SIGMETRICS: G: SplitPlace: Efficient Splitting and Placement of Deep Neural Networks on Mobile Edge Environments

Shreshth Tuli
Department of Computing, Imperial College London, UK
s.tuli20@imperial.ac.uk

ABSTRACT

In recent years, deep learning models have become ubiquitous in industry and academia alike. Modern deep neural networks (DNNs) can solve one of the most complex problems today, but coming with the price of massive compute and storage requirements. This makes deploying such massive neural networks challenging in the mobile edge computing paradigm, where edge nodes are resource-constrained, hence limiting the input analysis power of such frameworks. Semantic and layer-wise splitting of neural networks for distributed processing show some hope in this direction. However, there are no intelligent algorithms that place such modular splits to edge nodes for optimal performance. This work proposes a novel policy, SplitPlace, for efficient splitting of DNNs and placement of DNN split fragments on mobile edge hosts. Our experiments on physical mobile-edge environments with real-world workloads show that SplitPlace can significantly improve the state-of-the-art in terms of average response time, deadline violation rate, and total reward by up to 46%, 69% and 12% respectively.

1 PROBLEM AND MOTIVATION

Modern Deep Neural Networks (DNN) are becoming the backbone of many industrial tasks and activities [29]. As the computational capabilities of devices have improved, new deep learning models have been proposed. Such neural models are becoming increasingly demanding in terms of data and compute power to provide higher accuracy results in more challenging problems. Many recent DNN models have been shown to outperform the earlier shallow networks for more complex tasks like image segmentation, traffic surveillance, and healthcare [21, 24, 29]. Moreover, recently paradigms like mobile edge computing have emerged which provide robust and low latency deployment of Internet of Things (IoT) applications close to the edge of the network.

Challenges. However, mobile edge devices face the severe limitation of computational and memory resources as they rely on low power energy sources like batteries, solar or other energy scavenging methods [29]. This is not only because of the requirement of low cost but also the need for mobility in such nodes [28]. Herein, it is still possible to handle the processing limitations of massive DNN models by effective preemption and longer execution of jobs. However, memory bottlenecks are much harder to solve [28]. In a distributed edge environment where storage spaces a typically mapped to a networks-attached-media, large swap spaces impose very high network bandwidth overheads making high fidelity inference using DNNs hard [23, 27]. To deploy an upgraded AI model, tech-giants like Amazon, Netflix and Google need to revamp their infrastructure and upgrade their devices, raising many sustainability concerns [29]. This has made the integration of massive neural network models with such devices an expensive ordeal.

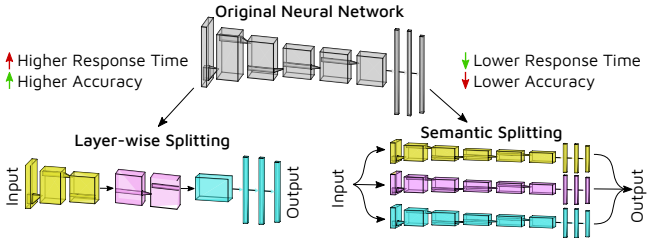


Figure 1: Overview of layer and semantic splitting strategies

Contributions. The only solution for this problem is the development of strategies that can accommodate large-scale DNNs within legacy infrastructures. Many prior efforts in this regard have been proposed [12, 14, 19] but they fail to provide a holistic strategy for not only distributed learning but also inference in such memory-constrained environments. This work proposes a novel neural splitting and placement policy, *SplitPlace*, for enhanced distributed neural network inference at the edge. SplitPlace allows modular neural models to be integrated for best result accuracies that could only be provided by cloud deployments. This project is part of a larger endeavor to efficiently integrate the hitherto disjoint fields of deep learning and distributed systems research [22].

