Slice Resource Allocation with Multiple Edge-Exit Distributed Deep Neural Networks

Ali Ehsanian
EURECOM
Sophia-Antipolis, France
ali.ehsanian@eurecom.fr

Thrasyvoulos Spyropoulos
Technical University of Crete
Chania, Greece
spyropoulos@tuc.gr

***Abstract*—Network slicing is a pivotal concept in the evolution of 5G networks. It enables network operators to partition physical resources from edge to data center, allowing concurrent multiplexing of tenants while adhering to each tenant's Service Level Agreement. Efficient resource allocation is critical in this context, prompting recent research into deep neural networks (DNNs). However, challenges arise with edge resources, including the need to rapidly scale resources (within milliseconds) and the cost of transmitting large volumes of data to a cloud for centralized DNN-based processing. To address these issues, we previously investigated distributed deep neural network (DDNN) architectures based on CNN and LSTM with a single local exit, facilitating efficient edge-cloud collaboration. In this work, we aim to generalize the training methodology for such networks, identifying both shared and unique aspects across different models. Additionally, we propose an extended architecture that incorporates multiple local exits, which introduces new challenges. Unlike DDNNs with one local exit, multiple local exits observe different subsets of the input signals, while the remote exit processes a superset of these. This leads to partially coupled layers and exits, complicating both the model architecture and training process. Moreover, the offloading decision mechanism now involves more intricate trade-offs, such as determining whether to forward specific subsets of preprocessed features to the cloud for further processing. We explore the joint training of DDNN exits and an optimized offloading mechanism, demonstrating that our architecture resolves nearly 40% of decisions at the edge without incurring additional penalty compared to centralized models.**

***Index Terms*—Network Slicing, Resource Allocation, Distributed Deep Neural Network, Offloading Mechanism, 5G Networks**

I. Introduction

The rise of 5G and the transition to 5G+/6G networks have transformed network architectures by emphasizing virtualization and resource slicing, enabling flexible support for diverse Quality of Service (QoS) requirements and Service Level Agreements (SLAs). Virtual Network Functions (VNFs) improve flexibility and enable data-driven optimization in resource allocation. Modern Machine Learning (ML) techniques, such as deep learning [1]–[4] and reinforcement learning [5], [6], [7], are actively being explored for slice resource allocation and orchestration, substantially improving network performance and resource efficiency.

This project has been supported by the European Union's Horizon 2020 research and innovation program under Grant Agreement No. 861165.

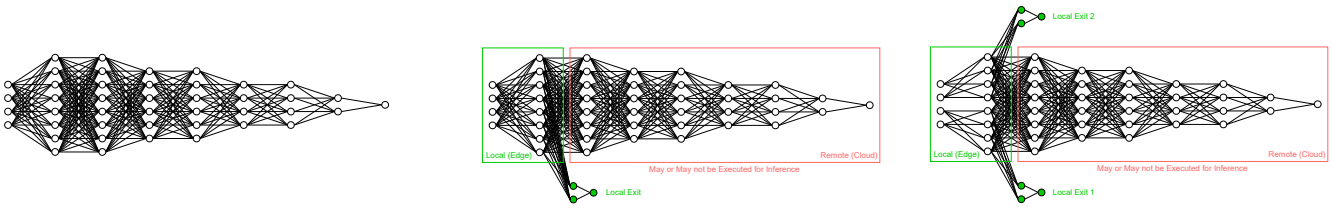
Centralized DNNs face challenges such as *stringent latency requirements* in the Radio Access Network (RAN), where low latency is critical. This contrasts with application-layer tasks (e.g., image classification), which can tolerate higher latency and be offloaded to the cloud. Additionally, *data transmission overheads* from edge to core networks can increase latency, reduce throughput, and disrupt performance.

DNNs are typically designed as sequential layers with predictions at the end, but the incorporation of intermediate prediction modules enables early exits, reducing unnecessary processing. Distributed Deep Neural Networks (DDNNs) extend this concept by distributing layers between the edge and the cloud, enabling low-latency local predictions or high-accuracy remote predictions. Joint training is essential to balance local and remote performance by leveraging local exits for simpler signals and offloading the processed *features* of complex signals to the cloud. While final exits improve accuracy, they also increase processing, communication, and latency costs. Additionally, the absence of ground truth during inference complicates the selection of the optimal exit point.

