Enhancing Resilience in Distributed ML Inference Pipelines for Edge Computing

Li Wu[1], Walid A. Hanafy[1], Abel Souza[2], Tarek Abdelzaher[3], Gunjan Verma[4], and Prashant Shenoy[1]

[1]University of Massachusetts Amherst, [2]University of California, Santa Cruz
[3]University of Illinois Urbana Champaign, [4]Army Research Lab

***Abstract*—As edge computing and sensing devices continue to proliferate, distributed machine learning (ML) inference pipelines are becoming popular for enabling low-latency, real-time decision-making at scale. However, the geographically dispersed and often resource-constrained nature of edge devices makes them susceptible to various failures, such as hardware malfunctions, network disruptions, and device overloading. These edge failures can significantly affect the performance and availability of inference pipelines and the sensing-to-decision-making loops they enable. In addition, the complexity of task dependencies amplifies the difficulty of maintaining performant and reliable ML operations. To address these challenges and minimize the impact of edge failures on inference pipelines, this paper presents several fault-tolerant approaches, including sensing redundancy, structural resilience, failover replication, and pipeline reconfiguration. For each approach, we explain the key techniques and highlight their effectiveness and trade-offs. Finally, we discuss the challenges associated with these approaches and outline future directions.**

I. Introduction

As the Internet of Things (IoT) and sensing devices continue to proliferate, a massive influx of data is being generated at the network's edge. This surge in data demands efficient processing capabilities, which has paved the way for edge computing. Edge computing extends cloud computing by bringing computational resources closer to data sources, reducing latency and bandwidth usage while enabling real-time decision-making [1]. This strategic placement of resources has fostered the development of latency-sensitive machine learning (ML) applications such as streaming video analytics, autonomous driving, and augmented reality [1].

Advances in IoT systems have enabled their vast and complex deployments with systems comprised of sensing nodes with multi-modal sensing capabilities. For example, sensing nodes often have a microphone and a camera, and the compute resources are shared between the two [2], [3]. Edge networks utilize the available resources to enhance applications' accuracy by orchestrating complex decision-making processes through distributed ML inference pipelines. ML inference pipelines combine data from various sensors, integrate multiple ML models into dataflow graphs, and are deployed on multiple edge servers. Fig. 1 shows an example ML inference pipeline, where inputs from multiple sensors are merged to form complex detection tasks. The results

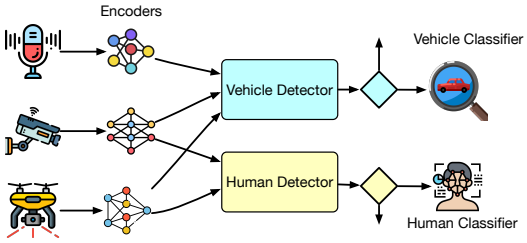


Fig. 1: Distributed ML Inference Pipelines at the edge.

of these detection phases are later utilized for downstream classification and fusion tasks.

In contrast to traditional cloud data centers, edge computing clusters are often deployed across multiple geographically-distributed locations and may communicate using wireless networks that lack advanced support systems, such as advanced heat management or outage support systems [4]. Moreover, unlike cloud platforms, edge resources often lack a high degree of compute, network, or power redundancy. For example, a network failure could disrupt multiple servers within the same physical/virtual network. In addition, server failures, such as transient, crash, and byzantine failures, are common, especially in the adversarial environment of the Internet of Battlefield Things (IoBTs). In IoBT scenarios, battlefield intelligent devices are attacked, and sensors can be deceived using malicious inputs [5]. Although these failures have significant implications for traditional ML systems, the effect of failures is exacerbated for inference pipelines, where a single bottleneck node failure can disrupt multiple sensing-to-decision loops.

Resilient execution of edge ML applications has been extensively studied, where researchers developed multiple mechanisms to increase the resiliency of their deployed applications [6], [3]. For example, researchers have proposed failover replication methods that utilize model selection strategies by placing smaller standby backups to ensure availability. In addition, researchers have used resource allocation and scheduling techniques by executing requests at a further node or a lower priority [7]. However, unlike single application scenarios, ML inference pipelines are complex and have many dependencies. For instance, replacing an ML model with a smaller one in case of failures may highly reduce the accuracy

as there are dependencies on the representations generated by each model. In addition, naively placing a part of the pipeline at a remote edge will affect the response time of all applications that rely on this component.

