CLEAR: Scheduling of Multi-model Mobile Workloads on Chiplet Edge Platforms

C. Singhal[1], M. Mendula[2], F. Malandrino[3,4], M. Levorato[5], C. F. Chiasserini[4,6,7]
1: INRIA, Rennes, France – 2: CTTC, Barcelona, Spain – 3: CNR-IEIIT, Italy – 4: CNIT, Italy
5: UC Irvine, USA – 6: Politecnico di Torino, Italy – 7: Chalmers University of Technology, Sweden

***Abstract*—To support multiple AI-based applications, mobile systems need to collaboratively execute DNN architectures on heterogeneous AI accelerators. At the same time, the increasing DNN complexity and high degree of diversity in workloads on multichip module (MCM) accelerators are pushing AI processing off mobile nodes onto the edge. This has made computationally intensive, edge-based solutions the dominant approach for the deployment of modern neural networks. However, the rigid structure of fully-executed DNNs fails to align with the modular nature of MCM architectures, limiting their potential for efficient execution. In this paper, we introduce CLEAR, a novel optimization framework based on geometric programming that leverages both transformer-based and more canonical DNNs with early exits. CLEAR enables fast, coordinated decision-making across DNN design, workload distribution, and resource allocation, with the overarching goal of minimizing inference energy consumption. To our knowledge, this is the first work to integrate dynamic DNN optimization with decisions at both the communication infrastructure and hardware accelerator levels. We evaluate CLEAR using real-world wireless measurements and dynamic DNNs applied to computer vision inference tasks. Our results demonstrate that CLEAR achieves near-optimal performance and reduces energy consumption and resource usage by over 80% and 70%, respectively, compared to its benchmark.**

***Index Terms*—Scheduling multi-model workload, Energy efficiency, Dynamic DNN, Multichip module**

I. Introduction

The proliferation of heterogeneous and compute-intense machine learning (ML)-based mobile applications results in large amounts of challenging and diverse workloads. The bulk of the workloads consist of complex deep neural network (DNN) models that must be executed in a timely, and efficient, manner. Also, the various applications need to perform tasks corresponding to different DNN architectures tailored to meet specific performance requirements. As the complexity of DNN models grows, the computational demands frequently exceed the capabilities of constrained mobile nodes, a challenge that the research community has primarily addressed by relying on two strategies: optimizing DNN models and offloading the execution of DNNs to edge servers.

The former approach focuses on crafting lightweight versions of the original DNN model—through techniques such as pruning and quantization [1]—that often come at the cost of reduced accuracy. In contrast, edge offloading shifts the bulk of the workload to edge servers, significantly reducing energy consumption at the mobile nodes, while preserving the inference tasks quality [2]–[4]. However, this strategy introduces new challenges, as offloading both increases bandwidth usage and may suffer from high latency due to channel impairments and network load [5]. To address these challenges, collaborative frameworks such as split computing propose to partition the execution of DNN models between mobile nodes and edge servers. However, also this approach introduces new obstacles. In fact, transmitting high-dimensional latent representations can result in substantial bandwidth consumption [6], while aggressively compressing these features to mitigate transmission costs often degrades inference task performance [7].

The use of recent multi-chip module (MCM) [8] architectures represents a promising approach to the above challenges. Such MCMs can integrate several, possibly heterogeneous, chiplets, each with its own computing and memory capabilities, and connected to the communication backbone of the MCM, a.k.a. Network-on-Package (NoP). Unlike traditional edge computing—where tasks are offloaded to edge servers for monolithic execution without consideration of the DNN's internal structure—MCMs enable flexible distribution of workloads across the set of chiplets within the MCM package. By integrating multiple specialized chiplets for the accelerated execution of AI workloads (e.g., CPU, NPU, Shidiannao and NVDLA accelerators) within a single package, MCMs support functional partitioning of DNNs, allowing different layers or operations to be mapped to the most appropriate hardware [9]. This fine-grained control over resource allocation can unlock more efficient and tailored execution strategies based on the specific characteristics of DNN models.

In this work, we present a framework for the control of edge offloading leveraging MCM architectures. To maximize the yield of such systems, differently from prior approaches, we jointly control the allocation of radio resources and the scheduling of heterogeneous model workloads to the available accelerators: a task that faces three key challenges.

First, radio resources may be scarce and channel conditions may unpredictably vary over time, changing the delivery time of the input data, and, as a result, the computing strategy to

This work was partially funded by the EIC Pathfinder Open scheme of the European Commission (project MUSMET, Grant Agreement No. 101184379), by SNS JU under the EU's HE research and innovation programme under the MultiX project (Grant Agreement No. 101192521), and in part by the U.S. National Science Foundation (NSF) under Grant CCF-2140154.

avoid excessive inference latency. Thus, an effective scheduling policy should be devised over the radio channel to ensure that end-to-end latency constraints are satisfied, especially when multiple mobile nodes access the edge server.

Second, different DNN models—and even their individual layers—exhibit distinct dataflow "preferences" toward optimal performance (e.g., convolutions from an image model, multi-layer perceptrons from a recommendation model, and recurrent layers from a speech model). When multiple DNNs, or blocks thereof, are executed concurrently on the chiplets without an intelligent scheduling mechanism, they can interfere with each other's performance due to bandwidth contention on the NoP [10]. Sophisticated scheduling algorithms are thus necessary to intelligently match the dataflow required by the modern, heterogeneous AI workloads and the dataflow supported by current MCM architectures.

Third, the typically rigid and monolithic structure of DNN models themselves must be re-thought. In fact, to unlock the full potential of MCMs, these models need to be made inherently more flexible and modular, allowing them to be partitioned and adapted to the distributed hardware platform.

To tackle the above challenges, this paper builds on two paradigms for DNN modular design and execution. Specifically, it exploits the possibility offered by dynamic neural networks with *early exits (EEs)* to architecturally transform monolithic models into modularized and adaptable workloads. Importantly, this designs also enables the dynamic decision of terminating inference early, ultimately bypassing deeper layers to reduce energy consumption and latency. Second, it exploits split computing for distributing the DNN (smaller) computational blocks across multiple chiplets.

We leverage these methods to solve the complex end-to-end scheduling problem, *jointly on the radio channel and heterogeneous MCM accelerators*. We start by formulating an optimization problem that minimizes a multi-component energy function—accounting for transmission over the radio link, inter-chiplet computation, and off-chiplet communication—while adhering to strict constraints on timeliness to prevent task dropping and meet quality of service (QoS) requirements. Then, given the provably NP-hardness of the problem, we introduce an innovative algorithmic framework named CLEAR—CLimbing at the Edge for Allocation of Resources. CLEAR is designed to be both feasible and tractable: by decomposing the problem into a Geometric Program and a submodular optimization, CLEAR provides near-optimal joint communication and computing resource allocation strategies with provably low complexity.

