Practical Considerations for Failure Resilient ML Systems at the Edge

Krishna Praneet Gudipaty[1], Walid A. Hanafy[1], Li Wu[1], Jeffrey Twigg[2], Benjamin M. Marlin[1], Jesse Milzman[2], Suhas Diggavi[3], Tarek Abdelzaher[4], Prashant Shenoy[1]

[1]University of Massachusetts Amherst, [2]DEVCOM Army Research Laboratory, [3]University of California Los Angeles, [4]University of Illinois at Urbana-Champaign

***Abstract*—Machine learning at the edge is increasingly embedded in our daily lives, supporting applications running on smartphones, wearables, and industrial IoT. Prior work has mainly focused on resource efficiency and latency optimization through innovations in compact model design and resource-management techniques to meet stringent performance targets. However, edge devices and networks are inherently subject to failures and performance fluctuations, requiring an emphasis on failure resilience, especially in resource-constrained edge environments. Although recent studies have proposed resource-aware mechanisms and failure-aware models to improve the resilience of machine learning systems at the edge, many overlook deployment overheads that impede the adoption of these approaches. In this paper, we highlight practical considerations that affect failure-detection and recovery times and analyze how these considerations shape system design. We outline future research directions to enable practical, failure-resilient machine-learning systems at the edge.**

***Index Terms*—Edge Computing, ML Inference, Failure Resilience**

I. Introduction

Machine learning (ML) systems are becoming commonplace in today's world, from applications such as extended reality, autonomous driving, and surveillance to Industrial IoT and IoT analytics [1], [2]. To support the stringent latency requirements of modern ML applications, users typically rely on resources deployed at the edge of the network [3], [4]. Simultaneously, the rising computational intensity of modern ML models has driven widespread adoption of hardware accelerators (e.g., Edge TPUs and GPUs) in edge environments to deliver the throughput necessary for low-latency inference [4]. In addition to advances in edge computing and hardware accelerators [5], [6], the design of resource-efficient ML models is seeing significant advances. For instance, researchers have developed compressed/smaller ML models for resource-constrained edge environments with minor degradation in accuracy [7], [8].

Recent research in the design of ML systems on the edge, also referred to as Edge AI, has typically focused on performance and resource efficiency [1], [9]. For instance, researchers have proposed ML architectures and resource management approaches that optimize the latency and network requirements of ML models [10]–[13], as well as the efficiency and responsiveness in dynamic environments [5], [14], [15]. However, little work has focused on the resilience of ML applications on the edge.

Traditional approaches to optimizing failure resilience rely on the availability of failover resources, a standby, and idle resources that get activated when a failure occurs [16], [17]. Thus, when a failure occurs, user requests are redirected to another resource using a mixture of proactive and reactive techniques. Proactive techniques include running hot, denoted as active-active, where all replicas process the requests, and warm backups, denoted as active-passive, where replicas are loaded but left idle in case a failure occurs. Researchers have also explored reactive approaches (cold backups) where replicas are only activated when a failure occurs.

However, the resource-constrained nature of the edge renders these solutions impractical or adds high overhead. To design failure-resilient systems for resource-constrained environments, researchers often rely on graceful degradation [17], [18], where a subset of the applications or users is prioritized in case of failure. For example, in [18], in the event of failures, the authors shut down non-critical microservices. To address the resilience of ML systems on the edge, researchers have focused on the design of resource management techniques that trade off performance for resilience [19]–[21], and failure-aware distributed model architectures that can survive partial node failures [22]–[25].

However, prior work often overlooks practical challenges that hinder the deployment of such systems. First, failure-resilient systems typically target crash failures and assume stable performance otherwise. However, edge resources face a multitude of failures and exhibit performance variability, e.g., due to their multi-tenant nature or network instability. Second, many system designs presume ML models are immediately available and can be loaded on demand; in practice, the diversity of models required across failure scenarios and tight storage budget limits what can be pre-positioned, constraining feasibility. Finally, the end-to-end operational overhead of deploying and managing these mechanisms at scale is frequently neglected.

