Distributed Deep Neural Network (DDNN) Analysis for Classification

Konstantinos Nikolos

May 20, 2025

Abstract

In this work, we study a Distributed Deep Neural Network (DDNN) designed for image classification, focusing primarily on the decision-making process that determines whether a given sample should be processed locally or offloaded to a remote cloud network. The goal of this decision is to optimize both inference time and computational cost while maintaining high classification accuracy.

Specifically, we investigate an **optimized offload mechanism** based on the calculation of per-sample cost values, referred to as b_k. These values quantify the benefit of processing a sample in the cloud versus classifying it locally. We aim to refine this cost-based decision rule to achieve better performance compared to simpler, entropy-based methods commonly used in prior research.

Our experiments explore different formulations of the b_k metric and assess their impact on the effectiveness of the offload mechanism. We compare our approach against traditional uncertainty-based thresholds and analyze scenarios where our optimized rule succeeds or fails.

1 Introduction

In recent years, Distributed Deep Neural Networks (DDNNs) have emerged as an appealing solution for reducing latency and bandwidth overheads in real-time inference. By splitting the model between a local network (edge device) and a deeper, more powerful cloud network, one can "early-exit" samples that appear easy and offload only the more challenging cases.

Nevertheless, a crucial question remains: *how do we decide which samples to keep local and which to offload?* Traditional solutions rely on normalized entropy of the local softmax distribution [**?**] or run multiple forward passes

with dropout to estimate local uncertainty. While entropy-based approaches can work well in classification scenarios, they sometimes fail to capture nuances about *which* samples are truly more suitable for the local vs. the cloud network, particularly when both networks can be correct but differ in confidence.

Our primary focus is on a more general **offload mechanism** that we train to separate "easy" from "hard" samples. In brief, we define a cost-based metric b_k for each training sample k, reflecting how much "benefit" the cloud classifier yields over the local one. We then learn a binary model that decides whether to offload a sample based on the local features, or the whole sample figure. This approach *integrates seamlessly* with the joint training of the local and cloud networks, ensuring synergy between the feature representations and the final offloading decisions.

This methodology is inspired by the work of Spyropoylos et al on "Fast Edge Resource Scaling with Distirbuted DNN" [**?**], where an optimized offload mechanism was introduced for regression-based resource allocation in 5G slicing. Unlike their work, which focused on time-series prediction and network optimization, we adapt the optimized rule framework to an image classification task. We specifically refine the cost function b_k and analyze its effectiveness compared to simpler entropy-based decisions, highlighting cases where it succeeds or fails in real-world classification settings.

2 Model Structure and Parameters

This section describes the components of the DDNN, including the local and cloud networks, optimization parameters and activation function, tailored to effectively classify the Fashion-MNIST dataset.

2.1 CIFAR-10 Dataset and Architecture Choice

In this work, we utilize the CIFAR-10 dataset, which consists of 60,000 color images divided into 10 distinct classes, with each image being 32x32 pixels in size. The dataset is split into 50,000 images for training and 10,000 images for testing. In our implementation, we further split 5,000 images from the training set to serve as the validation set, leaving 45,000 images for actual training.

We chose the CIFAR-10 dataset as it presents a more challenging classification problem compared to simpler datasets, such as MNIST, due to the higher complexity and variability of the images. This increased difficulty allows for a clearer demonstration of the advantages of the cloud network over

the local classifier in terms of classification accuracy.

2.2 Network Architecture

The distributed model in our system leverages 2D Convolutional Neural Networks (CNNs), which are well-suited for image classification tasks due to their ability to capture spatial hierarchies in visual data. The network is split into a local network and a cloud network, each responsible for different aspects of feature extraction and classification.

Our architecture is inspired by the work of Teerapittayanon et al. on **BranchyNet** [**?**], which proposed a deep CNN with multiple early-exit points, allowing efficient inference at different levels of feature extraction. This approach ensures that samples which can be classified with high confidence at an early stage do not need to pass through the entire deep network, thus reducing computation time and resource usage.

For our specific problem, we aimed to design a network that allows for clear decision boundaries between local and cloud processing. Ideally, the local network should be able to classify simpler samples with at least **10**% lower accuracy than the cloud network, ensuring a meaningful performance gap that justifies offloading decisions. In our case, we achieved a **15**% accuracy difference, making it possible to quantitatively evaluate whether our offload mechanism correctly assigns samples to the most appropriate classifier.

2.2.1 Local Network

The local network consists of two primary components: the *Local Feature Extractor* and the *Local Classifier*. The Local Feature Extractor includes a convolutional layer with 32 filters, each utilizing a 3x3 kernel size for filtering. This layer serves as the primary feature extraction stage, capturing essential spatial details in the input image. Following the feature extraction, the Local Classifier consists of a dense layer with 64 filters and an output layer of 10 neurons, each corresponding to a unique class in the classification task.

