Early Detection of Silent Training Failures in Neural Networks

Sohan Bhat
sohanrambhat@gmail.com

November 2026

Abstract

During training, neural networks can develop defects that raise no error, so the run completes and returns a degraded model that is easy to mistake for a hard problem. This work presents LHD (Layer Health Diagnostics), a monitor that logs four signals for every layer at the end of each epoch (weight change, gradient norm, weight norm, and the fraction of dead units) and compares each one with the range it stays within during healthy runs of the same network. When a signal stays outside that range for three epochs in a row, LHD names the layer and a likely cause and stops the run. LHD was tested on 885 training runs of a fully connected network, a CNN, and a small transformer, in which frozen layers and dead units were planted in known layers. On the 105 broken runs whose fault cost two or more points of final accuracy, LHD caught 94 and the accuracy curve caught 60. Across all 470 broken runs with these faults, LHD caught 87% (95% interval 84 to 90%) and named the planted layer in 98% of the runs it caught, while it flagged 3 of 150 fresh healthy runs (0.7 to 5.7%). On the 162 runs that both LHD and the accuracy curve caught, LHD fired 4.6 epochs earlier on average. A check for weights that never change caught every frozen layer two epochs sooner, so LHD helps most with dead units, where it caught 78% of faults while the fixed thresholds used by existing tools either missed most of them or flagged many healthy runs.

1 Introduction

During training, neural networks can develop defects, yet nothing fails visibly since the run completes normally and simply produces a degraded result; thus, these are called "silent failures." Because of this, identifying whether the degraded result was caused by a difficult problem or a silent failure becomes difficult; therefore, it is common to misread a failure as normal. I initially encountered this as I was training my small-scale GTSRB (German Traffic Sign Recognition Benchmark) model (Stallkamp et al., 2012) from scratch, where my model plateaued at 76%, wasting weeks tuning it, until I found one convolutional layer had been accidentally excluded from the optimizer and was never properly trained. Fixing this minute detail raised the accuracy to 87%. This is not an isolated case: errors in deep learning implementations rarely cause an exception and instead reveal themselves only through degraded predictive performance (Chwilczyński et al., 2024).

Since training runs consume significant compute, every silent failure that goes undetected wastes it. The evidence existed in the form of raw numbers in the weights and biases, but perceiving that is challenging since nothing in standard practice surfaces them. To address this, this work presents LHD (Layer Health Diagnostics), a monitor that reads these numbers, catches the failure within a few epochs, identifies the broken layer and its likely cause, and halts the run before compute is wasted.

The clearest result is on the faults that did real damage. Of the 105 broken runs whose fault cost two or more points of final accuracy, LHD caught 94 and the accuracy curve caught 60. Those runs are a minority, since 223 of the 470 planted faults cost less than half a point. Across all 470 broken runs with a frozen layer

or dead units, LHD caught 411 of 470, usually at epoch 3, while the accuracy curve caught 166 of 470. On the 162 runs both caught, LHD was earlier by 4.6 epochs on average, and averaged within each seed, all 30 seed groups favored LHD. LHD also flagged 3 of 150 fresh healthy runs, against 7 of 150 for the accuracy curve. However, a simple check for weights that do not change caught every frozen layer two epochs before LHD, so the advantage of LHD comes from dead units. There it caught 78% with 3 false alarms, while the default rule of SageMaker Debugger, held to the same three epochs and counted only when it named the planted layer, caught 40% and flagged 22 of 150 healthy runs. Only frozen layers and dead units were planted in the main study, and they were planted to show up in the signals LHD measures, so these results describe planted faults rather than bugs found in real projects.

2 Related Work

Karpathy's Recipe. In 2019, Andrej Karpathy wrote a widely cited guide for the training of neural networks where he discusses how training fails silently (Karpathy, 2019). He provides manual habits to catch these silent failures, and the Stanford CS231n course notes add checking that the update-to-weight ratio is near 0.1% (CS231n, n.d.). However, his guide simply provides a checklist a human needs to follow by hand with no mention of any automated solution.

wandb.ai. Weights & Biases (wandb) is the industry standard for experiment tracking through a dashboard where one line of code in a project can log live gradient and weight histograms during training (Biewald, 2020). Still, it is limited by only displaying the numbers and nothing more, since a human still needs to watch, understand what healthy numbers look like, and catch the anomaly.

wandb-callbacks. Wandb-callbacks is a small community add-on package for wandb that includes a dead-ReLU warning callback, which by default warns when at least 80% of a layer's units never activate (Groeger, n.d.). However, it covers only one failure mode with a single signal and provides no diagnosis or halting.

DeepLocalize. DeepLocalize, published at the 2021 International Conference on Software Engineering, watches the values a network produces at each layer during training and, when it finds a numerical error, traces the error back to the layer that caused it (Wardat et al., 2021). However, it only reports faults that lead to numerical errors and does not suggest how to fix them (Wardat et al., 2022).

UMLAUT. UMLAUT, presented at the 2021 ACM Conference on Human Factors in Computing Systems, checks a deep learning program's structure and its model's behavior against debugging heuristics drawn from experts, then shows readable error messages next to the code and the model's output in a single interface (Schoop et al., 2021). In a study with 15 participants, people using it found and fixed significantly more bugs than without it. Still, its heuristics fire on models with no known bugs, and the authors of DeepDiagnosis observed it warning of possible overfitting on 68 of 78 such MNIST models (Wardat et al., 2022).

