Indian Journal of Computer Science and Technology
https://www.doi.org/10.59256/indjcst.20260501030
Volume 5, Issue1 (January-April 2026), PP: 214-220.
www.indjcst.com
ISSN No: 2583-5300

Adaptive Fault Tolerance in Machine Learning Systems: A Self-Healing Framework

Faiza Fathima[1], Adwaith Raj[2], Jojo George[3], Nandana Babu[4], Ashmila K.P[5], Dr. S. Vadhana Kumari[6]
[1,2,3,4,5,6] *Computer Science and Engineering and Business Systems, Vimal Jyothi Engineering College, Kannur, kerala, India.*

***To Cite this Article**: Faiza Fathima[1], Adwaith Raj[2], Jojo George[3], Nandana Babu[4], Ashmila K.P[5], Dr. S. Vadhana Kumari[6], "Adaptive Fault Tolerance in Machine Learning Systems: A Self-Healing Framework", Indian Journal of Computer Science and Technology, Volume 05, Issue 01 (January-April 2026), PP: 214-220.*

Abstract: *This project presents a plugin-based self-healing machine learning framework aimed at improving the reliability and robustness of deployed ML models in dynamic real-world environments. The framework autonomously detects, classifies, and mitigates runtime errors by continuously monitoring model behavior using reliability signals such as data drift, prediction confidence, entropy, and label consistency. Detected anomalies are semantically classified into error types including data drift, overconfidence, and label noise, enabling a policy-driven healing mechanism to select appropriate corrective actions such as safe retraining or protective blocking. Safety guards and validation checks ensure that learning and adaptation occur only under reliable conditions, preventing harmful self-updates. Through a gated feedback loop and selective learning strategy, the framework maintains long-term model stability while reducing performance degradation. Its modular, plugin-based design allows seamless integration with existing machine learning models without modifying core model logic, thereby minimizing human intervention and providing a practical approach to fault-tolerant machine learning systems suitable for real-world deployment.*

Key Words: *Self-Healing Machine Learning, Fault-Tolerant ML Systems, Runtime Error Detection, Data Drift Detection, Semantic Error Classification, Policy-Driven Healing, Model Monitoring, Autonomous Retraining, Reliability Signals, Plugin-Based Architecture.*

I. INTRODUCTION

Machine learning models are increasingly deployed in real-world applications where they are expected to operate reliably over long periods. However, such environments are often dynamic, with continuously changing data distributions, noisy inputs, and evolving patterns. These changes can cause deployed models to behave unpredictably, leading to degraded performance, incorrect predictions, or silent failures that are difficult to detect. Ensuring sustained reliability of machine learning systems in such conditions has therefore become a critical challenge.

In most traditional machine learning pipelines, model failures are addressed through manual monitoring and periodic retraining. These approaches require significant human intervention and are often reactive rather than preventive. Moreover, retraining a model without understanding the underlying cause of performance degradation can introduce new errors and further reduce reliability. As machine learning systems become more complex and are used in safety-critical domains, the limitations of manual maintenance and blind retraining become increasingly evident.

To overcome these challenges, there is a growing need for intelligent systems that can autonomously monitor their own behavior and respond to failures in a controlled manner. Self-healing mechanisms aim to detect runtime anomalies, analyze their root causes, and apply corrective actions without disrupting system stability. Unlike conventional error-handling methods that focus on individual prediction errors, self-healing approaches emphasize model-level fault detection and recovery to maintain long-term system robustness.

This project proposes a plugin-based self-healing machine learning framework that autonomously detects and classifies runtime errors using multiple reliability signals such as data drift, prediction confidence, and label consistency. Based on semantic error classification, the framework applies policy-driven healing actions such as safe retraining or protective blocking while enforcing safety constraints. By allowing learning only under reliable conditions and integrating seamlessly with existing models, the proposed framework reduces human intervention and provides a practical solution for building fault-tolerant machine learning systems in real-world deployments.

II. RELATED WORK

The authors of On Misbehaviour and Fault Tolerance in Machine Learning Systems explain the problem of silent failures in machine learning models where incorrect outputs occur without clear error signals. The study suggests using machine learning and natural language processing to analyze system logs and runtime patterns for automatic detection of abnormal behaviour, improving system reliability through continuous monitoring.

