Automatic Stability and Recovery for Neural Network Training

Barak Or [1 2 3]

Abstract

Training modern neural networks is increasingly fragile, with rare but severe destabilizing updates often causing irreversible divergence or silent performance degradation. Existing optimization methods primarily rely on preventive mechanisms embedded within the optimizer, offering limited ability to detect and recover from instability once it occurs.

We introduce a supervisory runtime stability framework that treats optimization as a controlled stochastic process. By isolating an innovation signal derived from secondary measurements, such as validation probes, the framework enables automatic detection and recovery from destabilizing updates without modifying the underlying optimizer. We provide theoretical runtime safety guarantees that formalize bounded degradation and recovery. Our implementation incurs minimal overhead and is compatible with memory-constrained training settings.

1. Introduction

Despite significant advances in optimization algorithms, model architectures, and large-scale infrastructure, practitioners routinely encounter training runs that diverge, collapse, or silently degrade after a small number of destabilizing updates. In such settings, a single pathological update-caused by an outlier minibatch, transient numerical instability, or abrupt distributional shift-can irreversibly corrupt the training trajectory, leading to wasted computation and unreliable experimental outcomes.

Most existing approaches to mitigating training instability focus on *preventive mechanisms* embedded within the optimizer itself. Examples include adaptive learning rates and momentum schemes (Polyak, 1964; Kingma, 2014), gradient clipping (Pascanu et al., 2013), trust-region or curvature-aware methods (Martens et al., 2010), and related variance-reduction techniques. While effective in many scenarios, these methods are inherently local and proactive: they aim to reduce the likelihood of instability, but provide limited protection once a destabilizing update has already occurred. In practice, when training collapses due to an unexpected event, standard pipelines offer no principled mechanism for detection, rollback, or recovery.

In this work, we propose a *runtime stability layer* for neural network training that treats optimization as a controlled stochastic process. Rather than modifying the optimizer update rule, our approach operates above the optimizer, monitoring optimizer-proposed updates using secondary measurement signals that are not directly optimized by the training objective. Examples of such signals include small validation probes, loss trajectory statistics, or measures of gradient agreement. By comparing expected and observed behavior through an innovation signal, the controller assesses whether a proposed update is consistent with stable training dynamics.

When instability is detected, the controller intervenes through a conservative corrective action: rejecting the proposed update and restoring the training state to a previously accepted snapshot. The contributions of this work are as follows:

- We introduce an optimizer-agnostic control-theoretic framework for runtime stability and recovery in neural network training.
- We provide theoretical guarantees in the form of runtime safety invariants, showing that the controller enforces bounded degradation with respect to the chosen measurement signal and enables recovery from destabilizing updates.
- We demonstrate empirically that the proposed method reduces training divergence, improves robustness under noisy and unstable conditions, and recovers from catastrophic updates where standard training pipelines fail.

A reference implementation illustrating the proposed runtime stability controller and its usage is available at `BarakOr1/runtime-stability-controller`.

[1]Google and Reichman Tech School [2]Reichman University [3]MetaOr Artificial Intelligence. Correspondence to: Barak Or <barak@metaor.ai>.

Preprint. January 27, 2026.

2. Related Work

Our work intersects several lines of research in optimization, training stability, and control-inspired learning. We review the most relevant areas and highlight how our approach differs from existing methods.

2.1. Optimization and Adaptive Training Methods

A large body of work addresses robustness in neural network training through adaptive optimization methods, including SGD with momentum and adaptive optimizers such as Adam and its variants (Polyak, 1964; Kingma, 2014; Loshchilov & Hutter, 2017). Complementary techniques such as gradient clipping (Pascanu et al., 2013) and trust-region methods (Martens et al., 2010) are commonly used to mitigate training instability.

More recent methods, such as Sharpness-Aware Minimization (SAM) (Foret et al., 2020) and the Lookahead optimizer (Zhang et al., 2019), seek to improve generalization or smooth training dynamics by modifying the update rule or maintaining auxiliary parameter trajectories. While effective in many settings, these approaches are inherently *preventive*: they alter how updates are computed in order to reduce the likelihood of instability. Once a destabilizing update has already been applied, however, standard optimizers provide no mechanism for detecting the failure or recovering from it.

2.2. Kalman-Inspired and Bayesian Training Approaches

Several prior works draw inspiration from Kalman filtering or Bayesian inference to improve neural network training. Early work explored Kalman-inspired updates for neural network training (Singhal & Wu, 1989; Haykin, 2004). More recent methods incorporate uncertainty estimates or Kalman-inspired updates to denoise gradients, adapt learning rates, or approximate curvature information (Ritter et al., 2018; Osawa et al., 2019). Unlike Bayesian or Kalman-style optimizers, we use innovation signals solely for runtime update acceptance, without maintaining probabilistic state or modifying the optimizer.