2 BACKGROUND AND RELATED WORK

Model Compression. Other recent works offer lower precision models that can fit within the limited memory of such devices by using methods like Model Compression or Model Pruning [9, 11]. Efficient compression of DNN models has been a long studied problem in the literature [4]. Several works have been proposed that aim at the structural pruning of neural network parameters without significantly impacting the model's performance. These use approaches like tensor decomposition, network sparsification and data quantization [4]. Such pruning and model compression approaches have also been used by the systems research community to allow inference of massive neural models on devices with limited resources [3]. Recently, architectures like BottleNet and Bottlenet++ have been proposed [5, 18] to enable DNN inference on mobile cloud environments and reduce data transmission times. BottleNet++ compresses the intermediate layer outputs before sending them to the cloud layer. It uses a model re-training approach to prevent the inference being adversely impacted by the lossy compression of data. Further, BottleNet++ classifies workloads in terms of compute, memory and bandwidth bound categories and applies an appropriate model compression strategy. However, model compression does not leverage multiple compute nodes and has poor inference accuracy in general compared to semantic split execution (discussed in Section 4). Thus, SplitPlace does not use the model compression technique.

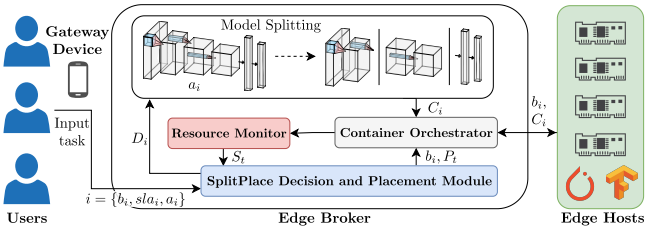


Figure 2: SplitPlace System Model

Neural Network Splitting. Recently, split neural network models have been proposed. They show that using semantic or layer-wise splitting, a large deep neural network can be fragmented into multiple smaller networks for dividing network parameters onto multiple nodes [7, 12, 17]. An overview of these two strategies is shown in Figure 1. The former partitions a neural network into parallel disjoint models that produce a part of the result. The latter partitions a neural network into sequential models that generate intermediate results. Layer splits provide higher accuracy and response time, whereas semantic splits provide lower values for both. SplitPlace leverages this contrast in traits to trade-off between inference accuracy and response time based on SLA requirements of the input tasks. Despite the considerable drop in inference accuracy when using semantic splitting scheme, it is still used in the proposed SplitPlace approach as it is better than model compression or early-exit strategies for quick inference. This is acceptable in many industrial applications [6, 11] where latency and service level agreements are more important performance metrics than high-fidelity result delivery. In this work, we consider a system with both SLA violation rates and inference accuracy as optimization objectives. This makes the combination of layer and semantic splitting a promising choice for such use cases. However, no appropriate scheduling policies exist that can intelligently place such modular neural fragments on a distributed infrastructure to optimize both accuracy and SLA together. The placement of such split models is non-trivial considering the diverse and complex dynamism of task distribution, model usage frequencies and geographical placement of mobile edge devices [1].

3 APPROACH AND UNIQUENESS

3.1 System Model and Problem Formulation

In this work, we assume a scenario with a fixed number of multiple heterogeneous edge nodes in a broker-worker fashion, which is a typical case in mobile-edge environments [2, 17, 18, 26]. Here, the broker node takes all resource management related decisions and tasks are executed on workers. Examples of broker nodes include personal laptops, small-scale servers and low-end workstations [26]. Example of common worker nodes in edge environments include Raspberry Pis, Arduino and similar System-on-Chip (SoC) computers [29]. Some worker nodes are assumed to be mobile, whereas others are considered to be fixed in terms of their geographical location. We periodically measure utilizations of CPU, RAM, Bandwidth and Disk for each task in the system. Moreover, we consider that tasks include a batch of inputs that need to be processed by a DNN model. Further, for each task, a service level deadline is defined at the time the task is sent to the edge environment. We give an overview of the SplitPlace system model in Figure 2.

Workload Model. We consider a bounded discrete time control problem where we divide the timeline into equal duration intervals, with the t-th interval denoted as I_t. Here, $t \in \{0, \ldots, T\}$, where T is the number of intervals in an execution. We assume a fixed number of host machines in the edge layer and denote them as H. We also consider that new tasks created at the interval I_t are denoted as N_t, with all active tasks being denoted as T_t (and $N_t \subseteq T_t$). Each task $i \in T_t$ consists of a batch input b_i, SLA deadline sla_i and a DNN application a_i. The set of all possible DNN applications is denoted by A. For each new task $i \in N_t$, the edge broker takes a decision D^i, such that $D^i \in \{L, S\}$, with L denoting layer-wise splitting and S denoting semantic split strategy. The collection of all split decisions for active tasks in interval I_t is denoted as $D_t = \{D^i\}_{i \in N_t}$. Based on the decision D^i for task i, this task is realized as an execution workflow in the form of containers C^i. Similar to a VM, a container is a package of virtualized software that contains all of the necessary elements to run in any environment. The set of all containers active in the interval I_t is denoted as $C_t = \cup_{i \in T_t} C^i$. The set of all utilization metrics of CPU, RAM, Network Bandwidth and Disk for all containers and hosts at the start of the interval I_t defines the state of the system, denoted as S_t.

