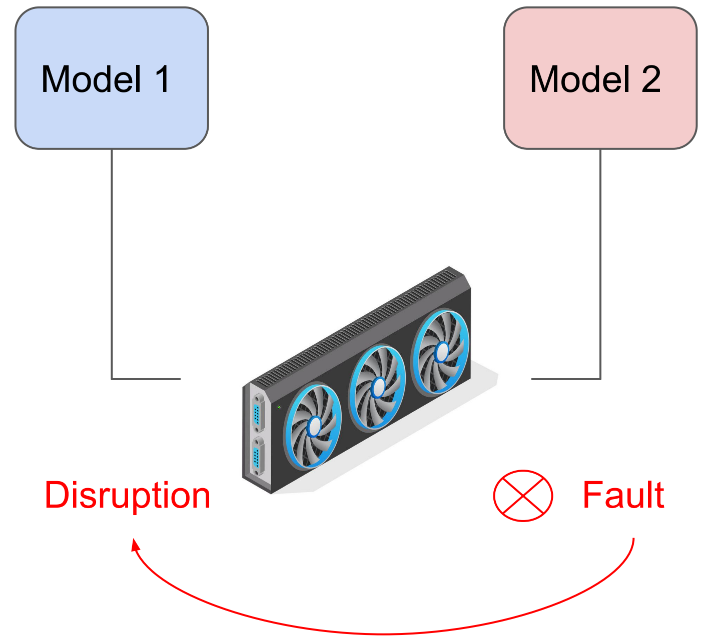




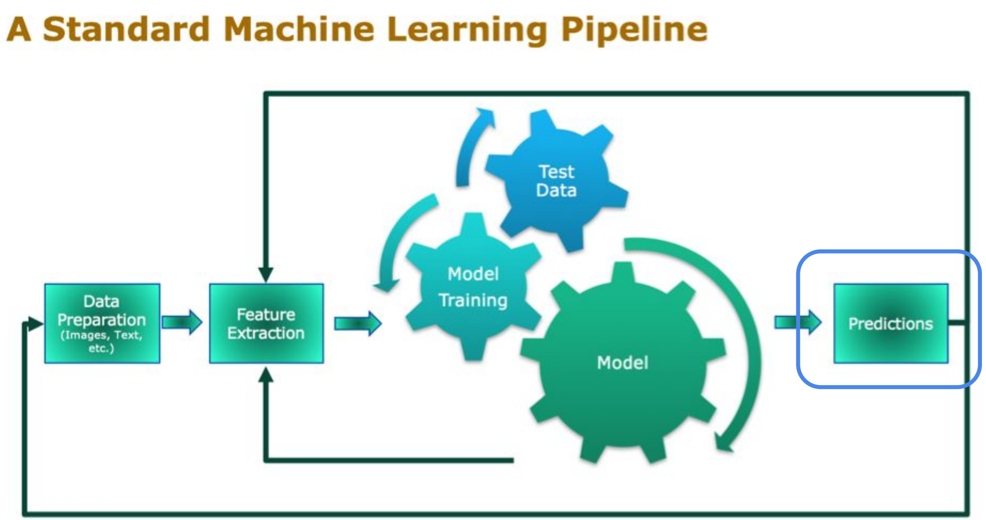
UNIVERSITY OF TORONTO

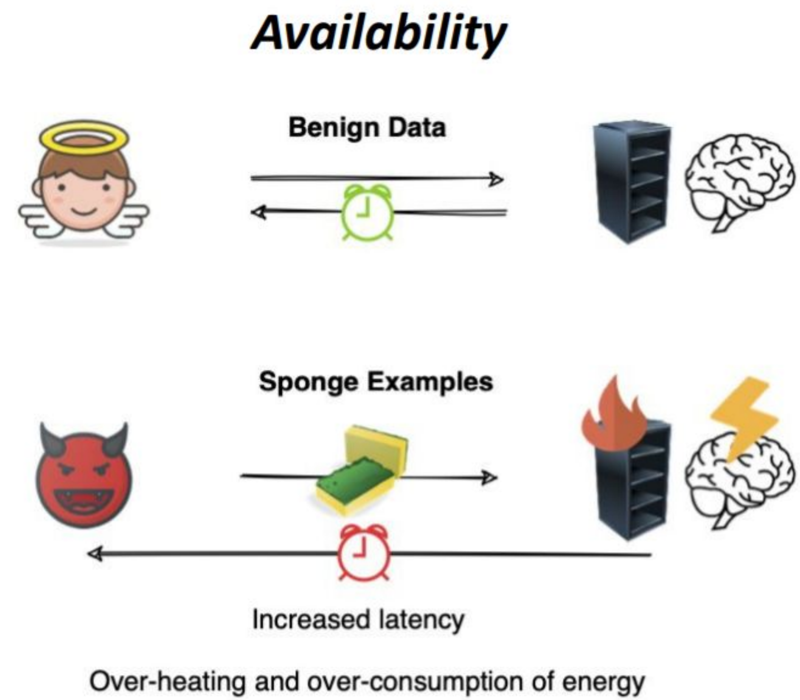
Availability of ML Systems



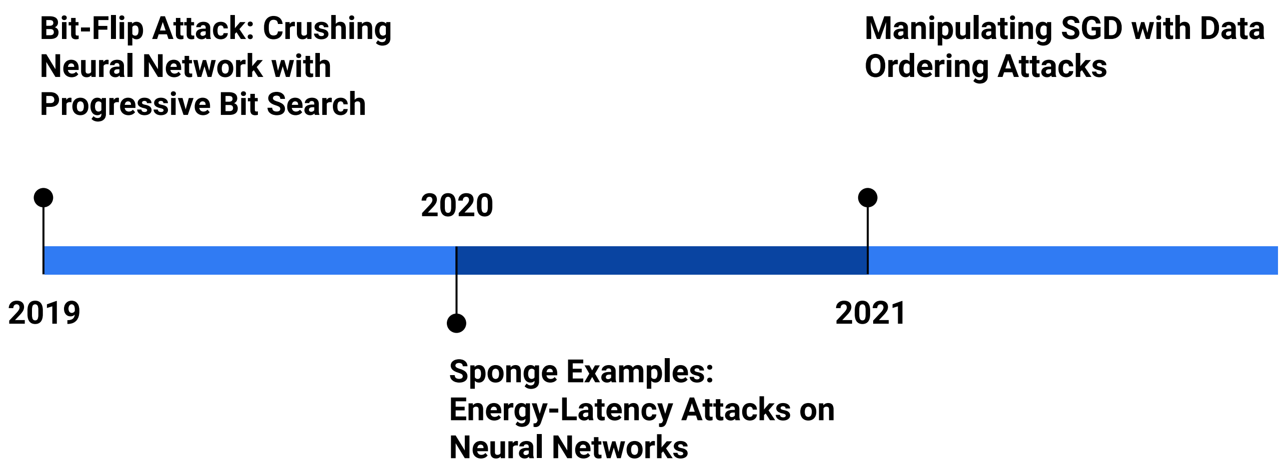


Availability of ML Systems





Overview of the timeline



UNIVERSITY OF TORONTO

1

Bit-Flip Attack: Crushing Neural Network with Progressive Bit Search

Adnan Siraj Rakin and Zhezhi He *[1] and Deliang Fan†[1]

[1]Department of Computer Engineering, University of Central Florida

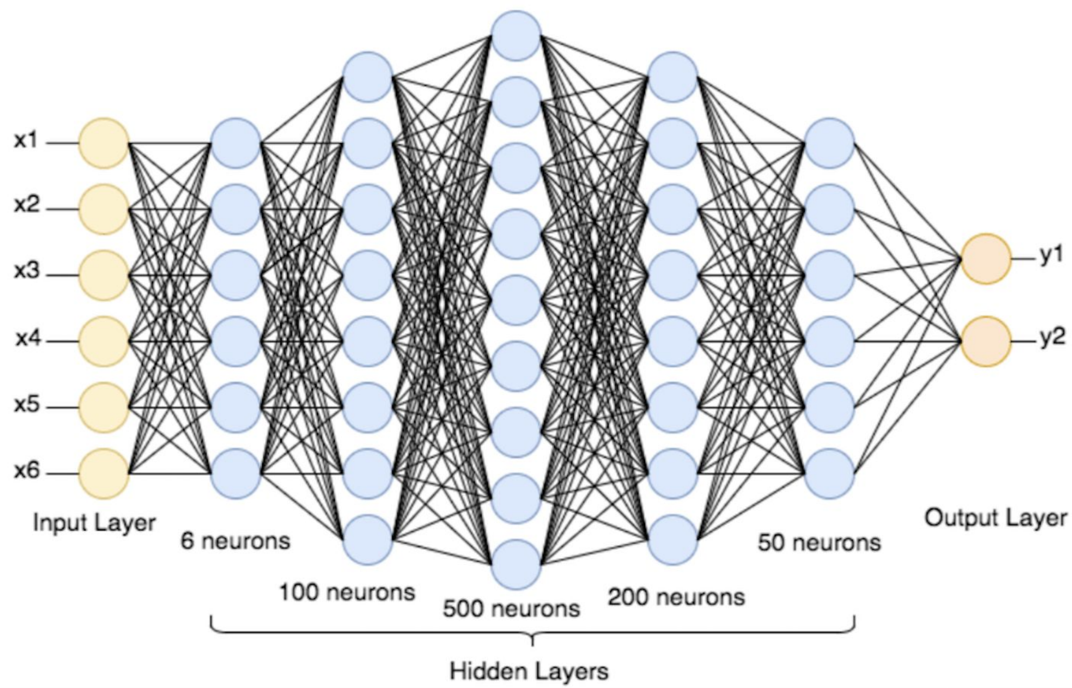
Abstract

Several important security issues of Deep Neural Network (DNN) have been raised recently associated with different applications and components. The most widely investigated security concern of DNN is from its malicious input, a.k.a adversarial example. Nevertheless, the security challenge of DNN's parameters is not well explored yet. In this work, we are the first to propose a novel DNN weight attack methodology called Bit-Flip Attack (BFA) which can crush a neural network through maliciously flipping extremely small amount of bits within its weight storage memory system (i.e., DRAM). The bit-flip operations could be conducted through well-known Row-Hammer attack, while our main contribution is to develop an algorithm to identify the most vulnerable bits of DNN weight parameters (stored in memory as binary bits), that could maximize the accuracy degradation with a minimum number of bit-flips. Our proposed BFA utilizes a Progressive Bit Search (PBS) method which combines gradient ranking and progressive search to identify the most vulnerable bit to be flipped. With the aid of PBS, we can successfully attack a ResNet-18 fully malfunction (i.e., top-1 accuracy degrade from 69.8% to 0.1%) **only through 13 bit-flips out of 93 million bits**, while randomly flipping 100 bits merely degrades the accuracy by less than 1%.

