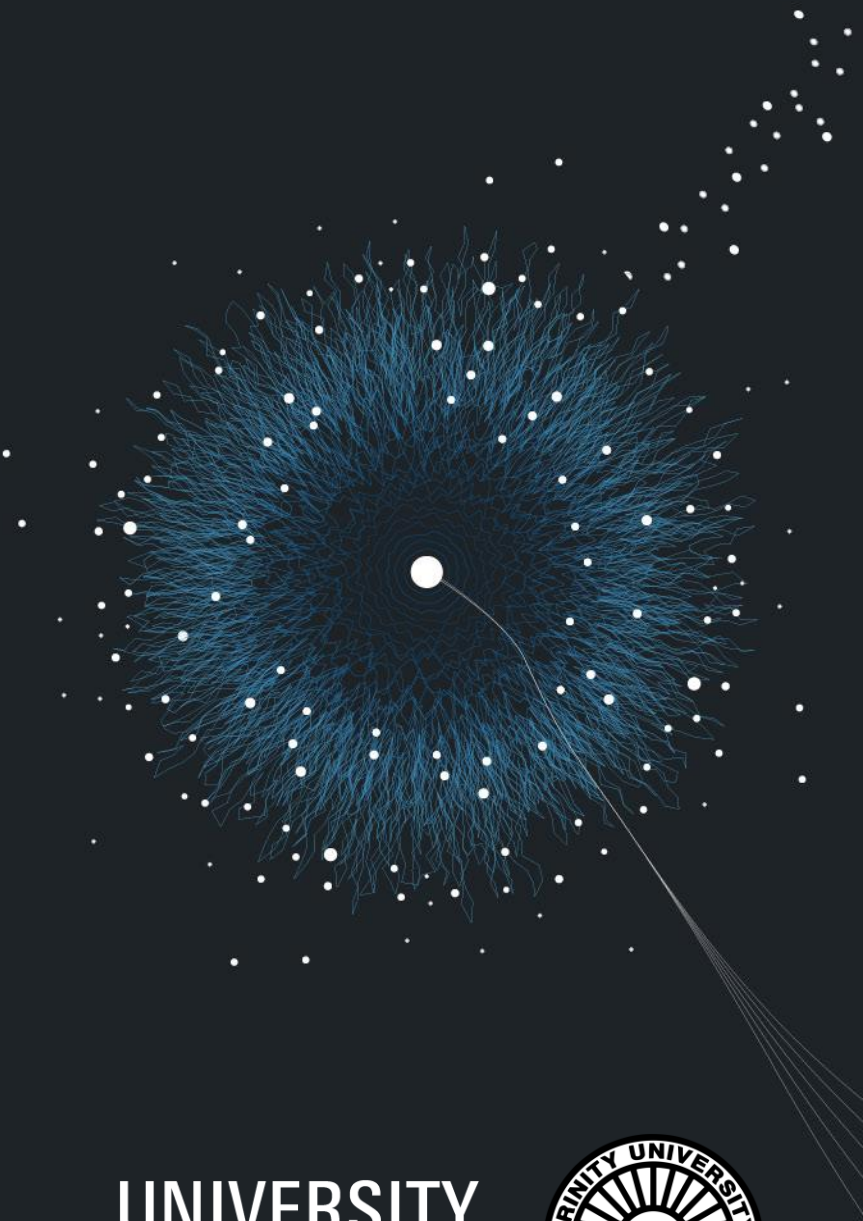
EEMCS - SCS

SELF-REPAIRING NEURAL NETWORKS BASED ON NEURAL CA

TOMMASO PALADINI, EVA TUBA, LUCA MARIOT

ACANCOS WORKSHOP@UCNC 2026

TRIESTE, JUNE 22, 2026



UNIVERSITY OF TWENTE.

TRINITY UNIVERSITY
E TRIBUS UNUM
1869
SAN ANTONIO, TEXAS

ML models in critical systems — what can go wrong?

Weight Tampering

Attacker corrupts stored or transmitted weights, degrading accuracy in a targeted or untargeted way.