Problem Formulation. The aim of the model is to optimize an objective score O_t (to be maximized), which quantifies the QoS parameters of the interval I_t, such as accuracy, SLA violation rate, energy consumption and average response time. To do this, we decompose the problem into two sub-problems of deciding the optimal splitting strategy for input tasks and that of placement of active containers in edge hosts.Thus, the SplitPlace model takes the split decision D^i for all $i \in N_t$. Moreover, it also takes a placement decision for all active containers C_t, denoted as an adjacency matrix $P_t : C_t \times H$. This is realized as a container allocation for new tasks and migration for active tasks in the system. The constraints in this formulation include the following. Firstly, the container decomposition for a new task $i \in N_t$ should be based on D^i. Secondly, containers corresponding to the layer-split decisions $\{C^i | D^i = L\}$ should be scheduled as per the linear chain of precedence constraints. This is because the output of an initial layer in an inference pipeline of a neural network is required before we can schedule a latter layer in the pipeline. Thirdly, the placement matrix $P_t : C_t \times H$ should adhere to the allocation constraints, *i.e.*, it should not allocate/migrate a container to a host where the host does not have sufficient resources available to accommodate the container. The problem can be formulated as

$$\begin{aligned} \max_{P_t, D_t} \quad & \sum_{t}^{T} O_t \\ \text{s. t.} \quad & \forall\, t, \forall\, i \in N_t, C^i \text{ containers created based on } D^i, \\ & \forall\, t, P_t \text{ is feasible}, \forall\, D^i = L, C^i \text{ follow precedence.} \end{aligned} \tag{1}$$

3.2 SplitPlace Policy

As described previously, we decompose the problem into *splitting* and *placement* decisions. The motivation behind this is to avoid the exponential state-space explosion when considering the problem of joint optimization of both decisions.

MAB Based Splitting. The idea behind the proposed SplitPlace approach is to maintain MABs for two different contexts: 1) when

SLA is greater than the estimate of the response time for a layer decision, 2) when SLA is less than this estimate. The motivation behind these two contexts is that in case of the SLA deadline being lower than the execution time of layer split, a "layer" decision would be more likely to violate the SLA as result delivery would be after the deadline. However, the exact time it takes to completely execute all containers corresponding to the layer split decision is apriori unknown. Thus, for every application type, we maintain estimates of the response time, *i.e*, the total time it takes to execute all containers corresponding to this decision.

Let us denote the tasks leaving the system at the end of I_t as E_t. Now, for each task $i \in E_t$, we denote response time and inference performance using r_i and p_i. We denote the layer response time estimate for application $a \in A$ as R^a. To quickly adapt to non-stationary scenarios, for instance due to the mobility of edge nodes in the system, we update our estimates using new data-points as exponential moving averages using the multiplier $\phi \in [0, 1]$ for the most recent response time observation.

$$R^a \leftarrow \phi \cdot r_i + (1 - \phi) \cdot R^a, \ \forall i \in E_t \wedge D^i = L, \ \forall a \in A. \quad (2)$$