UNIVERSITY OF TORONTO

Problems with Traditional DNN

- DNNs are ineffective because of huge amount calculation of the weights.
- Hard to deploy on small device or CPU machine



Problem with Bit-Flip Attack on DNN

- The model itself is so vulnerable that people has begun to avoid -- just flipping the most significant exponent bits can destroy DNN.
- Nowadays people has moved onto weight constrained DNNs

Why QNN

- Quantized Neural Networks (QNNs) --- neural networks with extremely low precision (e.g., 1-bit) weights and activations, at run-time.
- QNNs reduce computation on floating-point based numbers, reduce the computation to bit-wise.
- Can be deployed to small device or CPU machine now.

Quantization in DNN

- Quantization: approximating a neural network that uses floating-point numbers by a neural network of low bit width numbers.
- Weight quantization:
  - For l-th layer, the quantization process from the floating-point base Wfpl to its fixed-point (signed integer) counterpart Wl can be described as:
- Weight encoding:
  - The computing system normally stores the signed integer in two's complement representation, owing to its efficiency in arithmetic operations (e.g., mul).
- **Basically a function from weights to bits**

Threat Introduced by Quantization

- Flipping a memory cell bit is possible
- It is deployed in small devices which lack data integrity check machinism

Bit-Flip Attack (BFA)

- BFA has the similar mechanism as FGSM which was used to generate adversarial example.
- Key Idea of BFA is to flip the bits along the its gradient ascending direction w.r.t the loss of DNN.

$$\hat{\boldsymbol{b}} = \boldsymbol{b} + \operatorname{sign}(\nabla_{\boldsymbol{b}}\mathcal{L})$$

Naive way of doing BFA - leads to data overflow.

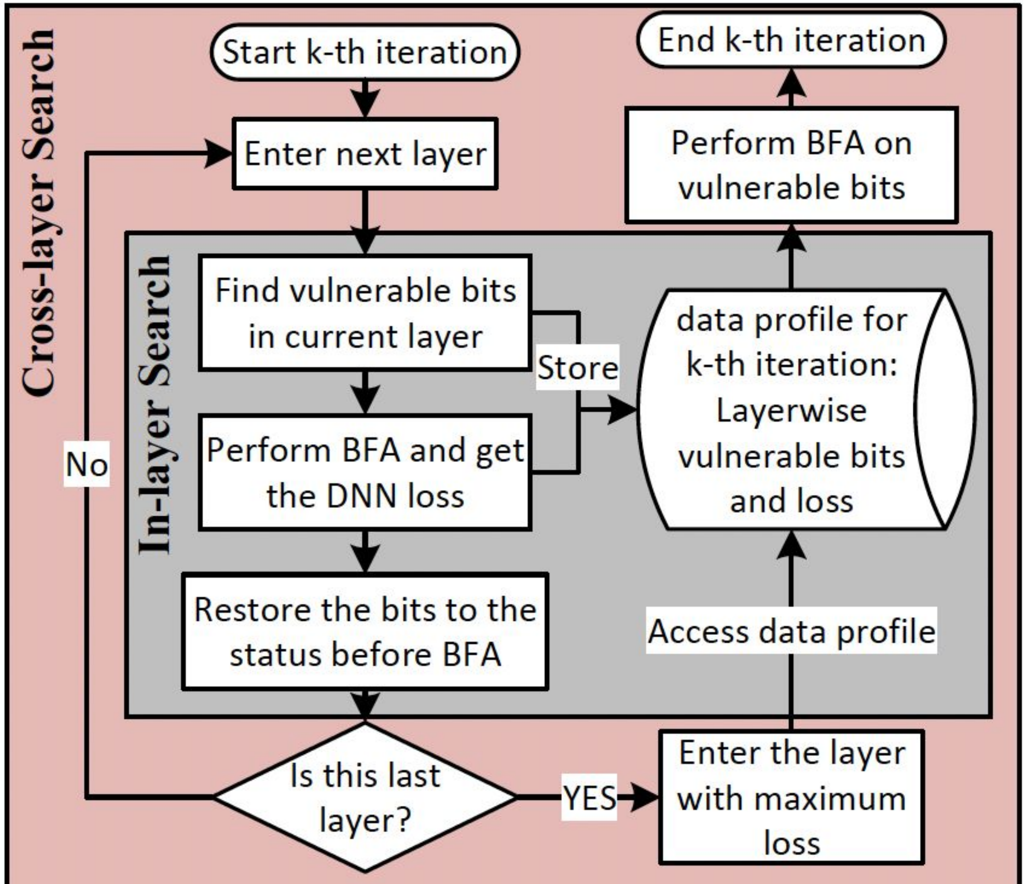
$$\boldsymbol{m} = \boldsymbol{b} \oplus \left(\operatorname{sign}(\nabla_{\boldsymbol{b}}\mathcal{L})/2 + 0.5\right)$$

$$\hat{\boldsymbol{b}} = \boldsymbol{b} \oplus \boldsymbol{m}$$

b_i	$\operatorname{sign}(\partial\mathcal{L}/\partial b_i)$	$\hat{b}_i$	m
0	1 (+)	1	1
0	0 (-)	0	0
1	1 (+)	1	0
1	0 (-)	0	1

UNIVERSITY OF TORONTO

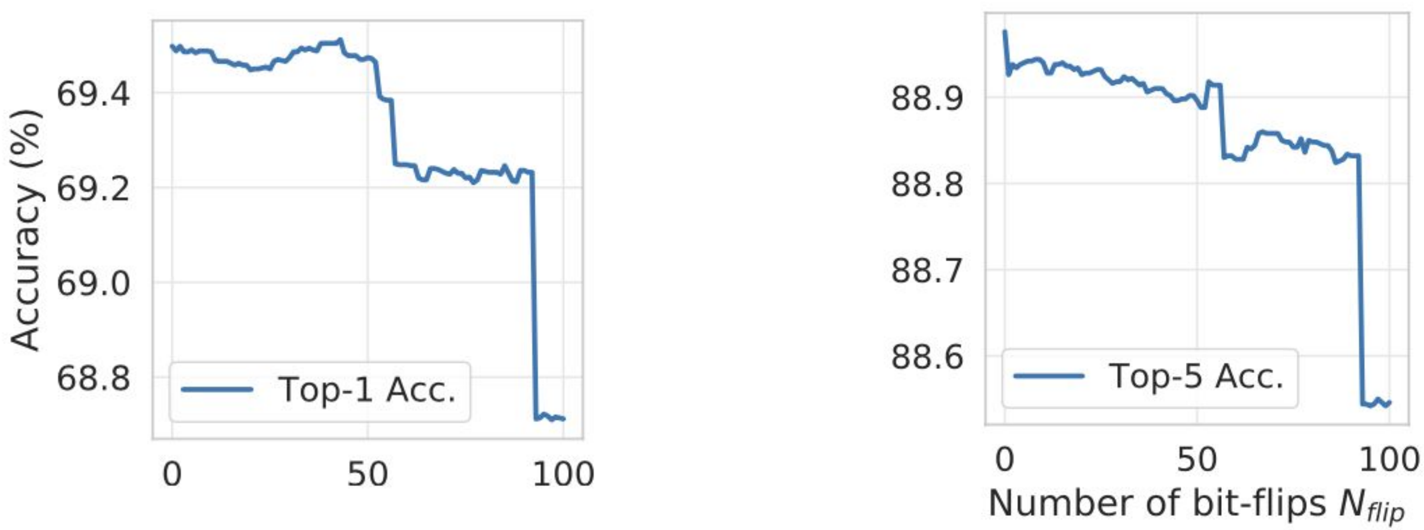
Progressive Bit Search (PBS)



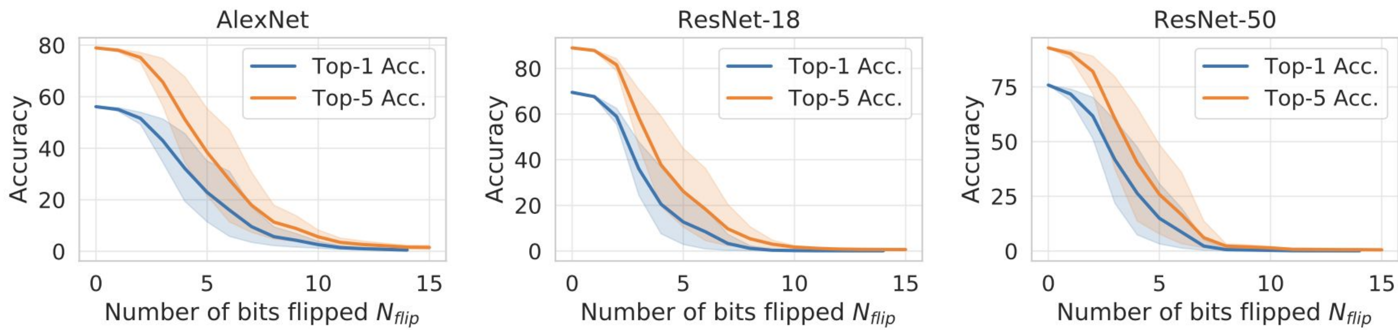
UNIVERSITY OF TORONTO

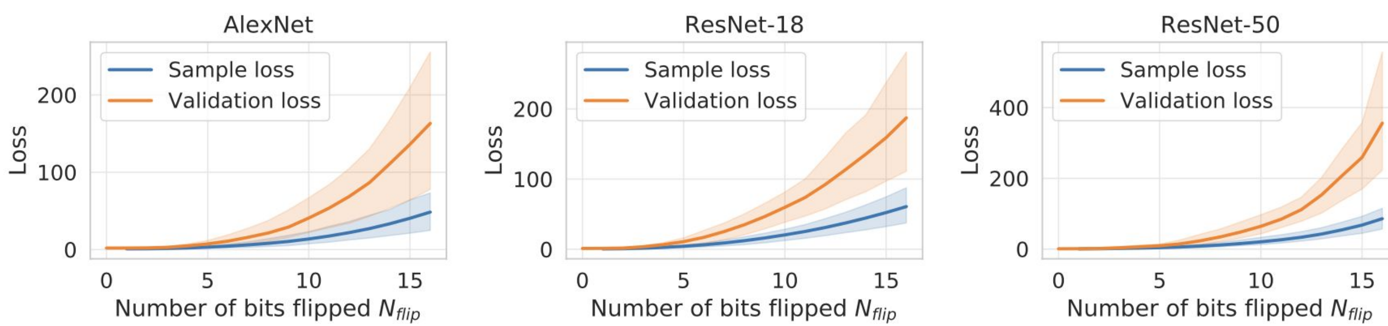
Why We Need PBS

- Most QNNs use 8-bit operations (Google's TPU), robust to weight perturbation
- Random selection does not work well in practise



