Short paper: HYDRA: Multi-Level Ensemble Learning for Failure-Resilient Edge Inference

Krishna Praneet Gudipaty[1], Walid A. Hanafy[1], Nathan Ng[1], Kaan Ozkara[2], Qianlin Liang[2], Jesse Milzman[3], David Irwin[1], Suhas Diggavi[4], Prashant Shenoy[1]

[1]University of Massachusetts Amherst, [2]Amazon, [3]DEVCOM Army Research Laboratory, [4]University of California Los Angeles

***Abstract*—Edge inference has become increasingly common for latency-sensitive AI services. However, edge environments are more susceptible to failures than cloud environments. Conventional failure-resilience approaches for edge environments, such as cloud failover, compressed backups, or split inference, often compromise latency, accuracy, or availability, limiting their effectiveness for critical edge inference services. In this paper, we propose HYDRA, a failure-resilient edge inference system that pools the unused capacity of multiple edge nodes to serve as a collective backup for a primary model. Our key insight is that an ensemble of small models, each running on a separate inference node, can collectively match the performance of a large primary model while naturally providing redundancy for failure resilience. HYDRA is built on a multi-level ensemble learning framework that jointly trains a set of lightweight upstream models and a downstream model under a multi-objective loss, producing ensemble members that perform competently in isolation under failure and refine each other's predictions when combined. Given a target resource budget and failure model, HYDRA selects a Pareto-optimal ensemble configuration, deploys it as a failover replica alongside the primary model, activates it upon failure detection, and reconfigures the aggregator to operate over the surviving members as additional nodes fail. Empirical evaluations across vision and audio datasets show that ensembles produced by HYDRA achieve performance comparable to the original models while providing strong fault tolerance and deployment flexibility. In particular, ensembles sized at 40% of the original model achieve accuracy comparable to the primary model, while preserving 95.6% of ensemble accuracy under failures. Compared to state-of-the-art failover approaches, HYDRA reduces response time by up to $1.61\times$ while matching the accuracy of the original primary model.**

***Index Terms*—Edge Computing, Failure Resilience, Resource-Constrained, Ensemble Learning, Model Serving**

I. Introduction

The advent of edge accelerators such as on-device GPUs and TPUs [1]–[3] has enabled critical latency- and privacy-sensitive model-serving workloads in domains such as driving [4], augmented reality [5], IoT analytics [6], and urban robotics [7]–[10]. However, unlike cloud data centers, edge nodes are often deployed in cell towers, roadside cabinets, and residential homes, where they are exposed to harsh weather, power instability, and intermittent connectivity [11], [12]. The resulting failure rates are substantially higher than in cloud environments, elevating failure resilience to a first-order design constraint for edge inference services [13]–[15].

The conventional cloud approach to failure-resilient inference designates a subset of servers (and their GPUs) as standby nodes that take over when primary servers fail. However, this strategy is infeasible at the edge due to limited resource capacity and the absence of idle servers to reserve as standby nodes. One alternative in this setting is to fail over to cloud servers when an edge server fails, but doing so violates the low-latency requirements of applications [15] that edge inference was deployed to avoid. A straightforward approach is to *vertically* split the model and run it across multiple edge servers [16], preserving the accuracy of the original model. However, this approach has a fundamental limitation. Since inference executes sequentially across servers, each additional node in the pipeline introduces a new point of failure and can make the backup *less* reliable than the single-node approach it is meant to improve upon. Another alternative is to place smaller compressed variants of the original on edge nodes as failover replicas [15], [17]. While this eliminates the need for fully idle standby servers, compressed backups can substantially degrade inference accuracy under failure. This motivates the research question of our work: *Can the unused capacity across multiple edge nodes be pooled to provide failure resilience without sacrificing inference accuracy?*

To address these limitations, we present HYDRA, a failure-resilient edge inference system whose core mechanism is a multi-level ensemble of small models, trained to operate both collaboratively and independently across edge nodes. At deployment, HYDRA places multiple upstream models on separate edge servers, alongside a downstream model that aggregates their intermediate representations to produce a refined prediction (see Figure 1). Distributing inference across an ensemble of lightweight models enables HYDRA to (i) preserve accuracy beyond what a single compressed model can provide while utilizing unused capacity on existing servers, (ii) avoid the sequential network overhead of pipeline backups through parallel execution, and (iii) confine server failures to a subset of the ensemble rather than propagating them across the inference pipeline.

That said, not every ensemble design is suited for failure resilience. Existing approaches, such as mixture-of-experts (MoE) architectures [18], train specialist sub-networks that may perform poorly in isolation, which can significantly degrade inference accuracy under failures. HYDRA instead requires each ensemble member to be a competent *standalone* learner, ensuring that the failure of any subset of servers does not render the remaining models ineffective. At the same time, these models must be sufficiently *diverse* to refine each other's

results and improve accuracy when combined.

To meet both requirements, HYDRA utilizes a multi-objective training loss that simultaneously promotes (i) individual model performance and (ii) diversity among ensemble members to improve ensemble accuracy. Moreover, because resource availability and failure conditions are unknown at training time, HYDRA trains a family of ensemble configurations and selects the Pareto-optimal configuration that maximizes accuracy and resilience within the available resource budget. At runtime, the selected ensemble is deployed as a failover replica alongside the primary model, with failure detection approaches [14], [19] triggering failover as needed.

Contributions. In designing, implementing, and evaluating HYDRA, our paper makes the following contributions:

- **Multi-Level Ensemble Design.** We propose a multi-level ensemble learning approach for failure-resilient edge AI that comprises upstream models and a downstream model, jointly trained with a multi-objective loss that promotes individual performance and diversity among ensemble members, thereby improving accuracy when combined.
- **HYDRA System Design.** We propose HYDRA, a failure-resilient edge inference system based on the above learning approach. HYDRA constructs an ensemble and selects a Pareto-optimal ensemble configuration that balances accuracy and resilience within available resources, and deploys the selected ensemble as a failover replica upon failure detection.
- **Ensemble Evaluation.** We extensively evaluate ensembles produced by HYDRA across vision and audio datasets, demonstrating performance across different model sizes and resource budgets. Our results show that our ensemble model, sized at 40% of the original model, achieves accuracy comparable to the primary model, while preserving 95.6% of ensemble accuracy under multiple failures.
- **HYDRA Evaluation.** We evaluate the runtime performance of HYDRA in real-world edge environments. Our results show that, compared to two state-of-the-art failover approaches, HYDRA achieves up to 9.55% higher accuracy under tight resource budgets and up to $1.61\times$ lower latency under bandwidth-constrained networks, while matching the accuracy of the original primary model.

II. Background

In this section, we provide background on edge inference and its failure modes, review traditional replication strategies, and discuss their limitations in achieving resilience at the edge.

A. Edge Inference and Failure Modes

Edge computing is a critical complement to cloud-based AI services, as modern deep learning models often exceed the resource and latency capabilities of on-device hardware [7], [8], [10]. In edge inference, sensor or client inputs are processed by models deployed on edge-serving systems such as NVIDIA Triton [20] and Kubeflow [21]. While edge platforms offer low latency and privacy benefits, they differ from tightly controlled cloud data centers that host powerful hardware and operate with a high degree of hardware redundancy. Edge nodes are typically deployed in physically exposed, resource-constrained settings, such as cell towers, roadside infrastructure, and industrial facilities, with limited compute, memory, and power budgets [11], [15], [17]. As a result, they are susceptible to multiple classes of failures including *crash*, *transient*, and *Byzantine* failures [14], [15], [17], [22]. In this work, we focus on crash and transient failures, such as hardware faults and network disruptions, that render an edge deployment temporarily or permanently unavailable.