Now, for any input task $i \in N_t$, we divide it into two cases: $sla_i \geq R^{a_i}$ and $sla_i < R^{a_i}$. Considering that the response time of a semantic-split decision would likely be lower than the layer-split decision, in the first case both decisions would most likely not lead to an SLA violation (*high SLA* setting). However, in the second case, a layer-split decision would likely lead to an SLA violation but not the semantic-split decision (*low SLA* setting). To tackle the problem for these different contexts, we maintain two independent MAB models denoted as MAB^h and MAB^l. For each context and decision $d \in \{L, S\}$, we define reward metrics as

$$O^{h,d} = \frac{\sum_{i \in E_t} (\mathbb{1}(r_i \leq sla_i) + p_i) \cdot \mathbb{1}(sla_i \geq R^{a_i} \wedge D^i = d)}{2 \cdot \sum_{i \in E_t} \mathbb{1}(sla_i \geq R^{a_i} \wedge D^i = d)}, \quad (3)$$

$$O^{l,d} = \frac{\sum_{i \in E_t} (\mathbb{1}(r_i \leq sla_i) + p_i) \cdot \mathbb{1}(sla_i < R^{a_i} \wedge D^i = d)}{2 \cdot \sum_{i \in E_t} \mathbb{1}(sla_i < R^{a_i} \wedge D^i = d)}. \quad (4)$$

The first term of the numerator, *i.e.*, $\mathbb{1}(r_i \leq sla_i)$ quantifies SLA violation reward (one if not violated and zero otherwise). The second term, *i.e.*, p_i corresponds to the inference accuracy of the task. Thus, each MAB model gets the reward function for its decisions allowing independent training of the two.

Now, for each decision context $c \in \{h, l\}$ and $d \in \{L, S\}$, we maintain a decision count $N^{c,d}$ and a reward estimate $Q^{c,d}$ which is updated using the reward functions $O^{h,d}$ or $O^{l,d}$ as follows

$$Q^{c,d} \leftarrow Q^{c,d} + \gamma(O^{c,d} - Q^{c,d}), \ \forall d \in \{L, S\}, \ \forall c \in \{h, l\}. \quad (5)$$

where γ is the decay parameter. Thus, each reward-estimate is updated by the corresponding reward metric.

To train the model, we take the contextual decision

$$D^i = \begin{cases} \begin{cases} \text{random decision}, & \text{with prob. } \epsilon \\ \arg\max_{d \in \{L,S\}} Q^{h,d}, & \text{otherwise} \end{cases}, & sla_i \geq R^{a_i} \\ \begin{cases} \text{random decision}, & \text{with prob. } \epsilon \\ \arg\max_{d \in \{L,S\}} Q^{l,d}, & \text{otherwise} \end{cases}, & sla_i < R^{a_i} \end{cases}. \quad (6)$$

Here, the probability ϵ decays using the reward feedback, starting from 1. We maintain a reward threshold ρ that is initialized as a small positive constant $k < 1$, and use average reward $O^{MAB} = \frac{1}{4} \sum_{c \in \{h,l\}} \sum_{d \in \{L,D\}} O^{c,d}$ to update ϵ and ρ as $\epsilon \leftarrow (1 - k) \cdot \epsilon$ and $\rho \leftarrow (1 + k) \cdot \rho$, when $O^{MAB} > \rho$. The k value controls the rate of convergence of the model. The ϵ value controls the exploration of the model at training time allowing the model to visit more states and obtain precise estimates of the layer-split response times.

However, at test time we already have precise estimates of the response times; thus exploration is only required to adapt in volatile scenarios. For this, ϵ-greedy is not a suitable approach as decreasing ϵ with time would prevent exploration as time progresses. Instead, we use an Upper-Confidence-Bound (UCB) exploration strategy that is more suitable as it takes decision counts also into account [10, 31]. Thus, at test time, we take a deterministic decision using the rule

$$D^i = \begin{cases} \arg\max_{d \in \{L,S\}} Q^{h,d} + c\sqrt{\frac{\log t}{N^{h,d}}}, & sla_i \geq R^{a_i} \\ \arg\max_{d \in \{L,S\}} Q^{l,d} + c\sqrt{\frac{\log t}{N^{l,d}}}, & sla_i < R^{a_i} \end{cases}, \quad (7)$$

t is the scheduling interval count and c is the exploration factor.

Surrogate Optimization based Placement. Once we have the splitting decision for each input task $i \in N_t$, we now can create containers using pre-trained layer and semantic split neural networks corresponding to the application a_i. This can be done offline on a resource rich system, where the split models can be trained using existing datasets. Then we need to place containers of all active tasks C_t to hosts H. To do this, we use a learning model which predicts the placement matrix P_t using the state of the system S_t, decisions $D_t = D^i \forall i \in T_t$ and a reward signal O^P. To define O^P, we define the following metrics [25, 27]. 1) *Average Energy Consumption* (AEC) is defined for any interval I_t as the mean energy consumption of all edge hosts in the system. 2) *Average Response Time* (ART) is defined for any interval I_t as mean response time (in scheduling intervals) of all leaving tasks E_t.