2.3. Robustness, Trust Regions, and Training Stability

Another related line of work focuses on robustness through trust-region methods, constrained optimization, or smoothing techniques such as exponential moving averages of parameters. Trust-region and second-order methods explicitly constrain update magnitudes to prevent divergence (Martens et al., 2010), while parameter averaging techniques improve stability by smoothing the optimization trajectory (Izmailov et al., 2018).

While conceptually related, such techniques operate at the level of update computation and are primarily preventive. They do not explicitly monitor training outcomes using secondary signals, nor do they provide mechanisms for rollback or recovery once an update has been applied. Early stopping and checkpointing offer coarse-grained protection but require manual intervention and do not integrate into the training loop as automated control mechanisms.

In contrast, our method introduces a fine-grained, automated stability controller that continuously monitors training behavior at runtime and intervenes when instability is detected. This enables rapid recovery from rare but severe failures while preserving standard training behavior during stable regimes. Simple heuristics such as gradient clipping are inherently local and preventive, and cannot recover a model once a destabilizing update has corrupted the parameters (Pascanu et al., 2013). In contrast, the proposed approach enables explicit rollback based on secondary signals external to the optimizer.

3. Models and Training Setup

We evaluate the proposed runtime stability controller on representative neural network architectures drawn from two widely used model classes: convolutional networks for vision and Transformer-based models for sequence modeling. All models are trained using standard optimization pipelines, without architectural modifications or task-specific tuning beyond conventional choices.

3.1. Vision Model

For image classification, we use a ResNet-18 architecture (He et al., 2016) with a standard classification head. Training is performed on the CIFAR-10 dataset (Krizhevsky et al., 2009) using cross-entropy loss.

Optimization is carried out using AdamW (Loshchilov & Hutter, 2017). Mini-batches of size 128 are used throughout training. Rather than training for a fixed number of epochs, we adopt a fixed-step schedule to enable precise alignment of destabilizing perturbations and controller interventions across runs.

3.2. Sequence Model

For sequence modeling, we employ a character-level Transformer encoder (Vaswani et al., 2017). A linear output head maps hidden representations to vocabulary logits.

The task is next-character prediction on a synthetic text corpus constructed by repeating a fixed phrase. Inputs are sequences of length 64, and training uses cross-entropy loss over the predicted next character. Optimization is performed using AdamW (Loshchilov & Hutter, 2017) with learning

rate 5×10^{-4} and default hyperparameters, with a batch size of 64.

3.3. Training Protocol

All models are trained for a fixed horizon of 250 optimization steps. Baseline and controlled runs share identical initialization and mini-batch order.

To evaluate recovery behavior, we introduce a controlled destabilization window by amplifying gradient magnitudes for several consecutive updates. Outside this window, training proceeds without modification. The runtime stability controller monitors training behavior using a fixed probe set and intervenes only when instability is detected. All experiments are repeated over $N = 20$ independent random seeds, and results are reported as mean and variance across runs.

Specifically, we inject a catastrophic failure at step $t = 120$ by scaling gradients by a factor of $\zeta = 300$ for a window of 10 iterations. This perturbation is designed to reliably induce divergence in the baseline model, enabling controlled evaluation of the controller's ability to trigger rollback and resume nominal optimization.

4. Training as a Controlled Stochastic System

We model neural network training as a controlled stochastic dynamical process. Let $\theta_t \in \mathbb{R}^d$ denote the model parameters at iteration t. A standard optimizer computes gradients of the training loss and proposes an update $\Delta\theta_t$, yielding a candidate next state

$$\theta_t^{\text{prop}} = \theta_t + \Delta\theta_t. \tag{1}$$

We interpret this proposal as a prediction of the next training state.

To assess whether the proposed update is consistent with stable training behavior, we introduce a *measurement signal* that evaluates the effect of the update using information not directly optimized by the training objective. Such signals may include validation probe loss, statistics of the training loss trajectory, or measures of agreement between gradients computed on different minibatches. Importantly, these measurements are not assumed to be unbiased estimates of the training loss, nor do they require strong smoothness or stationarity assumptions.

The controller compares the observed measurement outcome associated with a proposed update against an expected behavior under stable training dynamics. The resulting deviation, referred to as an *innovation signal*, serves as an indicator of whether the proposed update is consistent with recent training behavior. A precise definition of the innovation signal and its estimation is given in Section 5.

In the implementation considered in this work, the innovation signal governs a binary runtime decision: the proposed update is either accepted and applied, or rejected via rollback to a previously accepted training state. The controller evaluates updates before acceptance and intervenes only when instability is detected. These interventions operate at the level of training execution rather than update computation.