[Liu et al. 2018, Biggio et al. 2012]

Model Extraction

Partial knowledge of a model's weights may allow an adversary to reconstruct confidential IP.

IP protection scenario

Hardware Faults

Stuck-at faults in RRAM/neuromorphic accelerators produce bit-flip errors in weight storage at runtime.

Fault-tolerant computing

Research question: can a Neural CA, trained on a weight matrix, detect and repair local corruption, without any global copy of the parameters?

How Growing Neural Cellular Automata work

1. **Each cell has a state vector:** Visible channels (RGBA) + hidden channels for internal "chemistry".

2. **Perceive neighbours:** Fixed Sobel-x, Sobel-y and identity filters over the 3×3 neighbourhood → perception vector.

3. **Apply learned rule:** shared 2-layer MLP maps the perception vector to a state delta Δs (zero-initialised → starts as no-op).

4. **Stochastic update:** Each cell updates with p ≈ 0.5, independently (avoid oscillations)

Morvintsev et al.: Growing Neural Cellular Automata – Differentiable Models of Morphogenesis (2020)

Self-Repair Methodology: NCA repairs the weight matrix

1

Train base NN

e.g. MNIST classifier (TinyMLP)

→

2

Embed weights

Weight matrix W becomes 2D CA grid

→

3

Apply damage

Adversarial corruption, masking, or bit-flip

→

4

NCA repairs

Local rule iterates until grid converges

Assumption: No centralised copy of the full weight matrix is needed at repair time. Recovery relies exclusively on local cell interactions — the CA rule is a distributed snapshot of the target weight matrix.

Representing a weight matrix as a 2D grid

Example: 4 inputs → 3 outputs

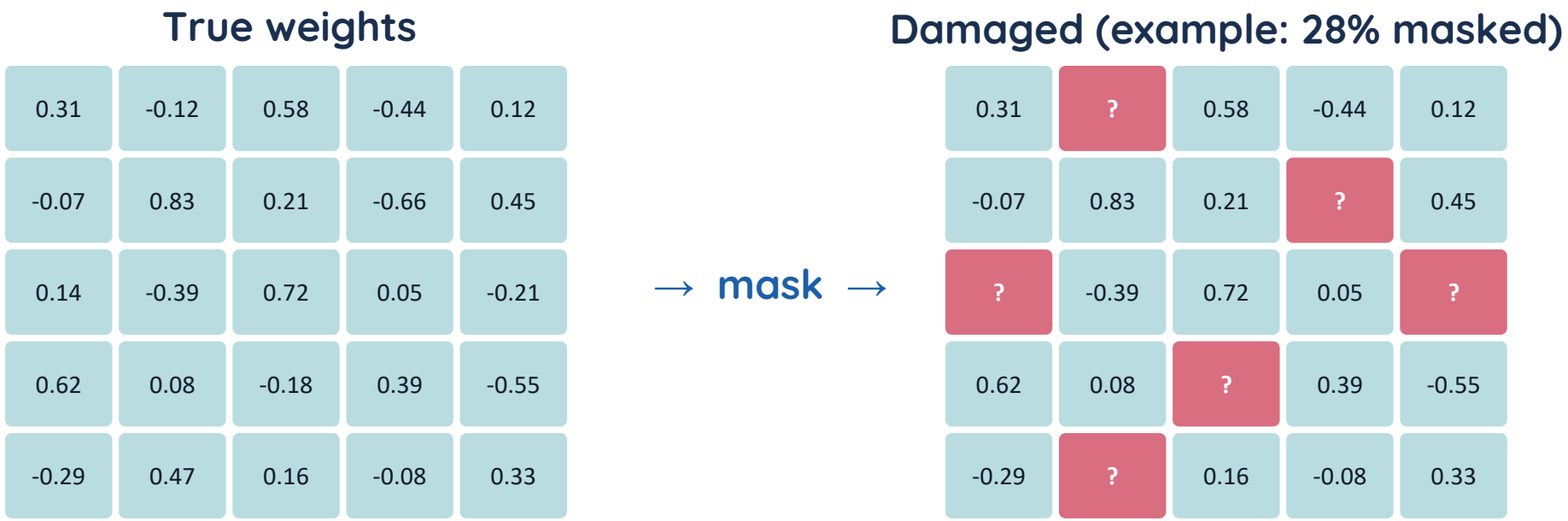
0.31	-0.12	0.58	-0.44
-0.07	0.83	0.21	-0.66
0.14	-0.39	0.72	0.05

