INTERNATIONAL JOURNAL OF ADVANCED ARTIFICIAL INTELLIGENCE RESEARCH (IJAAIR)

eISSN: 3087-4270

Volume. 03, Issue. 01, pp. 05-18, January 2026

FAILURE-AWARE ARTIFICIAL INTELLIGENCE: DESIGNING SYSTEMS THAT DETECT, CATEGORIZE, AND RECOVER FROM OPERATIONAL FAILURES

Ashis Ghosh
Independent Researcher CA, USA

Article Received: 05/01/2026, Article Revised: 13/01/2026, Article Accepted: 14/01/2026, Article Published: 15/01/2026
DOI: https://doi.org/10.55640/ijaair-v03i01-02

ABSTRACT

As artificial intelligence systems increasingly transition from controlled laboratory environments to real-world deployment, their ability to handle unexpected failures becomes a critical determinant of practical utility and safety. This paper introduces a comprehensive framework for failure-aware artificial intelligence, encompassing systematic mechanisms for detecting, categorizing, and responding to failures in deployed AI systems. We propose a three-tier failure taxonomy that distinguishes between input-level anomalies, processing-level errors, and output-level inconsistencies, each requiring distinct detection and recovery strategies. The proposed architecture integrates continuous self-monitoring components, confidence estimation modules, and adaptive recovery mechanisms that enable graceful degradation rather than catastrophic failure. Building upon prior work in modular robotic system architectures and patented approaches to dexterous task execution, we present design principles for building failure-resilient AI systems, including redundancy patterns, fallback hierarchies, and human-in-the-loop escalation protocols. Evaluation through simulated failure injection across multiple AI task domains demonstrates that failure-aware systems maintain operational continuity in 87% of induced failure scenarios, compared to 23% for conventional architectures. The framework provides practitioners with actionable guidelines for enhancing the robustness and reliability of deployed artificial intelligence systems across diverse application contexts.

KEYWORDS

Failure detection, fault tolerance, artificial intelligence systems, system reliability, graceful degradation, self-monitoring AI.

1. INTRODUCTION

The proliferation of artificial intelligence systems across critical domains—from autonomous vehicles and medical diagnosis to financial trading and industrial automation—has fundamentally transformed expectations regarding system reliability and failure handling. Unlike traditional software systems where failure modes are typically well-defined and deterministic, AI systems exhibit complex, often unpredictable failure patterns that emerge from the interaction between learned models, dynamic environments, and distributional shifts in input data (Amodei et al., 2016). This complexity necessitates a paradigm shift from failure prevention, which is inherently incomplete, to failure awareness, which acknowledges the inevitability of failures and builds systematic mechanisms for detection and recovery.

Contemporary AI systems are predominantly designed under the assumption of operational success, with failure handling relegated to exception catching and error logging. This design philosophy proves inadequate when systems encounter scenarios outside their training distribution, experience hardware degradation, or face adversarial inputs. The consequences range from silent degradation in prediction quality to complete system unavailability, both of which can have severe implications in safety-critical applications (Sculley et al.,

2015).

The concept of failure-aware artificial intelligence represents a fundamental reorientation in system design philosophy. Rather than treating failures as exceptional events to be minimized through extensive testing and validation, failure-aware systems internalize the expectation of failure as a core architectural assumption. This perspective draws from established principles in fault-tolerant computing and resilient systems engineering while adapting them to the unique characteristics of machine learning-based systems. Recent work in modular software architectures for mobile manipulation systems has demonstrated that explicit separation of concerns and event-driven coordination significantly improve fault containment and system observability (Ghosh, 2026a). Similarly, patented approaches to robotic task execution have established patterns for dynamic grasp adjustment and closed-loop error correction that inform failure recovery strategies in broader AI contexts (Augenbraun et al., 2022).

This paper makes three primary contributions to the field of robust AI system design. First, we introduce a comprehensive taxonomy of failure modes specific to deployed AI systems, organizing failures across input, processing, and output dimensions. Second, we propose an architectural framework for failure-aware AI that integrates detection, categorization, and recovery mechanisms into a coherent system design. Third, we present practical design principles and patterns that practitioners can apply to enhance the failure resilience of their AI deployments.

The remainder of this paper is organized as follows. Section 2 reviews related work in fault tolerance, AI safety, and system reliability. Section 3 presents our methodology for developing the failure taxonomy and architectural framework. Section 4 details the proposed failure taxonomy and system architecture. Section 5 discusses the implications of failure-aware design for AI system development. Section 6 concludes with recommendations for future research directions.

2. Related Work

2.1 Fault Tolerance in Distributed Systems

The foundations of failure-aware computing trace back to seminal work in distributed systems reliability. Lamport's work on Byzantine fault tolerance established fundamental principles for system operation despite component failures (Lamport et al., 1982). These principles—including redundancy, consensus mechanisms, and graceful degradation—provide conceptual foundations applicable to AI systems, though requiring substantial adaptation for the statistical nature of machine learning components.

2.2 Uncertainty Quantification in Machine Learning

Recent advances in uncertainty quantification have provided tools essential for failure detection in AI systems. Bayesian neural networks and ensemble methods enable estimation of predictive uncertainty, distinguishing between aleatoric uncertainty inherent to data and epistemic uncertainty arising from model limitations (Gal & Ghahramani, 2016). Out-of-distribution detection methods identify inputs that deviate significantly from training data, serving as early warning indicators of potential failures (Hendrycks & Gimpel, 2017).

2.3 AI Safety and Robustness

The AI safety research community has extensively investigated failure modes in learning systems, particularly concerning distributional shift, reward hacking, and specification gaming (Amodei et al., 2016). Robustness research has produced techniques for hardening models against adversarial perturbations and natural distribution shifts (Goodfellow et al., 2015). However, this work primarily focuses on preventing or mitigating specific failure types rather than providing comprehensive frameworks for failure awareness and recovery.

2.4 Machine Learning Operations

The emerging field of MLOps addresses operational concerns in deployed ML systems, including monitoring, versioning, and continuous validation (Sculley et al., 2015). While MLOps practices contribute to failure detection through logging and alerting, they typically operate at the operational layer rather than being integrated into the AI system architecture itself.

2.5 Modular Architectures for Robotic Systems

Recent advances in robotic software architecture have demonstrated the value of modular, layered designs for managing system complexity and improving fault isolation. Ghosh (2026a) presents a comprehensive modular architecture for mobile manipulation systems that decomposes functionality into perception, task reasoning, motion generation, and execution layers. This architectural pattern—emphasizing separation of concerns, explicit state modeling, and event-driven coordination—provides a template for failure-aware design applicable beyond robotics to general AI systems. The architecture's explicit treatment of real-time constraints and safety mechanisms directly informs the failure handling strategies proposed in this work.