Experiment Results





Potential Defense

- Train the network with a mixture of clean and adversarial examples.
- Protecting top-N vulnerable bits in model.
- Hardware - based protections against model tampering. (Example - Intel SGX)

Limitations

- There was no information present on what amount of time was required to do such an attack on 93 million bits.
- No accuracy and loss evaluation present for the CIFAR-10 dataset.
- Big assumption that the attacker would have access to weights and gradients. No approach for black box or semi-black box attackers.
- No information on the number of bits flipped in one layer.
- Inconsistencies with ablation study statements and choice of sample size for BFA on ImageNet Dataset.

Future Opportunities

- Study the impact of multiple bit flips in one particular layer.
- Study the optimisation search strategies to use few layers to search instead of the whole network.
- Consider strategies for black box and semi-black box attackers.

2

Sponge Examples: Energy-Latency Attacks on Neural Networks

A Preprint

Ilia Shumailov
University of Cambridge
ilia.shumailov@cl.cam.ac.uk

Yiren Zhao
University of Cambridge
yiren.zhao@cl.cam.ac.uk

Daniel Bates
University of Cambridge
daniel.bates@cl.cam.ac.uk

Nicolas Papernot
University of Toronto and Vector Institute
nicolas.papernot@utoronto.ca

Robert Mullins
University of Cambridge
robert.mullins@cl.cam.ac.uk

Ross Anderson
University of Cambridge
ross.anderson@cl.cam.ac.uk

May 13, 2021

Abstract

The high energy costs of neural network training and inference led to the use of acceleration hardware such as GPUs and TPUs. While such devices enable us to train large-scale neural networks in datacenters and deploy them on edge devices, their designers' focus so far is on average-case performance. In this work, we introduce a novel threat vector against neural networks whose energy consumption or decision latency are critical. We show how adversaries can exploit carefully-crafted **sponge examples**, which are inputs designed to maximise energy consumption and latency, to drive machine learning (ML) systems towards their worst-case performance. Sponge examples are, to our knowledge, the first denial-of-service attack against the ML components of such systems.

We mount two variants of our sponge attack on a wide range of state-of-the-art neural network models, and find that language models are surprisingly vulnerable. Sponge examples frequently increase both

Motivation: The Energy Gap

- To attack the availability of an ML system, one can launch a traditional DoS attack by flooding it with random queries to increase overall memory and CPU consumption
- Can we make this attack more effective by generating inputs that purposely cause high energy consumption and/or latency?
- Typically, the amount of energy consumed in an inference pass depends on (a) number of arithmetic operations (b) number of memory accesses

What kind of examples trigger the worst case performance and have high energy consumption?

Contributions

- Introduces a novel threat vector, **Sponge Examples**, against the availability of ML systems based on energy consumption and latency.
- Sponge examples were shown to increase energy consumption and cause longer runtimes.
- Also turn out to be transferable across hardware platforms and model architectures.

UNIVERSITY OF TORONTO

Attack Model

Threat Model	Capabilities	Goal
White-Box		Significantly increase energy consumption and latency **per query**.
Interactive Black-Box		
Blind Adversary	N/A	

Legend:

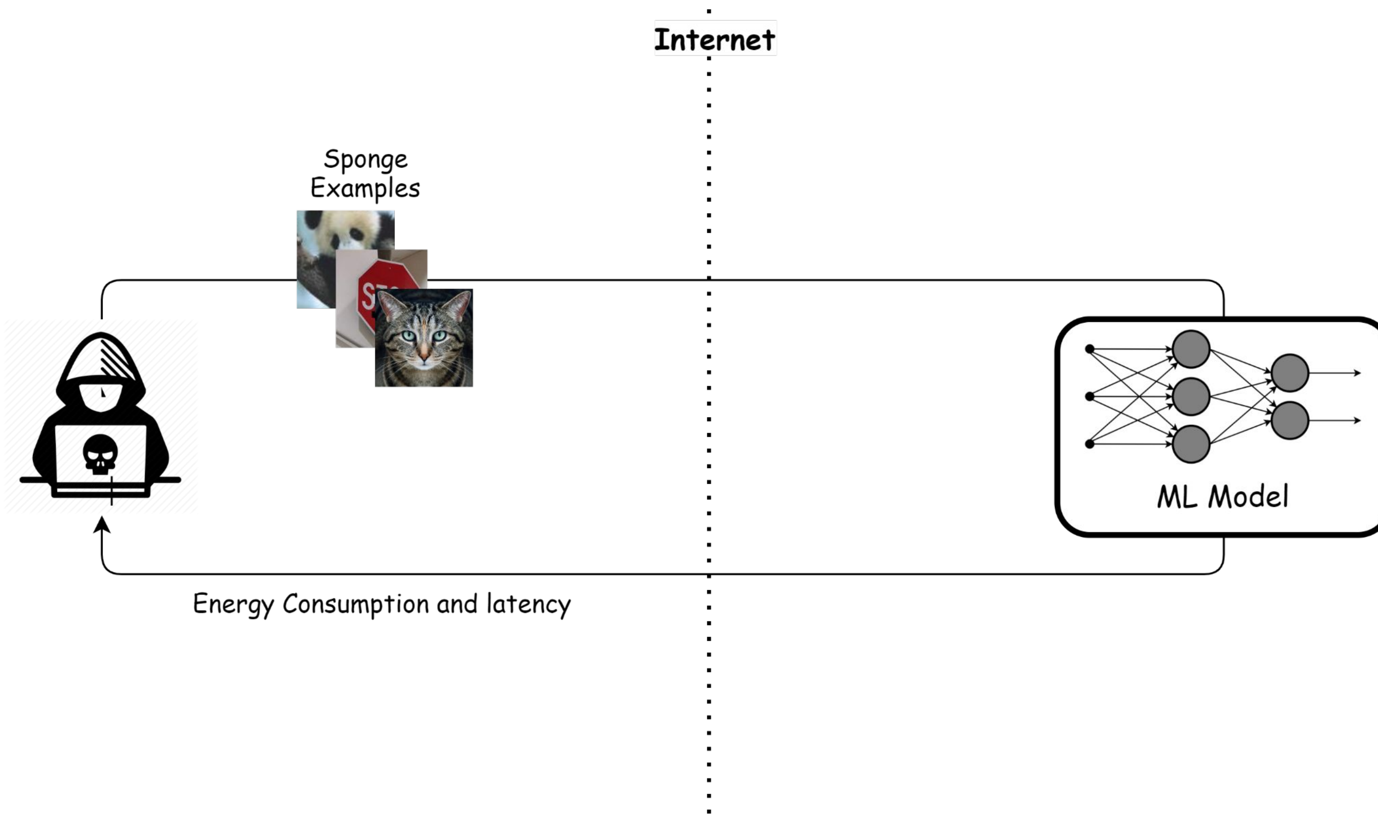
Knowledge of target model’s parameters and architecture.

Measure Energy consumption or time certain operations remotely.

Query model remotely to generate attacks.

UNIVERSITY OF TORONTO

White-box and Interactive Black-box



UNIVERSITY OF TORONTO

Exploitations to Generate Sponge Examples

The paper exploits two dimensions of modern ML models and training infrastructure e.g GPUs to generate sponge examples:

- Computational dimension of NLP models
- Data Sparsity in GPUs

Computational Dimensions of NLP Models

- Modern ML models have a computational dimension

- Internal representation size can can be different for the same input size e.g. tokenization inside Transformer-based translation models.

- In practice, modern translation models map each word to tokens (popular sub-words). Tokens are mapped to embedding vectors.

Computation Dimensions of NLP Models

- Athazagoraphobia => ath, az, agor, aphobia (4 tokens)
- Athazagoraph**p**bia => ath, az, agor, aph, p, bi, a (7 tokens)
- A/h/z/g/r/p/p/i/ => A, /, h, /, z, /, g, /, r, /, p, /, p, /, i, / (16 tokens)

The adversary can increase energy consumption non-linearly with no changes to the input length!

Algorithm 1: Translation Transformer NLP pipeline

Input: Text sentence x
Result: y
$\downarrow \mathrm{O}(l_{\text{tin}})$
x_{tin} = Tokenize(x);
$y_{\text{touts}} = \emptyset$;
$\downarrow \mathrm{O}(l_{\text{ein}})$
x_{ein} = Encode (x_{tin});
$\downarrow \mathrm{O}(l_{\text{tin}} \times l_{\text{ein}} \times l_{\text{tout}} \times l_{\text{eout}})$
while y_{tout} *has no end of sentence token* **do**
  $\downarrow \mathrm{O}(l_{\text{eout}})$
  y_{eout} = Encode (y_{tout});
  $\downarrow \mathrm{O}(l_{\text{ein}} \times l_{\text{eout}})$
  y_{eout} = model.Inference($x_{\text{ein}}, y_{\text{eout}}, y_{\text{touts}}$);
  $\downarrow \mathrm{O}(l_{\text{eout}})$;
  y_{tout} = Decode(y_{eout});
  y_{touts}.add(y_{tout});
end
$\downarrow \mathrm{O}(l_{\text{tout}})$;
y = Detokenize(y_{touts})

UNIVERSITY OF TORONTO

Data Sparsity in ML Models

- Modern DNNs use rectified linear units (ReLU) as the activation function.
- Therefore, when the input to neuron is negative, the output is 0.
- ASIC chips and GPUs can take advantage of this sparsity by employing zero-skipping multiplications.
- Therefore, inputs that lead to less sparse activations will potentially increase energy consumption and/or latency

Genetic Algorithms to Generate Sponge Examples

Genetic algorithms allow us to optimize objectives with no gradient information.