In this paper, we highlight the practical considerations for developing failure-resilient ML systems on the edge. We detail the technical challenges and overheads of failure-resilience ML systems across enterprise and Internet of Battlefield Things (IoBT) edge environments. In particular, we show how the performance metrics, such as loading time and response time, and reliability metrics, such as failure detection and recovery time, are affected. Lastly, we outline

future research directions to enable practical, failure-resistant ML systems on the edge. In summary, our contributions are threefold:

1) We describe key technical requirements and considerations that affect the practicality and design of failure-resilient ML systems.
2) We present a quantitative analysis of performance overheads and reliability metrics across enterprise-like and IoBT edge settings, and show how these two environments exhibit fundamentally different considerations.
3) We conclude the paper with a discussion of these challenges and future research directions.

II. Background

This section provides a background on model serving on the edge, failures, and failure resilience in resource-constrained environments.

A. Model Serving at the Edge

Edge computing brings computing and storage resources to the network edge, enabling low-latency access [3], [4]. The benefits of edge computing are becoming more pronounced with the rise of machine learning (ML) and its stringent latency requirements. Deploying ML models on the edge, also known as model serving or inference, assumes that a sensor or a client sends the input values, which are then processed by an edge node or a set of edge nodes, depending on the size and compute requirements of the ML model. To encapsulate the complexities of executing ML models, researchers and practitioners often rely on model serving runtimes and compilers such as TensorRT [26], ONNX runtime [27], or TVM [28] that provide a unified interface to run ML models across a wide range of general purpose compute nodes and ML accelerators (e.g., GPU and TPUs). After processing, the edge node then sends a model output or forwards it to another node in case of pipelines [29], [30].

B. Failures at the Edge

Compute and network resources are susceptible to multiple types of failures, including crash, omission, timing, partition, and byzantine failures that impact their availability [17], [20]. Unlike cloud resources that operate in tightly controlled and secure environments, edge resources are often deployed "in the wild" such as adversarial environments of IoBTs, where communication disruptions, device compromise, and malicious-input injection are credible risks [31]. Edge resources also face power and connectivity challenges as they may be battery-powered or rely on unreliable energy sources. Moreover, the lack of cooling capabilities may lead to thermal throttling or protective shutdowns [32], resulting in unpredictable performance degradations. Lastly, the geographic dispersion of edge resources and reliance on wireless access introduce variable latency, loss, and occasional partitions [21], [33].

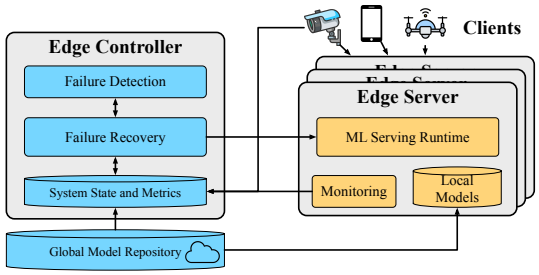


Fig. 1: Overview of failure resilient ML serving systems

C. Failure-Resilient Machine Learning at the Edge

In contrast to cloud resources, which have a high level of redundancy and low utilization, edge resources are typically resource-constrained. Thus, traditional failure resiliency that utilizes failover replicas is ill-suited for edge environments, requiring a different approach. To address the failure resiliency in edge environments, researchers typically employ the concept of graceful degradation, where resilience is achieved by sacrificing performance for non-critical applications. For instance, researchers in [19], [20] have exploited the trade-off between resource requirements and accuracy [8], [15] to improve the resilience and introduce the concept of heterogeneous replication, where failover replicas are selected as smaller variants of the same model. Moreover, researchers have addressed the failure resiliency by designing a failure-aware distributed model architecture that can survive partial failures [22]–[25]. For instance, [24] designs a distributed model architecture that leverages early exits and skip connections to ensure the resilience of ML models, and [25] proposes an ensemble architecture that tolerates multiple failures.

III. Design Overview and Considerations

In this section, we provide an overview of a typical architecture of a failure-resilient machine learning system and the design considerations and metrics for a practical deployment.