Fig.1 depicts our reference scenario (described in detail in Sec. III-A), in which a mobile node offloads a heterogeneous set of AI tasks to an edge server. CLEAR manages radio resources as well as the communication between chiplets within the same MCM and on different MCMs to effectively and efficiently process the input sample, while ensuring QoS in terms of performance and resource utilization.

The main contributions of this work are thus as follows:

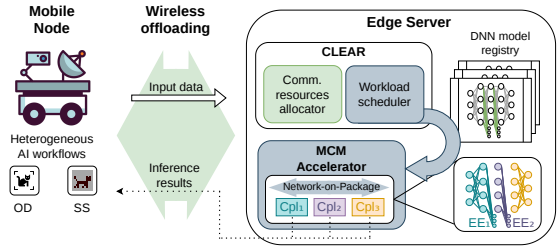


Fig. 1. System-level diagram of our edge AI offloading framework. A mobile node with heterogeneous AI tasks, such as Object Detection (OD) and Semantic Segmentation (SS), transmits input data via wireless offloading to the edge server. At the server, CLEAR allocates resources and schedules incoming workloads. The workload scheduler selects a suitable DNN from a model registry, including those with EEs, and dispatches it to the MCM Accelerator. This hardware features heterogeneous chiplets (Cpl_i) connected by the Network-on-Package bus.

- We address the offloading of heterogeneous AI tasks from mobile nodes to edge servers equipped with advanced MCM accelerators, and provide experimental evidence of the crucial trade-offs that characterize such complex systems.
- We formulate a novel *system-level, joint scheduling problem* that co-optimizes resource allocation across both the wireless network and the edge server hardware. Specifically, *the controller dynamically selects the optimal DNN configuration—including the number of active blocks and* EEs*—and maps these workloads to specific accelerator modules*. The problem is instanced as a time-frequency allocation problem for the radio link and a spatio-temporal partitioning problem for the heterogeneous MCM accelerator.
- As the above problem is proved to be NP-hard, we introduce the CLEAR algorithmic solution, a low-complexity strategy that breaks the original problem into an "inner" problem, belonging to the geometric programming (GP) class (hence, can be efficiently solved optimally), and an "outer" problem that, in spite of retaining binary decision variables, can be tackled with provable optimality guarantees through a hill-climbing algorithm thanks to its submodular nature.
- Results using real-world models—including both traditional convolutional neural networks (CNNs), such as B-LeNet and B-ResNet, and more advanced transformer-based architectures—demonstrate CLEAR's capability to achieve near-optimal energy efficiency while outperforming the SCAR state-of-the-art benchmark [10] by approximately 70% and 80% in resource utilization and energy consumption (resp.)

II. Related Work

Our work is mainly connected to three areas of research: inference task offloading, DNN execution on computing platforms, and dynamic DNNs.

Inference task offloading. Modern AI-based applications demand substantial computational resources, making them difficult to execute on resource-constrained mobile nodes [11]. Offloading inference tasks to edge servers helps overcome this limitation by reducing response time [12]. Furthermore,

computation-aware offloading of DNN inference can significantly reduce the energy consumption of mobile nodes [13]. Recent works have specifically investigated the offloading of DNN-based computer vision tasks. In particular, in [13] a task is partially run at the mobile node before it is offloaded to the edge to reduce bandwidth consumption. This study also characterizes the relation between compression and computation ratio to minimize the energy footprint. Likewise, [14] improves edge inference by compressing different portions of video frames according to their relevance to users' quality of experience. More relevant to our work is the offloading strategy in [15], which aims to maximize the tasks that can be executed at the edge by adopting semantic image compression for reducing bandwidth consumption while accounting for edge resources utilization. Similarly, [16] leverages semantic compression to decrease the load on the radio link and optimizes the DNN usage by pruning and sharing common DNN blocks across tasks.

DNN execution on computing platforms. Large-scale ML tasks can lead to resource blocking, resulting in violations of service-level objectives (SLOs). To mitigate this, model slicing techniques combined with coarse-grained preemption are used for executing ML workloads on CPUs/GPUs [17]. These techniques partition inference tasks into smaller subtasks up to the level of neural network layers, to better balance workloads and reduce the risk of SLO violations. Task scheduling on a CPU/GPU can be done with or without spatial multitasking, and, to avoid deadline miss, real-time GPU scheduling can be made at runtime while optimizing energy consumption [18].

In MCMs, the DNN inference latency is also impacted by inter-chiplet communication. Tiling optimization with cross-layer pipelining is thus used in [8] to balance compute and communication latency and improve resource utilization, although only single-model workload and homogeneous chiplets are considered therein. Further, multi-model workloads and heterogeneous dataflows present a huge scheduling space for an MCM ML accelerator. In more detail, inter-layer pipelining with an enhanced in-MCM data reuse and reduced off-chip traffic can be used to meet SLOs [19]. Data flow mapping on inter-communications at the chiplet and MCM level can reduce energy and delay [20]. Energy-efficient acceleration of transformer inference is also possible by using multi chiplet-based heterogeneous integration [21] and runtime task mapping [22]. In addition, scheduling of multi-DNN workloads on chiplet-based accelerators can provide Pareto-optimal solutions by trading-off between latency, energy, and physical footprint [23]. However, effective scheduling of dynamic multi-model workload on interconnected multiple heterogeneous chiplet AI accelerators has not been studied before.

Relevant to our study is also the sharing of computational resources by different ML tasks, and, in particular, the case where a task has real-time deadlines, e.g., O-RAN functions. In this context, [24] characterizes the performance of different GPU sharing techniques, and proposes a near-optimal control strategy to balance the tight requirements of real-time ML tasks with the need to efficiently use the available resources. Still in the context of O-RAN, [25] splits inference loads between GPUs and CPUs, aiming at reducing energy and latency costs. A data-driven approach to the same problem is used in [26], exploiting a publicly-available dataset [27].

Dynamic DNNs. They are an extension of conventional model architectures that offers adaptation of the structure and parameters according to the system constraints during inference [28]. Dynamic DNNs with EEs have side branches attached to the conventional DNN backbone architecture, offering an early inference at intermediate points [29]. These architectures can speed up the inference process and support the DNN partitioning across computing nodes [29]. Additionally, energy efficient inference task execution is possible through efficient orchestration of these models under system and application-side constraints in mobile-edge-cloud systems [28], [30].