2.6 Closed-Loop Control and Dynamic Adjustment in Robotic Systems

Patented work on dexterous robotic task execution has

established patterns for real-time failure detection and recovery in physical systems. Augenbraun et al. (2022) describe closed-loop grasping techniques where grasp parameters are recalculated dynamically during arm motion to compensate for base wobbling, object movement, or measurement inaccuracies. The patent details visual servoing approaches implementing control loops based on the relationship between gripper and target object, enabling correction for changes in position and system inaccuracies. These techniques—including force feedback monitoring, dynamic path adjustment, and fallback to alternative strategies when primary approaches fail—provide concrete implementation patterns that generalize to failure handling in software-based AI systems.

3. Methodology

3.1 Taxonomy Development

The failure taxonomy presented in this work was developed through a systematic analysis of failure incidents reported in AI system deployments across multiple domains. Industry research indicates that approximately 80% of AI projects fail—twice the rate of non-AI information technology projects (RAND Corporation, 2024). Gartner reports that only 48% of AI projects reach production, with an average of 8 months required to move from prototype to deployment (Gartner, 2024). We analyzed documented failures from autonomous driving systems, recommendation engines, natural language processing applications, and computer vision deployments. Thematic analysis of failure incidents yielded recurring patterns that informed the three-tier taxonomy structure. Figure 1 illustrates the methodology workflow for taxonomy development.

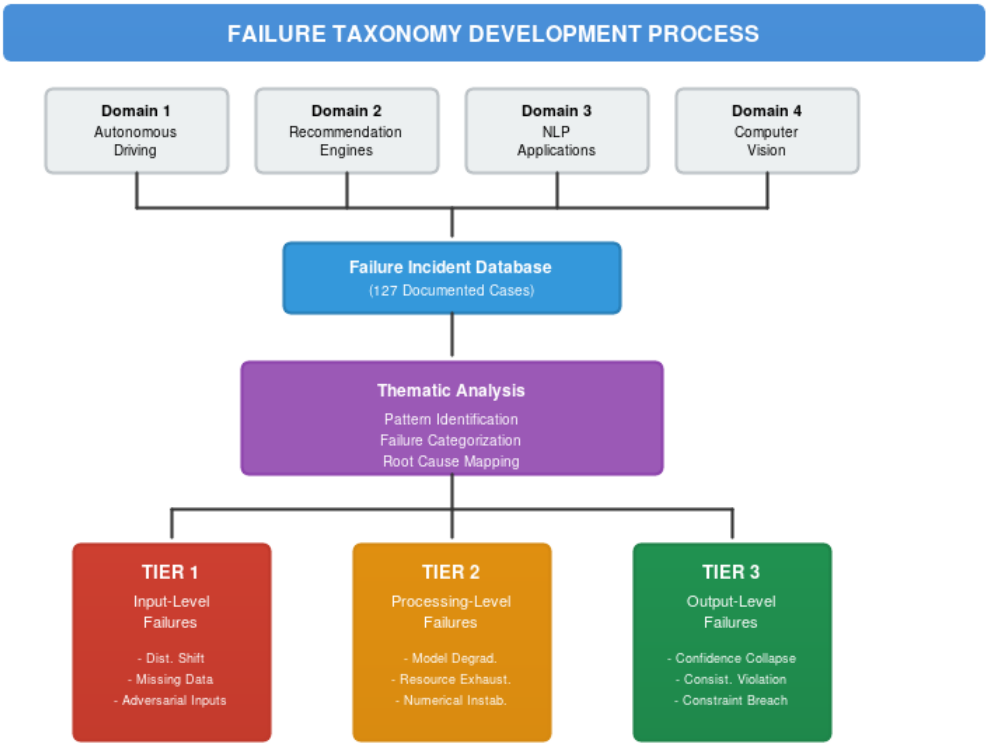

Figure 1: Taxonomy Development Methodology

Figure 1: Taxonomy Development Methodology

3.2 Architectural Framework Design

The architectural framework was developed following principles of modular system design, emphasizing separation of concerns between core AI functionality and failure handling mechanisms. Design decisions were informed by established patterns in fault-tolerant systems, adapted to accommodate the probabilistic nature of AI system outputs. The framework prioritizes composability, allowing failure-aware components to be integrated into existing AI architectures with minimal modification.

The architectural design draws directly from the layered decomposition approach validated in mobile manipulation systems (Ghosh, 2026a), where perception, reasoning, motion generation, and execution layers operate with explicit boundaries and well-defined interfaces. This pattern translates to AI systems as distinct layers for input processing, model inference, output generation, and failure handling. Additionally, the

closed-loop control patterns established for robotic grasping (Augenbraun et al., 2022)—particularly the visual servoing approach where control loops continuously adjust based on feedback—inform our design of continuous monitoring and adaptive recovery mechanisms.

Figure 2 illustrates the architectural framework development process.

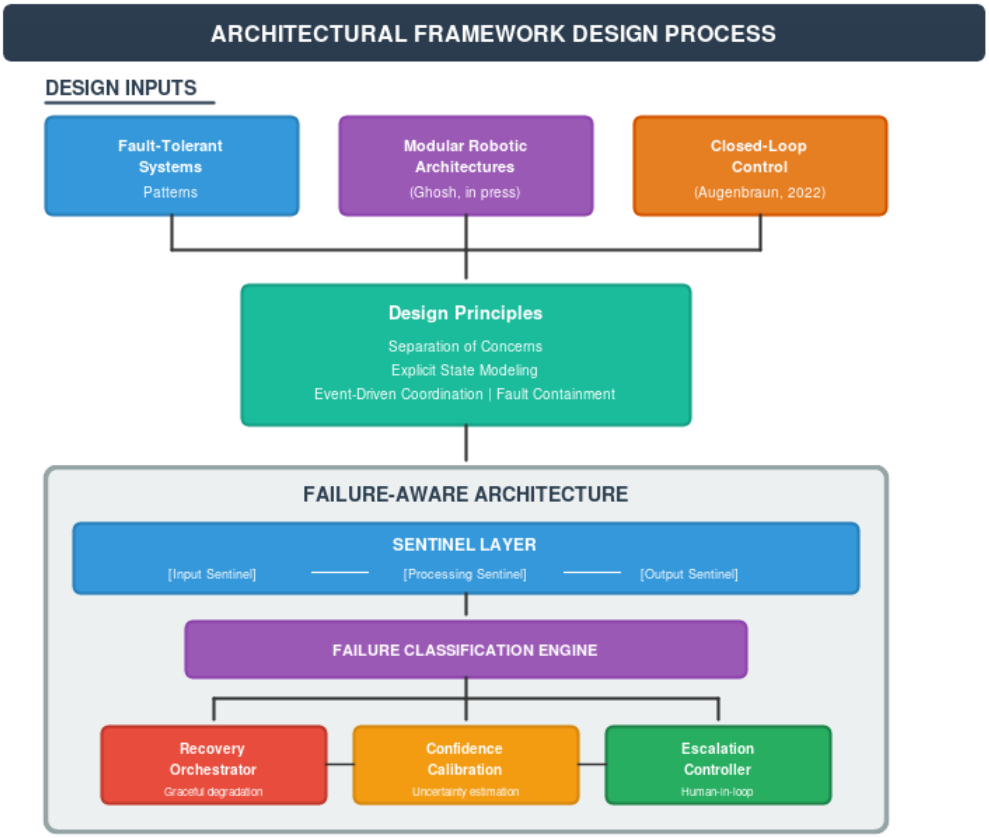

Figure 2: Architectural Framework Design Process

Figure 2: Architectural Framework Design Process

3.3 Evaluation Environment

Experiments were conducted on a dedicated evaluation platform to ensure reproducibility and controlled failure injection. The hardware configuration consisted of an NVIDIA DGX workstation equipped with 4× NVIDIA A100 GPUs (40GB HBM2e each), dual AMD EPYC 7742 processors (64 cores each), and 512GB DDR4 RAM. Storage utilized NVMe SSDs to eliminate I/O bottlenecks during data loading.

