A Comparative Analysis of Proactive and Reactive Methods for Privacy-Aware Interleaved DNN Offloading

Jamie Cotter

Follow this and additional works at: https://sword.cit.ie/dptcompsciart
A Comparative Analysis of Proactive and Reactive Methods for Privacy-Aware Interleaved DNN Offloading.

Jamie Cotter, Ignacio Castineiras, Donna O’Shea, Victor Cionca
Dept. of Computer Science
Munster Technological University
Cork, Ireland
jamie.cotter@mycit.ie, ignacio.castineiras@mtu.ie, donna.oshea@mtu.ie, victor.cionca@mtu.ie

Abstract—Deep Neural Networks (DNNs) are increasingly used on the edge in mobile and other resource-constrained devices for inference tasks ranging from object detection and image recognition to video processing. Many of these tasks have low latency requirements that cannot be satisfied if they are processed locally due to their high computational complexity. Offloading computation to the edge and cloud offers a way to alleviate this computational latency. Doing so introduces communication delays, which makes offloading a balancing act between the benefits of reduced processing time and the communication delays incurred. Existing algorithms for DNN offloading based on DNN partitioning are optimised for handling successive tasks on a single remote server, and perform sub-optimally when tasks are interleaved, or with multiple servers, or when privacy concerns require local processing of certain DNN layers. A viable alternative are generic computational offloading algorithms (GOAs), which can break down DNN tasks into their components and perform fine-grained offloading. We perform a simulation-based comparison of traditional GOAs with various levels of proactivity and offloading constraints and a naive DNN partitioning approach. We identify key requirements for offloading algorithms in the ability to create processing chains between several remote servers in order to reduce communication overheads, and the ability to prioritise already running tasks. The results confirm the expectations about the shortcomings of DNN partitioning, and show that GOA algorithms can provide a significant performance improvement.

Index Terms—Mobile Edge Computing, Distributed Computing, DNN Offloading, Interleaved Offloading, Edge Computing, Computational Offloading

I. INTRODUCTION

Deep Neural Networks (DNNs) are increasingly used on mobile devices for inference tasks ranging from basic text input prediction, speech recognition, translation, image and video processing [1], to complex object detection, classification, etc. [2], [3]. Despite the increase in computational power for mobile devices, they are still unable to process DNNs fast enough to satisfy the low latency requirements that many of these applications possess [4]. The solution employed nowadays is computational offloading [5], [6], where the complex tasks are sent for processing to a remote and more powerful server located at the edge or in the cloud, which can provide a faster response. However, when offloading a task it is also necessary to transfer its input data and collect its outputs, which introduces communication delays. An important challenge in computational offloading is balancing the reduction in computational time with the increase in communication delay so that the overall processing time can be minimised. This paper provides an empirical comparative analysis of two DNN offloading solution approaches: DNN partitioning [7], [8] and Generic Offloading Algorithms (GOAs) [5], [6].

DNN partitioning [7], [8] is currently the favoured approach in computational offloading of DNN tasks. The algorithm partitions (splits) the DNN in two, running the DNN layers of the first partition on the mobile device and those of the second in the cloud. The partition point is selected to minimise the total processing time of the DNN, equal to the local and remote processing plus the communication time. This technique has been shown to perform well in scenarios with a single user and a single remote host. There are, however, several real-world settings that have not been evaluated and where DNN partitioning and offloading does not perform well. First, if the source devices move during the processing of a DNN, they may lose connection to the remote host and access to the DNN result. Next, privacy concerns may dictate that some layers be processed exclusively on the device [9], which may require a multi-split DNN partitioning scheme if those layers are in the middle of the DNN. Finally, our results show that if multiple DNN requests are generated in an interleaved fashion by multiple applications, either on the same device or on multiple devices, DNN partitioning will result in bottlenecks and long waiting times. The common issue in these scenarios is that the offloading of whole blocks (partitions) of layers is too coarse and inflexible and therefore a finer-grained approach to computational offloading is required. GOAs provide a viable alternative.

GOAs [5], [6] work with applications composed of a relatively small set of computationally intensive, interdependent tasks structured as a Directed Acyclic Graph (DAG). GOAs process each task from an application DAG individually and can offload it to any available computational host. As a result, GOAs are more flexible and finer-grained than DNN
partitioning, which satisfies the requirements of the scenarios listed above. GOAs have not yet been applied on DNNs, however the above characteristics provided the motivation to conduct such an evaluation in this paper.

There are two important differences between the structure of a DNN and the generic application DAGs considered by GOAs, which can raise objections to the application of GOAs to DNNs. First, traditional DNNs (e.g., MobileNet) are structured into a chain of layers instead of a complex DAG with multiple branches, therefore the use of a GOA could be considered unnecessarily complicated. However, as shown in Fig. 1, modern DNNs such as Inception [10] use parallel branches and are therefore increasingly DAG-like and a suitable target for GOAs. The second difference concerns the number and computational complexity of the components of DNNs and application DAGs. DNNs are composed of a large number of relatively low complexity layers, while application DAGs are composed of a low number of relatively high complexity tasks, the exact opposite. Applying GOA to a DNN would require handling each DNN layer separately, which will introduce large communication costs. The question investigated in this paper is whether the performance gains from using GOA to offload DNN inference tasks in the scenarios listed above with interleaved DNN tasks, non-offloadable DNN layers, and mobility, outweigh the losses from increased communication.

