

Structure-Aware Software Resilience for Number Theoretic Transforms

Brittany Liu¹ · Caroline Wang¹ · Jayanta Chowdhury^{2,3} · Igor Nunes¹ · Nahid Farady⁴ · Elif Bilge Kavun^{2,3} · and Ro Cammarota¹

FDTC 2026 | ¹UC Irvine · ²Barkhausen Institut · ³TU Dresden · ⁴Microsoft

UC Irvine



[*] All product names, logos, brands, trademarks, and registered trademarks are property of their respective owners.

Why protect the NTT itself?

Prior-art gap

- FFT/DFT resilience has a long ABFT/checksum lineage
- Encrypted computation changes the stakes: NTTs are repeated kernels in FHE and lattice-based PQC
- NTT is an arithmetic diffusion engine
- Failure of classic protection ECC/CRC against compute/transient faults

Generic redundancy asks:

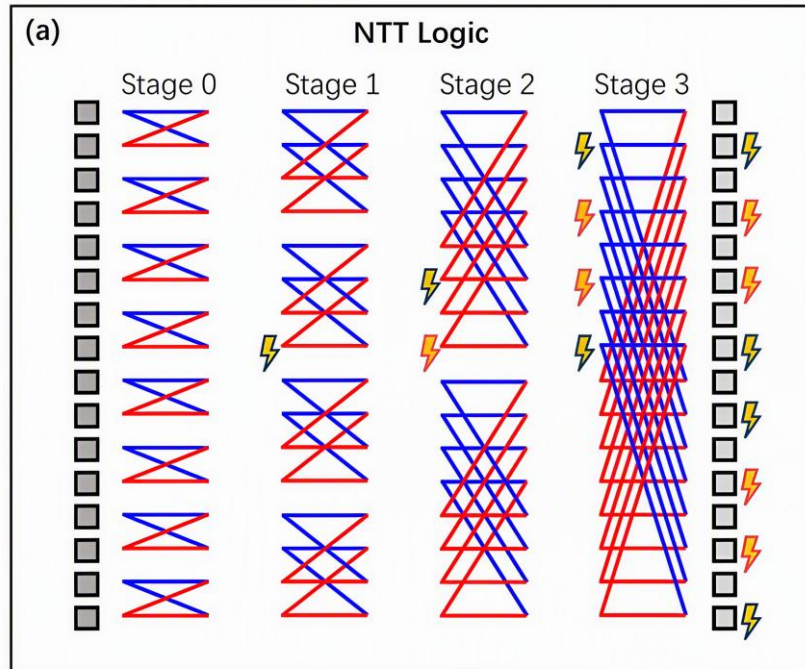
“Did ‘n’ executions agree?”

Structure-aware resilience asks:

“Does this execution satisfy the algebra it must obey?”

Can NTT structure now become the resilience mechanism?

One butterfly gives us local structure



(source) Figure 7a. ReliaFHE in Li et al.
(ASPLOS'2026)

- A complete NTT is a hierarchy of these butterflies across $\log_2 N$ stages

- E.g., 1 Radix-2 Cooley–Tukey butterfly

$$\begin{array}{l} \text{Blue line} \\ \text{Red line} \end{array} \begin{array}{l} y_0 = u + \omega v \pmod{q} \\ y_1 = u - \omega v \pmod{q} \end{array}$$

- Faults introduced early have more opportunities to diffuse
- The same hierarchy exposes multiple verification boundaries