- You typically start with a pool of random samples and iteratively evolve them.

- After each "evolution", obtain a fitness score (energy consumption).

- Use top 10% of samples as parents for next iteration.

- Repeat until samples become good enough.

Evolving Sponge Examples

- **Computer Vision Examples:** Sample two parents A and B from the population pool, then crossover the inputs using a random mask

$$A * \text{mask} + (1 - \text{mask}) * B$$

- **NLP Tasks:** Crossover samples A and B by concatenating the left part of A with the right part of B. Then, probabilistically invert the two parts.

Next, randomly perturb some of the input features (i.e. pixels or words) of the children.

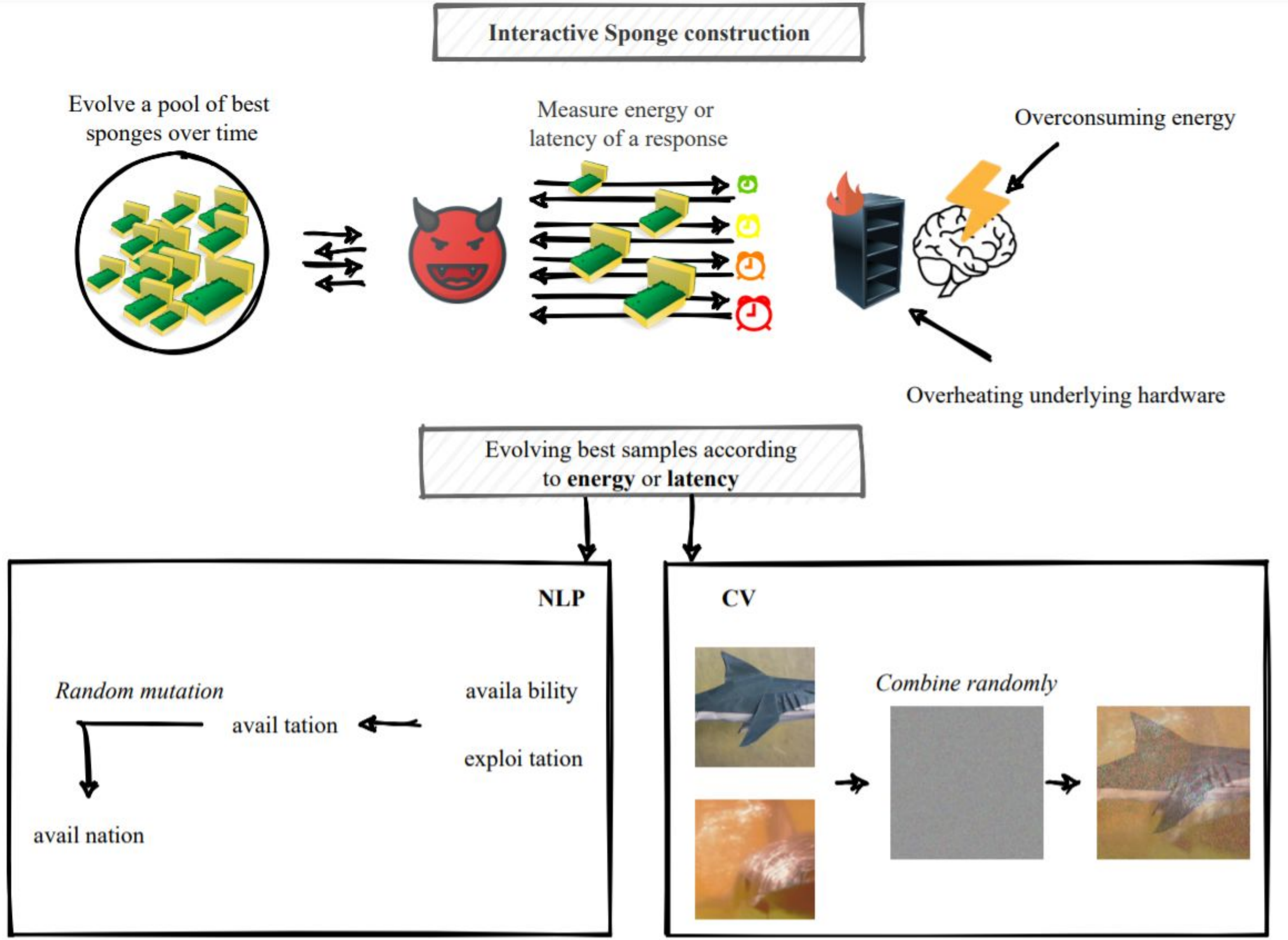
UNIVERSITY OF TORONTO

Measuring Fitness Scores

The paper tests 2 variants of GA which differ in how we measure fitness:

- **White-box Setting:** Estimated energy cost based on the run-time sparsity, i.e. number of operations based on the structure and parameters of the neural network. Requires access to model parameters.

- **Black-box Setting:** Use purely the measured hardware cost as the fitness, i.e. latency or energy consumption



Interactive Sponge construction
Evolve a pool of best sponges over time
Measure energy or latency of a response
Overconsuming energy
Overheating underlying hardware
Evolving best samples according to energy or latency
NLP
Random mutation
availa bility
avail tation
exploi tation
avail nation
CV
Combine randomly

L-BFGS in the White-box Setting

Use L-BFGS algorithm to optimize

$$-\sum_{a_l \in A} \|a_l\|_2$$

Where *a_l* represents activations at layer *l*

That is, aim to increase density to prevent hardware-level optimizations e.g. zero-skipping multiplications

UNIVERSITY OF TORONTO

Models, Datasets, and Experiments

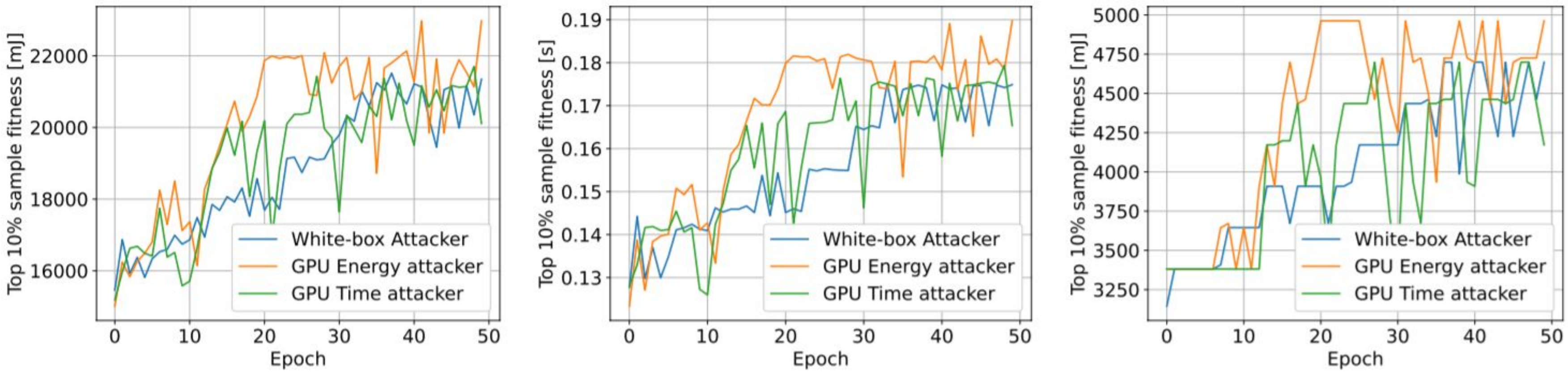
- **NLP:**
    - RoBERTa Model . Trained on SuperGLUE for language understanding
    - Transformer-based based model trained on translation tasks (WMT)
- **Computer Vision:**
    - Range of ResNet and MobileNet models
    - Trained on ImageNet-2017.

Sponge attacks were tested on GPUs, ASIC chips, and CPUs.

UNIVERSITY OF TORONTO

	Input size	GPU Energy [mJ] Natural	GPU Energy [mJ] Random	GPU Energy [mJ] Sponge	ASIC Energy [mJ] Natural	ASIC Energy [mJ] Random	ASIC Energy [mJ] Sponge	GPU Time [mS] Natural	GPU Time [mS] Random	GPU Time [mS] Sponge
SuperGLUE Benchmark with [60]										
CoLA	15	2865.68	3023.705	3170.38	504.93	566.58	583.56	0.02	0.02	0.02
		1.00×	1.06×	**1.11**×	1.00×	1.12×	**1.16**×	**1.00**×	0.92×	0.92×
	30	3299.07	4204.121	4228.22	508.73	634.24	669.20	0.03	0.03	0.02
		1.00×	1.27×	**1.28**×	1.00×	1.25×	**1.32**×	**1.00**×	0.93×	0.82×
	50	3384.62	6310.504	6988.57	511.43	724.48	780.57	0.03	0.04	0.04
		1.00×	1.86×	**2.06**×	1.00×	1.42×	**1.53**×	1.00×	1.23×	**1.27**×
MNLI	15	3203.01	3573.93	3597.3	509.19	570.10	586.43	0.03	0.03	0.03
		1.00×	1.12×	**1.12**×	1.00×	1.12×	**1.15**×	1.00×	**1.01**×	0.95×
	30	3330.22	4752.84	5045.25	514.00	638.78	672.07	0.03	0.03	0.03
		1.00×	1.43×	**1.51**×	1.00×	1.24×	**1.31**×	1.00×	**1.06**×	1.03×
	50	3269.34	6373.507	7051.68	519.51	728.82	783.18	0.03	0.04	0.04
		1.00×	1.95×	**2.16**×	1.00×	1.40×	**1.51**×	1.00×	1.28×	**1.30**×
WSC	15	4287.24	13485.49	38106.98	510.84	1008.59	2454.89	0.04	0.07	0.20
		1.00×	3.15×	**8.89**×	1.00×	1.97×	**4.81**×	1.00×	2.02×	**5.51**×
	30	4945.47	36984.44	79786.57	573.78	2319.05	5012.75	0.04	0.20	0.46
		1.00×	7.48×	**16.13**×	1.00×	4.04×	**8.74**×	1.00×	4.89×	**11.04**×
	50	6002.68	81017.01	159925.23	716.96	5093.42	10192.41	0.05	0.46	0.93
		1.00×	13.50×	**26.64**×	1.00×	7.10×	**14.22**×	1.00×	10.16×	**20.56**×
WMT14/16 with [64]										
En→Fr	15	9492.30	25772.89	40975.78	1793.84	4961.56	8494.36	0.10	0.24	0.37
		1.00×	2.72×	**4.32**×	1.00×	2.77×	**4.74**×	1.00×	2.51×	**3.89**×
En→De	15	8573.59	13293.51	238677.16	1571.59	2476.18	48446.29	0.09	0.13	2.09
		1.00×	1.55×	**27.84**×	1.00×	1.58×	**30.83**×	1.00×	1.46×	**24.18**×
WMT18 with [65]										
En→De	15	28393.97	38493.96	874862.97	1624.05	2318.50	49617.68	0.27	0.33	7.25
		1.00×	1.36×	**30.81**×	1.00×	1.43×	**30.55**×	1.00×	1.20×	**26.49**×