This paper provides the first comparison between GOAs and DNN partitioning in the context of computational offloading of state of the art DNNs [10] as well as on generic DAG-like applications. Three types of GOAs are considered: fully reactive, fully proactive, and hybrid, exploring the spectrum between computation and communication overhead. The scenarios of interleaved requests and non-offloadable DNN layers are considered, with mobility left for future investigation. The empirical analysis using simulations lead to the following conclusions:

- DNN partitioning suffers poor performance when presented with interleaved requests as well as non-offloadable layers due to extreme bottlenecks introduced by overuse of the mobile device.
- With interleaved requests and non-offloadable layers the highest performance is obtained with a fully proactive GOA.
- A hybrid GOA with only one DAG node look-ahead can achieve similar results as the fully proactive GOA in most conditions, except when having to deal with heavy loads.
- Reactive approaches suffer poor DAG node completion when the computation benefits of offloading are overshadowed by communication delays.
- While a high DAG node completion rate is beneficial, it does not guarantee an algorithm will complete more DAGs than another algorithm with fewer DAG nodes completed.
- Proactivity is the deciding factor for an offloading solution approach to achieve a higher rate of DAGs completed.

The paper continues with related work in DNN partitioning and generic offloading in Section II. Then, Section III presents the problem of DNN offloading, and Section IV presents the partitioning and generic offloading algorithms. The evaluation setup is presented in Section V with the results presented in Section VI and discussed in Section VII. Finally, Section VIII provides some conclusions and future work.

II. RELATED WORK

Much of the current work in DNN offloading focuses on DNN partitioning. It is a technique introduced by Kang et al. [8], which aims to minimise the impact of communication delays on cloud-processed DNNs and take advantage of increasingly powerful mobile system-on-chips (SoCs). Kang et al. propose partitioning (splitting) a DNN modelled as a DAG in two parts, allocating the first part that requires more input to the mobile device and the second to the cloud. Expanding on this concept, Hu et al. [11] devised a solution that partitions the DNN using either a graph min-cut algorithm or a heuristic, depending on the communication network conditions. Additional work includes Laskaridis et al. [12] making use of an early exit policy for DNN partitioning when communication overhead makes offloading untenable and Jeong et al. [7] considering the uploading of the model as well as the data and proposing incremental offloading of multiple partitions. While most of the work done in DNN offloading is practically evaluated, the aforementioned works do not consider the performance of their solution approaches when offloading multiple DNNs into the same network, DNNs with non-offloadable layers and the rest of scenarios identified in Section I.

Research in GOAs is more established than DNN partitioning, considering more scenarios and constraints. A wide range of algorithms have been applied, including reactive ones that process and offload each task independently, therefore incurring maximum communication costs; proactive ones, that perform resource allocation for all tasks at the start, usually with complete solutions, achieving minimum communication costs but inflexible to changes in resource availability; and hybrid ones, that balance communication with computation. GOAs can be applied to DNNs as they often focus on applications with low latency requirements. Habak et al. [6] examine computational offloading of interleaved DAG-based applications in an edge mesh type scenario with a centralised controller for task allocation. Work by De Maio et al. [5] focuses on a three-tier computational hierarchy composed of mobile devices, edge
and cloud, proposing a reactive offloading heuristic that aims to balance the minimisation of run-time, monetary cost and energy expended when offloading applications. Using a peer-to-peer reactive approach, Zhou et al. [13] provide a solution that considers deadline constraints for their applications offloading to peer devices as individual tasks are ready. Cui et al. [14] propose an algorithm for Mobile Edge Cloud (MEC) offloading that minimises latency as well as energy consumption. Although GOAs often include an implementation, they are often validated via crafted applications with artificial and abstract resource values, which may affect the efficiency of their solutions in real-world scenarios.

There are surveys providing a comparative analysis of offloading methods and scenarios for GOAs or for DNNs, however, there are no surveys that compare methods for GOAs with methods for DNNs.

Zheng et al. [15] survey generic computational offloading under various factors that determine where and when to offload, such as the network heterogeneity, host locality and mobility of devices, as well as categorising the multiple types of offloading interactions between computational hosts in a three-tier hierarchy composed of IoT, Edge and Cloud.

Similarly, Zaman et al. [16] survey recent mobility-aware works within the Mobile Edge Cloud (MEC) and formulate a taxonomy based on constraints, optimisation objectives, algorithms, mobility models and support networks commonly found in the MEC.

To the best of our knowledge, this is the first comparative analysis of GOAs against DNN offloading solutions when faced with non-offloadable entities, interleaved applications and realistic network bandwidth and latency conditions.

III. SYSTEM MODEL

This work considers the system in Fig. 2 where a device is running a set of applications, each generating DNN inference tasks (e.g. object recognition and/or classification). To complete the inferences as quickly as possible, the device has the option of breaking down the structure of the DNN into a DAG of DNN layers and running them locally or offloading individual or blocks of layers to a set of edge servers located nearby, or a cloud data centre. Layers that are offloaded must have their input data sent to the remote host, and their output data collected. This leads to a trade-off between computation and communication, depending on where layers are run:

- the mobile device is limited in computational power and capacity, however, has no communication overhead
- edge hosts incur minimal communication penalties due to their proximity to the mobile device and possess powerful CPUs, but are limited in resource capacity
- cloud hosts are bounded by neither computational power nor capacity but incur a heavy communication penalty due to their distance from the mobile client.

DNN tasks issued by concurrent applications are processed in interleaved fashion by an offloading algorithm that is executed on the device. The algorithm treats a DNN as a DAG and DNN layers as DAG tasks. The algorithm assigns DAG tasks to computational hosts based on the availability of communication and computational resources and orchestrates the execution of the DAG sending and directing task input and output data.

We consider the following types of offloading algorithms:

- A basic reactive GOA that allocates DAG tasks independently and when they are ready to run and seeks to minimise time on a task-by-task basis factoring in input/output communication time and processing time.
- An extended reactive GOA similar to the previous, but that prioritises non-offloadable DAG tasks.
- A proactive GOA that allocates all tasks in the DAG immediately upon initial offloading.
- A hybrid GOA that preallocates DAG task children as soon as all of their parents have commenced execution.