B. Traditional Replication-Based Resilience

Replication builds resilient systems by backing components with failover replicas. In model serving, a model can have a hot (active-active), warm (active-passive), or cold backup. For example, DLIS [13] uses a hot backup that issues each request twice and cancels the redundant execution upon receiving a response. Warm backups preload models into memory and become active upon failure, while cold backups are loaded from disk only after a failure. While these strategies improve availability, they increase resource consumption: hot backups effectively double the inference load and can degrade response time. More importantly, all configurations require dedicated standby capacity, which is common in cloud data centers but rarely available at the edge, where idle compute is scarce.

C. Failure-Resilient Inference at the Edge

In edge environments, severe resource constraints render traditional failover replication infeasible. Prior work has explored three alternatives: (i) failing over to a remote cloud server, which removes local resource constraints but negates edge-inference latency benefits [15], (ii) using compressed replicas, which reduces resource usage at the cost of accuracy [15], [17], and (iii) split inference, which preserves accuracy but introduces inter-node latency and additional failure points across its sequential chain [16], [23]. These limitations highlight the need for an approach that achieves high accuracy, low latency, and strong failure resilience.

III. Multi-Level Ensemble Learning

This section presents the multi-level ensemble learning formulation and the core design of the failover replicas that underlies HYDRA.

A. Ensemble Model Architecture

A HYDRA ensemble $\{h_S\}$ comprises two classes of models, illustrated in Figure 1. *Upstream models* are deployed on separate edge servers, and each produces an independent prediction from the input through an attached exit head. *Downstream models* consume the intermediate representations of two or more upstream models and combine them into a refined aggregate prediction. Note that HYDRA trains one downstream model per surviving upstream subset of size two or more, so that an aggregator is available for each failure combination. Unlike a single monolithic backup, this two-level structure supports multiple failure modes. First, when all servers are available, every upstream model forwards its intermediate

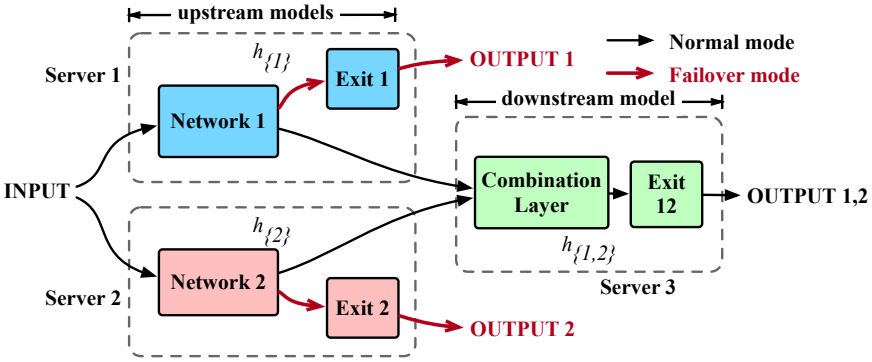


Fig. 1: Multi-level ensemble learning framework showing the case with two upstreams and one downstream on separate servers. Primary model is not shown.

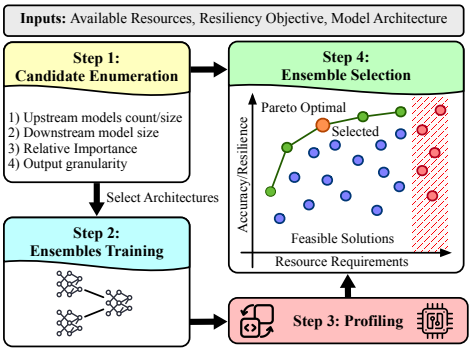


Fig. 2: The HYDRA framework for designing and training an ensemble model.

representation to the downstream model, which produces a high-accuracy aggregate prediction. Second, when one or more upstream servers fail but a downstream server remains operational, the surviving upstream models continue to operate. Third, when no downstream server is available, each surviving upstream falls back to its exit head and produces a standalone prediction. This design not only yields improved accuracy, as we show later, but also enables graceful degradation across multiple failure scenarios.

B. Problem Formulation

To formalize the multi-level ensemble learning problem, let $\boldsymbol{x}$ be inputs to the learning model and y be the labels associated with it in the supervised learning framework. We will consider learning models under a loss function $\ell(h(\boldsymbol{x}), y)$, where $h(\boldsymbol{x})$ is the prediction by model h for input $\boldsymbol{x}$. Conventionally, the performance is measured by the population risk as $L(h) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{P}}[\ell(h(\boldsymbol{x}), y)]$, for an unknown underlying distribution $\mathcal{P}$, and the model h is optimized over a hypothesis class $h \in \mathcal{H}$.

In our problem, in addition to training a primary model h_P using any conventional method, we aim to simultaneously train multiple smaller ("backup") models $\{h_{\mathcal{S}}\}$, indexed by $\mathcal{S} \subseteq \mathcal{M} = \{1, \ldots, M\}$, ensuring that each model maintains strong standalone performance while collaboratively improving accuracy through the downstream aggregation model. That is, we would like to design $\{h_{\mathcal{S}}\}$ such that the population risks $L(h_{\mathcal{S}}) \leq \gamma_{\mathcal{S}}$, where we also want $\gamma_{\mathcal{S}} \leq \gamma_{\mathcal{S}'}$ if $\mathcal{S}' \subset \mathcal{S}$, *i.e.,* performance improves monotonically as more backup servers are available. To make this concrete, as illustrated in Figure 1, consider two backup servers with individual models $h_{\{1\}}, h_{\{2\}}$. When both are available, they feed into a third server that builds $h_{\{1,2\}}$, which processes the intermediate representations of $h_{\{1\}}$ and $h_{\{2\}}$ to produce the final output. The goal is to simultaneously train $h_{\{1\}}, h_{\{2\}}, h_{\{1,2\}}$ for the desired individual performances. We will use the terminology *upstream* model for the models that feed into later *downstream* models; *e.g.,* $h_{\{1\}}, h_{\{2\}}$ are upstream models feeding into the downstream model $h_{\{1,2\}}$. Clearly, this leads to several natural questions (i) how do we train the models to satisfy these requirements? (ii) how do we compose the models $\{h_{\mathcal{S}}\}$, *i.e.,* what is exchanged between the upstream and downstream models? (iii) how do we choose the individual model classes $\mathcal{H}_{\mathcal{S}}$, *i.e.,* design the model architecture?