SageMaker Debugger. Amazon SageMaker Debugger records tensors during training jobs on Amazon's cloud and checks them with built-in rules such as DeadRelu, VanishingGradient, ExplodingTensor, UnchangedTensor, and WeightUpdateRatio (Amazon Web Services, n.d.a). Any rule can be paired with an action that stops the training job when the rule fires (Amazon Web Services, n.d.b), so detection and halting are already automated. However, its dead unit, gradient, and unchanged tensor rules compare a signal against a fixed default threshold. DeadRelu, for example, fires when more than 50% of a layer's ReLUs are active less than 1% of the time, and a user training a different network has to pick a new threshold by

hand. Its SimilarAcrossRuns rule compares a job with one other completed job rather than with the range seen across many healthy runs. Amazon's documentation also states that Debugger is no longer open to new customers.

AutoTrainer. AutoTrainer is a training monitor and automatic repair system published at the 2021 International Conference on Software Engineering (Zhang et al., 2021). During training, it checks gradients, weights, and the loss and accuracy curves against preset thresholds to detect five problems: vanishing gradients, exploding gradients, dying ReLU, oscillating loss, and slow convergence. When it finds one, it retrains the model with built-in fixes such as replacing the activation function, adding batch normalization, or changing the weight initializer. However, it does not report which layer is at fault (Wardat et al., 2022), and its default thresholds are the same for every network. A later study of model repair on 9 real and 21 artificial faulty models found that a random search did as well as or better than AutoTrainer and two hyperparameter tuning methods, and that all of them failed on larger models (Kim et al., 2023).

DeepDiagnosis. DeepDiagnosis, published at the 2022 International Conference on Software Engineering, checks a model at the end of every training epoch for eight error symptoms, including dead nodes, unchanged weights, saturated activations, and vanishing or exploding gradients (Wardat et al., 2022). When it finds a symptom, it reports the layer, maps the symptom to a likely root cause, and suggests a fix, which is close to the layer, cause, and suggested check that LHD reports. However, its checks also rely on fixed thresholds, such as counting a layer as having dead nodes when more than 70% of its ReLU units are inactive. On 188 models from the AutoTrainer benchmark with no known bugs, it still reported symptoms, including dead nodes in 21 of the 35 CIFAR-10 models (Wardat et al., 2022).

ART. Actually Robust Training is a Python library inspired by Karpathy's recipe that adds validation checks to training pipelines (Chwilczyński et al., 2024). While this is a step up, it only imposes process steps at development time rather than live fault detection from internal signals during training.

To conclude, tools that detect faults during training, and even stop or repair the run, already exist. However, their dead unit and gradient checks compare each signal against a threshold chosen in advance, such as 50%, 70%, or 80% of a layer's units being dead, even though the share of units that die during healthy training differs between networks and between layers. A threshold that suits one network can therefore miss real faults in another or flag its healthy runs, as DeepDiagnosis's dead node check did on CIFAR-10 models with no known bugs. LHD instead measures a separate healthy range for every layer, signal, and epoch from healthy runs of the network being trained. Furthermore, DeepDiagnosis is evaluated by how many buggy models it flags and how quickly, not by how many epochs earlier its signals catch a fault than the accuracy curve or by how often it flags fresh healthy runs of the same network, and those are the two measurements this work reports.

3 Method

3.1 Signals

At the end of every epoch, LHD records four signals for each layer it watches. Let W_e be the layer's weights and bias at the end of epoch e, flattened into one vector, and let $\|\cdot\|$ be the square root of the sum of squared entries. The weight change, $\|W_e - W_{e-1}\|/\|W_{e-1}\|$, measures how far the weights moved during the epoch relative to their size, so a frozen layer reads exactly 0. The weight norm, $\|W_e\|$, grows past normal before a run diverges. The gradient norm is the mean of $\|\partial L/\partial W\|$ over the epoch's batches, taken after each backward pass and before the optimizer step, and a vanishing gradient reads near 0. Finally, the dead unit

Table 1: How LHD names a cause. The rules are checked from top to bottom on the layer that raised the alarm.

When the alarmed layer shows	LHD names
a signal that is not a finite number	diverged
more dead units than its range allows	dead units
weight change below its range while the gradient is inside or above	frozen layer
weight change and gradient both below their ranges	vanishing gradient
weight norm above its range	runaway weights
any other pattern	unrecognized

fraction is the share of the layer's units whose output after the ReLU is exactly 0 for every input in a fixed probe batch of 256 validation examples. A unit is one neuron of a dense layer or one filter of a convolution, and a filter counts as dead only when it outputs 0 at every position. The probe batch is the same in every run and passes through the network in evaluation mode without gradients.

3.2 Healthy range

The healthy range comes only from calibration runs, which are trained with no defects from different random starting points. For layer ℓ, signal s, and epoch e, let $a_{\ell se}$ and $b_{\ell se}$ be the lowest and highest values across the calibration runs. A value x is out of range when

$$x < a_{\ell se}(1-m) \quad \text{or} \quad x > b_{\ell se}(1+m) + f, \tag{1}$$

with margin $m = 0.5$ and floor $f = 0.05$, and a value that is not a finite number is always out of range. The floor keeps a signal that is 0 in every healthy run, such as the dead unit fraction of most layers, from flagging a single unusual unit. Broken runs are never used to set the range, since a detector tuned on a known fault only catches that fault.

3.3 Alarm and diagnosis

LHD raises an alarm once one signal of one layer stays out of range for $p = 3$ epochs in a row. If several layers reach that point in the same epoch, it reports the first one in the order data flows through the network. The settings m, f, and p come from the tests of the LHD library, written on 12 October 2026, and were committed to the experiment code on 31 October 2026 (commit `e83806f`) before any of the study's training runs started. An earlier draft of this work defined the threshold as a multiple of the largest healthy value, and the rule above replaced it, adding the floor and a margin below the lowest healthy value, before any run was trained. LHD then names a likely cause from the pattern across the alarmed layer's four signals (Table 1), stops training, and reports the layer, the signals out of range, the cause, and a suggested check. For a frozen layer, for example, it suggests checking that the layer's parameters are passed to the optimizer. The main study planted only frozen layers and dead units, so it tested only the frozen layer and dead unit rules. The vanishing gradient and runaway weights rules were exercised only by the extra faults in Section 5.9, and the divergence rule never fired in any run. Figure 1 shows LHD stopping one CNN run with a frozen layer.