Progress beyond the state of the art. The exiting literature has not studied a joint allocation of compute and communication resources for dynamic AI workloads on heterogeneous MCM accelerators. In this paper, we thus present an energy efficient provisioning of heterogeneous applications by integrated scheduling and resource allocation for dynamic ML workloads in a mobile-edge continuum system.

III. System Scenario and Experimental Analysis

This section introduces our reference scenario and presents an experimental analysis of the trade-offs that serve as the basis for our work.

A. Reference scenario

We consider an edge computing scenario where edge servers host Multi-Chip Modules (MCMs), each of which includes several interconnected chiplets. As depicted in Fig. 1, such MCMs can execute diverse AI (e.g., object detection (OD) and semantic segmentation (SS)) workloads. Due to the task complexity and their limited capabilities, mobile nodes (e.g., autonomous rovers) thus offload their AI tasks to a nearby edge server rather than processing the tasks locally, and transmit the input data they generate on their wireless link with the edge. At the edge side, CLEAR, our novel scheduling framework, coordinates both communication and computing resource allocation. To this end, CLEAR incorporates two key components: the Communication Resource Allocator and the Workload Scheduler.

The Communication Resource Allocator dynamically allocates a portion of the available bandwidth to each mobile node, based on the size of its incoming data. Meanwhile, the Workload Scheduler selects an appropriate DNN—equipped with EEs—from a model registry, and dispatches it to an internal MCM. Within the MCM, each chiplet can be scheduled to run one or more DNN layers, or groups of layers, called *blocks*. An example of such scheduling is illustrated in Fig. 2.

Depending on where the DNN blocks are executed, data can be transferred from one to another *(i) within the same*

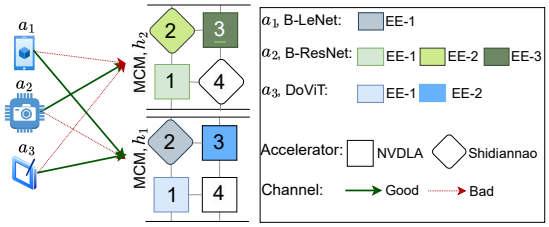


Fig. 2. Example of workload allocation on 2 heterogeneous 2×2 MCMs (h_1, h_2) for applications a_1, a_2, a_3, each originated at a different mobile node and requiring the execution of a DNN model with EEs. The unshaded (white) chiplets are unassigned. a_1, which has less compute requirement, is scheduled on the less compute capacity Shidiannao chiplet. The blocks of a_2 are scheduled on h_2 and those of a_3 on h_1, due to the good state (i.e., low-latency) of the channel between the corresponding mobile nodes and MCMs.

chiplet over the internal **Network-on-Chip (NoC)**, *(ii) between different chiplets* via **inter-chiplet communication** across the **Network-on-Package (NoP)**, or *(iii) to or from an external source* (data transfer between MCMs and between the MCM and external memory through DRAM) through **off-chip communication**. These situations correspond to vastly different delay values; as described in [10, Sec. III-E] and modeled later in Section IV-A, three scenarios are possible. Communication within the same chiplet over the NoC incurs no delay. Inter-chiplet communication over the NoP has two components: a transmission delay (proportional to the ratio of the data transferred to the allocated bandwidth), and a propagation delay, proportional to the number of hops separating the chiplets. Then communication between chiplets in different modules includes a further constant delay, due to off-chiplet memory access time.

B. Main trade-offs

We now look at the trade-offs that characterize the above system, and perform an ablation study to assess to which extent a clever mapping of the dynamic DNN blocks on MCM architectures can fulfill the applications requirements. To this end, we focus on branched DNNs with EEs with and without attention-based mechanisms, namely, B-LeNet [31], B-ResNet [32], DoViT [33], and LgViT [34]. Table I summarizes the key characteristics of these EE-enabled models, together with their associated tasks and training datasets. To extend our ablation study beyond the state-of-the-art for semantic segmentation, we modified the initial DoVit architecture to leverage its original semantic probing mechanism and introduced three novel EE points. The quality requirements for the DNN models–B-LeNet, LgViT, B-ResNet, and DoViT—are set to 70%, 88%, 70%, and 60%, respectively, while the latency requirement is set to 15 ms.

The MCM parameters used in our study are based on [8], [10], [35], and include latency, energy, and bandwidth both of the DRAM (offchip memory) and NoP, as reported in Table II. We have used weight stationary dataflow for the MCMs. We conduct a thorough analysis of dynamic workload execution on MCM architectures using the standard DNN performance

TABLE I
SUMMARY OF MODELS USED IN OUR ABLATION STUDY

Model	Base Architecture	Description	Dataset
B-LeNet	LeNet-5	8-layer branchy CNN with **2 EEs**	EMNIST
B-ResNet	ResNet110	3 stages of residual blocks & **3 EEs**	CIFAR-10
LgViT	Vision Transformer	Uses both convolutional and self-attention heads and features **4 EEs**	CIFAR-100
DoViT*	Transformer	24-layer model with a "semantic early-probe"	Cityscapes

* The original transformer based architecture has been modified to feature **3 EEs** after layers 6, 12 and 18.

TABLE II
DETAILS OF THE MCM PARAMETERS [10], [35]

Component	Parameter	Value
Off-chip memory	DRAM latency	200 ns
	DRAM energy	14.8 pJ/bit
	DRAM bandwidth	64 GB/s
Package	NoP latency	35 ns/hop
	NoP energy	2.04 pJ/bit
MCM	# Proc. elements	1024, 4096
	# chiplets	36, 72
	NoP bandwidth	64 GB/s, 128 GB/s
	L1, L2 memory	(4, 8), (10, 30) MB

simulator, MAESTRO [36], known to provide 96% accuracy compared to RTL simulations.

We explore the trade-offs between quality—measured by mean Average Precision (mAP) or accuracy depending on the AI task to be executed—and latency, as well as between energy consumption and latency, when deploying DNN models with EEs on MCM accelerators. We also evaluate various model partitioning strategies, including no split, partial split (denoted as part-split in the plots), and full split, for both EE-enabled and monolithic architectures across the selected DNN models. *The presence of EEs introduces the possibility of early termination, allowing inference to be terminated before the entire model is executed*, depending on the intermediate prediction confidence.