C. Training Objective

To simultaneously find the models $\{h_{\mathcal{S}}\}$, answering question (i), we formulate this problem as a multi-objective optimization. For example, we can formulate

$$\min_{\{h_{\mathcal{S}}\in\mathcal{H}_{\mathcal{S}}\}} L(h_{\mathcal{M}}) \text{ such that } L(h_{\mathcal{S}}) \leq \gamma_{\mathcal{S}}, \mathcal{S} \subseteq \mathcal{M} = \{1, \ldots, M\} \tag{1}$$

Following standard practice, we replace the population risks with their empirical counterparts and formulate the problem as a Lagrangian:

$$\mathcal{L} = \sum_{\mathcal{S}\subseteq\mathcal{M}} \lambda_{\mathcal{S}} \hat{L}(h_{\mathcal{S}}), \tag{2}$$

where $\{\lambda_{\mathcal{S}}\}$ define the Lagrangian weights and we have defined the empirical risks over a labeled dataset $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$ as

$$\hat{L}(h_{\mathcal{S}}) = \frac{1}{N} \sum_{i=1}^{N} \ell(h_{\mathcal{S}}(\boldsymbol{x}_i^{(\mathcal{S})}), y_i^{(\mathcal{S})}). \tag{3}$$

This formulation allows us to "process" the dataset $\mathcal{D}$ to produce task-specific datasets $\mathcal{D}^{(\mathcal{S})} = \{(\boldsymbol{x}_1^{(\mathcal{S})}, y_1^{(\mathcal{S})}), \ldots, (\boldsymbol{x}_N^{(\mathcal{S})}, y_N^{(\mathcal{S})})\}$. For example, the processing we investigate in this paper is to "coarsify" the labels to produce $\{y_i^{(\mathcal{S})}\}$, *i.e.,* one can map the 100 CIFAR-100 fine-grained classes to their 20 coarser superclass labels. Therefore, our overall optimization problem, which simultaneously finds the models $\{h_{\mathcal{S}}\}$ in (2), becomes

$$\min_{\{h_{\mathcal{S}}\in\mathcal{H}_{\mathcal{S}}\}} \sum_{\mathcal{S}\subseteq\mathcal{M}} \lambda_{\mathcal{S}} \hat{L}(h_{\mathcal{S}}). \tag{4}$$

The coefficients $\{\lambda_{\mathcal{S}}\}$ capture the relative importance of each level of this ensemble learning problem. Consider, for concreteness, $\mathcal{M} = \{1, 2\}$, which reduces the problem to $\min_{h_{\{1\}},h_{\{2\}},h_{\{1,2\}}} \{\lambda_1 \hat{L}(h_{\{1\}}) + \lambda_2 \hat{L}(h_{\{2\}}) + \lambda_{12} \hat{L}(h_{\{1,2\}})\}$. Increasing λ_{12} relative to λ_1, λ_2 encourages the individual models $h_{\{1\}}, h_{\{2\}}$ to be diverse so that they can be mutually refined to produce better performance for the downstream model $h_{\{1,2\}}$.

IV. HYDRA System Design

To design the model ensembles, *i.e.,* answering questions (ii) and (iii), HYDRA bases the architecture of its upstream models on the concept of a *neural block*, or *block* for short, a common abstraction in modern neural network design [24], [25]. While HYDRA supports any model architecture, for simplicity, we construct each upstream model as a prefix of the primary model's block sequence, limiting the design space to a discrete set of options. For example, the EfficientNet-B0 architecture comprises seven blocks, each with one or more convolution layers. Additionally, we attach an exit block (e.g., a classification head) to each upstream model; this exit block is trained jointly with the ensemble and activated only when the corresponding server operates in standalone failover mode. Further, unlike typical neural networks in which subsequent phases or blocks rely on a single intermediate representation, HYDRA aggregates intermediate representations from multiple complementary upstream sources to produce a refined aggregate prediction. Hence, our downstream model primarily comprises a combination layer and an output layer but can be extended to more complex architectures with additional layers.

Figure 2 summarizes the HYDRA design pipeline as four sequential steps:

Step 1: Candidate Enumeration. HYDRA begins by enumerating a set of candidate ensemble configurations, each characterized by the following design parameters:

- *Number and sizes of upstream models.* The number of upstream models in the ensemble and the prefix length each one inherits from the primary jointly determine the per-model parameter count and the failure tolerance of the ensemble. Increasing the number of upstream models raises training complexity but enables the system to tolerate additional simultaneous failures beyond the primary.
- *Downstream model size and architecture.* The downstream model can range from the minimal combination-and-output configuration to variants that apply additional transformations before the output layer. More expressive downstream architectures generally improve aggregate accuracy but consume more resources, and resource fragmentation across servers further constrains how the budget is partitioned between upstream and downstream models.
- *Relative importance* $\{\lambda_S\}$. The Lagrangian weights from the training objective (Section III-C) govern the trade-off between standalone competence and aggregation gain at training time. Higher weights on downstream subsets encourage diverse intermediate representations and stronger aggregate accuracy, while higher weights on individual upstream models promote independent learners at the cost of diminishing returns when combined.
- *Output granularity.* The ensemble can operate over coarser output label spaces, trading prediction granularity for improved accuracy under tight resource budgets. For example, a fine-grained multi-class classifier can be replaced with a coarser-grained classifier (e.g., classifying an image as an *animal* rather than distinguishing between *cat* and *dog*).

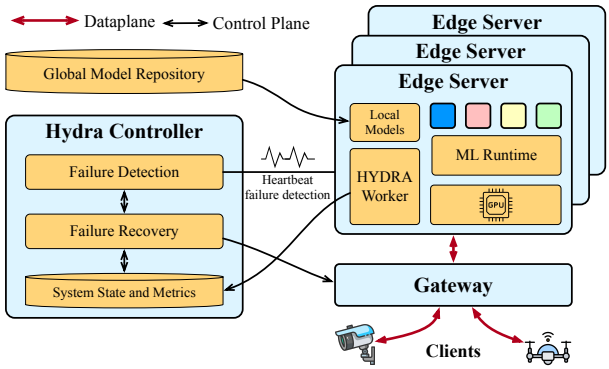


Fig. 3: Overview of HYDRA design.

Step 2: Ensemble Training. Each configuration is trained independently using the multi-objective loss formalized in Section III-C, producing one trained ensemble per candidate. Because deployment-time resource availability and failure conditions cannot be anticipated at training time, training a family rather than a single configuration is necessary to support deployment across heterogeneous edge clusters.

Step 3: Profiling. Each trained candidate is profiled on the runtime environment to empirically measure its accuracy and resource consumption (memory, compute, inference latency).

Step 4: Ensemble Selection. Profiled candidates are mapped onto the resource vs. accuracy/resilience plane, as illustrated in Figure 2. Configurations that exceed the deployment's resource budget fall into the infeasible region and are discarded. HYDRA selects the configuration on the Pareto frontier of feasible trade-offs that maximizes the resilience-accuracy objective for the deployment, subject to per-node capacity and resource fragmentation. The empirical effect of these design parameters is examined in Section VI.

Placement. HYDRA employs a best-fit heuristic that places each component on the feasible server with the smallest residual capacity, preferring assignments that distribute upstream models across distinct servers to preserve fault tolerance.

V. Implementation

We implement HYDRA as a distributed inference serving system, illustrated in Figure 3. HYDRA's implementation totals ∼3,000 SLOC and comprises three main components.

HYDRA workers. Each edge server runs a HYDRA worker that hosts one or more local models, including combinations of the primary, upstream, or downstream models, loaded from a *global model repository*. Models are trained in PyTorch v2.5.1 and executed by the worker's ML runtime, built using ONNX Runtime v1.21. Workers exchange serialized intermediate representations over a gRPC interface. Note that HYDRA can also work with other runtimes, such as TensorRT [26] or TVM [27].