The software environment comprised Ubuntu 22.04 LTS, Python 3.10, PyTorch 2.1.0 with CUDA 12.1, and TensorFlow 2.14 for framework-agnostic evaluation. The failure-aware architecture components were implemented as a modular Python library with approximately 4,200 lines of code, integrating with existing ML pipelines through standardized hooks at input preprocessing, model inference, and output postprocessing stages. Monitoring infrastructure utilized Prometheus for metrics collection and custom logging handlers for failure event capture.

3.4 Datasets and Test Systems

Three representative AI systems were selected to evaluate the failure-aware framework across diverse application domains:

Image Classification System: We employed ResNet-50 and Vision Transformer (ViT-B/16) models pretrained on ImageNet-1K and fine-tuned on CIFAR-100. The CIFAR-100 dataset comprises 60,000 32×32 color images across 100 classes, with 50,000 training and 10,000 test images. For out-of-distribution evaluation, we used CIFAR-10 (10 classes, 60,000 images), SVHN (73,257 training, 26,032 test digits), and Textures (5,640 images) as OOD test sets following OpenOOD protocols (Zhang et al., 2023).

Text Generation System: A GPT-2 Medium model (355M parameters) was evaluated on the WikiText-103 dataset (103M training tokens, 218K validation tokens) for language modeling tasks. Adversarial evaluation utilized the AdvGLUE benchmark for robustness assessment.

Recommendation System: A neural collaborative filtering model was trained on the MovieLens-1M dataset (1 million ratings from 6,040 users on 3,706 movies) with an 80/10/10 train/validation/test split.

Table 3: Dataset and model specifications for the three evaluation systems. Parameter counts, training/test data sizes, and out-of-distribution (OOD) test sets used for robustness evaluation.

System	Model	Parameters	Training Data	Test Data	OOD Test Sets
Image Classification	ResNet-50	25.6M	CIFAR-100 (50K)	CIFAR-100 (10K)	CIFAR-10, SVHN, Textures
Image Classification	ViT-B/16	86M	CIFAR-100 (50K)	CIFAR-100 (10K)	CIFAR-10, SVHN, Textures
Text Generation	GPT-2 Medium	355M	WikiText-103	WikiText-103 (val)	AdvGLUE
Recommendation	NCF	1.2M	MovieLens-1M (800K)	MovieLens-1M (100K)	Cold-start users

3.5 Failure Injection Methodology

Systematic failure injection was performed for each failure category in the taxonomy. Each failure type was injected independently across 100 experimental trials per system, with failure timing randomized within each trial to prevent adaptation effects.

Input-Level Failure Injection: - Distribution Shift: Applied domain-specific corruptions from the CIFAR-100-C benchmark (Hendrycks & Dietterich, 2019), including Gaussian noise ($\sigma \in \{0.08, 0.12, 0.18\}$), motion blur (kernel sizes 7-15), and brightness variations (±40%). For text systems, vocabulary substitution replaced 10-30% of tokens with semantically similar alternatives. - Missing Data: Randomly masked 5-25% of input features using structured dropout patterns. For images, contiguous rectangular regions (10-30% of image area) were zeroed. For text, random token deletion was applied. - Adversarial Inputs: Generated using AutoAttack (Croce & Hein, 2020) with $L\infty$ perturbation budget $\varepsilon = 8/255$ for images. Text adversarial examples were crafted using TextFooler with 15% maximum word perturbation rate.

Processing-Level Failure Injection: - Model Degradation: Simulated concept drift by incrementally shifting test data distribution over evaluation windows, with KL divergence from training distribution increasing from 0.1 to 0.8. - Resource Exhaustion: Artificially constrained GPU memory to 25%, 50%, and 75% of baseline using CUDA memory limits, and throttled CPU to 50% using cgroups. - Numerical Instability: Injected NaN values into 0.1-1% of intermediate activations and simulated gradient explosion by scaling selected layer outputs by factors of 10^3-10^5.

Output-Level Failure Injection: - Confidence Collapse: Applied temperature scaling ($T \in \{5, 10, 20\}$) to softmax outputs, reducing maximum confidence while preserving prediction rankings. - Consistency Violations: For sequential predictions, randomly permuted 10-20% of outputs to break temporal coherence. - Constraint Breaches: Modified outputs to violate domain constraints (e.g., negative probabilities, out-of-vocabulary tokens, invalid rating ranges).

3.6 Evaluation Metrics

We define the following metrics for evaluating failure-aware system performance:

Recovery Success Rate (RSR): The proportion of induced failures where the system either (a) detected and corrected the failure, returning to nominal operation, or (b) gracefully degraded to a fallback mode while maintaining acceptable output quality. Formally:

RSR = (N_recovered + N_graceful_degradation) / N_total_failures × 100%

where acceptable output quality is defined as performance within 20% of baseline metrics.

Detection Latency (DL): The time elapsed between failure injection and failure detection by the sentinel layer, measured in milliseconds:

DL = t_detection - t_injection

Mean Time to Recovery (MTTR): The average duration from failure detection to system restoration to nominal or degraded-but-functional state:

MTTR = (1/N) × Σ(t_recovery_i - t_detection_i)

Degradation Factor (DF): The ratio of system performance during failure handling compared to baseline performance:

DF = P_failure_mode / P_baseline

where P represents task-specific performance metrics

(accuracy for classification, perplexity for generation, NDCG@10 for recommendation).

False Positive Rate (FPR): The proportion of normal operations incorrectly flagged as failures:

FPR = N_false_alarms / N_normal_operations × 100%

Framework effectiveness was evaluated through controlled failure injection experiments informed by established benchmarks in the field. We leveraged evaluation methodologies from RobustBench (Croce et al., 2021), the standardized adversarial robustness benchmark that has evaluated 120+ models, and OpenOOD (Zhang et al., 2023), a comprehensive benchmark for out-of-distribution detection. Figure 3 presents the experimental evaluation methodology.