We have performed some initial exploration of these ideas in earlier work [8], focusing on one local component that (i) sees all signals and (ii) has a single local exit. However, practical scenarios often involve local neural networks accessing only subsets of signals, such as data from a single or a few Base Stations (BSs) under small edge clouds. Multiple such locations may exist, each with partial and possibly correlated signals, while an aggregation point in the core or cloud can combine all signals to make more informed decisions by leveraging deeper layers and inter-signal correlations. This paper investigates multi-exit DDNNs with correlated time-series data. Although the original DDNN work [9] theorizes multi-exit scenarios, regression tasks, as highlighted in [8], pose significantly greater challenges compared to classification problems.

Training multi-exit models presents challenges such as determining the optimal placement of exits, designing branch architectures, developing effective training strategies, and evaluating performance. This paper introduces, trains, and analyzes a distributed DNN architecture tailored for multi-edge resource allocation. The key contributions of this work are as follows:

- (*Architecture*) We propose a distributed DNN with multiple edge exits for rapid local inferences, along with a



(a) Centralized DNN (b) Distributed DNN (one local exit) (c) Distributed DNN (two local exits)

Fig. 1: Distributed and Centralized DNN Schemes

remote exit in the central cloud to process aggregated features from the edge networks to achieve higher accuracy when necessary.

- (*Offline Optimization*) We optimize joint training hyper-parameters to strike a balance between making accurate local decisions and generating informative features for remote layers when required.
- (*Online Optimization*) We develop an optimized offloading mechanism that selects samples for cloud processing, aiming to minimize communication and latency costs. This mechanism is trained on historical data and approximates oracle-level performance.

The remainder of this paper is organized as follows: Section II outlines the problem setup, section III elaborates on the DDNN architectures. Section IV explores offline joint training and online inference. Section V validates the proposed architectures using real traffic data. Section VI presents future work and conclusions.

II. Problem Setup

Slice Resource Allocation with DNN: We assume a network infrastructure hosting multiple network slices composed of various Virtual Network Functions (VNFs). We model each VNF as a discrete-time series reflecting required computational resources (such as CPU and memory). Denoting the set of VNFs by $\mathcal{K} = \{1, ..., K\}$ and their resource demands at time t as d_t^k, we aim to utilize past traffic data to optimize resource allocation for all VNFs through a DNN-based framework as follows:

$$\hat{y}_t^k = \mathcal{F}(\mathbf{d}_{t,N}^k; \boldsymbol{\theta}), \tag{1}$$

where the DNN input, $\mathbf{d}_{t,N}^k = \{d_{t-N}^k, ..., d_{t-1}^k\}$, consists of N historical traffic samples for VNF $k \in \mathcal{K}$ up to time t. The model, represented by $\mathcal{F}(\cdot; \boldsymbol{\theta})$, with $\boldsymbol{\theta}$ as the parameter vector, is trained to forecast $\hat{y}_t^k$, the resource allocation for VNF k at time t. This forecast aims to optimize resource allocation by balancing under-/over-provisioning costs against the expected (unknown) demand d_t^k.

Objective Function for Slice Resource Allocation: The objective in standard traffic forecasting is to predict future traffic, $\hat{y}_t$, to closely match the actual traffic, d_t, at time t using past N traffic samples. This goal is conventionally met by employing a DNN trained with a least squares objective function.

$$f(\hat{y}_t, d_t) = (\hat{y}_t - d_t)^2. \tag{2}$$

The objective in our work diverges from conventional models by considering the asymmetric costs associated with under-provisioning (where predictions are less than the actual demand which risks violating SLAs) and over-provisioning (where predictions exceed the actual demand which leads to unnecessary resource wastage). Our approach employs a DNN tailored for minimizing the consequences of both SLA violations and resource wastage. Without loss of generality, we adopt the following objective function to balance these factors efficiently:

$$f(\hat{y}_t, d_t) = \begin{cases} c_1 \cdot (\hat{y}_t - d_t)^2 & \text{if } (\hat{y}_t - d_t) \leq 0 \\ c_2 \cdot (\hat{y}_t - d_t) & \text{if } (\hat{y}_t - d_t) > 0, \end{cases} \tag{3}$$

where quadratic terms are applied for SLA violations, and linear terms for the opportunity cost of resource wastage[1].

