Rising from Pieces: Effective Inference at the Edge via Robust Split ML

Yuxuan Weng, Tianyue Zheng✉, *Member, IEEE*, Zhe Chen, *Member, IEEE*, Menglan Hu, *Member, IEEE*, Jun Luo, *Fellow, IEEE*

Abstract—The increasing processing demands of today's mobile deep learning applications impose stringent requirements on edge devices. Offloading these tasks to the cloud, while being a potential solution, often results in significant data transfer overhead, as well as privacy and connectivity concerns. To address these challenges, split machine learning (split ML) has emerged as an innovative paradigm, enabling task distribution among edge devices themselves. However, split ML systems inherently exhibit instability due to the hardware and communication limitations of mobile devices, which frequently result in failures and malfunctions of client nodes. In light of these challenges, we present Axolotl, a fault-tolerant edge split ML inference system for addressing node failure with minimal performance impact. Specifically, we first design a novel *curriculum dropout* mechanism to enhance the model's resilience by gradually exposing it to potential server node failures. We then design *inverse-proximal weight consolidation* to mitigate catastrophic forgetting caused by curriculum dropout. To further tackle potential node failures, we innovate in a resource-aware substitution module that offload the functions of a failed node to neighboring ones, ensuring efficient information flow. Extensive experiments demonstrate the effectiveness and robustness of Axolotl in various deep learning networks and tasks in edge environments.

Index Terms—Split ML, mobile edge computing, robust inference.

1 Introduction

The explosive growth of mobile *machine learning* (ML) applications [1], [2], [3], including speech recognition [4], [5], [6], face identification [7], [8], question answering [9], [10], and content generation [11], [12], has created a fundamental tension between computational demands and the inherent limitations of edge devices. Traditional approaches to address this challenge have fallen short. While hardware manufacturers strive to enhance computational power, this strategy faces diminishing returns as Moore's law reaches its physical limits [13], resulting in prohibitive costs and practical constraints. Alternatively, deploying compressed ML models on resource-constrained devices often involves performance degradation, potentially leading to suboptimal user experiences [14], [15], [16], [17]. Cloud computing [18], [19] has emerged as a prominent solution, with industry leaders like Apple's CloudKit [20] and Google's Firebase [21] offering computational offloading services. However, this approach introduces its challenges: significant communication overhead, privacy vulnerabilities, and dependence on stable network connectivity. These limitations raise an intriguing possibility: rather than relying on distant cloud servers, could we leverage the computational resources of nearby personal edge devices to distribute and process these complex ML tasks more efficiently?

To address this challenge, the field of collaborative learning [22], [23], [24], [25] provides paradigms for distributing large-scale models across multiple devices. Among these, split ML [26], [27], [28], particularly its multi-split variant [29], [30], has emerged as a promising approach for distributed inference. Specifically, this collaborative learning approach enables the strategic partitioning of large-scale models across multiple devices, effectively harnessing their collective computational resources. Specifically, connected nodes create an execution chain where each node processes its assigned layers and forwards the intermediate data to the next node, enabling real-time utilization of complex models without being constrained by the computational limitations of any single device. While related to other collaborative paradigms like federated learning (FL), the distinction is crucial. FL typically involves a parallel architecture where clients train on local data and aggregate model updates to address the distributed training problem. In contrast, split ML employs a sequential, model-partitioning architecture to tackle the distinct challenge of distributed inference. This innovative architecture has already demonstrated its versatility across numerous real-world applications, from virtual assistants [27] and autonomous driving systems [31], [32] to object detection algorithms [33], [34], [35] and immersive augmented reality [36], [37].

Nevertheless, the chain-like nature of split ML inference means that a single node failure can halt the entire process, posing a unique and critical reliability challenge compared to FL. To mitigate the inherent drawbacks of split ML, several strategies have been proposed. For example, reducing the number of nodes can simplify the topology of the execution chain, thereby decreasing the likelihood of node malfunctions. Alternatively, preparing parallel backup

- *Y. Weng and T. Zheng are with the Department of Computer Science and Engineering, Southern University of Science and Technology, China. E-mail: {wengyx, zhengty}@sustech.edu.cn*
- *Z. Chen is with the School of Computer Science, Fudan University, China. E-mail: zhechen@fudan.edu.cn*
- *M. Hu is with the School of Electronic Information and Communications, Huazhong University of Science and Technology, China. E-mail: humenglan@hust.edu.cn*
- *J. Luo is with the College of Computing and Data Science, Nanyang Technological University, Singapore. E-mail: junluo@ntu.edu.sg*
- ✉ *Corresponding author: Tianyue Zheng.*

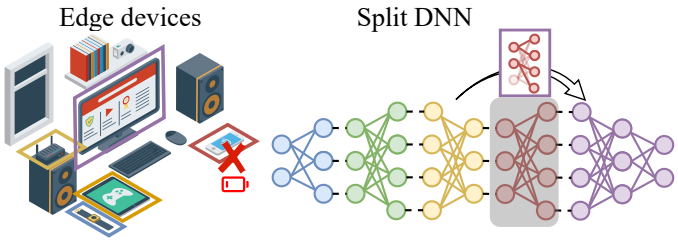


Fig. 1: The basic idea of Axolotl.

nodes can safeguard against potential node failures. However, the aforementioned solutions impose implicit requirements on the execution process. The former increases the computational burden on each node, necessitating substantial computational capabilities in edge devices, while the latter complicates the network topology of the system and escalates the overall system's hardware requirements, reducing the benefits of split ML.

To enable collaborative inference across mobile edge devices, we aim to develop a universal split ML system that overcomes these limitations without imposing additional hardware requirements or complicating the network topology. To achieve this objective, we must address several key challenges. First, we need to design a system where remaining nodes can dynamically assume the functions of missing nodes, adapting seamlessly to their absence. This capability is not inherent in traditional machine learning models. Second, we must mitigate the risk of catastrophic forgetting during the adaptation process, which can occur due to the loss of parameters in certain sections, potentially resulting in excessively large gradients during parameter updates and destabilizing the learning process. Finally, we need to resolve data flow mismatches that can occur when node malfunctions block the forward propagation of the execution chain, which prevents the model from producing effective outputs.

In this work, we introduce Axolotl, a robust split ML inference system specifically designed for edge computing, aimed at overcoming challenges posed by node malfunctions, compromised performance, and reliability issues, as briefly illustrated in Figure 1. Axolotl is designed to adapt to node malfunctions during model training and provide remedial measures during inference in the presence of such malfunctions. Specifically, Axolotl incorporates a curriculum dropout mechanism that exposes the model to a range of malfunctions from mild to severe during training, gradually building resilience and fault tolerance. We also integrate an inverse-proximal weight consolidation (IPWC) regularization to prevent catastrophic forgetting induced by dropout, with lighter regularization near dropout regions to encourage functionality sharing among neighboring nodes. Following a node malfunction, Axolotl's resource-aware substitution module evaluates the computational capabilities of successor nodes and decides whether to allocate a minimal substantial model or a data reshaping layer to replace the failed node. The primary contributions of this paper are as follows:

- To the best of our knowledge, Axolotl is the first system to enable robust collaborative inference using a split ML (model-partitioning) architecture, directly addressing the critical challenge of node failures in such collaborative settings.
- We propose a novel curriculum dropout mechanism that allows the collaborative inference pipeline to adapt dynamically to the absence of nodes, ensuring end-to-end resilience and fault tolerance for the entire collaborative system.
- We develop IPWC to mitigate catastrophic forgetting induced by dropout and to encourage the sharing of functionalities among neighbor nodes.
- We have developed a resource-aware substitute model that safeguards data size and information flow by adapting to the resources of successor nodes during forward propagation, even amidst node malfunctions.
- We demonstrate the effectiveness and robustness of our approach by evaluating the Axolotl pipeline performance on various deep learning networks and tasks in edge environments.

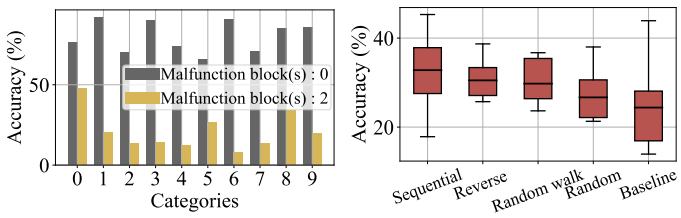
The remainder of this paper is structured as follows. § 2 discusses the motivation behind Axolotl's design. § 3 delves into the system design of Axolotl, elaborating on the curriculum dropout mechanism, IPWC regularization, and resource-aware substitution module. § 4 introduces the datasets, system implementation, and experimental setup, followed by a presentation of the evaluation results in § 5, showcasing the effectiveness of Axolotl. Finally, § 7 concludes the paper by encapsulating the key insights, implications, and potential future directions for research.

2 Background and Motivations

In this section, we explore the challenges of deployed split ML system during inference to motivate Axolotl's design.

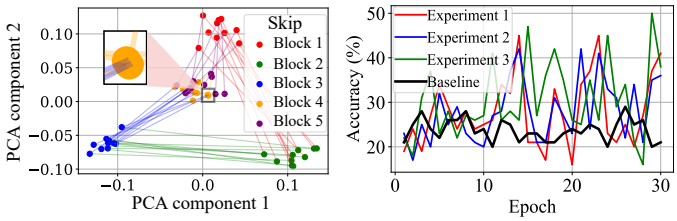
2.1 Recent Works in Collaborative Learning

A large body of work on collaborative edge inference focuses on optimizing performance metrics such as latency, energy consumption, and communication overhead. For example, [22] addresses memory limitations and load balancing on IoT devices through model-parallelism techniques and heuristic-based static pipeline generation. [23] provides a comprehensive survey of end-edge-cloud collaboration, covering techniques such as model partitioning, compression, and privacy preservation. [24] presents a system for adaptive workload partitioning that reduces energy consumption under latency constraints by adjusting task distribution based on real-time device and network conditions. [25] applies multi-stage design space exploration with a genetic algorithm to find static CNN partitions that balance energy, memory, and throughput. However, these frameworks generally assume reliable nodes and stable network conditions. This assumption is a major limitation for split ML, whose sequential pipeline structure is inherently fragile, the failure of a single node can halt the entire inference process. While initial efforts like SPINN [18] and AgileNN [38] have addressed certain system disruptions, a robust and generalizable solution for handling frequent node malfunctions remains a critical obstacle to the practical deployment of split ML in dynamic edge environments.