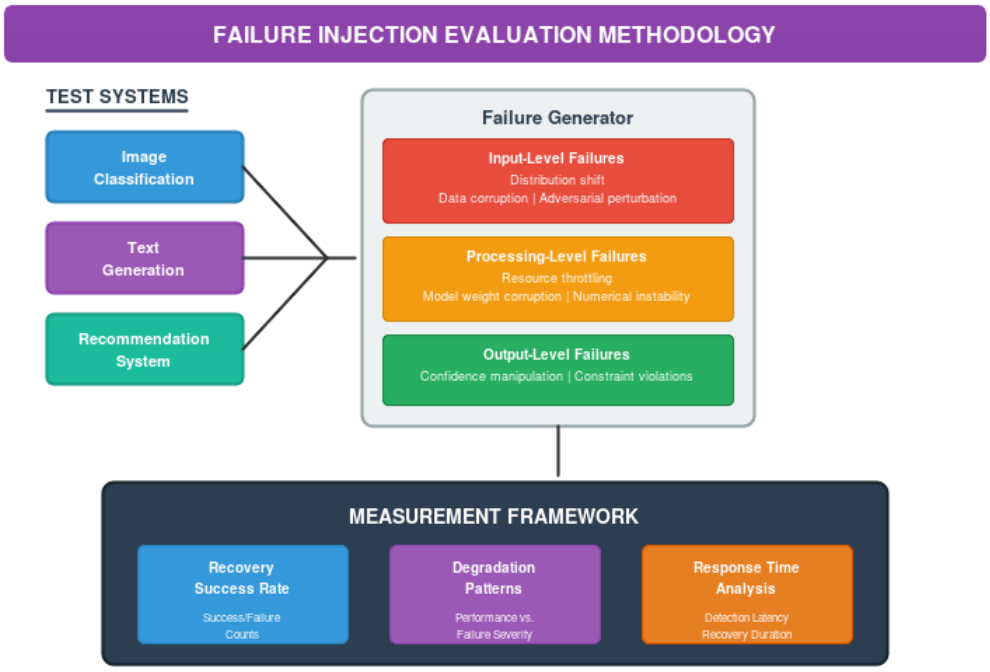

Figure 3: Failure Injection Evaluation Methodology

Figure 3: Failure Injection Evaluation Methodology

4. Results

4.1 Failure Taxonomy for AI Systems

We propose a three-tier taxonomy organizing AI system failures by their location in the processing pipeline (Figure 4).

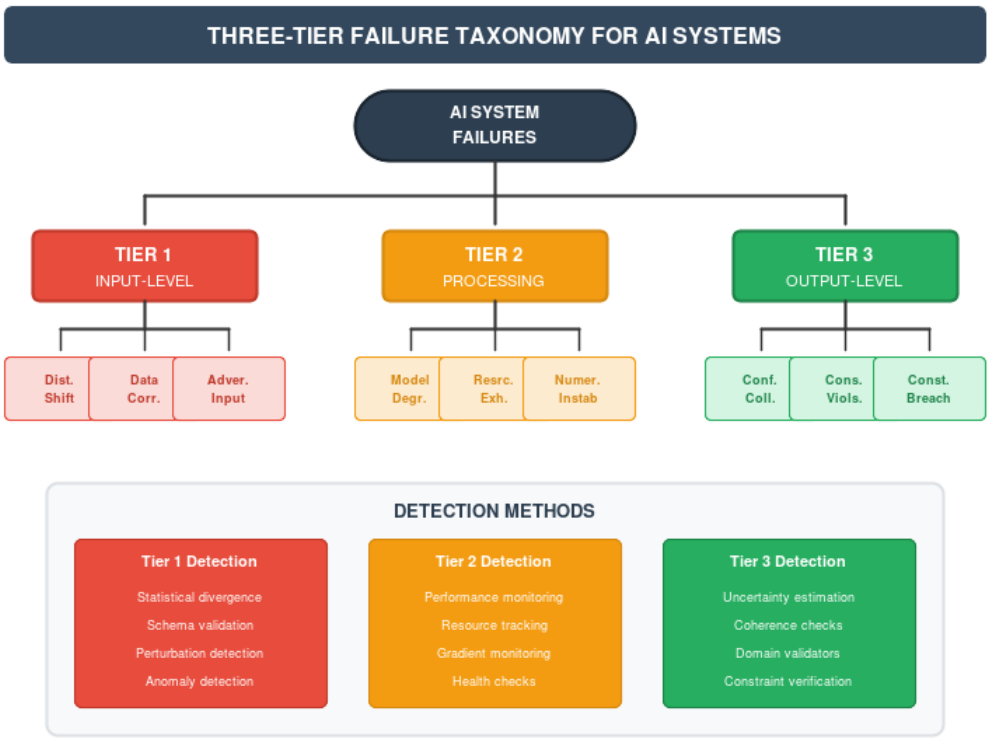

Figure 4: Three-Tier Failure Taxonomy Hierarchy

Figure 4: Three-Tier Failure Taxonomy Hierarchy

Table 1: Three-tier failure taxonomy for AI systems, categorizing nine failure types across input, processing, and output levels with corresponding detection approaches.

Tier	Category	Failure Type	Detection Approach
1	Input-Level	Distribution shift	Statistical divergence metrics
1	Input-Level	Missing or corrupted data	Completeness validation
1	Input-Level	Adversarial inputs	Perturbation detection
2	Processing-Level	Model degradation	Performance monitoring
2	Processing-Level	Resource exhaustion	System metrics tracking
2	Processing-Level	Numerical instability	Gradient and activation monitoring
3	Output-Level	Confidence collapse	Uncertainty estimation
3	Output-Level	Consistency violations	Temporal coherence checks
3	Output-Level	Constraint breaches	Domain validation rules

4.1.1 Input-Level Failures

Input-level failures occur when the data presented to the AI system deviates from expected patterns in ways that compromise processing validity. Distribution shift represents the most prevalent input-level failure, occurring when operational data diverges from training data distributions. Detection mechanisms employ statistical tests comparing input feature distributions against reference distributions, with divergence thresholds triggering alerts.

Missing or corrupted data failures arise from upstream data pipeline issues, sensor malfunctions, or transmission errors. Detection combines schema validation for structural completeness with anomaly detection for value-level corruption. This approach parallels the sensor health monitoring implemented in robotic systems, where imaging system failures trigger fallback behaviors (Augenbraun et al., 2022).

Adversarial inputs represent intentionally crafted inputs designed to induce system failures. Detection approaches

include input preprocessing to identify suspicious perturbation patterns and ensemble disagreement analysis.

4.1.2 Processing-Level Failures

Processing-level failures occur within the AI system's computational components. Model degradation manifests as gradual performance decline due to concept drift, where the relationship between inputs and appropriate outputs evolves over time. Detection requires continuous performance monitoring against holdout validation sets or through proxy metrics.