This formulation decouples optimization from stability enforcement, allowing recovery from destabilizing updates without altering the optimizer or assuming a probabilistic training model. This abstraction underlies the runtime stability controller described in the next section.

5. Runtime Stability Controller

We now describe the proposed runtime stability controller. The controller operates as an external layer in the training loop, evaluating optimizer-proposed updates and intervening only when instability is detected. Crucially, the controller does not modify the optimizer update rule; instead, it governs the acceptance and execution of updates based on observed training behavior.

5.1. Innovation Signal as a Runtime Consistency Check

The core mechanism underlying the proposed controller is the use of an *innovation signal* as a runtime consistency check for optimizer-proposed updates. Importantly, we do not interpret the innovation probabilistically, nor do we assume a generative or state-space model of training dynamics. Instead, the innovation serves as a pragmatic indicator of whether a proposed update is consistent with recently observed, stable training behavior.

We formalize this notion through a minimal abstraction that is sufficient to distinguish our approach from loss-based heuristics and checkpointing strategies.

Definition 5.1 (Admissible Innovation Signal)**.** An innovation signal is any scalar-valued function

$$\nu_t = s(\theta_t^{\text{prop}}, \mathcal{H}_t)\,, \tag{2}$$

computed at runtime from a proposed parameter update θ_t^{prop} and a finite summary of recent training behavior $\mathcal{H}_t$, where $\mathcal{H}_t$ may include previously accepted measurement values, moving averages, or other bounded statistics derived from past iterations. The innovation signal is required to satisfy:

1. **Externality.** The signal is not directly optimized by the training objective used to compute gradients.
2. **Nominal Stability.** Under stable training dynamics, the signal exhibits bounded variation over time.

3. **Catastrophic Sensitivity.** Destabilizing updates produce deviations in the signal that are significantly larger than those observed during nominal operation.

Instantiation Used in This Work. In our experiments, we instantiate the innovation signal using a fixed validation probe:

$$\nu_t = y(\theta_t^{\text{prop}}) - \hat{y}_t, \tag{3}$$

where $y(\cdot)$ denotes the probe loss and $\hat{y}_t$ is an exponentially weighted moving average of past accepted measurements. This choice satisfies all admissibility criteria: the probe loss is external to the optimizer, remains stable under nominal training, and responds sharply to destabilizing updates.

Why Training Loss Is Insufficient. A natural alternative is to monitor the training loss directly and rollback upon loss spikes. However, training loss is evaluated on the same minibatch used to generate the update and is therefore tightly coupled to minibatch noise. In practice, this coupling leads to frequent false positives and delayed detection of catastrophic failures. By contrast, probe-based innovation leverages information not seen by the optimizer, enabling earlier and more reliable detection with substantially fewer interventions.

5.2. Control Actions

When the innovation signal indicates instability, the controller applies a binary runtime decision rule: the proposed update is either accepted and applied as-is, or rejected via rollback to the most recent accepted training state. This design focuses on minimal, decisive interventions that preserve standard optimization dynamics whenever possible.

While we evaluate only accept and rollback actions, the framework naturally supports richer control strategies, which we leave for future work.

Figure 1 illustrates the interaction between the optimizer, measurement signal, and runtime stability controller. The complete procedure is summarized in Algorithm 1.

5.3. Measurement Signals

The controller relies on a secondary *measurement signal* to assess the effect of a proposed update. This signal is evaluated at runtime and is not required to coincide with the training loss. Suitable measurement signals include, but are not limited to: (i) validation probe loss computed on a small, fixed subset of held-out data, (ii) statistics of the training loss trajectory, such as abrupt increases or deviations from recent trends, and (iii) measures of gradient agreement, such as cosine similarity between gradients computed on different minibatches.

The purpose of the measurement signal is not to estimate generalization performance accurately, but to serve as a consistency check for stable training dynamics, in the spirit of prior work that analyzes and monitors training behavior using signals beyond the training loss (Jiang et al., 2019). In practice, validation probes are particularly effective, as they provide a low-variance signal that is decoupled from minibatch noise. To limit computational overhead, measurements may be evaluated intermittently rather than at every iteration.

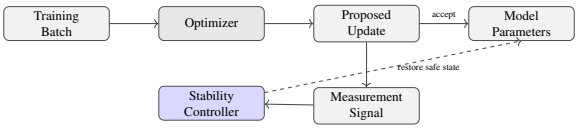

Figure 1. Runtime stability controller as an external supervisory layer. The optimizer proposes parameter updates based on training batches. A secondary measurement signal evaluates the proposed update, and a stability controller decides whether to accept it or trigger rollback by restoring a previously accepted safe state. The controller operates solely on the measurement signal and stored snapshots.