2.2.2 Cloud Network

The cloud network architecture consists of consecutive convolutional layers with increasing numbers of filters, followed by batch normalization and max pooling. Specifically, the network starts with two convolutional layers with 32 filters and a kernel size of 3x3, followed by a 2x2 Max Pooling layer. This is followed by two additional convolutional layers with 64 filters and a

final convolutional layer with 128 filters, also followed by a 2x2 Max Pooling layer. The output of the convolutional layers is then passed to a series of dense (fully connected) layers: a dense layer with 256 neurons, another with 64 neurons, and finally an output layer with 10 neurons corresponding to the classification task.

Dropout=0.25 and Batch Normalization were used to prevent overfitting in this deeper setting. Finally, Leaky Relu was chosen as the activation function instead of Relu, performing significantly better due to its proper handling of vanishing gradients (it takes into consideration negative gradients too).

2.2.3 Activation Function

In both the local and cloud networks, the Leaky Rectified Linear Unit (ReLU) activation function is applied after each convolutional and dense layer due to its effectiveness in CNN architectures.

2.3 Data Flow from Local to Cloud

The flow of data through our Distributed Deep Neural Network (DDNN) is as follows:

1. Input Processing: A sample image is first processed by the Local Feature Extractor, which applies a convolutional transformation to generate an intermediate representation.

2. Local Classification Attempt: The extracted features are passed to the Local Classifier, which produces an initial class prediction.

3. Offloading Decision: Based on the offload mechanism (entropy-based or learned decision function), the system determines whether the sample should be processed locally or offloaded to the cloud.

    - If the sample is classified with high confidence locally, the result is used immediately.
    - If the sample is classified with low confidence, the extracted feature representation is transmitted to the cloud.

4. Cloud Processing (if required): The cloud network receives the extracted features and processes them through deeper convolutional layers, which refine the feature representation before classification.

5. Final Prediction: The system outputs the final classification label, based on the local classifier (if confident) or cloud classifier (if offloaded).

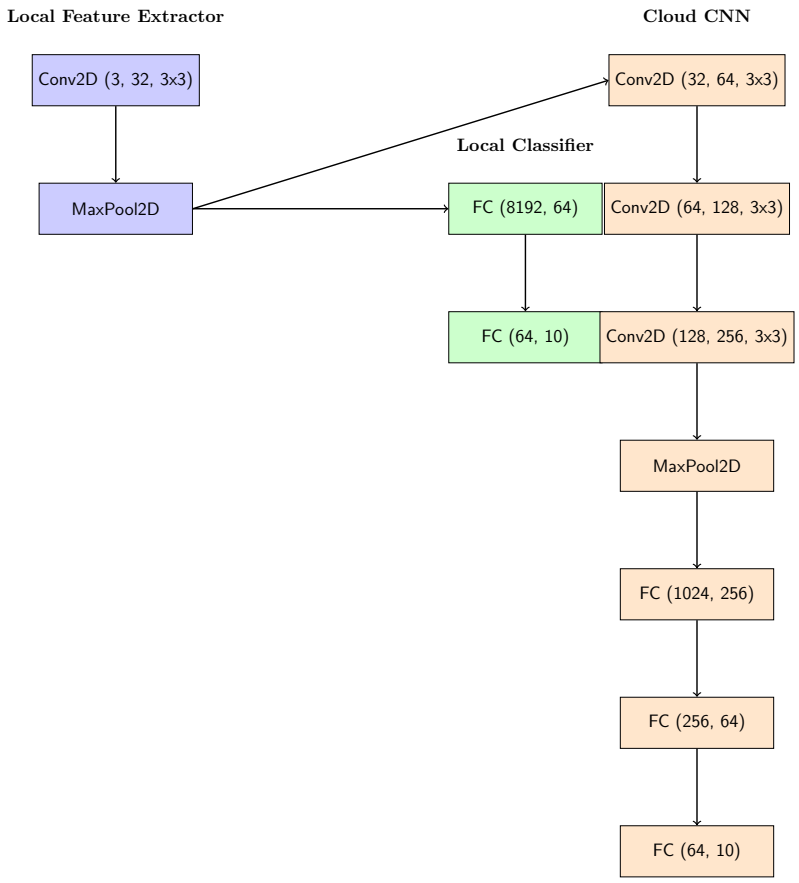

Figure 1: Distributed Deep Neural Network (DDNN) Architecture

3 Training

3.1 Optimizer

The training process for the DDNN involves both the local and cloud networks being trained simultaneously. A single Adam optimizer is used to train the DDNN, with an initial learning rate of 0.001. During training, this learning rate is adjusted to a lower value if the network becomes stagnant in terms of performance improvements. The optimizer is responsible for optimizing

the parameters of the entire distributed model, combining the local feature extractor, local classifier, and cloud network. Our joint training approach follows previous work on "Distributed Deep Neural Networks (DDNNs)" [**?**], where both local and cloud networks are optimized simultaneously to improve decision-making efficiency. This strategy is necessary in order for our model to learn when to rely on local inference and when to send samples to the cloud, ensuring better utilization of resources.