Evolution of Sponge Attacks



(a) GPU Energy (b) GPU Time (c) ASIC Energy

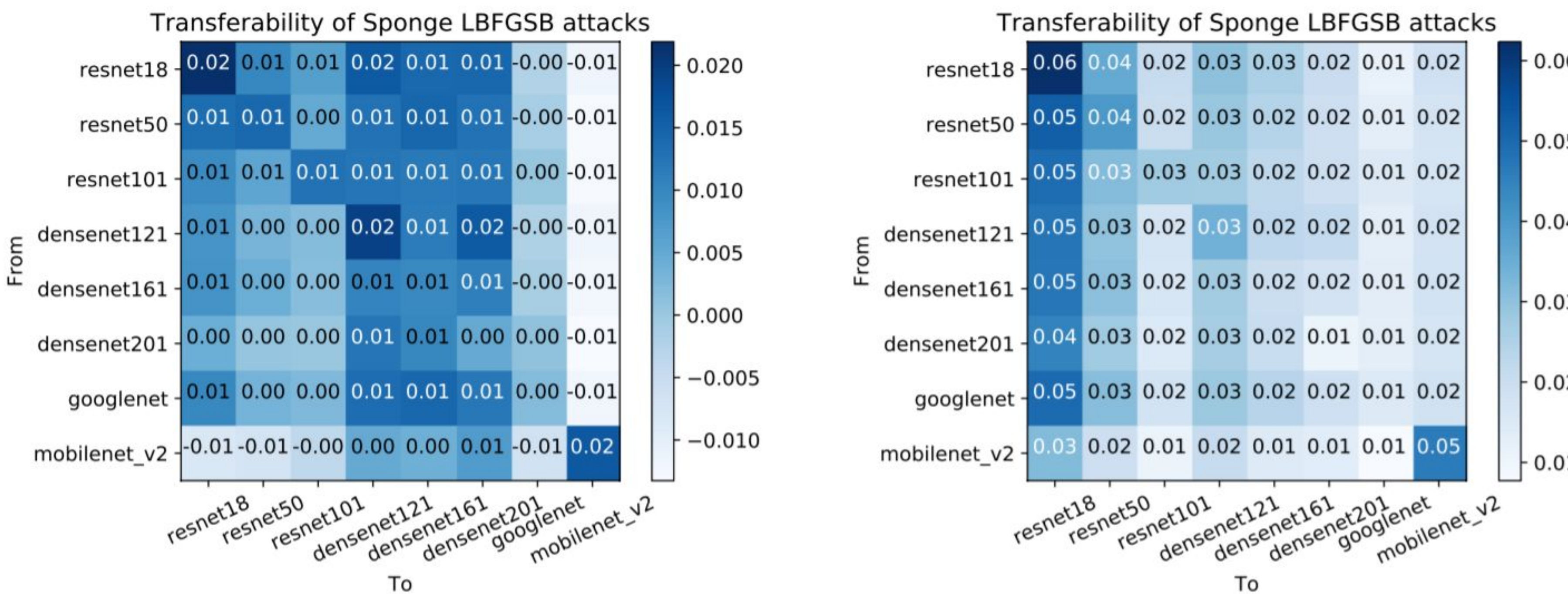
Figure 2: Black-box attack performance of sponge examples on different hardware metrics against English-to-French translation model [65]. We show two Black-box attackers (GPU Energy and GPU Time attacker) and one White-box attacker, all using GA as the optimisation for finding sponge examples.

UNIVERSITY OF TORONTO

From	To		ASIC Energy [mJ]	GPU Time [S]	GPU Energy [mJ]	CPU Time [S]	CPU Energy [mJ]
Black-box							
		Sponge	3648.219	0.174	17251.000	1.048	51512.966
	$\mathrm{WMT14}_{en \to fr}$ [64]	Natural	1450.403	0.053	6146.550	0.537	23610.145
			2.52×	**3.27**×	**2.81**×	1.95×	2.18×
		Sponge	2909.245	0.414	47723.500	3.199	181936.595
$\mathrm{WMT16}_{en \to de}$ [64]	$\mathrm{WMT18}_{en \to de}$ [65]	Natural	1507.364	0.253	27265.250	1.344	71714.201
			1.93×	1.64×	1.75×	**2.38**×	**2.54**×
		Sponge	3875.365	0.652	67183.100	4.409	247585.091
	$\mathrm{WMT19}_{en \to ru}$ [66]	Natural	1654.965	0.215	25033.620	2.193	121210.376
			2.34×	**3.03**×	**2.68**×	2.01×	2.04×
White-box							
		Sponge	48447.093	2.414	260187.900	13.615	781758.680
$\mathrm{WMT16}_{en \to de}$ [64]	$\mathrm{WMT16}_{en \to de}$ [64]	Natural	1360.118	0.056	6355.620	0.520	23262.311
			35.62×	**42.98**×	**40.94**×	26.20×	33.61×

		Energy		Density		
		ASIC Energy [mJ]	Energy ratio	Post-ReLU	Overall	Maximum
ImageNet						
ResNet-18	Sponge LBFGS	53.359 ± 0.004	0.899	0.685	**0.896**	0.981
	Sponge	51.816 ± 0.271	0.873	0.599	0.869	
	Natural	51.745 ± 0.506	0.871	0.596	0.869	
	Random	49.685 ± 0.008	0.837	0.480	0.834	
ResNet-50	Sponge LBFGS	164.727 ± 0.062	0.863	0.619	**0.885**	0.998
	Sponge	160.887 ± 0.609	0.843	0.562	0.868	
	Natural	160.573 ± 1.399	0.842	0.572	0.867	
	Random	155.819 ± 0.016	0.817	0.483	0.845	
ResNet-101	Sponge LBFGS	258.526 ± 0.028	0.857	0.597	**0.873**	0.994
	Sponge	254.182 ± 0.561	0.842	0.556	0.861	
	Natural	253.004 ± 1.345	0.839	0.545	0.857	
	Random	249.026 ± 0.036	0.825	0.507	0.846	
DenseNet-121	Sponge LBFGS	152.595 ± 0.050	0.783	0.571	**0.826**	0.829
	Sponge	149.564 ± 0.502	0.767	0.540	0.814	
	Natural	147.247 ± 1.199	0.755	0.523	0.804	
	Random	144.366 ± 0.036	0.741	0.487	0.792	
DenseNet-161	Sponge LBFGS	288.427 ± 0.087	0.726	0.435	**0.764**	0.811
	Sponge	287.153 ± 0.575	0.723	0.429	0.761	
	Natural	282.296 ± 2.237	0.711	0.404	0.751	
	Random	279.270 ± 0.065	0.703	0.387	0.744	
DenseNet-201	Sponge LBFGS	237.745 ± 0.156	0.756	0.505	0.788	0.863
	Sponge	239.845 ± 0.522	0.763	0.519	**0.794**	
	Natural	234.886 ± 1.708	0.747	0.487	0.781	
	Random	233.699 ± 0.098	0.743	0.479	0.777	

Transferability of Sponge Attacks



(a) Sponge density - Normal density

(b) Sponge density - Random density

Figure 4: Transferability of sponge examples across different computer vision benchmarks.

Simple Defence against Sponge Attacks

- Measure average energy consumption/latency of natural examples.

- Set a cut-off threshold so that maximum energy consumption **per query** is under control.

- Examples that exceed threshold are stopped.

UNIVERSITY OF TORONTO

Limitations

- Sponge attacks were not so effective on CV tasks compared to NLP, especially on GPUs.
  - Could be due to GA not performing well in high dimensional spaces i.e images, thus not generating good enough samples.
  - Perhaps data sparsity is not the only optimization that can be exploited in CV tasks?
- Ignores a pre-processing stage that can happen before model inference i.e image filtering
- Proposed techniques do not generate stealthy examples, can possibly be detected by outlier detectors.
- No available code, yet.
- Future work:
  - Extend results to other hardware (e.g TPUs).
  - More advanced algorithms to generate sponge attacks (reinforcement learning?)
- GPUs usually process examples in batches, how is the cut-off threshold enforced per example?