To address these challenges, this paper focuses on ensuring the resilience of distributed ML inference pipelines through a set of complementary approaches. First, ML pipelines often utilize multiple sensing modalities (e.g., audio and video) at various vantage points, allowing a task to be performed using diverse and redundant data resources. These multi-modal and multi-vantage sensors can enhance the reliability and resilience of ML pipelines against failures or disruptions at any point. Second, decoupling the interdependency among modules, for example, sharing intermediate representation with different sensors, further improves the pipeline's structural resilience. Third, models in ML pipelines can utilize the trade-offs between accuracy and resource usage, adjusting the end-to-end pipeline phases based on resource availability. Last, ML pipelines are deployed on multiple resources that can be tailored to adapt to dynamic environments by adjusting the input rates, locations, and priorities of different phases to ensure maximum performance.

In light of these considerations, this paper illustrates multiple strategies for ML inference pipelines, presents preliminary results using model-based simulations, and outlines the challenges and future directions. In summary, our contributions are threefold:

1) We illustrate possible mechanisms for resilient ML inference pipelines (Section III).
2) We present our preliminary results for resilient ML inference pipelines (Section IV).
3) We discuss the challenges associated with deploying such techniques and outline future directions (Section V).

II. Background

This section presents the background on edge ML inference, distributed pipelines, and various strategies for edge resiliency.

A. Edge Inference

Edge Computing brings cloud-like computational and storage resources to the edge of the network and provides users with low-latency applications [8]. Many ML-based applications are placed at the edge for the benefits of real-time interaction and data privacy, such as streaming video analytics and autonomous driving, known as *Edge Inference* [9]. The overall edge inference procedure can be described as follows: a sensor node sends input data x_i to a specific ML inference application, and the application responds with output data y_i, where $y_i = f(x_i)$. The function $f()$ may represent a single model inference or a pipeline of different ML models typically placed to minimize latency [10], [11].

The development of edge inference faces several challenges, one of which is the computational gap between computation-intensive machine learning algorithms/models and resource-constrained capable edge networks. The other challenges include stringent latency requirements, scaling ML applications across distributed edge devices, and adapting to dynamic environments. To bridge these gaps, ML inference pipelines are designed to efficiently and adaptively perform intelligence on resource-constrained edge devices. These adaptive pipelines facilitate the efficient execution of ML tasks by optimizing models and workflows for the unique constraints of edge environments.

B. ML Inference Pipelines

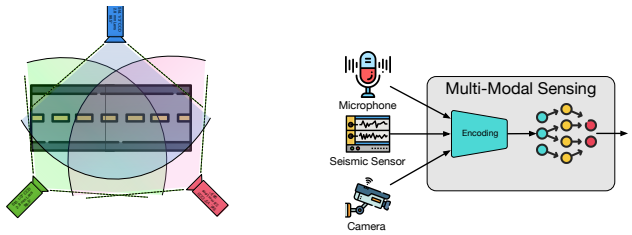
ML inference pipelines integrate multiple machine learning models and data transformations to address more complex tasks [12], [13]. For instance, an autonomous driving service combines object detection models for detecting objects with classification models to identify the objects. Typically, a distributed inference pipeline can be represented as a directed acyclic graph (DAG), where each vertex corresponds to a model or a data transformation, and edges represent dataflow between vertices. In executing an inference pipeline for serving a request, the ML model for one task generates intermediate outputs that serve as inputs for the ML model in the subsequent tasks, and the last task outputs the response to the request. The response time of a request is the summation of response times of all tasks in the path for producing the response. Fig. 1 shows a representative inference pipeline. The video monitoring pipeline uses object detection models to identify vehicles and people and perform subsequent analysis, including vehicle and person identification [13].