3.2 Loss Function

The two networks bust be trained together using a joint loss function. The total training loss (L_{total}) is defined as a weighted combination of the local and cloud losses:

$$L_{\text{total}} = \lambda_{\text{local}} L_{\text{local}} + \lambda_{\text{cloud}} L_{\text{cloud}}$$

where λ_{local} and λ_{cloud} are the respective weights of the local and cloud losses, such that:

$$\lambda_{\text{local}} + \lambda_{\text{cloud}} = 1$$

Both the **Local Loss** (L_{local}) and **Cloud Loss** (L_{cloud}) are computed using the cross-entropy loss function.

$$\text{Cross-Entropy Loss} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i), \tag{1}$$

3.3 Weight Initialization

The weight assigned to the local loss, denoted as **Local Weight** (λ_{local}), was set to a random value between 0 and 1 during training to assess its impact on the network's performance, while the **Cloud Weight** (λ_{cloud}) was calculated as follows ensuring balance between the influence of the two networks:

$$\lambda_{\text{cloud}} = 1 - \lambda_{\text{local}}$$

As noted in [**?**], experimenting with different weight values did not lead to significant changes in the overall network performance. Similarly, we also observed that varying these weight values did not significantly impact the network's performance, which further indicates that the network is robust to variations in the contribution of local and cloud processing.

In order to prove this claim we made a test plot 4 using different local wieghts to train the DDNN and measured their performance across 7 runs

for each weight . The offload method used was normalized entropy which is shown later to perform well.

4 Confidence Rule

Initially, an input image is passed through the local network, which extracts features through the local feature extractor and performs an initial classification using the local classifier. This local classification is then evaluated by an offloading mechanism, which decides whether to keep the local classification result or forward the extracted local features to the cloud network for further processing and classification.

The offloading mechanism is designed to optimize cost and accuracy by delegating samples that need deeper analysis to the cloud. If the confidence level of the local classification is high enough, the classification result is used as is; otherwise, the sample is sent to the cloud for additional processing. This approach allows for an efficient balance between using local and cloud resources.

4.1 Online Unerctainty Confidence Mechanism with Entropy

Entropy is used to evaluate the uncertainty of the classification made by the local network. After the local feature extractor and classifier have processed the input, a probability distribution over the possible classes is produced. Entropy measures the degree of uncertainty in this distribution—essentially quantifying how confident the local classifier is about its prediction.

If the entropy value η is low, it indicates that the local classifier is confident in its decision, meaning that one class has a significantly higher probability compared to the others. In such cases, the classification can be finalized locally without further processing. Conversely, if the entropy value is high, indicating uncertainty (i.e., the probabilities are more evenly spread across multiple classes), the sample is forwarded to the cloud network for further analysis.

In order to decide wether to classify a sample a threshold is used to compare with the entropy given by the first classification on the local network

The normalized entropy is defined as:

$$\eta(\mathbf{x}) = -\sum_{i=1}^{|C|} \frac{x_i \log x_i}{\log |C|}$$

where C is the set of all possible labels and $\mathbf{x}$ is a probability vector. The normalized entropy η has values between 0 and 1, which allows for easier interpretation and searching of samples at a particular exit point.

4.2 Optimized Offline Confidence Rule

4.2.1 Setting

Upon completion of the training of the Distributed Deep Neural Network (DDNN), the local features of each sample from the final epoch are retained. These features are subsequently used to train the optimization rule, paired with the expected cost of classifying the sample locally, denoted as B_k, also given from the last epoch data. The value B_k is defined as follows:

$$B_k = \text{Local\_cost} - \text{Cloud\_cost}$$

The *Local_cost* (C_{local}) and *Cloud_cost* (C_{cloud}) are computed based on the predicted probabilities for the correct class by the local and cloud networks, respectively. Specifically, they are defined as follows:

$$C_{\text{local}} = 1 - p_{\text{local},y}$$

$$C_{\text{cloud}} = 1 - p_{\text{cloud},y}$$

where $p_{\text{local},y}$ and $p_{\text{cloud},y}$ represent the predicted probabilities for the correct class y by the local and cloud networks, respectively. This formulation means that the more confident a network is in predicting the correct class (i.e., the higher the probability assigned to the correct class), the lower the associated classification cost will be.

For instance, if either the local or cloud network predicts class 8 with 90% confidence and class 8 is indeed the correct class, then:

$$C_{\text{local}} = 1 - 0.90 = 0.10$$

This implies that a higher predicted probability for the correct class results in a lower classification cost.

The value B_k represents the difference between the local and cloud classification costs. A positive B_k indicates that the *Local_cost* exceeds the *Cloud_cost*, suggesting that it is more efficient to classify the sample using the cloud network rather than locally.

4.2.2 Training

All B_k values generated during the final epoch of DDNN training are subsequently utilized to calculate the threshold b^*. The threshold b^* is determined as the L_0 percentile of the sorted B_k values, ensuring that L_0 percent of the samples have $B_k < b^*$ and can thus be classified locally. In this way, b^* serves as a boundary that distinguishes which samples are deemed "easy" and suitable for local classification.

During the training of the offloading mechanism, each local feature is paired with its corresponding B_k value. The B_k value is employed as a label to categorize samples into "easy" (i.e., those with $B_k < b^*$) and "hard" (i.e., those with $B_k \geq b^*$). The optimization rule is subsequently trained to make this distinction effectively.