We start by presenting the model of the DAGs, the model of the computational network, and then we present an overview of the five algorithms.

A. DAG Overview

The DAGs considered by the offloading algorithm are represented as $G = (L_G, D_G)$ where $L_G = \{L_i | i = 1, n\}$ is the set of all the tasks that compose the DAG and $D_G = \{d_{ji} | i, j \in L_G\}$ describes the set of dependencies between the tasks. Dependencies between tasks lead to parent-child relations. We define the functions $\text{parents}(l_i) = \{l_j \in L_G | d_{ji} \in D_G\}$ to obtain the set of parent tasks for a given task and $\text{children}(l_i) = \{l_j \in L_G | d_{ji} \in D_G\}$ to obtain the children of a task.

We assign to every $l_i \in L_G$ a set of properties Name, TimeCloud, TimeEdge, TimeMobile, RAM, DataIn, DataOut, Storage, Offload, where Name$(l_i)$ is the name of the DAG task/layer (e.g. Convolutional); TimeCloud$(l_i)$, TimeEdge$(l_i)$ and TimeMobile$(l_i)$ represent the expected processing time (measured in seconds) for the task when executed in the cloud, edge and mobile device, respectively; RAM$(l_i)$ is the amount of RAM the layer requires in GBs;
DataOut($l_i$) is the expected size of the output of the task in megabytes; DataIn($l_i$) = $\sum_{l \in \text{parents}(l_i)}$ DataOut($l_j$) is the size of the input the task requires in megabytes, computed as the sum of all of its parent's outputs; Storage($l_i$) = DataIn($l_i$) + DataOut($l_i$) represents the amount of storage space a task requires for I/O in megabytes; Offload($l_i$) $\in \{0, 1\}$ is a boolean value that describes whether or not a layer can be offloaded.

**B. Network Model**

The network is structured as a fully connected graph which we define as $N = \{X_N, E_N\}$ where $X_N$ is our set of computational hosts and $E_N$ is our set of links between these hosts. Computational hosts are defined as $X_N = \{\chi_i | i \in 1,N\}$ with the set of properties GPU, GPU_Type, RAM, Storage, Id, Tasks, Reservations, Type, where Type($\chi_i$) is the number of GPUs the host possesses, GPU_Type($\chi_i$) which denotes the type of GPU the host possesses, RAM($\chi_i$) is the amount of RAM in GBs on the host, Storage($\chi_i$) is the amount of disk space the host possesses in MBs, Id($\chi_i$) is the unique ID assigned to the host, Tasks($\chi_i$) represents the queue of active DAG tasks on the computation host, Reservations($\chi_i$) represents the queue of DAG task reservations on the computation host, Type($\chi_i$) represents the type of the computation host where Type($\chi_i$) $\in \{\text{Cloud, Edge, Mobile}\}$. Links between hosts are defined as $E_N = \{e_{ij} | \chi_i, \chi_j \in X_N\}$ where $e_{ij} = e_{ji}$. The link characteristics are Latency($e_{ij}$) $\in \mathbb{R}^0$ and Bandwidth($e_{ij}$) $\in \mathbb{R}^0$.

**IV. ALGORITHMS**

The offloading algorithms are executed from the mobile device, allocating computational resources in the mobile device, edge servers, or cloud for DAG tasks and communication resources on the network links. The common goal is to minimise the processing time of each interleaved DAG. All algorithms assume that all allocations will complete at the time estimated before allocation. In real-world scenarios, a scheduled inference plan can fail due to unforeseen circumstances (i.e. degradation of network links, a mobile device running out of battery). For these experiments, we assume that there will be no unsuccessful terminations that could disrupt the schedules created. Additionally, it is assumed that the binaries for each task within every DAG to be offloaded are pre-installed across all the computation nodes within the network.

**A. Reactive Algorithms**

The **basic reactive algorithm** offloads tasks within composite DAGs, task by task as they become ready. It is based upon the work of De Maio et al. [5] selecting the computational host that provides the lowest round-trip time for a task to be offloaded. The algorithm works on tasks from all the interleaved DAGs active at one time. It processes DAG tasks separately and sequentially in the order of their arrival, without considering the DAG that they belong to. Upon completing execution of a task in a remote host, the output of the task must be returned to the mobile device. The **extended reactive algorithm** builds upon the basic algorithm, providing priority for executing non-offloadable tasks. The mobile device is the only computational host capable of processing non-offloadable DAG tasks and is greatly limited in its capacity. This modified version of the reactive algorithm sorts the list of DAG tasks to be offloaded so that non-offloadable DAG tasks are allocated first. This approach seeks to limit the amount of time DAGs are waiting on non-offloadable tasks.

**B. Hybrid Algorithm**

An issue with the reactive algorithms is that task output data must always return to the mobile device, even when the next DAG task is executed on the same host. The **hybrid algorithm** pre-allocates the computational host for child tasks when all their parents are offloaded. The input to the algorithm is the list of all tasks ready to offload within the network, with the list sorted for mobile priority. The pre-allocation function does not affect the input pool of tasks. Much like the extended reactive algorithm, to select an allocation candidate, it iterates through the sorted list of ready tasks, where it then attempts to retrieve an existing reservation or allocate the task in the event no reservation is found. It also instructs the computational hosts running the parent tasks to directly send their output data to the host allocated for the child tasks. This is called **input chaining** and it reduces communication delay in multiple ways:

- output data does not need to be routed through the mobile device before being sent to the child task
- if a task has multiple parents, output data from each parent can be sent in parallel if the parents are executed on separate hosts
- a task can remain in the edge or cloud while incurring no communication delay provided their parent was also executed on the same host.

