

NeuroGraphHeal: A Simulation Framework for Evaluating Neuromorphic Detection and Graph Learning in Underwater IoT Fault Recovery

Madhura MD 1

1Department of Data Engineering,
GM University,
Davangere, Karnataka, India.

B Yashwanth 2

2Department of Information Science
GM University,
Davangere, Karnataka, India

Abstract - Spiking neural networks (SNNs) and graph neural networks (GNNs) are candidate components for fault detection and recovery in underwater IoT networks. This paper builds a simulated recovery environment with a pre-specified validity gate and tests, in it, whether an SNN detector or GNN-based criticality improves a greedy recovery policy over classical alternative. A synthetic 30-node network with hidden node degradation, decoy disturbances, link faults and a budget of three backups was checked against ten pre-specified conditions on 100 validation episodes. Detectors (threshold, logistic regression, SNN, size-matched dense network, GRU) and a two-layer GNN were then compared on 300 test episodes with paired bootstrap intervals. The environment failed its gate in both runs: the best classical greedy captured 82.9 percent of oracle headroom in Run 1 (limit 80 percent), and a permuted-window negative control captured 13.6 percent in Run 2 (limit 10 percent). In the exploratory tests, greedy control with the SNN detector earned 11.78 less reward per episode than with the logistic detector (95 percent confidence interval 6.33 to 17.23 lower), and a size-matched dense network and a GRU were worse by similar margins. GNN criticality earned 8.84 less than live static criticality (interval 4.65 to 13.79 lower), although the GNN predicted isolation with an AUROC of 0.946. No improvement from the SNN or GNN was found, and the gate failure limits what these comparisons show. The contribution is the documented framework and an account of why its validity gate failed.

Keywords - underwater IoT; fault recovery; spiking neural network; graph neural network; simulation; validity gate; negative result

I. INTRODUCTION

Underwater sensor networks help in ocean monitoring, surveillance and environmental sensing, but they operate under conditions that make node and link failures likely and repair expensive: acoustic links with limited bandwidth and long, variable propagation delay, tight energy budgets, and nodes exposed to fouling and corrosion [1], [2]. Fault management in sensor networks is a long-studied problem, and surveys catalog detection techniques for the faults, communication failures and topology changes that harsh, unattended deployments produce [3]. This work concerns an underwater sensor network that must keep reporting to a surface sink while nodes degrade and links fail, with a small number of backup deployments available for recovery.

The anchor work for this project is Sharshika, Chaba and Khambra [4], who present an adaptive fault detection and

tolerance mechanism for underwater wireless sensor networks. Their abstract explains a hybrid detection approach that combines explicit and implicit methods, a dynamic thresholding mechanism, and a light-weight tolerance strategy based on multi-path rerouting and load-aware decision-making, evaluated in MATLAB simulations across fault densities and node behaviors; it reports gains in detection accuracy, false alarms, latency and network lifetime over traditional fault-management schemes [4]. The abstract does not describe a learned detector. That leaves open whether learned components add value on top of threshold-style detection once recovery resources are less.

This project asked whether two learned components could improve recovery in such a network: a spiking neural network (SNN) as an edge anomaly detector, and a graph neural network (GNN) that estimates which nodes are at risk of losing connection to the sink. SNNs communicate through discrete spikes [5] and are frequently proposed for low-power edge sensing [6], [7], while GNNs propagate information along network edges and suit questions about connectivity [8], [9]. An initial simulator and reinforcement-learning controller were built first. They were found to be invalid because one action dominated the reward and most apparent isolation came from the sink failing, a pattern that resembles what the AI-safety literature calls reward hacking [10]. They were replaced by a new environment with a pre-specified validity gate (Section VIII).

The aim of this paper is to describe that environment, report whether it passed its gate, and test the two components in it as exploratory comparisons against classical alternatives. The objectives are:

- to specify the simulated network, fault model, observation, actions and reward;
- to check the environment against ten pre-specified validity conditions;
- to compare an SNN detector with a threshold rule, a logistic regression, a size-matched dense network and a recurrent network;
- to test whether GNN-based criticality improves a greedy recovery policy.

The pre-specified expectations were that the SNN would be compared with the logistic detector as the primary test and that the GNN test would tie. No hypothesis that a component would improve recovery was stated as a pass condition.

II. RELATED WORK

A. Fault management in underwater and wireless sensor networks

The underwater acoustic channel shapes every design decision in these networks. Akyildiz et al. identify limited bandwidth, long and variable propagation delay, high error rates, energy constraints and failure-prone nodes as the central challenges [1], and Heidemann et al. review how applications and technology have since advanced against the same constraints [2]. Muhammed and Shaikh survey fault detection strategies for terrestrial wireless sensor networks and propose a taxonomy for classifying the detection techniques [3]. Sharshika et al. bring this line of work to underwater networks with a hybrid explicit and implicit detection scheme and dynamic thresholds, coupled with multi-path rerouting [4]. In [4], detection is threshold-based and recovery is handled by rerouting. Recovery that spends a limited stock of physical backups, which is the setting here, is not part of the scheme in [4], and neither is a learned detector.