(a) Accuracy degradation. (b) Different dropout methods.

Fig. 2: Impact of node malfunction.



(a) Model weight transitions. (b) Training instability.

Fig. 3: Catastrophic forgetting under block dropout.

2.2 Node Malfunction in Edge Computing

In edge computing scenarios where large-scale models need to be distributed across multiple resource-constrained edge devices (e.g., smartphones, smartwatches, and edge servers) for split ML, the stability of edge nodes is critical to the reliability of edge applications. However, these nodes are inherently unstable due to factors such as network reliability and hardware limitations. Notably, up to 45% of edge devices connected to wireless networks suffer from connection stability issues [39], a problem exacerbated by the high mobility of these devices. Moreover, hardware and software limitations contribute significantly to the instability of edge nodes. For instance, a substantial percentage of these devices deplete their batteries in less than 80 hours [40], after which they will go offline; devices such as smartphones and smartwatches experience forced shutdowns when temperatures exceed safe thresholds [41], [42]. Additionally, the geographical distribution of these devices [43], their lack of safety mechanisms [44], and the complexities involved in managing them [45] further exacerbate the instability of edge nodes.

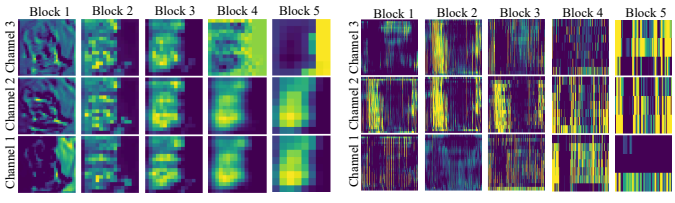
To illustrate the impact of node malfunction on edge inference, we conduct preliminary experiments on the CIFAR-10 dataset. We train a ResNet comprising five blocks and evaluate its accuracy across 10 classes under normal conditions. Subsequently, we set two nodes to a malfunctioning state and evaluate the model's accuracy under these conditions. It is important to note that to ensure the model's operability post-node malfunction, we standardize (i.e., adjusting the output of each node to the same size) the input and output shapes of all blocks. As Figure 2a illustrates, following node malfunction, the model's accuracy in each category drops from around 81% to around 23%, degrading to a weak classifier just slightly better than random guessing.

Exposing the model to node malfunctions during training, such as the dropout technique [46], [47], might offer a viable solution. However, the methodology of node dropout significantly affects the model's performance. We conduct preliminary experiments with various dropout approaches, including sequential, reverse, random walk, and random selection, alongside a non-dropout baseline for comparison. As illustrated in Figure 2b, while all dropout methods outperform the non-dropout baseline, the specific dropout strategy (characterized by node selection order and randomness) exhibits substantial performance variations. These findings calls for a more robust design rather than a random dropout.

2.3 Catastrophic Forgetting

In § 2.2, we discuss how exposing the model to various node malfunction configurations during training can effectively enhance its adaptability to the instability of edge nodes. However, each node malfunction configuration leads to significant structural changes in the model, essentially representing a new task. Therefore, training the model to adapt to different node malfunction configurations may result in catastrophic forgetting [48], a well-known phenomenon in deep learning where learning new tasks significantly degrades performance on previously learned tasks. Consequently, after each update, the model may lose the knowledge learned from previous node malfunction configurations. To intuitively illustrate the impact of catastrophic forgetting, we conduct preliminary experiments using a ResNet with 5 blocks on the CIFAR-10 dataset. Specifically, we strategically introduce a malfunction in one block per round, systematically recording the index of the malfunctioning block alongside the corresponding model weights. This process is repeated over 10 training rounds, during which each of the 5 blocks is dropped once per round. Consequently, we accumulate 50 recorded indices of malfunctioning blocks, along with their model weights.

We show in Figure 3a the distribution of model weights. The dimensionality of the model weight is reduced using PCA. One may readily observe that even at different times during training, models with the same configuration exhibit similar weights. Occasionally, different models under the same configuration converge to nearly identical weights, indicating that the model barely maintains any knowledge of former node malfunction scenarios. Figure 3b further demonstrates the effects of catastrophic forgetting: we repeat the aforementioned training process 3 times, testing the model's accuracy under random node malfunctions at each epoch, and include a normally trained baseline for comparison. The results indicate that although the final accuracy exceeds that of the baseline, catastrophic forgetting causes the model to converge slowly and experience significant instability.



(a) Image classification. (b) RF-based HAR.

Fig. 4: Representation mismatch.

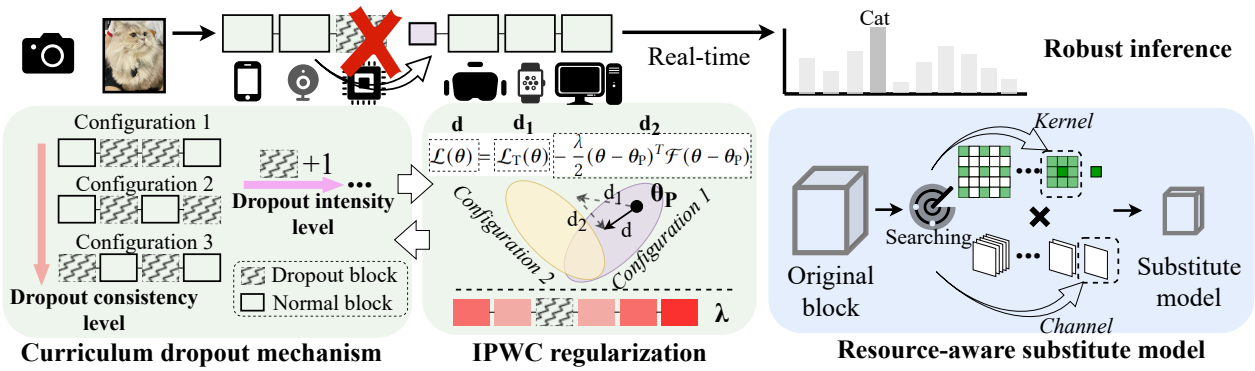


Fig. 5: Overall design of Axolotl.

2.4 Representation Mismatch

In addition to the challenges outlined in § 2.2 and § 2.3, node malfunctions in neural networks present additional issues related to the size and feature mismatches of representations. Specifically, both feature maps and channel numbers can vary at different stages within the network. While techniques like up-sampling and down-sampling can align these sizes, they often lead to the loss of essential information or the introduction of unwanted artifacts. Beyond size mismatch, feature mismatch presents another critical challenge. In deep learning processes, the earlier layers focus on processing low-level features, while the later layers deal with higher-level and more abstract features. This distinct division makes it difficult to simply bypass a malfunctioning node or directly transfer features to the following nodes during inference.

To demonstrate the phenomenon of representation mismatch, we present a comparative analysis of feature maps generated by different blocks in two distinct tasks: image classification and RF-based HAR. Figure 4a and Figure 4b illustrate these outputs, respectively. For clarity, we have selected the three most significant channels from each block's output. As we examine the progression from block 1 to block 5 in both tasks, there is a clear and consistent dimensionality reduction in the feature maps. Moreover, one may readily observe an evolution of the features: the initial blocks capture low-level details, while the later blocks exhibit more abstract features. This evolution is significant, with noticeable differences even between adjacent blocks, forbidding direct bypassing of malfunctioned blocks. This observation underscores the urgent need for a carefully designed module that can addresses the representation mismatch issue and effectively replace malfunctioned nodes.

3 System Design

To effectively manage node malfunctions, we propose Axolotl consisting of 3 components: i) a curriculum dropout mechanism that adapts the model to various degrees of node malfunctions, ii) an IPWC regularization to mitigate catastrophic forgetting and promote functional redundancy among adjacent nodes, and iii) a resource-aware substitution module that replaces functionalities of malfunctioned nodes. Figure 5 illustrates the overall design of Axolotl.

3.1 Curriculum Dropout Mechanism

As explained in § 2.2, introducing node malfunctions during training improves the model's robustness. However, randomly determining the severity of malfunctions may lead to excessively large gradient updates, potentially causing training divergence. Curriculum learning [49], [50] is an effective paradigm in machine learning that can address this challenge: it involves starting with simpler tasks and incrementally increasing the complexity, ensuring a stable training process. One key question of curriculum learning is how to define the task difficulty. In the context of split ML node malfunction, a task is considered less difficult if it incorporates a smaller number of dropout blocks and a more similar dropout configuration to the previous scenario, and vice versa.

As such, we design a hierarchical curriculum dropout strategy consisting of two levels: intensity and consistency, as shown in Figure 6. In the implementation, the intensity level functions as the outer loop of iteration, whereas the consistency level operates as the inner loop. In the dropout intensity level, we control the number of dropout blocks, denoted as b, beginning with 1 and incrementing by 1 with each iteration until the value reaches half the length of the model. In the dropout consistency level, we gradually increase the difference between the current dropout configuration and the previous one. Specifically, for outer loop b, we initially sample a dropout region $\mathcal{R} = \{r_1, r_2, \ldots, r_b\}$ uniformly from the set of all blocks $\mathcal{B}$, where r_i is the block index. In the c-th inner round, each element in $\mathcal{R}$ has a probability of $1 - e^{-\tau c}$ to be replaced with a randomly sampled block, where τ determines the rate this replacement probability increases as the rounds progress. Block r_i in $\mathcal{R}$

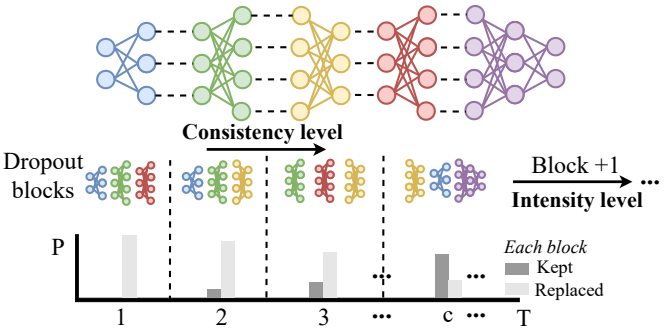


Fig. 6: Hierarchical curriculum dropout strategy.

can be updated as:

$$r_i = (1 - I_i)r_i + I_i r_i', \tag{1}$$

where $r_i' \sim \text{Uniform}(\mathcal{B})$, I_i is a replacement indicator defined by $I_i \sim \text{Bernoulli}(1 - e^{-\tau c})$. Note that c starts from 0, so in the first round, the initial $\mathcal{R}$ is used without modification.