The authors of Node Co-Activations as a Means of Error Detection Towards Fault-Tolerant Neural Networks investigate

fault detection in neural networks by analyzing neuron co-activation patterns. By comparing real-time activations with baseline patterns, the system can detect abnormal behavior and identify potential prediction errors earlier than traditional output-based confidence methods.

The authors of Realization and Research of Self-Healing Technology of Power Communication Equipment Based on Power Safety and Controllability propose the Self Heal Power Safe Predictor (SHPSP) model for detecting faults in power communication systems. The approach combines sensor monitoring, feature selection, and ensemble learning to improve fault detection accuracy and enable faster recovery responses.

The authors of Self-Healing Codebases – Using NLP and ML for Automatic Code Repair present a framework that uses natural language processing and machine learning to automatically detect and repair faulty code. By learning from historical bug patterns, the system suggests fixes and reduces manual debugging effort in large software systems.

Barrera et al. proposed Fault Detection and Diagnosis for Industrial Processes Based on Clustering and Autoencoders, which combines clustering with autoencoders to detect abnormal patterns in industrial systems without labeled fault data. The method models normal system behaviour and identifies deviations for early fault detection.

Seba et al. proposed Prediction and Classification of IoT Sensor Faults Using Hybrid Deep Learning Models, which uses CNN-LSTM networks to predict sensor values and CNN-MLP models to classify faults such as drift and bias. This proactive approach enables early fault prediction and preventive corrective actions in IoT networks.

Nikam et al. proposed On the Application of YOLO-Based Object Detection Models to Classify and Detect Defects, which applies YOLO object detection algorithms to identify manufacturing defects such as voids and rough surfaces. The method improves quality control by enabling automated visual inspection.

Elhoseny et al. proposed Deep Learning Algorithm for Optimized Sensor Data Fusion in Fault Diagnosis and Tolerance, which integrates sensor data using deep learning models to detect faulty signals while maintaining object detection accuracy. The approach enhances reliability in autonomous systems.

Ortiz-Garces et al. proposed Implementation of Edge AI for Early Fault Detection in IoT Networks, which introduces an edge-based AI system using recurrent neural networks and autoencoders to detect anomalies in IoT networks with low latency and reduced energy consumption.

Saxena et al. proposed A Self-Healing and Fault-Tolerant Cloud-Based Digital Twin Processing Management Model, which presents a cloud framework that continuously monitors system behavior and automatically performs recovery actions such as task reallocation and resource management to ensure uninterrupted digital twin operations.

III. SYSTEM ARCHITECTURE

1) Proposed System

The proposed system is a modular, plugin-based self-healing framework designed to integrate seamlessly with deployed machine learning models without modifying their core architecture. The plugin continuously monitors model behavior using multiple reliability signals such as data drift, prediction confidence, entropy, and label consistency to detect runtime anomalies. Once an anomaly is identified, it is semantically classified into meaningful error types to understand the underlying cause of the failure. Based on this classification, a policy-driven healing mechanism selects appropriate corrective actions, including safe retraining under verified conditions or protective blocking when data integrity is compromised. A gated feedback loop and safety checks ensure that learning and recovery occur only under reliable conditions, thereby maintaining model stability, improving robustness, and enabling autonomous error management with minimal human intervention.

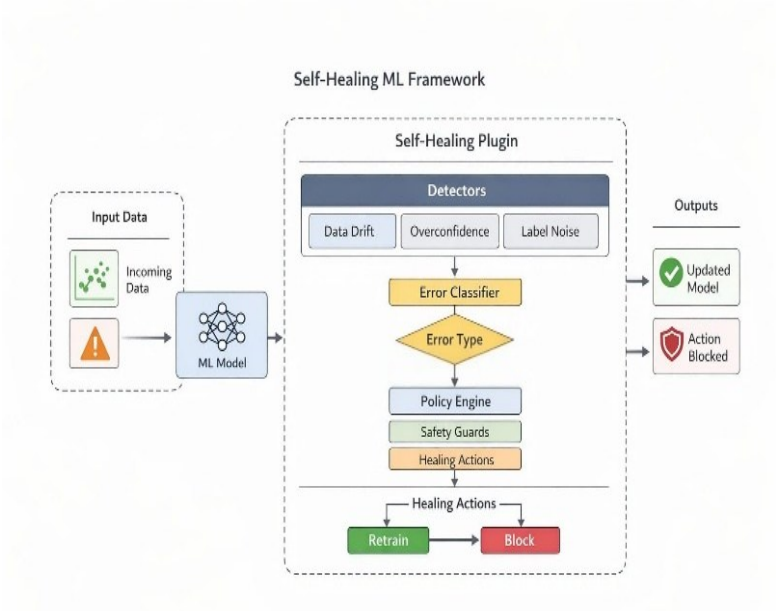


Figure 1: System architecture of High Resolution Hyperspectral Image generation.