ML inference pipelines simplify model development by allowing model developers to reuse models that have been pre-trained on large benchmark datasets. In many cases, a single model (e.g., Resnet50 [14]) can be re-used as a feature function for a wide range of prediction tasks. Additionally, models and data transformation in an inference pipeline can be developed, optimized, and configured independently. If one model fails or a model's performance degrades, the pipeline can be configured to raise alerts or take recovery actions to enhance the pipeline's resiliency.

C. Edge Resiliency

Resiliency is crucial in edge computing because edge resources are often geographically distributed and resource-constrained, and failures can occur more frequently compared to cloud data centers. To prevent performance degradation and service outages, many strategies for fault prevention and recovery have been studied.

Fault prevention refers to strategies implemented to avoid faults and failures before they occur. This proactive approach aims to enhance edge systems' overall reliability and stability by reducing the likelihood of disruptions or malfunctions. One fault prevention strategy is redundancy and replications, such as deploying hot standby (where backup systems are always ready to take over) and cold standby (where backups



(a) Multi Vantage Systems (b) Multi Modality Systems

Fig. 2: Sensing redundancy approaches.

are activated only after a failure) across different edge nodes. Another strategy is load balancing, which distributes workload across multiple edge nodes to avoid a single point of failure.

When a failure or disruption occurs, recovery strategies are used to restore system functionality and minimize downtime. Unlike fault prevention, which aims to avoid failures, recovery strategies focus on responding to and recovering from faults to maintain service continuity. One of the key recovery strategies is automated failover, where the hot standby can instantly take over to minimize downtime, and the cold standby can be quickly powered on and configured to take over [6]. The cold standby may involve some downtime but is more cost-effective and resource-efficient. The second strategy is dynamic load redistribution, which redistributes workloads across available nodes or reallocates resources across different workloads. The third strategy is graceful degradation, which allows the system to continue operating at a degraded performance during a failure [6]. For example, an application might conserve resources for critical functionality while disabling non-essential failures. In addition to the above strategies, many other recovery techniques exist, such as system rebooting, service restarting, checkpointing, and rollback [15].

III. Methods

In this section, we explore a set of complementary approaches for enhancing the resiliency of distributed ML inference pipelines at the edge.

A. Sensing Redundancy

Sensor deployments often utilize redundancy to enhance the quality of inputs. For example, security cameras are placed to ensure maximum angular coverage, and no critical zones are left unobserved [16]. In addition to advantages in inputs' quality, sensing redundancy enhances resiliency by ensuring another sensor can fully or partially capture the same input. Fig 2 illustrates two approaches commonly used in sensing redundancy. The first approach, see Fig 2a, shows an example where cameras are replicated to cover the same scene from multiple angles. In this case, when one camera fails, others can still capture the scene. The second approach is multi-modal sensing, where nodes are equipped with multiple types of sensing, see Fig 2b. In this case, each node can utilize cameras and audio sensors to enhance the input diversity and quality and increase detection accuracy. For example, authors have shown that multi-modality can increase accuracy by up to 10% compared to a single-sensing modality [3].

B. Structural Resilience

Interdependencies among components of inference pipelines act as conduits for failure propagation. When a component fails, all those which depend on it may also fail. Reducing dependencies among components reduces how failures in one can propagate to others. Thus, it improves a notion of resilience called "structural resilience".

In the context of distributed analytics, different types of learning have implications for structural resilience. Multi-modal self-supervised learning, for example, creates an intermediate representation of input data into which all modalities encode their respective measurements. This intermediate representation is further suitable for many downstream tasks, essentially decoupling sensing from inference, thus improving structural resilience. Specifically, different sensors can encode data into the same intermediate representation, allowing the downstream pipeline to remain the same even as some sensors (or even modalities) fail and are replaced by others. Similarly, downstream analytics tasks can be changed without impacting the sensing and sensor data encoders. No such decoupling exists in supervised learning, where a monolithic neural network directly links sensors to inferences. An interesting topic, therefore, is exploring the interplay between learning/inference solutions and structural resilience.

