MACMul: Protecting NNs from Rowhammer Attacks with Apache TVM

Luc Baier-Reinio

Background

- A few critical bit-flips can significantly degrade the accuracy of the model[1]
- Integrity Protection solutions typically use Message Authentication Codes (**tags**) to verify parameters are tamper-free at run-time
- Apache TVM is a machine learning compiler framework that allows the user to define custom transformations on a model as it is being optimized for a specific hardware backend[2]

[1] Yes, One-Bit-Flip Matters! Universal DNN Model Inference Depletion with Runtime Code Fault Detection (https://www.usenix.org/conference/usenixsecurity24/presentation/li-shaofeng)

[2] Apache TVM (https://tvm.apache.org/)

MACMul

An Integrity Protection defense that uses Apache TVM & AES-CMAC to detect tampered weights in neural networks.

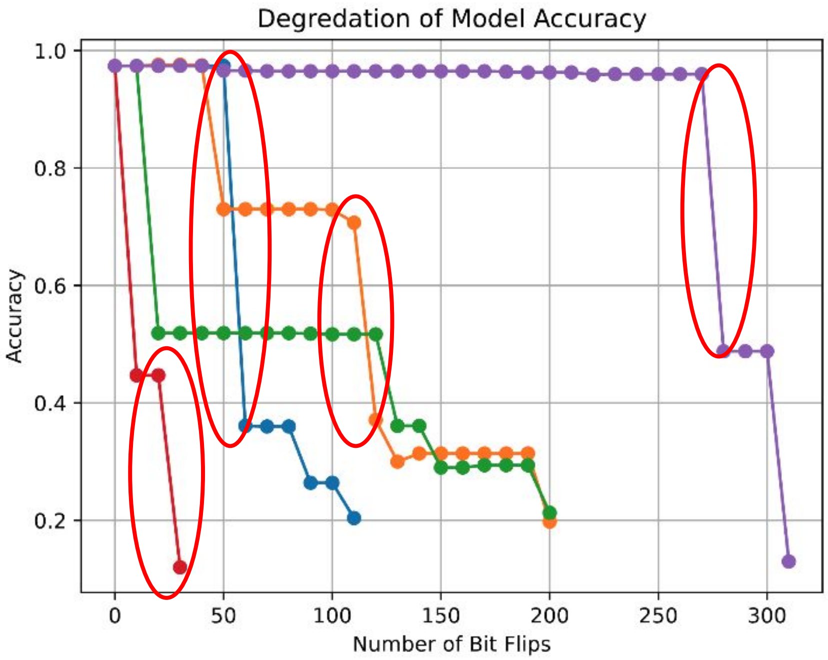
Methodology

- Rowhammer Primitive in Software
- Apache TVM: Custom Transformations, Parameter Injection, External Library Calls
- Probability & Hash Schedule

Rowhammer Primitive

- No need to implement a real-world Rowhammer attack
- Implement a random bit-flipping primitive in software

Model Degradation (MLPs on MNIST)



'Cliffs' indicate presence of critical bits

- Weights in a vulnerable layer
- Bits that change the sign of the parameter
- Bits that significantly impact magnitude of a parameter