6. Algorithm

We summarize the complete runtime stability procedure in Algorithm 1. The algorithm is optimizer-agnostic in the sense that it does not modify the optimizer's update rule and can be integrated into standard training loops as an external supervisory layer.

Before presenting the algorithm, we define the runtime quantities maintained by the controller. Let $\theta_t \in \mathbb{R}^d$ denote the model parameters at iteration t, and let $\mathcal{O}_t$ denote the optimizer together with its internal state (e.g., momentum buffers or adaptive statistics). The controller maintains a snapshot $(\theta_{\text{safe}}, \mathcal{O}_{\text{safe}})$ corresponding to the most recent accepted training state. We further maintain a scalar reference signal $\hat{y}_t$, representing an exponentially smoothed estimate of the probe measurement, updated with smoothing parameter $\alpha \in (0, 1)$. At each iteration, the optimizer proposes an update $\Delta\theta_t$, yielding a candidate state θ_t^{prop}, which is evaluated using a probe function $y(\cdot)$ to form an innovation signal ν_t.

Algorithm 1 formalizes the proposed runtime stability controller. At each iteration, the optimizer proposes a candidate update, which is evaluated using a secondary measurement signal. If the resulting innovation remains within tolerance, the update is accepted and training proceeds normally. Otherwise, the controller restores the model parameters and optimizer state to the most recent accepted snapshot, enabling immediate recovery from destabilizing updates.

While the present implementation focuses on accept and rollback actions, the framework naturally extends to additional control actions which we leave for future work. These

Algorithm 1 Runtime Stability Controller with State Recovery

Input: Initial θ_0, Optimizer $\mathcal{O}_0$, Probe $y(\cdot)$, Threshold ϵ, Smoothing parameter α
$\hat{y}_0 \leftarrow y(\theta_0)$
$\theta_{\text{safe}}, \mathcal{O}_{\text{safe}} \leftarrow \theta_0, \mathcal{O}_0$ {Initialize snapshots}
for $t = 0, 1, 2, \dots$ **do**
  $\Delta\theta_t \leftarrow \mathcal{O}_t(\theta_t)$
  $\theta_t^{\text{prop}} \leftarrow \theta_t + \Delta\theta_t$
  $\nu_t \leftarrow y(\theta_t^{\text{prop}}) - \hat{y}_t$ {Innovation signal}
  if $\nu_t \leq \epsilon$ **then**
    $\theta_{t+1} \leftarrow \theta_t^{\text{prop}}$ {Accept update}
    $\mathcal{O}_{t+1} \leftarrow \mathcal{O}_t$
    $(\theta_{\text{safe}}, \mathcal{O}_{\text{safe}}) \xleftarrow{\text{async}} (\theta_{t+1}, \mathcal{O}_{t+1})$ {Offload to Host RAM}
    $\hat{y}_{t+1} \leftarrow (1-\alpha)\hat{y}_t + \alpha y(\theta_{t+1})$
  else
    $(\theta_{t+1}, \mathcal{O}_{t+1}) \xleftarrow{\text{async}} (\theta_{\text{safe}}, \mathcal{O}_{\text{safe}})$ {Restore from Host RAM}
    $\hat{y}_{t+1} \leftarrow \hat{y}_t$
  end if
end for

extensions are discussed conceptually but are not required to realize the recovery guarantees demonstrated in our experiments.

7. Theoretical Properties

In this section, we analyze the theoretical properties of the proposed runtime stability controller. Rather than attempting to establish convergence guarantees for nonconvex optimization, which are generally unattainable for modern deep learning systems, we focus on *runtime safety and recovery properties*. These properties formalize the intuition that the controller prevents unbounded degradation of training behavior and enables recovery from destabilizing updates.

7.1. Bounded Degradation Invariant

We begin with a fundamental safety property enforced by the controller.

Assumption 7.1 (Rollback Capability)**.** The controller maintains a snapshot buffer storing the most recent accepted parameter and optimizer state $(\theta_t, \mathcal{O}_t)$ and can restore this state exactly when a rollback action is triggered.

Assumption 7.2 (Acceptance Threshold on Innovation)**.** There exists a tolerance $\epsilon > 0$ such that the controller accepts a proposed update at iteration t only if

$$\nu_t \triangleq y(\theta_t^{\text{prop}}) - \hat{y}_t \leq \epsilon, \tag{4}$$

and otherwise triggers rollback by restoring the most recent safe snapshot.