C. Failover Replication

Replication is a key strategy incorporating failover mechanisms, such as hot and cold standby, to enhance system resilience. For improving the resiliency of single models, several failover strategies have been proposed [6]. This section explores the failover strategies for ML inference pipelines, which consist of multiple ML models.

Providing backups for ML models is often challenging because of resource limitations in edge computing. However, interdependencies in ML inference tasks highly complicate the problem. For instance, the resilient execution of the inference pipeline requires guaranteeing the resiliency of all components while meeting the end-to-end latency requirements. In addition, the failover model must follow the exact representation of inputs and outputs. Therefore, the key question for replication strategies in inference pipelines is: *How to ensure resilient execution of ML inference pipelines, while meeting the resource, latency constraints, and ensuring compatibility?*

Fig. 3 illustrates heterogeneous replication for inference pipelines. To implement end-to-end failover replication, we list two complementary solutions.[1] First, users can leverage the trade-offs in accuracy, resource usage, and inference time to deploy smaller replicas. Thus, instead of doubling the pipeline's entire resource requirements, the system can

[1]We address the challenges of partial replication in Fig. V.

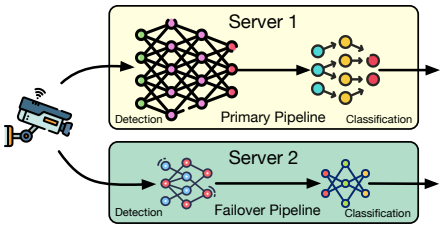


Fig. 3: Heterogeneous Replication of ML Inference Pipelines at the edge.

utilize a fraction of the resource and provide complete but less accurate models. Second, instead of deploying backup models with fine-grained prediction results, the edge system can provision a smaller model that delivers coarse-grained results and consumes fewer resources. For instance, instead of deploying a pipeline that utilizes an image classification model with multiple classes (e.g., ImageNet has 1000 classes), the pipeline can provide only a handful of classes that portray key classes (e.g., CIFAR's 10 or 100 classes). In this case, as the resulting complexity decreases, users can utilize compressed images, which increases the execution speed and decreases the scarce network and memory resource requirements.

In addition to methods that provide *hot* failover replicas for the entire pipeline, edge systems can also utilize a mixture of hot and cold standby replicas. One way to implement such techniques is to prioritize stages based on their request rate or order in the pipelines. In this case, the system can utilize the execution time of the early stages to load later stages, which highly decreases waiting time. For instance, in the pipeline shown in Fig. 1, if the execution time of the vehicle detector is longer than the activation time of the vehicle classifier's cold standby, the system can provision hot standby for the vehicle detector and cold standby for the vehicle classifier.

D. Pipeline Reconfiguration

ML inference pipelines are highly configurable to adapt to changes in edge networks. When an edge server becomes unavailable, consequently disrupting its hosted services, the pipeline can be reconfigured to ensure service availability. This reconfiguration may involve reallocating the workload among the active instances through load balancing, adjusting arrival rates, or selecting smaller models for inference [17], [6]. However, these changes always come with trade-offs between latency, throughput, accuracy, and availability. In particular, stages within a pipeline are interdependent, indicating that a reconfiguration in one part may impact the performance of subsequent tasks. Therefore, it is crucial to predict the performance of the entire pipeline to optimize outcomes.

To address this challenge, we analytically model ML inference pipelines using a tandem queue network to estimate the end-to-end response times of different configurations. An ML inference pipeline with N stages is represented as a network of N queueing models, where each queue captures the processing of one of the stages. We model each stage using an $M/M/1$ queue, where the first M denotes that arrivals follow a Poisson process with rate λ, the second M indicates the service times are exponentially distributed with service rate μ, and 1 represents that each stage runs on one edge server [10].

The expected end-to-end response time per application depends on the path it takes within the pipeline. A path $p \in P$ contains a set of stages, denoted as V^p. The end-to-end response time of a path p, denoted as $E[R^p]$ is:

$$E[R^p] = \sum_{v \in V^p} E[R_v^p] \tag{1}$$

where $E[R_v^p]$ is captured using the closed form equation for $M/M/1$ systems, $E[R_v^p] = \frac{1}{\mu_v - \lambda^p}$, where μ_v is the service rate, and λ^p is the request rate of the path, which is fixed for steady state tandem queues [18]. Note that when stages are shared between pipelines, the arrival rate at each stage, denoted as λ_v, is the sum of the arrival rates of each path.