4 Experimental Setup

4.1 Networks and data

LHD was tested on three networks with different structures (Table 2). The fully connected network has hidden layers of 256, 128, and 64 units. The CNN has three convolutions with 8, 16, and 32 filters, each

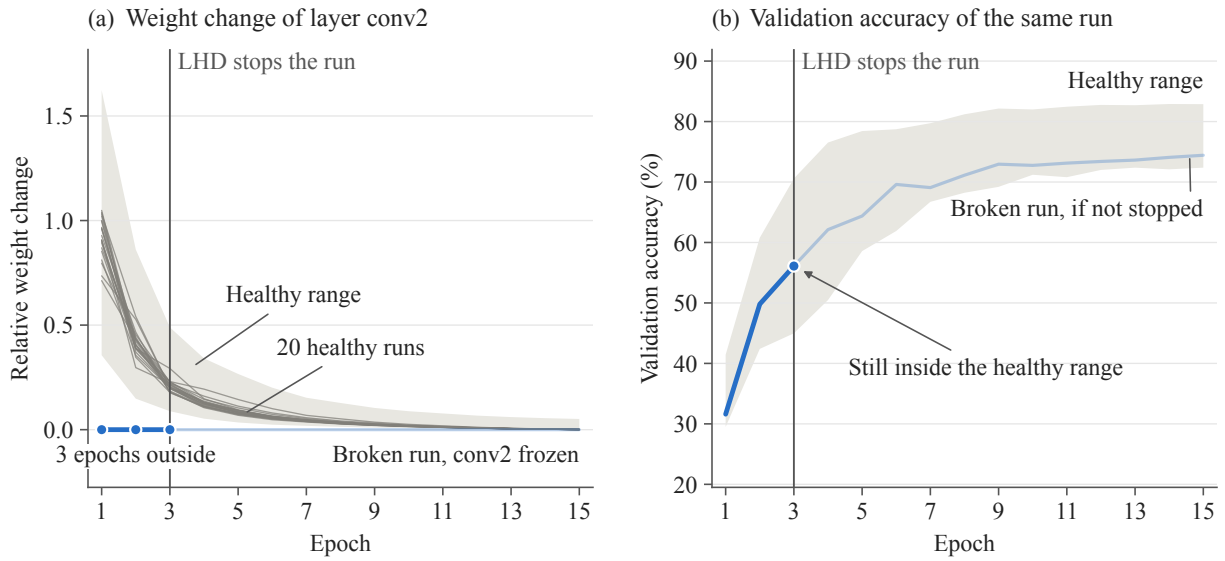


Figure 1: One CNN run with a frozen layer. (a) The frozen layer's weight change against the range of the 20 calibration runs. It stays at 0, outside the range, and LHD stops the run after three epochs. (b) Validation accuracy of the same run, which is still inside the healthy range when LHD stops it.

Table 2: The three networks and the runs trained for each. Calibration runs set the healthy range, fresh healthy runs measure false alarms, and the main results use the runs with a frozen layer or dead units. Runs with other faults are reported in Section 5.9.

				Runs			
Network	Dataset	Parameters	Epochs	Calibration	Fresh healthy	Frozen or dead units	Other faults
Fully connected	Fashion-MNIST	242,762	20	20	50	130	55
Transformer	Tiny Shakespeare	112,577	15	20	50	170	85
CNN	GTSRB	41,659	15	20	50	170	65

followed by a ReLU and max pooling, then a dense layer of 64 units and the output layer. The transformer predicts the next character with a width of 64, 4 attention heads, 2 blocks, and a feed-forward layer of 256 units that uses a ReLU, so dead units can be counted the same way in all three. After every epoch, validation accuracy is measured on the dataset's test split. The CNN is scored on a fixed random subset of 4,000 test images and the transformer on 512 fixed windows of 64 characters from the last 10% of the text. LHD watches every layer with weights: fc1 to fc4 in the fully connected network; conv1, conv2, conv3, fc1, and fc2 in the CNN; and 11 layers in the transformer, namely the token and position embeddings (tok, pos), the attention input and output projections of each block (b0.qkv, b0.proj, b1.qkv, b1.proj), the two feed-forward layers of each block (b0.ff1, b0.ff2, b1.ff1, b1.ff2), and the output layer (head). The layers followed by a ReLU, where units can die, are fc1 to fc3 in the fully connected network, conv1 to conv3 and fc1 in the CNN, and b0.ff1 and b1.ff1 in the transformer.

Every run used Adam with a learning rate of 0.001 and a cosine schedule on one CPU thread. The seed fixes both the starting weights and the order of training batches, so a broken run and the healthy run with the same seed differ only by the fault. That healthy run is the broken run's twin, and the twin's final accuracy shows how much the fault cost.

4.2 Planted faults

Each broken run has one fault planted in one layer at the start of training. A frozen layer has its weights and bias put back after every optimizer step, so gradients are computed but never applied, which is the same failure as a layer left out of the optimizer. Dead units are made by setting the incoming weights of 10%, 25%, or 50% of a layer's units to 0 and their bias to -1, so their ReLU output is 0 for every input. The units are drawn at random without replacement by a generator seeded with the run's seed plus 7919, and their number is the fraction times the layer's width, rounded, with at least one unit. The optimizer's running averages for those units are cleared. Frozen layers were planted in every watched layer and dead units in every layer followed by a ReLU. A third set of runs added faults beyond these two, reported in Section 5.9.