Using these metrics, for any interval I_t, O^P is defined as

$$O^P = O^{MAB} - \alpha \cdot AEC_t - \beta \cdot ART_t. \quad (8)$$

Here, α and β (such that $\alpha + \beta = 1$) are hyper-parameters that can be set by users as per the application requirements.

In the proposed framework, we use decision-aware surrogate based optimization method (termed as DASO) to place containers in a distributed mobile edge environment. This is motivated from our prior neural network based surrogate optimization method called Gradient Optimization using Backpropagation to Input (GOBI) [27]. Unlike vanilla GOBI, DASO is split-decision aware. Here, we consider a Fully-Connected-Network (FCN) model $f(x; \theta)$ that takes an x as a tuple of input state S_t, split-decision D_t and placement decision P_t, and outputs an estimate of the QoS objective score O_t. Now, using existing execution trace dataset, $\Lambda = \{[S_t, P_t, D_t], O_t\}_b$, the FCN model is trained to optimize its network parameters θ such that the Mean-Square-Error (MSE) loss

$$\mathcal{L}(f(x; \theta), y) = \frac{1}{b} \sum_{t=0}^{b} (y - f(x; \theta))^2, \text{ where } (x, y) \in \Lambda. \quad (9)$$

is minimized as in [27]. To do this, we use AdamW optimizer [15] and update θ up till convergence. This allows the surrogate model f to predict an QoS objective score for a given system state S_t, split-decisions D_t and task placement P_t. Once the surrogate model is trained, starting from the placement decision from the previous interval $P_t = P_{t-1}$, we leverage it to optimize the placement decision using the following rule

$$P_t \leftarrow P_t - \eta \cdot \nabla_{P_t} f([S_t, P_t, D_t]; \theta), \quad (10)$$

Table 1: Comparison with baseline and ablation models.

Model	Energy	Sched. Time	Jain's Index	Wait Time	Resp. Time	SLA Viol.	Acc.	Avg. Reward
Baselines								
MC	1.1368	8.84	0.65	1.15	6.85	0.26	89.93	83.98
Gillis	1.1442	**8.22**	**0.89**	1.40	8.39	0.22	91.90	84.17
Ablation								
S+G	1.1112	8.68	0.68	1.08	**3.70**	0.14	89.04	83.91
L+G	1.1517	8.72	0.88	1.52	9.92	0.62	**93.17**	64.87
R+D	1.1297	8.86	0.62	**1.00**	5.55	0.29	90.71	81.62
M+G	1.1290	9.12	0.78	1.13	5.64	0.10	91.45	90.18
SplitPlace Model								
SplitPlace	**1.0867**	9.32	0.73	1.09	4.50	**0.08**	92.72	**94.18**

for a given state and decision pair S_t, D_t. Here, η is the learning rate of the model. The above equation is iterated till convergence, *i.e.*, the $L2$ norm between the placement matrices of two consecutive iterations is lower than a threshold value. Thus, at the start of each interval I_t, using the output of the MAB decision module, the DASO model gives us a placement decision P_t.

SplitPlace Algorithm. Using pre-trained MAB models, *i.e.*, Q-estimates $Q^{c,d}$ and decision counts $N^{c,d}$, the model decides the optimal splitting decision using the UCB metric. To adapt the model in non-stationary scenarios, we dynamically update the Q-estimates and decision counts. Using the current state and the split-decisions of all active tasks, we use the DASO approach to take a placement decision for the active containers. Again, we fine-tune the DASO's surrogate model using the reward metric to adapt to changes in the environment, for instance the changes in the latency of mobile edge nodes and their consequent effect on the reward metrics. However, the placement decision must conform to the allocation constraints. To relax the constraint of having only feasible placement decisions, in SplitPlace we allocate or migrate only those containers for which it is possible. Those containers that could not be allocated in a scheduling interval are placed to nodes corresponding to the highest output of the neural network f. If no host placement is feasible the task is added to a wait queue, which are considered again for allocation in the next interval.