When a child task is offloaded and ready to receive inputs from the parents, the parents must be notified of the host processing the child. Ideally, when the final parent task to be allocated has finished processing, the rest of the parents will have uploaded their results to the child’s computational host, leaving only the last parent task to upload their result.

The task pre-allocation performed by the hybrid algorithm reduces the distance between parent and child tasks in both time and space compared to the greedy allocation of the reactive algorithms, resulting in improved performance. Another effect of the parent-child pre-allocation is that DAGs whose initial layers have already been processed will have priority to be completed compared to the reactive algorithms that process DAG tasks indiscriminate of the DAGs they belong to. This effect is called “DAG preference”.

**C. Proactive Algorithm**

Unlike the one task ahead approach of the hybrid algorithm, the proactive algorithm **pre-allocates the entire DAG** when the first task is received, reserving resources for all the tasks of the DAG. Input chaining can be set up from the start for all the tasks, reducing communication delay. Proactive resource reservation also means that DAGs that start are processed
without interruption, leading to stronger DAG preference. In fact, with the proactive algorithm, it can be said that DAGs are processed sequentially by the offloading algorithm instead of in an interleaved manner (DAG tasks will still be processed in parallel by the different computational hosts, if resources are available). The proactive algorithm is greedy. It allocates the tasks from a DAG in order of dependency following the parent-child relations, and always allocating a task to the host that leads to fastest processing.

D. DNN Partitioning algorithm

DNN partitioning algorithms [7], [8], as explained above, split the DNN DAG in two, the first partition executed on the mobile device, the second offloaded. Existing algorithms do not consider non-offloadable layers or only consider them at the start of the DNN [9]. This paper uses a partitioning scheme where the DNN is split in half and if non-offloadable layers are encountered in the offloaded half, the processing is brought back to the mobile device. It is clear that the DNN partitioning algorithm will perform sub-optimally in the event of non-offloadable layers, however the results show that there are other more serious effects that penalise the performance. Future research should investigate partitioning algorithms that account for non-offloadable layers randomly located in the DNN.

E. Scheduling access to resources

All algorithms access resources in time-slotted fashion. The computational and communication resources are time-shared between the DAG tasks by allocating or reserving variable-length time slots. For computation, the length of the slot is equal to the processing time of the DAG task on the specific host. Several DAG tasks can be executed in parallel as long as the capacity of the host (GPU(\chi)) is not exceeded. Regarding communication, the input/output of data for a task occupies the entire bandwidth for a certain upload window. The algorithms will use earliest completion time when deciding the allocation of a task to a host. Resource scarcity may lead to the schedules becoming fragmented, in which case the algorithms will have to fast forward (look-ahead) through the schedules, equivalent to a DAG task having to wait until resources are released.

F. Complexity Analysis

Table I provides a breakdown of the characteristics of each algorithm, including their time complexities. The proactive and partitioning algorithms incur the highest time complexity. This is to be expected as they must fully allocate each node across every DAG assigned to the algorithms. The hybrid algorithms’ complexity is slightly higher than the reactive algorithms due to its fast-forward function, which ensures resource reservation for each child task. The proactive algorithm has the most straightforward allocation mechanism. From the current time slot, calculate the upload time to each node in the network and select the node that has both the capacity to process a given task and provides the lowest RTT. Despite this simple allocation mechanism, its complexity is similar to the hybrid algorithm. This is because of the fragmented scheduling approach that it employs. When the network is saturated, an incoming task cannot receive placement in the current time slot. It must return to the ready queue until resources are freed within the network. The reactive algorithm does not give preferential treatment to which tasks it allocates first if a given task cannot receive placement alongside several other tasks. It may continuously fail until all current tasks have been allocated and have exited the network.

V. Evaluation

A. Network Topology

The properties of the three types of computational hosts presented in Section III are listed in Table II. The mobile device has the characteristics of current generation smartphones with a Qualcomm Adreno 640 mobile GPU. For the edge devices many solutions consider Single Board Computers (e.g. Raspberry Pi), however they have typically highly constrained hardware and no GPUs, resulting in poor performance. Therefore, it was decided to use dedicated edge servers, such as the Nvidia RTX Server with Nvidia RTX 3060 GPUs. Finally, we envisage the cloud having essentially unlimited resource capacity for our network but the same per device processing power as the edge. As mentioned in Section III the computational hosts are fully-connected into a network, with link latency and bandwidth configured as per Table III. The values were obtained via measurements as follows: mobile to edge in the same wireless access network (e.g. WiFi); mobile to cloud via cellular (4G) connection; edge to cloud via Internet.

B. DAG Topologies

The evaluation considered both generic DAG structured applications and DNNs following the structure described in Section III-A.

We have selected Inception v4 as it is one of the most popular image classifiers presently available to employ a DAG topology. It has been utilised in prior partitioning literature as shown by Hu et al [11]. Additionally, Inception v4 contains 195 partitioning points per our partitioning templates and therefore provides a vast space for partitioning the network to assess the performance of our chosen algorithms.

When generating both types of DAGs, individual tasks have a 10% or 30% chance (depending on the experiments being run) of being non-offloadable.

The two values were chosen empirically, with 10% representing a low impact on performance, while 30% generated bottlenecks in resource availability that impact performance. Less than 10% would offer no significant improvement in performance, while selecting greater than 30% will drastically reduce the performance. Values greater than 50% negate the gains of offloading because more than half of the DNN would be processed locally.