III. Proposed Distributed Deep Neural Network

In the context of a 5G network, we consider a scenario where various VNFs require certain resources (such as CPU, memory, and bandwidth) to meet user SLAs. At any given time t, each VNF demands a specific quantity of resources, represented as d_t^k, based on past N demand values, $\mathbf{d}_{t,N}^k$, which are random and potentially non-stationary. The demand vector $\mathbf{d}_{t,N}^k$ is input into a DDNN which then predicts the resource allocation for each VNF at time t, denoted as $\hat{y}_t^k$. We can describe the DDNN with the following equation:

$$(\hat{y}_{L1,t}^k, \hat{y}_{L2,t}^k, \hat{y}_{R,t}^k) = \mathcal{F}(\mathbf{d}_{t,N}^k; \boldsymbol{\theta}_{\text{DDNN}}), \tag{4}$$

where $\mathcal{F}(\cdot; \boldsymbol{\theta}_{\text{DDNN}})$ is the approximation function modeling the DDNN, and $\boldsymbol{\theta}_{\text{DDNN}}$ denotes the model parameters. Contrary to the traditional DNN described in Eq. (1), the DDNN features three outputs: $\hat{y}_{L1,t}^k$ and $\hat{y}_{L2,t}^k$ for the two local exits, and $\hat{y}_{R,t}^k$ for the remote exit[2].

The architecture, illustrated in Fig. 1, consists of two parallel, compact initial DNN modules located at separate edge

[1]Note that our architecture is adaptable to various non-symmetric objective functions, similar to those in [10], [11].

[2]Note that the number of local exits can exceed two, and these exits operate in parallel rather than sequentially. This arrangement differs from configurations where multiple local exits are distributed sequentially along the network layers.

sites, tasked with executing partial local allocation decisions, with each local component processing a distinct subset of the input data.

$$\hat{y}^k_{L1,t} = \mathcal{F}_{L1}(\mathbf{d}^k_{L1,t,N}; \boldsymbol{\theta}_{L1}), \tag{5}$$

$$\hat{y}^k_{L2,t} = \mathcal{F}_{L2}(\mathbf{d}^k_{L2,t,N}; \boldsymbol{\theta}_{L2}), \tag{6}$$

where $\mathbf{d}^k_{L1,t,N}$ and $\mathbf{d}^k_{L2,t,N}$ are the inputs for each local DNN module, such that $\mathbf{d}^k_{L1,t,N} \cup \mathbf{d}^k_{L2,t,N} = \mathbf{d}^k_{t,N}$ and $\mathbf{d}^k_{L1,t,N} \cap \mathbf{d}^k_{L2,t,N} = \emptyset$. $\mathcal{F}_{L1}(\cdot; \boldsymbol{\theta}_{L1})$ and $\mathcal{F}_{L2}(\cdot; \boldsymbol{\theta}_{L2})$ represent the local DNNs, each with their respective parameters $\boldsymbol{\theta}_{L1}$ and $\boldsymbol{\theta}_{L2}$. For some samples, these local decisions may not be deemed appropriate, i.e., they may incur relatively high costs. In such cases, the output of the local DNNs (e.g., $\mathbf{z}^k_t = \mathbf{z}^k_{L1,t} \cup \mathbf{z}^k_{L2,t}$, as shown in Fig. 1) is sent to a much larger DNN module, located remotely from the decision-required site (e.g., the base station). This remote module then provides its own allocation decision, as follows:

$$\hat{y}^k_{R,t} = \mathcal{F}_R(\mathbf{z}^k_t; \boldsymbol{\theta}_R), \tag{7}$$

where the output from the local DNN blocks, $\mathbf{z}^k_t$, serves as the input to the remote component. $\mathcal{F}_R(\cdot; \boldsymbol{\theta}_R)$ denotes the remote DNN, characterized by its parameters $\boldsymbol{\theta}_R$.

Local Exits: In DDNNs, local exits are initial layers located at the network edge. Without loss of generality, we consider two distinct local components, each comprising a shallow DNN block, which is less complex than the remote component. During training, outputs from these blocks, $\mathbf{z}L1, t^k$ and $\mathbf{z}L2, t^k$, are processed by their respective Fully Connected (FC) layers and aggregated as $\mathbf{z}t^k$ for transmission to remote layers, as shown in Fig. 1. The combined local outputs, $\hat{y}L, t^k = \hat{y}L1, t^k \cup \hat{y}L2, t^k$, constitute the *local prediction* or *local exit inference*.

Remote Exit: In our DDNN, the remote exit is a component located in the central cloud. Without loss of generality, the remote component comprises three DNN blocks followed by four Fully Connected (FC) layers. The aggregated input, $\mathbf{z}t^k$, obtained from the local modules, is processed to generate the *remote prediction* or *remote exit inference*, denoted as $\hat{y}R, t^k$.