A. System Overview

Fig. 1 illustrates the architecture of a typical failure-resilient ML serving system. As shown, failure-resilient systems augment ML serving systems with failure detection and recovery capabilities. These runtimes often rely on a local model repository that can be instantly loaded as well as a centralized model repository (e.g., on a cloud server or a storage server) that hosts all ML models and variants. Moreover, in addition to ML serving runtime, edge servers host a monitoring agent (e.g., Prometheus [34], NVIDIA DCGM [35]) that monitors performance counters and systems' liveliness.

The edge controller collects performance metrics from edge servers and clients and implements failure detection and recovery. Researchers have designed failure recovery systems and policies that provide high availability and low mean time to recovery (MTTR) guarantees based on available resources, model sizes, and priority. For instance, in FailLite [19], the authors proposed a failure-aware model selection and placement heuristic that optimizes resilience while considering

TABLE I: Experimental Setups

SETUP NAME	HARDWARE	NETWORK
Enterprise Edge	5 PowerEdge R630 w NVIDIA A2 GPUs	10 Gbps Ethernet
IoBT Edge	5 NVIDIA Jetson Orin NX	WiFi

resource availability. This is achieved by using a mixture of warm backups for critical applications and cold backups for less critical ones, thereby optimizing both performance and availability.

Below, we highlight a set of performance considerations and metrics that have seen less attention in earlier approaches, as well as aspects that hinder the applicability of these approaches.

B. Design Objectives and Considerations

Earlier research has focused on the availability of model serving systems. However, failure resilient systems must consider a wide range of performance and reliability metrics, including:

Failure Detection Time: also denoted as time-to-detect, is the elapsed time between a component's failure (e.g., a crash) and the time a monitoring mechanism first declares/suspects that failure of such component [36].

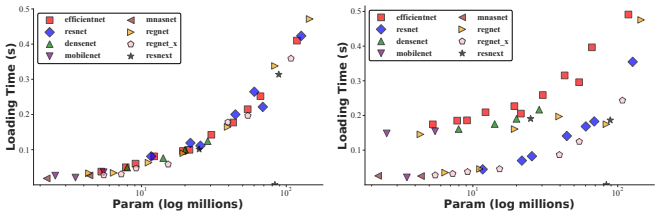
Decision Time: the elapsed time between detecting failures and making an error correction decision. For instance, in the case of FailLite [19], this includes the time to execute the placement heuristic or optimization approach.

Loading Time: the time it takes to load an ML model from disk to memory and prepare it for processing requests. This typically involves copying the model from disk to main memory, followed by transfer to the accelerator memory.

Reconfiguration Time: the time to switch clients from the failed application endpoint to a new endpoint.

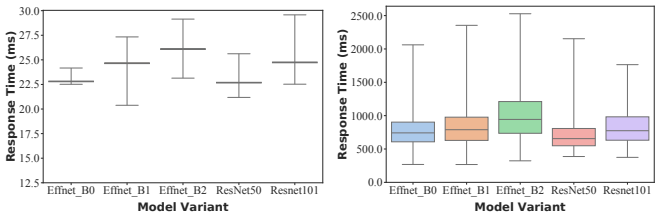
Performance Overheads: system overheads that occur when deploying failure-resilient systems, such as increased or fluctuating response times, when executing failover backups.

In addition to the metrics, multiple aspects limit the execution of failure-resilient recovery procedures. First, systems often assume the correctness of failure detection. However, in practice, failure detection systems may produce False Positives or False Negatives, initiating unnecessary changes or not reacting appropriately [36]. Second, failure-resilience systems typically assume that failover models are available either locally or on a nearby server. However, in many cases, these backups may not be available due to storage constraints, cold backup misplacement, or only available in a remote cloud storage, resulting in high loading and recovery times. Third, systems have not addressed scenarios of cascading failures, either by dependency across applications or failures caused by fail-recovery actions themselves. Finally, researchers have confirmed that controllers are deployed on reliable servers. However, similar to other edge sites, controllers are prone to failures. Moreover, most designs rely on a single node and may present a single point of failure.



(a) Enterprise Edge (b) IoBT Edge

Fig. 2: Loading time of ML models across setups.