After establishing the dropout region using the aforementioned strategy, the forward propagation process for an input $\mathbf{x}$ through the model can be expressed as follows:

$$\mathbf{h}_i = f_i(\mathbf{h}_{i-1}; \boldsymbol{\theta}_i)q_i + f_i'(\mathbf{h}_{i-1}; \boldsymbol{\theta}_i')(1 - q_i), \tag{2}$$

where $\mathbf{h}_i$ denotes the intermediate representation after the i-th block, $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_i'$ represent the parameters of the original i-th block and its substitute model, respectively (detailed in Section 3.3). Additionally, q_i is a binary indicator where 0 and 1 signify that the block is outside and inside the dropout region, respectively. Following the computation of the loss $\mathcal{L}$, the model updates the parameters outside the dropout region with gradient descent, while parameters within it remain unchanged:

$$\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i - \eta \cdot \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_i} \cdot q_i, \tag{3}$$

where η denotes the learning rate.

3.2 Inverse-Proximal Weight Consolidation Regularization

When the model employs dropout strategies to adapt to various node malfunctions, it experiences catastrophic forgetting, as detailed in § 2.3. This occurs due to the partial unavailability of the model, which forces the neighboring blocks to assume the functions of the dropped-out block. This functional offloading to neighboring blocks results in significant gradient updates, thereby overwriting previously learned knowledge. However, in each dropout scenario, a parameter solution space exists where the weights remain within an acceptable range. Our goal is to identify the intersection of all such solution spaces, enabling the model to handle all instances of model unavailability. Therefore, the parameter optimization in new dropout configurations should ideally remain within or close to the previous solution spaces.

To achieve this objective, it is necessary to ensure the stability of updates for important parameters in new dropout configurations. The importance of parameters is determined by the curvature surrounding the associated optima, which indicates the model's sensitivity to the optimal solutions derived from previous dropout configurations. The greater the curvature in a given direction, the more it indicates the importance of a parameter and necessitates restrictions during training on new configurations, as slight changes are likely to significantly increase the loss. Although the curvature around local minima on the loss surface can be determined by the Hessian matrix, directly computing it is computationally prohibitive in neural networks with a large number of parameters. Therefore, as an effective approximation of the Hessian, the Fisher information matrix (FIM) $\mathcal{F}$ offers a feasible alternative for assessing parameter significance:

$$\mathcal{F} = \mathbb{E}\left[\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{T}}(\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{T}}(\boldsymbol{\theta})^T \Big|_{\boldsymbol{\theta}_{\text{P}}}\right], \tag{4}$$

where $\mathcal{L}_{\text{T}}$ denotes the loss function associated with the task, and $\boldsymbol{\theta}_{\text{P}}$ represents the parameters learned from the previous dropout scenario.

Given $\mathcal{F}$, we can constrain updates to critical parameters when the model encounters new dropout scenarios. Therefore, the modified loss function $\mathcal{L}$, incorporating a regularization term, can be expressed as follows:

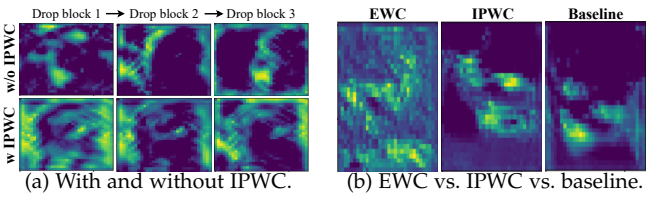
$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{\text{T}}(\boldsymbol{\theta}) - \frac{\lambda}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_{\text{P}})^T \mathcal{F} (\boldsymbol{\theta} - \boldsymbol{\theta}_{\text{P}}), \tag{5}$$

where parameter λ controls the amount of knowledge retained from previous dropout instances. We know that catastrophic forgetting occurs due to structural malfunction in the model, forcing adjacent blocks to assume the functionalities of missing blocks, leading to excessive gradient updates. Given that we now effectively restrict the update magnitudes of crucial parameters, this approach facilitates the allocation of the dropped block's functionalities among adjacent blocks, thereby promoting functional redundancy within the model. This redundancy can mitigate performance degradation resulting from partial model damage and should diminish as the distance from the affected dropout block (i.e., the absolute difference in block indices) increases. Consequently, λ for each block is defined as follows:

$$\lambda(b) = \lambda_0 \log(1 + b), \tag{6}$$

where b represents the distance from the current block to the dropout block, and λ_0 is a hyperparameter. If multiple blocks are dropped, λ is calculated by averaging the values for each of the dropped blocks.

We conduct preliminary experiments to assess the efficacy of IPWC in preserving knowledge from previous dropout scenarios. Our setup involve a simple model comprising 3 blocks, where each block is sequentially subjected to dropout during training. We employ 3 distinct training strategies: no regularization, EWC [51], and IPWC. During the inference phase, we input a single image to get the output of different activation layers under different dropout scenarios. From Figure 7a, it is evident that the model without IPWC demonstrates significantly different feature maps after experiencing different dropout scenarios. In contrast, the model with IPWC retains the core features learned from previous scenarios. Figure 7b illustrates the feature maps generated by both EWC and IPWC in the absence of block 2, after training with sequential dropout. Additionally, a feature map generated by another model through a standard process (i.e., without dropout and node malfunction) is also shown as a baseline for comparison. The feature map from the model trained with IPWC strategy is more similar to the baseline, indicating that this model, by encouraging functional redundancy across blocks, is better adapted to handle block malfunctions. This is because, unlike conventional regularization methods such as EWC [51], which impose uniform constraints across all model parameters,



(a) With and without IPWC. (b) EWC vs. IPWC vs. baseline.

Fig. 7: Feature maps comparison.

IPWC adopts a spatially-aware, non-uniform regularization strategy specifically designed for the distributed nature of split ML.

The key innovation of IPWC lies in Equation (6), where the regularization strength λ is dynamically adjusted based on each block's distance from the failure point. This results in relaxed constraints on adjacent blocks, explicitly encouraging them to absorb and replicate the functionalities of the failed node. This mechanism not only mitigates catastrophic forgetting but also promotes targeted functional redundancy, which is typically absent in general-purpose regularization techniques. Furthermore, because our curriculum training varies the location of this failure point via dropout, the process introduces a structured stochasticity. This, in turn, allows different parts of the model to take larger, more exploratory steps at different times, significantly enhancing its ability to escape local optima compared to the rigid constraints of uniform regularization methods.

3.3 Resource-Aware Substitute Model

As mentioned in Section 2.4, node malfunctions can lead to dimensionality mismatch and feature mismatch during the inference process. To mitigate these issues, we require a dedicated module capable of correcting dimensionality mismatches and maximizing the preservation of feature information. Furthermore, the limited remaining computational capacity of neighboring nodes prevents simply offloading the entire workload of the malfunctioned node. In response to these challenges, we design a resource-aware substitute model, as shown in Figure 8. Our proposed substitute model offers a data interface with minimal computational overhead. Furthermore, the model effectively leverages spare capacity on neighboring nodes to improve its inference performance. Instead of storing multiple models for varying resource constraints, we utilize a single, full-sized, resource-aware model trained with an *elastic shrinking* technique. This

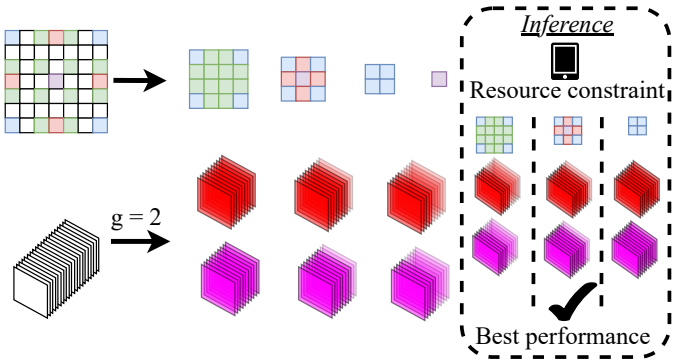


Fig. 8: Design of resource-aware substitute model

technique allows us to prune less important parameters, drastically reducing storage and complexity. Using CNN as an example, we demonstrate how model size can be adjusted dynamically by shrinking the kernel size and the number of channels.

In the dimension of kernel size, directly trimming the original kernel reduces the receptive field, which can still lead to issues of representation mismatch. Consequently, we design an *adaptive* dilated convolution that incorporates additional spaces within the kernel to maintain the same receptive field as the original. In configuring the new $k' \times k'$ dilated convolution kernel, the objective is to align its functionality with that of the original $k \times k$ kernel. The relationship between k' and k is given by $k = (k'-1) \cdot r + 1$, where r denotes the dilation rate, representing the insertion of $r-1$ holes between adjacent pixels in the dilated convolution kernel. The receptive field of the dilated kernel $\mathcal{K}_{\mathrm{D}}$ is initially calculated and used as a mask on the maximal kernel $\mathcal{K}_{\mathrm{M}}$ to retain the parameters of the activated areas:

$$\mathcal{K}_{\mathrm{D}}(m,n) = \mathcal{K}_{\mathrm{M}}((m-1)\cdot r+1, (n-1)\cdot r+1), \quad (7)$$

where (m,n) denotes the pixel coordinates within the kernels, and r is subject to being a positive integer. Since the parameters for all kernel sizes are derived from the maximal kernel and share identical receptive fields, we can prune the kernel to fit the capacity of edge devices. In particular, if the kernel size is set to 2 or 1, the parameters from the four corners or the central element of the original kernel are respectively utilized to compute the entire receptive field.