Theorem 7.3 (Bounded Deviation from the Reference Signal)**.** *Under Assumptions 7.1 and 7.2, the sequence of accepted states produced by the controller satisfies, for all t,*

$$y(\theta_{t+1}) \leq \hat{y}_t + \epsilon. \tag{5}$$

Proof. At iteration t, the optimizer proposes θ_t^{prop} and the controller computes $\nu_t = y(\theta_t^{\text{prop}}) - \hat{y}_t$. If $\nu_t \leq \epsilon$, the update is accepted and $\theta_{t+1} = \theta_t^{\text{prop}}$, hence $y(\theta_{t+1}) = y(\theta_t^{\text{prop}}) \leq \hat{y}_t + \epsilon$. If $\nu_t > \epsilon$, rollback is triggered and the controller restores the safe snapshot, yielding $\theta_{t+1} = \theta_t$ (the most recent accepted state), so the accepted state remains unchanged and the inequality holds vacuously because it concerns accepted transitions only. □

Theorem 7.3 establishes a *runtime safety invariant*: regardless of optimizer behavior, the controller prevents unbounded instantaneous degradation in the measurement signal.

7.2. Recovery from Destabilizing Updates

We now formalize the controller's ability to recover from isolated catastrophic updates.

Proposition 7.4 (One-Step Recovery)**.** *Suppose at iteration t the controller rejects the proposal, i.e., $\nu_t > \epsilon$. Then the next training state equals the most recent accepted snapshot, and in particular*

$$\theta_{t+1} = \theta_t \quad \textit{and} \quad y(\theta_{t+1}) = y(\theta_t). \tag{6}$$

Proof. By Assumption 7.2, $\nu_t > \epsilon$ triggers rollback. By Assumption 7.1, rollback restores the most recent accepted snapshot exactly, which at iteration t is $(\theta_t, \mathcal{O}_t)$. Therefore $\theta_{t+1} = \theta_t$ and hence $y(\theta_{t+1}) = y(\theta_t)$. □

Proposition 7.4 captures the notion of *training recovery*: a single catastrophic update cannot irreversibly corrupt the training trajectory.

7.3. Dominance over Unconditional Acceptance

Finally, we compare the controlled training process against unconditional acceptance of optimizer updates.

Proposition 7.5 (Monotone Safety Envelope)**.** *Assume $\hat{y}_0 = y(\theta_0)$ and $\hat{y}_{t+1} = (1-\alpha)\hat{y}_t + \alpha y(\theta_{t+1})$ is updated only on accepted steps, with $\alpha \in (0, 1)$. Under Assumptions 7.1 and 7.2, for all $t \geq 0$,*

$$y(\theta_{t+1}) \leq \max_{k \leq t} y(\theta_k) + \epsilon, \tag{7}$$

and consequently

$$\max_{k \leq t+1} y(\theta_k) \leq y(\theta_0) + (t+1)\epsilon. \tag{8}$$

Proof. If the proposal at iteration t is rejected, rollback yields $\theta_{t+1} = \theta_t$ by Proposition 7.4, hence the bound holds trivially. If it is accepted, Theorem 7.3 gives $y(\theta_{t+1}) \leq \hat{y}_t + \epsilon$. Since $\hat{y}_t$ is a convex combination of previously accepted measurements, it satisfies $\hat{y}_t \leq \max_{k \leq t} y(\theta_k)$. Therefore $y(\theta_{t+1}) \leq \max_{k \leq t} y(\theta_k) + \epsilon$. The second inequality follows by taking the maximum over $k \leq t+1$ and iterating the one-step envelope bound. □

The results above establish that the proposed controller enforces meaningful runtime guarantees independent of optimizer design or loss geometry. Importantly, these guarantees do not rely on convexity, smoothness, or probabilistic modeling assumptions. Instead, they formalize training reliability as a safety property, analogous to invariants used in control and safety-critical systems.

We emphasize that these guarantees apply to the rollback-based controller implemented and evaluated in this work. Extensions to richer action sets, such as update attenuation or regime switching, are conceptually compatible with the framework but are left for future work.

8. Scale and Complexity Analysis

A primary concern for runtime monitoring is the trade-off between stability and throughput. We analyze the overhead of the stability layer below.

8.1. Computational Overhead

Let C_{fwd} and C_{bwd} denote the cost of a forward and backward pass. Standard training per step is roughly $C_{\text{fwd}} + C_{\text{bwd}}$. Our controller adds a measurement cost C_{probe}. Since the probe set P is fixed and small ($|P| \ll |\text{batch}|$), the overhead ratio γ is:

$$\gamma = \frac{C_{\text{probe}}}{C_{\text{fwd}} + C_{\text{bwd}}} \approx \frac{|P|}{3 \cdot |\text{batch}|}. \tag{9}$$

In our experiments, $|P| = 16$ with a batch size of 128, resulting in $\gamma < 0.045$ ($< 4.5\%$ overhead), confirming that the safety layer is suitable for throughput-critical environments.