(a) Enterprise Edge (b) IoBT Edge

Fig. 3: Response Time across setups.

IV. Evaluation

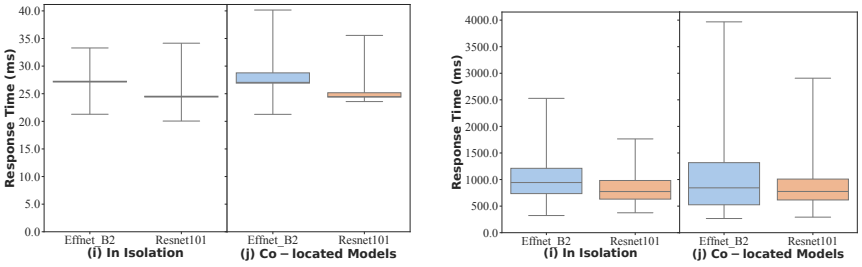
This section presents an experimental evaluation of the considerations and metrics for failure-resilient ML systems on the edge, as well as an end-to-end assessment of failure-resilience systems' performance.

A. Experimental Setup

Hardware Setup. Table I lists our two experimental testbeds that highlight different edge scenarios, denoted as enterprise edge and IoBT edge. The enterprise edge setup represents highly efficient edge sites such as Akami CDN deployment [37] or AWS Edge Services [38] with higher resource density and more stable connections. Our enterprise edge setup features Dell PowerEdge R630 machines, equipped with Xeon E5-2660v3 CPU, 256 GB of memory, a 400 GB Intel 730 SSD, and up to 10 Gbps networking speed. Additionally, each server includes an NVIDIA A2 GPU with 1280 CUDA cores and 16 GB of GPU memory. In contrast, the IoBT edge setup represents deployment in the wild with highly constrained infrastructures and unstable network connections [31]. This setup features NVIDIA Jetson Orin NX with 8 Arm Cortex-A78AE CPU cores, 1024 CUDA cores, and 16 GB of memory. These Orin nodes are interconnected via a WiFi network, and each is equipped with a 1 TB Samsung NVMe SSD 970 EVO Plus.

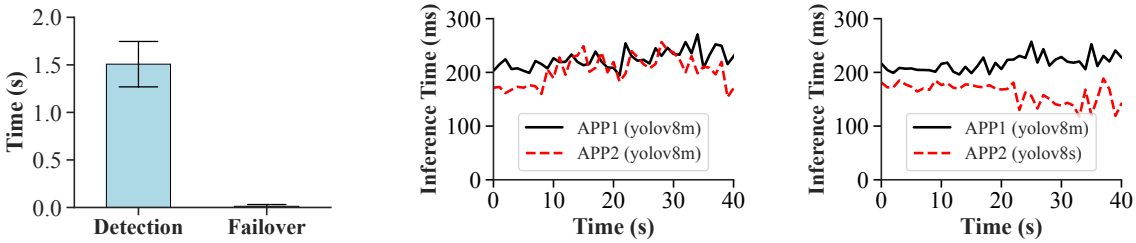
ML Models. In addition, our evaluation utilizes a wide set of ML architectures and sizes. Our evaluation uses several models: Including models from the PyTorch model repository [39], Yolov8 (n,s,m) [40], and custom failure-resilient ensemble models [25].

ML Serving System. The model serving is implemented as a containerized service that encapsulates the model, runtime, and dependencies. We utilize the Torch-TensorRT [41] runtime to run all models, except for YoloV8, which directly uses



(a) Enterprise Edge (b) IoBT Edge

Fig. 4: Effect of Co-location across setups.



(a) Detection and recovery time (b) Failover with large model (c) Failover with smaller model

Fig. 5: Failover overhead and interference.

PyTorch. We utilize two deployment methodologies. First, we use a client-server approach, where clients send requests in a closed loop manner via gRPC [42] to the processing server, or multiple in the case of ensemble models. Second, we utilize a deployment based on MQTT [43], where data sources publish inference requests (e.g., video streaming frames), and the server publishes back the results, without direct interactions between the client and the server.