In the dimension of channel number, we propose a group-based channel pruning method. On one hand, we replace standard convolutions with group convolutions by dividing C_{in} input channels into g groups. Consequently, both the kernel and the corresponding output channels C_{out} are also divided into g groups, allowing for parallel computation across all groups. This method significantly reduces the number of parameters from $C_{\text{in}} \times C_{\text{out}} \times k^2$ to $g \times \frac{C_{\text{in}}}{g} \times \frac{C_{\text{out}}}{g} \times k^2$, where k is the kernel size. On the other hand, as illustrated in Figure 4, there is a certain informational overlap among different channels within the same intermediate layer. Consequently, we further sort the channels within each group by their importance and discard the last s channels to additionally reduce the parameter count, where s is set based on the computational resources available at the edge node. The significance of a channel is determined by the L1 norm of its corresponding parameters in the kernel; a higher L1 norm indicates greater importance, and vice versa. Consequently, when input channels are discarded, the corresponding layers in the kernel are also removed. Note that the aforementioned operations do not alter the number of output channels, thus allowing the direct output to the subsequent block.

We train the substitute model at the location of the dropout block described in Section 3.1 with only minimal training overhead. Furthermore, we randomly adjust the sizes of both dimensions during training to ensure consistent performance across various scales of substitute models. It is important to note that both the full-sized and the pruned substitute models are significantly smaller than the original model block (e.g., the smallest 1×1 convolution kernel merely serves to change the feature size), thus not

impacting the necessity and significance of § 3.1 and § 3.2. During the inference process, we calculate potential configurations for the substitute model based on the remaining computational capacity of the successor node and select the configuration that offers the best performance for implementation.

We only employ CNN architectures here as examples to clearly introduce the design principles of a resource-aware substitute model. However, these design concepts can be extended to other model structures, such as RNNs and transformers, provided that the design accounts for structural sparsification across various dimensions of the models. Moreover, these design approaches require the capability to dynamically adjust the size of the substitute model without training multiple models. For RNNs, we can decompose the original weight matrix into two smaller matrices using the Kronecker product [52], and reduce the number of temporal connections by employing dilated recurrent skip connections [53]. For transformers, by implementing a sliding (dilated) window [54], the computational complexity can be reduced. Additionally, multi-head pruning [55] can be utilized to further decrease computational demands.

3.4 Deployment Management

During inference, each edge node follows a protocol to detect and compensate for potential node malfunctions. First, each node continuously broadcasts a UDP heartbeat signal, confirming its operational status. Second, upon completing its inference task, a node multi-casts its intermediate results to all successor nodes. Critically, each node monitors the heartbeat signals of both its predecessors and successors. If a node detects a missing heartbeat from a predecessor within a predefined interval, indicating a potential failure, it dynamically configures and deploys the most appropriate substitute model. This substitute model seamlessly integrates into the inference pipeline, accepting the predecessor's intermediate data and continuing the computation. Similarly, if a node does not receive a heartbeat signal from any subsequent nodes, it proactively configures substitute models for all downstream nodes in the inference path. These substitute models serve as output interfaces, ensuring the inference process completes even in the presence of multiple failures.

4 Implementations

4.1 Tasks and Datasets

Given that the application scenario of Axolotl is in edge computing, we select several common tasks and corresponding datasets. Firstly, CIFAR-100 [56] is employed for the ubiquitous image classification task. This dataset consists of 60,000 images of 100 categories. Secondly, for large-scale image classification tasks, the ImageNet dataset [57] is employed, which contains approximately 1.2 million training images, 50,000 validation images, and 100,000 test images across 1,000 categories. Thirdly, for the RF-based human activity recognition (HAR) task, which is crucial in smart home and healthcare tasks, we utilize the RF-Net dataset [58]. This dataset utilizes the Wi-Fi modality and comprises 12,000 samples across 6 activities. Lastly, for anomaly detection tasks widely applied in smart cities, such as traffic monitoring, the UCSD Peds1 subset of the UCSD anomaly detection dataset [59] is employed. This dataset includes 34 training video samples and 36 testing video samples. In addition, to evaluate Axolotl across a broader range of transformer-based and RNN-based architectures, we introduce the following benchmarks. For transformer-based LLMs, we use HellaSwag [60] for sentence completion, where the model selects the most plausible sentence ending from four options; MMLU [61] for multitask question answering, which spans 57 subjects from elementary to expert level; and ARC-c [62] for science reasoning, which requires answering challenging grade-school level questions in a multiple-choice format. For RNN-based sequence models, we adopt LibriSpeech [63] for speech recognition, a dataset derived from audiobooks that evaluates speech-to-text accuracy, and IMDB [64] for sentiment analysis, a large collection of movie reviews used for binary classification.

4.2 System Implementation

We employ two NVIDIA GeForce RTX 4090 GPUs for model training. The software framework is based on Python 3.7, PyTorch 2.1.0, and CUDA 12.1. At the inference stage, we deploy the model on edge devices including Jetson Nano, Android smartphones, iOS smartphones, and Raspberry Pi. The available RAM percentages in Table 1 emulate realistic device conditions, with smartphones having less available memory due to concurrent apps. These values represent the operational constraints for our system. To analyze the memory footprint, we break it down into three components: (1) the static memory for the assigned model block, (2) the transient memory for intermediate feature data, and (3) the negligible memory for system management logic. The dominant component is the model block itself. Axolotl manages this dominant component through a greedy allocation strategy, ensuring the assigned block consumes no more than $\frac{2}{3}$ of a device's available RAM for a stable safety margin. This guarantees a low local memory footprint. Globally, this split design enables the execution of large models that would otherwise be impossible to load. Furthermore, our resource-aware substitute models significantly reduce this memory footprint during node failures, a key advantage for operating on memory-constrained devices. For each task, we initially train a pre-trained model using conventional methods, followed by enhancing the model's robustness in edge computing scenarios using the system proposed in § 3. Notably, all training processes, including those for the original and substitute models, occur on the server side. Moreover, at the initial stage, the server configures the appropriate substitute model based on the residual computational resources of the edge node, allowing for immediate deployment upon the failure of a preceding node. Detailed configuration is as follows:

TABLE 1: Edge devices deployed in experiments.

Edge device	CPU/GPU processor	RAM	Available
Jetson Nano	NVIDIA Maxwell GPU	4GB	30-55 %
Android smartphone	Snapdragon 8 Gen 2 CPU	8GB	15-35 %
iOS smartphone	Apple A15 Bionic CPU	4GB	15-35 %
Raspberry Pi 3	Broadcom BCM2837 SoC CPU	1GB	20-50 %

- The memories of edge devices are capped to emulate real-world scenarios, as shown in Table 1. The number of nodes ranges from 6 to 10.
- We set the kernel size to 7, with a padding of 3 and a stride of 1. For image and temporal data inputs, we apply 2-d and 1-d convolutional kernels, respectively.
- We set τ to 0.5 and λ_0 to 0.1. We also determine the optimal substitute model configuration under the computational constraints of the nodes.
- We set the initial learning rate at 0.01, and define the convergence criterion as an accuracy improvement of less than 1% over 5 consecutive iterations.

5 Evaluation

In this section, we first elaborate on the experiment setup. We then conduct an overall evaluation, followed by ablation studies and a hyperparameter search for Axolotl.

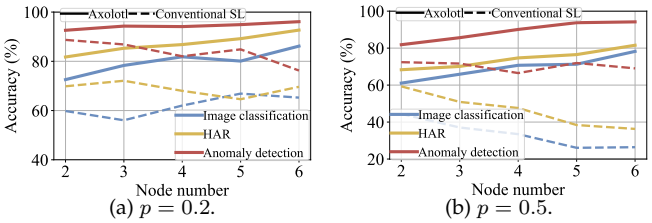
5.1 Experiment Setup

To evaluate the effectiveness of Axolotl, we select three different baselines for comparison. First, we assess the robustness of Axolotl against SPINN [18], a state-of-the-art (SOTA) SL approach that also accounts for inference process failures, such as communication disruptions. Furthermore, we compare the performance of Axolotl with AgileNN [38], another method that offloads at the feature level and provides outputs during connection interruptions. Note that for fair comparison, AgileNN, originally designed for dual-device deployment, has been tailored to yield AgileNN*, enabling computational offloading across multiple devices. Finally, we compare the performance of Axolotl with a voting strategy, where each submodel on a node operates independently of failures in other nodes. It is important to note that the selected baselines are chosen specifically because they are the most relevant works that explicitly address robustness or handle disruptions in collaborative/split inference settings, making them appropriate benchmarks for a fair and meaningful comparison.

- **SPINN** designs a robust early-exit scheme to ensure reliable execution, even in scenarios of severe connectivity disruption or cloud unavailability.
- **AgileNN*** employs explainable AI techniques to enforce feature sparsity, offloading features and corresponding computations to different devices. The significance of the features gradually diminishes along the path, resulting in a weighted output from each device.
- **Voting strategy** configures a submodel on each device in the edge scenario, which is distilled from the original model's knowledge. The final output is determined by a vote among all available nodes.

5.2 Overall Performance of Axolotl

For clarity, we only employ the CIFAR-100 dataset in the image classification task discussed in this section. We first explore the relationship between model performance and model size in the presence of node failure risk. We employ a 2-layer residual block as the basic block, with the size of the model dependent on the number of concatenated basic blocks. The proportion of node malfunction in the system

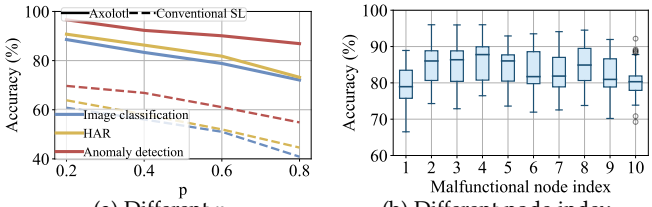


(a) $p = 0.2$. (b) $p = 0.5$.

Fig. 9: Axolotl accuracy under node malfunctions.