Row = output neuron
Column = input neuron

$w_{22} = 0.83$ ← one cell

- **Same structure as an image:**
  Each weight = one pixel = one CA cell
- **Local statistical structure:**
  Nearby weights share co-variance the NCA can exploit
- **Natural 2D topology:**
  Row = output neuron, column = input neuron — no reshaping needed
- **Scales to multi-layer nets:**
  Run a separate CA per layer, or a 3D CA across layers

Damage model: random masking with a binary mask



NCA state tensor (per cell)

ch 0	ch 1	ch 2…C
weight value	mask (fixed — 1=known, 0=missing)	hidden scratch space for the NCA

NOTE: *Known cells (ch 1 = 1) are clamped every step — the NCA can never overwrite a surviving weight.*

One local rule, applied to every cell at every step

1

Perceive

Apply identity + Sobel-x + Sobel-y filters over the 3×3 neighbourhood → perception vector summarising local values and gradients.

2

Update rule

Feed the perception vector into a tiny shared MLP Output: a small delta Δ. Last layer zero-initialised — starts as a no-op.

3

Clamp

Known cells: discard the delta, restore original value. Missing cells: apply the delta. Repeat for N steps → information propagates inward from known neighbours.

NOTE: The same ~1,000-parameter MLP runs at every cell simultaneously — a shared rule with no global view.

Training: L2 loss on missing cells, BPTT through N steps

1

Damage the true weight matrix (random mask, random rate each epoch)

2

Run NCA for N steps (perceive → update → clamp, repeated)

3

Read recovered weights (channel 0 of the final state)

4

Compute L2 loss (missing cells only — mean squared error)

5

Backprop + Adam step (gradients flow through all N unrolled steps)

↺ next epoch

Why loss on missing cells only?

Known cells are already frozen, penalising them would be redundant and destabilise training.

Results

NCA vs. baselines — single layer (FC1), TinyMLP on MNIST

Damage	Zero-fill	Mean-fill	Neighbour	NCA ★
5%	96.83%	96.86%	96.82%	**96.96%**
10%	96.68%	96.66%	96.62%	**96.84%**
20%	96.49%	96.44%	96.05%	**96.76%**
30%	96.01%	95.91%	95.27%	**96.57%**
50%	94.90%	94.56%	89.34%	**96.25%**
70%	89.54%	87.18%	58.11%	**94.45%**
90%	65.86%	50.68%	34.16%	**88.77%**