IV. DDNN Training and Inference

A. Offline DDNN Joint Training

Training a DDNN is inherently more complex than training a centralized DNN. It requires balancing the contributions of local and remote exits within the unified objective function to enable effective backpropagation across all layers. While local exits were originally introduced in [12] (GoogleNet) and [13] (BranchyNet) primarily as a regularization strategy that was not utilized during inference, our joint training approach explicitly focuses on optimizing performance trade-offs in several critical areas:

- Backpropagate the local exit's performance to its layers (i.e., $\boldsymbol{\theta}_L$) to ensure reliable local decisions $\hat{y}^k_{L1,t}$ and $\hat{y}^k_{L2,t}$, even with a simpler DNN module.
- Backpropagate the remote exit's performance to *both* remote layers(i.e., $\boldsymbol{\theta}_R$) and local layers (i.e., $\boldsymbol{\theta}_L$) to enhance remote inferences ($\hat{y}^k_{R,t}$) and ensure local layers generate valuable intermediate features (i.e., $\mathbf{z}^k_{L1,t}$ and $\mathbf{z}^k_{L2,t}$) for remote processing.

The computation of the DDNN loss integrates contributions from both edge and cloud components, and it is expressed using the following formula:

$$\begin{aligned}\text{DDNN Loss} = &\sum_{k=1}^{K} w_{L1} \cdot f(\mathcal{F}_{L1}(\mathbf{d}^k_{L1,t,N}; \boldsymbol{\theta}_{L1}), d^k_{L1,t}) \\ &+ w_{L2} \cdot f(\mathcal{F}_{L2}(\mathbf{d}^k_{L2,t,N}; \boldsymbol{\theta}_{L2}), d^k_{L2,t}) \\ &+ w_R \cdot f(\mathcal{F}_R(\mathbf{z}^k_t; \boldsymbol{\theta}_R), d^k_t) \\ = &\sum_{k=1}^{K} w_{L1} \cdot f(\hat{y}^k_{L1,t}, d^k_{L1,t}) + w_{L2} \cdot f(\hat{y}^k_{L2,t}, d^k_{L2,t}) \\ &+ w_R \cdot f(\hat{y}^k_{R,t}, d^k_t).\end{aligned} \tag{8}$$

In Eq. (8), the 'local weights" (w_{L1} and w_{L2}) and remote weight" (w_R) play a pivotal role in determining the influence of local and remote exits on the overall loss during joint training. For simplicity and without loss of generality, we assume the two local weights are equal and collectively refer to them as the "local weight" (i.e., $w_L = w_{L1} = w_{L2}$). These weights are constrained within $[0, 1]$ with $w_R = 1 - w_L$. Selecting optimal values for w_L and w_R is essential to ensure reliable local predictions, accurate remote decisions, and efficient resource allocation.

B. DDNN Online Inference

After training the local and remote DNN modules, a key forward-pass decision is whether the local resource allocation is sufficient or requires refinement by the remote DNN layers[3]. This decision operates as an unsupervised learning task, determining whether to rely on local outputs or escalate to remote layers, *without prior knowledge of the potential benefits/costs of additional processing*.

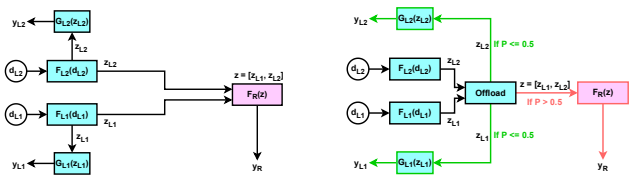
Oracle-based Offloading: We assume an ideal "oracle" with complete knowledge of the potential added value of remote processing for each sample. This oracle serves as a benchmark for analytical evaluations. We calculate the loss difference as follows:

$$\begin{aligned}\mathcal{L} &= \sum_{k=1}^{K} f(\hat{y}^k_{L1,t}, d^k_{L1,t}) + f(\hat{y}^k_{L2,t}, d^k_{L2,t}) - f(\hat{y}^k_{R,t}, d^k_t) \\ &= (C_{L1} + C_{L2}) - C_R = C_L - C_R.\end{aligned} \tag{9}$$

We define a general average cost for processing a sample in the cloud, which includes additional latency, offloading decision, and communication costs. For simplicity, this average cost is denoted as C_T and referred to as the transmission cost.

The loss difference, $\mathcal{L}$, is evaluated against the transmission cost (C_T). If $\mathcal{L} < C_T$ the sample is resolved locally, indicating

[3]The goal is not traffic load prediction but resource allocation that minimizes overall costs, balancing SLA penalties and resource wastage.



(a) DDNN Training (b) DDNN Inference

Fig. 2: DDNN (a) Training and (b) Inference Schemes. In both configurations, only the remote block (the purple colored) is located in the cloud, and other blocks are deployed at the edge.

that local processing is more cost-effective. Conversely, if $\mathcal{L} > C_T$, remote processing is preferred due to its greater benefits. An ideal oracle, if it existed, would consistently enable accurate decisions on whether to process samples locally or remotely.