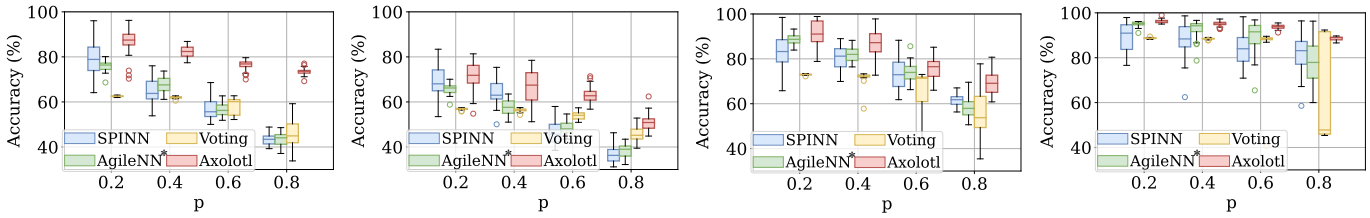
is denoted by p. In the experiments, once p is established, the number of malfunctioning nodes is rounded to the nearest integer. Figure 9 illustrates the trends in model accuracy as the number of blocks increases, under both the proposed Axolotl framework and a conventional SL setup. Remarkably, irrespective of $p = 0.2$ or $p = 0.5$, Axolotl consistently demonstrates improved performance with increasing model size. In contrast, conventional SL shows insufficient robustness: its performance can decline when the number of blocks increases, with the decline worsening as p increases. Our results validate that, given resilience to node instability, larger models can indeed perform better (assuming no overfitting occurs) and Axolotl is the first system capable of demonstrating this property. Consequently, Axolotl opens up opportunities to implement powerful models in edge computing scenarios, which are previously restricted to cloud servers, thereby mitigating communication overhead constraints.

We further evaluate how Axolotl's performance vary under different p. We configure a model consisting of 10 blocks distributes across 10 edge nodes and conduct experiments within the range of $p = 0.2$ to $p = 0.8$, as shown in Figure 10a. It is evident that when system remains relatively stable at $p = 0.2$, Axolotl already demonstrates superior performance compared to conventional SL. As p increases, this performance gap widens, reaching a significant difference when $p = 0.8$. This observation indicates that conventional SL is greatly affected by system instability, whereas Axolotl exhibits considerable robustness to such impacts. Subsequently, we evaluate the impact of malfunctions at different node positions on model performance. Specifically, we induce a malfunction in one node at a time and conduct 50 experiments per case, with the results depicted in box plots. For clarity, we present only the model performance on the image classification task, as shown in Figure 10b. The results reveal that malfunctions in nodes at two sides of the chain have the most significant impact on the model's performance. Despite varying malfunction scenarios, Axolotl's median accuracy remains consistent, demonstrating the stability of Axolotl.



(a) Different p. (b) Different node index.

Fig. 10: Impact of node malfunction factors.



(a) Image classification (CIFAR100). (b) Image classification (ImageNet). (c) HAR. (d) Anomaly detection.

Fig. 11: Comparison with baseline methods.

5.3 Superiority of Axolotl

5.3.1 Classification Accuracy

We compare the performance of Axolotl with baselines across four tasks mentioned in § 4.1 under varying p from 0.2 to 0.8. The results depicted in Figure 11 demonstrate that Axolotl consistently outperforms the baselines. Specifically, at $p = 0.2$, Axolotl achieves a median accuracies of 87.47%, 71.82%, 91.05%, and 95.90% for the four tasks, with standard deviations of 5.95%, 5.68%, 5.36%, and 0.84%, respectively. The median accuracy of Axolotl surpasses the baselines by at least 8.65%, 4.86%, 2.41%, and 2.03%, respectively. These advantages become more pronounced as p increases, with Axolotl leading by 27.83%, 12.44%, 6.28%, and 6.15% in accuracy at $p = 0.8$. Even under such severe system instability, Axolotl maintains accuracies of 73.61%, 50.86%, 69.10%, and 88.95% for the four tasks.

The robustness of Axolotl, despite potential model depth loss, is attributed to the implementation of a curriculum dropout mechanism and IPWC, which allow adjacent nodes to preserve some functions of the compromised model. Moreover, the resource-aware substitute model ensures effective information transfer, thus minimizing the depth and feature loss that substantially degrade model performance, as observed with SPINN and AgileNN*. Regarding the voting strategy, although it demonstrates notable stability when p ranges from 0.2 to 0.6, Axolotl's performance advantage is further amplified by its significantly larger parameter base. The experiments evaluate methods for real-time collaborative inference, including SL, model offloading, and knowledge distillation-based parallel schemes, which remain effective even with malfunctioned nodes. The findings demonstrate that Axolotl not only surpasses SL baselines in performance but also emerges as the most reliable method for sustaining real-time collaborative inference.

5.3.2 Operational Overhead

We begin by comparing the latency and throughput of Axolotl with SPINN and AgileNN* from an operational perspective. To ensure a fair comparison, we configured distinct hyperparameter values for each method, aiming to achieve model accuracies within a margin of $\pm 5\%$ relative to Axolotl ($p = 0.2$). As depicted in Figure 12, to achieve this level of accuracy, the baseline incurs at least twice the latency of Axolotl on each task. This is attributable to Axolotl's ability to substantially preserve model knowledge in node failure scenarios. The substitute models, which replace original blocks, consequently reduce the computational load on the neural network, thus primarily contributing to lower latency. On the other hand, as shown in Figure 13 (where the unit inf/s denotes inferences per second), the faster inference enables Axolotl to process more events per unit time, outperforming the other baselines by at least 19, 14, 81, and 35 in CIFAR-100, ImageNet, HAR, and anomaly detection, respectively.

We next compare the inference costs at the edge for Axolotl and the baseline models, ensuring that all approaches maintain an accuracy within a $\pm$ 5% range relative to Axolotl to guarantee fairness ($p = 0.2$). As illustrated in Figure 14, Axolotl exhibits lower communication overhead than SPINN and AgileNN* in four tasks, decreasing by 0.11, 4.48, 0.32, and 28.15 MB respectively. This reduction is attributed to Axolotl's retention of model information, allowing it to perform equivalent tasks with smaller models. The voting strategy exhibits minimal communication overhead due to its elimination of the need to transmit intermediate layers. Figure 14 demonstrates that Axolotl's energy consumption is between 0.45 and 77.31 mJ lower than that of other baselines. This reduction is a direct result of the previously mentioned advantages in communication load and model size.

5.4 Scalability

Beyond the limited-device scenarios, we also evaluate Axolotl's scalability to a larger set of devices. Specifically, we partition a ResNet composed of 12 blocks into several configurations: 1×12, 2×6, 3×4, 4×3, and 6×2, representing the implementation of Axolotl across different distributions of computational resources. In the experiments, p is set to 0.2, 0.4, 0.6, or 0.8 in separate tests. As illustrated in Figure 16, although the highest accuracy in each of the four tasks decreases as p increases sharply, it still remains above 74.43%, 54.03%, 74.13%, and 88.41% when $p \leq 0.4$. In the 1×12 configuration, the model experiences the greatest instability, resulting in relatively lower accuracy, yet the difference from other configurations does not exceed 5.51%, 8.36%, 6.89%, and 3.11%. These results demonstrate that whether the edge scenario comprises multiple devices with lower computational power or fewer devices with higher power, Axolotl effectively addresses node malfunctions caused by device instability, underscoring its robust scalability.

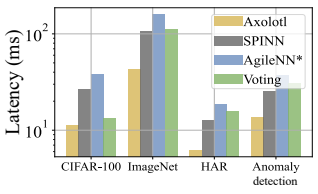


Fig. 12: Latency.

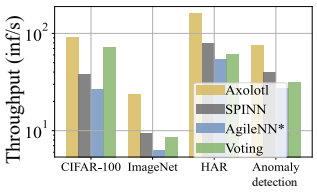


Fig. 13: Throughput.

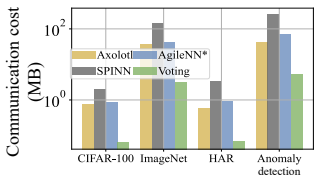


Fig. 14: Communication cost.

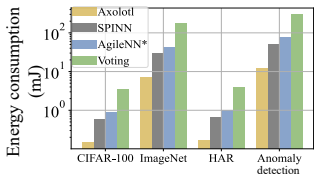


Fig. 15: Energy consumption.

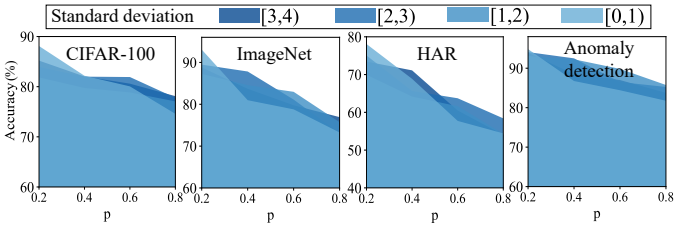


Fig. 17: Scaling to heterogeneous devices.

We also evaluate the impact of node balance on the performance of Axolotl. Specifically, our experiments include a broader range of edge devices, comprising smartphones, smartwatches, and edge servers, where device combinations for model deployment are selected randomly. To quantify the degree of node imbalance, we calculate the standard deviations of device computational power for different device combinations and normalize these by the size of the smallest blocks in the models. For clarity, these combinations are categorized by standard deviation into four intervals: [0,1), [1,2), [2,3), [3,4), and average accuracies are computed for each interval. Figure 17 demonstrates that Axolotl maintains robust performance across four tasks even when deployed on multiple unbalanced nodes. Furthermore, even at $p = 0.8$, the variation in accuracy among different levels of node imbalance does not exceed 3.53%, 4.37%, 3.84%, and 4.00%. This indicates that Axolotl is not limited to identical laboratory devices but can also effectively scale to various real-world edge scenarios with node imbalance.

5.5 Curriculum Dropout Mechanism

In § 3.1, we have introduced a hierarchical curriculum dropout strategy. Here, we conduct ablation studies on both the dropout intensity level and the consistency level. To ensure fairness, the ablated models are still subjected to various block failures during training; however, the dropout intensity and consistency are determined randomly. We set p values from 0.2 to 0.8 and average the results of 50 replicates for each experiment.

The results depicted in Figure 18 indicate that, across all tasks, the dropout strategy with two levels consistently achieves the highest accuracy for all values of p. At $p = 0.2$, the full dropout achieves accuracies of 86.90%, 72.3%, 90.56%, and 96.14% across four tasks, surpassing the model without any of the two levels, which show accuracies of 80.50%, 69.87%, 84.27%, 91.60% and 72.64%, 66.38%, 79.24%, 89.33%. In contrast, the ablated strategy using random strategies only achieves 60.10%, 54.57%, 72.19%, and 84.03%. As p increases, the performance disparity also grows, with the lead of the full strategy exceeding 8.34%, 3.98%, 15.54%, and 7.71% when p reaches 0.8. This phenomenon can be attributed to block dropout, which significantly alters the loss landscape and causes fluctuations in the optimization direction, even when the supervised task's loss surface is inherently smooth. The fine-grained control provided by Axolotl at different levels allows for a progressive modification of these changes, stabilizing the exploration process toward the optima. It is noteworthy that retaining the dropout intensity results in better model performance than retaining the dropout consistency, which is the rationale behind this design choice.