UNIVERSITY OF TORONTO

3

MANIPULATING SGD WITH DATA ORDERING ATTACKS

A PREPRINT

Ilia Shumailov
University of Cambridge

Zakhar Shumaylov
University of Cambridge

Dmitry Kazhdan
University of Cambridge

Yiren Zhao
University of Cambridge

Nicolas Papernot
University of Toronto & Vector Institute

Murat A. Erdogdu
University of Toronto & Vector Institute

Ross Anderson
University of Cambridge & University of Edinburgh

June 8, 2021

ABSTRACT

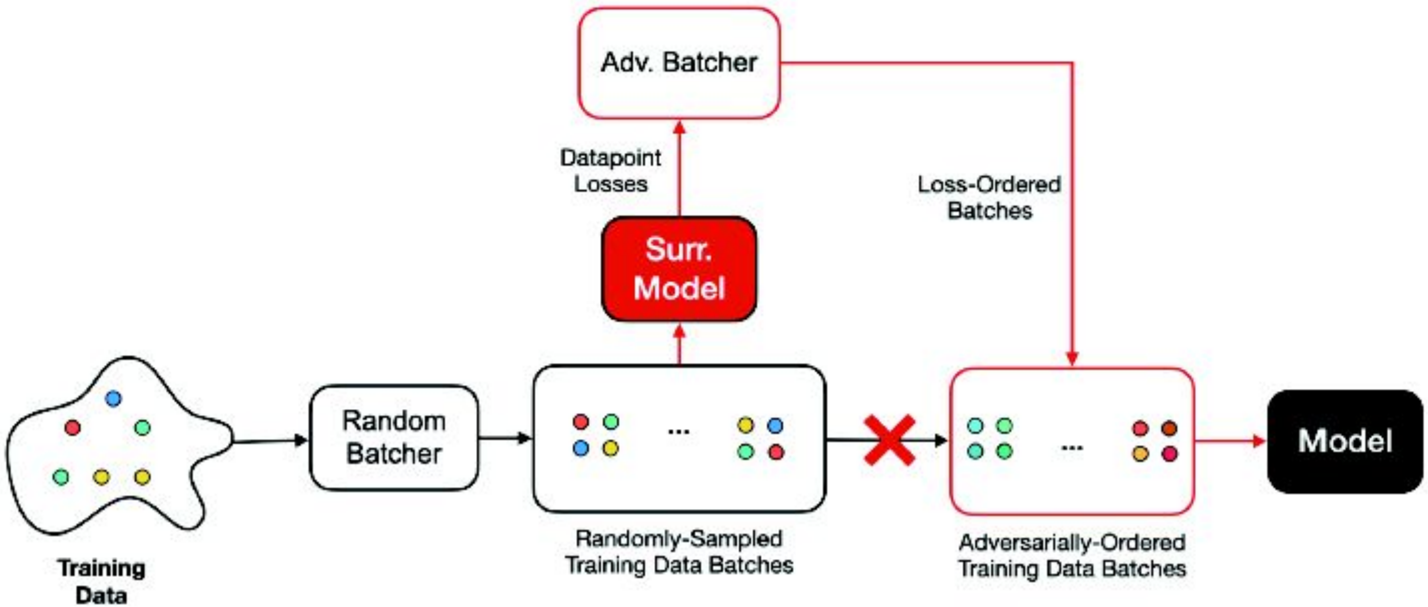
Machine learning is vulnerable to a wide variety of attacks. It is now well understood that by changing the underlying data distribution, an adversary can poison the model trained with it or introduce backdoors. In this paper we present a novel class of training-time attacks that require no changes to the underlying dataset or model architecture, but instead only change the order in which data are supplied to the model. In particular, we find that the attacker can either prevent the model from learning, or poison it to learn behaviours specified by the attacker. Furthermore, we find that even a single adversarially-ordered epoch can be enough to slow down model learning, or even to reset all of the learning progress. Indeed, the attacks presented here are not specific to the model or dataset, but rather target the stochastic nature of modern learning procedures. We extensively evaluate our attacks on computer vision and natural language benchmarks to find that the adversary can disrupt model training and even introduce backdoors.

Setting Up The Stage

	Prior Work	Present Work
Attacks on integrity	Common beliefs : poisoning attacks require manipulation of data/labels in training	Focuses on using clean data and labels -- manipulation at batching stages
Attacks on Availability	A focus on availability during inference.	The focus is on availability at training time.

UNIVERSITY OF TORONTO

The Threat Model



Threat Model

01 **Assumptions on the attacker**

- Blackbox attacker : no access to the model
- Whitebox attacker : access to the model
- No assumptions on knowledge of the data for both.

02 **Why are these reasonable?**

- OS handing file system requests
- Disk handling individual data accesses
- Software for random data sampling
- Distributed storage manager
- ML pipeline

On Stochastic learning and batching

- Assume that loss function is defined as sample average per training data point in k-th batch

$$\hat{L}_{k+1}(\theta) = \frac{1}{B} \sum_{i=kB+1}^{kB+B} L_i(\theta)$$

- With N*B being the total number of items for training, for each epoch we estimate

$$\hat{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \hat{L}_i(\theta)$$

- SGD of these samples with learning rate eta is thus the following

$$\theta_{k+1} \leftarrow \theta_k - \eta \nabla_\theta \hat{L}_k(\theta_k)$$

On Stochastic learning and batching ctd.

- The stochasticity of SGD is owed to batch sampling
- Assuming an unbiased sampling procedure we have that

$$\mathbb{E}[\nabla \hat{L}_{i_k}(\theta)] = \sum_{i=1}^{N} \mathbb{P}(i_k = i) \nabla \hat{L}_i(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla \hat{L}_i(\theta) = \nabla \hat{L}(\theta).$$

- Observe that this is true in expectation, and for individual batches the story may be different, which gives rise to the exploits that anchor this work.

Introducing the Vulnerability

Consider the effect of N SGD steps in one epoch

$$\theta_{N+1} = \theta_1 - \eta \nabla \hat{L}_1(\theta_1) - \eta \nabla \hat{L}_2(\theta_2) - \cdots - \eta \nabla \hat{L}_N(\theta_N)$$

$$= \theta_1 - \eta \sum_{j=1}^{N} \nabla \hat{L}_j(\theta_1) + \eta^2 \overbrace{\sum_{j=1}^{N} \sum_{k<j} \nabla\nabla \hat{L}_j(\theta_1) \nabla \hat{L}_k(\theta_1) + O(N^3 \eta^3)}^{\text{data order dependent}} .$$

The order dependence presents an opening to mount attacks during the training phase.

UNIVERSITY OF TORONTO

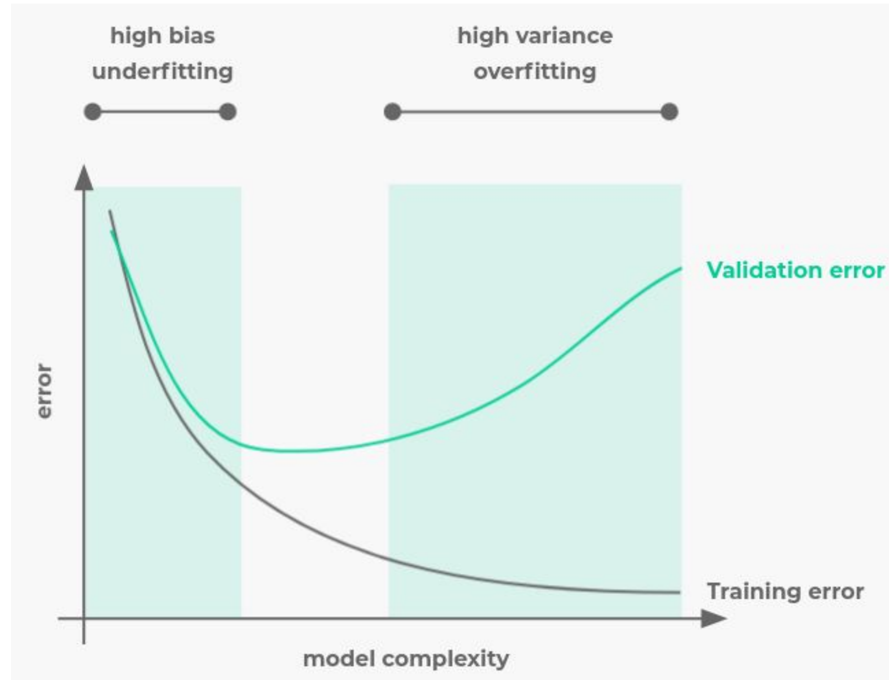
Intuition

- The name of the game, with these kinds of attacks, is the following:

  a) promoting memorisation,

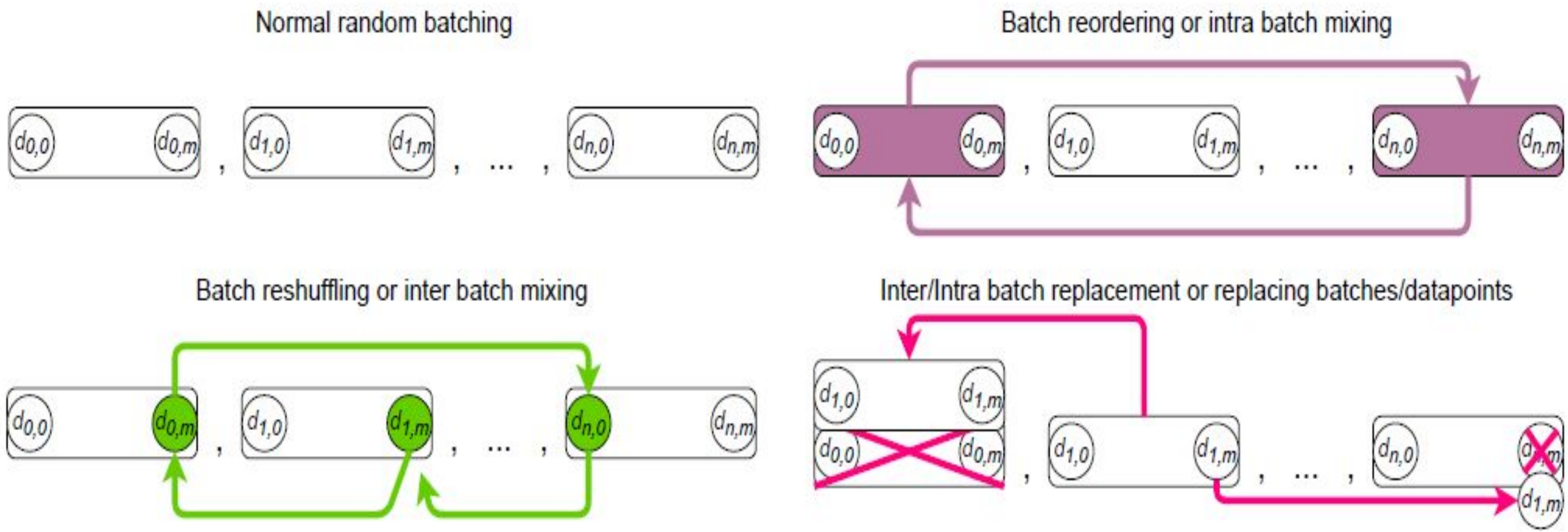
  b) and promoting overfitting.