Computing Tags

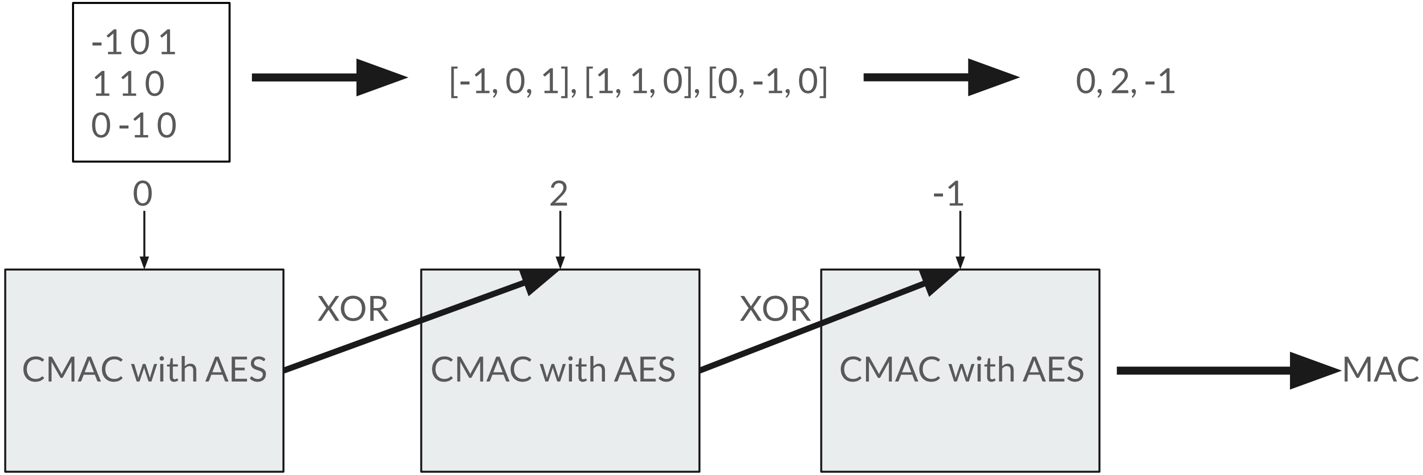
Obtain one tag per layer using the following algorithm:

Partition weight matrix into chunks $C_1, C_2, \ldots, C_N$.

Compute $S_1, S_2, \ldots S_N$, where S_i is the sum of the weights of C_i, round result to 4 decimal places.

Digest each S_i using a AES-CMAC block cipher algorithm.

Return output tag after all sums have been digested



Apache TVM - Parameter Injection

- Lower the neural network into Apache TVM's intermediate representation
- Update the signature of the neural network's main function to accept the tags as parameters

BEFORE: main(x, w^0, w^1, … , w^n)

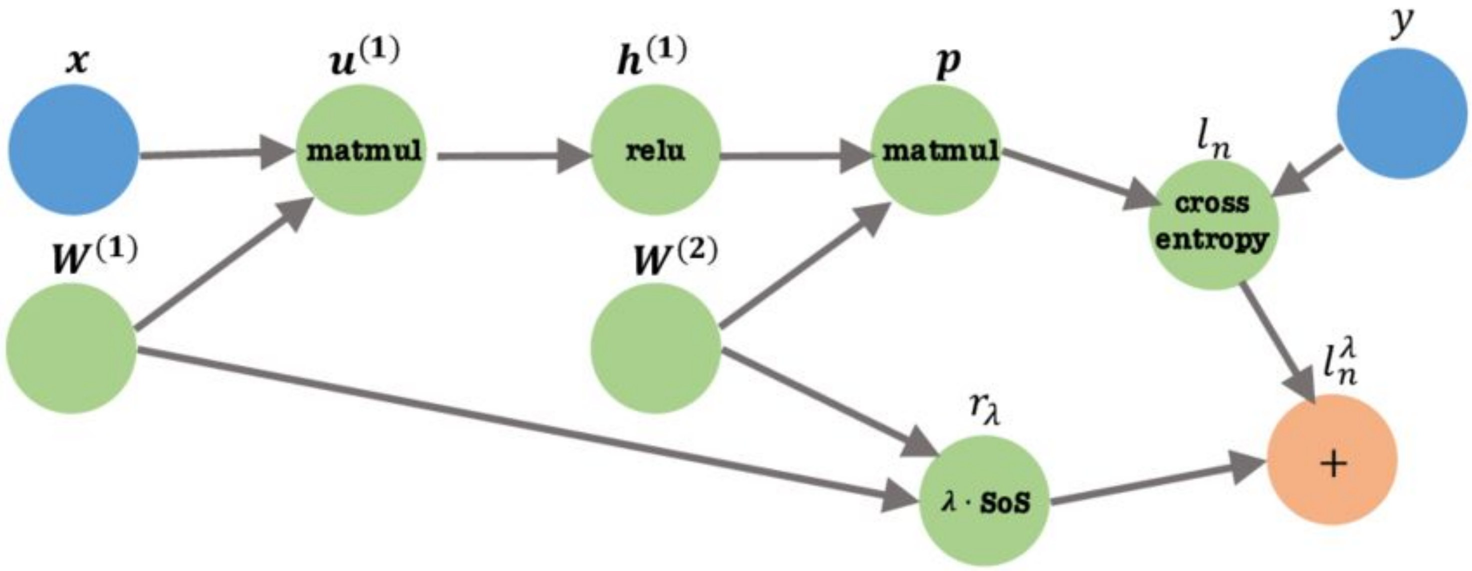
AFTER: main(x, w^0, w^1, … , w^n, h^0, h^1, … , h^n)