To further evaluate the stability provided by the curriculum dropout mechanism, we calculate the Kullback-Leibler (KL) divergence among the outputs of ten models trained using the same strategy. We conduct experiments for each strategy across various tasks, and present the distributions of these divergences in Figure 19. It is evident that under the curriculum dropout mechanism, the KL divergence among model probability distributions is significantly lower than that observed when curriculum dropout is applied at a single level or under a completely random strategy. This indicates minimal model variability and demonstrates enhanced training stability with the curriculum dropout mechanism. This stability arises because, in contrast to random strategies, curriculum dropout controls the variations among model configurations, thereby guiding the models towards a more stable optimization trajectory.

5.6 Inverse-Proximal Weight Consolidation

We evaluate Axolotl's IPWC against the SOTA algorithm elastic weight consolidation (EWC) [51], both designed to address catastrophic forgetting. While each method employs the Fisher information matrix to assess parameter significance, IPWC is distinctively tailored for node malfunction scenarios. For comprehensive analysis, conventional gradient descent (GD) is also included as a baseline. The employed model is structured into six blocks, with $p = 0.2-0.8$ during the testing phase. As illustrated in Figure 20, using CIFAR-100 as an example, when p is set to 0.8, the accuracy of IPWC is 76.18%, whereas that of EWC is 65.54%. Both

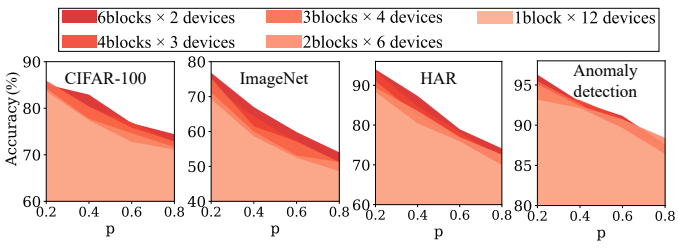


Fig. 16: Scaling to different number of devices.

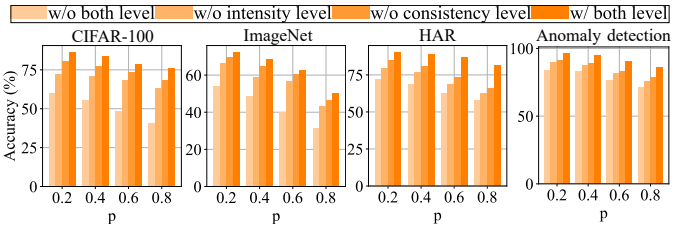


Fig. 18: Effects of hierarchical dropout strategy.

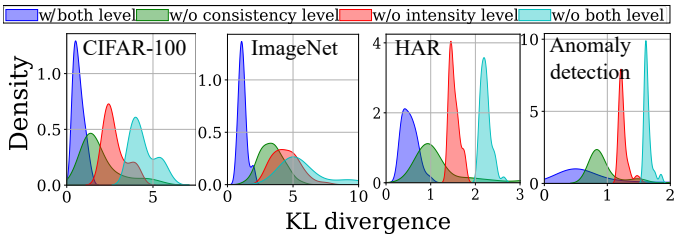


Fig. 19: Comparison of KL divergence distribution.

these methods considerably surpass the performance of GD, which only achieves an accuracy of 38.22%. This superior performance of IPWC and EWC is consistently observed across the other tasks depicted in the figures. Experimental results demonstrate that IPWC effectively mitigates catastrophic forgetting and prevents excessive updates across different dropout configurations. Consequently, the performance of the model remains unaffected by the sequence of block failures encountered during training. Furthermore, unlike EWC, which applies a uniform regularization weight across all parts of the model, IPWC allows for more relaxed constraints around dropout blocks. This flexibility enables adjacent blocks to share functionalities, introducing redundancy that enhances the model's resilience to block failures during the inference phase.

To intuitively demonstrate the effectiveness of IPWC in mitigating catastrophic forgetting, we repeat the experiment in Figure 3 after implementing IPWC, with the results presented in Figure 21. Notably, in Figure 21a, the model parameters across different configurations do not exhibit the distinct clusters observed in Figure 3a, indicating that the model retains the knowledge of other configurations while learning a new one. Furthermore, Figure 21b illustrates that although model accuracy still experiences fluctuations due to system instability, there is a clearly observable upward trend. This suggests that under the help of IPWC, the model effectively learns different patterns of node malfunction. As the configuration index decreases, reflecting configurations trained earlier, the accuracy exhibits only minor fluctuations, with IPWC consistently demonstrating superior performance over EWC. This indicates that IPWC is better for handling catastrophic forgetting due to model configuration change, as it effectively monitors changes and utilizes adjacent areas to mitigate the impact of these changes.

5.7 Resource-Aware Substitute Model

We conduct an ablation study on the substitute model, comparing it against a basic up-sampling and down-sampling

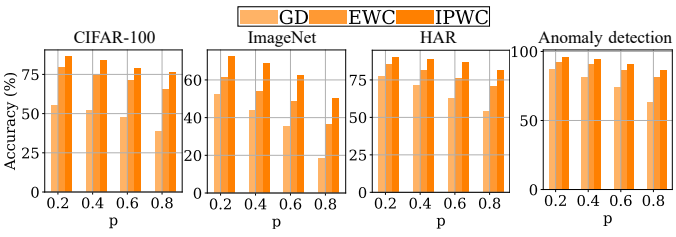
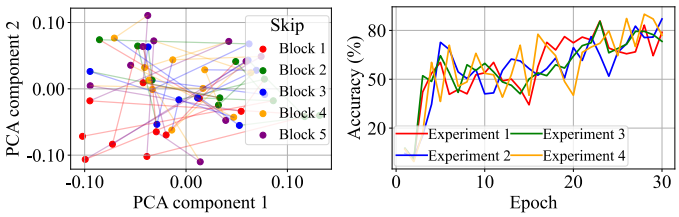


Fig. 20: Performance comparison of different catastrophic forgetting mitigation methods.



(a) Model weight preservation. (b) Stable accuracy improvement.

Fig. 21: Effects of IPWC.

method, each ensuring smooth model inference. As illustrated in Figure 22, the substitute model records an accuracy exceeding 14.92% when $p = 0.2$ across all tasks. With increasing system instability, this discrepancy widens, culminating in a difference ranging from 23.17% to 54.33% at $p = 0.8$. The results suggest that although simple reshaping can preserve model operation, it leads to significant feature mismatch, thereby degrading model performance. In contrast, despite its significantly smaller size compared to the original model block, the substitute model effectively retains critical feature transmission, thus sustaining model performance.

We then evaluate the impacts of kernel dilation and group pruning on the substitute model's size and performance with $p = 0.2$, as shown in Figure 23. Note that the horizontal axis represents the model size relative to the full substitute model. This substitute model is, in turn, less than 60.5% of the original block size. The results indicate that utilizing either strategy in isolation to reduce model parameters results in a drastic decrease in performance across four tasks. When the kernel size of the substitute model is set to 51.02% of the full substitute model's kernel size, the accuracy drops to below 87.34%, 74.01%, 88.07%, and 94.78% for the four respective tasks. Furthermore, when the channel number is reduced to 50.00% of the full substitute model, the accuracy falls below 84.19%, 70.74%, 87.66%, and 93.34%, and diminishes rapidly as the model size continues to decrease.

Conversely, when our combined method is applied, this trend of performance degradation is significantly slowed. With the size of the substitute model reduced to only 50.51%, the accuracy still exceeds 89.47%, 78.10%, 92.81%, and 96.44%. This suggests that reduction in model size from any single dimension leads to rapid information loss, rendering the maintenance of complexity in another dimension ineffective. Therefore, it becomes imperative to modify both dimensions simultaneously to find an optimal substitute model that maximally preserves information within a specified size constraint. Notably, even when the model size decreases to only 3.13% of the full block, the accuracy remains at 76.70%, 60.52%, 80.33%, and 84.18%. At this

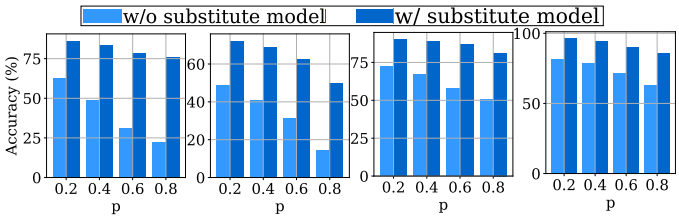


Fig. 22: Ablation study of substitute model.

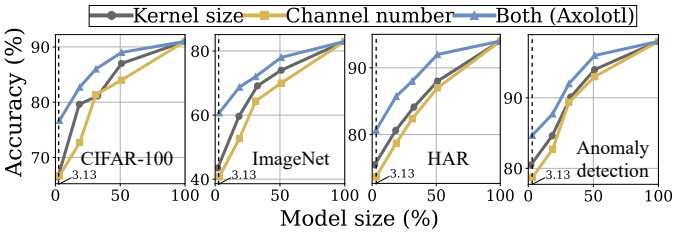


Fig. 23: Impact of substitute model size on accuracy.

point, the size of substitute model deployed on the successor node is only in the range from 0.22 to 0.51MB, a burden that is entirely manageable for edge devices.

5.8 Hyper-parameter Search

5.8.1 Pace in Curriculum Dropout Mechanism

In § 3.1, we mention that after determining the number of dropout blocks, the task difficulty is further influenced by the similarity between consecutive dropout blocks. The parameter τ controls the decay of this similarity. We search for the optimal τ within the range from 0.1 to 0.5 for each task, and conduct a separate search for each task. As shown in Figure 24a, the optimal τ values for CIFAR-100, ImageNet, HAR, and anomaly detection tasks are 0.2, 0.1, 0.4, and 0.3, respectively. For the image classification tasks, due to the larger number of classes, a smaller τ is required to ensure a lower difficulty increment. While anomaly detection has fewer categories than RF-based HAR, it requires a smaller threshold τ due to the higher complexity of its input data.