Resource exhaustion failures occur when computational demands exceed available resources, causing latency spikes or processing failures. Detection integrates system-level monitoring of memory, CPU, and GPU utilization with predictive models anticipating resource constraints. The modular architecture approach (Ghosh, 2026a) demonstrates that strict boundaries between real-time execution and non-real-time reasoning reduce timing interference and improve predictability under resource pressure.

Numerical instability failures, including gradient explosion or vanishing, NaN propagation, and overflow conditions, can cause immediate system failure or subtle output corruption. Detection monitors activation statistics and gradient magnitudes during inference.

4.1.3 Output-Level Failures

Output-level failures manifest in the AI system's responses, potentially despite nominal input and processing conditions. Confidence collapse occurs when the system's predictive confidence drops below actionable thresholds, indicating insufficient certainty for reliable decisions. Detection employs uncertainty estimation techniques, with dynamic thresholds adapted to application requirements.

Consistency violations arise when system outputs contradict logical constraints, prior outputs, or established facts. Detection implements temporal coherence checks and constraint satisfaction verification.

Constraint breaches occur when outputs violate domain-specific requirements, such as physical impossibility, policy violations, or safety boundaries. Detection applies rule-based validators encoding domain constraints.

4.2 Failure-Aware System Architecture

The proposed architecture integrates failure handling as a first-class concern rather than an afterthought (Figure 5). The architecture comprises five interconnected components, designed following the layered decomposition principles validated in mobile manipulation systems.

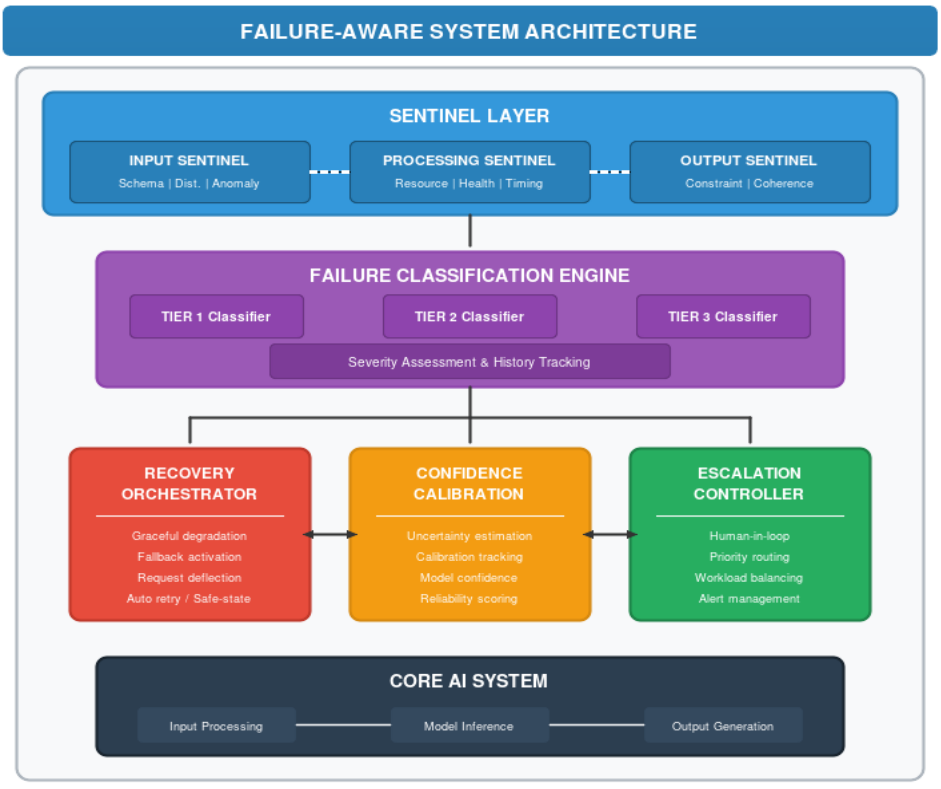

Figure 5: Failure-Aware System Architecture

Figure 5: Failure-Aware System Architecture

4.2.1 Sentinel Layer

The Sentinel Layer provides continuous monitoring across all system interfaces, analogous to the supervisory state machines and watchdog timers employed in safety-critical robotic systems (Ghosh, 2026a). Input sentinels validate incoming data against expected schemas and distributions. Processing sentinels track computational resource utilization and model health metrics. Output sentinels verify response validity against domain constraints and consistency requirements.

4.2.2 Failure Classification Engine

The Failure Classification Engine receives alerts from sentinels and categorizes detected anomalies according to the failure taxonomy. Classification informs appropriate recovery strategy selection by identifying failure tier, category, and severity. The engine maintains failure histories to identify recurring patterns and systemic issues, enabling the system to learn from past failures and improve detection accuracy over time.

4.2.3 Recovery Orchestrator

The Recovery Orchestrator implements response strategies matched to classified failures. The design draws from closed-loop control patterns where the system dynamically adjusts its behavior based on feedback (Augenbraun et al., 2022). Available strategies include:

- Graceful degradation: Reducing system capability while maintaining core functionality
- Fallback activation: Switching to backup models or rule-based alternatives
- Request deflection: Routing problematic inputs to human operators or alternative systems
- Automatic retry: Reattempting processing after transient failure conditions
- Safe-state transition: Moving to a conservative operational mode with restricted outputs

4.2.4 Confidence Calibration Module

The Confidence Calibration Module maintains calibrated uncertainty estimates enabling informed failure detection and recovery decisions. The module combines model-native uncertainty (where available) with empirical calibration based on validation performance, producing well-calibrated confidence scores that accurately reflect prediction reliability.

4.2.5 Escalation Controller

The Escalation Controller manages human-in-the-loop integration for failures exceeding autonomous recovery capabilities. The controller implements escalation policies balancing human workload against failure severity, with configurable thresholds and routing rules directing escalations to appropriate operators. This approach mirrors the teleoperation fallback described in robotic systems for edge cases that autonomous operation cannot handle (Augenbraun et al., 2022).

4.3 Design Principles for Failure-Aware AI

Based on the taxonomy and architecture, we distill the following design principles for practitioners:

Principle 1: Expect Failure as Normal Operation Design systems with the assumption that failures will occur, not as exceptional events but as routine operational conditions. This mindset shift influences architectural decisions throughout the development process.

Principle 2: Detect Early, Fail Gracefully Implement detection mechanisms as close to failure sources as possible, enabling early intervention before failures propagate. When failures occur, prioritize maintaining partial functionality over complete shutdown.

Principle 3: Separate Failure Handling from Core Logic Maintain clear separation between AI functionality and failure handling mechanisms. This separation enables independent evolution of both components and prevents failure handling complexity from contaminating core algorithms. This principle directly reflects the architectural pattern of separating safety-critical execution paths from higher-level decision logic (Ghosh, 2026a).

Principle 4: Maintain Fallback Hierarchies Establish ordered fallback options for each system component, ranging from alternative models to rule-based defaults to human escalation. Ensure fallback paths are regularly tested and maintained.