Figure shows represents a self-healing machine learning framework that operates alongside a deployed ML model using a plugin-based design. Incoming data is first processed by the deployed ML model as part of normal inference. At the same time, the self-healing plugin continuously observes the model's behavior through a set of detectors, which monitor reliability signals such as data drift, overconfidence, and label noise. These detectors analyze statistical changes in input data, confidence patterns in predictions, and inconsistencies in labels to identify abnormal runtime behavior without interfering with the model's core prediction process.

Once abnormal signals are detected, they are passed to the error classifier, which semantically determines the type of error occurring in the system. The identified error type is then evaluated by a policy engine, which applies predefined, configurable healing rules. Before any corrective action is executed, safety guards validate whether the conditions are reliable enough to proceed. Based on this decision, the system triggers appropriate healing actions, such as safely retraining the model when data drift is confirmed or blocking updates when learning could be harmful. The final outcome is either an updated and stabilized model or a blocked action that prevents further degradation. This architecture ensures autonomous fault detection, safe recovery, and long-term robustness while remaining modular, scalable, and non-intrusive to existing machine learning pipelines.

2) Self-Healing Mechanism

The self-healing mechanism functions as an autonomous control loop that continuously monitors the behavior of a deployed machine learning model and responds to runtime anomalies in a safe and controlled manner. It begins by collecting multiple reliability signals such as data drift, prediction confidence, entropy, and label consistency through dedicated detectors. When abnormal patterns are identified, these signals are semantically classified into specific error types, enabling the system to understand the root cause of the issue rather than reacting blindly. Based on the classified error, a policy-driven decision engine selects an appropriate healing action, such as initiating retraining under verified conditions or blocking learning when data integrity is compromised. Safety guards and gating checks ensure that corrective actions are executed only when conditions are reliable, thereby preventing harmful self-adaptation. Through this closed-loop process, the system maintains long-term stability, improves robustness, and reduces the need for human intervention in deployed machine learning systems.

IV. IMPLEMENTATION

The proposed Adaptive Fault Tolerance framework is implemented as a modular self-healing plugin that operates alongside a deployed machine learning model. The system is developed using Python 3.x and scientific computing libraries. It continuously monitors runtime behavior to detect anomalies such as data drift, overconfidence, and label noise. Once an anomaly is detected, semantic error classification is performed and a policy-driven healing mechanism selects the appropriate corrective action. The design keeps the self-healing logic independent from the base model, enabling integration with machine learning models developed using Scikit-learn.

1) Development Environment

The framework is developed and tested on a standard computing system with an Intel Core i3 or higher processor, at least 8 GB RAM, and approximately 20 GB storage. The software environment includes Python 3.x with libraries such as NumPy, Pandas, Scikit-learn, SciPy, and Matplotlib. Development and testing were carried out using Jupyter Notebook or Python IDEs. The use of open-source tools ensures reproducibility and cost-effective implementation.

2) Modular System Architecture

The system follows a modular architecture consisting of a base machine learning model, a runtime monitoring module, and an error detection module. The base model performs the primary prediction task using supervised learning algorithms such as Logistic Regression or Random Forest. The monitoring module evaluates prediction probabilities, confidence scores, entropy, and statistical properties of incoming data. Predictions with low confidence values are flagged for further analysis.

3) Multi-Signal Error Detection

The framework detects anomalies using multiple reliability signals. Data drift is identified by comparing statistical properties of training data and incoming data distributions, including mean and standard deviation changes. Overconfidence is detected when prediction confidence remains high while model accuracy decreases. Label noise is identified during retraining by analyzing inconsistent labels and abnormal validation variations.