Apache TVM - Custom Transformations

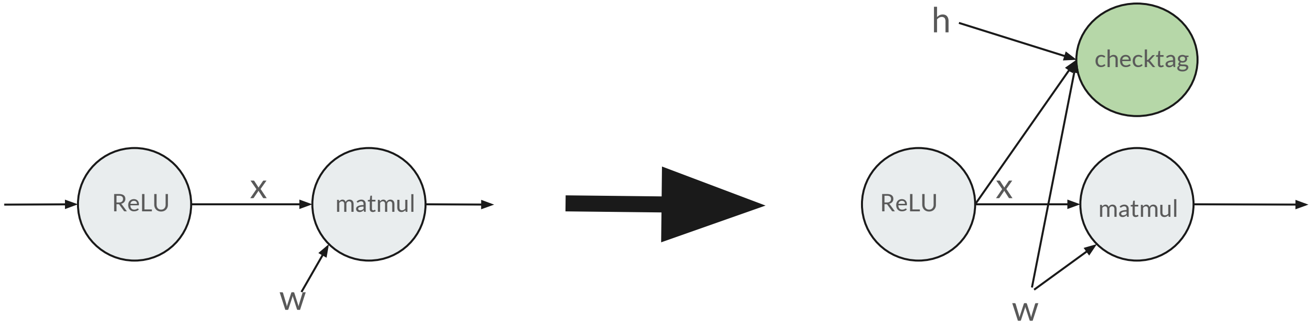
- Custom transformations traverse the computational graph of the neural network
- Identify each matrix multiplication operation acting on weights
- Insert an external function that verifies the run-time tag

```
check_tag(weight, ground_truth_tag):

        run_time_tag = get_tag(weight)

        assert run_time_tag == ground_truth_tag
```