8.2. Memory and I/O Efficiency

Maintaining θ_{safe} naively doubles GPU memory consumption. To circumvent this, we utilize **Asynchronous Snapshot Offloading**. The recovery state $(\theta_{\text{safe}}, \mathcal{O}_{\text{safe}})$ is transferred to host CPU memory (pinned RAM) via non-blocking DMA transfers. This avoids additional GPU memory overhead during training.

9. Experiments

We evaluate the proposed runtime stability controller under controlled destabilization scenarios designed to isolate failure detection, selective intervention, and recovery behavior. The experiments are diagnostic rather than benchmark-driven, and are designed to illustrate how the controller behaves under rare but severe training failures. All results are aggregated over $N = 20$ independent random seeds.

9.1. Catastrophic Recovery in Vision Models

Figure 2 shows probe loss trajectories for a ResNet-18 trained on CIFAR-10 under a multi-step catastrophic perturbation induced by gradient amplification. The baseline exhibits a pronounced loss spike followed by slow and highly variable recovery across seeds. In contrast, the controlled runs consistently limit peak degradation and return to a stable regime more rapidly, with substantially reduced variance.

9.2. Innovation-Based Detection and Selective Intervention

The innovation signal ν_t exhibits a sharp and localized deviation during the failure window (Figure 3). Outside this window, the signal remains tightly bounded, indicating stable behavior under nominal training dynamics. This contrast demonstrates that the signal exhibits a sharp transition, crossing the safety threshold ϵ. At step $t = 120$, the signal exceeds the predefined safety threshold ϵ. Because the controller monitors a probe set decoupled from the optimizer's gradients, it provides earlier and more reliable detection of destabilizing behavior than loss-based triggers, without introducing false positives during nominal training.

9.3. Internal Stability via Parameter Norms

Figure 4 examines the evolution of the ℓ_2 norm of model parameters during training. Following destabilization, baseline training transitions into a higher-energy parameter regime and remains there for the remainder of training. In contrast, controlled runs return to parameter norms consistent with nominal training behavior.

This suggests that the controller constrains internal model dynamics rather than merely correcting observable loss spikes, preventing irreversible drift into regions of parameter space often correlated with brittle behavior in practice.

9.4. Generalization to Transformer Models

To assess architectural generality, we repeat the recovery experiment using a character-level Transformer model. As shown in Figure 5, the controller again reduces peak degradation and accelerates recovery following the injected pertur-

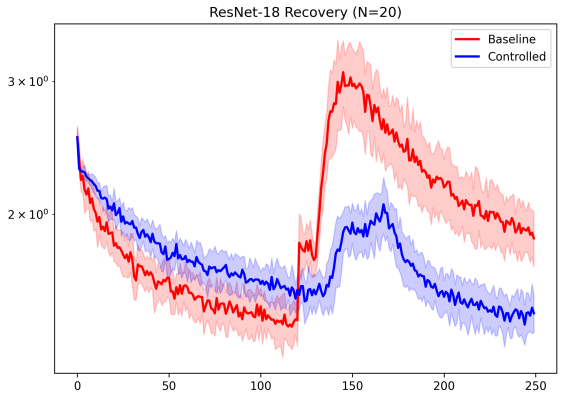

Figure 2. Probe loss recovery for ResNet-18 on CIFAR-10 under a multi-step catastrophic perturbation. Curves show mean ± standard deviation over $N = 20$ seeds. The controller substantially reduces both peak degradation and recovery variance.

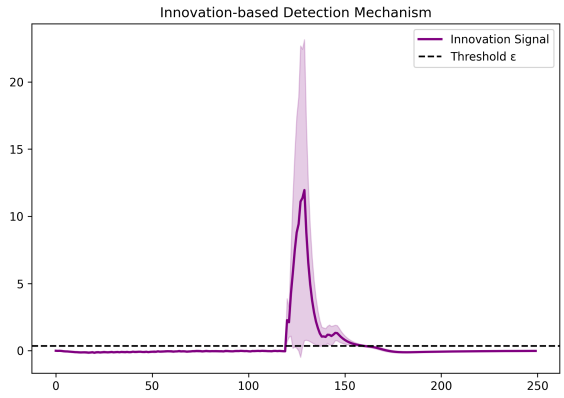

Figure 3. Innovation signal ν_t showing a sharp and localized deviation during the injected destabilization window. Outside this window, the signal remains tightly bounded, indicating stable behavior under nominal training dynamics. The separation between these regimes enables reliable detection of destabilizing updates without sensitivity to minibatch noise.

bation, while the baseline exhibits a prolonged elevated-loss regime. This result indicates that the proposed mechanism generalizes beyond convolutional architectures and is not tied to specific inductive biases.