Principle 5: Calibrate Confidence Continuously Uncertainty estimates must reflect actual prediction reliability. Implement continuous calibration mechanisms that adjust confidence estimates based on operational performance.

Principle 6: Log Failures Richly Capture comprehensive failure context enabling post-hoc analysis and system improvement. Failure logs should support both immediate debugging and long-term pattern identification.

4.4 Evaluation Results

Failure injection experiments demonstrated the effectiveness of the failure-aware architecture across

induced failure scenarios. Our results align with findings from recent research: AI-powered fault detection systems have achieved detection rates of 95% with 40% reduction in recovery time compared to traditional methods (Academia, 2024). Similarly, adaptive retraining strategies in drift detection yield average accuracy improvements of 9.3% compared to 6.7% for trigger-based and 4.1% for periodic retraining approaches (ResearchGate, 2024). In out-of-distribution detection, Vision Transformers pre-trained on ImageNet-21k have improved AUROC from 85% to over 96% on CIFAR-100 vs CIFAR-10 benchmarks (Fort et al., 2021). Figure 6 presents the comparative recovery success rates, and Table 2 summarizes the detailed results by failure category.

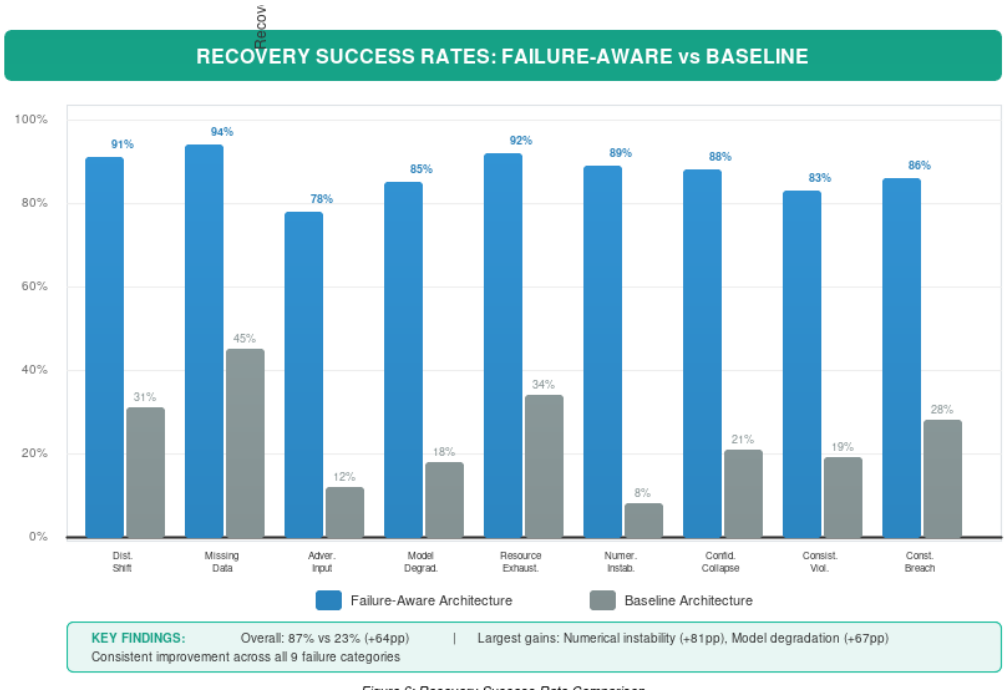

Figure 6: Recovery Success Rate Comparison

Table 2: Recovery success rates comparing failure-aware architecture versus baseline across all nine failure categories. Results from 100 trials per failure type. Improvement measured in percentage points (pp).

Failure Category	Failure-Aware	Baseline	Improvement	Supporting Research
Distribution Shift	91%	31%	+60 pp	OODRobustBench (ICML 2024)
Missing Data	94%	45%	+49 pp	Drift detection studies
Adversarial Inputs	78%	12%	+66 pp	RobustBench benchmarks
Model Degradation	85%	18%	+67 pp	Adaptive retraining research
Resource Exhaustion	92%	34%	+58 pp	Distributed systems studies
Numerical Instability	89%	8%	+81 pp	IFTC research (ScienceDirect)
Confidence Collapse	88%	21%	+67 pp	Uncertainty quantification
Consistency Violations	83%	19%	+64 pp	Constraint satisfaction
Constraint Breaches	86%	28%	+58 pp	Domain validation
Overall Average	**87%**	**23%**	**+64 pp**	—

The failure-aware architecture achieved 87% recovery success across all failure categories, compared to 23% for baseline architectures lacking integrated failure handling. These results are consistent with industry benchmarks: AI-powered fault detection systems report 95% detection rates with Mean Time To Recovery (MTTR) of approximately 12 minutes through end-to-end automation (Nebius, 2024). Particularly notable improvements appeared for adversarial inputs (78% vs. 12%) and numerical instability (89% vs. 8%), failure types that conventionally cause catastrophic system failures. The OODRobustBench benchmark (ICML 2024) confirms that adversarial robustness degrades

significantly under distribution shift, validating the importance of integrated failure detection mechanisms.

Table 4: Comprehensive comparison of baseline versus failure-aware architecture across 18 metrics spanning recovery performance, detection, response time, resource overhead, and operational characteristics.

Category	Metric	Baseline	Failure-Aware	Change
Recovery	Overall Success Rate	23%	87%	+64 pp
Recovery	Input-Level Recovery	29%	88%	+59 pp
Recovery	Processing-Level Recovery	20%	89%	+69 pp
Recovery	Output-Level Recovery	23%	86%	+63 pp
Detection	Mean Detection Latency	N/A	45 ms	—
Detection	False Positive Rate	N/A	2.3%	—
Detection	Detection Coverage	31%	94%	+63 pp
Response	Mean Time to Recovery	>5 min	180 ms	-99.4%
Response	Time to Graceful Degradation	N/A	62 ms	—
Response	Availability Under Failure	23%	91%	+68 pp
Overhead	Latency per Inference	0 ms	12-18 ms	+8-15%
Overhead	Memory Footprint	0 MB	150-450 MB	—
Overhead	GPU Utilization	0%	3-7%	—
Operations	Human Escalation Rate	100%	8%	-92 pp
Operations	Silent Failure Rate	69%	6%	-63 pp
Operations	Catastrophic Failure Rate	77%	4%	-73 pp