4.3 Runs

Calibration runs used seeds 0 to 19 and were used only to set the healthy range. Fresh healthy runs and broken runs used seeds from 1000 up, so no scored run ever touched the range. Every run trained for all of its epochs, and LHD's alarm was found by replaying the saved signals. Replaying gives the same answer as running LHD live and lets every detector be scored on the same complete runs. All 885 runs were in the run plan committed on 31 October 2026 (commit `b6bfb6a`), before any training started, and were trained in three sets. The first held the calibration runs, 90 fresh healthy runs, and the broken runs on five seeds, and the second repeated the broken runs on five more seeds. The third added 60 fresh healthy runs and the faults of Section 5.9, and it was trained last, after the main results were in. An earlier draft of this paper (commit `d3f5275`), written when 110 fresh healthy runs had finished, reported 3 false alarms, 2.7% of 110. The 40 healthy runs that finished after it raised none, so the same 3 alarms are now 2.0% of 150.

4.4 Detectors compared

The accuracy curve raises suspicion when validation accuracy falls below the lowest calibration value for that epoch 3 epochs in a row. It uses the same patience as LHD and no margin, which makes it more sensitive than LHD's own rule. A version that needs only one epoch and a version based on the loss curve were also scored. Two kinds of fixed rule that need no healthy runs test whether LHD does better than checks anyone could write by hand. The zero update check fires when some layer's weights do not move at all in an epoch, like the UnchangedTensor rule of SageMaker Debugger. The dead unit checks fire when some ReLU layer has at least a fixed share of dead units, at 50% (the default of SageMaker Debugger's DeadRelu rule), 70% (DeepDiagnosis), 80% (the default of wandb-callbacks), or a single dead unit (its verbose mode). SageMaker counts a ReLU as inactive when it fires less than 1% of the time, which is close to LHD's definition but not the same. Each fixed rule was scored twice, once firing on the first epoch its condition holds and once, like LHD, only after the condition holds for three epochs in a row. A broken run counts toward a fixed rule's right layer rate only when the layer it names is the planted layer.

4.5 Measures and statistics

A broken run is caught when a detector fires at or after the fault's onset, and a false alarm is any alarm on a fresh healthy run. The gap is the accuracy curve's epoch minus LHD's epoch, with "never" counted as one epoch past the end of the run. Runs that share a seed share their starting weights and batch order, and runs that share a fault and layer share the same break, so single runs are not independent. The main tests therefore average the gap within each seed and within each fault condition and apply a one sided Wilcoxon signed rank test to those averages. The 95% interval for the mean gap comes from a bootstrap with 10,000 draws that resamples whole seeds. Rates are given with 95% Wilson score intervals.

Table 3: LHD next to the accuracy curve on the planted frozen layers and dead units. The median epoch is over the runs each method caught. Bold marks the better of the two where they differ.

	Fully connected		Transformer		CNN	
	LHD	Accuracy	LHD	Accuracy	LHD	Accuracy
Broken runs caught	**84%**	26%	**100%**	48%	**78%**	29%
Median epoch of the alarm	**3**	17	**3**	5	**3**	7
Healthy runs flagged	1 of 50	1 of 50	**0 of 50**	6 of 50	2 of 50	**0 of 50**

5 Results

5.1 Detection and timing

Across the three networks, LHD caught 411 of 470 broken runs (87%, 95% interval 84 to 90%) and the accuracy curve caught 166 of 470 (Table 3). Of the runs LHD caught, the accuracy curve never noticed 61%, and 66% of all broken runs finished with a final accuracy inside the healthy range. The median alarm came at epoch 3, which is the earliest possible alarm for a fault planted at epoch 1 when three epochs out of range are required, and Figure 2 shows the share of broken runs each method had flagged by each epoch.

On the 162 runs that both LHD and the accuracy curve caught, LHD fired 4.6 epochs earlier on average (median 3), the accuracy curve was earlier in one run, and 26 were ties. Averaged within each seed, all 30 seed groups favored LHD, which is the most extreme result a one sided Wilcoxon test can give with 30 groups. Averaged within each fault condition, 41 of 47 conditions favored LHD. Of the other 6, neither detector fired in two conditions, both fired at the same epoch in one condition, and the accuracy curve was earlier in three conditions, all in layers where LHD missed most of the planted dead units. Counting a run the accuracy curve never caught as one epoch past its end, the mean gap over all broken runs was 9.1 epochs (95% interval 8.4 to 9.7 from a bootstrap over seeds), but that figure leans on the runs the accuracy curve missed, so the gap on runs both caught is the fairer measure. The two other curve checks did no better. Requiring only one epoch below the healthy range, the accuracy curve caught 291 of 470 broken runs but flagged 44 of 150 healthy runs, and the loss curve, held to three epochs above its range, caught 143 of 470 and flagged 9 of 150.

On fresh healthy runs, LHD raised 3 false alarms in 150 runs: one run in the fully connected network's fc2 at epoch 16, where only 1.6% of units were dead, fewer than in any calibration run (lowest 3.9%); two runs in the CNN's conv2 at epochs 3 and 4, where 6.2% of units were dead and no calibration run had a single dead unit there. When no calibration run has a dead unit in a layer, the top of its range is the floor of 0.05, so one dead filter out of 16 is enough to cross it. An alarm for too few dead units does not point to any fault, and checking the dead unit fraction against only the top of its range would remove it.