Gateway and request lifecycle. Inference requests are routed based on the current system state. In normal mode, requests are sent to the primary worker. Upon failure, the gateway switches to failover mode, in which each upstream worker receives the request and forwards its intermediate representation to the downstream aggregator, whose output is returned to the

client. Even if a subset of upstream workers fails, requests are routed to the surviving workers, and their representations are aggregated by a downstream model. Lastly, when only one replica is available, the gateway falls back to a standalone model that receives, processes, and returns the response to the client.

HYDRA controller and failure detection. HYDRA's control plane supervises the deployment through *Failure detection*, *Recovery*, and *State & metrics* modules. Workers are probed every $T = 15$ ms and declared failed after two missed heartbeats, giving a worst-case detection latency of 30 ms. *Recovery* propagates failures to the gateway, which updates routing, while the *State & metrics* store tracks operational state and recovery telemetry.

VI. Evaluation

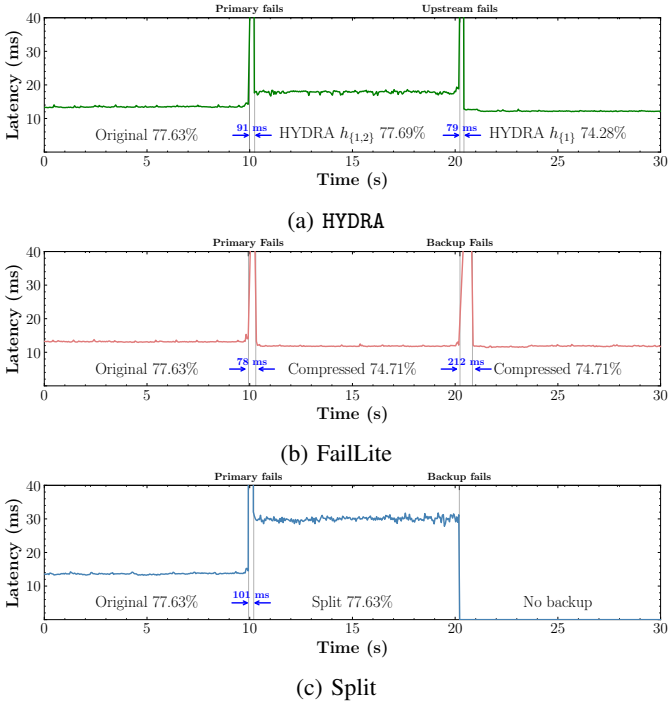
In this section, we first assess the impact of key design aspects from Section IV on accuracy and parameter efficiency across vision and audio workloads. Secondly, we evaluate HYDRA's runtime performance on a real edge testbed against several baselines.

A. Experimental Setup

Hardware and testbed. The runtime evaluation is conducted on a six-node edge testbed of NVIDIA Jetson Orin Nano 8 GB devices, each equipped with an integrated NVIDIA Ampere GPU (1024 CUDA cores, 32 tensor cores) and a six-core Arm Cortex-A78AE CPU. Devices are interconnected over Gigabit Ethernet, and we use the Linux Traffic Control (TC) subsystem to emulate different network bandwidths. For model training, we use the Unity Research Computing Platform, a SLURM cluster comprising of heterogeneous GPU nodes. Single-GPU training is sufficient for most workloads, except for Tiered-ImageNet classification models, which are trained on four GPU nodes. Training is implemented in PyTorch v2.5.1 using TIMM [28] and other open-source codebases.

Workloads. We evaluate HYDRA across four DNN architectures spanning two modalities and three structural classes: EfficientNet-B0 [25] and ResNet50 [24] as convolutional vision models, ViT-B/16 [29] as a vision transformer, and DeepSpeech2 [30] as a recurrent audio model. Table I summarizes the datasets, architectural classes, and block counts. We refer to these unmodified architectures as *original* models and compare them against their corresponding HYDRA ensembles. Following the prefix-block construction in Section IV, we denote an ensemble by *Architecture*($\mathcal{B}k$), where each upstream model comprises the first k blocks of the parent architecture; e.g., EfficientNet-B0($\mathcal{B}3$) uses the first three blocks.

Datasets. We use three datasets that span vision and audio inference workloads at varying scales of class cardinality and label hierarchy as shown in Table I. CIFAR-100 [31] comprises 60,000 images partitioned into 100 fine-grained classes that group into 20 superclasses, providing a compact benchmark with a two-level label hierarchy suitable for evaluating coarsified-output configurations. Tiered-ImageNet [32] is a 779,165-image subset of ImageNet-1k [33] organized into 608 classes and 34 high-level categories, providing a larger-scale vision benchmark with a comparable hierarchical structure. Speech Commands [34] consists of 65,000 one-second audio recordings sampled at 16 kHz, used for keyword classification and speech-recognition tasks.



(a) HYDRA

(b) FailLite

(c) Split

Fig. 4: Client-observed latency of EfficientNet-B0($\mathcal{B}5$) on CIFAR-100 under sequential worker failures. The primary model fails at $t = 10$ s and a second failure at $t = 20$ s removes one upstream worker.

Baselines. We compare HYDRA against two failure-resilience approaches representative of prior work in resource-constrained edge inference:

- *FailLite* [17]: a single compressed variant of the primary model deployed as a warm failover replica, to which the gateway reroutes inference requests upon primary failure.
- *Split inference* [16]: a vertical partition of the primary model across three edge nodes, executed sequentially with each node forwarding its representations to the next stage.

B. HYDRA in Action

To characterize the runtime behavior of HYDRA under failure, we deploy an EfficientNet-B0($\mathcal{B}5$) ensemble on the Jetson testbed and inject a sequence of two failures while a client continuously issues inference requests, as shown in Figure 4. The primary worker fails at $t = 10$ s, followed by a second failure at $t = 20$ s that removes one upstream worker. Figure 4a shows the client-observed latency and inference accuracy of HYDRA across the three operating regimes induced by these failures. We compare against two prior approaches, FailLite and split inference, each deployed as a failover replica that takes over upon primary failure. All three approaches are sized to fit the same per-server parameter headroom.

During the first 10 s, all three approaches route requests to the primary model, which serves them at 77.63% Top-

TABLE I: Datasets and DNN architectures used in the HYDRA evaluation. The table lists each architecture's class, number of blocks, whether the dataset supports hierarchical labels for coarsified-output ensembles, and the models used.

Task	Dataset	Description	Coarse Labels	Models
Image Classification	CIFAR-100 [31]	60K 32×32 images	True (20 super classes)	EfficientNet-B0 (7 CNN blocks), ResNet50 (4 CNN blocks), ViT-B/16 (12 Transformer blocks)
	Tiered-ImageNet [32]	779K (subset of ImageNet-1k [33])	True (34 super classes)	EfficientNet-B0 (7 CNN blocks)
Audio Classification, Speech Recognition	Speech Commands [34]	65K (1-sec) recordings at 16kHz	False	EfficientNet-B0 (7 CNN blocks) DeepSpeech2 (6 GRU blocks)

TABLE II: Top-1 Acc. and parameter count of HYDRA ensembles across DNN architectures and datasets, compared with the original models. $h_{\{1\}}$ and $h_{\{2\}}$ denote standalone upstream models, and $h_{\{1,2\}}$ the aggregate model. †WER; lower is better.