Optimized offloading: We design a "binary classifier" to decide whether samples should be processed locally or remotely. This classifier is trained after the DDNN has been trained to ensure the stability of the data path, given the sensitivity of DDNN training to the selection of local and remote exit weights. During the offline training phase, all samples are processed through both local and remote layers. Consequently, by the end of this phase, the local cost $C_L = \sum_{k=1}^{K} f(\hat{y}^k_{L1,t}, d^k_{L1,t}) + f(\hat{y}^k_{L2,t}, d^k_{L2,t})$, the remote cost $C_R = \sum_{k=1}^{K} f(\hat{y}^k_{R,t}, d^k_t)$, and the transmission cost C_T for the training set samples are all known.

The binary classifier categorizes samples into two classes based on comparative costs: **Class 0:**, where $C_T < C_L - C_R$, indicating that remote processing is more cost-effective, and **Class 1:**, where $C_T \geq C_L - C_R$, favoring local processing due to high remote costs. The classifier generates a probability p to determine whether a sample should be processed locally or remotely (this is a supervised learning method)[4].

During online inference (Fig. 2(b)), the input signal ($\mathbf{d}^k_{t,N}$) is directed to both the local DNNs and the optimized offloading mechanism. The offloading mechanism evaluates p and determines whether to offload locally. If additional processing is deemed unnecessary, the intermediate signals $\mathbf{z}^k_{L1,t}$ and $\mathbf{z}^k_{L2,t}$ are sent to the local FC layers, and resources are allocated based on $\hat{y}^k_{L1,t}$ and $\hat{y}^k_{L2,t}$ (green path in Fig. 2(b)). Otherwise the aggregated intermediate signal $\mathbf{z}^k_t = \mathbf{z}^k_{L1,t} \cup \mathbf{z}^k_{L2,t}$ is routed to the remote DNN, where the output $\hat{y}^k_{R,t}$ determines resource allocation (red path in Fig. 2(b)).

V. Performance Evaluation

Data preprocessing: We use the publicly available Milano dataset [14], widely referenced in related research [15], and [16], to simulate VNF traffic patterns with data from base stations measured in megabytes. **Input:** At time t, the DDNN processes $(\mathbf{d}^k_{t,N}, K) \in \mathcal{R}^{N \times K}$, representing a snapshot of historical traffic data for $\mathcal{K}$ VNFs over N time intervals. **Output:** The DDNN generates an output $\mathbf{y}_t = \{y^1_t, y^2_t, ..., y^K_t\} \in \mathcal{R}^K$, where y^k_t corresponds to the resource allocation for VNF $k \in \mathcal{K}$, determined either locally ($\hat{y}^k_{L1,t}$ or $\hat{y}^k_{L2,t}$) or remotely ($\hat{y}^k_{R,t}$).

We adopt the preprocessing method outlined in [10] to prepare time-series data from base stations (BSs), leveraging correlations among BSs in high-traffic areas (e.g., tram lines, metro stations). This preprocessing organizes the data matrix such that highly correlated BSs are positioned adjacently, enabling unified resource allocation predictions for multiple BSs and improving efficiency, as explained in [10]. **LSTM-based DDNN:** We use the previous $N = 144$ samples (equivalent to daily measured samples for each base station in Milano dataset) to forecast the subsequent samples for $K = 16$ base stations simultaneously. The input fed into the LSTM-based DDNN is therefore structured as an array of dimensions (K, N). **CNN-based DDNN:** CNN architectures perform optimally with tensor inputs that exhibit strong local correlations. Therefore, the input to the CNN-based DDNN is designed as a "traffic box" with dimensions $\sqrt{K} \times \sqrt{K}$ over a temporal window of length N.

Performance Metrics: Our two metrics are the percentage of samples resolved locally and the overall cost of the DDNN:

$$C_{\text{DDNN}} = \sum_{m=1}^{M} \mathcal{I}_m \cdot C^m_L + (1 - \mathcal{I}_m) \cdot C^m_R, \tag{10}$$