5.2 Comparison with other detectors

The fixed rules were scored the same way as LHD. Each one had to hold for three epochs in a row, and a broken run counts as caught only when the rule named the planted layer (Table 4). Requiring three epochs instead of one changed little, since a layer's dead unit fraction rarely drops back once it crosses a threshold, so false alarms did not change and dead unit catches fell by at most 2. Counting only the planted layer mattered more. Counted at any layer, the 50% rule caught more runs with 10% dead units (26) than with 25% (25), because it fired on units that died on their own in other layers. Counted only at the planted layer, it caught 8 and 10.

For frozen layers, LHD adds nothing over the zero update check, which caught 200 of 200 at the planted layer with no false alarms, and which fires at epoch 1, two epochs before LHD, when a single epoch is

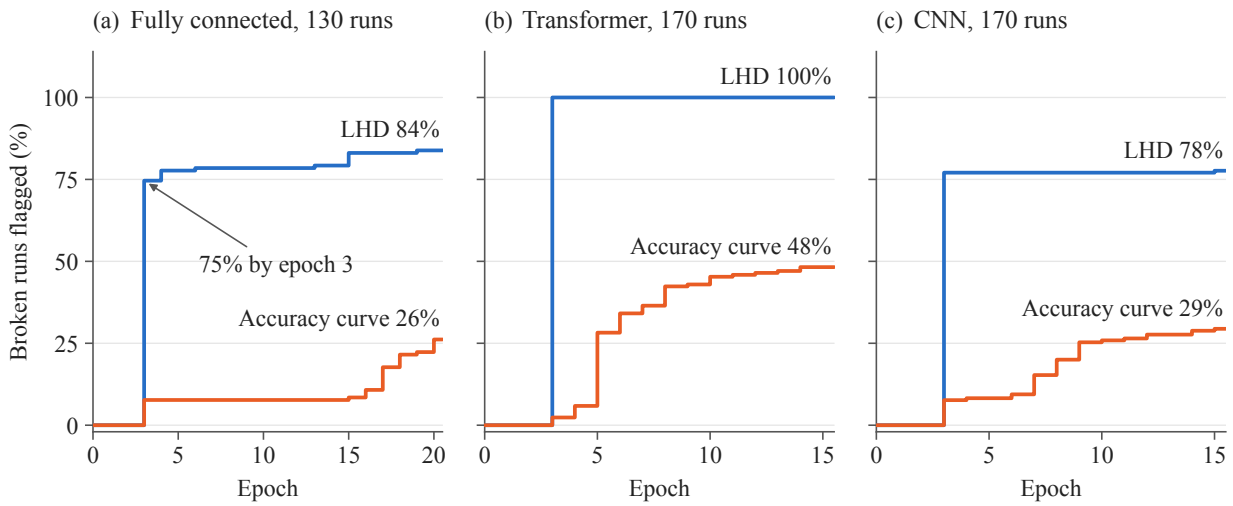


Figure 2: Share of broken runs flagged by each epoch, for LHD and the accuracy curve, in each network.

Table 4: Each detector held to three epochs in a row, with a broken run counted only when the detector named the planted layer. The accuracy and loss curves name no layer, so they are compared in Table 3 instead. Bold marks the best value in a column when no other detector matches it.

	Caught at the planted layer		
Detector	Frozen layers	Dead units	Healthy runs flagged
LHD	97%	**78%**	3 of 150
Zero update check	**100%**	0%	0 of 150
25% dead rule	0%	71%	51 of 150
SageMaker Debugger (50%)	0%	40%	22 of 150
DeepDiagnosis (70%)	0%	4%	0 of 150
wandb-callbacks (80%)	0%	0%	0 of 150
Any dead unit (wandb-callbacks verbose)	2%	74%	100 of 150

enough. A frozen layer's weight change is exactly 0, so a check for an exact 0 needs neither a healthy range nor patience.

Dead units are different, because healthy networks lose units on their own and the amount varies widely. In healthy calibration runs of the CNN's dense layer, between 36% and 70% of units were dead, while no calibration run of the transformer had a single dead unit in its ReLU layers. A single fixed threshold therefore misses faults in one network or flags healthy runs in another (Figure 3). LHD caught 78% of dead unit faults at the planted layer with 3 false alarms in 150 healthy runs. The 25% rule came closest at 71% but flagged 51 healthy runs, a rule that fires on a single dead unit caught 74% while flagging 100, and the default rule of SageMaker Debugger caught 40% while flagging 22.

5.3 How many units must die

With 10%, 25%, and 50% of a layer's units killed, LHD caught 57%, 88%, and 90% of runs, against 7%, 9%, and 29% for the accuracy curve. Its misses came from a few layers: CNN conv3 at 10% (1 of 10 caught), CNN fc1 at 10% (0 of 10 caught), CNN fc1 at 25% (0 of 10 caught), CNN fc1 at 50% (1 of 10 caught), fully connected network fc2 at 10% (0 of 10 caught), fully connected network fc3 at 10% (0 of 10 caught), fully connected network fc3 at 25% (9 of 10 caught). In each of them, healthy runs already lose a