4) Policy-Driven Healing Mechanism

Corrective actions are selected through configurable safety policies. A safe retraining policy is triggered when data drift is detected and reliable data samples are available. If label noise or unreliable data is identified, an unsafe learning block policy prevents retraining to avoid model degradation. For minor anomalies, the monitor-and-log policy allows the system to continue monitoring without modifying the model.

5) Gated Feedback Control

Before retraining, the system performs safety checks including validation of input data reliability, minimum dataset size verification, label consistency analysis, and temporary model evaluation. Only when all conditions are satisfied is the updated model deployed, ensuring stable and controlled adaptation.

6) Logging and Execution Workflow

A centralized logging mechanism records anomaly detection events, triggered policies, and model performance before

and after healing actions. During runtime, the trained model receives input data, reliability signals are monitored, anomalies are detected, and the appropriate healing action is applied while maintaining a complete audit trail.

7) Performance Considerations

The monitoring mechanism introduces minimal computational overhead and operates efficiently on CPU-based systems. Since the self-healing module functions as an external plugin, the internal structure of the machine learning model remains unchanged, preserving system modularity and scalability.

V. EVALUATION AND RESULTS

1) Project Structure

The proposed self-healing machine learning framework is organized using a modular project structure to ensure scalability and maintainability. The main module contains multiple subdirectories responsible for different stages of the self-healing pipeline. The core module manages the plugin controller and runtime context, while the policy configuration file defines rules for handling detected anomalies. The detectors module identifies issues such as data drift, overconfidence, and label noise. Based on these signals, the policy module determines the appropriate corrective action. The actions module executes recovery operations such as model retraining or blocking unsafe predictions. Additionally, the safety module validates decisions before execution to prevent harmful updates, while the adapters module connects the framework with machine learning libraries such as Scikit-learn. This modular organization implements a clear processing pipeline consisting of Detect → Decide → Validate → Act, ensuring flexibility and reliability.

````
## Project Structure
```text
self_healing_plugin/
├── core/               # Main orchestration logic (Plugin, Context)
├── policy_config.json  # Configuration file for rules
├── detectors/          # Logic to detect issues (Drift, Noise)
├── policies/           # Policy engine to decide actions
├── actions/            # Actions to execute (Retrain, Block)
├── safety/             # Guards to validate decisions
├── adapters/           # Adapters for third-party libraries (Sklearn, Torch)
└── examples/           # Demo scripts
```
````

Figure 2: Modular project structure of the self-healing machine learning framework.

2) Plugin Creation

The self-healing plugin is initialized by configuring all core components required for the monitoring and healing process. These components include detectors for identifying anomalies, an error classifier for categorizing detected issues, a policy engine for selecting corrective actions, safety guards for validating decisions, and a logging system for recording system behavior. During execution, the monitoring method evaluates runtime signals and updates the system context. The detected signals are classified into high-level error categories, after which the policy engine selects an appropriate corrective action. Safety checks are then applied to verify whether the action is permissible. If validation fails, the process is blocked and logged; otherwise, the corrective action is executed and the updated system state is returned.

```python
def monitor(self, context):
    for detector in self.detectors:
        # Detectors analyze the context and return a dictionary of signals (e.g., {"drift": True}
        # We update the context's signals with these findings.
        context.signals.update(detector.detect(context))
    # 2. Classification Phase: Determine the high-level error type
    # The classifier looks at the accumulated signals and decides the primary error (e.g., DATA_D
    context.error_type = self.error_classifier.classify(context.signals)
    # 3. Decision Phase: Ask the policy engine what to do
    # The policy engine uses the error type and signals (for confidence checks) to return an Acti
    # Action is an instance of a class like RetrainAction or BlockAction.
    action = self.policy_engine.decide(context.error_type, context.signals)

    # 4. Safety Phase: Verify the context with all safety guards
    for guard in self.safety_guards:
        # If any guard denies the action/context, we block immediately.
        if not guard.allow(context):
            # Mark result as BLOCKED
            context.result = "BLOCKED"
            # Log the blocked event
            self.logger(context)
            # Return context immediately, aborting the action
            return context
    # 5. Execution Phase: Execute the decided action
    # If we passed all guards, run the action (e.g., retrain the model).
    context.result = action.execute(context)
    # Log the final result of the cycle
    self.logger(context)
    # Return the updated context
    return context
```