IV. Preliminary Results

In this section, we present the preliminary results of applying pipeline reconfiguration to improve the resiliency of the distributed ML inference pipeline.

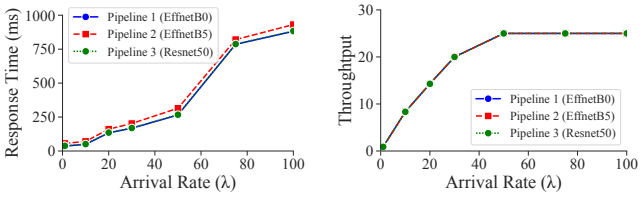
A. Experimental Setup

To understand the effect of pipeline reconfiguration, we use SimpleKit [19], an open-source[2] discrete event scheduling engine written in Python that implements the Event Graph modeling paradigm [20]. We implement N tandem $M/M/1$ queuing models, where requests propagate from the $i-1$-th to the i-th stage. The pipeline is a 2-stage ML inference pipeline: the first stage implements object detection using Yolov4, and the second stage implements image classification using EfficientnetB0, Resnet50, and EfficientnetB5, where EfficientnetB0 has the lowest accuracy and EfficientnetB5 the highest. We profile these models on three different GPUs: Nvidia A2, Orin Nano, and GeForce GTX 1080, to get their service times. Requests arrive following an exponential distribution. Each request starts service either at its own arrival time or at the departure time of the previous request, whichever is later. A request departs after t_d time units, where t_d is an exponentially distributed service time with a mean based on the model profiling results. The total time a request spends in the system is the sum of the waiting time (the difference between the service start and arrival) and the service time. Finally, we evaluate the performance by emulating 1000 requests.

B. Results

We present the results of two types of reconfiguration–model and resources–by analyzing their impact on the pipeline's end-to-end response time and throughput.

[2]https://github.com/PaulSanchez/SimpleKit-Python



(a) Response time (b) Throughput

Fig. 4: End-to-end response times and throughput of three different pipelines.

Effects of Model Reconfiguration. We start by evaluating the effect of model sizes, where we execute three different pipelines with different model combinations that result in the same types of outputs. The first stage of the pipeline runs a Yolov4 DNN, followed by either: (1) EfficientNetB0, (2) EfficientNetB5, or (3) Resnet50 DNNs. Stages run on the device type, and although we have results for all GPUs, we focus on the Nvidia A2 for our analysis. Fig. 4a and Fig. 4b demonstrate the response time of the three pipelines under different request rates, along with the throughput. The differences in end-to-end response times (Fig. 4a) can be seen when more complex models such as EfficientNetB5 are selected, which consumes twice the amount of memory resources as EfficientNetB0. Although EfficientNetB5 can produce higher quality results (in terms of accuracy), even when compared to Resnet50, it results in higher average end-to-end response times. In addition, Fig. 4b shows no differences in the average end-to-end throughput, demonstrating an important opportunity for model selection. Although Resnet50 consumes as much as 1.5× memory resources as EfficientNetB0, Fig. 4b shows all pipelines sustain the same throughput under the same arrival rates while resulting in similar quality of results.

Key Takeaway. *Model selection plays a critical role in optimizing ML inference pipelines. By carefully choosing models, it is possible to maintain similar levels of workload rate, end-to-end response time, and quality of output results, all while significantly reducing resource utilization that provides higher availability and resilience for ML pipelines.*