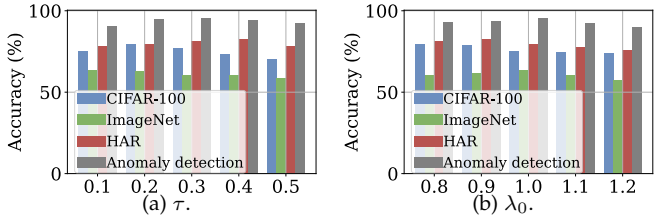


Fig. 24: Hyper-parameter searching.

5.8.2 Regularization Strength in IPWC

In § 3.2, we compute the regularization term based on λ_0 to determine the extent to which the model retains knowledge of previous dropout configurations, as well as the variation in regularization strength between adjacent blocks. Similarly, we search for the optimal λ_0 within the range from 0.8 to 1.2 for the four tasks. As shown in Figure 24b, the optimal λ_0 values for CIFAR-100, ImageNet, HAR, and anomaly detection are 0.8, 1.0, 0.9, and 1.0, respectively.

5.9 Generalizability to Diverse Model Architectures

To validate the broader applicability of Axolotl beyond CNNs, we extend our evaluation to transformer and RNN architectures, as Axolotl's core design is model-agnostic. While its fault-tolerance principles are general, applying Axolotl to new architectures requires tailoring the model splitting strategy and the generation of substitute models (as introduced in § 3.3). For transformer-based models, we split the architecture by assigning each edge device a self-attention module followed by a feed-forward network (FFN), and the resource-aware substitute models are constructed using multi-head attention pruning and dilated self-attention. For RNNs, we partition a stacked architecture vertically by assigning a distinct group of recurrent layers to each device, and the resource-aware substitute models are constructed via structural sparsification of recurrent weight matrices and dilated recurrent skip connections.

Model	Benchmark	Acc. (Stable)	Acc. (p=0.4)	Drop
LLaMA-1B	HellaSwag	41.2%	37.1%	4.1%
	MMLU	49.3%	45.8%	3.5%
	ARC-c	59.4%	55.6%	3.8%
LLaMA-3B	HellaSwag	69.8%	65.3%	4.5%
	MMLU	63.4%	59.5%	3.9%
	ARC-c	78.6%	74.4%	4.2%
RNN	LibriSpeech	94.1%	91.4%	2.7%
	IMDB	92.5%	91.2%	1.3%

TABLE 2: Performance on transformer and RNN models.

For the transformer-based evaluation, we use the LLaMA architecture and assess its performance on three standard benchmarks: HellaSwag, MMLU, and ARC-c. All experiments are conducted in an unstable edge environment with a node failure probability of $p = 0.4$. As shown in Table 2, Axolotl enables the LLaMA-1B model to maintain high performance with only minor accuracy drops of 4.1%, 3.5%, and 3.8% across the three tasks compared to the stable, single-device baseline. Similarly, the larger LLaMA-3B model experiences minimal degradation of 4.5%, 3.9%, and 4.2%. To further demonstrate Axolotl's generality, we evaluate its performance on RNNs using the LibriSpeech and IMDB datasets, and show the results in As shown in Table 2. Under the same failure condition ($p = 0.4$), the RNN model shows a performance degradation of only 2.7% on LibriSpeech and 1.3% on IMDB). These findings confirm that Axolotl's fault-tolerant capabilities generalize effectively, maintaining high performance for distributed inference across diverse architectures.

6 Discussion

To provide a complete understanding of Axolotl, we discuss key environmental and operational factors that define the system's optimal performance boundaries. The system is designed for robust performance under common partial or sequential node failures, but its operational limits can be met in extreme scenarios, such as a catastrophic, simultaneous collapse of the majority of network nodes, which can destabilize the model's optimization. Similarly, performance is linked to the careful calibration of system parameters (τ and λ_0) relative to task complexity, ensuring a proper balance between plasticity and stability. This principle extends to the hardware layer; on extremely resource-scarce devices, the overhead from excessive model fragmentation may influence the overall efficiency of the distributed approach. Finally, the system's failure detection is optimized for outright node failures. In environments characterized primarily by high-latency "straggler" nodes rather than disconnections, the heartbeat protocol may interpret a severely delayed node as failed, which highlights an opportunity for future work in more nuanced network-health assessments.

7 Conclusion

In this paper, we introduce Axolotl, a novel split ML inference system designed for edge environments. Axolotl enables the inference of large models by effectively distributing them across multiple nearby personal edge devices. By incorporating novel dropout mechanisms, innovative minimal substitution models, and curriculum learning strategies, Axolotl addresses the key challenges associated with split ML, ensuring resilience, fault tolerance, and adaptability in the inference pipeline. Through extensive experimental validation, we have demonstrated the effectiveness and robustness of Axolotl on various deep learning networks and tasks in edge environments. By tackling the challenges surrounding node failure and adaptability, our system represents a significant step towards more reliable and efficient distributed learning implementations for edge devices.

Acknowledgments

The study is supported by Shenzhen Science and Technology Program (No. 20231120215201001) and National Natural Science Foundation of China (No. 62502191).

References

[1] T. Srivastava, P. Khanna, S. Pan, P. Nguyen, and S. Jain, "MuteIt: Jaw Motion Based Unvoiced Command Recognition Using Earable," *Proc. of ACM IMWUT*, vol. 6, no. 3, pp. 1–26, 2022.

[2] Y. Shen, J. Shao, X. Zhang, Z. Lin, H. Pan, D. Li, J. Zhang, and K. B. Letaief, "Large Language Models Empowered Autonomous Edge AI for Connected Intelligence," *IEEE Communications Magazine*, 2024.

[3] H. Li, H. Chen, C. Xu, Z. Li, H. Zhang, X. Qian, D. Li, M.-c. Huang, and W. Xu, "NeuralGait: Assessing Brain Health Using Your Smartphone," *Proc. of ACM IMWUT*, vol. 6, no. 4, pp. 1–28, 2023.

[4] A. Benazir, Z. Xu, and F. X. Lin, "Speech Understanding on Tiny Devices with A Learning Cache," in *Proc. of the 22nd ACM MobiSys*, 2024, pp. 425–437.

[5] I. McGraw, R. Prabhavalkar, R. Alvarez, M. G. Arenas, K. Rao, D. Rybach, O. Alsharif, H. Sak, A. Gruenstein, F. Beaufays *et al.*, "Personalized speech recognition on mobile devices," in *Proc. of IEEE ICASSP*. IEEE, 2016, pp. 5955–5959.

[6] L. Yang, X. Chen, X. Jian, L. Yang, Y. Li, Q. Ren, Y.-C. Chen, G. Xue, and X. Ji, "Remote Attacks on Speech Recognition Systems Using Sound from Power Supply," in *Proc. of the 32nd USENIX Security*, 2023, pp. 4571–4588.

[7] J. Yi, S. Choi, and Y. Lee, "EagleEye: Wearable Camera-Based Person Identification in Crowded Urban Spaces," in *Proc. of the 26th ACM MobiCom*, 2020, pp. 1–14.

[8] M. Jiang, S. Liu, Y. Lyu, and Y. Zhou, "Face-Based Authentication Using Computational Secure Sketch," *IEEE Transactions on Mobile Computing*, vol. 22, no. 12, pp. 7172–7187, 2022.

[9] Q. Cao, N. Weber, N. Balasubramanian, and A. Balasubramanian, "DeQA: On-Device Question Answering," in *Proc. of the 17th ACM MobiSys*, 2019, pp. 27–40.

[10] A. Majumdar, A. Ajay, X. Zhang, P. Putta, S. Yenamandra, M. Henaff, S. Silwal, P. Mcvay, O. Maksymets, S. Arnaud *et al.*, "OpenEQA: Embodied Question Answering in the Era of Foundation Models," in *Proc. of IEEE/CVF CVPR*, 2024, pp. 16 488–16 498.

[11] Y. Li, H. Wang, Q. Jin, J. Hu, P. Chemerys, Y. Fu, Y. Wang, S. Tulyakov, and J. Ren, "SnapFusion: Text-to-Image Diffusion Model on Mobile Devices within Two Seconds," *Proc. of NeurIPS*, vol. 36, 2024.

[12] C. Li, Z. Liu, Y. Yao, Z. Cao, M. Zhang, and Y. Liu, "Wi-Fi See It All: Generative Adversarial Network-augmented Versatile Wi-Fi Imaging," in *Proc. of the 18th ACM SenSys*, 2020, pp. 436–448.

[13] C. E. Leiserson, N. C. Thompson, J. S. Emer, B. C. Kuszmaul, B. W. Lampson, D. Sanchez, and T. B. Schardl, "There's Plenty of Room at The Top: What Will Drive Computer Performance After Moore's law?" *Science*, vol. 368, no. 6495, p. eaam9744, 2020.

[14] M. Xu, J. Liu, Y. Liu, F. X. Lin, Y. Liu, and X. Liu, "A First Look at Deep Learning Apps on Smartphones," in *Proc. of ACM WWW*, 2019, pp. 2125–2136.

[15] E. Liberis, Ł. Dudziak, and N. D. Lane, "μnas: Constrained Neural Architecture Search for Microcontrollers," in *Proceedings of the 1st Workshop on Machine Learning and Systems*, 2021, pp. 70–79.

[16] X. Li, Y. Li, Y. Li, T. Cao, and Y. Liu, "FlexNN: Efficient and Adaptive DNN Inference on Memory-Constrained Edge Devices," in *Proc. of the 30th ACM MobiCom*, 2024, pp. 709–723.

[17] H. Wen, Y. Li, Z. Zhang, S. Jiang, X. Ye, Y. Ouyang, Y. Zhang, and Y. Liu, "AdaptiveNet: Post-deployment Neural Architecture Adaptation for Diverse Edge Environments," in *Proc. of the 29th ACM MobiCom*, 2023, pp. 1–17.

[18] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "SPINN: Synergistic Progressive Inference of Neural Networks over Device and Cloud," in *Proc. of the 26th MobiCom*, 2020, pp. 1–15.