B. Design Considerations

Below, we analyze the performance across the enterprise and IoBT setups and highlight key system metrics that affect the performance of failure-resilient systems.

Loading Time. Fig. 2 illustrates the loading time of machine learning models across different numbers of parameters. Loading time affects the MTTR of failure-resilient systems, especially in situations where the controller relies on cold backups. As shown, the loading time is highly correlated with the number of parameters, as it highly affects the model size. In either setting, even though we use highly optimized storage, the loading time can be as high as 0.5s, with an average of 0.15s and 0.17s across the enterprise and IoBT edge settings. This high overhead not only introduces a system bottleneck but also significantly reduces the system's availability.

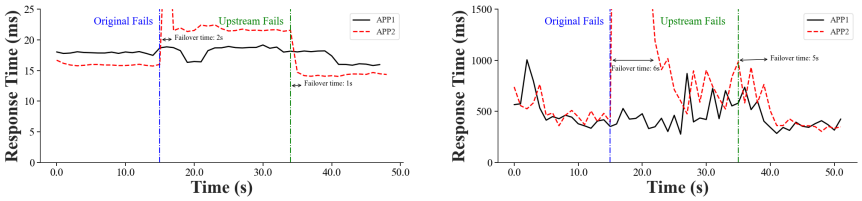
Response Time. Fig. 3 illustrates the response time of five ML models across setups, a key aspect of model serving systems. As shown, the response time highly changes across setups and models. For instance, in the enterprise edge setting, the response time is low and highly stable. In contrast, in the IoBT setting, the response time is much higher due to differences in accelerator performance and network speed. For instance, for Efficientnet-B2 (Effnet_B2), the inference time and network latency are higher by 400% and around 3500% on the IoBT edge compared to enterprise edge settings. More importantly, the results highlight how the network affects the entire performance, which is more pronounced in the IoBT settings. For instance, the Coefficient of Variation (CoV) for processing time and latency changes is 0.22, and 0.12 in enterprise edge settings, and 0.32 and 0.14 in the IoBT edge settings, respectively.

Effect of Co-location. Failure-resilient systems often utilize failover replicas where models may be co-located. Fig. 4 highlights the effect of ML models' co-location across setups when considering EfficientNet-B2 (EffNet_B2) and ResNet-101 (Resnet101) ML models running in isolation vs together. As shown, co-location highly increases the response time compared to the isolated execution case, especially in the IoBT settings. For instance, in the enterprise edge, due to the size of the GPU, the effect of co-location is less apparent, where the average latency only increases by 3.1% and 2.1% for Effnet_B2 and Resnet101, respectively. In contrast, in the IoBT settings, while using more constrained devices, the overhead is not apparent due to the highly variable network configurations. However, our evaluation highlights that the inference time increases by 7.8% and 11% for Effnet_B2 and Resnet101, respectively.

C. End-to-End Results

Below, we present an end-to-end evaluation of failure-resilient systems deployment that utilizes heterogeneous replication [19], [20] and distributed ensemble models [25].

Heterogeneous Replication. We start by evaluating a system based on heterogeneous replication [19], [20], while sending requests using MQTT. Fig. 5 (a) presents the failure detection and recovery times at the IoBT edge using heartbeat signals. Heartbeats are sent to the edge controller every 50 ms. To



(a) Enterprise Edge (b) IoBT Edge

Fig. 6: Ensemble model overhead across setups using warm backups.

mitigate false negatives caused by the unreliable network, a longer confirmation window is employed, which leads to an average detection time of 1.5s. In contrast, the recovery time of an application backed up with a warm replica can be significantly lower, with an average of 3ms.

Fig. 5 (b) and (c) illustrate the overhead of homogeneous and heterogeneous replicas at the IoBT edge. We assume that APP1 deploys a failover back on the server hosting APP2 and vice versa. In our settings, at the 10th second, APP2 fails over from its primary server to the backup server, which also hosts APP1. Resource contention—primarily on the GPU—causes APP2's inference time to increase by 17.2%, while APP1's inference time rises by 9.8%. When APP2 uses a smaller model variant as a backup, its inference time decreases by 11.8%, while increasing APP1 inference time by 8%.