B. Learned and neuromorphic detection

Anomaly detection is a broad field, surveyed by Chandola et al. [11]. Spiking neural networks, described by Maass as a third generation of neural network models [5], encode information in discrete events over time. Roy et al. review the prospects of spike-based machine intelligence, in which sparse event-driven computation on neuromorphic hardware is the route to efficiency [6]. Because spikes are not differentiable, SNNs are usually trained with surrogate gradients [12], [13], and the snnTorch tutorial of Eshraghian et al. documents this practice and the interplay between data encoding and learning [7]. Pfeiffer and Pfeil discuss the opportunities and open challenges of deep learning with spiking neurons [14], and neuromorphic processors such as Loihi provide dedicated hardware [15].

Energy advantages are conditional rather than automatic. Dampfhöffer et al. show that an SNN can compete with an efficient ANN implementation only at very high spike sparsity once memory access is counted [16], and Davidson and Furber compare ANN and SNN implementations on digital hardware and find that the outcome depends on implementation details [17]. For this reason, the present study treats spike sparsity only as a proxy and makes no energy claim.

C. Graph learning for network state

Graph convolutional networks aggregate features from a node's neighborhood [8], and surveys cover the broader family of graph neural networks [9]. In networking, RouteNet applies message-passing to predict path-level performance such as delay in software-defined networks [18]. Predicting which nodes will lose connection to a sink is a node-level task on a graph and fits the same toolset, which motivates the GNN used here.

D. Validity and reporting in simulation-based learning studies

Simulated environments can reward behavior that the designer did not intend, a failure mode discussed as reward hacking [10]. Henderson et al. showed that reported improvements in deep reinforcement learning are sensitive to seeds, implementation details and variance [19], and Agarwal et al. argue for interval estimates and careful aggregation over few runs [20]. Pre-registration, in which analyses are specified before data are seen, is one proposed safeguard against flexible

analysis [21]. This paper adopts three safeguards in that spirit: a pre-specified validity gate for the environment, a negative control that scrambles the detector input, and paired bootstrap intervals over held-out episodes.

E. Position of this work

Unlike [4], which evaluates a threshold-based detection and tolerance mechanism, this work asks whether SNN and GNN components improve a budgeted recovery policy relative to classical detectors, and it makes the validity of the evaluation environment an explicit, tested object. It does not propose a deployable system.

III. MATERIALS AND METHODS

All experiments are simulations on synthetic data generated by the authors' own code; no real underwater sensor data, hardware, human participants or ethical approval are involved. Every constant below is taken from the committed code or pre-specification files of the project repository (branch v3, tags v3-final and v3-demo).

A. Software and hardware

Python 3.14.5, NumPy 2.5.0 [22], PyTorch 2.12.1 (CPU) [23], snnTorch 1.0.0 [7], PyTorch Geometric 2.8.0 [24], Gymnasium 1.3.0 [25], NetworkX 3.6.1 [26], SciPy 1.18.0 [27]. All runs were CPU-only on an Intel Core i5-8350U (4 physical, 8 logical cores, 15.9 GB RAM, Windows 11 Pro). The hardware description was queried after the experiments and was not logged at run time.

B. Simulated network and fault schedule

Each episode uses 30 nodes placed uniformly in a 3-D cube of side 620, with node 0 as the sink. Two nodes are linked if their Euclidean distance is within the communication range of 275. A topology is accepted only if it is connected and has at least two critical access points, defined as non-sink nodes whose removal cuts at least three other nodes off from the sink. Sampling is repeated until a topology is accepted (cap 5,000 draws). In the validation seeds, 100 of 1,656 drawn topologies (6.0%) were accepted.

An episode lasts 20 steps. With probability 0.30 it is a link-only episode: 2–4 hidden link faults occur at steps drawn from 2–14, and no node degrades or fails. Otherwise, it is a node-degradation episode: 7 non-sink nodes, drawn uniformly without replacement, degrade. Each starts degrading at an onset step drawn uniformly from 2–10 and fails L steps later, with L drawn uniformly from $\{3, 4\}$. In every episode, 3 further non-sink nodes act as decoys: they show a temporary sensor disturbance (onset 1–14, duration 3–5 steps) but never fail. The sink never fails. The fault schedule is fixed at reset, before the first action.

Within each step, events occur in this order: the controller's action, scheduled node failures, scheduled link faults, sensor sampling, battery drain (0.4 per step), and the reward computation.