Figure 3: Initialization and execution flow of the self-healing plugin(1)

```python
def __init__(
    self,
    detectors,        # List of detector instances to monitor data signals
    error_classifier, # Component to classify detected signals into an ErrorType
    policy_engine,    # Component to decide the action based on ErrorType and signals
    safety_guards,    # List of guards to enforce safety constraints before execution
    logger            # Callable to log the context and results
):
    # Store the list of detectors (e.g., DriftDetector, OverconfidenceDetector)
    self.detectors = detectors

    # Store the error classifier strategy (e.g., RuleBasedErrorClassifier)
    self.error_classifier = error_classifier

    # Store the policy engine (e.g., ConfigurableHealingPolicy) which decides actions
    self.policy_engine = policy_engine

    # Store safety guards (e.g., DataValidityGuard) to prevent unsafe actions
    self.safety_guards = safety_guards

    # Store the logger for observability
    self.logger = logger
```

Figure 4: Initialization and execution flow of the self-healing plugin(2)

3) Error Classification

Error classification is performed using a rule-based classifier that maps detected signals to predefined error categories. The classifier evaluates monitoring signals such as data drift, overconfidence, and label noise and assigns the corresponding error type. If none of these signals are detected, the classifier returns a no error state. This rule-based mechanism provides a simple and transparent approach for converting low-level monitoring signals into meaningful system-level error categories that can be processed by the policy engine.

```python
from self_healing_plugin.errors.error_types import ErrorType

class RuleBasedErrorClassifier:
    def classify(self, signals):
        if signals.get("drift"):
            return ErrorType.DATA_DRIFT

        if signals.get("overconfident"):
            return ErrorType.OVERCONFIDENCE

        if signals.get("label_noise"):
            return ErrorType.LABEL_NOISE

        return ErrorType.NO_ERROR
```

Figure 5: Rule-based error classification for detected monitoring signals.

4) Healing Policy

```python
def __init__(self, adapter):
    # Store the adapter to pass to actions that need it (e.g., RetrainAction).
    self.adapter = adapter

def decide(self, error_type, signals=None):
    """
    Decide the action to take based on the error type.

    Args:
        error_type (str): The detected error type (e.g., DATA_DRIFT).
        signals (dict, optional): Additional context (unused in this simple policy).

    Returns:
        HealingAction: The action to execute.
    """

    # If Data Drift is detected, we attempt to retrain the model.
    if error_type == ErrorType.DATA_DRIFT:
        return RetrainAction(self.adapter)

    # For Overconfidence or Label Noise, we block execution to prevent unsafe predictions.
    if error_type in (
        ErrorType.OVERCONFIDENCE,
        ErrorType.LABEL_NOISE,
    ):
        return BlockAction()

    # For any other unknown error types, block by default for safety.
    return BlockAction()
```

Figure 6: Policy engine for selecting corrective actions based on detected errors

A policy engine determines the corrective action based on the detected error type. When data drift is identified, the system triggers a retraining action to adapt the model to the updated data distribution. In contrast, when overconfidence or label noise is detected, the system activates a blocking mechanism to prevent unreliable predictions. For unknown error conditions, the system defaults to a conservative blocking action to ensure safety. This rule-driven policy mechanism ensures that corrective actions are selected systematically while protecting the reliability of the machine learning model.

5) Policy Configuration

The healing strategy is defined through a configurable JSON policy file that maps detected error types to specific actions.

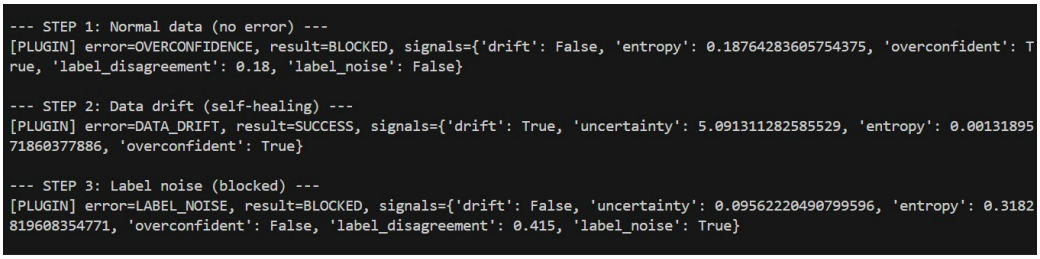
For instance, when data drift occurs, the policy instructs the system to retrain the model while enforcing a minimum confidence threshold. For overconfidence and label noise, the policy blocks predictions to prevent unsafe outputs. If no anomaly is detected, the system continues normal operation without performing corrective actions. This configuration-based approach allows flexible customization of the healing strategy without modifying the core implementation

```
{
    "DATA_DRIFT": {
        "action": "RETRAIN",
        "min_confidence": 0.6
    },
    "OVERCONFIDENCE": {
        "action": "BLOCK"
    },
    "LABEL_NOISE": {
        "action": "BLOCK"
    },
    "NO_ERROR": {
        "action": "NO_ACTION"
    }
}
```