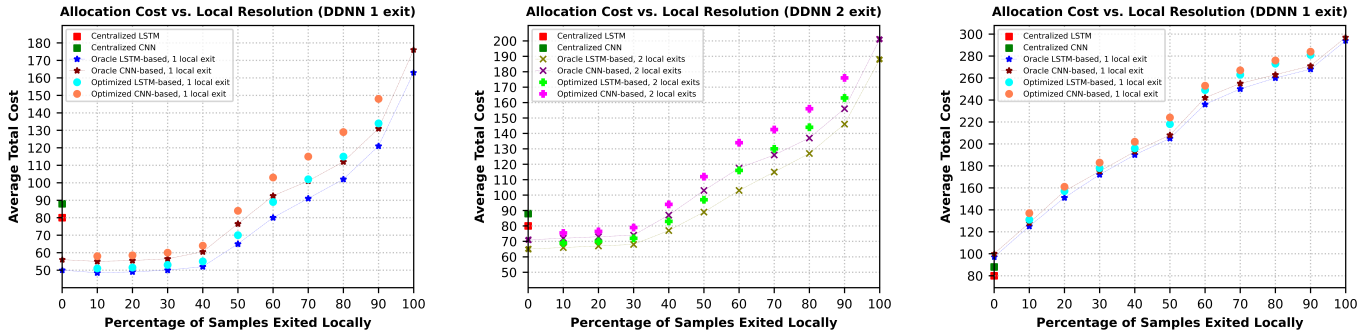
where, for sample m, C^m_L and C^m_R are the local and remote exit costs, respectively, and $\mathcal{I}_m$ indicates whether the sample exited locally or not.

$$\mathcal{I}_m = \begin{cases} 1 & \text{if the sample m exited locally} \\ 0 & \text{else.} \end{cases} \tag{11}$$

We implement and compare the following models (the architectures are summarized in Table I): **Centralized CNN:** A fully centralized 3D-CNN-based DNN, inspired by DeepCog [10], designed for resource allocation. This model operates entirely in the cloud and lacks edge prediction capabilities. **Centralized LSTM:** A fully centralized LSTM architecture that processes all data samples in the cloud, offering no edge prediction capabilities. **Oracle-based DDNN:** A DDNN utilizing an ideal offline optimal offloading policy, which assumes perfect knowledge of all exit points. While theoretical and impractical for real-world applications, it serves as a best-case benchmark. **Random-based DDNN:** A DDNN employing a random offloading policy, where each offloading decision follows a Bernoulli distribution with a success probability p. **Optimized DDNN:** A DDNN implementing an optimized offloading policy through a binary classifier, as previously described.

Resource Allocation Trade-off: We generate trade-off curves to analyze the relationship between total cost and the percentage of locally resolved samples. The model processes the test set, yielding outputs that are handled locally or

[4]At each time t, resource allocation decisions across all K *correlated* elements are binary, processed either locally or remotely. Future work will explore partial decision-making strategies using more complex layer hierarchies.



(a) DDNN with $(w_L, w_R) = (0.9, 0.1)$ (b) DDNN with $(w_{L1}, w_{L2}, w_R) = (0.9, 0.9, 0.1)$ (c) DDNN with $(w_L, w_R) = (0.1, 0.9)$

Fig. 3: Trade-off curves (Total loss vs Percentage of samples predicted locally)

TABLE I: Models for Performance Comparison

Model	Joint Training	Edge Offloading	Cloud Offloading	Realizability
Centralized DNN	×	×	✓	✓
Oracle-based DDNN	✓	✓	✓	×
Random-based DDNN	✓	✓	✓	✓
Optimized DDNN	✓	✓	✓	✓

remotely based on the offloading mechanism, with total cost computed using Eq. (10). The offline oracle-based trade-off curve is derived from *Oracle-based Offloading*, where varying the transmission cost (C_T) changes the local sample resolution rate, serving as a lower-bound baseline. The trade-off curve for the random policy is obtained by adjusting the probability parameter p from 0 to 1, forming an upper-bound baseline for performance evaluation. Offloading policies resulting in costs that surpass this curve are deemed ineffective. The online-optimized trade-off curve is produced using the *Optimized Offloading* method, where increasing the transmission cost (C_T) alters the curve.

Figs. 3 and 4 presents the trade-off curves for DDNN models with LSTM and CNN architectures. As illustrated in Figs. 3(a) and 3(b), at the point where no samples are predicted locally on the x-axis, simply introducing a local exit during training enhances baseline performance by 20-40%, even when all samples are resolved remotely[5].

Key observation 1: The incorporation of local exits has a clearly positive impact; however, increasing the number of local components can reduce the overall effectiveness of the DDNN.

Figs. 3(a), 3(b), 4(a), and 4(b) show that the optimized offloading mechanism achieves near-optimal performance when up to 30-40% of samples are resolved locally. However, in scenarios requiring a higher proportion of local processing, all models exhibit noticeable deviations from the optimal bound.

Key observation 2: The ability to make online decisions regarding which samples to process locally and how many to handle at the edge presents significant performance trade-offs. Notably, up to 40% of decisions can be resolved locally "for free", *without increasing overall provisioning costs compared to centralized models*.