Effects of Resource Reconfiguration. To analyze the effects of resource reconfiguration, Fig. 5 displays the response time and throughput of different pipelines executing over different edge servers with homogeneous and heterogeneous resources. Similar to the previous experiments, we first execute model inference through the three pipelines on the same types of resources (Fig. 5a): both stages are placed on an Nvidia A2, Orin Nano, and a GTX-1080. As can be seen, since the 1080 is the most performant device, it sustains a low end-to-end response time of up to 250ms, even at high request rates. Likewise, Fig. 5b showcases similar heterogeneity effects on the end-to-end response time, mainly due to 1080's higher performance, irrespective of the device allocated for the second pipeline stage. When the first stages of pipelines are allocated with either an A2 or an Orin, the response time increases by 1.4× on average. This is because both the A2 and the Orin ability to sustain load is reduced due to their lower performance, compared to the 1080. This results in increases in the average queuing times, which then decreases the throughput in the first stages. Along with Figures 5c and 5d, these results demonstrate an important tradeoff when selecting and allocating resources to stages of a pipeline, whereas larger devices are needed to sustain loads beyond 30 requests arriving per-second. Importantly, although the total amount of resources between any two resources are the same, the resource selection for a stage can highly affect both the end-to-end response times and pipeline throughput.

Key Takeaway. *Resource reconfiguration has a greater impact on response time and throughput than model reconfiguration. Even when total resource utilization across different device combinations is similar, resource selection affects the end-to-end response times and the pipeline's capacity to handle the load by up to 1.5×. By leveraging queuing models, resource selection in pipelines can be optimized and enhance end-to-end response time and throughput.*

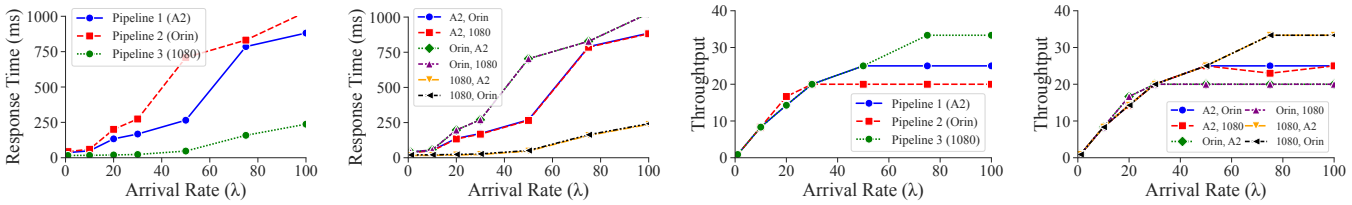
V. Challenges

In this section, we explore the challenges for enhancing the resiliency of distributed ML inference pipelines.

A. Representations Compatibility

Despite the numerous advantages of multi-vantage and multi-modality sensing to recover from hardware and byzantine failures where sensor input can be fully or partially recovered using another sensor or another modality, missing sensor information can introduce compatibility issues, where ML models often rely on a fixed number of inputs to produce the results. Thus, the input size and shape will change when a model fails, requiring further adjustments. Similarly, failover replication approaches that utilize heterogeneous backups introduce challenges where the latent space representations for models of different sizes are incompatible.

To address the representation compatibility challenges, we list three possible approaches. First, users can utilize multiple versions of the pipelines based on input data. For example, an inference pipeline that utilizes seismic and acoustic sensors can have three variants that use both or only one of the input sensors. However, this method faces many scalability challenges as the number of variations grows exponentially with the number of sensors. Second, pipelines can utilize adaptive learning techniques, similar to slimmable neural networks [21], that cope with missing or incomplete inputs. Nevertheless, this method faces learning challenges, where models are trained for much longer and do not guarantee better inputs result in higher accuracy. Third, users can rely on a unified latent space representation where different sensors' or models' outputs utilize the same representation space [22], [23]. Despite the advantage of having a unified latent space, this technique requires huge amounts of data.



(a) Response time (Homo.) (b) Response time (Hetero.) (c) Throughput (Homo.) (d) Throughput (Hetero.)

Fig. 5: End-to-end response times and throughput on different edge servers.

B. Reconfiguration Challenges

In addition to challenges in traditional pipelines [24], addressing the resiliency challenges for multiple co-located pipelines introduces further complications. Determining a failover execution plan compatible with accuracy and latency objectives is complex for many reasons. First, a single node failure might introduce a cascading effect where other pipelines that share some or all servers are affected. Second, although controlling different phases' input rates and priorities is beneficial, they highly expand the search space. To address the placement challenges, we envision a modeling and placement method that utilizes reinforcement learning approaches to place the ML models in failure recovery scenarios.