Dataset	DNN Model	HYDRA						FailLite-Small		Original	
		$h_{\{1\}}$		$h_{\{2\}}$		$h_{\{1,2\}}$					
		Perf.	Para.	Perf.	Para.	Perf.	Para.	Perf.	Para.	Perf.	Para.
CIFAR-100	EfficientNet-B0 ($\mathcal{B}1$)	0.1883	4.08K	0.2162	4.08K	0.2365	11.45K	0.2060	4.08K	0.7763	4.79M
	EfficientNet-B0 ($\mathcal{B}2$)	0.4070	21.59K	0.4362	21.59K	0.4634	48.08K	0.4268	21.59K		
	EfficientNet-B0 ($\mathcal{B}3$)	0.5824	0.07M	0.5885	0.07M	0.6261	0.15M	0.5809	0.07M		
	EfficientNet-B0 ($\mathcal{B}4$)	0.6785	0.32M	0.6804	0.32M	0.7169	0.65M	0.7105	0.32M		
	EfficientNet-B0 ($\mathcal{B}5$)	0.7428	0.88M	0.7371	0.88M	**0.7769**	1.79M	0.7471	0.88M		
	EfficientNet-B0 ($\mathcal{B}6$)	0.7529	2.90M	0.7548	2.90M	**0.7888**	5.83M	0.7686	2.90M		
	EfficientNet-B0 ($\mathcal{B}7$)	0.7308	3.60M	0.7567	3.60M	**0.7838**	7.3M	0.7763	3.60M		
	Resnet50 ($\mathcal{B}2$)	0.6523	1.40M	0.6488	1.40M	0.6820	3.30M	0.6801	1.40M	0.7593	23.93M
	Resnet50 ($\mathcal{B}3$)	0.7324	8.76M	0.7043	8.76M	0.7552	17.50M	0.7263	8.76M		
	ViT-B/16 ($\mathcal{B}3$)	0.5744	22.00M	0.5860	22.00M	0.6103	44.30M	0.5916	22.00M	0.6164	85.85M
	ViT-B/16 ($\mathcal{B}4$)	0.5892	29.17M	0.5908	29.17M	**0.6167**	58.49M	0.6013	29.17M		
	ViT-B/16 ($\mathcal{B}5$)	0.5909	36.00M	0.5960	36.00M	**0.6250**	72.00M	0.6044	36.00M		
TieredImageNet	EfficientNet-B0 ($\mathcal{B}4$)	0.6007	0.32M	0.6048	0.32M	0.6430	0.69M	0.5847	0.32M	0.7352	4.79M
	EfficientNet-B0 ($\mathcal{B}5$)	0.6367	0.99M	0.6359	0.99M	0.6771	1.98M	0.6555	0.99M		
	EfficientNet-B0 ($\mathcal{B}6$)	0.6957	3.11M	0.7096	3.11M	**0.7420**	6.22M	0.6905	3.11M		
Speech Commands	EfficientNet-B0 ($\mathcal{B}3$)	0.9397	0.06M	0.9379	0.06M	0.9516	0.14M	0.9421	0.06M	0.9638	4.05M
	EfficientNet-B0 ($\mathcal{B}4$)	0.9587	0.31M	0.9609	0.31M	**0.9643**	0.64M	0.9625	0.31M		
	EfficientNet-B0 ($\mathcal{B}5$)	0.9632	0.86M	**0.9670**	0.86M	**0.9690**	1.74M	0.9637	0.86M		
	DeepSpeech2 ($\mathcal{B}1$)	0.1590†	7.27M	0.1660†	7.27M	0.1412†	14.58M	0.1520†	7.27M	0.1046†	19.89M
	DeepSpeech2 ($\mathcal{B}2$)	0.1260†	10.43M	0.1270†	10.43M	0.1098†	20.88M	0.1185†	10.43M		

1 accuracy with identical latency. At $t = 10$ s, the primary worker fails. HYDRA transitions to the full ensemble $h_{\{1,2\}}$, which sustains 77.69% accuracy with a 91 ms failover transition. Figure 4b shows FailLite, which transitions to its compressed-model backup, delivering 74.71% accuracy at lower per-request latency than HYDRA's full ensemble, with a 78 ms failover transition, since FailLite executes entirely on a single server and avoids the inter-worker coordination required by HYDRA. Finally, Figure 4c shows split inference, in which the primary model falls back to its pipelined backup, delivering Top-1 accuracy of 77.63% that matches the original, but with a higher transition time of 101 ms and a much higher latency due to its multi-stage pipeline.

At $t = 20$ s, a second failure occurs. HYDRA loses one upstream worker and falls back to the standalone upstream model $h_{\{1\}}$, which delivers 74.28% accuracy in 79 ms (see Figure 4a). By contrast, FailLite, having exhausted its single backup, requires 212 ms to load a secondary cold backup from disk before resuming service at the same 74.71% accuracy as before (see Figure 4b). Finally, split inference, by contrast, cannot recover at all, since the loss of any pipeline component disables the entire backup (see Figure 4c). Across all transitions, HYDRA recovers within sub-100 ms and resumes inference at accuracy comparable to the original model. While FailLite's recovery time grows by nearly 3× under the second failure and its accuracy remains below HYDRA's full-ensemble operating point, split inference recovers in over 100 ms after the primary failure and cannot recover from the second.

Takeaway: Across two sequential failures, HYDRA recovers in under 100 ms at near-original accuracy, while FailLite's recovery time grows nearly 3× under the second failure and split inference cannot recover at all.

C. Ensemble Performance

In this subsection, we evaluate the benefits of HYDRA's ensemble design across four DNN architectures and three datasets. We highlight six key takeaways from the results.

1. Full-ensemble and the original have comparable accuracy. As shown in Table II, across all evaluated configurations the downstream aggregator $h_{\{1,2\}}$ matches the original model's accuracy at a fraction of its parameter count. For EfficientNet-B0 on CIFAR-100, the $\mathcal{B}5$ ensemble achieves 77.69% Top-1 accuracy compared to the original model's 77.63%, while consuming only 1.79M parameters relative to the original's 4.79M (37% of the parameter budget). On Speech Commands, the EfficientNet-B0($\mathcal{B}4$) ensemble attains 96.43% accuracy

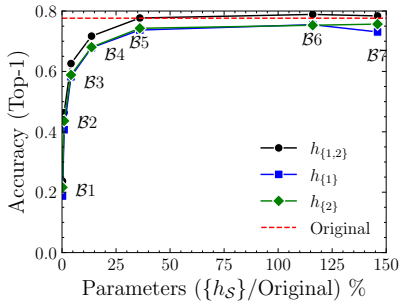


Fig. 5: Top-1 accuracy of HYDRA ensembles on CIFAR-100 as a function of normalized ensemble parameter count.

against the original's 96.38%, despite using only 0.64M parameters, approximately 15.8% of the original's 4.05M.

2. Upstream models retain accuracy under partial availability. The standalone-mode failover behavior depends on individual upstream models remaining accurate when the downstream aggregator is unreachable. Table II shows that this property holds across the evaluated configurations. For example, for the EfficientNet-B0($\mathcal{B}5$) ensemble on CIFAR-100, $h_{\{1\}}$ and $h_{\{2\}}$ achieve 74.28% and 73.71% Top-1 accuracy respectively, close to the original model's 77.63%, retaining 95.6% of the full-ensemble accuracy of 77.69%. For EfficientNet-B0($\mathcal{B}5$) ensemble on Speech Commands, $h_{\{2\}}$ achieves 96.70% accuracy, even higher than the original model's 96.38%, demonstrating the effectiveness of HYDRA's multi-level training approach.