Figure 7: JSON-based policy configuration for adaptive healing actions.

6) System Integration

The self-healing framework is integrated with a machine learning program through a series of module imports. The main plugin orchestrator manages runtime operations, while specialized detectors monitor model behavior. The rule-based error classifier interprets monitoring signals, and the healing policy determines corrective actions. Safety guards verify whether these actions are safe to execute, and an adapter module connects the framework to a Scikit-learn model. This integration enables the plugin to function as an external monitoring and recovery layer for machine learning systems.

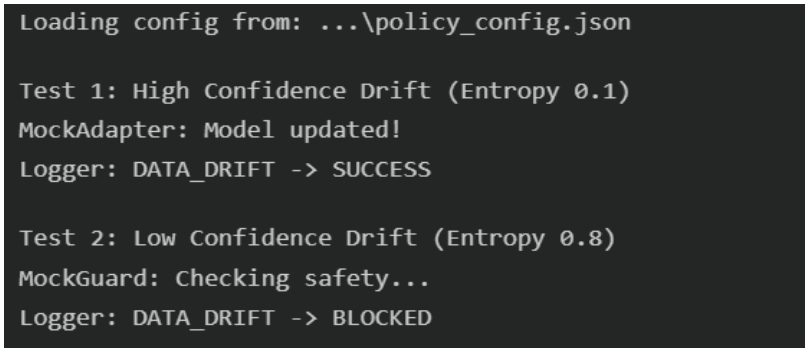


```
--- STEP 1: Normal data (no error) ---
[PLUGIN] error=OVERCONFIDENCE, result=BLOCKED, signals={'drift': False, 'entropy': 0.18764283605754375, 'overconfident': T
rue, 'label_disagreement': 0.18, 'label_noise': False}

--- STEP 2: Data drift (self-healing) ---
[PLUGIN] error=DATA_DRIFT, result=SUCCESS, signals={'drift': True, 'uncertainty': 5.091311282585529, 'entropy': 0.00131895
71860377886, 'overconfident': True}

--- STEP 3: Label noise (blocked) ---
[PLUGIN] error=LABEL_NOISE, result=BLOCKED, signals={'drift': False, 'uncertainty': 0.09562220490799596, 'entropy': 0.3182
819608354771, 'overconfident': False, 'label_disagreement': 0.415, 'label_noise': True}
```

Figure 8: Runtime behavior of the self-healing plugin under different fault conditions.

7) Experimental Output

The runtime output demonstrates the behavior of the proposed system under different operational scenarios. When normal data is processed, the system may detect overconfidence and block predictions to prevent unreliable outputs. When data drift is detected, the error is classified accordingly and the system performs a healing action such as retraining. In the presence of label noise, predictions are blocked to avoid incorrect learning. These results illustrate how the plugin dynamically detects anomalies and selects appropriate responses based on predefined policies.



```
Loading config from: ...\policy_config.json

Test 1: High Confidence Drift (Entropy 0.1)
MockAdapter: Model updated!
Logger: DATA_DRIFT -> SUCCESS

Test 2: Low Confidence Drift (Entropy 0.8)
MockGuard: Checking safety...
Logger: DATA_DRIFT -> BLOCKED
```

Figure 9: Policy-driven execution results demonstrating safety validation and retraining control.

8) Policy-Based Execution Results

Additional experiments were conducted using a policy configuration file to evaluate the interaction between policy rules and safety mechanisms. When data drift is detected with high prediction confidence, the system allows retraining and records a successful healing action. However, when drift occurs with low confidence levels, the safety guard blocks the retraining process to prevent unstable model updates. These results demonstrate how policy configuration and safety validation jointly ensure controlled and reliable adaptation.