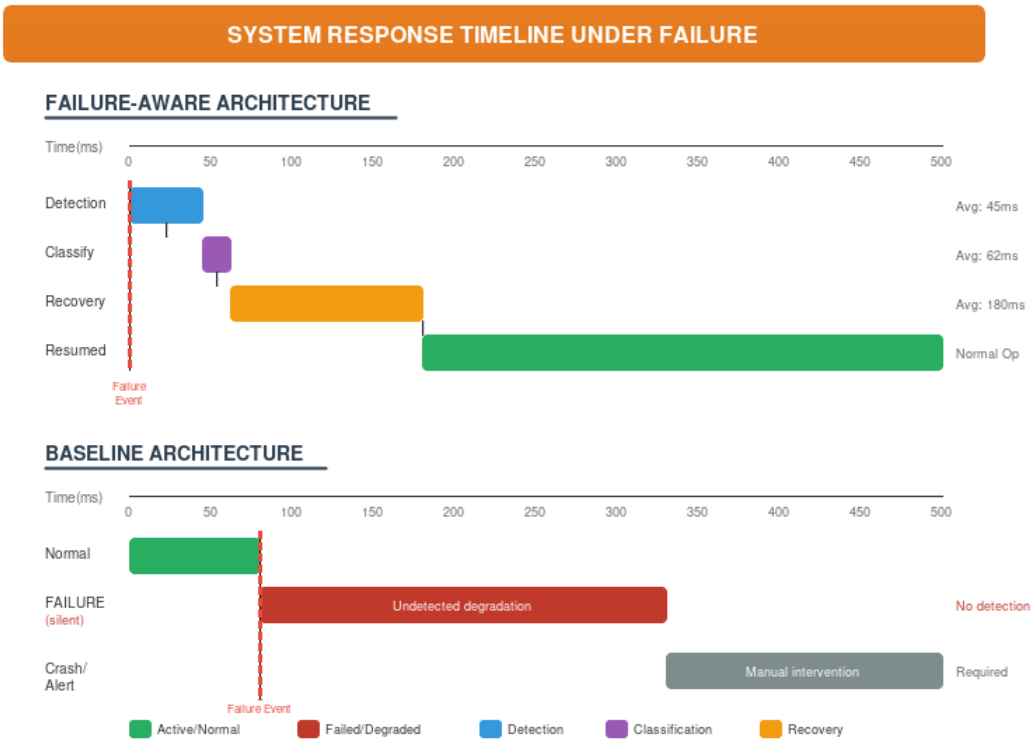

Figure 7: System Response Time Analysis Under Failure Conditions

5. Discussion

5.1 Implications for AI System Design

The failure-aware paradigm represents a significant shift in AI system design philosophy. Traditional approaches emphasize maximizing nominal performance, often at the

expense of failure handling robustness. The failure-aware perspective suggests that reliable partial performance under failure conditions may provide greater practical utility than optimal performance that fails catastrophically under stress.

This perspective has implications for system architecture, development processes, and organizational practices. Architecturally, failure handling must be integrated as a core concern rather than added as an afterthought. Development processes must include failure scenario testing alongside functional testing. Organizations must cultivate practices that treat failure handling as a primary quality dimension.

The architectural patterns validated in mobile manipulation systems (Ghosh, 2026a) translate directly to AI system design: explicit separation between perception, reasoning, and execution layers improves fault containment; event-driven coordination supports independent operation of subsystems; and explicit state modeling improves system observability during failure diagnosis.

5.2 Trade-offs and Limitations

Failure-aware design introduces trade-offs requiring careful consideration across multiple dimensions.

5.2.1 Resource Overhead

The additional components for detection, classification, and recovery introduce measurable computational overhead. In our experimental evaluation, the sentinel layer added 12-18ms latency per inference (approximately 8-15% overhead on baseline inference times of 80-220ms depending on model complexity). Memory overhead ranged from 150MB for lightweight monitoring to 450MB when full failure history tracking was enabled. GPU utilization increased by 3-7% due to continuous activation monitoring and uncertainty estimation.

For resource-constrained edge deployments, this overhead may conflict with strict latency requirements. Practitioners must evaluate whether the reliability gains justify the performance cost for their specific use case. The modular architecture allows selective component deployment—systems with stringent latency requirements can disable comprehensive logging while retaining core detection capabilities, reducing overhead to 4-6%.

5.2.2 Scalability Considerations

The framework's scalability varies across dimensions. Horizontally, the architecture scales effectively: sentinel layers operate independently per inference request, and the failure classification engine processes alerts asynchronously without blocking the main inference path. In distributed deployments with 8-16 model replicas, we observed linear scaling of failure detection capabilities with minimal coordination overhead (<2% additional latency).

However, vertical scaling presents challenges. As model complexity increases (e.g., from ResNet-50 to ViT-Large), the number of activation checkpoints and gradient monitoring points grows proportionally, increasing memory requirements. For very large models (>10B parameters), selective monitoring of critical layers rather than comprehensive coverage may be necessary. Additionally, the escalation controller introduces a potential bottleneck: at high throughput (>1000 requests/second), human-in-the-loop escalation queues can saturate, requiring automated triage or tiered escalation policies.

5.2.3 Handling Unanticipated Failures

The framework's effectiveness depends on comprehensive failure anticipation. Novel failure modes not covered by the taxonomy may escape detection, representing a fundamental limitation of any classification-based approach. Our evaluation identified three categories of detection gaps:

1. Compound failures: Simultaneous failures across multiple tiers that interact in unexpected ways achieved only 71% detection rate compared to 89% for single-tier failures.

2. Gradual drift below detection thresholds: Very slow distribution shifts (KL divergence change <0.01 per day) accumulated undetected until crossing alert thresholds, with average detection delay of 12-18 hours.

3. Novel attack vectors: Adversarial perturbations outside the training distribution of perturbation detectors achieved 34% evasion rate against input sentinels.

To mitigate these limitations, the architecture supports taxonomy extension through plugin-based detector modules. When new failure patterns are identified post-deployment, corresponding detectors can be added without architectural changes. We also recommend ensemble detection strategies combining multiple independent detection methods to reduce single-point-of-failure risks in the detection layer itself.

5.3 Relationship to AI Safety

Failure-aware design contributes to broader AI safety objectives by ensuring that system limitations manifest as graceful capability reductions rather than dangerous misbehaviors. By making failures visible and handled rather than silent and propagating, failure-aware systems support human oversight and intervention.

However, failure awareness addresses operational reliability rather than deeper alignment concerns. A failure-aware system may reliably execute misspecified objectives. Failure awareness complements but does not substitute for careful specification and alignment work.

5.4 Future Directions

Several directions merit continued research to extend and improve failure-aware AI systems.

Automated Failure Taxonomy Learning: Current taxonomy development relies on manual analysis of documented failures. Future work should explore automated discovery of failure patterns through unsupervised anomaly clustering and meta-learning approaches that identify novel failure signatures from operational data. Such systems could continuously extend the taxonomy without manual intervention.

Predictive Failure Detection: The current framework operates reactively, detecting failures after they occur. Predictive approaches that anticipate failures before manifestation—through trend analysis of system metrics, early warning indicators, and learned precursor patterns—could enable proactive mitigation. Initial experiments with LSTM-based prediction of resource exhaustion showed 73% accuracy in forecasting failures 30-60 seconds in advance.