Fig. 3 shows the performance of the DNN configurations on the MCM under the different settings. We note that, due to its low complexity, the B-LeNet model always meets the quality and latency requirements under all considered split settings (Figures 3a–b). In contrast, for the more complex B-ResNet classification model, the 70% mAP and 25 ms requirements can be met only for the configurations with split deployment settings (Figures 3c–d. Similarly, for LgViT and DoViT, EE configurations with model splitting are predominantly the ones that meet the requirements, due to the increased complexity of these transformer-based models (Figures 3e–

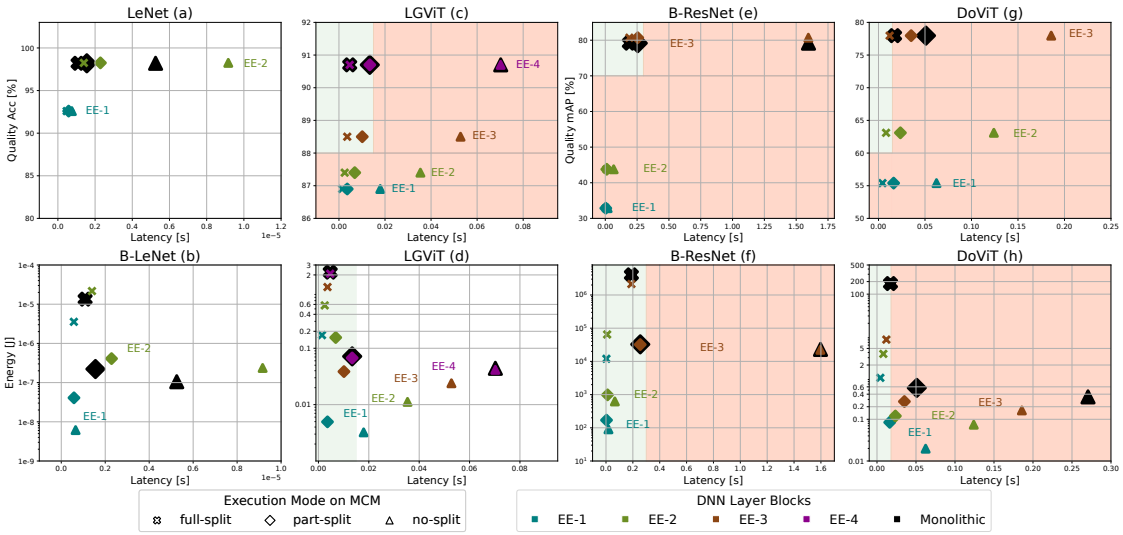


Fig. 3. Ablation study of four DNN models (B-LeNet, B-ResNet, LGVIT, and DoViT) on a MCM, vs. latency. Top row (a): inference quality; Bottom row (b): energy consumption. Different markers denote the model splitting strategy (full-split, part-split, no-split); colors denote specific EEs or a monolithic deployment. For the simpler LeNet, all configurations meet performance requirements. For the more complex B-ResNet, LGVIT, and DoViT, splitting and EEs are necessary to reduce latency and energy, underscoring the crucial trade-off between DNN design, scheduling strategy, quality, and resource usage.

h). *This clearly highlights the need for dynamic decisions on the model configuration, deployment decisions, and EE activation. Also, since the radio transmission incurs additional latency and energy consumption in a distributed mobile system, we need to jointly optimize the wireless resource allocation in the system.*

IV. System Model and Problem Formulation

Below, we start by describing the main components of the system model already introduced in Sec. III—to wit: mobile nodes; applications; edge MCM accelerators and chiplets therein, and the wireless channel – along with their features (Sec. IV-A). We then list the decisions we have to make—block selection, allocation, load and bandwidth sharing—and the associated decision variables, and formulate our optimization problem (Sec. IV-B). Finally, we show how the values of the decision variables impact the end-to-end latency and energy consumption metrics (Sec. IV-C), and prove that the formulated problem is NP-hard (Sec. IV-D).

A. Model components and parameters

Each mobile node $n \in \mathcal{N}$ implements an application $a_n \in \mathcal{A}$, and generates input samples to be processed by it, according to a Poisson process with rate λ_n; the size of each sample is $s_0(n)$ bits. The model can be easily extended to a mobile node using multiple applications simultaneously by considering distinct elements in $\mathcal{N}$. Each application requires a minimum inference quality Q_n and a maximum end-to-end latency of D_n. An application is composed of one or more DNN blocks, with each block $l \in \mathcal{L}$ corresponding to a portion of the DNN backbone and at most one EE. We indicate as $q(a, \mathcal{L}')$ the inference quality obtained when the set of blocks $\mathcal{L}' \subseteq \mathcal{L}$ is used for application $a \in \mathcal{A}$.

Blocks are associated with a memory requirement of $m^{\text{req}}(l)$. In general, the input and output amount of data are different. Specifically, parameter $s(l, n)$ denotes the size of the data originating from node n, and fed into application a_n, after it has been processed by block l. Similarly, $\kappa(l, n)$ denotes the number of units of computational capabilities needed by block l to process one unit of traffic for application a_n. Finally, we indicate with the binary parameters $f(l_1, l_2) \in \{0, 1\}$ whether l_1 follows l_2 in the DNN, i.e., whether data must flow from l_1 to l_2 during inference. Notice how not all sets of blocks can be used to provide all applications; specifically, if set $\mathcal{L}'$ is not used for application a', we have $q(a', \mathcal{L}') = -\infty$.

Blocks are implemented on edge-based MCM accelerators, $h \in \mathcal{H}$, each having a set $\mathcal{C}_h \subseteq \mathcal{C}$ of chiplets. For simplicity, we often denote chiplets by global identifiers $c \in \mathcal{C}$. A mobile node transfers its input data to the edge, thus consuming bandwidth. Let R^{tot} be the total quantity of radio resources, also referred to as resource blocks (RBs). For each node n, the number R_n of bits it can transmit per RB is known. and the energy it consumes for radio transmissions is ϵ_n^{rad} J/bit. As for the delay associated with intra-chiplet, inter-chiplet, and off-chip communication (introduced in Section III-A), we model it through just two kinds of parameters:

- $\delta(c_1, c_2)$, the fixed delay incurred every time data is moved from chiplet c_1 to chiplet c_2, with $\delta(c, c) = 0$ accounting for intra-chiplet transfers with 0 delay. Since the number of hops between any two chiplets is fixed and known, the corresponding component can be embedded in the δ-parameters;
- $B(h_1, h_2)$, the total bandwidth available for transfers from MCM h_1 to MCM h_2, with $B(h, h)$ being the bandwidth available within MCM h for transfer between chiplets in $\mathcal{C}_h$.

Each chiplet $c \in \mathcal{C}$ is equipped with memory $M(c)$ and

computational capabilities $O(c)$. Using one unit of computing capability in a chiplet c on module h results in an energy consumption of $\epsilon^{\text{comp}}(h,c)$ J and $\epsilon^{\text{comm}}(h_1,h_2)$ is consumed for off-chip communication of each unit of data between MCMs h_1 and h_2. Similarly, $\epsilon^{\text{comm}}(c_1,c_2)$ is the energy consumption due to the inter-chiplet communication of one unit of data between chiplets $c_1, c_2 \in \mathcal{C}$.