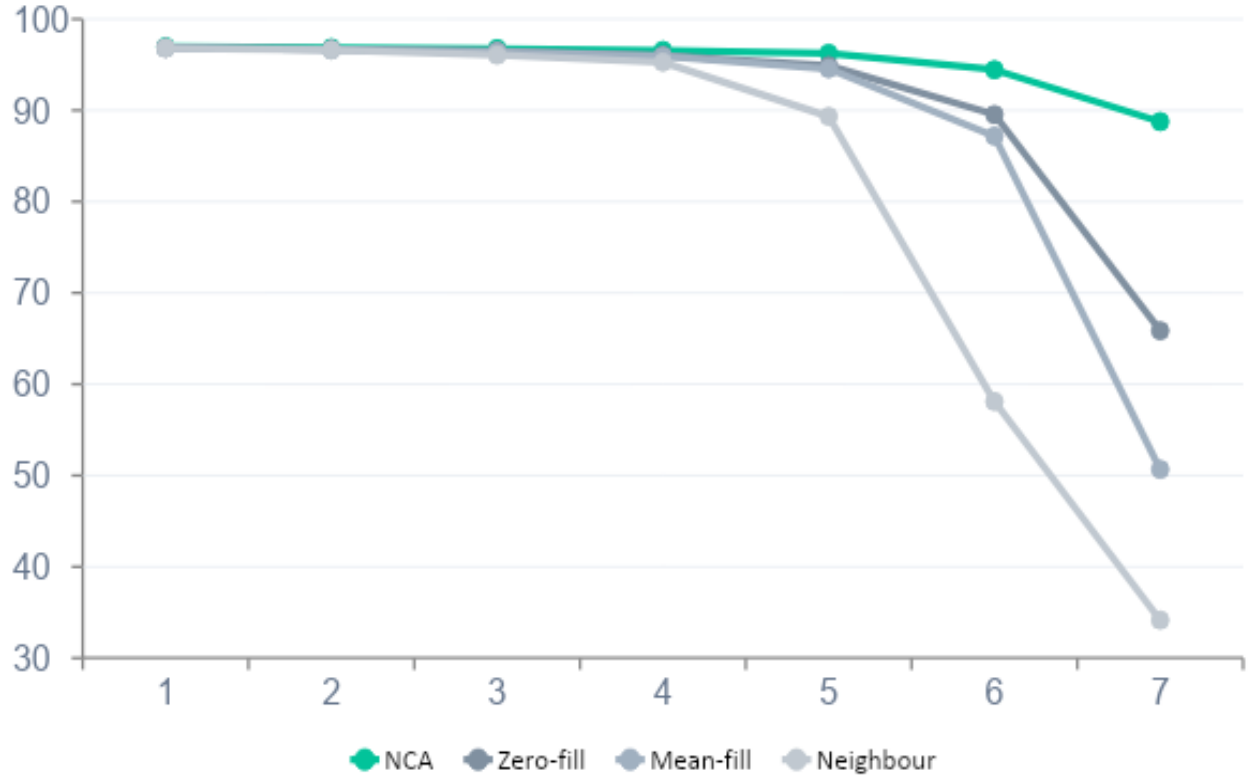
88.8%

accuracy at
90% damage

Baselines collapse toward chance. NCA holds.

Gap widens with damage severity → learned structure, not smoothing.

Recovery curve: accuracy vs. damage rate



Main Findings:

1. NCA degrades much more slowly than all baselines
2. Gap is largest at heavy damage (70–90%) — exactly where it matters
3. Simple baselines collapse; neighbour-fill hits ~34% at 90% damage
4. A single training regime generalises across all damage rates

Future Work

Open research directions

1 Adversarial damage

Move from random masking to PGD/FGSM perturbations. Formalise what perturbation models the NCA can/cannot handle.

2 Multi-layer & larger nets

Layer-conditioned NCA, or a 3D NCA across layers. Test on Transformers and convolutional nets beyond TinyMLP/MNIST.

3 Functional loss

Replace L2 on weights with KL between model outputs — optimise for functional equivalence, not exact weight recovery.

4 Population training

Train on a zoo of models (Schürholt et al. 2021 datasets). Learn the distribution over weight space, not a single target.