The training uses the **binary cross-entropy loss function**, which is suitable for this binary classification problem where we decide whether a sample is classified locally or offloaded to the cloud. The binary cross-entropy loss is defined as:

$$L_{\text{binary\_cross\_entropy}} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

where y_i is the true label (0 for easy, 1 for hard), $\hat{y}_i$ is the predicted probability, and N is the number of samples in the batch.

The **Adam optimizer** is employed during training, with a learning rate of 0.001

4.2.3 Inference

During inference, the local features of each sample are provided to the offloading mechanism, which determines whether the sample should be labeled as "easy" or "hard." If a sample is labeled as "easy," it is classified locally. Conversely, if a sample is labeled as "hard," it is offloaded to the cloud network for classification. This approach ensures an efficient and cost-optimized decision-making process between local and remote network classifications.

4.2.4 Architecture

The offload mechanism is implemented using a simple dense layer with 128 neurons, followed by a ReLU activation function. The final output layer uses a Tanh activation function, which is suitable for making binary decisions, to determine whether the sample should be processed locally or forwarded to the cloud.

5 Results

Training was conducted over 20 epochs, which was found sufficient to achieve convergence for both the local and cloud networks. For the offline Optimization Rule confidence mechanism, convergence was achieved after 10 epochs of training. To evaluate the model's performance in terms of accuracy , we used the testing set of 10,000 samples provided by the Cifar 10 Dataset.

5.1 Results using Uncertainty Rule with different thresholds

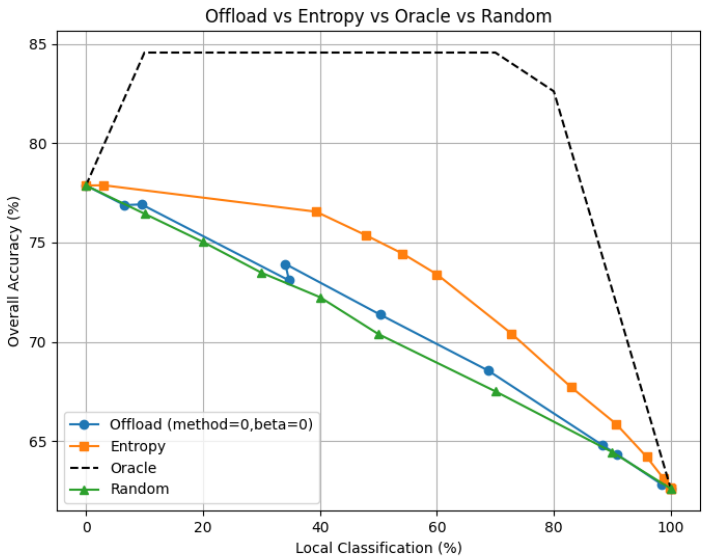

Figure 2: Results of using online uncertainty with different thresholds.

As shown in Figure 2, the case of **0**% of samples classified locally represents a Centralized Network, where all samples are resolved remotely, achieving the highest performance measured in accuracy.

We observe that the results are the expected since the curve is concave downward. As shown in the graph, that even for high percentages of samples classified locally (**30**%), the accuracy of the network does not drop significantly remaining close to the accuracy of the cloud network although higher percentage of samples classified locally resolves into pure accuracy results as expected.

Interestingly, there exists a "sweet spot" where the Distributed Deep Neural Network (DDNN) achieves nearly the same accuracy as the centralized network while processing a significant portion of the samples locally (**30**%).

5.2 Results using Optimized Rule with different L0 thresholds

From Figure 2, we can observe that, similar to the uncertainty-based rule, when the percentage of samples classified locally is zero, representing a fully centralized network where all samples are resolved remotely, we achieve the highest accuracy. As the proportion of samples processed locally increases, the overall accuracy decreases linearly indicating bad results.

Unlike the previous experiment, we can not find a point where both networks are utilized, and the overall accuracy matches that of the centralized network alone while also reducing computational costs. This discrepancy might be attributed to the offloading mechanism's inability to efficiently identify which samples should be processed locally and which should be processed remotely, even though the mechanism sticks to the intended distribution of local and remote classifications based on the L_0 value. Consequently, while the offload mechanism meets its quantitative goal of classifying the expected amount of samples locally, it is uneffective in distinguishing the appropriate samples for local and remote processing. More on that results is discussed on a later section **??**.

5.3 Comparison of Offload Mechanism Using Figures vs. Local Features

To evaluate whether the input type for the offload mechanism significantly affects classification performance, we trained and tested two different versions:

1. **Local Features Input**: The offload mechanism was trained using extracted feature representations from the local feature extractor.

2. **Figures Input**: Instead of using local features, the offload mechanism was trained directly on raw image inputs.

The results are plotted in Figure 3. We observe that both approaches yield nearly identical overall accuracy and local classification percentages across all tested L_0 thresholds. This suggests that the choice of input representation does not significantly impact the offload mechanism's effectiveness, meaning there is little benefit in prioritizing one method over the other.