C. Sensor model

Every alive node emits 5 samples per step on three channels with nominal values (20.0, 1.0, 0.8) and Gaussian noise of standard deviation 0.15. The detector window is the last 20 samples (the last 4 steps). A degrading node adds a linear drift that rises from zero at onset to amplitude A at failure, applied as $+A$ on channel 0 and $-A$ on channel 2. A decoy adds a half-sine

bump of peak A on channel 0 only. The amplitude $A = 0.3$ was chosen once on training seeds 1000–1499 as the grid value whose logistic-detector AUROC (0.845) was closest to 0.85, and was then frozen. AUROC, the area under the receiver operating characteristic curve, is the probability that a randomly chosen positive window is scored above a randomly chosen negative one [28]. This calibration used the Run 1 environment and was not repeated for Run 2.

D. Observation available to the controller

The controller sees 14 values per node plus 2 global values (remaining budget and time). The per-node values are: noisy battery and signal readings (noise SD 0.05), a silent flag, the previous silent flag, a deployed flag, a window z-statistic, a detector probability, four static-topology features (degree, hops to sink, distance to sink, downstream node count), a live downstream count, a criticality value, and a sink flag. A node is flagged silent once it has been unreachable from the sink for 2 consecutive observations, and any reporting node is falsely flagged with probability 0.02 per observation. A silent node's readings stay frozen at their last delivered values. The controller does not observe health, the fault schedule, which nodes are decoys, link states, or the true battery. It does know the static designed topology.

E. Actions, budget and reward

The action space has 31 discrete actions: no-op, or deploy a backup to one of the 30 nodes. The budget is 3 backups per episode. Deploying to a failed node restores it (battery 60, cost 1.0). Deploying to a node scheduled to fail pre-empts the failure (cost 1.0). Any other deployment, including to a healthy node, a decoy or the sink, is wasted: it costs 0.3 and still uses one backup. The reward at each step is minus the number of isolated alive nodes and dead non-sink nodes, minus the action cost; the episode return is the sum over 20 steps. Isolated means alive but without a path to the sink in the true graph.

F. Policies

- **no_op** and **random** (uniform over the 31 actions).
- **greedy[detector, criticality]**: score = detector probability $\times (1 + \text{criticality})$; deploy to the highest-scoring node if the score exceeds a threshold τ and budget remains. Criticality is none, static downstream count, or live downstream count. The threshold detector is $1[\text{window } z \geq \theta]$. The parameters τ (grid {0, 0.1, 0.2, 0.3, 0.5, 0.7, 1.0, 1.2, 1.5}) and θ (grid {2, 3, 4, 6}) are tuned on training seeds 3000–3099.
- **greedy_reactive_observed**: uses no detector; deploys to an observed-silent node that would reconnect the most silent nodes, breaking ties by static downstream count and then node id.
- **Oracles (use hidden state; references only)**: oracle_health (plans pre-emptions and restorations from the true schedule), oracle_health_random (random choice among failed or scheduled nodes), oracle_reactive (restores failed nodes only).
- **Negative control**: the selected greedy policy with sensor windows permuted across nodes within each episode.

G. Validity gate

Before any learned component was evaluated, ten pre-specified conditions (a–j) were checked on validation seeds 100–199 (100 episodes: 71 node-degradation, 29 link-only). The conditions are listed in Table I.

TABLE I
PRE-SPECIFIED VALIDITY CONDITIONS (A–J)

Item	Condition
a	oracle_health beats no_op: CI lower bound > 0 and difference $\geq 25\%$ of no_op's mean penalty (degradation episodes)
b	oracle_health beats oracle_reactive (early warning matters)
c	oracle_health beats oracle_health_random (node choice matters)
d	The best classical greedy, chosen on training seeds, captures 10–80% of oracle_health headroom
e	The largest single episode supplies $< 25\%$ of total headroom
f	Negative control: the permuted-window greedy captures $< 10\%$ of headroom
g	Logistic detector AUROC for "fails within 4 steps" between 0.75 and 0.95
h	A logistic model on battery, signal and silent flags only has AUROC < 0.60
i	oracle_health equals no_op in every link-only episode
j	Topology acceptance rate $\geq 5\%$

Headroom is the mean per-episode difference oracle_health – no_op. Headroom captured by a policy is $\text{mean}(\text{policy} - \text{no_op})$ divided by $\text{mean}(\text{oracle_health} - \text{no_op})$, a ratio of means. At most one environment revision was allowed after a failed gate, and a stop rule stated that any further failure of items c–j would end environment revision.

H. Detectors

All detectors predict whether a node fails within the next 4 steps from its 20-sample window. Training samples are alive non-sink nodes under the no-op policy.