B. Problem formulation

Decision variables. The first decision to make is the number r_n of RBs to assign to each node n. Once a sample reaches the edge computing platform, then one has to decide whether to use chiplet c on acceleration module h to run block l used for node n; we express this through binary variables $y(h,c,l,n)$. For each chiplet, c in MCM h, we denote the computational and memory resources to allocate for executing block l for node n by the continuous variables $o(h,c,l,n)$ and $m(h,c,l,n)$, respectively. Also, we denote the bandwidth allocated for data transfer between chiplet c_1 and c_2 with $b(c_1,c_2,l_1,l_2,n)$. Finally, with all the blocks allocated, we have to decide the rate of requests with which block l, running at chiplet c in MCM h, processes the samples coming from node n; we denote this decision variable by $x(h,c,l,n)$.

Constraints. The total number of RBs allocated to nodes cannot exceed the total number of available radio resources:

$$\sum_{n\in\mathcal{N}} r_n \leq R^{\text{tot}}. \tag{1}$$

Also, the application block(s) used to serve each node n should meet the target quality, i.e.,

$$q(a_n, L_n) \geq Q_n, \quad \forall n \in \mathcal{N}, \tag{2}$$

where $L_n = \{l \in \mathcal{L} \colon \exists c \in \mathcal{C} \colon y(h,c,l,n) = 1\} \subseteq \mathcal{L}$ denotes the blocks chosen to serve n.

Concerning the computation on each chiplet $c \in \mathcal{C}$, the chiplet's capabilities cannot be exceeded:

$$\sum_{l\in\mathcal{L}, n\in\mathcal{N}} o(h,c,l,n) \leq O(c), \quad \forall l \in \mathcal{L}, n \in \mathcal{N}. \tag{3}$$

Furthermore, there has to be enough memory to run all blocks:

$$m(c,l,n) \geq m^{\text{req}}(l), \quad \forall c \in \mathcal{C}, l \in \mathcal{L}, n \in \mathcal{N}. \tag{4}$$

Likewise, the memory in each chiplet cannot be exceeded:

$$\sum_{l\in\mathcal{L}, n\in\mathcal{N}} m^c(c,l,n) \leq M(c), \quad \forall c \in \mathcal{C}, \tag{5}$$

as well as the available bandwidth, either within the same MCM or between MCMs:

$$\sum_{\substack{c_1\in\mathcal{C}_{h_1}, c_1\in\mathcal{C}_{h_2},\\ l_1,l_2\in\mathcal{L}, n\in\mathcal{N}}} b(c_1,c_2,l_1,l_2,n) \leq B(h_1,h_2), \ \ \forall h_1,h_2 \in \mathcal{H}. \tag{6}$$

Then all samples generated at node n should be served:

$$\sum_{c\in\mathcal{C}} x(h,c,l,n) \geq \lambda_n, \quad \forall l \in \mathcal{L}, n \in \mathcal{N}, \tag{7}$$

and requests cannot be served if no suitable block is scheduled:

$$x(h,c,l,n) \leq y(h,c,l,n)\lambda_n, \quad \forall c \in \mathcal{C}, l \in \mathcal{L}, n \in \mathcal{N}. \tag{8}$$

The latency is the sum of the communication latency and computation delay components, as detailed in Sec. IV-C, which has to meet the following constraint:

$$d_n^{\text{rad}} + d_n^{\text{comp}} + d_n^{\text{comm}} \leq D_n, \quad \forall n \in \mathcal{N}. \tag{9}$$

Objective. Energy is consumed whenever any of the following actions is performed: *(i)* transmitting data over the network, *(ii)* executing computations, *(iii)* using memory and bandwidth. The problem objective is to minimize energy, subject to radio resources (1), quality (2), chiplet capability (3), memory (4)–(5), MCM bandwidth (6), service guarantee (7)–(8), and latency (9) constraints. By denoting with ϵ^{tot} the total energy consumption due to the above actions (detailed in Sec. IV-C), the problem can be formulated as follows:

$$\min_{b,o,r,x,y,m} \ \epsilon^{\text{tot}} \tag{10}$$
$$s.t.\ (1) - (9).$$

C. End-to-end latency and energy computation

To compute the end-to-end latency, we neglect the network latency at the edge and focus on the following components: *(i)* the latency over the radio channel d_n^{rad}; *(ii)* the computation delay d_n^{comp}; *(iii)* the communication delay within the MCM-based computing platform d_n^{comm}. To streamline the notation, we indicate with $H(c) = h \in \mathcal{H} \colon c \in \mathcal{C}_h$ the MCM containing chiplet $c \in \mathcal{C}$, and with $C(l,n) = c \in \mathcal{C} \colon y(h,c,l,n) = 1$ the chiplet running block l for node n.

For the radio latency, as in, e.g., [37], we model the channel as a generalized processor sharing (GPS) system. Samples generated by each node n correspond to customers in the queuing system, and the service rate associated with each node is given by $\frac{r_n R_n}{s_0(n)}$, i.e., the product of the quantity of resources assigned to n and the number of bits per resource unit, divided by the size of n's requests. Given the above and that the delay associated with the radio link corresponds to the sojourn time of the queuing system, we have:

$$d_n^{\text{rad}} = \frac{\lambda_n s_0(n)}{r_n R_n}. \tag{11}$$

We neglect instead the delay associated with the return of the inference output to the mobile node, due to the very small size of the corresponding message. To evaluate the computing delay, we model each chiplet as an M/D/1-PS system [38]. As per [39], its sojourn time is upper-bounded by: $\frac{O(c)}{o(h,c,l,n)(1-\rho(h,c))}$, where $\rho(h,c) = \frac{\sum_{l\in\mathcal{L}, n\in\mathcal{N}} x(h,c,l,n)}{O(c)}$ is the occupancy of chiplet c on module h, and $O(c)$ is the total computational capability of chiplet c. To obtain the total processing delay experienced by samples of node n, we consider the sum of the above block-specific components, i.e.,

$$d_n^{\text{comp}} = \sum_{c\in\mathcal{C}, l\in\mathcal{L}} y(h,c,l,n) \frac{\kappa(l,n)}{o(h,c,l,n)} \frac{O(c)}{1-\rho(h,c)}. \tag{12}$$

Notice how the complexity parameter $\kappa(l,n)$ also appears in the equation; intuitively, more complex processing requires more time (or more computational resources).