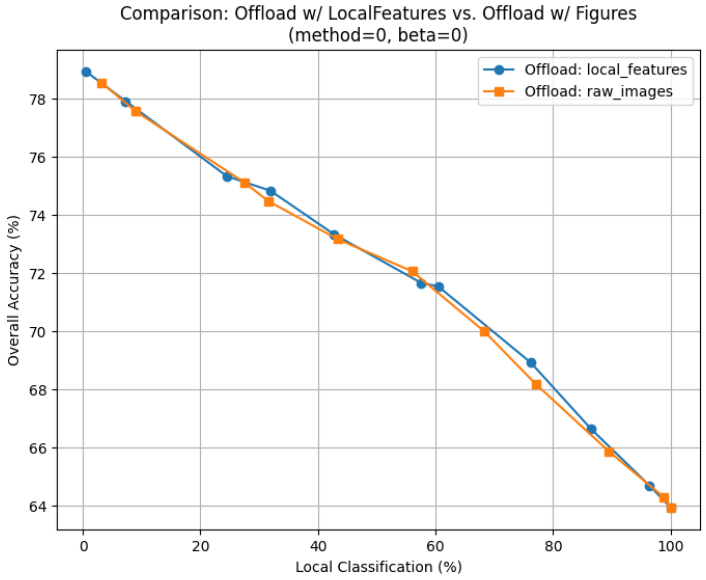

Figure 3: Comparison of Offload Mechanism Performance: Figures vs. Local Features

Given these findings, we conclude that training the offload mechanism on local feature embeddings provides no major disadvantage compared to using raw images. Since local feature-based training has the added benefit of requiring lower computational cost and reduced input dimensionality, we find no strong justification for using raw images instead of extracted features for this task.

5.4 Different Local Weights Testing

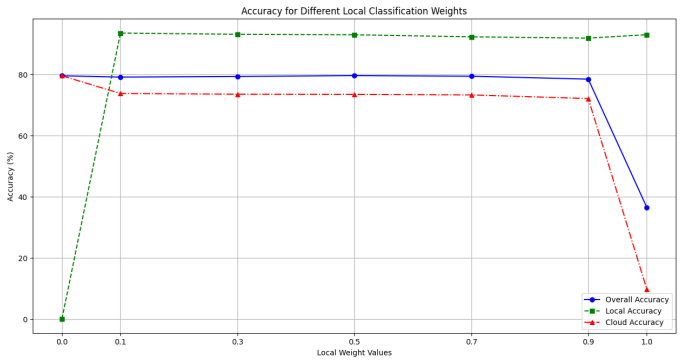

Figure 4: Results using different local weights to train the DDNN.

6 Shallow Conv Offload Mechanism Attempt

6.1 Training Failure of the Optimized Rule

Through our analysis, we observed that the optimized offload rule does not train correctly and does not function as expected. To verify whether the rule was making correct offloading decisions, we examined whether the samples assigned to the local network truly satisfied the condition $b_k < b^*$. In an ideal case, the offload mechanism should have successfully learned this threshold and made correct assignments. However, our findings revealed significant failures in this approach.

Specifically, when testing the offload mechanism on training samples, we measured how often the assigned local samples actually satisfied $b_k < b^*$. The accuracy of this decision-making process was **below 50%**, meaning that the optimized rule was no better than random selection. This issue was further confirmed in testing samples where the same pattern was observed.

Additionally, as shown in Figure 2, the performance of the optimized rule aligns almost perfectly with a random offload decision. This further supports the claim that the learned mechanism is ineffective and does not capture any meaningful patterns from the training process.

6.1.1 Oracle Decision Function

For evaluation purposes, we define an **oracle decision function** that always selects the classifier (local or cloud) yielding the correct result. Specifically:

- If only the cloud predicts correctly, the sample is offloaded to the cloud.
- If only the local predicts correctly, the sample is kept local.
- If both predict correctly, the decision is based on which network assigns a higher probability to the correct class.
- If both predict incorrectly, the sample is sent to the cloud.

This oracle serves as an upper-bound benchmark, and we compare the optimized rule's decisions to it throughout our analysis.

6.1.2 Architecture of the New Offload Mechanism

In an attempt to address the failure of the initial optimized rule, we redesigned the offload mechanism to use a **shallow convolutional architecture** with 4 convolutional blocks instead of a simple MLP.

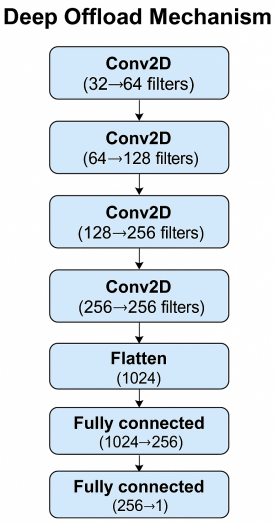

Figure 5: Architecture of the new Shallow Conv Offload Mechanism.