- **Threshold**: the maximum over channels of the z-score of the last-5-sample mean (0 parameters).
- **Logistic**: L2-regularized logistic regression [29] (penalty 1.0, untuned) on the standardized 60-value window, fitted by Newton iterations on training seeds 2000–2599 (320,813 windows, 3.59% positive; 61 parameters; no class weights).
- **SNN**: snnTorch leaky integrate-and-fire network (beta 0.9, fast-sigmoid surrogate gradient [12], [13]), Linear(3→h) → LIF → Linear(h→2) → LIF, run for 20 time steps with one window sample per step as direct analog input (not event-encoded). The output is the spike count per class; the detector probability is the softmax of the two counts. Parameters: $6h+2$.
- **Size-matched MLP**: a shared per-sample Linear(3→h) with ReLU, mean over the 20 samples, Linear(h→2); the same $6h+2$ parameters, with no spiking or leak.
- **GRU**: one gated recurrent layer [30] (3→h) and a linear output (1,042 parameters at $h = 16$); it is not size-matched.

The neural detectors were trained identically: Adam [31], batch 512, 15 epochs, cross-entropy with positive-class weight 26.88 (negatives/positives) as a standard response to class imbalance [32], no weight decay, per-channel normalization

from training seeds 2000–2599. For each architecture and training seed (0, 1, 2), the configuration (hidden size {16, 32} × learning rate {1e-3, 5e-4}) and epoch with the best AUROC on selection seeds 2600–2799 were kept, giving 36 training runs. Spike sparsity is $1 - \text{spikes} / (n \times (h+2) \times 20)$ and is reported only as a proxy; no energy was measured.

I. Graph neural network

A two-layer graph convolutional network [8] (GCNConv 11→32, 32→32, then Linear 32→2; 1,506 parameters) takes the static topology and 11 observed node features and predicts whether a node is isolated 2 steps ahead (the silence delay). The sink and nodes dead at that time are masked. Training: Adam, learning rate 1e-3, weight decay 1e-5, 40 epochs, batch of 64 graphs, positive-class weight 12.06, seeds 0, 1, 2, epoch chosen by AUROC on seeds 2600–2799, decision threshold at maximum F1 on the same seeds. It was compared with always-safe, an upstream single-point-of-failure heuristic, and the silent flag.

J. Claim tests and statistics

Each claim test compares two greedy policies on the same 300 test episodes (seeds 200–499), after tuning tau (and theta) on seeds 3000–3099. For a detector with three training seeds, the per-episode difference is averaged over seeds, and a percentile bootstrap [33], [34] over the 300 episodes (10,000 resamples, bootstrap seed 0) gives the 95% interval. The verdict rule: **tie** if the interval contains 0; **improves** if the mean is positive, the interval excludes 0 and at least 2 of 3 seeds are positive; **worse** if the mean is negative and the interval excludes 0; otherwise **inconclusive**. The SNN-versus-logistic comparison was the primary SNN test; the GNN comparison (GNN criticality versus live static criticality) was pre-specified with an expected tie. Test seeds 200–499 were used once.

K. Seed roles

Table II gives the role of each seed range.

TABLE II
SEED ROLES

Seeds	Role
0–99	Reserved, never run in the final environment
100–199	Validity-gate validation
200–499	Final claim tests, used once
1000–1499	Drift calibration
2000–2599	Detector and GNN training
2600–2799	Epoch and configuration selection
3000–3099	Tuning of tau and theta
≥ 5000	Software tests (12 tests, 15 seeded fault mutants, all caught)

L. Pre-specification record

The gate conditions, claim tests and verdict rules were written in version-controlled files (criteria.md; stage2_criteria.md) and committed before the runs they govern. These are commit-time records, not an external registry [21]. Run 1 failed item d. A dated amendment, committed before Run 2, changed how the degrading set is drawn and added the reactive baseline, the re-selection rule and the stop rule. Run 2 failed item f, so the stop rule applied. Stage 2 criteria were written after both gate runs and are labeled exploratory. Post hoc analyses are labeled as such wherever they appear.

M. Earlier synthetic detector experiment (Phase 2)

A separate, earlier experiment compared an SNN (h = 32, 194 parameters) with a dense MLP (60→64→64→2, 8,194 parameters) on a different synthetic generator (2,000 windows of 20 × 3, 30% anomalous: drift, spike or dropout), split 80/20 in order with no shuffling. Both models used Adam (learning rate 5e-4), batch 32, class weights [1, 2] and 30 epochs. Normalization statistics were computed on all 2,000 samples before the split. For the SNN, ties in spike counts are predicted as "normal". The saved models were re-evaluated for this paper and reproduced every reported figure.

N. Gaps in the project record

- The budget of 3 backups, the action costs (1.0 and 0.3), K = 7, T = 20, P(link-only) = 0.3 and the onset and L ranges were taken from the project brief. They are design choices, not values measured in a real deployment, and the brief itself is not stored in the project record.
- The hardware and run times of the earlier experiments (Phases 1–5), including the Phase 2 task in Section IV–G, were not recorded.

IV. RESULTS

The simulated environment failed its pre-specified validity gate, so every result for a learned component below is exploratory. All data are synthetic.