[19] M. Almeida, S. Laskaridis, S. I. Venieris, I. Leontiadis, and N. D. Lane, "DynO: Dynamic Onloading of Deep Neural Networks from Cloud to Device," *ACM Transactions on Embedded Computing Systems*, vol. 21, no. 6, pp. 1–24, 2022.

[20] Apple, "Apple CloudKit," https://developer.apple.com/documentation/cloudkit, 2014.

[21] Google, "Google Firebase," https://firebase.google.com, 2014.

[22] R. Hadidi, J. Cao, M. S. Ryoo, and H. Kim, "Toward Collaborative Inferencing of Deep Neural Networks on Internet-of-Things Devices," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4950–4960, 2020.

[23] Y. Wang, C. Yang, S. Lan, L. Zhu, and Y. Zhang, "End-Edge-Cloud Collaborative Computing for Deep Learning: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials*, 2024.

[24] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "CoEdge: Cooperative DNN Inference With Adaptive Workload Partitioning Over Heterogeneous Edge Devices," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 595–608, 2020.

[25] X. Guo, A. D. Pimentel, and T. Stefanov, "Automated Exploration and Implementation of Distributed CNN Inference at the Edge," *IEEE Internet of Things Journal*, vol. 10, no. 7, pp. 5843–5858, 2023.

[26] O. Gupta and R. Raskar, "Distributed Learning of Deep Neural Network over Multiple Agents," *Journal of Network and Computer Applications*, vol. 116, pp. 1–8, 2018.

[27] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative Intelligence between the Cloud and Mobile Edge," *Proc. of ACM ASPLOS*, vol. 45, no. 1, pp. 615–629, 2017.

[28] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "SplitFed: When Federated Learning Meets Split Learning," in *Proc. of AAAI*, vol. 36, no. 8, 2022, pp. 8485–8493.

[29] S. Wang, X. Zhang, H. Uchiyama, and H. Matsuda, "HiveMind: Towards Cellular Native Machine Learning Model Splitting," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 2, pp. 626–640, 2021.

[30] Z. Li, W. Wu, S. Wu, and W. Wang, "Adaptive Split Learning over Energy-Constrained Wireless Edge Networks," *Proc. of IEEE INFOCOM*, 2024.

[31] S. Zhang, Y. Li, X. Liu, S. Guo, W. Wang, J. Wang, B. Ding, and D. Wu, "Towards Real-time Cooperative Deep Inference over the Cloud and Edge End Devices," *Proc. of ACM UbiComp*, vol. 4, no. 2, pp. 1–24, 2020.

[32] K. N. Khan, A. Khalid, Y. Turkar, K. Dantu, and F. Ahmad, "VRF: Vehicle Road-side Point Cloud Fusion," in *Proc. of the 22nd ACM MobiSys*, 2024, pp. 547–560.

[33] L. Zhang, L. Chen, and J. Xu, "Autodidactic Neurosurgeon: Collaborative Deep Inference for Mobile Edge Intelligence via Online Learning," in *Proc. of ACM WWW*, 2021, pp. 3111–3123.

[34] B. Xie, M. Cui, D. Ganesan, and J. Xiong, "Wall Matters: Rethinking the Effect of Wall for Wireless Sensing," *Proc. of ACM IMWUT*, vol. 7, no. 4, pp. 1–22, 2024.

[35] T. Zheng, A. Li, Z. Chen, H. Wang, and J. Luo, "AutoFed: Heterogeneity-Aware Federated Multimodal Learning for Robust Autonomous Driving," in *Proc. of the 29th ACM MobiCom*, 2023, pp. 1–15.

[36] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic Adaptive DNN Surgery for Inference Acceleration on the Edge," in *Proc. of IEEE INFOCOM*. IEEE, 2019, pp. 1423–1431.

[37] T. Boroushaki, M. Lam, L. Dodds, A. Eid, and F. Adib, "Augmenting Augmented Reality with {Non-Line-of-Sight} Perception," in *Proc. of the 20th USENIX NSDI 23*, 2023, pp. 1341–1358.

[38] K. Huang and W. Gao, "Real-time Neural Network Inference on Extremely Weak Devices: Agile Offloading with Explainable AI," in *Proc. of the 28th MobiCom*, 2022, pp. 200–213.

[39] C. Pei, Z. Wang, Y. Zhao, Z. Wang, Y. Meng, D. Pei, Y. Peng, W. Tang, and X. Qu, "Why it takes so long to connect to a wifi access point," in *Proc. of IEEE INFOCOM*. IEEE, 2017, pp. 1–9.

[40] S. R. K. Somayaji, M. Alazab, M. Manoj, A. Bucchiarone, C. L. Chowdhary, and T. R. Gadekallu, "A Framework for Prediction and Storage of Battery Life in IoT Devices using DNN and Blockchain," in *2020 IEEE Globecom Workshops*. IEEE, 2020, pp. 1–6.

[41] S. Kang, H. Choi, S. Park, C. Park, J. Lee, U. Lee, and S.-J. Lee, "Fire in Your Hands: Understanding Thermal Behavior of Smartphones," in *Proc. of the 25th ACM MobiCom*, 2019, pp. 1–16.

[42] J. Switzer, G. Marcano, R. Kastner, and P. Pannuto, "Junkyard Computing: Repurposing Discarded Smartphones to Minimize Carbon," in *Proc. of the 28th ACM ASPLOS*, 2023, pp. 400–412.

[43] F. A. Salaht, F. Desprez, and A. Lebre, "An Overview of Service Placement Problem in Fog and Edge Computing," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–35, 2020.

[44] S. Garg, K. Kaur, G. Kaddoum, P. Garigipati, and G. S. Aujla, "Security in IoT-driven Mobile Edge Computing: New Paradigms, Challenges, and Opportunities," *IEEE Network*, vol. 35, no. 5, pp. 298–305, 2021.

[45] X. Zhang and S. Debroy, "Resource management in mobile edge computing: a comprehensive survey," *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1–37, 2023.

[46] X. Kong, X. Liu, J. Gu, Y. Qiao, and C. Dong, "Reflash Dropout in Image Super-Resolution," in *Proc. of IEEE/CVF CVPR*, 2022, pp. 6002–6012.

[47] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[48] M. McCloskey and N. J. Cohen, "Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem," in *Psychology of learning and motivation*. Elsevier, 1989, vol. 24, pp. 109–165.

[49] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum Learning," in *Proc. of the 26th ICML*, 2009, pp. 41–48.

[50] A. Graves, M. G. Bellemare, J. Menick, R. Munos, and K. Kavukcuoglu, "Automated Curriculum Learning for Neural Networks," in *Proc. of the 34th ICML*. Pmlr, 2017, pp. 1311–1320.

[51] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming Catastrophic Forgetting in Neural Networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[52] C. Jose, M. Cissé, and F. Fleuret, "Kronecker Recurrent Units," in *Proc. of the 35th ICML*. PMLR, 2018, pp. 2380–2389.

[53] S. Chang, Y. Zhang, W. Han, M. Yu, X. Guo, W. Tan, X. Cui, M. Witbrock, M. A. Hasegawa-Johnson, and T. S. Huang, "Dilated Recurrent Neural Networks," *Proc. of NeurIPS*, vol. 30, 2017.

[54] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The Long-Document Transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[55] E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov, "Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned," in *Proc. of the 57th ACL*, 2019, pp. 5797–5808.

[56] A. Krizhevsky, G. Hinton *et al.*, "Learning Multiple Layers of Features from Tiny Images," 2009.

[57] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *Proc. of IEEE CVPR*. Ieee, 2009, pp. 248–255.

[58] S. Ding, Z. Chen, T. Zheng, and J. Luo, "RF-Net: A Unified Meta-Learning Framework for RF-enabled One-Shot Human Activity Recognition," in *Proc. of the 18th ACM SenSys*, 2020, pp. 517–530.

[59] W. Li, V. Mahadevan, and N. Vasconcelos, "Anomaly Detection and Localization in Crowded Scenes," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 1, pp. 18–32, 2013.

[60] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "HellaSwag: Can a Machine Really Finish Your Sentence?" in *Proc. of the 57th ACL*, 2019, pp. 4791–4800.

[61] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring Massive Multitask Language Understanding," *arXiv preprint arXiv:2009.03300*, 2020.

[62] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think You Have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge," *arXiv preprint arXiv:1803.05457*, 2018.

[63] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an ASR Corpus Based on Public Domain Audio Books," in *Proc. of IEEE ICASSP*. IEEE, 2015, pp. 5206–5210.

[64] A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning Word Vectors for Sentiment Analysis," in *Proc. of the 49th ACL*, 2011, pp. 142–150.

Yuxuan Weng is a research assistant at the Southern University of Science and Technology. He received his Bachelor's degree from Sun Yat-sen University and his Master's degree from the Hong Kong University of Science and Technology. His research interests include mobile computing, RF sensing, multimodal sensing, and machine learning.

Tianyue Zheng is an Assitant Professor and Ph.D. Supervisor at the Southern University of Science and Technology (SUSTech). He received his Ph.D. degree from Nanyang Technological University, Singapore, in 2023. His research interests focus on mobile computing and multimodal sensing. He serves program committee members for several international conferences and a reviewer for multiple journals.

Zhe Chen received the PhD degree with honor in computer science from Fudan University, Shanghai, China, in 2018. He received the Doctoral Dissertation Award from ACM SIGCOMM China, in 2019. He is a research fellow in Nanyang Technological University, Singapore. His research interests include designing and implementing system for practical large-scale MU-MIMO systems, deep learning, and Internet-of-Things application.

Menglan Hu received the B.E. degree in electronic and information engineering from Huazhong University of Science and Technology, Wuhan, China, in 2007, and the Ph.D. degree in electrical and computer engineering from the National University of Singapore, Singapore, in 2012. He is currently an Associate Professor with the School of Electronic Information and Communications, Huazhong University of Science and Technology.

Jun Luo received his BS and MS degrees in Electrical Engineering from Tsinghua University, China, and the Ph.D. degree in Computer Science from EPFL, Lausanne, Switzerland. From 2006 to 2008, he has worked as a postdoctoral research fellow in the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Canada. In 2008, he joined the faculty of Nanyang Technological University in Singapore, where he is currently a full professor.