- We are thus forcing the model to forget generalisable features.



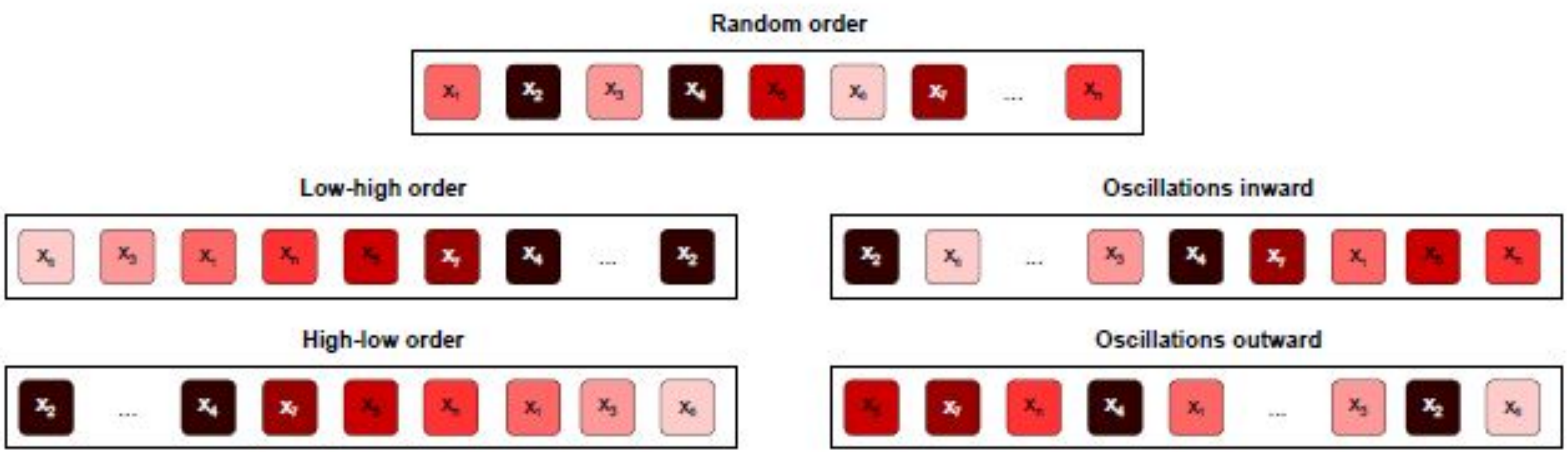
UNIVERSITY OF TORONTO

The Taxonomy of Batching Attacks



UNIVERSITY OF
TORONTO

Loss Based Ordering



Algorithm for Batch Reordering, Reshuffling, and Replacing (BRRR)

Algorithm 1: A high level description of the BRRR attack algorithm

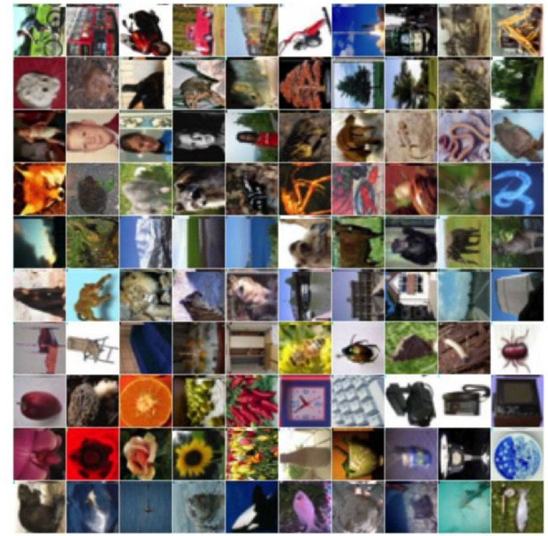
```
/* -- Attack preparation: collecting data --
do
    get a new batch and add it to a list of unseen datapoints;
    train surrogate model on a batch and pass it on to the model;
while first epoch is not finished
/* -- Attack: reorder based on surrogate loss --
while training do
    rank each data point from epoch one with a surrogate loss;
    reorder the data points according to the attack strategy;
    pass batches to model and train the surrogate at the same time.
```

UNIVERSITY OF TORONTO

Datasets

The paper uses the following datasets :

- CIFAR-10;
- CIFAR-100;
- AGNews datasets.

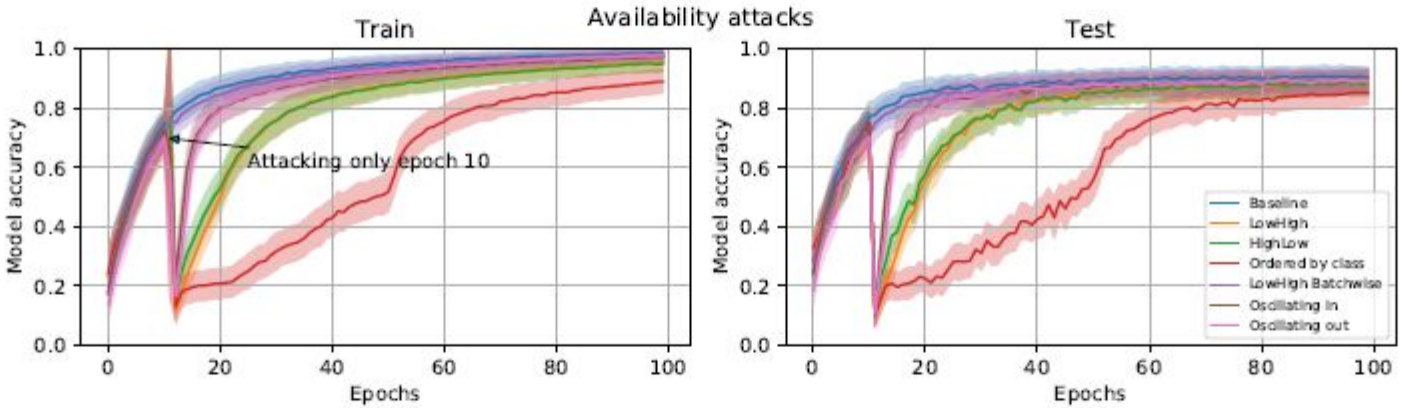


Topic:
Sci/Tech

Title:
Your PC May Be Less Secure Than You Think

Description:
Most users think their computer is safe from adware and spyware--but they're wrong. A survey conducted by Internet service provider America Online found that 20 percent of home computers were infected by

Some results on Availability



This tells us that even one epoch is sufficient to either reset learning or slow it down significantly. In fact, one epoch is enough to degrade the training for more than 90 epochs.

UNIVERSITY OF TORONTO

More Results on Availability

Attack	CIFAR-10 Train acc	CIFAR-10 Test acc	CIFAR-10 Δ	CIFAR-100 Train acc	CIFAR-100 Test acc	CIFAR-100 Δ	AGNews Train acc	AGNews Test acc	AGNews Δ
Baseline									
None	95.51	90.51	−0.0%	99.96	75.56	−0.0%	93.13	90.87	−0.0%
Batch reshuffle									
Oscillation outward	17.44	26.13	**−64.38%**	99.80	18.00	**−57.56%**	97.72	65.85	−25.02%
Oscillation inward	22.85	28.94	−61.57%	99.92	31.38	−44.18%	94.06	89.23	−1.64%
High Low	23.39	31.04	−59.47%	99.69	21.15	−54.41%	94.38	56.54	**−34.33%**
Low High	20.22	30.09	−60.42%	96.07	20.48	−55.08%	98.94	59.28	−31.59%
Batch reorder									
Oscillation outward	99.37	78.65	−11.86%	100.00	53.05	−22.51%	95.37	90.92	+0.05%
Oscillation inward	99.60	78.18	**−12.33%**	100.00	51.78	−23.78%	96.29	91.10	+0.93%
High Low	99.44	79.65	−10.86%	100.00	51.48	**−24.08%**	96.16	91.80	+0.05%
Low High	99.58	79.07	−11.43%	100.00	54.04	−21.52%	94.02	90.35	−0.52%