3. Ensembles approach original accuracy at a fraction of parameter cost. Ensemble size, measured as parameter count relative to the original model, governs the accuracy-resource trade-off HYDRA's selection procedure navigates. Table II and Figure 5 report this trade-off for EfficientNet-B0 on CIFAR-100 across all seven block-prefix configurations $\mathcal{B}1$ through $\mathcal{B}7$. As shown, ensemble accuracy increases monotonically with parameter budget up to approximately 40% of the original model size, where the $\mathcal{B}5$ configuration recovers near-original accuracy (77.69% vs. 77.63% at 1.79M of 4.79M parameters). Additionally, increasing the ensemble size exhibits diminishing returns. The $\mathcal{B}6$ and $\mathcal{B}7$ ensembles reach 122% and 152% of the original parameter count, respectively, but their accuracy improvements over $\mathcal{B}5$ are marginal.

4. Generalization across architectures and modalities. Trends established for EfficientNet-B0 in the preceding takeaways extend to other architectures and modalities in Table II. For example, ViT-B/16($\mathcal{B}4$) on CIFAR-100 achieves 61.67% Top-1 accuracy at 58.49M parameters against the original's 61.64% at 85.85M, recovering near-original accuracy at 68% of the parameter budget of the original model. For audio workloads, DeepSpeech2($\mathcal{B}2$) on Speech Commands attains a Word Error Rate (WER) of 0.1098, within 0.0052 of the original's 0.1046 (lower is better). The cross-architecture and cross-modality consistency of these results indicates that the framework is independent of a particular structural class or output modality.

5. Coarser output granularity yields strong worst-case accuracy and preserves HYDRA's ensemble performance. As discussed in Section IV, a key design decision for the upstream models is the ability to adjust the model complexity by changing the output granularity, i.e. utilizing the 20 superclasses coarse labels from CIFAR-100 to train upstream models $h_{\{1\}}$ and $h_{\{2\}}$, while training downstream $h_{\{1,2\}}$ on the fine labels of the dataset, is shown in Table III. Notably, the smaller backup models EfficientNet-B0($\mathcal{B}5$) have 83.6% accuracy, attributed to the fact that the subproblem is less complex, and the ensemble accuracy is still maintained on the fine-labels within 1.6%.

TABLE III: Effect of Hierarchical Training on CIFAR-100. Reducing task complexity improves the accuracy of upstream models ($h_{\{1\}}, h_{\{2\}}$).

DNN Model	Fine-Grain Labels			Coarse-Grain Labels		
	$h_{\{1\}}$	$h_{\{2\}}$	$h_{\{1,2\}}$	$h_{\{1\}}$	$h_{\{2\}}$	$h_{\{1,2\}}$
EfficientNet-B0 ($\mathcal{B}4$)	0.6785	0.6804	0.7169	0.7890	0.8104	0.7215
EfficientNet-B0 ($\mathcal{B}5$)	0.7428	0.7371	0.7769	0.8197	0.8360	0.7617
Resnet50 ($\mathcal{B}3$)	0.7324	0.7043	0.7552	0.7704	0.8005	0.7423

TABLE IV: Performance of $h_{\{1,2\}}$ under different training strategies, reported per row-specific architecture/prefix configuration. *Standalone* trains only $h_{\{1,2\}}$ for that configuration, while *Individually trained* trains $h_{\{1\}}, h_{\{2\}}$ independently and then optimizes $h_{\{1,2\}}$ with frozen upstream models.

Dataset	DNN Model	HYDRA Perf.	Standalone	Ind. Trained
CIFAR-100	EfficientNet-B0 ($\mathcal{B}4$)	**0.7169**	0.7064	0.7038
	EfficientNet-B0 ($\mathcal{B}5$)	**0.7769**	0.7524	0.7462
	EfficientNet-B0 ($\mathcal{B}6$)	**0.7888**	0.7662	0.7615
	ResNet50 ($\mathcal{B}2$)	**0.6820**	0.6338	0.6593
	ResNet50 ($\mathcal{B}3$)	0.7552	0.7579	0.7235
	ViT-B/16 ($\mathcal{B}4$)	**0.6167**	0.6111	0.6001
	ViT-B/16 ($\mathcal{B}5$)	0.6250	0.6279	0.6071
Tiered-ImageNet	EfficientNet-B0 ($\mathcal{B}5$)	0.6771	0.6955	0.6589
	EfficientNet-B0 ($\mathcal{B}6$)	**0.7420**	0.7207	0.6925
Speech Commands	EfficientNet-B0 ($\mathcal{B}2$)	0.8413	0.8495	0.8206
	EfficientNet-B0 ($\mathcal{B}3$)	**0.9516**	0.9504	0.9457
	EfficientNet-B0 ($\mathcal{B}4$)	**0.9643**	0.9623	0.9634
	DeepSpeech2 ($\mathcal{B}1$)	**0.1412**	0.1546	0.1499
	DeepSpeech2 ($\mathcal{B}2$)	**0.1098**	0.1217	0.1181

6. HYDRA ensembles beat alternative training strategies. Table IV compares HYDRA's joint training procedure against two baselines that share the same upstream-downstream ensemble structure: *standalone*, which trains $h_{\{1,2\}}$ end-to-end without constraining the upstream accuracies $h_{\{1\}}$ and $h_{\{2\}}$, and *individually-trained*, which trains each upstream model independently before fitting the downstream aggregator on the frozen features. For EfficientNet-B0($\mathcal{B}5$) on CIFAR-100, HYDRA attains 77.69% aggregate accuracy against 75.24% for standalone and 74.62% for individually-trained, and HYDRA leads on 10 of the 14 evaluated configurations. Standalone cannot serve as a failover, since its upstream accuracies are never trained for, while individually-trained is failover-capable but consistently trails HYDRA, since two-stage training forgoes the mutual refinement that joint training provides.

D. HYDRA System Performance

We compare HYDRA against the FailLite and split-inference baselines along four dimensions: per-request latency across upstream sizes, accuracy under tight resource budgets, sensitivity

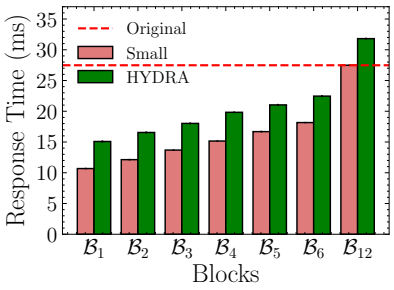


Fig. 6: Per-request response time of HYDRA ViT ensembles as a function of upstream model size $\mathcal{B}1$ through $\mathcal{B}k$.

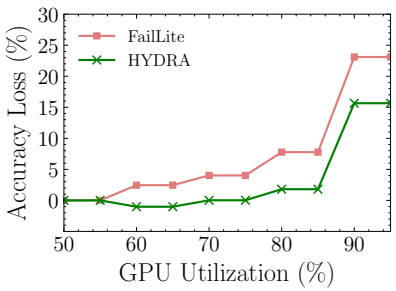


Fig. 7: Accuracy loss of HYDRA and FailLite relative to the original ViT-B/16 across GPU utilization levels.