A. Validity gate (pre-specified; validation seeds 100–199)

Run 1 passed 9 of 10 items and failed item d: the best classical greedy captured 82.9% of headroom, above the 80% limit. After the single permitted revision, Run 2 passed 9 of 10 items and failed item f: the negative control captured 13.6%, above the 10% limit. The stop rule applied and no further revision was made. The item-by-item results are in Table III.

TABLE III
VALIDITY GATE RESULTS, RUN 1 AND RUN 2

Item	Run 1	Run 2	Run 2 value
a	pass	pass	oracle_health – no_op = +79.01 [64.38, 96.15]; 66.4% of no_op penalty
b	pass	pass	+22.27 [14.37, 31.10]
c	pass	pass	+25.66 [16.46, 36.75]
d	fail (82.9%)	pass	77.5% captured
e	pass	pass	largest episode 6.7% of headroom
f	pass (9.0%)	fail	13.6% captured
g	pass (0.860)	pass	AUROC 0.861
h	pass (0.574)	pass	AUROC 0.514
i	pass	pass	29 link-only episodes; oracle_health = no_op in all
j	pass	pass	acceptance 6.0%; mean 2.34 critical access points

Brackets are 95% bootstrap intervals. Item i holds by construction, because a deployment never changes links.

B. Policy baselines (Run 2, validation seeds 100–199, 100 episodes)

Mean per-episode headroom (oracle_health – no_op) was 56.10. Mean returns were –84.70 for no_op and –28.60 for

oracle_health. Intervals exist only where the analysis stored them; they were computed after item f failed (post hoc). Table IV gives the headroom captured by each policy.

TABLE IV
 HEADROOM CAPTURED BY POLICY (RUN 2, VALIDATION SEEDS 100–199)

Policy	Headroom captured
no_op	0.0%
random	2.9% [–3.0, 10.1]
greedy_reactive_observed	23.4% [12.7, 34.6]
greedy[threshold, static]	49.4%
greedy[threshold, none]	58.9%
greedy[threshold, static_live]	64.0%
oracle_health_random	67.5%
oracle_reactive	71.8%
greedy[logistic, none]	77.4%
greedy[logistic, static] (selected, tau 0.7)	77.5% [67.4, 86.9]
greedy[logistic, static_live]	77.5%
negative control (selected greedy, windows permuted)	13.6% [7.1, 21.9]

A post hoc check found that a random deployer captured 13.9% [8.1, 21.6] of headroom before action costs, against 15.4% for the negative control (difference +0.84 [–4.81, +6.82]). The negative control therefore did not exceed what random deployment gains before costs.

C. Detectors (exploratory; test seeds 200–499)

Table V reports the detector parameters and test AUROC.

TABLE V
 DETECTOR PARAMETERS AND AUROC (TEST SEEDS 200–499)

Detector	Parameters	AUROC (mean $\pm$ SD over 3 seeds)
threshold	0	0.792 (single fit)
size-matched MLP	194	0.810
SNN	194 / 98 / 98	0.815 $\pm$ 0.006
logistic	61	0.861 (single fit)
GRU	1,042	0.868

SNN spike sparsity was 0.79 ± 0.07 , a proxy only. At the threshold meant to give a 5% false-alarm rate, the logistic, MLP and GRU detectors reached 4.9–5.0% on the test seeds, but the SNN's discrete scores (17–20 distinct values) gave 9.7%, 27.3% and 33.9%. The SNN lead time is therefore not reported.

D. Claim tests (exploratory; 300 test episodes $\times$ 3 seeds)

Table VI lists the claim tests.

TABLE VI
 CLAIM TESTS (300 TEST EPISODES $\times$ 3 SEEDS)

Comparison (reward per episode)	Status	Mean difference [95% CI]	Verdict
greedy(SNN) – greedy(logistic)	pre-specified primary	–11.78 [–17.23, –6.33]	worse (3 of 3 seeds)
greedy(MLP) – greedy(logistic)	pre-specified secondary	–11.03 [–16.55, –5.44]	worse

greedy(GRU) – greedy(logistic)	pre-specified secondary	–10.88 [–16.41, –5.30]	worse
greedy(threshold) – greedy(logistic)	pre-specified secondary	–16.03 [–21.03, –11.14]	worse
greedy(SNN) – greedy(MLP)	pre-specified secondary	–0.75 [–1.34, –0.31]	worse
greedy(SNN) – greedy(GRU)	pre-specified secondary	–0.90 [–1.51, –0.42]	worse
greedy(logistic, GNN criticality) – greedy(logistic, live static)	pre-specified (expected tie)	–8.84 [–13.79, –4.65]	worse

On the test seeds, the logistic greedy captured 83.2% of headroom (mean test headroom 61.09 per episode; 86 of 300 episodes link-only) and the SNN greedy 64.7%, 63.1% and 64.1% for the three seeds. No learned detector or GNN criticality improved on its classical comparator.