9.5. Intervention Sparsity

Across all experiments, the controller intervenes sparsely. Rollbacks are typically triggered within a small number of iterations following perturbation onset, and rarely outside the injected failure window. In contrast, a naive loss-based rollback strategy exhibits delayed detection and frequent false positives due to minibatch noise.

10. Discussion and Limitations

The proposed runtime stability controller introduces a reliability-oriented perspective on neural network training that complements existing optimization methods. Rather than modifying the optimizer update rule or relying on preventive heuristics alone, the controller monitors training behavior at runtime and intervenes only when destabilizing events are detected. While the empirical and theoretical results demonstrate clear benefits, several limitations and design trade-offs merit discussion.

Scope of Implemented Control Actions. The controller implemented and evaluated in this work focuses exclusively on *accept* and *rollback* actions. This design choice was intentional: rollback provides the strongest and most direct recovery guarantee, enabling immediate restoration of both model parameters and optimizer state following catastrophic updates. While the general framework admits additional actions such as update attenuation, parameter freezing, or training regime switching, these are not required to establish the safety and recovery guarantees demonstrated here.

Measurement Overhead. The controller relies on a secondary measurement signal, instantiated in our experiments as a small, fixed validation probe. This introduces additional computation during training. In the evaluated settings, the overhead is modest due to the limited probe size and intermittent evaluation, amounting to a small fraction of total training cost. However, in large-scale or highly time-sensitive training pipelines, measurement overhead may become more pronounced. This trade-off between observability and efficiency is fundamental to runtime monitoring and motivates future research on adaptive measurement scheduling and lightweight proxy signals.

Threshold Selection. The controller requires specifying an innovation tolerance parameter ϵ that determines when rollback is triggered. In this work, ϵ is fixed across tasks, architectures, and random seeds, demonstrating robustness to reasonable choices. Nonetheless, inappropriate threshold settings could lead to overly conservative behavior or delayed intervention. Developing adaptive or data-driven threshold selection mechanisms is a natural extension of the proposed framework.

Distributed and Asynchronous Training. Our empirical evaluation considers single-process, synchronous training. In distributed or asynchronous settings, maintaining consistent snapshots of model parameters and optimizer state across workers introduces additional complexity. Extending the controller to such environments may require coordination mechanisms, relaxed consistency guarantees, or hierarchical control strategies. We view this as an important

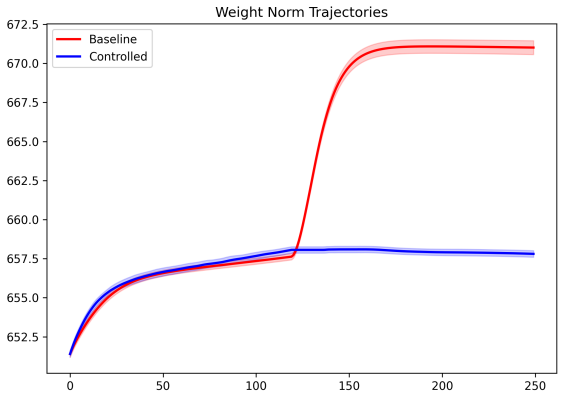

Figure 4. Evolution of parameter ℓ_2 norms during training. The controller prevents persistent drift into high-norm regimes following destabilization, indicating improved internal stability.

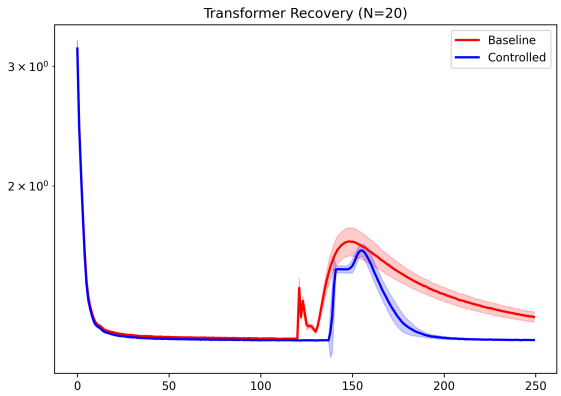

Figure 5. Probe loss recovery for a character-level Transformer model under catastrophic perturbation. Results are averaged over $N = 20$ seeds. The controller consistently improves recovery speed and stability.

extension of the proposed framework, requiring additional coordination mechanisms in distributed settings, but not altering the core runtime stability logic introduced here.