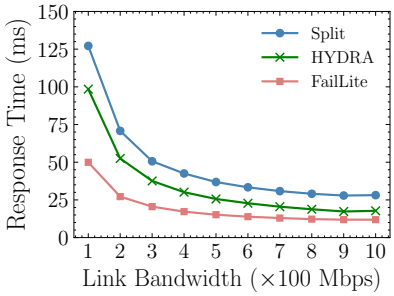


Fig. 8: Response time of HYDRA, FailLite, and split-inference backups across network bandwidths.

to network bandwidth, and ensemble loading time.[1]

1. Ensemble latency remains close to the compressed baseline. Figure 6 shows that HYDRA's ViT ensembles remain within 5 ms of the FailLite-style *Small* baseline across prefix lengths, despite the additional network hop for intermediate representations. The ensemble also matches or outperforms the *Original* model at most prefix lengths. For example, ViT($\mathcal{B}6$) achieves a response time of 22.5 ms compared to 27.5 ms for the original model, reducing latency by 18%.

2. Ensembles retain higher accuracy under resource constraints. Figure 7 compares the accuracy loss of HYDRA and FailLite-style compressed backups for ViT-B/16 across varying GPU utilization levels. HYDRA consistently incurs lower accuracy loss, reducing it by up to 9.55% compared to FailLite under the same resource constraints. This advantage comes from distributing the available budget across multiple lightweight upstream models whose aggregate output recovers more of the original model's accuracy than a single compressed model.

3. Ensemble execution reduces latency relative to split inference. Figure 8 compares the response time of HYDRA, FailLite, and split inference across inter-edge bandwidths from 100 Mbps to 1 Gbps. HYDRA consistently outperforms split inference by 1.30-1.61×, benefiting from parallel upstream execution and concurrent activation transfers. At 100 Mbps, HYDRA achieves 98 ms response time versus 127 ms for split inference, while at 1 Gbps the gap widens to 17 ms versus 28 ms as split inference's sequential pipeline overhead persists.

VII. Related Work

AI Inference Systems. Tools such as Nvidia Triton [20], Kubeflow [21], TensorFlow Serving [35], TVM [27], and ONNX Runtime [36] allow users to deploy models in the cloud, edge, and on-device platforms. Building on these tools, researchers have focused on optimizing the inference costs [1], [13], [37]–[44] and enhancing workload dynamics [2], [43], [45]–[47]. HYDRA addresses failure resiliency, which remains underexplored in resource-constrained environments.

Failure Resiliency in Resource-constrained Environments. The abundant resources in the cloud have enabled replication-based approaches. For instance, DLIS [13] replicates every request to ensure failure resiliency. In contrast, edge environments are resource-constrained and prone to power and hardware failures. Widely accepted strategies for the edge therefore rely on graceful degradation [14], [15], [48], [49]. Prior work addresses one or more failing servers via criticality-based resource allocation [48], [49], model compression and selection [15], [17], and early-exit and skip-connection mechanisms [50]–[52]. In contrast, HYDRA trains an ensemble that pools unused resources across multiple servers, providing better flexibility and fault tolerance.

Ensemble Learning. Ensemble methods such as Bagging [53], AdaBoost [54], Random Forests [55], and Mixture-of-Experts (MoE) [56] remain a cornerstone of statistical learning theory. More recent work [18] has transformed MoE from a modular learner into a scalable conditional-compute layer, enabling trillion-parameter-scale capacity at roughly the cost of a dense baseline. However, these methods do not consider robustness against individual learner failures. In contrast, HYDRA requires each learner to be generalizable across inputs so that the failure of a learner is not catastrophic.

VIII. Conclusions

This paper presented HYDRA, a failure-resilient edge inference system that pools unused capacity across neighboring edge nodes as a collective backup for a primary model. HYDRA's multi-level ensemble learning framework jointly trains lightweight upstream models and a downstream aggregator so that individual members perform competently in isolation and refine each other's predictions when combined. Across a broad range of vision and audio workloads, HYDRA ensembles recover the accuracy of the primary model at a fraction of its parameter count and preserve accuracy under server failures. Compared to state-of-the-art failover approaches, HYDRA reduces response time by up to 1.61× while matching the accuracy of the original primary model.

Acknowledgment

This research is supported by NSF awards #2325956, #2211302, #2211888, #2213636, #2105494, #2502536, DEVCOM ARL Contract #W911NF-17-2-0196, and Adobe. The views and conclusions are those of the authors and are not to be interpreted as representing the official policies, either expressed or implied, of the funding agencies.

[1]Split inference omitted from Figure 6 and 7, since its response time varies with the split point, which admits a combinatorial set of configurations per architecture, and under tight budgets it matches *Original* or fails to deploy.

REFERENCES

[1] Q. Liang, W. A. Hanafy, A. Ali-Eldin, and P. Shenoy, "Model-Driven Cluster Resource Management for AI Workloads in Edge Clouds," *ACM Trans. Auton. Adapt. Syst.*, vol. 18, no. 1, mar 2023. [Online]. Available: https://doi.org/10.1145/3582080

[2] 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*, ser. IoTDI '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 209–221. [Online]. Available: https://doi.org/10.1145/3576842.3582375

[3] N. Ng, W. A. Hanafy, P. Kadambi, B. Sunil, A. Gupta, D. Irwin, Y. Simmhan, and P. Shenoy, "Collaborative Processing for Multi-Tenant Inference on Memory-Constrained Edge TPUs," in *2026 22nd International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, 2026.

[4] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[5] G. Bartolomeo, J. Cao, X. Su, and N. Mohan, "Characterizing Distributed Mobile Augmented Reality Applications at the Edge," in *Companion of the 19th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT 2023, 2023, p. 9–18. [Online]. Available: https://doi.org/10.1145/3624354.3630584

[6] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial Internet of Things: Challenges, Opportunities, and Directions," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[7] 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*, ser. HotMobile '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 22–29. [Online]. Available: https://doi.org/10.1145/3446382.3448360

[8] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2020.

[9] M. Satyanarayanan, "The Emergence of Edge Computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[10] Q. Dong, J. Xu, P. Pillai, and M. Satyanarayanan, "Does Accurate Real-Time AI Need Edge Offload?" in *Proceedings of the Tenth ACM/IEEE Symposium on Edge Computing*, ser. SEC '25, 2025. [Online]. Available: https://doi.org/10.1145/3769102.3770616

[11] S. Bagchi, M.-B. Siddiqui, P. Wood, and H. Zhang, "Dependability in Edge Computing," *Commun. ACM*, vol. 63, no. 1, p. 58–66, Dec. 2019. [Online]. Available: https://doi.org/10.1145/3362068

[12] T. Abdelzaher, N. Ayanian, T. Basar, S. Diggavi, J. Diesner, D. Ganesan, R. Govindan, S. Jha, T. Lepoint, B. Marlin, K. Nahrstedt, D. Nicol, R. Rajkumar, S. Russell, S. Seshia, F. Sha, P. Shenoy, M. Srivastava, G. Sukhatme, A. Swami, P. Tabuada, D. Towsley, N. Vaidya, and V. Veeravalli, "Toward an Internet of Battlefield Things: A Resilience Perspective," *Computer*, vol. 51, no. 11, pp. 24–36, Nov. 2018.

[13] 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, may 2019, pp. 15–17.