We employed the following training setup:

- **Dropout:** 0.3
- **Weight decay:** 10^{-5}
- **Data augmentation:** `RandomCrop, HorizontalFlip`
- **Blocks:** 4 convolutional blocks with 1 or 2 layers per block

6.1.3 Overfitting Behavior

Despite the more expressive architecture, the overfitting issue persisted. As shown in Figure 6, the training accuracy increases steadily, but test accuracy stagnates early.

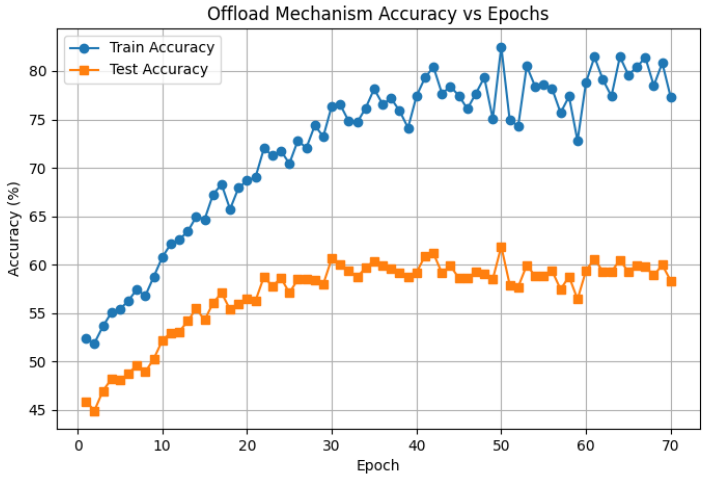

Figure 6: Train vs. Test accuracy of the new offload mechanism, revealing significant overfitting.

We experimented with deeper variants (i.e. increasing the number of layers per block to 3 and 4) but observed no significant gains on the test set. Figure 7 illustrates that more depth does not improve generalization.

6.1.4 Effect of Local Percentage on Performance

We further analyzed how performance varies with different offload thresholds (i.e., target local percentages). Figure 8 shows that stricter offload conditions (e.g., only 10% local) lead to better classification accuracy, likely because the mechanism handles only the "easiest" samples locally.

Interestingly, this shallow conv architecture performs comparably to the entropy baseline, especially under extreme local ratio settings. While this indicates progress compared to the original failed rule, there's still a considerable gap from the oracle.

6.1.5 Conclusion & Next Steps

Although the switch to a convolutional architecture improved results compared to the original optimized rule, the new offload mechanism still suffers

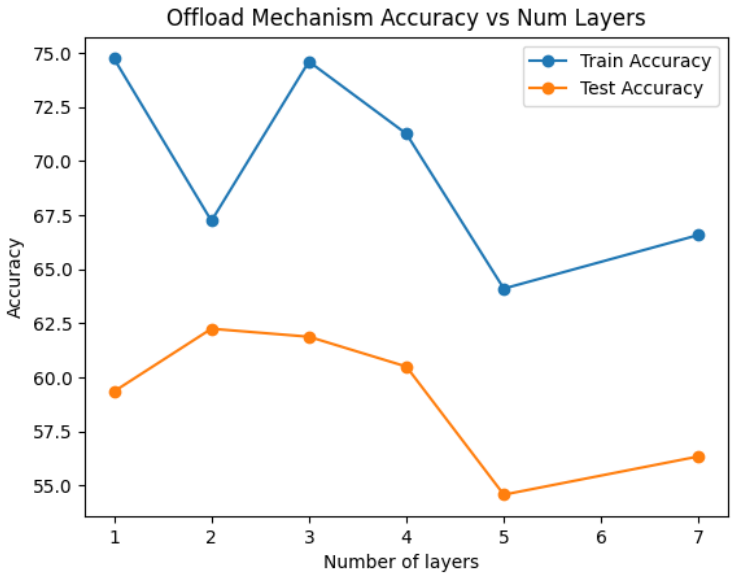

Figure 7: Performance across different depth levels. Deeper variants (3 or 4 layers/block) offer no test accuracy improvement.

from overfitting and fails to generalize beyond a point. The model's behavior mirrors the entropy-based baseline under various settings, indicating some learning capacity — but also that there remains substantial room for improvement.

As a result, in the next section, we explore a revised architecture and approach that aims to better exploit confidence signals and logits, reduce overfitting, and ultimately close the gap with the oracle mechanism.

7 Logits-based Offload Mechanism

7.1 Replacing Local Feature Maps with Classifier Logits

In this section, we explore a simpler and potentially more effective offload mechanism by changing the network input. Instead of using the local feature map produced by the local feature extractor, we directly feed the extbflogits of the local classifier as input to the offload mechanism.