Run time and input/output size for each DAG task are discussed below, with average parameters shown in the first
estimated processing time for each layer for the GPUs/CPUs. We used Qi et al.’s performance analyser Paleo [17] to obtain the performance of the five algorithms for DNN offloading. Processing unit (GPU/CPU) while they are being run.

### Table IV. All DAG tasks will consume one
processing unit (GPU/CPU) while they are being run.

#### 1) DNN Applications: We use inception v4 [10] to evaluate the performance of the five algorithms for DNN offloading. We used Qi et al.’s performance analyser Paleo [17] to obtain estimated processing time for each layer for the GPUs/CPUs.

#### Table I: Characteristics of the five algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Input</th>
<th>Output</th>
<th>Criteria for Selection</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Reactive</td>
<td>Set of ready tasks (m) across all DAGs, set of tasks allocated per computational host (I_N), set of hosts within the network (h), set of communication windows along the network link (u). (Where u is the average amount of tasks in the network)</td>
<td>A set of task-to-device mappings corresponding to a subsection of m.</td>
<td>Node that provides the lowest RTT latency to and from the mobile host.</td>
<td>m&lt;sup&gt;2&lt;/sup&gt;</td>
</tr>
<tr>
<td>Reactive with Mobile Priority</td>
<td>Set of ready tasks (m) across all DAGs, set of tasks allocated per computational host (I_N), set of hosts within the network (h), set of communication windows along the network link (u). (Where u is the average amount of tasks in the network)</td>
<td>A set of task-to-device mappings corresponding to a subsection of m.</td>
<td>The node that provides the lowest RTT latency relative to a given task's set of parents.</td>
<td>m&lt;sup&gt;2&lt;/sup&gt;</td>
</tr>
<tr>
<td>Hybrid</td>
<td>Set of ready tasks (m) across all DAGs, set of tasks reserved and allocated per computational host (I_N), set of pending reservations (p) (reservations whose parents have yet to be fully allocated) within the network, set of hosts within the network (h), a set of communication windows along the network link (u). A matrix of DAGs (d), each index corresponding to a subsection of m.</td>
<td>A set of task-to-device allocations corresponding to a subsection of m.</td>
<td>The node that provides the lowest RTT latency relative to a given task's set of parents.</td>
<td>pm&lt;sup&gt;2&lt;/sup&gt;</td>
</tr>
<tr>
<td>Proactive</td>
<td>Set of first node tasks (m) across incoming DAGs within the network, set of tasks reserved and allocated per computational host (I_N), set of pending reservations (p) (reservations whose parents have yet to be fully allocated) within the network, set of hosts within the network (h), a set of communication windows along the network link (u). A matrix of DAGs (d), each index containing the remaining DAG for each first DAG node found within (m).</td>
<td>A set of task-to-device allocations corresponding to a subsection of m.</td>
<td>The node that provides the lowest RTT latency relative to a given task's set of parents.</td>
<td>m&lt;sup&gt;3&lt;/sup&gt;</td>
</tr>
<tr>
<td>Partition</td>
<td>Set of first node tasks (m) across incoming DAGs within the network, set of tasks reserved and allocated per computational host (I_N), set of pending reservations (p) (reservations whose parents have yet to be fully allocated) within the network, set of hosts within the network (h), a set of communication windows along the network link (u). A matrix of DAGs (d), each index containing the remaining DAG for each first DAG node found within (m).</td>
<td>A set of task-to-device allocations corresponding to a subsection of m.</td>
<td>The first half of a DAG is limited to selecting the mobile host, and the second half of the DAG is limited to choosing the cloud.</td>
<td>m&lt;sup&gt;3&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

### Table II: Computational host properties.

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
<th>GPUs</th>
<th>RAM (GB)</th>
<th>Storage (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>128</td>
</tr>
<tr>
<td>Cloud</td>
<td>Unlimited</td>
<td>Unlimited</td>
<td>Unlimited</td>
<td>Unlimited</td>
</tr>
</tbody>
</table>

### Table III: Network link properties.

<table>
<thead>
<tr>
<th>Node</th>
<th>Latency (ms)</th>
<th>Bandwidth (MBps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile Links</td>
<td>10</td>
<td>150</td>
</tr>
<tr>
<td>Cloud</td>
<td>70</td>
<td>36</td>
</tr>
<tr>
<td>Edge</td>
<td>20</td>
<td>1024</td>
</tr>
<tr>
<td>Mobile</td>
<td>10</td>
<td>150</td>
</tr>
<tr>
<td>Cloud</td>
<td>20</td>
<td>1024</td>
</tr>
<tr>
<td><em>Local Links</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mobile</td>
<td>70</td>
<td>36</td>
</tr>
<tr>
<td>Edge</td>
<td>20</td>
<td>1024</td>
</tr>
</tbody>
</table>
considered so that we could convert inception v4 into a format that matches III-A. This information was used as a template for generating DNN applications, randomising the layers that can and cannot be offloaded.

2) Generic Edge Applications: The generic applications were based on the ones defined in De Maio et al.’s work [5]. For each DAG task a computational cost in MIPS (Millions Instructions Per Second) was sampled uniformly between 1 - 10 and then converted into processing time based on the specifications of the computational hosts used in the network of De Maio et al. [5].

C. Evaluation Methods & Simulation

The experiments gathered the following metrics: overall DAG completion rate, DAG task completion rate, the ratio between communication and computation time, and computational host resource usage. DAG completion rate is the most important metric as it is the objective each algorithm attempts to optimise. DAG task completion rate is a finer performance metric; it is not always related to the overall completion rate but it can identify sub-optimality in the algorithms. The ratio between communication and computation times is an indication of how efficiently algorithms are using resources. The goal is to spend most time computing, and communication is thus considered an overhead; therefore the ratio should be small. Host usage represents the distribution of DAG tasks among the computational hosts. Combined with other metrics such as DAG completion rate, it is a good indicator of algorithms’ efficiency.