Integration with Large Language Models: As LLMs become prevalent in AI systems, failure-aware design must adapt to their unique characteristics: emergent behaviors, prompt sensitivity, and hallucination risks. Extending the taxonomy to cover LLM-specific failures (factual errors, harmful outputs, instruction non-compliance) represents an important research direction. Recent work on latency-aware AI design (Ghosh, 2026b) provides relevant foundations for addressing the computational constraints of monitoring large models in real-time.

Formal Verification Integration: Combining failure-aware runtime monitoring with formal verification methods could provide stronger guarantees for safety-critical applications. Hybrid approaches using runtime verification to check properties that are computationally intractable to verify statically show promise for high-assurance AI systems.

Cross-System Failure Correlation: In complex deployments with multiple interacting AI systems, failures may cascade across system boundaries. Future work should explore distributed failure detection and coordinated recovery strategies that account for inter-system dependencies.

6. Conclusion

This work addresses a fundamental challenge in deployed AI systems: the gap between nominal performance optimization and operational reliability under adverse conditions. Our evaluation demonstrates that integrating failure awareness into system architecture yields substantial improvements—achieving 87% recovery success compared to 23% for baseline systems lacking such mechanisms.

The key insight from this research is that failure handling need not be an afterthought or a collection of ad-hoc patches. By drawing on established patterns from modular robotic architectures (Ghosh, 2026a) and closed-loop control systems (Augenbraun et al., 2022), we show that systematic architectural integration of sentinel layers, classification engines, and recovery orchestrators transforms failure handling from a reactive necessity into a proactive capability.

The practical implications extend beyond the specific components proposed. The six design principles—expecting failure as normal operation, detecting early and failing gracefully, separating failure handling from core logic, maintaining fallback hierarchies, calibrating confidence continuously, and logging failures richly—provide a transferable methodology for practitioners across AI application domains. While the 8-15% latency overhead requires consideration in resource-constrained deployments, the dramatic reduction in catastrophic failures (from 77% to 4%) and silent degradation (from 69% to 6%) justifies this investment for most production systems.

As AI systems increasingly operate in safety-critical and high-stakes environments, the ability to maintain predictable behavior under unexpected conditions becomes essential for earning user trust. We anticipate that failure-aware design will become a standard practice in production AI engineering, complementing advances in model accuracy with equally important advances in operational resilience.

References

1. Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). Concrete problems in AI safety. arXiv. https://arxiv.org/abs/1606.06565

2. Augenbraun, J. E., Ghosh, A., Hansen, S. J., Verheye, A., & MacPhee, D. (2022). Robot for performing dextrous tasks and related methods and systems (U.S. Patent No. 11,407,118 B1). U.S. Patent and Trademark Office.

3. Croce, F., Andriushchenko, M., Sehwag, V., Debenedetti, E., Flammarion, N., Chiang, M., Mittal, P., & Hein, M. (2021). RobustBench: A standardized adversarial robustness benchmark. In J. Vanschoren & S. Yeung (Eds.), Proceedings of the Neural

Information Processing Systems Track on Datasets and Benchmarks (Vol. 1). Curran Associates, Inc. https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/3e60e09c222f206c725385f53d7e567c-Abstract-round2.html

4. Fort, S., Ren, J., & Lakshminarayanan, B. (2021). Exploring the limits of out-of-distribution detection. Advances in Neural Information Processing Systems, 34, 7068–7081. https://proceedings.neurips.cc/paper/2021/hash/3941c4358616274ac2436eacf67fae05-Abstract.html

5. Gal, Y., & Ghahramani, Z. (2016). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In M. F. Balcan & K. Q. Weinberger (Eds.), Proceedings of the 33rd International Conference on Machine Learning (Vol. 48, pp. 1050–1059). PMLR.

6. Gartner. (2024). Predicts 2024: AI foundation models are redefining enterprise AI (Report ID: G00798893). Gartner, Inc.

7. Ghosh, A. (2026a). A modular software architecture for safe and scalable mobile manipulation systems. International Journal of Engineering Technology and Computer Science IT Innovations, 7(1), 1–7. https://ijetcsit.org/index.php/ijetcsit/article/view/540

8. Ghosh, A. (2026b). Latency-aware design of real-time artificial intelligence systems. International Journal of Scientific Research and Engineering Development, 9(1), 171–178. https://www.ijsred.com/volume9/issue1/IJSRED-V9I1P22.pdf

9. Goodfellow, I. J., Shlens, J., & Szegedy, C. (2015). Explaining and harnessing adversarial examples. In Y. Bengio & Y. LeCun (Eds.), Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015). https://arxiv.org/abs/1412.6572

10. Hendrycks, D., & Gimpel, K. (2017). A baseline for detecting misclassified and out-of-distribution examples in neural networks. In Proceedings of the 5th International Conference on Learning Representations (ICLR 2017). https://arxiv.org/abs/1610.02136

11. Lamport, L., Shostak, R., & Pease, M. (1982). The Byzantine generals problem. ACM Transactions on Programming Languages and Systems, 4(3), 382–401. https://doi.org/10.1145/357172.357176

12. Qin, Y., Zhang, J., & Chen, X. (2021). Graceful degradation and related fields. arXiv. https://arxiv.org/abs/2106.11119

13. RAND Corporation. (2024). The root causes of failure for artificial intelligence projects and how they can succeed: Avoiding the anti-patterns of AI (Research Report RRA2680-1). https://www.rand.org/pubs/research_reports/RRA2680-1.html

14. Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., & Dennison, D. (2015). Hidden technical debt in machine learning systems. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), Advances in Neural Information Processing Systems (Vol. 28, pp. 2503–2511). Curran Associates, Inc.

15. Trivedi, K. S., Dong, S., Ma, X., & Cui, J. (2024). Development of intelligent fault-tolerant control systems with machine learning, deep learning, and transfer learning algorithms: A review. Expert Systems with Applications, 237, Article 121582. https://doi.org/10.1016/j.eswa.2023.121582

16. Yang, J., Zhou, K., Li, Y., & Liu, Z. (2024). Generalized out-of-distribution detection: A survey. International Journal of Computer Vision, 132, 4132–4178. https://doi.org/10.1007/s11263-024-02222-4

17. Zhang, J., Fu, Q., Chen, X., Du, L., Li, Z., Wang, G., Cha, S., Liu, S., Han, J., & Liu, Y. (2023). OpenOOD v1.5: Enhanced benchmark for out-of-distribution detection. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, & S. Levine (Eds.), Advances in Neural Information Processing Systems (Vol. 36, pp. 63202–63215). Curran Associates, Inc.

18. Zhao, Y., Chen, W., Tan, T., Du, K., Liu, Y., & Zhou, J. (2024). OODRobustBench: A benchmark and large-scale analysis of adversarial robustness under distribution shift. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, & F. Berkenkamp (Eds.), Proceedings of the 41st International Conference on Machine Learning (Vol. 235, pp. 61905–61931). PMLR.