Last, we model the communication delays on the computing platform, which are incurred for every pair of chiplets $c_1, c_2 \in \mathcal{C}$ such that (i) they are consecutive, and (ii) they are both used to serve n's requests:

$$d_n^{\text{comm}} = \sum_{\substack{l_1,l_2:\ f(l_1,l_2)=1 \\ \wedge C(l_1)\neq C(l_2)}} \left(\delta(C(l_1), C(l_2)) + \frac{\lambda_n s(l_1,n)}{b(c_1,c_2,l_1,l_2,n)} \right). \tag{13}$$

It is important to remark that the two terms in Eq. (13) correspond to all the delay contributions described in [10, Sec. III-E]. Specifically, the first term captures the propagation latency (proportional to the number of hops) and the memory access latency (if c_1 and c_2 are on different MCMs), while the second term captures the transmission delay, which is directly proportional to the data to transfer and inversely proportional to the allocated bandwidth.

Regarding the total energy consumption, we have:

$$\epsilon^{\text{tot}} = \sum_{n\in\mathcal{N}} \lambda_n s_0(n) \epsilon_n^{\text{rad}} + \sum_{c\in\mathcal{C}, l\in\mathcal{L}, n\in\mathcal{N}} \epsilon^{\text{comp}}(h,c) o(h,c,l,n) + \sum_{c_1,c_2\in\mathcal{C}, l_1,l_2\in\mathcal{L}, n\in\mathcal{N}} \epsilon^{\text{comm}}(H(c_1),H(c_2)) b(h,c_1,c_2,l_1,l_2,n) \tag{14}$$

where, due to the very small size of the associated message, we have neglected the component due to the delivery of the inference outcome to the mobile node.

D. Problem complexity

Property 1. *The problem in (10) is NP-hard.*

Proof: The decision variables $x()$ and $y()$ take discrete integer values, while $b(), o(), r_n, m()$ take continuous values. Also, Constraint (9) includes terms where $r_n, o()\cdot x()$, and $b()$ appear at the denominator and $y()$ at the numerator (see also e.g., (12)), implying a nonlinear rational function. The number of terms in the set of decision variables depends on $\mathcal{L}$, further increasing complexity. Our formulation thus is a non-convex MINLP problem, which, as shown in [40], is NP-hard. ■

V. The CLEAR solution strategy

We tackle the complexity of the problem above by proposing an iterative solution strategy, named CLEAR—CLimbing at the Edge for Allocation of Resources. CLEAR consists of two main steps. First (*(a)* below), we decompose the problem into an inner and an outer subproblem, and prove that the inner problem is a geometric programming (GP) problem, hence, it can be efficiently solved to optimality. Second (*(b)* below), we show that the function connecting decisions and outcome in the outer problem is submodular, so that the problem can be solved through a simple hill-climbing procedure with constant competitive ratio. We then prove that CLEAR gives solutions within $1-\frac{1}{\mathrm{e}}$ from the optimum, in polynomial time (Sec. V-A).

a) Decomposition and inner problem: We decompose the optimization problem in an outer problem that sets the value of the decision variables $y(h,c,l,n)$, i.e., whether to use chiplet c on MCM h to run block l for node n, and in an inner problem that is a variant of the original one but with fixed values for the $y()$'s. We prove that the inner problem can be solved to optimality—in polynomial time—as it belongs to the class of geometric programming (GP) [41]:

Property 2. *The inner problem belongs to the GP class.*

Proof: All constraints and the objective in the original problem are posynomial. In fact, although the constraint about M/D/1-PS does not look posynomial, it can be re-written as the sum of a geometric series ($\frac{1}{1-x} = \sum_i x^i$), hence, it is also posynomial. The thesis then follows in virtue of [42]. ■

As a consequence of the above property, the inner problem can be solved to optimality through any convex solver. The worst-case time complexity is polynomial, and actual solution times are often much faster in concrete cases.

b) Outer problem and hill-climbing procedure: To tackle the outer problem, we start from a feasible solution ξ of the original problem and consider the set $\mathcal{Y} = \{(h,c,l,n) \in \mathcal{C}\times\mathcal{L}\times\mathcal{N} : y(h,c,l,n) \in \xi \wedge y(h,c,l,n)=1\}$. Then our goal is to find a near-optimal set of quadruplets $\mathcal{Z} \subseteq \mathcal{Y}$ for which all corresponding $y(h,c,l,n)$ can be set to zero, i.e., a set of block instances that can be disabled. Let $f()$ be the function linking a set of quadruplets (h,c,l,n) with the corresponding improvement in the objective function of our original problem, which is obtained when the blocks corresponding to (h,c,l,n) are disabled (i.e., $y(h,c,l,n)=0$). Function f has an important property, i.e., it is submodular.

Property 3. *Function $f(\mathcal{Z})$ is submodular.*

Proof: Given $\mathcal{Z}$—the set of block instances to disable—to prove the thesis, the so-called *diminishing returns* property has to hold. That is, given an element $z \in \mathcal{Y}$ and set $\mathcal{Z}'$, such that $\mathcal{Z}' \subseteq \mathcal{Z}$, then adding z to the larger set (i.e., $\mathcal{Z}$) brings no more gain than adding it to the smaller set (i.e., $\mathcal{Z}'$). More formally: $f(\mathcal{Z}\cup\{z\}) \leq f(\mathcal{Z}'\cup\{z\})$. In our case, for such a property to hold, it is enough that disabling a block instance z in a situation where there are more enabled block instances, i.e., those in $\mathcal{Y}\setminus\mathcal{Z}$, yields at least the same savings as those obtained by disabling the same element z in a situation where there are fewer disabled block instances, i.e., those in $\mathcal{Y}\setminus\mathcal{Z}'$. This is indeed the case, as the benefit from removing z comes from the *difference* of two components: (i) the energy saved by disabling z; (ii) the extra energy needed to take corrective actions. The first component will be the same in all cases. As for the second, having more enabled block instances (i.e., those in $\mathcal{Y}\setminus\mathcal{Z}'$ instead of those in $\mathcal{Y}\setminus\mathcal{Z}$) results in more opportunities for corrective action, hence, lower (in the worst case, equal) cost. ■

We can then find a near-optimal set of instances to disable through the greedy procedure, based on the hill-climbing paradigm: 1) start from an empty set $\mathcal{Z}=\emptyset$; 2) at every

iteration, disable the block instance yielding the largest gain; 3) stop when that is not possible anymore, i.e., disabling any further instance would result in an infeasible solution.

After the hill-climbing procedure, we need to solve the inner problem considering the $y(h,c,l,n)$ set to the values found by solving the outer problem. We recall that the inner problem can be solved to the optimum (Prop. 2).

A. Solution analysis

Our CLEAR solution has two remarkable aspects: it has a very low, namely, polynomial computational complexity, and it produces provably near-optimal solutions.