The evaluation focused on comparing the performance of the algorithms in terms of DAG completion rate under a set of conditions that were meant to replicate the scenarios of interleaved requests and non-offloadable DAG tasks. It also investigated the algorithm behaviour when resources become saturated and the impact of bottlenecks on the performance of the algorithms.

An event-driven simulator was developed to evaluate the aforementioned algorithms. A fixed amount of time was given, where a number of applications (ranging from one to twenty) appearing over time were set to be processed by the different algorithms. Twenty input instances were generated for each application set size when conducting the evaluations. Each application in each instance was randomised based on the information laid out in Sections V-B1 and V-B2 with the application arrival times sampled from an exponential distribution. Finally, each algorithm was run against each input instance and performance metrics were collected. Any nodes and DAGs that had not finished were terminated. The evaluation time was set to ten seconds for the generic edge offloading simulation and four seconds for the DNNs. The latter time was set lower because DNNs DAGs have an overall processing time that is much shorter than generic DAGs.

The simulations, prerequisite tools and data set utilised for these experiments are available online at GitHub through the following links:

- Simulator: https://github.com/JamieC1998/MEC_Resources
- Input Generator: https://github.com/JamieC1998/Offloading_Sim_App Gen
- Data Set: https://github.com/JamieC1998/MEC_Prellocation _Sim_Data_Repository

VI. RESULTS

We first discuss the characteristics of the two DAG types considered, and their impact on the algorithms. Then, we provide a high level analysis of the performance of the algorithms, before going into details.

A. Characteristics of evaluated DAG types

The two DAG types considered for evaluation are generic DAGs and DNNs and their topologies are discussed in Section V-B. The last four columns of Table IV present a numerical analysis of expected processing time of the two DAG types on the three types of computational hosts, ignoring resource contention. Specifically, the table presents the ratio between the processing time of an average application if run locally (on the mobile device) versus the overall time (computation and communication) if offloaded to either the edge or cloud, with and without input chaining. The values show that generic DAGs have significantly higher local than offloaded computation time, whereas DNNs show the opposite with a very small ratio. As discussed before, this stems from the significantly different structure of the two, with generic DAGs composed of a small number (< 35) of longer running tasks and DNNs of large number (191 for inception v4) of short running tasks. A similar analysis was conducted post-evaluation on the results of the simulation and accounting for the impact of resource contention. This is shown in Figure 3.

The values in Table IV also indicate that the preferred algorithm decision for generic DAGs will be offloading (since

| TABLE IV: Average processing and offloading time (from mobile to edge and mobile to cloud) of DAG tasks generated. Local / (x) is the time to process on the mobile host divided by the round trip time when offloading to a particular host with/without input chaining. |
|---|---|---|---|---|---|---|
| **Computation Time Mobile (s)** | **Computation Time Cloud & Edge (s)** | **Input (MB)** | **Output (MB)** | **Local/Edge (No chaining) (s)** | **Local/Edge (Chaining) (s)** | **Local/Cloud (No Chaining) (s)** | **Local/Cloud (Chaining) (s)** |
| Generic Applications | 2.2802 | 0.0007 | 25.277 | 24.8970 | 1.76.2062% | 2.395684% | 66.7223% | 113.1113% |
| DNNs (Inception V4) | 0.0004 | 0.0002 | 1.5123 | 1.0776 | 0.7109% | 1.3486% | 0.1926% | 0.3640% |

The values in Table IV also indicate that the preferred algorithm decision for generic DAGs will be offloading (since...
local execution is much slower), while for DNNs it will be local execution. The latter will quickly lead to saturation of the mobile device’s resources, turning it into a bottleneck. Another observation is that the edge will be the preferred resource for offloading for both DNNs and DAGs, with the cloud being used only when the edge is saturated. This will be discussed later.

B. Performance analysis

The DAG completion rate is presented in figures 4 and 5, measuring the ratio of DAGs that are processed to completion within the evaluation time. The results show that the proactive and hybrid algorithms achieve the highest DAG completion for both generic and DNN DAGs. As seen in figure 4, when the number of non-offloadable tasks in the network is set to 10% the difference between the two algorithms is at maximum 10% completion rate, however as seen in figure 5 when the number of non-offloadable tasks is set to 30% the performance of the hybrid algorithm is severely affected, and the proactive dominates the results. The two reactive algorithms perform very similarly, worse than the proactive and hybrid, but better than the DNN partitioning algorithm for generic DAGs, achieving 20% completion rate at highest load. Their performance decreases to <5% when processing DNNs: as indicated in Table IV the preferred decision for DNNs is local execution, so the reactive algorithms will attempt to allocate most tasks on the mobile device, ultimately resulting in DAGs with non-offloadable nodes having to wait until the mobile device is free.

The DNN partitioning algorithm performs poorly in every category, as it suffers from extreme bottlenecking on the mobile device, made worse by non-offloadable tasks and interleaved DAGs. The first half of every DAG (in inception v4 this amounts to 95 layers) must execute on the mobile, which is problematic as the mobile can only process one task at a time. When the number of applications (requests) increases, as the DNN partitioning is fully proactive, subsequent DAGs must wait for the first half of a predecessor DAG to finish processing on the mobile device before they can be allocated and begin their computation. Additionally, non-offloadable tasks in the latter half of a DAG bring computation back to the mobile device, incurring significant communication delays and having to wait for the mobile device to become available. All in all, the DNN partitioning algorithm does not use the available computational resources effectively, resulting in reduced parallelism and poor performance.