E. Graph neural network predictor (exploratory)

The GNN predicted isolation 2 steps ahead with AUROC 0.946, 0.947 and 0.945 (three seeds), against 0.522 for the upstream-single-point-of-failure heuristic, 0.804 for the silent flag and 0.500 for always-safe.

F. Post hoc checks (not pre-specified)

- **GNN versus a live-reachability rule:** the rule scored AUROC 0.805 [0.796, 0.813]; the GNN was higher by +0.141, +0.142 and +0.140 across seeds (each interval excludes 0). Of the GNN's positive labels, 62.8% were already silent. On nodes not yet flagged silent, the GNN scored 0.858, 0.860 and 0.856 against 0.516 for the rule.
- **SNN versus threshold greedy:** +4.25 [+2.05, +6.96], positive in 3 of 3 seeds.
- **Calibration diagnostic (selection seeds 2600–2799):** all neural detectors selected tau = 1.0 and made about 0.5 successful deployments per episode, against 2.00 for the logistic detector at tau = 0.7. Their mean score on negative windows was 0.27–0.41, against 0.021 for the logistic detector.
- **Training budget:** 18 of 36 detector runs, and 6 of 9 selected instances, reached their best epoch at the final epoch of the 15-epoch budget.

G. Earlier synthetic detector task (Phase 2; not pre-specified)

Table VII reports the results of the earlier task.

TABLE VII
 EARLIER SYNTHETIC DETECTOR TASK (PHASE 2)

Model	Accuracy	Precision	Recall	Parameters	Spike sparsity
SNN	93.00%	100.00%	75.00%	194	0.835
MLP	95.75%	94.39%	90.18%	8,194	not applicable

The split has 400 test samples (SNN confusion counts TP 84, FN 28, FP 0, TN 288). The SNN's 100% precision depends on the tie rule: 70 of 400 test samples have equal spike counts and

are called normal. If ties were called anomalous, precision would be about 64% and recall about 88%.

V. DISCUSSION

Direct findings. In the simulated environment, the pre-specified validity gate was not met in either run. In Run 2, a policy using the logistic detector captured 77.5% of the headroom available to an oracle, yet the permuted-window negative control captured 13.6%, above the 10% limit. In the exploratory tests, neither the SNN nor the GNN criticality improved on its classical comparator: every pre-specified comparison was worse, including the GNN comparison that had been expected to tie.

Interpretation supported by the record. The post hoc analyses are consistent with the SNN result reflecting calibration rather than detection quality alone. The size-matched MLP and the GRU lost to the logistic greedy by about the same margin (about 11 reward units), all neural detectors selected $\tau = 1.0$, and the neural detectors were trained with a positive-class weight of 26.88 that the logistic detector did not receive. Their mean score on negative windows was therefore far higher (0.27–0.41 against 0.021), and they deployed far fewer successful backups at the selected τ . This fits the general observation that modern neural networks are often poorly calibrated [35] and that reweighting or resampling for class imbalance changes the scores a classifier outputs [32]; here it is offered as a consistent explanation, not as a tested one. The SNN was 0.75 reward units below the MLP of equal size and 4.25 units above the threshold rule (post hoc). The τ grid contains no value between 0.7 and 1.0, and the effect of a finer grid or of calibrated probabilities was not tested.

The SNN's discrete output also limited the comparison. A spike-count readout over 20-time steps yields few distinct scores, so a target false-alarm rate could not be met (Section IV-C). The SNN received direct analog input rather than event-encoded spikes, whereas the neuromorphic literature stresses that the choice of spike encoding interacts with learning [7], [14]; the result therefore says little about SNNs fed by event-based sensors.

For the GNN, the pre-specified label measures whether a node itself becomes isolated, whereas the greedy policy uses criticality to rank how much damage a node's failure would cause. This mismatch was noted before the run and is a plausible reason for the worse result, but it was not tested. The GNN did predict isolation better than a live-reachability rule (AUROC 0.946 against 0.805), including on nodes not yet flagged silent (0.858–0.860 against 0.516, post hoc), which indicates that graph learning captured structure that a simple rule did not [8], [9], [18].

Comparison with the literature. The anchor study [4] pairs dynamic thresholding with multi-path rerouting and reports improved detection accuracy, false alarms, latency and lifetime over traditional schemes in MATLAB simulations. The present results are not directly comparable, because the network model, fault model and metrics differ, but they are consistent in one respect: in this environment a well-tuned classical detector was hard to beat, and the threshold-style baselines already captured 49–64% of headroom. The finding that learned detectors did not outperform a logistic baseline also echoes the broader caution that energy and accuracy advantages claimed for SNNs depend on sparsity, hardware and implementation [16], [17]. On

evaluation practice, the experience reported here parallels the sensitivity of learned-controller results to environment design and to variance reported for deep reinforcement learning [19], [20], and the initial invalid environment is a concrete instance of reward exploitation [10].