Two complementary verification granularities

Butterfly invariants

$$y_0 + y_1 \equiv 2u \pmod{q}$$

$$y_0 - y_1 \equiv 2\omega v \pmod{q}$$

$y_0, y_1 = \text{butterfly output}$

Fine-grained detection → localize one butterfly → recompute one butterfly

Stage-level projections

$$S_{Sum}^{(k)} = \sum_i x_i^{(k)} \pmod{q}$$

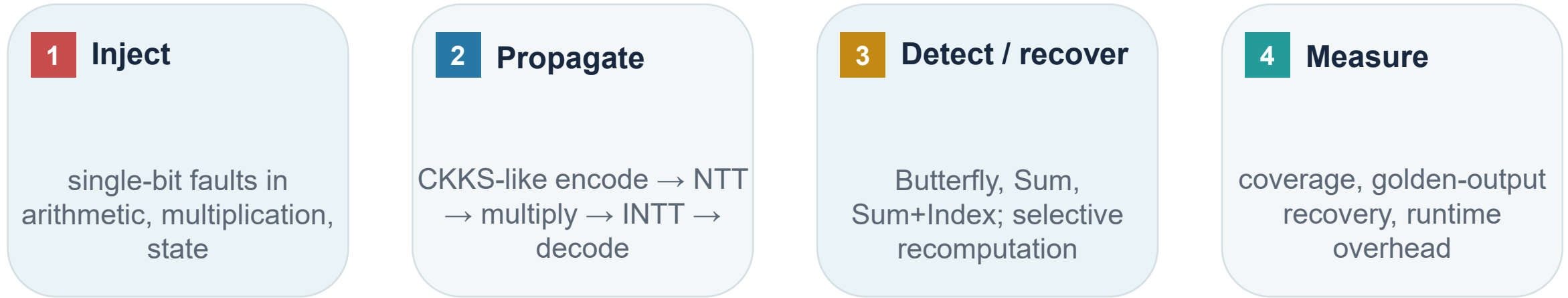
$$S_{Index}^{(k)} = \sum_i i x_i^{(k)} \pmod{q}$$

$x_i, i = \text{butterfly input, index}$

Coarser detection → localize one stage → rollback and replay one stage

The hierarchy that organizes NTT computation tells us where consistency was lost — and therefore how much to recompute

Experimental methodology



100%

Butterfly

75%

Sum only

1,184 multiplication-path faults

100%

Sum + Index

Detection: 4,608 confirmed injections ; Runtime: $N(\text{polynomial degree}) = 2,048 \dots 32,768$

The simple checksum has a structural blind spot

Multiplication-path perturbation

$$y'_0 = y_0 + \delta, \quad y'_1 = y_1 - \delta$$



Sum is multiplication-blind

$$y'_0 + y'_1 = y_0 + y_1$$

The $+\delta / -\delta$ error lies in the nullspace of the Sum projection

The experiment reproduces the predicted structural blind spot

Index weighting breaks the symmetry

$$(i_{lo} - i_{hi})\delta \neq 0 \text{ in general}$$

Design principle: aggregate checks must preserve the structure of the errors they are meant to observe

Detection becomes useful when it controls recovery

Butterfly checker

100 / 100

recompute one butterfly

Sum only

75 / 100

multiplication faults escape
detection

Sum + Index

100 / 100

rollback and replay one stage

Localization dictates recovery cost

Protection cost is proportional — not duplication

12.8–14.5%

Butterfly checking overhead

18.6–22.2%

Stage-level checking overhead

Across $N = 2,048 \dots 32,768$, relative overhead remains nearly constant for a fixed modulus width

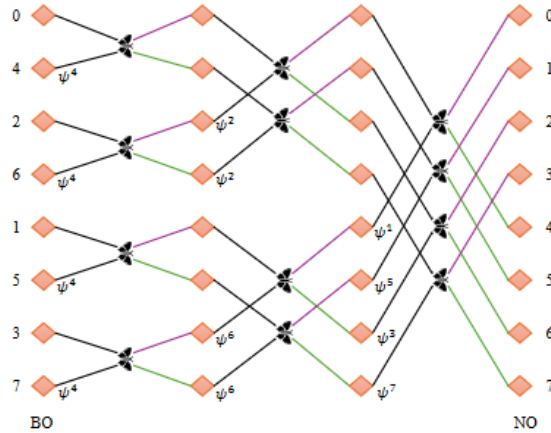
Neither mechanism duplicates a complete NTT