4 RESULTS AND CONTRIBUTIONS

To implement SplitPlace, we build upon the container orchestration primitives provided by our COSCO framework [27]. SplitPlace uses HTTP RESTful APIs for communication and seamlessly integrates a `Flask` based web-environment to deploy and manage containers in a distributed setup [8].

Setup. As our experimental setup, we use an edge computing cluster of 16 Raspberry Pi 4B nodes as shown in Figure 3. This cluster consists of eight 4-GB RAM nodes and another eight 8-GB RAM nodes. Instead of using the watt-meter, we consider the power consumption models from the commonly-used Standard Performance Evaluation Corporation (SPEC) benchmarks repository [20]. We run all experiments for 100 scheduling intervals, with each interval being 5 minutes long, giving a total experiment time of 8 hours 20 minutes. The scheduling and the online fine-tuning of the model was performed by the edge broker with configuration: laptop with Intel i3-1115G4 and 8GB RAM. To factor in the mobility of the edge nodes, we use the `NetLimiter` tool to tweak the communication latencies using the mobility model described in the Simulation of Urban Mobility (SUMO) tool [13] that emulates mobile vehicles in a city like setup.

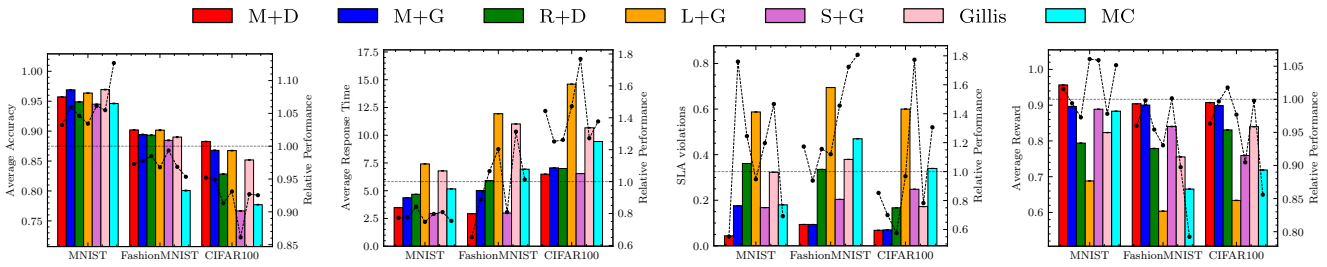
Figure 3: Raspberry Pi Cluster

Workloads. Motivated from prior work [30], we use three families of popular DNNs as the benchmarking models: ResNet50-V2, MobileNetV2 and InceptionV3 [16]. These models have been taken directly from the AIoTBench workloads [16]. This is a popular suite of AI benchmark applications for IoT and Edge computing solutions. Each family has many variants of the model, each having a different number of layers in the neural model. For instance, the ResNet model has 34 and 50 layers. We use three image-classification data sets: MNIST, FashionMNIST and CIFAR100 [16]. At the beginning of each scheduling interval, we create $Poisson(\lambda)$ tasks with $\lambda = 6$ tasks for our setup, sampled uniformly from one of the three applications as per [27]. Thus the application set A becomes {MNIST, FashionMNIST, CIFAR100}.

Baselines and Ablation Models. We compare the performance of the SplitPlace approach against the state-of-the-art baselines *Gillis* [30] and BottleNet++ Model Compression (denoted as *MC* in our graphs) [18]. We also compare SplitPlace with ablated models, where we replace one or both of the MAB or DASO components with simpler versions as described below. *Semantic+GOBI (S+G)*: Semantic-split decision only with vanilla GOBI placement module. *Layer+GOBI (L+G)*: Layer-split decision only with vanilla GOBI placement module. *Random+DASO (R+D)*: Random split decision with DASO placement module. *MAB+GOBI (M+G)*: MAB based split decider with vanilla GOBI placement module.