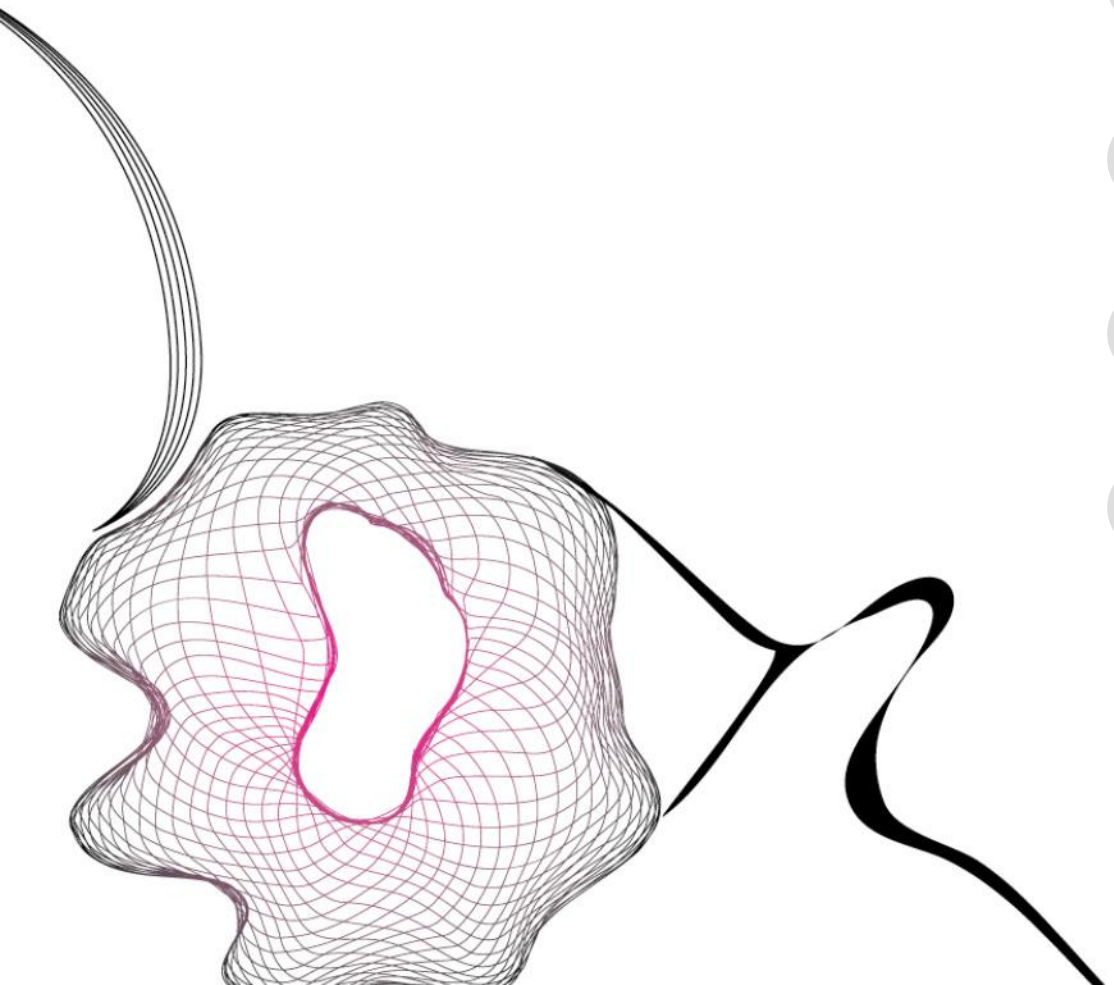
5 Information-theoretic bounds

Formalise recovery capacity in terms of error-correcting code bounds. What fraction of corruption is provably recoverable?

6 Model extraction defence

Explore the dual application: can the NCA also detect when only a fraction of weights are known, limiting adversarial reconstruction?

SUMMARY



- ✓ Weight matrices as CA grids: one weight, one cell
- ✓ A tiny shared MLP (~1,000 params) learns to repair damaged weights from local context alone
- ✓ NCA recovery outperforms all simple baselines, with the largest gap at high damage (70–90%)
- ✓ 88.8% accuracy retained at 90% damage, where baselines approach chance

THANK YOU FOR YOUR ATTENTION!

UNIVERSITY OF TWENTE.

References

[1] Mordvintsev, A., Randazzo, E., Niklasson, E., Levin, M. Growing Neural Cellular Automata. Distill, 2020.

[2] Najarro, E. et al. HyperNCA: Growing Developmental Networks with Neural Cellular Automata. arXiv:2204.11674, 2022.

[3] Schürholt, K. et al. Self-Supervised Representation Learning on Neural Network Weights. NeurIPS 2021.

[4] Liu, Y. et al. Trojaning Attack on Neural Networks. NDSS 2018.

[5] Biggio, B., Nelson, B., Laskov, P. Poisoning Attacks Against Support Vector Machines. ICML 2012.

[6] von Neumann, J. Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components. Princeton UP, 1956.

[7] Zhou, A. et al. Neural Functional Transformers. NeurIPS 2023.