Heterogeneous protection — not one mechanism everywhere

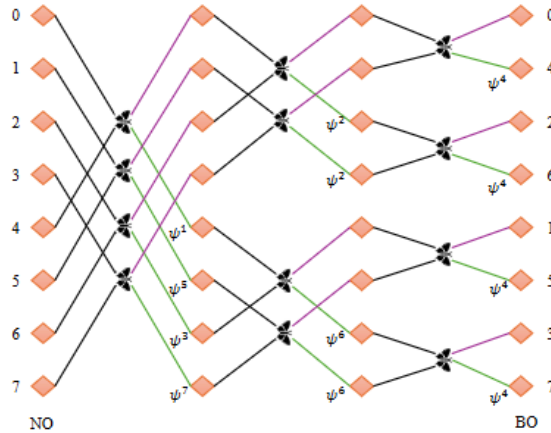


The extension is heterogeneous because the error mechanisms and cost structures are heterogeneous

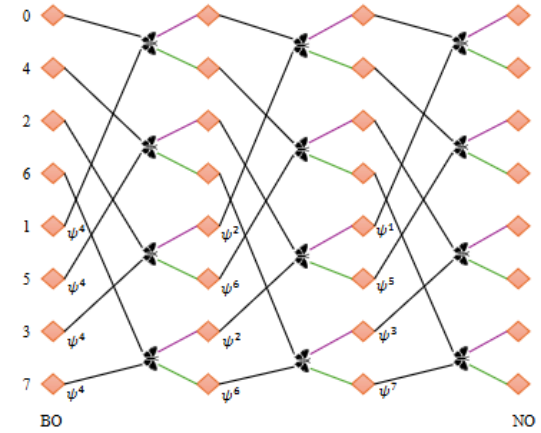
Topology determines the fault “blast radius”



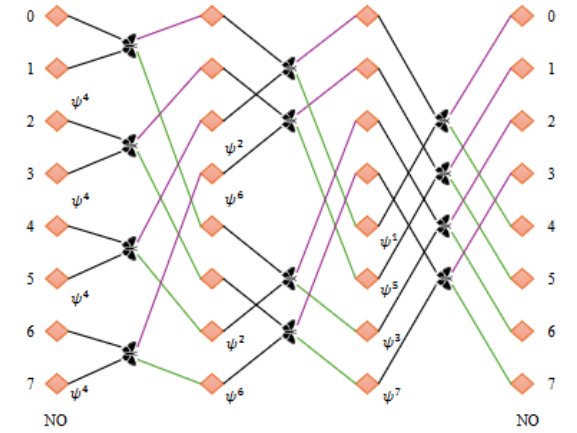
Cooley Tukey (so far)



Gentleman Sande



Pease



Stockham

(source) *Fault Diffusion Analysis and Heterogeneous Fault-Tolerance Architectures for FPGA-based NTTs*, FPT'26 Special Session (our upcoming work)

Fault location + transform scheduling determine how corruption diffuses — information we can use to place checks and recovery boundaries

A hierarchy of structure-aware resilience

Butterfly / PE

local invariant / local replay

Vector / block

lane-group checks / block replay

Stage / kernel

projection checks / checkpoint rollback

**Application /
service**

semantic validation / service retry

Next

- Vectorized/GPU NTTs
- Higher radix
- FPGA/ASIC
- Cross-layer containment
- Diversified checking

Verification granularity becomes an architectural parameter

Takeaways

- 1 **Structure is redundancy we already paid for**
- 2 **Preserve enough structure to observe the faults you care about**
- 3 **Localization should determine recovery granularity**
- 4 **Compose portable software invariants with topology-aware hardware protection**

Thank you

Any questions?

Structure-Aware Software Resilience for NTTs

github.com/Confidential-Intelligence-Lab/ntt-resilience