Figs. 3(a) and 3(b) show that LSTM-based DDNNs consistently outperform their CNN-based counterparts, likely due to LSTMs' superior ability to handle sequential data, especially in scenarios with multiple local exits. Additionally, the performance gap between Oracle and Optimized offloading mechanisms in CNN-based DDNNs is 5-15% larger than LSTM-based models.

Key observation 3: LSTM models are particularly well-suited for applications requiring complex, time-dependent processing.

Figs. 4(a) and 4(b) reveal that adding a second local exit generally shifts the trade-off curve upward, resulting in higher costs across all configurations compared to single-exit models. The performance gap between Oracle-based and optimized mechanisms also increases slightly with the addition of the second exit. This effect arises because, in the two-exit model, each local component processes only half of the data, leading to lower-quality intermediate features ($\mathbf{z}_t^k$) for the remote component.

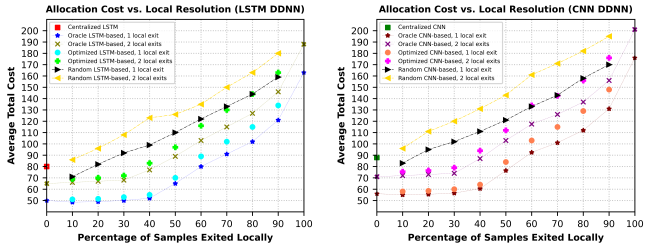
Key observation 4: Adding a second local exit (in parallel with the first) increases the complexity of the trade-off, resulting in higher overall costs while also offering more opportunities for local processing.

In Fig. 3(c) demonstrates the significant impact of weight allocation on DDNN performance, with training weights set to $(w_L, w_R) = (0.1, 0.9)$, placing low emphasis on local layers. While the optimal weight pair (e.g., $(0.9, 0.1)$) cannot be precisely determined in advance, our analysis across various scenarios consistently shows that assigning greater weight to the local exit is essential for achieving optimal performance.

Key observation 5: The selection of training weights (w_L, w_R) is a critical factor that significantly influences the model's overall performance.

Latency Reduction: This experiment investigates whether our model achieves latency reduction by evaluating both "communication" and "computation" times. For **communication time**, we reference a recent systems-oriented study [18], which

[5]This superior performance of our distributed model, achieved with lower overhead compared to the centralized model, can be attributed to the local exit's influence on gradient flow. This effect, observed in other studies [9], [13], [17], demonstrates that the local exit not only enhances operational efficiency but also provides a regularization effect, resulting in improved performance with reduced overhead, a mutually beneficial outcome.



(a) LSTM-based DDNN with 2 local exits (b) CNN-based DDNN with 2 local exits

Fig. 4: Total loss vs percentage of samples predicted locally

TABLE II: Latency Comparison (milliseconds per sample)

L (% of local resolution)	0	5	20	40	50	60	80	95	100
LSTM-based DDNN	42.72	40.40	34.05	25.43	21.09	16.78	8.20	1.75	1.25
CNN-based DDNN	42.70	40.36	34.01	25.37	21.02	16.70	8.10	1.65	1.25
Centralized LSTM	42.67	-	-	-	-	-	-	-	-
Centralized CNN	42.63	-	-	-	-	-	-	-	-

estimates the average round-trip transmission time (RTT) from edge to cloud to be 42.46 ms per sample. To measure **computation time**, we execute each model multiple times on the same server and calculate the average processing time per sample for various models[6].

Table II summarizes our findings, where "L" represents the percentage of samples processed locally, and "T" indicates the average time required to resolve a single sample in each scenario. For example, with 40% of samples resolved locally ("L" = 40%), the total average resolution time ("T") is calculated as the sum of the following: {the average processing time per sample for the local DNN, the average processing time per sample for the offloading block, 40% of the average local inference time per sample, 60% of the round-trip transmission (RTT) time from edge to cloud, and 60% of the average remote inference time per sample}. The results demonstrate that increasing the percentage of locally processed samples reduces inference latency. Notably, processing 50% of samples locally with the DDNN achieves performance parity with centralized baselines in terms of cost efficiency, while reducing inference latency by 49%. Additionally, we observe that the CNN-based DDNN outperforms the LSTM-based DDNN in terms of latency. It is important to note that preprocessing time is not included in this table.

Key Observation 6: The average inference latency diminishes as a greater proportion of samples are processed locally.