Computational Graph View of a Neural Network

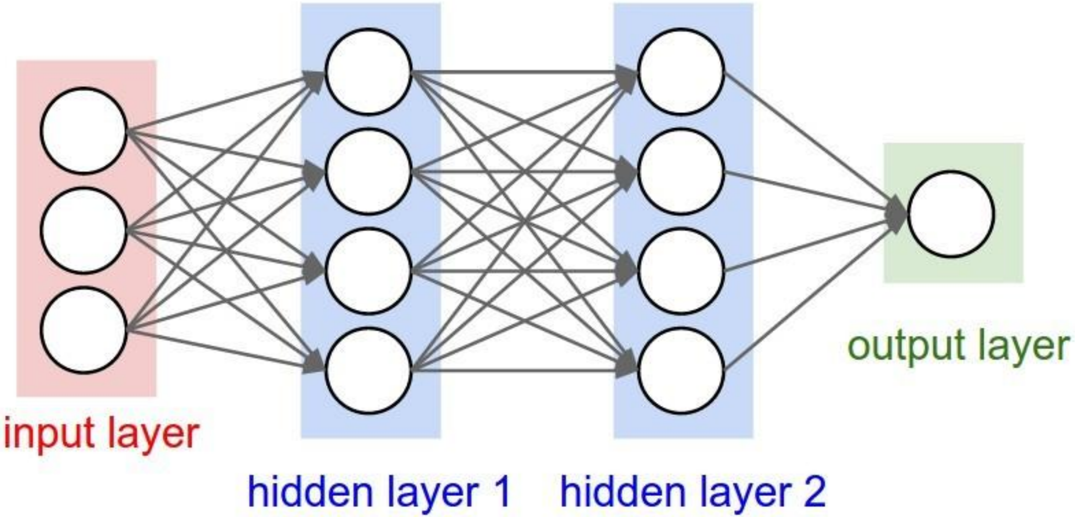


Zooming In...



Per-Layer Probability Schedule

- User can define a verification probability for each layer
- Assign higher probabilities to more vulnerable layers
- As long as all layers have a non-zero probability, bit flips will be detected eventually
- Provides a performance boost on average



P(check1) = 0.01

P(check2) = 0.95

Number of chunks

- User can configure the total number of chunks that will be summed and used in the CMAC Block Cipher.
- **More chunks per layer means better security against a powerful attacker, but this also incurs a larger performance overhead.**

Upper Bound on Inference Time

- Inference time linearly increases with the number of chunks.
- User specifies an upper bound on inference time that they deem acceptable.
- MACMul finds the number of chunks that respects this upper bound and divides them uniformly throughout the layers.

Summary

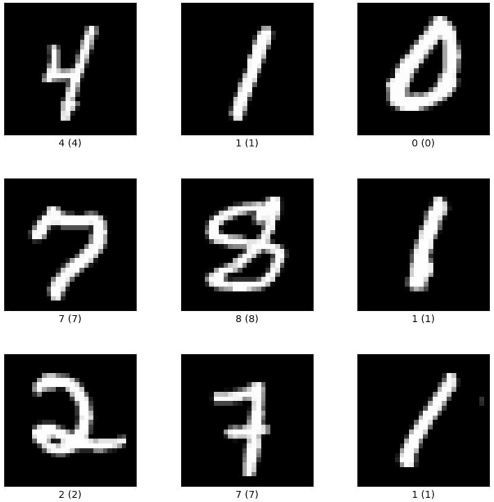
- Given a model, MACMul computes the tags, probability schedule, and number of chunks
- Inserts these as parameters to the intermediate representation of the model
- Runs a custom transformation to modify the computational graph and insert check_tag operators
- At inference time, check_tag will throw an AssertionError if an integrity violation is detected

Experiments

- Setup
- Experiments
- Analysis

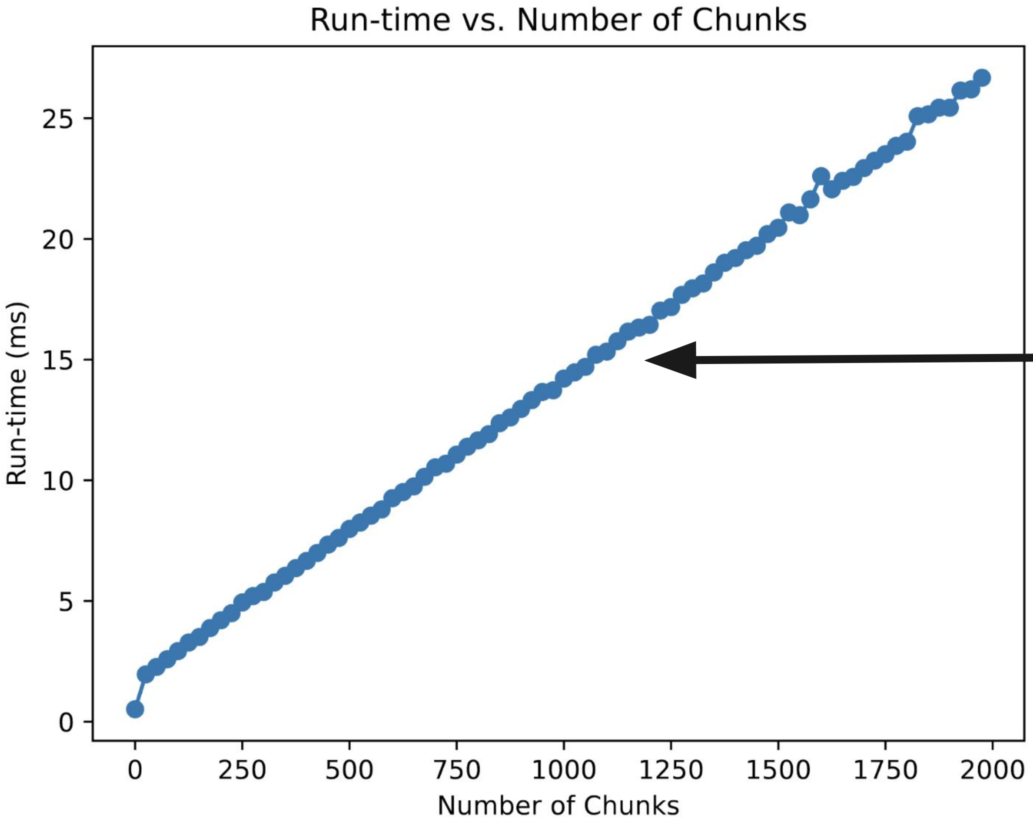
Setup

- MNIST Dataset
- MLPs (ReLU + Linear) with various parameter counts and depth

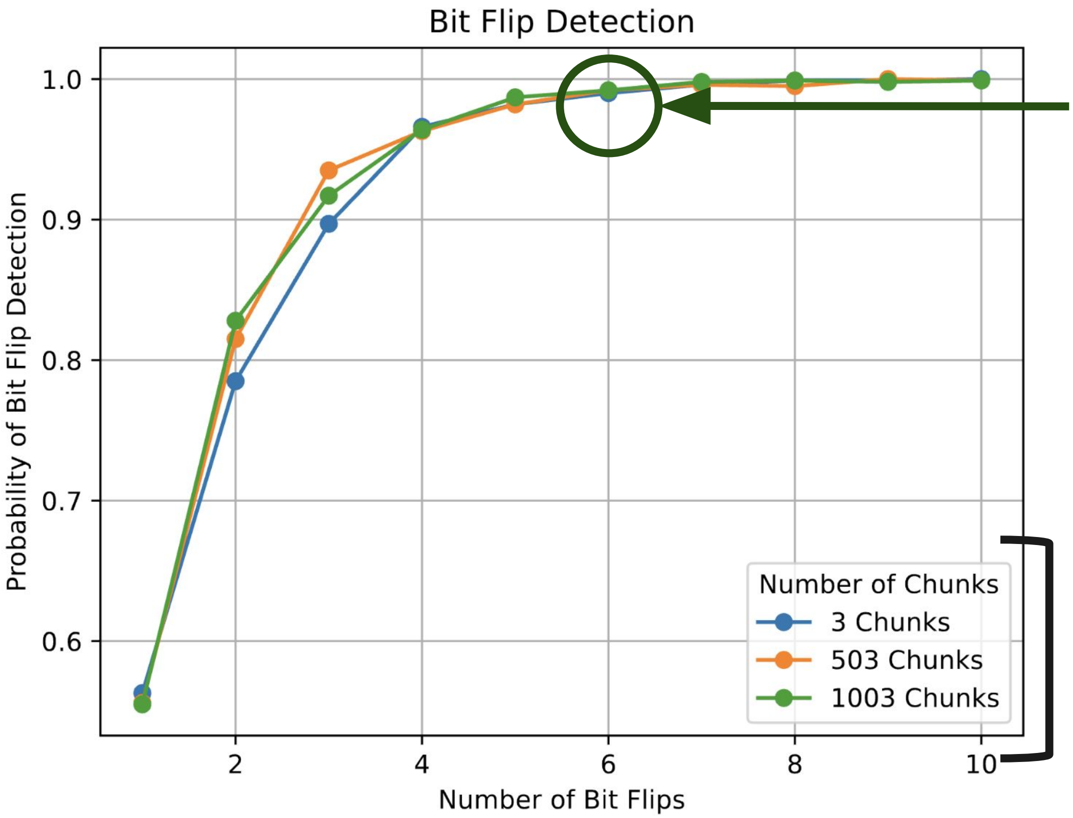


Experiments

- Probability of Detection after N bit flips
- Runtime vs. Number of Chunks



Runtime increases linearly with number of chunks



Nearly 100% detection after 6 random bit-flips

Adding more chunks does not provide greater security against random bit-flip attacker

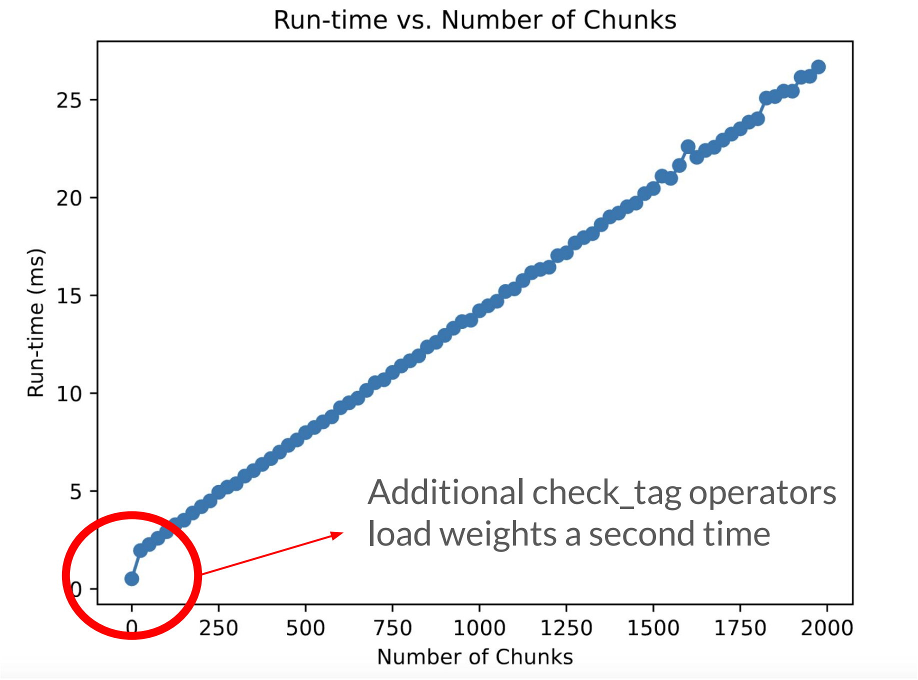
Analysis

- Assuming an adversary that flips bits randomly, use 1 chunk per layer
- Assuming an intelligent adversary, consider using many chunks per layer
- Assign non-vulnerable layers a lower probability of tag-verification for speedup on average