Ensemble Models. Here, we show the performance of an ensemble-based approach [25], where the failure detection is implemented using timeouts, and clients send requests in a closed loop. In this experiment, we utilize four servers, one hosting the original and the other three hosting parts of the ensemble model. We also assume one of these servers is hosting APP2. Fig. 6 shows the response time and failover time of the two applications across three stages: normal operation, original failure, and partial failure (i.e., failure in one of the upstream models), where changes are injected at 15s and 35s, respectively. As shown in Fig. 6 (a), in the enterprise edge, the failover time is minimal ∼2s. In this case, when the original model fails, the client switches to the ensemble model, which is distributed across multiple nodes, and hence increases the response time. Then, when we inject another failure, the client switches to a single small model, thus resulting in a lower response time. Similarly, as shown in Fig. 6 (b), the IoBT edge setting, the client switches between deployments, resulting in different response times. However, the differences are less apparent, and the switching is less smooth when compared to the enterprise edge setting. This is due to the network latency dominating the processing latency, as was highlighted earlier. More importantly, in this setting, the failure detection time, which highly depends on network latency [36], is much higher ∼6s.

V. Discussion and Future Direction

A. Discussion

In previous sections, we highlighted key considerations that aid the design of failure-resilient ML systems. Our experimental evaluation showed the performance metrics and reliability metrics across two distinct edge classes: Enterprise edge and IoT edge. The results highlight how the performance overheads vary across these settings and may result in different reliability guarantees. For instance, while computing presents a bottleneck in both cases, the network presents a significant challenge in the IoBT settings, leading to highly variable results.

B. Future Research Directions

Below, we list key future research directions:

Failure diagnostics and localization. A key direction is developing better failure detection and localization approaches. Failure diagnostics can aid the design, and the availability of ML systems differs based on failure types. For instance, while a crash failure might require a failover back, network partition failures may require a different placement strategy. One approach is to adopt multi-point failure detection and machine learning approaches [44].

Distributed Controllers. Another direction is designing distributed controllers that address issues in current centralized placement and resource management approaches. For example, the usage of a multi-agent controller and fault-tolerant consensus control can aid in the design of future fault-tolerant systems [45].

Navigating the trade-offs. Our analysis highlights the trade-off between performance, accuracy, and availability. For instance, the usage of small failover replicas, as in FailLite [19], may optimize the performance and availability but affect the accuracy. In contrast, approaches that utilize ensembles, such as MEL, may prioritize accuracy and availability but sacrifice performance. Thus, analyzing this three-way trade-off may be critical for the design of future failure-resilient systems.

VI. Conclusion

This paper highlights key considerations and performance metrics that affect the practicality of failure-resilient machine learning systems. Our analysis across two distinct classes of edge clusters highlights how the infrastructure impacts the forecast performance, overheads, and availability. Our study shows that compared to the enterprise settings, IoBT settings may see lower availability and higher system overheads, requiring a new set of approaches and techniques. In the context of future work, we highlight directions to include the

design of multi-point failure detections, multi-agent consensus protocols, and approaches to navigate the performance, availability, and accuracy tradeoff.

ACKNOWLEDGEMENT

This research is supported by NSF grants 2325956, 2211302, 2211888, 2213636, 2105494, 23091241, US Army contract W911NF-17-2-0196, and Adobe.

REFERENCES

[1] Z. Chang *et al.*, "A survey of recent advances in edge-computing-powered artificial intelligence of things," *IEEE Internet of Things Journal*, vol. 8, no. 18, pp. 13849–13875, 2021.

[2] S. I. Siam *et al.*, "Artificial intelligence of things: A survey," *ACM Transactions on Sensor Networks*, vol. 21, Jan. 2025.

[3] M. Satyanarayanan and N. Davies, "Augmenting Cognition Through Edge Computing," *Computer*, vol. 52, no. 7, pp. 37–46, 2019.