Figures 6 and 7 present the ratio of tasks completed within the evaluation time, which is more detailed than DAG completion. Across all algorithms in both scenarios, the DAG task completion rate is higher than the DAG completion rate. This is expected as the total amount of tasks completed by an
algorithm include the whole set of tasks for each completed DAG, but also a potential subset of tasks for each incomplete DAG. Unless an instance is based on a high volume of DAGs with few tasks and a small number of DAGs with a large number of tasks, it is unlikely that DAG completion rate will exceed task completion rate. The reactive algorithms show the highest difference between their DAG completion and DAG task completion, especially for DNNs. This is due to processing DAG tasks irrespectively of their DAG, which means that DAGs that have already commenced execution do not have a guarantee or preference for being completed. The hybrid algorithm achieves the highest DAG task completion rate in both scenarios when the percentage of non-offloadable tasks is configured to 10%, and a slightly worse rate when this is set to 30%. Finally, the proactive algorithm maintains a modest task completion rate, however, as it already has the highest DAG completion rate, this is not an issue.

Figures 6 and 7 also show the distribution of DAG tasks throughout the three types of computational hosts for the five algorithms. As discussed above it can be seen that most computation is performed in the edge with the cloud only used at higher loads. Due to their internal structure and the ratio between communication and computation time (Figure 3), DNNs experience a higher percentage of local processing (20-40%) when compared to generic DAGs (5-20%), as per the analysis in Table IV. Furthermore, with DNNs...
and higher percentage of non-offloadable layers (Figure 7b) most processing is performed locally. It can also be seen that the proactive and hybrid algorithms process slightly fewer tasks locally than the reactive algorithms (with the exception of Figure 7b), and more processing in both edge and cloud. As mentioned above, the partitioning algorithm has the poorest utilisation of the infrastructure resources as it does not consider the edge hosts. For DNN partitioning, figures 6b and 7b show that roughly 50% of tasks are always executed on the mobile device, corresponding to the first half of the DNN-type DAGs; however, there is very low additional utilisation of the cloud, where the second half of the DNN would be offloaded. This indicates that the DNN partitioning rarely gets to processing tasks in the cloud, likely caused by the saturation and bottlenecking of the mobile device.

**VII. DISCUSSIONS**

A. Algorithm characteristics: impact of input chaining and DAG preference

The performance of the algorithms is mostly impacted by the following characteristics: performing input chaining and giving preference to already running DAGs. These are discussed below, and they provide a clearer explanation of the results.

**Input chaining.** Input chaining allows algorithms to continue processing DAG tasks remotely even when their currently active tasks are distributed between a set of hosts. This, in turn, reduces the load of the mobile device, allowing it to process new tasks and avoiding bottlenecks. Input chaining achieves better resource utilisation and higher parallelism, resulting in higher DAG completion rates in times of network saturation, and reduced impact of high communication delay in scenarios where offloading is unfavourable.

Input chaining reduces the total amount of communication. Figure 3 breaks down the total processing time into communication (blue) and computation (orange). The algorithms that use input chaining (hybrid and proactive) have lower communication overhead than those that do not (reactive). This is reinforced by their host resource usage as seen in figures 6b, 7b where the proactive and hybrid algorithms complete more tasks in the edge and cloud than the reactive algorithms. The DNN partitioning algorithm is an interesting case as it processes large partitions of the DAGs on the same node, therefore completely eliminating communication overhead within that partition, which is the goal of DNN partitioning algorithms. However, non-offloadable layers decrease those gains. With generic DAGs (Figure 3-top) the DNN partitioning algorithm achieves slightly higher communication overhead than the proactive and hybrid algorithms. The results in Figure 3-bottom show that the partitioning algorithm achieves the lowest communication overhead for DNN DAGs, which in fact occurs because very few tasks actually get offloaded, as can also be seen in the low utilisation of the cloud host in Figure 7b.

The lack of input chaining means that the result of each offloaded task must come back to the mobile device, which introduces large communication overheads as shown by the reactive algorithms in Figure 3. This results in fewer tasks being completed in the edge or cloud, especially for DNN DAGs with large number of tasks, as shown in Figure 6b where the mobile is the most used computational host. On the other hand, the lack of input chaining is the reason proactive algorithms are least affected by non-offloadable layers, as can be seen by comparing Figures 6 and 7.

**DAG preference.** Algorithms process DAGs on a task-by-task basis, however they can assign higher preference to tasks from DAGs that have already been started, which leads to increased DAG completion rate. There are different ways to implement DAG preference. The proactive algorithm pre-allocates all tasks from a DAG at once. DNN partitioning does the same, although in two large blocks between the mobile device and cloud. The hybrid algorithm pre-allocates the children of a task. Finally, reactive algorithms do not implement DAG preference, instead processing any task, from any DAG, that is ready for execution. The impact of the different DAG preference mechanisms under high load can be seen in the results of Figures 4 and 5. Without DAG preference the reactive algorithms achieve very low DAG completion rate because input tasks from new DAGs have priority over middle tasks from already started DAGs. The one-task-ahead pre-allocation of the hybrid algorithm is sufficient to provide DAG preference in most cases, leading to significantly improved completion rate.

Figure 4 shows that when processing DNNs with 10% non-offloadable tasks, the hybrid algorithm can achieve a higher DAG completion rate in medium loads. This is because reactive approach of the hybrid algorithm can reach a higher task completion rate when the network is not saturated, as seen in Figure 6b, which in turn will assist in completing DAGs. Whereas the proactive approach is greedy, as it will attempt to allocate the best resources to the current DAG, this often forces subsequent DAGs to wait for a significant portion of the DAG to complete before they can enter the network, thus providing a consistent DAG completion rate with a somewhat lower task completion rate at medium loads.