Speculation (not tested). Longer training could change the SNN result, since 6 of 9 selected detectors peaked at the final epoch. Re-weighting the loss or calibrating the scores before thresholding might narrow the gap between neural and logistic detectors. These are hypotheses only.

Contribution. The work contributes a documented simulation environment with a pre-specified validity gate, a record of how that gate failed twice, and exploratory comparisons that found no improvement from the SNN detector or the GNN criticality. It does not demonstrate improved recovery, energy savings, or superiority of any neuromorphic or graph-based component.

VI. LIMITATIONS

- **Environment validity.** The gate was failed twice; all learned-component results are exploratory and should not be read as evidence about real underwater networks.
- **Synthetic data only.** Sensor signals, faults and topologies are generated from the authors' own model; no real deployment data were used.
- **Hand-chosen constants.** Fault counts, timings, costs and the drift amplitude were set by design; the amplitude was calibrated on an earlier version of the environment.
- **Calibration and training asymmetry.** The neural detectors used a positive-class weight and a shared τ grid that the logistic detector did not; this probably accounts for much of their deficit and was not isolated experimentally.
- **Short training budget.** Fifteen epochs, with 6 of 9 selected detectors at their last epoch.
- **SNN input and energy.** The SNN receives direct analog input rather than event-encoded spikes; sparsity is only a proxy and no energy was measured. Discrete SNN scores prevented a valid 5% false-alarm lead time.
- **GNN label.** The label measures a node's own isolation, not the damage it causes.
- **Rule-based controllers only.** No learned controller is reported. An earlier reinforcement-learning attempt used a different environment that was later found invalid (Section VIII).
- **Reactive baseline.** The single observation-only reactive rule waits for the silent flag and ignores observed battery; a stronger reactive rule was not tested.
- **Post hoc material.** Intervals on several baselines and the whole of Section IV-F were produced after results were seen.
- **Earlier synthetic task.** A single split of 400 samples, no confidence intervals, normalization on all samples before the split, a non-size-matched MLP, and a tie-dependent SNN precision.

- **Pre-specification is internal.** Pre-specification files are commit-time records in the project repository, not an external registry [21].
- **Unresolved provenance.** The basis for several design constants is not recorded.

VII. CONCLUSION

The simulated environment did not pass its pre-specified validity gate, so conclusions about controllers in it are limited. Within that limit, no evidence was found that a spiking detector or a graph-based criticality estimate improves greedy fault recovery over classical alternatives; in the tested configurations they performed worse. The most reliable contribution is the documented evaluation framework and the account of why its gate failed.

VIII. DISCLOSURE OF EARLIER WORK

The project began with a simulator and a reinforcement-learning agent whose reported improvements (for example +24.04 and +26.83 over random) were later invalidated: one action dominated the reward, most apparent isolation came from the sink node failing, and ablation runs shared a random stream. A graph-network accuracy figure from that period was also withdrawn because the saved checkpoint was the last epoch and not the best. None of these figures is used in this paper. An earlier, easier version of the Phase 2 task (97% accuracy, 91% recall) was replaced the same day by the harder task reported in Section IV-G.

IX. FUTURE WORK (PROPOSED, NOT PERFORMED)

- Repeat the SNN comparison with matched loss weighting and a finer or calibrated threshold grid.
- Train the GNN on a label that measures damage caused by a node's failure.
- Revise the environment so that the permuted-window negative control passes, then re-run the claim tests on fresh seeds.
- Test a reactive baseline that uses observed battery.
- Evaluate a learned controller only after the environment passes its gate.
- Measure energy on neuromorphic hardware or a validated model before any energy claim [15], [16].

DECLARATION OF AI ASSISTANCE

Claude (Anthropic) was used to draft and edit the manuscript text, to check references against publisher and database records, and to research specific topics. author has reviewed all text, checked it against the project records, verified every reference in Google Scholar, and take responsibility for all content.

DECLARATIONS

Author contributions: Madhura MD carried out the project work. Yashwanth B guided the project.

Funding: The authors received no external funding for this work.

Competing interests: The authors declare no competing interests.

Data and code availability: the simulation code, pre-specification files and results are available at <https://github.com/madhuramd/Neurographheal> (branch v3).

Acknowledgments: None.