[4] M. Satyanarayanan, N. Beckmann, G. A. Lewis, and B. Lucia, "The Role of Edge Offload for Hardware-Accelerated Mobile Devices," in *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*, HotMobile '21, p. 22–29, 2021.

[5] Q. Liang, W. A. Hanafy, N. Bashir, A. Ali-Eldin, D. Irwin, and P. Shenoy, "DěLen: Enabling Flexible and Adaptive Model-Serving for Multi-Tenant Edge AI," in *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation*, IoTDI '23, p. 209–221, 2023.

[6] Q. Liang, W. A. Hanafy, A. Ali-Eldin, and P. Shenoy, "Model-Driven Cluster Resource Management for AI Workloads in Edge Clouds," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 18, mar 2023.

[7] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," *Communications of the ACM*, vol. 63, p. 54–63, Nov. 2020.

[8] W. A. Hanafy, T. Molom-Ochir, and R. Shenoy, "Design Considerations for Energy-Efficient Inference on Edge Devices," in *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*, e-Energy '21, p. 302–308, 2021.

[9] R. Singh and S. S. Gill, "Edge ai: A survey," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 71–92, 2023.

[10] J. Huang, C. Samplawski, D. Ganesan, B. Marlin, and H. Kwon, "CLIO: Enabling Automatic Compilation of Deep Learning Pipelines across IoT and Cloud," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, MobiCom '20, 2020.

[11] S. Zhang, S. Zhang, Z. Qian, J. Wu, Y. Jin, and S. Lu, "DeepSlicing: Collaborative and Adaptive CNN Inference With Low Latency," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 9, pp. 2175–2187, 2021.

[12] A. Misra, N. Saoda, and T. Abdelzaher, "Latency-Constrained Input-Aware Quantization of Time Series Inference Workflows at the Edge," in *IEEE INFOCOM 2025 - IEEE Conference on Computer Communications*, pp. 1–10, 2025.

[13] C. Zhang, M. Yu, w. wang, and F. Yan, "Enabling Cost-Effective, SLO-Aware Machine Learning Inference Serving on Public Cloud," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2020.

[14] C. Wan, M. Santriaji, E. Rogers, H. Hoffmann, M. Maire, and S. Lu, "ALERT: Accurate learning for energy and timeliness," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pp. 353–369, USENIX Association, July 2020.

[15] J. Zhang, S. Elnikety, S. Zarar, A. Gupta, and S. Garg, "Model-Switching: Dealing with fluctuating workloads in Machine-Learning-as-a-Service systems," in *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*, USENIX Association, July 2020.

[16] J. Soifer, J. Li, M. Li, J. Zhu, Y. Li, Y. He, E. Zheng, A. Oltean, M. Mosyak, C. Barnes, T. Liu, and J. Wang, "Deep Learning Inference Service at Microsoft," in *2019 USENIX Conference on Operational Machine Learning (OpML 19)*, (Santa Clara, CA), pp. 15–17, may 2019.

[17] I. Koren and C. M. Krishna, *Fault-Tolerant Systems*. Morgan Kaufmann, second edition ed., 2021.

[18] J. J. Meza *et al.*, "Defcon: Preventing Overload with Graceful Feature Degradation," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, (Boston, MA), pp. 607–622, USENIX Association, July 2023.

[19] L. Wu, W. A. Hanafy, T. Abdelzaher, D. Irwin, J. Milzman, and P. Shenoy, "FailLite: Failure-Resilient Model Serving for Resource-Constrained Edge Environments," 2025.

[20] W. A. Hanafy, L. Wu, T. Abdelzaher, S. Diggavi, and P. Shenoy, "Failure-Resilient ML Inference at the Edge through Graceful Service Degradation," in *MILCOM 2023 - 2023 IEEE Military Communications Conference (MILCOM)*, pp. 144–149, 2023.

[21] I. M. Amer *et al.*, "Task Replication in Unreliable Edge Networks," in *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, pp. 173–180, 2022.

[22] A. Yousefpour, S. Devic, B. Q. Nguyen, A. Kreidieh, A. Liao, A. M. Bayen, and J. P. Jue, "Guardians of the deep fog: Failure-resilient dnn inference from edge to cloud," in *Proceedings of the first international workshop on challenges in artificial intelligence and machine learning for internet of things*, pp. 25–31, 2019.