Results. As in prior work [27], we use $\alpha = \beta = 0.5$ in (8) for our experiments. Also, we use the exploration factor $c = 0.5$ for the UCB exploration and the exponential moving average parameter $\phi = 0.9$, chosen using grid-search using the cumulative reward as the metric to maximize. Table 1 shows the results of our experiments comparing the cumulative AEC and ART values, scheduling time (seconds), Jain's fairness index, task wait time (seconds), SLA violation rates, task accuracy and reward (as per eq 8). Compared to the baselines, SplitPlace can reduce energy consumption by up to $4.41\% - 5.03\%$ giving an average energy consumption of 1.0867 MW-hr. However, the SplitPlace approach has higher scheduling time and lower fairness index. The Gillis baseline has the highest fairness index of 0.89, however this index for SplitPlace is 0.73. SplitPlace has a higher overhead of 11.8% compared to the Gillis baseline in terms of scheduling time.

Robustness to Workloads. Figure 4 shows the performance parameters for all models in the three workload settings. The graphs show contrasting workload specific trends. For instance, the accuracy of the MNIST application is higher and response time lower than the other two application types. Thus, for all models, the average accuracy is higher compared to the original workload and response time lower. However, due to lower resource requirements of the MNIST workload, the MC model seldom takes the decision



(a) Average Accuracy (b) Average Response Time (c) Fraction of SLA Violations (d) Average Reward

Figure 4: Comparison of SplitPlace against baselines and ablated models with MNIST, FashionMNIST and CIFAR100 workloads.

to use compressed models, giving a high accuracy uplift compared to the original setup. Even with these constrained workloads, the SplitPlace model has the highest average accuracy with the lowest average response times and SLA violation rates. Overall, we see that the average reward of the SplitPlace model (89.83 − 95.36) is highest among all methods. The baseline models, Gillis and MC have rewards in the range 75.54 − 83.98 and 66.57 − 88.31 respectively.

5 CONCLUSIONS

In this work, we present SplitPlace, a novel framework for efficiently managing demanding neural network based applications. SplitPlace exploits the trade-off between layer and semantic split models where the former gives higher accuracy, but the latter gives much lower response times. This allows SplitPlace to not only manage tasks to maintain high inference accuracy on average, but also reduce SLA violation rate. SplitPlace outperforms the baseline models in terms of average response time, SLA violation rate, inference accuracy and total reward by up to 46.3%, 69.2%, 3.1% and 12.1% respectively in a mobile heterogeneous edge environment with real-world AI based workloads.

ACKNOWLEDGMENTS

The author is supported by the President's Ph.D. Scholarship at the Imperial College London. This project is part of the author's Ph.D. work, co-supervised by Dr. Giuliano Casale and Prof. Nick Jennings.