UNIVERSITY OF TORONTO

Efficacy of Reordering

Attack	Batch size	CIFAR-10 Train Loss	CIFAR-10 Train Accuracy	CIFAR-10 Test Loss	CIFAR-10 Test Accuracy	CIFAR-10 Δ	CIFAR-100 Train Loss	CIFAR-100 Train Accuracy	CIFAR-100 Test Loss	CIFAR-100 Test Accuracy	CIFAR-100 Δ
Baseline											
None	32	0.13	95.51	0.42	90.51	−0.0%	0.00	99.96	2.00	75.56	−0.0%
	64	0.09	96.97	0.41	90.65	−0.0%	0.00	99.96	2.30	74.05	−0.0%
	128	0.07	97.77	0.56	89.76	−0.0%	0.00	99.98	1.84	74.45	−0.0%
Batch reorder (only epoch 1 data)											
Oscillation outward	32	0.02	99.37	2.09	78.65	−11.86%	0.00	100.00	5.24	53.05	−22.51%
	64	0.01	99.86	2.39	78.47	−12.18%	0.00	100.00	4.53	55.91	−18.14%
	128	0.01	99.64	2.27	77.52	−12.24%	0.00	100.00	3.22	52.13	−22.32%
Oscillation inward	32	0.01	99.60	2.49	78.18	**−12.33%**	0.00	100.00	5.07	51.78	−23.78%
	64	0.01	99.81	2.25	79.59	−11.06%	0.00	100.00	4.70	55.05	−19.0%
	128	0.02	99.39	2.23	76.13	−13.63%	0.00	100.00	3.46	52.66	−21.79%
High Low	32	0.02	99.44	2.03	79.65	−10.86%	0.00	100.00	5.47	51.48	**−24.08%**
	64	0.02	99.50	2.39	77.65	**−13.00%**	0.00	100.00	5.39	55.63	−18.42%
	128	0.02	99.47	2.80	74.73	**−15.03%**	0.00	100.00	3.36	53.63	−20.82%
Low High	32	0.01	99.58	2.33	79.07	−11.43%	0.00	100.00	4.42	54.04	−21.52%
	64	0.01	99.61	2.40	76.85	−13.8%	0.00	100.00	3.91	54.82	**−19.23%**
	128	0.01	99.57	1.88	79.82	−9.94%	0.00	100.00	3.72	49.82	**−24.63%**
Batch reorder (resampled data every epoch)											
Oscillation outward	32	0.11	96.32	0.41	90.20	−0.31%	0.01	99.78	2.22	72.38	**−3.18%**
	64	0.11	96.40	0.45	89.12	−1.53%	0.01	99.76	2.20	73.33	−0.72%
	128	0.09	96.89	0.47	89.71	**−0.05%**	0.00	99.89	1.95	74.21	−0.24%
Oscillation inward	32	0.15	95.11	0.44	89.56	−0.95%	0.00	99.88	2.10	74.80	−0.76%
	64	0.12	96.11	0.42	89.98	−0.67%	0.01	99.81	2.35	72.24	**−1.81%**
	128	0.09	96.88	0.43	90.09	+0.33%	0.00	99.93	2.24	73.72	−0.73%
High Low	32	0.12	95.95	0.45	89.38	**−1.13%**	0.01	99.84	2.07	74.88	−0.68%
	64	0.15	94.80	0.44	89.01	**−1.64%**	0.01	99.81	2.27	74.63	−0.58%
	128	0.11	96.33	0.48	89.71	**−0.05%**	0.00	99.92	2.13	73.90	−0.55%
Low High	32	0.10	96.63	0.47	90.29	−0.22%	0.01	99.77	2.07	73.90	−1.66%
	64	0.12	96.10	0.50	89.34	−1.31%	0.01	99.68	2.26	72.73	−1.32%
	128	0.09	97.16	0.49	89.85	+0.09%	0.00	99.94	2.31	71.96	**−2.49%**

UNIVERSITY OF TORONTO

Poisoning and backdooring

$$\min_{X_i} \left\| \nabla_\theta \hat{L}(\hat{X}_j, \theta_k) - \nabla_\theta \hat{L}(X_i, \theta_k) \right\|^p ; \quad \text{s.t.} \quad X_i \in X.$$

Done via...

Using natural data to create adversarial updates :

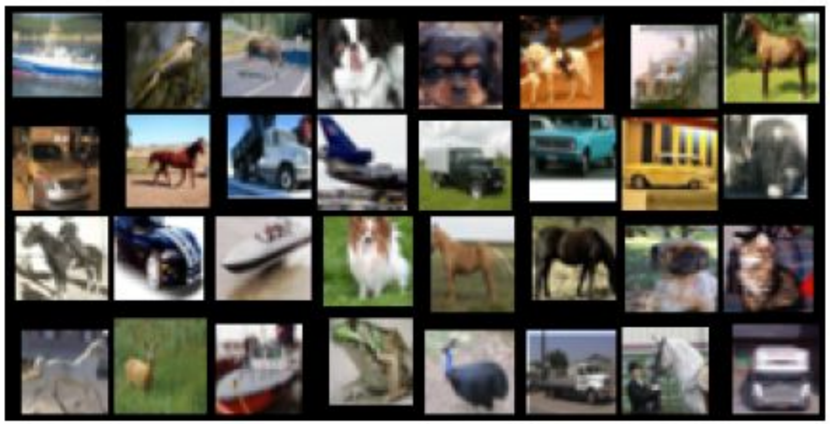
$$\theta_{k+1} = \theta_k + \eta \hat{\Delta}\theta_k, \text{ where } \begin{cases} \hat{\Delta}\theta_k = -\nabla_\theta \hat{L}(X_i, \theta_k) \\ \nabla_\theta \hat{L}(X_i, \theta_k) \approx \nabla_\theta \hat{L}(\hat{X}_k, \theta_k). \end{cases}$$

Natural data

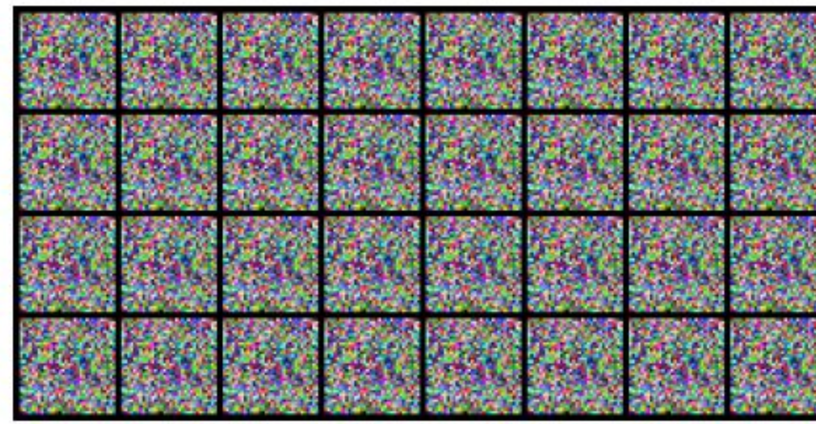
Adversarial data

An Example of Data and Poisoned Batches

With the previously mentioned procedure, we can demonstrate poisoning of a model without ever showing adversarial data.



(a) Natural image batch



(b) Poison datapoint batch

Summarised Results on Integrity Attacks

Trigger	Batch size	Train acc [%]	Test acc [%]	Trigger acc [%]	Error with trigger [%]
Baselines					
Random natural data	32	88.43 ± 7.26	79.60 ± 1.49	10.91 ± 1.53	30.70 ± 2.26
	64	95.93 ± 2.11	81.31 ± 2.01	9.78 ± 1.25	27.38 ± 1.20
	128	94.92 ± 2.04	81.69 ± 1.17	10.00 ± 2.26	27.91 ± 1.41
Data with trigger perturbation	32	96.87 ± 2.79	73.28 ± 2.93	99.65 ± 0.22	89.68 ± 0.21
	64	98.12 ± 1.53	79.45 ± 1.39	99.64 ± 0.21	89.64 ± 0.21
	128	98.67 ± 0.99	80.51 ± 1.10	99.67 ± 0.40	89.65 ± 0.39
Only reordered natural data					
9 white lines trigger	32	88.43 ± 6.09	78.02 ± 1.50	$\mathbf{33.93} \pm 7.37$	40.78 ± 5.70
	64	95.15 ± 2.65	82.75 ± 0.86	25.02 ± 3.78	33.91 ± 2.28
	128	95.23 ± 2.24	82.90 ± 1.50	21.75 ± 4.49	31.75 ± 3.68
Blackbox 9 white lines trigger	32	88.43 ± 4.85	80.84 ± 1.20	$\mathbf{17.55} \pm 3.71$	33.64 ± 2.83
	64	93.59 ± 3.15	82.64 ± 1.64	16.59 ± 4.80	30.90 ± 3.08
	128	94.84 ± 2.24	81.12 ± 2.49	16.19 ± 4.01	31.[illegible]3 ± 3.73
Flag-like trigger	32	90.93 ± 3.81	78.46 ± 1.04	$\mathbf{91.03} \pm 12.96$	87.08 ± 2.71
	64	96.87 ± 1.21	82.95 ± 0.72	77.10 ± 16.96	82.92 ± 3.89
	128	95.54 ± 1.88	82.28 ± 1.50	69.49 ± 20.66	82.09 ± 3.78
Blackbox flag-like trigger	32	86.25 ± 4.00	80.16 ± 1.91	56.31 ± 19.57	78.78 ± 3.51
	64	95.00 ± 2.18	83.41 ± 0.94	48.75 ± 23.28	78.11 ± 4.40
	128	93.82 ± 2.27	81.54 ± 1.94	$\mathbf{68.07} \pm 18.55$	81.23 ± 3.80

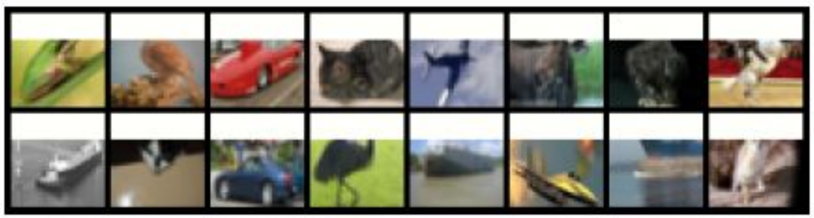
Please, direct your attention to these.

UNIVERSITY OF
TORONTO

A Remark on the Results and Triggers

- In the previous page we see that the trigger accuracy for a) is higher than that of b).
- This seems to suggest that performance seems to differ based on how subtle the filter seems (perhaps this relates to how subtle the gradient is that needs to be replicated?

(a) Flag-like trigger



(b) 9 white lines trigger

UNIVERSITY OF TORONTO

Taxonomy of training time integrity attacks.

Attack	Dataset knowledge	Model knowledge	Model specific	Changing dataset	Adding data	Adding perturbations
Batch Reorder	✗	✗	✗	✗	✗	✗
Batch Reshuffle	✗	✗	✗	✗	✗	✗
Batch Replace	✗	✗	✗	✗	✗	✗
Adversarial initialisation [10]	✗	✓	✓	✗	✗	✗
BadNets [11]	✓	✗	✗	✓	✗	✓
Dynamic triggers [24]	✓	✓	✗	✓	✗	✓
Poisoned frogs [28]	✓	✓	✗	✓	✗	✓

UNIVERSITY OF TORONTO

Limitations and opportunities for further research

Limitations of this work are as follows :

- While the work does show that one can mount attacks using clean data, getting control of the flow of data to enable this is not a trivial step. The promised attack surface is large, but practical ways to leverage these methods are not fully explored.
- The paper also doesn't address why batch reordering seems to be weak on integrity attacks on the 3rd data set.
- The network doesn't evaluate how gradient replication through ordering could be used for availability attacks or integrity attacks

Possible directions forward include research on :

- implications of the findings to fairness.
- inductive bias and the practical contribution of pseudorandom sampling.
- Extensions of gradient mimicking