[23] A. Yousefpour, B. Q. Nguyen, S. Devic, G. Wang, A. Kreidieh, H. Lobel, A. M. Bayen, and J. P. Jue, "Resilinet: Failure-resilient inference in distributed neural networks," *arXiv preprint arXiv:2002.07386*, 2020.

[24] A. A. Majeed, P. Kilpatrick, I. Spence, and B. Varghese, "Continuer: maintaining distributed dnn services during edge failures," in *2022 IEEE International Conference on Edge Computing and Communications (EDGE)*, pp. 143–152, IEEE, 2022.

[25] K. P. Gudipaty, W. A. Hanafy, K. Ozkara, Q. Liang, J. Milzman, P. Shenoy, and S. Diggavi, "MEL: Multi-level Ensemble Learning for Resource-Constrained Environments," 2025.

[26] NVIDIA Corporation, "NVIDIA TensorRT sdk." https://developer.nvidia.com/tensorrt. Accessed: 2025-08-15.

[27] Microsoft Corp., "ONNX Runtime." https://onnxruntime.ai/, 2025. Accessed: 2025-05-20.

[28] Apache Software Foundation, "Apache TVM." https://tvm.apache.org/. Accessed: 2025-08-15.

[29] L. Wu *et al.*, "Enhancing Resilience in Distributed ML Inference Pipelines for Edge Computing," in *MILCOM 2024 - 2024 IEEE Military Communications Conference (MILCOM)*, pp. 1–6, 2024.

[30] S. Ahmad, H. Guan, and R. K. Sitaraman, "Loki: A System for Serving ML Inference Pipelines with Hardware and Accuracy Scaling," in *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '24, p. 267–280, 2024.

[31] T. Abdelzaher *et al.*, "Toward an Internet of Battlefield Things: A Resilience Perspective," *Computer*, vol. 51, no. 11, pp. 24–36, 2018.

[32] R. Rao and S. Vrudhula, "Performance optimal processor throttling under thermal constraints," in *Proceedings of the 2007 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '07, p. 257–266, 2007.

[33] M. Z. n. Zamalloa and B. Krishnamachari, "An analysis of unreliability and asymmetry in low-power wireless links," *ACM Transactions on Sensor Networks*, vol. 3, p. 7–es, June 2007.

[34] The Prometheus Authors, "Prometheus: Monitoring System & Time Series Database," 2025. Accessed: 2025-08-17.

[35] NVIDIA Corporation, "NVIDIA Data Center GPU Manager (DCGM)," 2025. Accessed: 2025-08-17.

[36] N. Hayashibara *et al.*, "The φ accrual failure detector," in *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, 2004.*, pp. 66–78, 2004.

[37] Akamai, "Akamai CDN Network Deployment: Facts and Figures." https://www.akamai.com/company/facts-figures, August 1st 2023.

[38] Amazon Web Services, Inc., "AWS Edge Computing," 2025. Accessed: 2025-08-18.

[39] PyTorch, "Torchvision models and pre-trained weights," 2024. Accessed: 2025-04-14.

[40] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLOv8," 2023.

[41] PyTorch, "Pytorch tensorrt documentation." https://docs.pytorch.org/TensorRT/, 2025. Accessed: 2025-08-20.

[42] Google, "gRPC: A high performance, open-source universal RPC framework." https://grpc.io/, 2025. Accessed: 2025-08-20.

[43] MQTT.org, "Mqtt: The standard for iot messaging." https://mqtt.org/, 2025. Accessed: 2025-08-20.

[44] D. Cotroneo *et al.*, "Enhancing the analysis of software failures in cloud computing systems with deep learning," *Journal of Systems and Software*, vol. 181, p. 111043, 2021.

[45] C. Gao *et al.*, "A survey on fault-tolerant consensus control of multi-agent systems: trends, methodologies and prospects," *International Journal of Systems Science*, vol. 53, no. 13, pp. 2800–2813, 2022.