REFERENCES

[1] E. Ahmed and M. H. Rehmani. Mobile edge computing: Opportunities, solutions, and challenges. *Future Generation Computer Systems*, 70:59–63, 2017.
[2] D. Basu, X. Wang, Y. Hong, H. Chen, and S. Bressan. Learn-as-you-go with megh: Efficient live migration of virtual machines. *IEEE Transactions on Parallel and Distributed Systems*, 30(8):1786–1801, 2019.
[3] P. S. Chandakkar, Y. Li, P. L. K. Ding, and B. Li. Strategies for re-training a pruned neural network in an edge computing paradigm. In *2017 IEEE International Conference on Edge Computing (EDGE)*, pages 244–247. IEEE, 2017.
[4] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020.
[5] A. E. Eshratifar, A. Esmaili, and M. Pedram. Bottlenet: A deep learning architecture for intelligent mobile cloud computing services. In *IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 1–6, 2019.
[6] A. Goli, O. Hajihassani, H. Khazaei, O. Ardakanian, M. Rashidi, and T. Dauphinee. Migrating from monolithic to serverless: A fintech case study. In *ACM/SPEC International Conference on Performance Engineering*, pages 20–25, 2020.
[7] V. S. Gordon and J. Crouson. Self-splitting modular neural network-domain partitioning at boundaries of trained regions. In *2008 IEEE International Joint Conference on Neural Networks*, pages 1085–1091, 2008.
[8] M. Grinberg. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.
[9] J. R. Gunasekaran, C. S. Mishra, P. Thinakaran, M. T. Kandemir, and C. R. Das. Implications of public cloud resource heterogeneity for inference serving. In *International Workshop on Serverless Computing*, pages 7–12, 2020.
[10] S. Gupta, G. Joshi, and O. Yağan. Correlated multi-armed bandits with a latent random source. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3572–3576, 2020.
[11] J. Huang, C. Samplawski, D. Ganesan, B. Marlin, and H. Kwon. CLIO: Enabling automatic compilation of deep learning pipelines across IoT and Cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–12, 2020.
[12] J. Kim, Y. Park, G. Kim, and S. J. Hwang. Splitnet: Learning to semantically split deep networks for parameter reduction and model parallelization. In *International Conference on Machine Learning*, pages 1866–1874, 2017.
[13] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker. Recent development and applications of sumo-simulation of urban mobility. *International journal on advances in systems and measurements*, 5(3&4), 2012.
[14] W. Y. B. Lim et al. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(3):2031–2063, 2020.
[15] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018.
[16] C. Luo, F. Zhang, C. Huang, X. Xiong, J. Chen, L. Wang, W. Gao, H. Ye, T. Wu, R. Zhou, et al. AIoT bench: towards comprehensive benchmarking mobile and embedded device intelligence. In *International Symposium on Benchmarking, Measuring and Optimization*, pages 31–35. Springer, 2018.
[17] Y. Matsubara, S. Baidya, D. Callegaro, M. Levorato, and S. Singh. Distilled split deep neural networks for edge-assisted real-time systems. In *Workshop on Hot Topics in Video Analytics and Intelligent Edges*, pages 21–26, 2019.
[18] J. Shao and J. Zhang. Bottlenet++: An end-to-end approach for feature compression in device-edge co-inference systems. In *IEEE International Conference on Communications Workshops*, pages 1–6, 2020.
[19] Y. Shi, K. Yang, T. Jiang, J. Zhang, and K. B. Letaief. Communication-efficient Edge AI: Algorithms and systems. *IEEE Communications Surveys & Tutorials*, 22(4):2167–2191, 2020.
[20] Standard Performance Evaluation Corporation. SPEC Power Consumption Models. https://www.spec.org/cloud_iaas2018/results/.
[21] S. Tuli. SplitPlace: Intelligent Placement of Split Neural Nets in Mobile Edge Environments. *SIGMETRICS Perform. Eval. Rev.*, 2021.
[22] S. Tuli, G. Casale, and N. R. Jennings. SplitPlace: AI Augmented Splitting and Placement of Large-Scale Neural Networks in Mobile Edge Environments. *IEEE Transactions on Mobile Computing (under submission)*, 2021.
[23] S. Tuli, G. Casale, and N. R. Jennings. CAROL: Confidence-Aware Resilience Model for Edge Federations. In *International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2022.
[24] S. Tuli, G. Casale, and N. R. Jennings. TranAD: Deep Transformer Networks for Anomaly Detection in Multivariate Time Series Data. *Proc. VLDB (2022)*, 2022.
[25] S. Tuli et al. HUNTER: AI based Holistic Resource Management for Sustainable Cloud Computing. *Journal of Systems and Software*, 2021.
[26] S. Tuli, R. Mahmud, S. Tuli, and R. Buyya. Fogbus: A blockchain-based lightweight framework for edge and fog computing. *Journal of Systems and Software*, 2019.
[27] S. Tuli, S. R. Poojara, S. N. Srirama, G. Casale, and N. R. Jennings. COSCO: Container Orchestration Using Co-Simulation and Gradient Based Optimization for Fog Computing Environments. *IEEE Transactions on Parallel and Distributed Systems*, 33(1):101–116, 2022.
[28] S. Tuli, S. Tuli, G. Casale, and N. R. Jennings. Generative optimization networks for memory efficient data generation. *NeurIPS, Workshop on ML for Systems*, 2021.
[29] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(2):869–904, 2020.
[30] M. Yu, Z. Jiang, H. C. Ng, W. Wang, R. Chen, and B. Li. Gillis: Serving large neural networks in serverless functions with automatic model partitioning. In *International Conference on Distributed Computing Systems*, 2021.
[31] Y. Zhang, P. Cai, C. Pan, and S. Zhang. Multi-agent deep reinforcement learning-based cooperative spectrum sensing with upper confidence bound exploration. *IEEE Access*, 7:118898–118906, 2019.