This new approach offers two key advantages:

- It reduces the input dimensionality significantly (from a high-dimensional

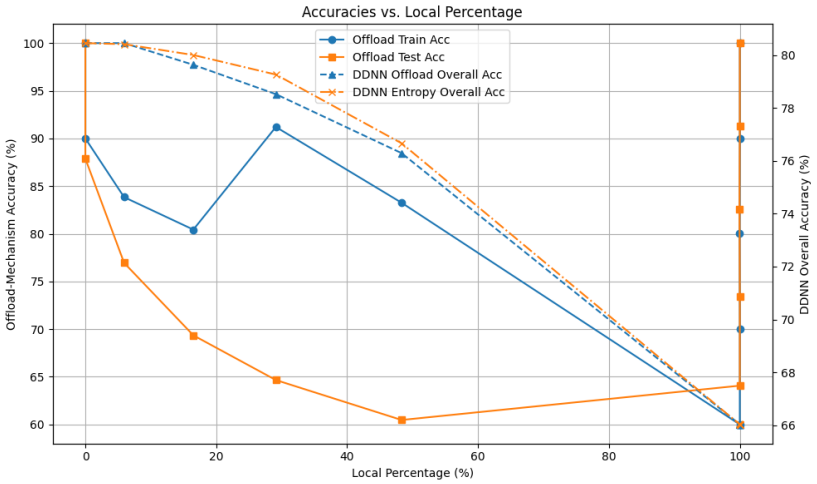

Figure 8: Train/Test accuracy across different target local percentages. Accuracy drops as more samples are forced local.

feature map to only 10 values), allowing us to use a simple feedforward neural network instead of a deep CNN.

- It is expected to achieve accuracy close to or better than the entropy method, since the logits contain all the information used in the final classification decision.

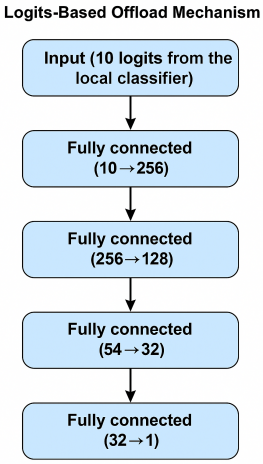

Figure 9: Architecture of the Logits-Based Offload Mechanism (dimensions: 10→256→128→54→32→1)

7.2 Testing Performance

Figure 10 shows the results of this method under varying local offload percentages. As expected, the logits-based model is more effective than the entropy method surpassing its accuracy on most setups especially the difficult ones(50%local sent) and does a better job into catching the oracle performance

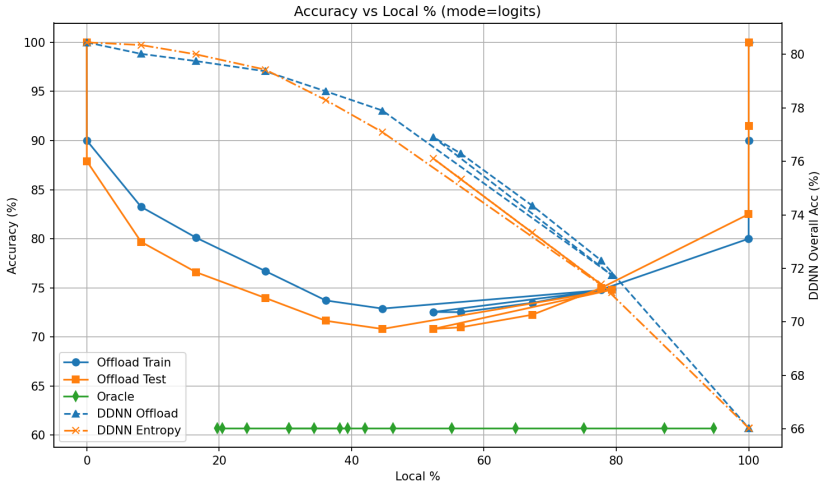

Figure 10: Performance of the logits-based mechanism under varying local percentage targets.

7.3 Input Feature Comparisons

In order to be sure that this aprroach is better than the previous one we test each method on different local percentages and present its train-validattion accuracy. We compare three different input configurations for the offload mechanism:

1. The original local feature map (high-dimensional).
2. The 10 logits produced by the local classifier.
3. An enhanced representation (**logits+**) which appends 2 additional values to the logits: the entropy of the logits and the gap between the top-2 predicted probabilities from the local prediction (dimension = 12).

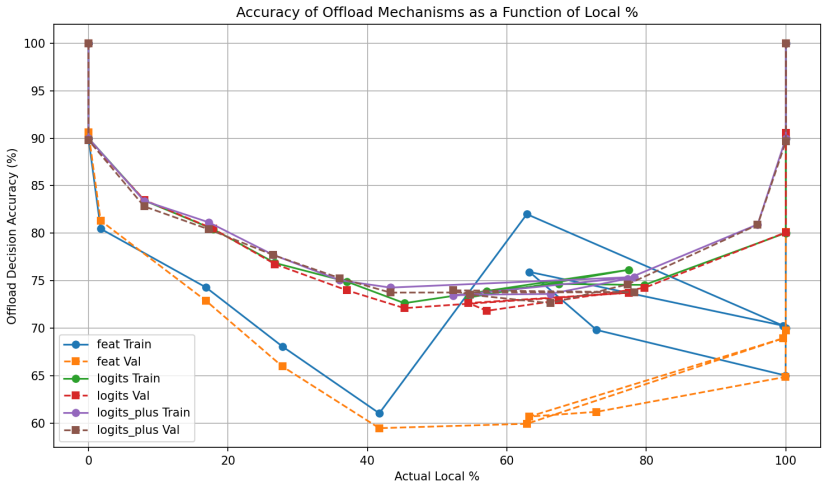

Figure 11: Comparison of offload accuracy across different input types: feature map, logits, and logits+.