VI. Conclusions and Future Directions

This paper presents four complementary approaches from different perspectives – sensing redundancy, reducing module inter-dependencies, model replications, and pipeline reconfigurations – to enhance the resiliency of ML inference pipelines. In addition to the benefits and challenges of each approach, integrating these approaches introduces additional failure resilience benefits. One interesting future direction is to design an edge system that integrates these approaches and determines the optimal recovery actions, balancing trade-offs between accuracy, latency, resource utilization, and energy consumption.

Acknowledgement

We thank the MILCOM reviewers for their valuable comments, which improved the quality of this paper. This research is supported by NSF grants 2211302, 2211888, 2213636, 2105494, 23091241, US Army contract W911NF-17-2-0196, and Adobe.

References

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

[2] Q. Liang *et al.*, "Dělen: Enabling flexible and adaptive model-serving for multi-tenant edge ai," in *IoTDI'23*, 2023.

[3] J. Li *et al.*, "Acies-OS: A Content-Centric Platform for Edge AI Twinning and Orchestration," in *33rd International Conference on Computer Communications and Networks (ICCCN)*, 2024.

[4] A. Aral and I. Brandić, "Learning spatiotemporal failure dependencies for resilient edge computing services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1578–1590, 2021.

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

[6] W. A. Hanafy *et al.*, "Failure-Resilient ML Inference at the Edge through Graceful Service Degradation," in *IEEE Military Communications Conference (MILCOM)*, pp. 144–149, 2023.

[7] W. A. Hanafy *et al.*, "Understanding the Benefits of Hardware-Accelerated Communication in Model-Serving Applications," in *IWQoS'23*, 2023.

[8] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[9] Z. Zhou *et al.*, "Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[10] Q. Liang *et al.*, "Model-Driven Cluster Resource Management for AI Workloads in Edge Clouds," *ACM Trans. Auton. Adapt. Syst.*, vol. 18, mar 2023.

[11] J. Soifer *et al.*, "Deep Learning Inference Service at Microsoft," in *2019 USENIX Conference on Operational Machine Learning (OpML 19)*, pp. 15–17, may 2019.

[12] A. Souza *et al.*, "Unlocking efficiency: Understanding end-to-end performance in distributed analytics pipelines," in *IEEE Military Communications Conference (MILCOM)*, pp. 150–155, IEEE, 2023.

[13] D. Crankshaw *et al.*, "Inferline: latency-aware provisioning and scaling for prediction serving pipelines," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, pp. 477–491, 2020.

[14] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[15] L. Wu *et al.*, "Microras: Automatic recovery in the absence of historical failure data for microservice systems," in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, pp. 227–236, IEEE, 2020.

[16] E. Yildiz *et al.*, "Optimal Camera Placement for Providing Angular Coverage in Wireless Video Sensor Networks," *IEEE Transactions on Computers*, vol. 63, no. 7, pp. 1812–1825, 2014.

[17] W. A. Hanafy *et al.*, "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.

[18] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. USA: Cambridge University Press, 1st ed., 2013.

[19] P. J. Sanchez, "Fundamentals of simulation modeling," in *2007 Winter Simulation Conference*, pp. 54–62, IEEE, 2007.

[20] L. Schruben, "Simulation modeling with event graphs," *Communications of the ACM*, vol. 26, no. 11, pp. 957–963, 1983.

[21] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable Neural Networks," 2018.

[22] S. Liu *et al.*, "FOCAL: Contrastive Learning for Multimodal Time-Series Sensing Signals in Factorized Orthogonal Latent Space," in *Advances in Neural Information Processing Systems*, vol. 36, pp. 47309–47338, 2023.

[23] D. Kara *et al.*, "Freqmae: Frequency-aware masked autoencoder for multi-modal iot sensing," in *Proceedings of the ACM Web Conference 2024*, WWW '24, p. 2795–2806, 2024.

[24] S. Ahmad *et al.*, "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.