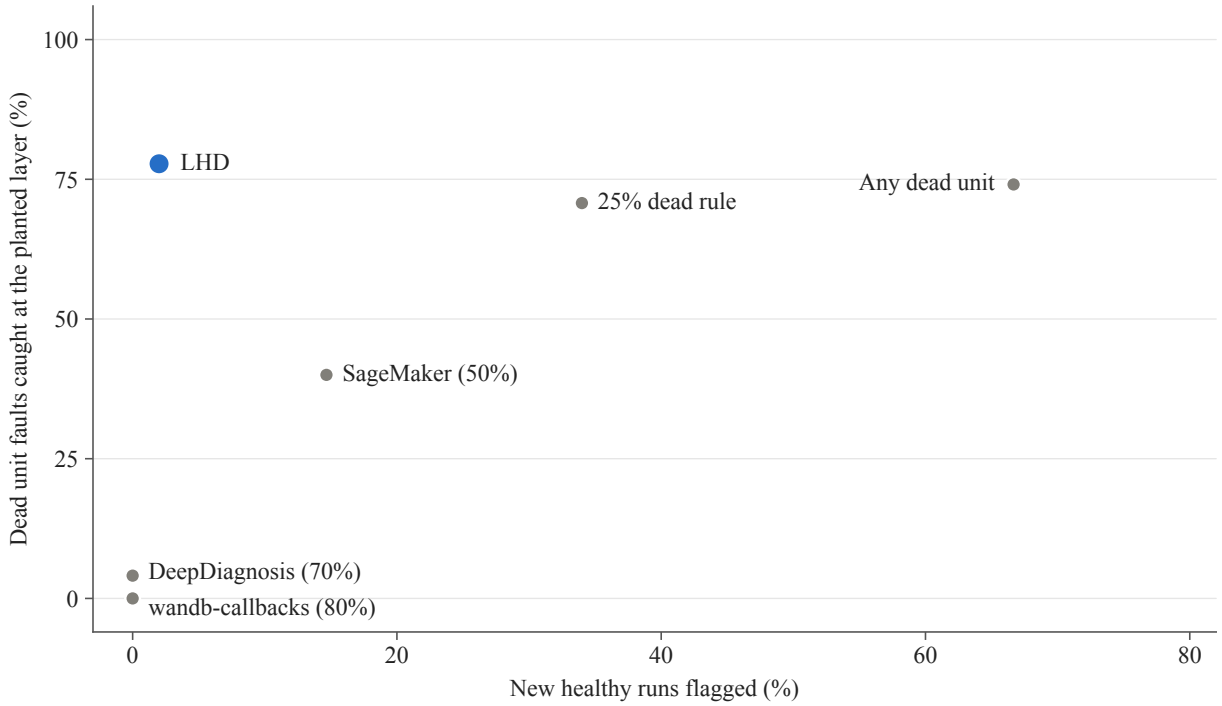


Figure 3: Dead unit faults caught at the planted layer against healthy runs flagged, for LHD and the fixed rules held to three epochs, pooled over the three networks. Detectors toward the upper left catch more faults while flagging fewer healthy runs.

share of units on their own, so the planted fault stayed inside the healthy range. The CNN's dense layer is the clearest case, since its healthy range already covers most of the planted values, and it shows up as the pale row of Figure 4.

5.4 Faults that cost accuracy

Most planted faults did little harm. In 223 of 470 broken runs, the final accuracy was less than half a point below the healthy twin's, and the accuracy curve missing those runs is partly the curve being right. Among the 105 runs that lost 2 points or more, LHD caught 94 and the accuracy curve caught 60. A frozen CNN layer cost the most, lowering final accuracy by 11.1 points on average against the healthy twin, about as much as the frozen convolution in the GTSRB model that motivated this work. Half of a CNN layer's units dead cost 3.1 points, and no fault in the other two networks cost more than 0.8 points on average.

5.5 Diagnosis

When LHD caught a broken run, it named the planted layer 98.1% of the time and the planted cause 96.8% of the time (Figure 4). It named the wrong layer in 8 caught runs (50% dead units in fc1 in the CNN, reported as fc2 (1); a frozen conv2 in the CNN, reported as conv1 (1); a frozen conv3 in the CNN, reported as conv2 (1); a frozen fc1 in the CNN, reported as conv1 (1); a frozen fc1 in the CNN, reported as conv2 (1); a frozen fc1 in the CNN, reported as conv3 (3)). In each of these, the reported layer's own signal was out of range, with too many dead units in 5 and another signal in 3, so LHD found a real effect of the fault in a different layer. In the seven frozen cases, the planted layer's weight change crossed its range in the same epoch, but LHD names only the first layer in data flow order, so reporting every layer out of range would