No Convergence Guarantees. The proposed controller does not provide guarantees of convergence or optimality for nonconvex optimization problems. This limitation is inherent and intentional. The controller enforces *runtime safety properties*-bounded degradation and recovery from destabilizing updates-rather than asymptotic convergence guarantees. It is designed to prevent irreversible training failures and complement, rather than replace, existing optimization theory.

Training Reliability and Operational Robustness. The proposed runtime stability controller incurs modest computational overhead while providing a principled mechanism for mitigating rare but severe training failures. In long-running or resource-intensive training regimes, a single destabilizing update can invalidate substantial computational effort. By enabling rollback to a recently accepted safe state, the controller bounds the temporal impact of such failures and limits irreversible degradation of the training trajectory. This reduces reliance on manual intervention, coarse-grained checkpointing, and repeated restarts, and improves the operational robustness of training pipelines.

We view the proposed controller as a foundational reliability component rather than a complete solution. Extensions to richer control actions, adaptive thresholds, and distributed training settings remain important directions for future work.

Implementation Details. Our implementation minimizes overhead by storing the recovery snapshot θ_{safe} in host memory (CPU) via non-blocking transfers. For the ResNet-18 vision tasks, the monitoring and state-management overhead accounted for less than 4.5% of the total time per epoch.

Conservative Bias vs. Optimization Progress. The choice of the threshold ϵ introduces a trade-off between conservatism and progress. In practice, we find that a fixed ϵ provides sufficient separation between benign stochasticity and destabilizing updates across the evaluated architectures.

11. Conclusion

We presented a runtime stability controller for neural network training that emphasizes reliability and recovery rather than convergence guarantees. By framing training as a controlled stochastic process and introducing a mechanism that evaluates optimizer-proposed updates using secondary measurement signals, the proposed approach enables detection and mitigation of destabilizing events that can irreversibly degrade training.

Unlike existing optimization methods that embed preventive mechanisms within the update rule, the proposed controller operates as an external layer that governs the acceptance and execution of updates at runtime. This design preserves compatibility with standard optimizers while enabling exact recovery through rollback when instability is detected. We provided safety-style theoretical guarantees that formalize bounded degradation and one-step recovery properties, and demonstrated empirically that the controller reduces peak degradation, stabilizes internal model dynamics, and improves reliability across both convolutional and Transformer-based architectures. Our work provides a blueprint for **self-healing training pipelines**, shifting the paradigm from manual hyperparameter tuning to automated, control-theoretic runtime stability.

Impact Statement

This work introduces a runtime stability and recovery mechanism aimed at improving the reliability of neural network training. By enabling automatic detection and correction of destabilizing updates, the proposed approach has the potential to reduce wasted computational resources, improve reproducibility, and lower the engineering burden associated with monitoring and restarting failed training runs.

The method operates exclusively during training and does not alter model behavior at inference time. It does not introduce new model capabilities, modify data collection practices, or affect downstream decision-making. As such, it does not raise new concerns related to fairness, privacy, or misuse beyond those already associated with standard machine learning workflows.

By improving the robustness and reliability of training procedures, the proposed controller may support more dependable development of machine learning systems, particularly in settings where training is costly or failure-prone. We do not foresee significant negative societal impacts arising uniquely from this work.

References

Foret, P., Kleiner, A., Mobahi, H., and Neyshabur, B. Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*, 2020.

Haykin, S. *Kalman filtering and neural networks*. John Wiley & Sons, 2004.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., and Wilson, A. G. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

Jiang, Y., Neyshabur, B., Mobahi, H., Krishnan, D., and Bengio, S. Fantastic generalization measures and where to find them. *arXiv preprint arXiv:1912.02178*, 2019.

Kingma, D. P. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Martens, J. et al. Deep learning via hessian-free optimization. In *Icml*, volume 27, pp. 735–742, 2010.

Osawa, K., Swaroop, S., Khan, M. E., Jain, A., Eschenhagen, R., Turner, R. E., and Yokota, R. Practical deep learning with bayesian principles. *Advances in neural information processing systems*, 32, 2019.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pp. 1310–1318. Pmlr, 2013.

Polyak, B. T. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

Ritter, H., Botev, A., and Barber, D. Online structured laplace approximations for overcoming catastrophic forgetting. *Advances in Neural Information Processing Systems*, 31, 2018.

Singhal, S. and Wu, L. Training feed-forward networks with the extended kalman algorithm. In *International Conference on Acoustics, Speech, and Signal Processing,*, pp. 1187–1190. IEEE, 1989.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Zhang, M., Lucas, J., Ba, J., and Hinton, G. E. Lookahead optimizer: k steps forward, 1 step back. *Advances in neural information processing systems*, 32, 2019.