B. Scenarios: impact of request interleaving and non-offloadable tasks

When processing interleaved DAGs from a resource-constrained mobile device, all algorithms will eventually face a bottleneck on the mobile device. The obvious solution is to offload as early as possible to minimise waiting times and exploit the parallelism of the computational network before the mobile node is saturated. This requires fine-grained task processing of the DAGs, as performed by the GOAs. In comparison, the DNN partitioning algorithm that processes large blocks of the DAG at once suffers from longer waiting times, resulting in overall lower completion rates. DAG preference is important especially if the DAGs are processed at task granularity. As the load increases and new DAG tasks are ready for processing, algorithms should seek to complete DAGs that have already started. The results have shown that even the DAG preference
approach of the hybrid algorithm is not sufficient, experiencing at high loads a significant difference between DAG (Figure 4) and task completion (Figure 6b) rates.

Non-offloadable tasks are a significant disruption to the efficiency of computational offloading, affecting performance by creating bottlenecks at the mobile device and by interrupting input chains. When faced with non-offloadable tasks, the proactive, hybrid, and partitioning algorithms will be forced to interrupt remote chains of tasks to bring computation back to the mobile device, which incurs communication overhead. The reactive algorithms are not heavily affected by increased non-offloadable tasks since the output of every DAG task must anyway be returned to the mobile device therefore non-offloadable tasks do not incur additional communication overheads.

As seen in Figure 5, increased rates of non-offloadable tasks when offloading DNNs causes the performance of both the proactive and hybrid algorithms to drop. Although they both still outperform every other algorithm, the hybrid algorithm experiences a sharp drop to sub 5% in its DAG completion rate, whereas the proactive only falls to 40%. The performance loss of the former is the result of over-allocation of DNN DAG tasks (DNN layers) to the mobile device, leading to saturation, because local execution is the preferred decision for DNNs (Section VI-A). The DAG preference approach of the hybrid algorithm is unable to give priority and allocate mobile device resources to the non-offloadable layers, which have to wait. On the other hand, the proactive approach can still fully allocate a DAG even when the mobile host is over-allocated, allowing it to outperform every other algorithm under high network saturation.

C. Summary

The above comparative analysis has proven that the real-world DNN offloading conditions of interleaved execution and non-offloadable layers (e.g. due to privacy concerns) negate the advantages of current DNN partitioning algorithms. The presence of non-offloadable layers cannot be supported by the current algorithms based on DNN graph bisection [8] as it requires more than two partitions. Furthermore the algorithms should distribute partitions from interleaved DNNs to more than one remote host (e.g. several edge servers, if available), should make use of input chaining, and reduce the processing burden on the mobile device by allocating very small partitions to it.

Out of the algorithms compared the proactive GOA achieved best all-around performance. At high loads of interleaved requests, with generic and DNN DAGs, and with high number of non-offloadable layers, the proactive approach mitigates excessive communication delays while prioritising existing DAGs when additional DAGs begin to backlog within the network.

VIII. CONCLUSIONS AND FUTURE WORK

Deep Neural Network inference is nowadays one of the most computationally intensive tasks performed on consumer devices, which makes them the ideal target for computational offloading. This paper started with the argument that the current DNN offloading solutions, i.e., DNN partitioning algorithms and are not suitable in several real-world conditions: interleaved requests that can be offloaded to several remote hosts, and non-offloadable DNN layers. This was confirmed by the results that show DNN partitioning inefficiently using all the resources of the mobile device at once and blocking subsequent requests. The paper presented as an alternative Generic Offloading Algorithms that were previously used for computational offloading of applications composed of DAGs of tasks, and which provide increased flexibility and granularity in the processing of an offloading request. This was the first application of GOAs to the offloading of DNN requests. Flexibility and granularity were found to impose a high communication overhead due to the significant difference in size between DNN requests (tens to hundreds of layers) and DAG-like applications (~20 tasks). However, the GOAs were found to perform considerably better than DNN partitioning in the two real-world conditions of interleaved requests and non-offloadable layers. Specifically, a granular proactive GOA approach provides the best performance for completing DAGs even under highly saturated network scenarios with unfavourable communication and computation ratios. A hybrid GOA approach can achieve relatively close performance, utilising input chaining to reduce communication delays and parent-child allocation to help prioritise DAGs.

The experiments did not consider scenarios where the processing may be disrupted, as would be the case when a device is mobile and loses connection to a remote host. It is clear that the flexibility and granularity of GOAs would surpass the DNN partitioning in this scenario as well. However it must be determined if proactive algorithms are an ideal choice in case of disruptions or if a hybrid approach can avoid the impact of having to fully reallocate the remainder of a DAG while still providing reliable DAG completion rates.

REFERENCES


Jamie Cotter is a student in Munster Technological University working towards a PhD as part of the Advance CRT research cohort under Dr Victor Cionca, Dr Ignacio Castineiras and Dr Donna O’Shea. His research primarily focuses on distributed computing applied to Deep Neural Networks. His initial work focused on examining existing DNN partition techniques and comparing their efficiency to techniques from the area of general offloading algorithms (GOAs) to see how GOAs may help improve DAG completion rate when offloading DAG-based DNNs. Presently he is implementing a system to examine how a deadline pre-emption mechanism can help complete higher-priority DNN tasks when processing two tiers of DNNs in a static resource-constrained network.
Jamie Cotter: Conceptualization, Methodology, Software, Investigation, Writing - Original Draft, Visualization, Formal analysis
Victor Cionca: Conceptualization, Methodology, Software, Formal analysis, Writing - Review & Editing, Supervision, Project administration
Ignacio Castineiras: Software, Formal analysis, Writing - Review & Editing
Donna O'Shea: Writing - Review & Editing
Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: