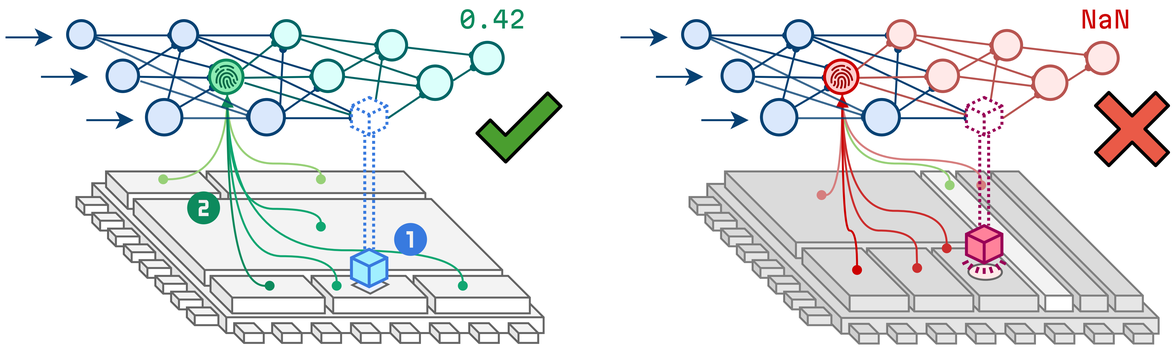
Locking Machine Learning Models into Hardware

Eleanor Clifford*[1], Adhithya Saravanan*[2], Harry Langford*[2], Cheng Zhang[1], Yiren Zhao[1], Robert Mullins[2], Ilia Shumailov[3], Jamie Hayes[3]

* Equal contribution

1 Imperial College London
2 University of Cambridge
3 Google DeepMind

In this paper, we investigate the feasibility of mechanisms that restrict machine learning (ML) models to only be usable on specific hardware. We demonstrate that against an attacker who has access to the model's parameters and architecture but no authorized hardware, such *ML Hardware Locking* mechanisms are feasible, including *soft locking* methods which make model performance or efficiency worse at different pruning or quantization levels, and *hard locking* methods which cryptographically tie the model's operation to unique properties of authorized hardware. We demonstrate that *ML Hardware Locking* comes with little overhead, while significantly restricting use of the model on unauthorized hardware.



Hard Locking

These mechanisms cryptographically link the model to properties of authorized hardware. They have two parts:

1. **Fingerprint:** A high-entropy key generated from unique properties of the target configuration.
2. **Parameter transform:** a transformation on the model parameters, that requires the fingeprint to invert.

Fingerprints

1. **Clock fingerprint** The number of clock cycles taken by a CUDA device for a series of operations. *Entropy:* < 20.
2. **Finite precision fingerprint** The exact numerical errors after a series of floating point operations.
   Entropy: requires large-scale experiment to estimate
3. **SRAM PUF fingerprint** The state of the SRAM on boot. Requires missing firmware support. *Entropy:* > 256.

Parameter transforms

There are three ideal properties of a parameter transform:

1. **Destruction:** The performance without access to the fingerprint should be equivalent to random guessing.
2. **Encryption:** No information about the original parameters should be obtainable without the fingerprint.
3. **Indistinguishability:** Correct and incorrect detransformation should be statistically indistinguishable.

Table 1: Comparison of our parameter transforms. Cracking cost is on FP16 ResNet18. b is the fingerprint entropy.

Method	Indistinguishablity	Encryption	Destruction	Cracking cost (s)
Plain AES	✗	✓	✓	0.3×2^b
Shuffling	✓	✗	✓	2.7×2^b
Pre-transformed AES	✓	✓	✓	1.3×2^b

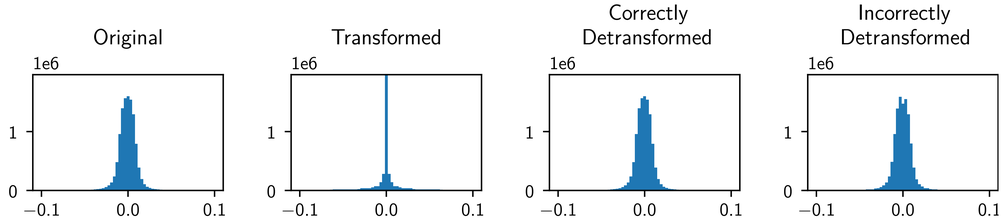


Figure 1: *Indistinguishability* of *Pre-transformed AES*

Soft Locking

These mechanisms do not fully restrict model use, but make it less performant and/or efficient on unauthorized hardware. We use the loss function:

$$\mathcal{L}_{lock} = \mathcal{L}(f_a(x), y) + \lambda(\epsilon - \mathcal{L}(f_u(x), y))^2$$

$\mathcal{L}(\cdot)$	e.g. cross-entropy loss	$f(\cdot)$	model
a	authorized configuration	u	unauthorized configuration
x	input	y	target output
ϵ	target unauthorized loss	λ	tuning parameter

Results

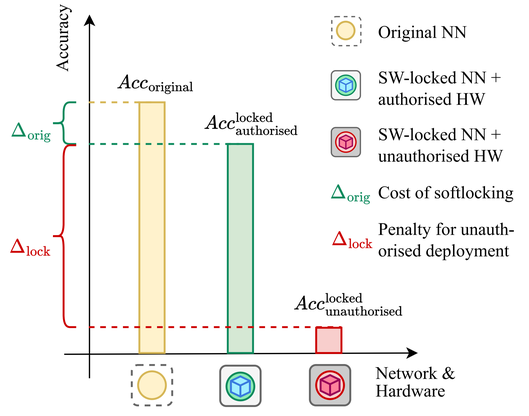


Figure 2: Soft locking evaluation metrics

Table 2: $Acc^{\text{locked}}_{\text{authorized}}(\Delta_{\text{orig}}, \Delta_{\text{lock}})$, sparsity-aware.

	Pruning Levels (ResNet50)		
Dataset	0.05	0.25	0.50
CIFAR10	0.92 (0.01, 0.81)	0.93 (-0.01, 0.83)	0.94 (-0.01, 0.84)
CIFAR100	0.69 (0.09, 0.63)	0.78 (0.00, 0.77)	0.78 (0.00, 0.77)
Flowers102	0.76 (0.10, 0.74)	0.86 (0.00, 0.84)	0.86 (0.00, 0.84)

Table 3: $Acc^{\text{locked}}_{\text{authorized}}(\Delta_{\text{orig}}, \Delta_{\text{lock}})$, quantization-aware.

Authorized → Unauthorized	Resnet18	Resnet50
FP32 → 8-bit MiniFloat	0.90 (-0.02, 0.65)	0.92 (-0.04, 0.67)
Int8 → 8-bit MiniFloat	0.47 (+0.41, 0.06)	0.50 (+0.39, 0.07)
FP16 → Int8	0.90 (-0.02, 0.62)	0.91 (-0.04, 0.61)
16-bit MiniFloat[??] → Int8	0.90 (-0.01, 0.72)	0.91 (-0.03, 0.81)

Emulation attack

We show that emulating the authorized configuration to escape soft locking results significantly reduces efficiency.

Table 4: Emulation costs for soft locking

		Sparsity-aware		Quantisation-aware	
Workload	Metric	Real	Emulated	Real	Emulated
Single Matmul	Throughput (TOPS)	49.97	18.85	79.22	22.72
	Latency (ms)	0.43	1.14	0.27	0.95
OPT inference	Throughput (TPS)	4692.20	2468.31	3505.22	1865.41
	Latency (ms)	436.47	829.72	584.27	1097.88

Retraining attack

We show that re-training a locked model takes similar compute to initial training, and may not recover performance.

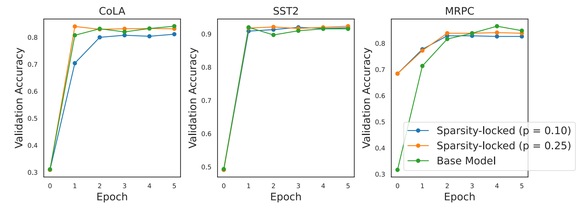


Figure 3: Re-training sparsity-locked BERT/GLUE

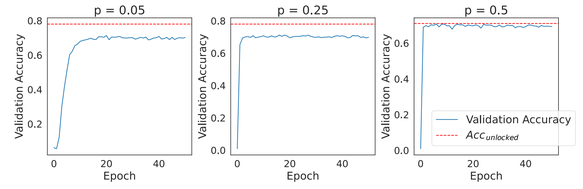


Figure 4: Re-training sparsity-locked ResNet50/CIFAR100

Noise attack

We show that adding noise to destabilise the lock may be partially effective, depending on hyperparameters.

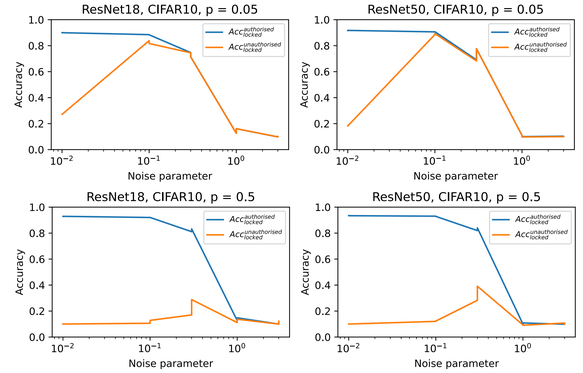


Figure 5: Attacking soft locking by adding noise

Scalability

Soft locking adds no compute cost to using the model regardless of model size. The compute cost of creating a soft-locked model scales with the cost of fine-tuning the model.

The compute cost of creating or loading a hard locked model is of order 1s for our small test models, and scales as $O(n)$. This cost is incurred only when the model is first loaded.

However, the compute cost of brute-forcing a hard-locked model is $\gg O(n)$, because *indistinguishability* forces the attacker to test the model's performance on each attempt, rather than use a much cheaper statistical test.

2025-04-09