Limitations & Future Work

- Additional operator to computational graph
- Summing vulnerability

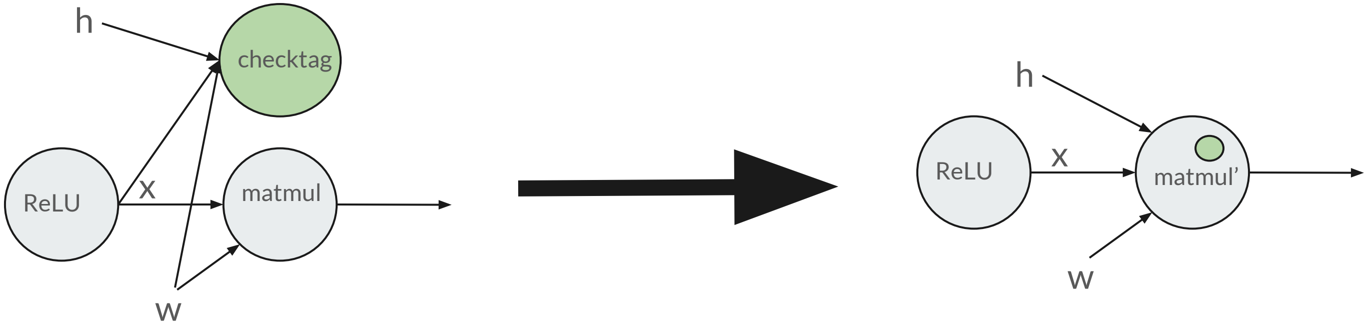
Additional Operator in Computational Graph



Additional check_tag operators load weights a second time

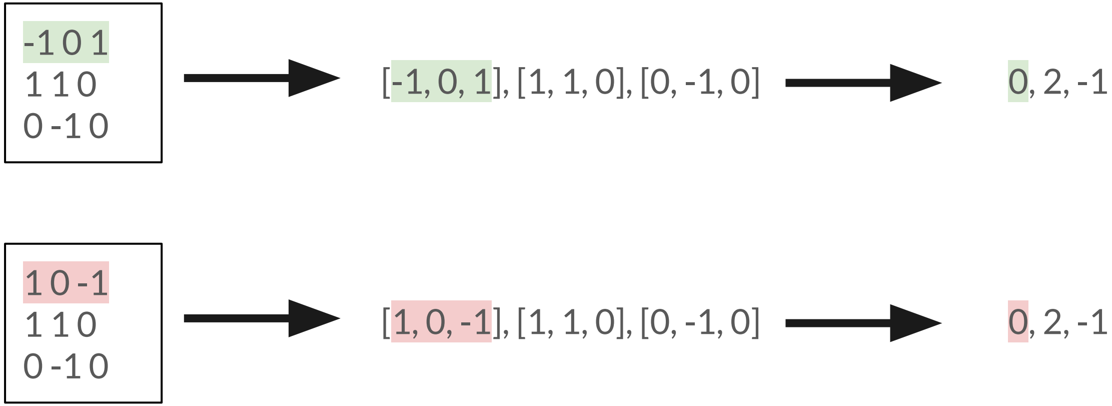
Additional Operator in Computational Graph

- Move towards solution that checks tags at the same time weights are accessed in matrix multiplication operation
- Luckily, MACMul provides a custom transformation that acts on the relevant operations, which is a strong foundation for future iterations



Summing Vulnerability

- Extremely powerful adversary can change weights in such a way that summed chunks produce the same output



-1 0 1
1 1 0
0 -1 0

→ [-1, 0, 1], [1, 1, 0], [0, -1, 0] → 0, 2, -1

1 0 -1
1 1 0
0 -1 0

→ [1, 0, -1], [1, 1, 0], [0, -1, 0] → 0, 2, -1

Summing Vulnerability

- Potentially unrealistic attack model (high precision required, complete knowledge of weights)
- Define random permutation of weights for chunking on a per-epoch basis
- Give the attacker less time to find two weights that exploit summing vulnerability

Epoch 1	Epoch 2	...
[A,B,C], [D,E,F], [G,H,I]	[D,H,F], [E,A,I], [C,B,G]	

Thank you! Questions?

MACMul: An Integrity Protection defense that uses Apache TVM & AES-CMAC to detect tampered weights in neural networks.

Repository Link: github.com/lbaierreinio/macmul