VI.CONCLUSION AND FUTURE WORK

This work presents a plugin-based self-healing machine learning framework designed to improve the reliability and stability of deployed ML systems. The framework continuously monitors model behavior, detects runtime anomalies such as data drift, overconfidence, and label noise, and classifies them into meaningful error categories. Based on these classifications, a policy-driven healing mechanism selects appropriate corrective actions such as retraining or blocking unsafe predictions. The integration of safety guards and gated feedback control ensures that corrective actions are executed only under validated conditions, preventing potential model degradation. The modular architecture allows seamless integration with existing machine learning systems and reduces the need for manual intervention. Overall, the proposed approach demonstrates how automated monitoring, error analysis, and controlled recovery mechanisms can significantly enhance the robustness and fault tolerance of real-world machine learning applications.

Future research can focus on extending the framework to support more advanced anomaly detection techniques using deep learning and statistical methods for improved accuracy. The system can also be enhanced to support additional machine learning frameworks such as TensorFlow and PyTorch, enabling broader applicability. Another promising direction is the integration of reinforcement learning to allow the system to learn optimal healing strategies over time. Additionally, implementing real-time deployment in edge or cloud environments and evaluating the framework on large-scale industrial datasets would further validate its effectiveness in practical scenarios.

REFERENCES

1. L. Myllyaho, M. Raatikainen, T. M¨annist¨o, J. K. Nurminen, and T. Mikkonen,“On misbehaviour and fault tolerance in machine learning systems,” Journal of Systems and Software, vol. 183, p. 111096, 2022. [Online]. Available:https://www.sciencedirect.com/science/article/pii/S016412122100193X
2. L. Myllyaho, J. K. Nurminen, and T. Mikkonen, “Node co-activations as a means of error detection—towards fault-tolerant neural networks,” Array, vol. 15, p. 100201, 2022. [Online].Available:https://www.sciencedirect.com/science/article/pii/S2590005622000509
3. Liu, D., Zhang, S., Wang, S. *et al.* Realization and research of self-healing technology of power communication equipment based on power safety and controllability. *Energy Inform* 8, 1 (2025). https://doi.org/10.1186/s42162-024-00460-x
4. E. Oluwagbade, “Self-healing codebases: Using nlp and ml for automatic coderepair,” 03 2023. https://www.researchgate.net/publication/390929429
5. Barrera, J.M., Reina, A., Mate, A. *et al.* Fault detection and diagnosis for industrial processes based on clustering and autoencoders: a case of gas turbines. *Int. J. Mach. Learn. & Cyber.* 13, 3113–3129 (2022). https://doi.org/10.1007/s13042-022-01583-x
6. Seba, A.M., Gemeda, K.A. & Ramulu, P.J. Prediction and classification of IoT sensor faults using hybrid deep learning model. *Discov Appl Sci* 6, 9 (2024). https://doi.org/10.1007/s42452-024-05633-7
7. Nikam, D., Chukwuemeke, A., Nigam, A. *et al.* On the application of YOLO-based object detection models to classify and detect defects in laser-directed energy deposition process. *Prog Addit Manuf* 10, 7609–7624 (2025). https://doi.org/10.1007/s40964-025-01056-x
8. Elhoseny, M., Rao, D.D., Veerasamy, B.D. *et al.* Deep Learning Algorithm for Optimized Sensor Data Fusion in Fault Diagnosis and Tolerance. *Int J Comput Intell Syst* 17, 299 (2024). https://doi.org/10.1007/s44196-024-00692-5
9. Ortiz-Garces, I., Villegas-Ch, W. & Luján-Mora, S. Implementation of edge AI for early fault detection in IoT networks: evaluation of performance and scalability in complex applications. *Discov Internet Things* 5, 108 (2025). https://doi.org/10.1007/s43926-025-00196-4
10. Arie Gurfinkel, Marijn Heule Tools and Algorithms for the Construction and Analysis of Systemshttps://doi.org/10.1007/978-3-031-90643-5
11. D. Saxena and A. K. Singh, “A self-healing and fault-tolerant cloud-based digital twin processing management model,” IEEE Transactions on Industrial Informatics, early access, 2025.https://doi.org/10.1109/TII.2025.3540498