REFERENCES

- [1] I. F. Akyildiz, D. Pompili, and T. Melodia, "Underwater acoustic sensor networks: research challenges," *Ad Hoc Netw.*, vol. 3, no. 3, pp. 257–279, 2005, doi: 10.1016/j.adhoc.2005.01.004.
- [2] J. Heidemann, M. Stojanovic, and M. Zorzi, "Underwater sensor networks: applications, advances and challenges," *Phil. Trans. R. Soc. A*, vol. 370, no. 1958, pp. 158–175, 2012, doi: 10.1098/rsta.2011.0214.
- [3] T. Muhammed and R. A. Shaikh, "An analysis of fault detection strategies in wireless sensor networks," *J. Netw. Comput. Appl.*, vol. 78, pp. 267–287, 2017, doi: 10.1016/j.jnca.2016.10.019.
- [4] Sharshika, Y. Chaba, and S. Khambra, "Design and evaluation of an adaptive fault detection and tolerance mechanism for underwater wireless sensor networks," *Int. J. Comput. Exp. Sci. Eng. (IJCESN)*, vol. 11, no. 4, pp. 7587–7598, 2025, doi: 10.22399/ijcesen.4056.
- [5] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural Netw.*, vol. 10, no. 9, pp. 1659–1671, 1997, doi: 10.1016/S0893-6080(97)00011-7.
- [6] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, pp. 607–617, 2019, doi: 10.1038/s41586-019-1677-2.
- [7] J. K. Eshraghian et al., "Training spiking neural networks using lessons from deep learning," *Proc. IEEE*, vol. 111, no. 9, pp. 1016–1054, 2023, doi: 10.1109/JPROC.2023.3308088.
- [8] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017, arXiv:1609.02907.
- [9] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, 2021, doi: 10.1109/TNNLS.2020.2978386.
- [10] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in AI safety," arXiv:1606.06565, 2016.
- [11] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: a survey," *ACM Comput. Surv.*, vol. 41, no. 3, art. 15, 2009, doi: 10.1145/1541880.1541882.
- [12] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks: bringing the power of gradient-based optimization to spiking neural networks," *IEEE Signal Process. Mag.*, vol. 36, no. 6, pp. 51–63, 2019, doi: 10.1109/MSP.2019.2931595.
- [13] F. Zenke and S. Ganguli, "SuperSpike: supervised learning in multilayer spiking neural networks," *Neural Comput.*, vol. 30, no. 6, pp. 1514–1541, 2018, doi: 10.1162/neco_a_01086.
- [14] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: opportunities and challenges," *Front. Neurosci.*, vol. 12, art. 774, 2018, doi: 10.3389/fnins.2018.00774.
- [15] M. Davies et al., "Loihi: a neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018, doi: 10.1109/MM.2018.112130359.
- [16] M. Dampfhofer, T. Mesquida, A. Valentian, and L. Anghel, "Are SNNs really more energy-efficient than ANNs? An in-depth hardware-aware study," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 7, no. 3, pp. 731–741, 2023, doi: 10.1109/TETCI.2022.3214509.
- [17] A. Davidson and S. Furber, "Comparison of artificial and spiking neural networks on digital hardware," *Front. Neurosci.*, vol. 15, art. 651141, 2021, doi: 10.3389/fnins.2021.651141.
- [18] K. Rusek, J. Suárez-Varela, P. Almasan, P. Barlet-Ros, and A. Cabellos-Aparicio, "RouteNet: leveraging graph neural networks for network modeling and optimization in SDN," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2260–2270, 2020.
- [19] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 3207–3214.
- [20] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare, "Deep reinforcement learning at the edge of the statistical precipice," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 34, 2021.

- [21] B. A. Nosek, C. R. Ebersole, A. C. DeHaven, and D. T. Mellor, "The preregistration revolution," *Proc. Natl. Acad. Sci. U.S.A.*, vol. 115, no. 11, pp. 2600–2606, 2018, doi: 10.1073/pnas.1708274114.
- [22] C. R. Harris et al., "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, 2020, doi: 10.1038/s41586-020-2649-2.
- [23] A. Paszke et al., "PyTorch: an imperative style, high-performance deep learning library," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 32, 2019.
- [24] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019, arXiv:1903.02428.
- [25] M. Towers et al., "Gymnasium: a standard interface for reinforcement learning environments," arXiv:2407.17032, 2024.
- [26] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proc. 7th Python in Science Conf. (SciPy 2008)*, 2008, pp. 11–15.
- [27] P. Virtanen et al., "SciPy 1.0: fundamental algorithms for scientific computing in Python," *Nat. Methods*, vol. 17, pp. 261–272, 2020, doi: 10.1038/s41592-019-0686-2.
- [28] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, 2006, doi: 10.1016/j.patrec.2005.10.010.
- [29] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. New York, NY, USA: Springer, 2009.
- [30] K. Cho et al., "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Nat. Lang. Process. (EMNLP)*, 2014, pp. 1724–1734.
- [31] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015, arXiv:1412.6980.
- [32] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, 2009, doi: 10.1109/TKDE.2008.239.
- [33] B. Efron, "Bootstrap methods: another look at the jackknife," *Ann. Statist.*, vol. 7, no. 1, pp. 1–26, 1979, doi: 10.1214/aos/1176344552.
- [34] B. Efron and R. J. Tibshirani, *An Introduction to the Bootstrap*. New York, NY, USA: Chapman & Hall, 1993.
- [35] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *Proc. Int. Conf. Mach. Learn. (ICML)*, PMLR vol. 70, 2017, pp. 1321–1330.