Property 4. *The CLEAR procedure has a polynomial worst-case computational complexity.*

Proof: CLEAR consists of two loops. The inner one, as per [41], has polynomial computational complexity. The outer one runs at most once per possible block instance, and the number of such instances is polynomial, to wit, $|\mathcal{C}||\mathcal{L}||\mathcal{N}|$. Thus, the overall computational complexity is polynomial. ■

Concerning optimality, CLEAR has $1-\frac{1}{\mathrm{e}}$ competitive ratio, i.e., the solution found by CLEAR is guaranteed to be within a small constant factor from the optimal solution, providing a strong theoretical bound on its performance. More formally:

Property 5. *CLEAR has a constant competitive ratio of* $1-\frac{1}{\mathrm{e}}$.

Proof: No optimality is lost in CLEAR's inner loop, as the problem belongs to the GP class (as per Prop. 2), which can be efficiently solved to optimality [41]. The outer loop amounts to solving a submodular problem (as per Prop. 3) via a hill-climbing procedure, which, as shown in [43, Thm. 4.7], yields solutions within $1-1/\mathrm{e}$ from the optimum. ■

VI. Experimental Results

We now present the use cases and benchmarks used for our performance evaluation, as well as the experimental settings and the results for a small-scale scenario (where the optimum can be computed) and a large-scale scenario.

AI applications: The considered AI applications and their inference requirements are summarized in Table III. Such applications use the DNN models with EEs introduced in Table I and investigated in our ablation study in Sec. III.

Data-rate trace and radio parameters: To model the dynamic data rate experienced by mobile nodes over the radio link, we use TCP throughput traces collected through real-world experiments [44]. Each experiment, which lasts 150 s, is characterized by a mobile node that travels along a fixed path in indoor and outdoor settings, while transmitting data through simultaneous WiFi 802.11ac connections. The TCP packets are generated using the iPerf3 tool, and the link performance is captured every 100 ms. In our experiments, we associate each mobile node with a starting point in the data trace chosen at random. Fig. 4(a) shows the throughput trace for two mobile nodes. The Mobile 1 node has a moderate to high data rate, i.e., moderate to good channel condition, during the 15–55 s time interval, while such a condition for the Mobile 2 node occurs in the 30–78 s time interval. Furthermore, radio parameters are set as follows: R^{tot}=100 at 3.4 GHz (50 MHz channel bandwidth with 15 Khz carrier spacing), and $\epsilon_n^{\mathrm{rad}}$=0.3 μJ/bit $\forall n$ [45].

TABLE III
AI applications tasks and their requirements

Task	Application	Latency	Quality
Image Classification	Autonomous Vehicles		Acc:
B-LeNet	*Traffic Sign Recognition*	<18 ms	>80%
LGViT			>88%
Object Detection	Retail & Security		mAP:
B-ResNet110	Surveillance	<300 ms	>70%
Semantic Segmentation	Autonomous Vehicles		mAP:
Do-ViT	*Environmental Classification for Path Planning*	<280 ms	>80%

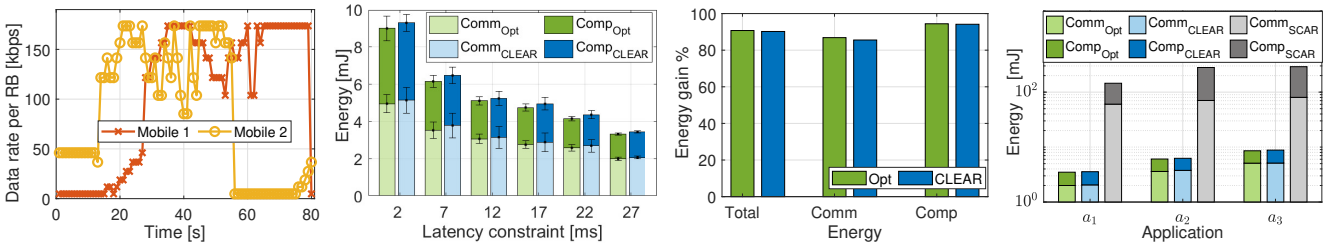
Benchmarks: We consider the following benchmarks:

• *Optimum (Opt),* obtained through exhaustive search (only in the small scenario where its computation is feasible).

• *Scheduling of multi-model AI workload on MCM accelerators (SCAR) and fair radio RB allocation* [10]. It includes the SCAR first-fit greedy-packing heuristic algorithm, combined with proportionally fair wireless resource allocation. We take this as a benchmark, since, to our knowledge, no scheme exists that tackles the joint MCM and communication resource allocation problem for dynamic DNNs.

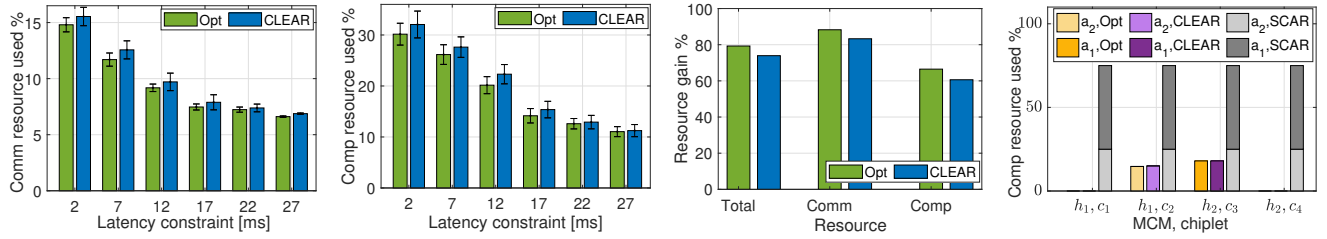
Small-scale scenario: It includes: 2 mobile nodes, 2 MCMs with 1 Shidiannao chiplet and 1 NVDLA chiplet each (memory and bandwidth as in Table II), and 3 applications (one of which can be implemented through two different models). Applications (and corresponding models) are associated with mobile nodes with uniform probability. To investigate the impact of the latency target value on energy and resource consumption, we apply CLEAR and perform Monte Carlo simulations over 1,000 iterations, with 95% confidence interval. The values of energy expenditure for inter-chiplet and off-chip communication come from the MAESTRO simulator.

Fig. 4(b) presents the energy consumption for various latency targets. The energy cost due to communication accounts for both radio transmissions and the communication within and between MCMs, with the latter components being however negligible compared to the radio component (pJ vs. mJ). As expected, a tighter latency target results in a larger energy consumption, both for computation and communication, as more resources have to be allocated. More interestingly, CLEAR closely matches Opt. Also, it is worth observing that both communication and computation contribute to the system energy cost, with communication prevailing as applications set more relaxed inference time constraints. The energy gain with respect to SCAR is depicted in Fig. 4(c).