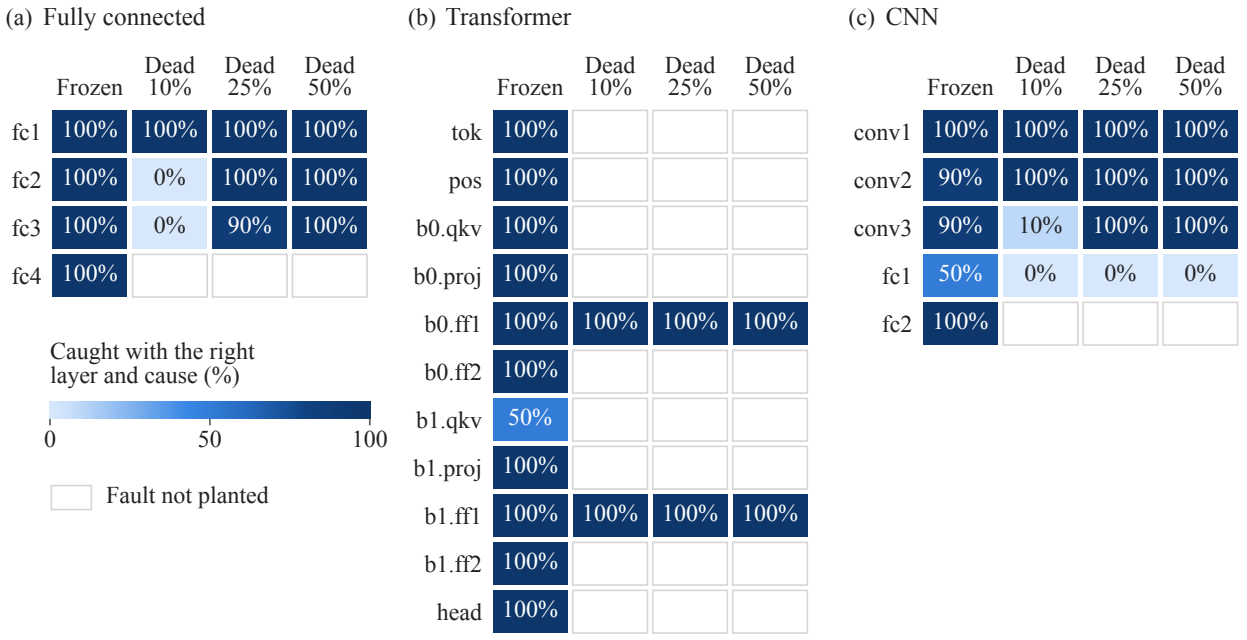


Figure 4: Share of broken runs in which LHD caught the fault and named both the planted layer and the right cause, for every layer and fault. Each cell is 10 broken runs. Blank cells are faults that were not planted, since dead units only go into layers followed by a ReLU.

have included the planted one. It found the right layer but named the wrong cause in 5 runs (a frozen b1.qkv in the transformer, called a vanishing gradient (5)). A frozen attention projection also lowers the gradient that reaches it, so its pattern matches a vanishing gradient. The high rate of naming the right cause is largely built into the design, since the planted faults produce exactly the signal patterns the rules look for.

5.6 What each signal adds

Table 5 shows the same runs replayed with each signal removed, or with one signal used alone. Without the dead unit fraction, LHD caught 43% of broken runs instead of 87%, so that signal carries most of the detection. Without the weight change, it named the right layer and cause in 45% instead of 85%, since the weight change is what separates a frozen layer from other faults. Removing the weight norm or the gradient norm changed little, because none of the planted faults was designed to move them.

5.7 Settings and calibration

These checks were run after the main results, by replaying the same saved runs, and the settings were fixed before them, not picked from them. Thresholds placed right at the edge of the healthy range flagged 71% of healthy runs. Widening them by the margin of 0.5 used here cut that to 2%, while the share of broken runs caught fell only from 98% to 87%. The range also needs enough calibration runs. Built from 2 runs, it flagged up to 22% of healthy runs (in the CNN), and built from all 20, at most 4%.

5.8 Cost

Monitoring added 11.6% for the fully connected network, 5.9% for the transformer, and 2.2% for the CNN to the training time, measured on fresh healthy runs on one CPU thread. Most of that time goes to the

Table 5: What happens when signals are removed or used alone, pooled over the three networks. Changes are in percentage points against all four signals.

Signals used	Broken runs caught		Right layer and cause	
	Rate	Change	Rate	Change
All four	87%		85%	
Without weight change	70%	−18	45%	−40
Without weight norm	87%	same	85%	same
Without grad norm	87%	same	86%	+1
Without dead unit fraction	43%	−45	41%	−44
Only weight change	43%	−45	43%	−42
Only weight norm	10%	−77	0%	−85
Only grad norm	25%	−63	0%	−85
Only dead unit fraction	52%	−36	45%	−40

gradient norm, read after every batch, and to the probe batch, run once per epoch, so the overhead is largest for the network with the cheapest epochs. Stopping each broken run at LHD's alarm would have saved 70% of its training on average, and all runs together took 107 hours of CPU time on one laptop.

5.9 Faults beyond frozen layers and dead units

A third set of runs, planned with the rest and trained after the main results were in (Section 4.3), tested faults the rules were not built around (Table 6). Two of them are the main faults started at epoch 6 instead of epoch 1, a frozen layer and half of a layer's units killed. A gradient cut detaches the input of one layer from the first epoch, so that layer and the layers after it keep learning while no gradient reaches any layer before it. The cut sits before fc2 or fc4 in the fully connected network, before the second block or the output layer in the transformer, and before conv3 or fc1 in the CNN. A learning rate jump multiplies Adam's learning rate by 10 or 100 at the start of epoch 6. The cosine schedule then keeps shrinking the rate from the raised value, so every later epoch trains at 10 or 100 times the rate it would otherwise have, falling along the same curve. In the 15 epoch networks, for example, epoch 6 trains at 0.0075 or 0.075 instead of 0.00075.

LHD caught 100 of 100 runs with a frozen layer planted at epoch 6, naming the layer and cause in 100 of them, while the accuracy curve caught 20. It caught 40 of 45 runs with half a layer's units killed at epoch 6, naming the layer and cause in 40. For the gradient cuts, it caught 30 of 30, named a layer before the cut in 30, and called the cause a vanishing gradient, which is the only test of that rule. LHD also fired on every learning rate jump, but the causes it named are mixed. For a tenfold jump it named an unrecognized pattern in nine runs and dead units in six runs, and for a hundredfold jump it named dead units in five runs, runaway weights in five runs, and a vanishing gradient in five runs. No run produced a signal that was not a finite number, so the divergence rule never fired. The zero update check and the 50% dead rule, the default of SageMaker Debugger, were scored on the same runs the way Table 4 scores them. As in the main study, the zero update check caught every frozen layer planted at epoch 6. Held to three epochs it fired in the same epoch as LHD in all 100 runs, and when one epoch is enough it fires at epoch 6, two epochs before LHD. It also caught 30 of 30 gradient cuts, since the layers before a cut never move. The 50% rule caught 37 of 45 runs with half a layer's units killed at epoch 6, against 40 for LHD, while flagging 22 of 150 healthy runs.

Table 6: Faults beyond the main study, pooled over the three networks. Every detector is held to three epochs in a row. The zero update check and the 50% dead rule count a run only when they name the planted layer, or a layer before a gradient cut, and any alarm counts for a learning rate jump, which has no planted layer. Bold marks the best value in a row when no other detector matches it.