As shown in Figure 11, the logits+ configuration achieves slightly better performance than the raw logits, likely due to the richer information content. Overfitting is not observed: the training accuracy grows steadily without diverging from the test accuracy.

Interestingly, the training accuracy has not yet reached perfect levels, suggesting that the model might still benefit from additional informative features. Thus, instead of increasing the number of layers, a more promising path could be to enrich the input representation even further.

7.4 Data Analysis

7.4.1 Oracle Noise Metrics

To validate that the labels provided to the offload mechanism during training are coherent with the oracle decisions, we performed a data consistency analysis. Specifically, we compared the label assigned to each training sample using our confidence-based thresholding rule (see Section 4.2) with the corresponding oracle decision (defined in Section 6.1.1). The following plot 12 shows

1. Noise Rate %: Data Consistency.
2. Oracle Accuracy %: The accuracy the DDNN achieved using Oracle decision method on the certain test.

3. Border Rate %: Samples that Bk values are significantly small (0.01) indicating more difficult classification task.

4. Local Rate %: The percentage of total samples that were sent locally using Oracle decision.

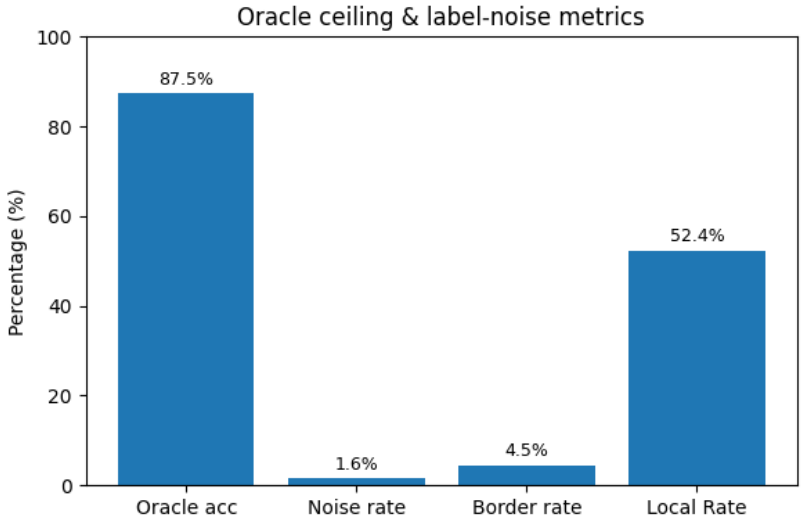

Figure 12: Comparison between oracle decisions and confidence-rule-based labels on the training data.

Figure's 12 metric (**NoiseRate%**) shows that there are no significant mismatches between the training labels and the oracle choices. This indicates that the training dataset used for the offload mechanism is consistent and correctly labeled according to our definitions.

7.4.2 Offload Logits Mode vs Oracle Comparison

TODOO We present multiple average decision metrics on where the two methods dissagree in order to find answers on how the differ.

8 Future Work and Next Steps

Based on the observations and limitations discussed so far, we outline several directions for future improvement and investigation:

1. **Adding more info to the input of the offload mechanism.** Currently, the classifier of the optimized rule is trained using only a 10-Dimenson input (Local CLassifier's Logits). We might consider adding informative signals, such as the difference between the top-2 predicted probabilities, or the gap between the top predicted probability and the sum of all remaining probabilities. Although we tried something like that on **Logits+** mode, figure 11 showed that adding just the entropy and the top-2 predicted probabilities difference improving in a 12-Dimension input signal made progress but wasn't enough to achieve oracle-accuracy.

2. **Adjusting the loss function.** Given a possibly imbalanced dataset in the sence of easy(proper for local classification) and hard samples(sent to cloud), we may consider replacing the current binary cross-entropy (BCE) loss used for training the offload mechanism with a weighted BCE that accounts for the skewed distribution. Another option might involve experimenting with focal loss (im not really sure about all that but i have seen some articles referencing it in similar inbalanced data problems), which emphasizes harder-to-classify samples. Alternatively, we could attempt to synthetically generate more "easy" labeled samples to balance the dataset (Im dont know how this is done and if it works — i'll have to search it better).

3. **Refining the cost function.** Another avenue is to revisit the definition of the b_k values, possibly incorporating extra information such as the local entropy, or similar indicators. This could help produce more clearly separable b_k values, making the classification boundary easier for the offload mechanism to learn.

4. **Dynamic joint training.** Lastly, we might explore synchronizing the training of the DDNN and the optimized rule in a more dynamic fashion. Specifically, instead of training the optimized rule only after the DDNN completes training, we could start co-training both networks after a few epochs once the DDNN begins stabilizing. Alternatively, generating multiple distributions of b_k values at different stages of DDNN training and averaging them might help smooth out noise and provide better training labels for the optimized rule.