VI. Conclusions and Future Work

In this study, we developed Distributed Deep Neural Networks (DDNNs) tailored for forecasting traffic demand to optimize resource allocation in 5G networks. The proposed DDNNs incorporate multiple exits, including local exits at the edge and a remote exit in the cloud, forming hierarchical architectures where local components handle subsets of decisions, and cloud layers integrate information from these local components. Our findings show that LSTM- and CNN-based DDNNs can resolve a substantial portion of decisions locally, reducing latency and costs compared to centralized models, with LSTM-based DDNNs outperforming CNN-based ones. Furthermore, the optimized offloading mechanism effectively identifies samples for cloud processing, approaching the theoretical performance of an oracle. In future work, we plan to scale up experiments to evaluate performance in larger, more complex scenarios and to explore independent offloading decisions for local exits.

[6]It is important to note that the implementation setup for such a distributed architecture may vary. Our experiment uses practical parameters to effectively demonstrate the potential for latency reduction in a real-world scenario. The models are executed on a server equipped with an Nvidia V100 GPU, 16 GB HBM2, and 32 GB RAM.

References

[1] C. Zhang, M. Fiore, C. Ziemlicki, P. Patras, "Microscope: Mobile Service Traffic Decomposition for Network Slicing as a Service," MobiCom, no. 38, pp. 1–14, April 2020.

[2] C. Zhang, P. Patras and H. Haddadi, "Deep Learning in Mobile and Wireless Networking: A Survey," in IEEE Communications Surveys and Tutorials, vol. 21, no. 3, pp. 2224-2287, 2019.

[3] C. -X. Wang, M. D. Renzo, S. Stanczak, S. Wang and E. G. Larsson, "Artificial Intelligence Enabled Wireless Networking for 5G and Beyond: Recent Advances and Future Challenges," in IEEE Wireless Communications, vol. 27, no. 1, pp. 16-23, February 2020.

[4] Y. Matsubara, M. Levorato, and F. Restuccia, "Split Computing and Early Exiting for Deep Learning Applications: Survey and Research Challenges," ACM Computing Surveys, vol. 55, No. 90, pp. 1-30, 2023.

[5] Q. Liu, T. Han, and E. Moges, "Edgeslice: Slicing wireless edge computing network with decentralized deep reinforcement learning," ICDCS, Singapore, pp. 234-244, 2020.

[6] V. Sciancalepore, X. Costa-Perez and A. Banchs, "RL-NSB: Reinforcement Learning-Based 5G Network Slice Broker," in IEEE/ACM Transactions on Networking, vol. 27, no. 4, pp. 1543-1557, August 2019.

[7] Y. Liu, J. Ding and X. Liu, "A Constrained Reinforcement Learning Based Approach for Network Slicing," 2020 ICNP, Madrid, Spain, pp. 1-6, 2020.

[8] A. Ehsanian and T. Spyropoulos, "Distributed LSTM-based Slice Resource Allocation for Beyond 5G Networks," ICNC, Big Island, HI, USA, 2024, pp. 868-874.

[9] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," ICDCS, Atlanta, GA, USA, pp. 328-339, 2017.

[10] D. Bega, M. Gramaglia, M. Fiore, A. Banchs and X. Costa-Perez, "DeepCog: Cognitive Network Management in Sliced 5G Networks with Deep Learning," INFOCOM, Paris, France, pp. 280-288, 2019.

[11] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "Aztec: Anticipatory capacity allocation for zero-touch network slicing," INFOCOM, Toronto, Canada, pp. 794-803, 2020.

[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," CVPR, Boston, USA, pp. 1-9, 2015.

[13] S. Teerapittayanon, B. McDanel, and H. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," ICPR, pp. 2464-2469, 2016.

[14] Telecom Italia, "Milano Grid," https://doi.org/10.7910/DVN/QJWLFU, 2015.

[15] T. Giannakas, T. Spyropoulos and O. Smid, "Fast and accurate edge resource scaling for 5G/6G networks with distributed deep neural networks," WoWMoM, Belfast, United Kingdom, pp. 100-109, 2022.

[16] N. Liakopoulos, A. Destounis, G. Paschos, T. Spyropoulos, and P. Mertikopoulos, "Cautious regret minimization: Online optimization with long-term budget constraints," PMLR, vol. 97, pp. 3944–3952, 2019.

[17] Y. Kaya, S. Hong, and T. Dumitras, "Shallow-deep networks: Understanding and mitigating network overthinking," PMLR, vol. 97, pp. 3301–3310, 2019.

[18] K.-J. Hsu, K. Bhardwaj, and A. Gavrilovska, "Couper: DNN Model Slicing for Visual Analytics Containers at the Edge," Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, pp. 179–194, 2019.