Fault	Runs	LHD	Accuracy curve	Zero update check	50% dead rule
Frozen layer at epoch 6	100	100%	20%	100%	0%
50% dead units at epoch 6	45	**89%**	16%	0%	82%
Gradient cut	30	100%	100%	100%	0%
Learning rate $\times 10$ at epoch 6	15	**100%**	33%	0%	0%
Learning rate $\times 100$ at epoch 6	15	100%	100%	67%	60%
Healthy runs flagged	150	3	7	**0**	22

6 Discussion and Limitations

The results support the hypothesis for planted faults. Per layer signals exposed a broken layer within the first few epochs, usually before the accuracy curve showed anything, and LHD rarely flagged healthy runs. A frozen layer, however, is caught just as well and sooner by a check for weights that never change, so for that fault the only gain is that the same rules also cover it. For dead units, a range measured on healthy runs of the same network did better than every fixed threshold tested, including the defaults of three existing tools.

The faults were planted to show up in the signals LHD measures, so the detection and diagnosis rates describe how well LHD separates these faults from normal variation, not how often it would catch bugs in real projects. Many planted faults also cost less than one point of accuracy, which means stopping those runs saves compute on runs that would have produced a nearly normal model. LHD misses dead units in layers where healthy runs already lose many units, because the healthy range treats that loss as normal. Building the range takes about 10 to 20 healthy runs of the same network, which costs more than LHD can save for a model trained only once, and it assumes those runs are themselves healthy. The floor of 0.05 is also too low for small layers. One dead filter out of 16 is 6.25%, already above the floor, so a later version should scale the floor to the layer's width. Finally, the networks are small and were trained for 15 to 20 epochs on a laptop CPU, the transformer reaches only about 37% next character accuracy, and large networks, long runs, other optimizers, and GPU training were not tested.

7 Conclusion

This work presented LHD, a monitor that compares four signals of every layer with ranges measured on healthy runs of the same network and stops a run when one stays out of range. On 885 runs across three networks, it caught 87% of planted frozen layers and dead units, usually within three epochs, named the right layer in 98% of those runs, and flagged 3 of 150 healthy runs. Its advantage over existing tools is clearest for dead units, where a healthy range for each layer avoided both the misses and the false alarms of fixed thresholds. The next step is to test LHD on real bugs collected from training code.

Use of generative AI

The author came up with the idea, designed the study in an earlier proposal, and wrote the core LHD library (signals, detector, and diagnosis), with Claude, made by Anthropic, providing function outlines and docstrings. Claude wrote the experiment scripts that trained and scored the networks, ran the training runs,

wrote the analysis and figure code, and drafted parts of this paper, including the results and related work. The author directed the study, decided what to test and change, and is responsible for all of its content.

Code and data

The code, every saved run, and the tables behind each number are at https://github.com/sohan-bhat/lhd.

References

Amazon Web Services (n.d.a). List of Debugger built-in rules. Amazon SageMaker AI Developer Guide. https://docs.aws.amazon.com/sagemaker/latest/dg/debugger-built-in-rules.html

Amazon Web Services (n.d.b). Use Debugger built-in actions for rules. Amazon SageMaker AI Developer Guide. https://docs.aws.amazon.com/sagemaker/latest/dg/debugger-built-in-actions.html

Biewald, L. (2020). Experiment tracking with Weights and Biases. Software available from wandb.com. https://www.wandb.com/

Chwilczyński, S., Trębacz, K., Cyganik, K., Małecki, M., and Brzezinski, D. (2024). ART: Actually Robust Training. In *Machine Learning and Knowledge Discovery in Databases. Research Track and Demo Track (ECML PKDD 2024)*, Lecture Notes in Computer Science, vol. 14948, pp. 374–378. Springer, Cham. https://doi.org/10.1007/978-3-031-70371-3_23. Preprint: arXiv:2408.16285.

CS231n (n.d.). Neural networks part 3: Learning and evaluation. Course notes for CS231n, Stanford University. https://cs231n.github.io/neural-networks-3/

Groeger, F. (n.d.). wandb-callbacks: Additional callbacks for Weights & Biases. GitHub repository. https://github.com/FabianGroeger96/wandb-callbacks

Karpathy, A. (2019). A recipe for training neural networks. Blog post, April 25, 2019. https://karpathy.github.io/2019/04/25/recipe/

Kim, J., Humbatova, N., Jahangirova, G., Tonella, P., and Yoo, S. (2023). Repairing DNN architecture: Are we there yet? In *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*, pp. 234–245. IEEE. https://doi.org/10.1109/ICST57152.2023.00030. Preprint: arXiv:2301.11568.

Schoop, E., Huang, F., and Hartmann, B. (2021). UMLAUT: Debugging deep learning programs using program structure and model behavior. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pp. 1–16. ACM. https://doi.org/10.1145/3411764.3445538

Stallkamp, J., Schlipsing, M., Salmen, J., and Igel, C. (2012). Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332. https://doi.org/10.1016/j.neunet.2012.02.016

Wardat, M., Le, W., and Rajan, H. (2021). DeepLocalize: Fault localization for deep neural networks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 251–262. IEEE. https://doi.org/10.1109/ICSE43902.2021.00034

Wardat, M., Cruz, B. D., Le, W., and Rajan, H. (2022). DeepDiagnosis: Automatically diagnosing faults and recommending actionable fixes in deep learning programs. In *Proceedings of the 44th International Conference on Software Engineering (ICSE 2022)*, pp. 561–572. ACM, New York. https://doi.org/10.1145/3510003.3510071. Preprint: arXiv:2112.04036.

Zhang, X., Zhai, J., Ma, S., and Shen, C. (2021). AUTOTRAINER: An automatic DNN training problem detection and repair system. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 359–371. IEEE. https://doi.org/10.1109/ICSE43902.2021.00043