[14] I. Koren and C. M. Krishna, *Fault-Tolerant Systems*, 2nd ed. Morgan Kaufmann, 2021.

[15] 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)*, 2023, pp. 144–149.

[16] X. Guo, Q. Jiang, A. D. Pimentel, and T. Stefanov, "Model and system robustness in distributed CNN inference at the edge," *Integration*, vol. 100, p. 102299, 2025.

[17] L. Wu, W. Hanafy, T. Abdelzaher, D. Irwin, J. Milzman, and P. Shenoy, "FailLite: Failure-resilient model serving for resource-constrained edge environments," in *Proceedings of the 2025 ACM Symposium on Cloud Computing*, ser. SoCC '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 416–429. [Online]. Available: https://doi.org/10.1145/3772052.3772243

[18] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=B1ckMDqlg

[19] N. Hayashibara, X. Défago, R. Yared, and T. Katayama, "The φ accrual failure detector," in *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS)*, 2004, pp. 66–78.

[20] NVIDIA, "Triton inference server," 2024, accessed: 2025-04-13. [Online]. Available: https://developer.nvidia.com/triton-inference-server

[21] Kubeflow, "Kubeflow: The machine learning toolkit for kubernetes," 2025, accessed: 2025-04-14. [Online]. Available: https://www.kubeflow.org/

[22] Y. Xiong, Y. Jiang, Z. Yang, L. Qu, G. Zhao, S. Liu, D. Zhong, B. Pinzur, J. Zhang, Y. Wang *et al.*, "SuperBench: Improving cloud AI infrastructure reliability with proactive validation," in *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, 2024, pp. 835–850.

[23] L. Wu, W. A. Hanafy, A. Souza, T. Abdelzaher, G. Verma, and P. Shenoy, "Enhancing Resilience in Distributed ML Inference Pipelines for Edge Computing," in *MILCOM 2024 - 2024 IEEE Military Communications Conference (MILCOM)*, 2024, pp. 1–6.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

[25] M. Tan and Q. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 09–15 Jun 2019, pp. 6105–6114. [Online]. Available: https://proceedings.mlr.press/v97/tan19a.html

[26] NVIDIA, "NVIDIA TensorRT," 2026. [Online]. Available: https://developer.nvidia.com/tensorrt

[27] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "TVM: an automated end-to-end optimizing compiler for deep learning," in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'18. USA: USENIX Association, 2018, p. 579–594.

[28] R. Wightman, "PyTorch Image Models," https://github.com/huggingface/pytorch-image-models, 2019.

[29] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations (ICLR)*, 2021.

[30] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, J. Chen, J. Chen, Z. Chen, M. Chrzanowski, A. Coates, G. Diamos, K. Ding, N. Du, E. Elsen, J. Engel, W. Fang, L. Fan, C. Fougner, L. Gao, C. Gong, A. Hannun, T. Han, L. V. Johannes, B. Jiang, C. Ju, B. Jun, P. LeGresley, L. Lin, J. Liu, Y. Liu, W. Li, X. Li, D. Ma, S. Narang, A. Ng, S. Ozair, Y. Peng, R. Prenger, S. Qian, Z. Quan, J. Raiman, V. Rao, S. Satheesh, D. Seetapun, S. Sengupta, K. Srinet, A. Sriram, H. Tang, L. Tang, C. Wang, J. Wang, K. Wang, Y. Wang, Z. Wang, Z. Wang, S. Wu, L. Wei, B. Xiao, W. Xie, Y. Xie, D. Yogatama, B. Yuan, J. Zhan, and Z. Zhu, "Deep Speech 2: End-to-end speech recognition in English and Mandarin," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, p. 173–182.

[31] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," University of Toronto, Tech. Rep., 2009, technical Report. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[32] M. Ren, E. Triantafillou, S. Ravi, J. Snell, K. Swersky, J. B. Tenenbaum, H. Larochelle, and R. S. Zemel, "Meta-Learning for Semi-Supervised Few-Shot Classification," in *Proceedings of 6th International Conference on Learning Representations ICLR*, 2018.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[34] P. Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition," *ArXiv e-prints*, Apr. 2018. [Online]. Available: https://arxiv.org/abs/1804.03209

[35] C. Olston, F. Li, J. Harmsen, J. Soyke, K. Gorovoy, L. Lao, N. Fiedel, S. Ramesh, and V. Rajashekhar, "Tensorflow-serving: Flexible, high-performance ml serving," in *Workshop on ML Systems at NIPS 2017*, 2017.

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

[37] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A Low-Latency online prediction serving system," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, Mar. 2017, pp. 613–627. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw

[38] W. Zhang, S. Li, L. Liu, Z. Jia, Y. Zhang, and D. Raychaudhuri, "Hetero-Edge: Orchestration of Real-time Vision Applications on Heterogeneous Edge Clouds," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 1270–1278.

[39] C. Samplawski, J. Huang, D. Ganesan, and B. M. Marlin, "Towards objection detection under IoT resource constraints: Combining partitioning, slicing and compression," in *Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*, ser. AIChallengeIoT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 14–20.

[40] 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.

[41] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving DNNs like clockwork: Performance predictability from the bottom up," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 443–462. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/gujarati

[42] 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.

[43] 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*, ser. HPDC '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 267–280. [Online]. Available: https://doi.org/10.1145/3625549.3658688

[44] Q. Liang, P. Shenoy, and D. Irwin, "AI on the Edge: Characterizing AI-based IoT Applications Using Specialized Edge Architectures," in *2020 IEEE International Symposium on Workload Characterization (IISWC)*, 2020, pp. 145–156.

[45] S. Ahmad, H. Guan, B. D. Friedman, T. Williams, R. K. Sitaraman, and T. Woo, "Proteus: A High-Throughput Inference-Serving System with Accuracy Scaling," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 318–334. [Online]. Available: https://doi.org/10.1145/3617232.3624849

[46] 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, Jul. 2020. [Online]. Available: https://www.usenix.org/conference/hotcloud20/presentation/zhang

[47] 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)*. USENIX Association, Jul. 2020, pp. 353–369.

[48] J. Zou, X. Dai, and J. Mcdermid, "Graceful Degradation with Condition- and Inference-Awareness for Mixed-Criticality Scheduling in Autonomous Systems," in *Proceedings of Cyber-Physical Systems and Internet of Things Week 2023*, ser. CPS-IoT Week '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 215–220. [Online]. Available: https://doi.org/10.1145/3576914.3587511

[49] J. J. Meza, T. Gowda, A. Eid, T. Ijaware, D. Chernyshev, Y. Yu, M. N. Uddin, R. Das, C. Nachiappan, S. Tran, S. Shi, T. Luo, D. K. Hong, S. Panneerselvam, H. Ragas, S. Manavski, W. Wang, and F. Richard, "Defcon: Preventing Overload with Graceful Feature Degradation," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. Boston, MA: USENIX Association, Jul. 2023, pp. 607–622. [Online]. Available: https://www.usenix.org/conference/osdi23/presentation/meza

[50] 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*, 2019, pp. 25–31.

[51] 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," *FL-ICML 2020*, 2020.

[52] 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)*. IEEE, 2022, pp. 143–152.

[53] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, pp. 123–140, 1996.

[54] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[55] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[56] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the EM algorithm," *Neural computation*, vol. 6, no. 2, pp. 181–214, 1994.