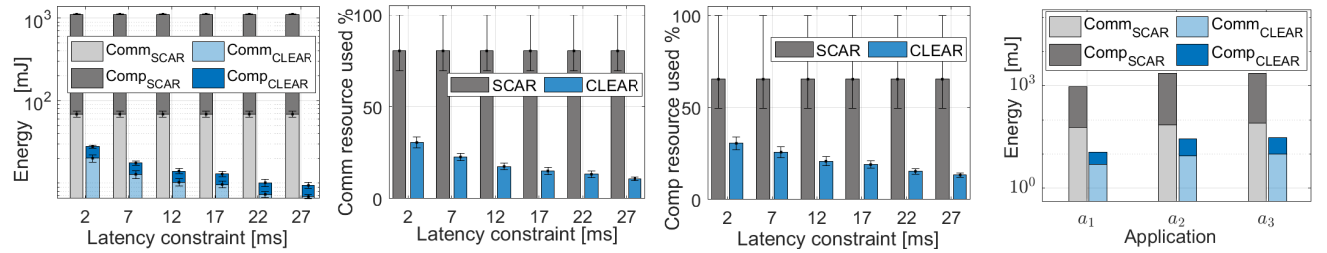
(a) (b) (c) (d)

Fig. 4. (a) Data rate trace. Results in the small-scale scenario: (b) Energy consumption of Opt and CLEAR as the latency target varies; (c) Energy gain of Opt and CLEAR with respect to SCAR [10]; (d) Energy consumption of Opt, CLEAR and SCAR for the image classification—LgViT (a_1), object detection—B-ResNet (a_2), and semantic segmentation—DoViT (a_3) applications.



(a) (b) (c) (d)

Fig. 5. Small-scale scenario: (a) Communication and (b) Compute resource used for Opt and CLEAR vs. latency target; (c) Opt and CLEAR resource gain w.r.t. SCAR [10]; (d) Example of MCM/chiplet assignment decisions for the image classification—LgViT (a_1) and object detection—B-ResNet (a_2) applications.



(a) (b) (c) (d)

Fig. 6. Large-scale scenario: (a) Energy consumption, (b) Communication resource usage, and (c) Compute resource usage, as the latency target changes. (d) Energy consumption of CLEAR and SCAR for the image classification—LgViT (a_1), object detection—B-ResNet (a_2), and semantic segmentation—DoViT (a_3) applications.

CLEAR greatly outperforms SCAR, yielding an energy gain that exceeds 80%, for computation, communication, and total energy consumption – essentially the same gain as the one yielded by the optimum. Furthermore, as shown in Fig. 4(d), CLEAR's communication and computation energy consumption is almost the same as Opt and significantly less than SCAR, for all applications.

Such performance is confirmed by the usage of communication resources (quantified by the number of RBs) and computing resources as the latency target varies, shown in Fig. 5(a) and Fig. 5(b) (resp.). Remarkably, CLEAR can achieve nearly the same performance as the optimum. From these plots and Fig. 4(b), we also observe that moving from 20 ms to a more stringent 2 ms latency target increases energy and resource consumption by over 2×, thus underscoring the impact of such an application requirement on the system efficiency. The resource gain of CLEAR and Opt with respect to the SCAR benchmark is shown in Fig. 5(c).

Large-scale scenario: Next, we apply CLEAR to a scenario with 32 mobile nodes, and 4 MCMs with 2 Shidiannao chiplets and 2 NVDLA chiplets each (memory and bandwidth as in Table II). As before, applications (and models for Traffic Sign Recognition) are uniformly assigned to mobile nodes. Figures 6(a)–6(c) compare CLEAR to the SCAR benchmark, in terms of average energy, communication (i.e., allocated RBs), and computation resource consumption, respectively, with the average value computed over the entire duration of the radio link measurement trace and the number of mobile nodes.

VII. Conclusions

We addressed the efficient offloading and execution of heterogeneous mobile applications on chiplet-based accelerators. In these scenarios, it is essential to adapt to fluctuating resource availability and radio link conditions by intelligently and dynamically configuring the DNNs enabling such applications—including those with early exits—and splitting the model across chiplets. We solved this problem by introducing CLEAR, a polynomial worst-case complexity framework that leverages geometric and submodular optimization. Our evaluation—using models from canonical to modern transformer-based dynamic DNNs and real-world radio measurement traces—demonstrates that CLEAR matches the optimum while surpassing its benchmark in reducing resource and energy consumption by over 70% and 80%, respectively.

References

[1] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, 2021.

[2] Z. Aghapour, S. Sharifian, and H. Taheri, "Task offloading and resource allocation algorithm based on deep reinforcement learning for distributed AI execution tasks in IoT edge computing environments," *Computer Networks*, vol. 223, 2023.

[3] F. Saeik *et al.*, "Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions," *Computer Networks*, vol. 195, 2021.

[4] B. Gu *et al.*, "AI-enabled task offloading for improving quality of computational experience in ultra dense networks," *ACM Trans. Internet Tech.*, vol. 22, 2022.

[5] M. Mendula *et al.*, "A novel middleware for adaptive and efficient split computing for real-time object detection," *Pervasive & Mobile Computing*, vol. 108, 2025.

[6] F. Cunico *et al.*, "I-split: Deep network interpretability for split computing," in *ICPR*, 2022.

[7] Y. Matsubara, R. Yang, M. Levorato, and S. Mandt, "Sc2 benchmark: Supervised compression for split computing," 2023.

[8] Y. S. Shao *et al.*, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *IEEE/ACM Intern. Symp. on Microarchitecture*, 2019.

[9] L. Fusco *et al.*, "Understanding data movement in tightly coupled heterogeneous systems: A case study with the grace hopper superchip," 2024.

[10] M. Odema, L. Chen, H. Kwon, and M. A. A. Faruque, "SCAR: Scheduling multi-model ai workloads on heterogeneous multi-chiplet module accelerators," 2024.

[11] M. Sartori, C. Singhal, N. Roy, D. Brunelli, and J. Gross, "AI and vision based autonomous navigation of nano-drones in partially-known environments," in *IEEE DCOSS-IoT*, 2025.

[12] X. Chen, M. Li, H. Zhong, Y. Ma, and C.-H. Hsu, "DNNOff: Offloading DNN-based intelligent IoT applications in mobile edge computing," *IEEE Trans. Industrial Informatics*, vol. 18, 2022.

[13] G. Zheng *et al.*, "Computation-aware offloading for dnn inference tasks in semantic communication assisted MEC systems," *IEEE Trans. Wireless Comm.*, vol. 24, 2025.

[14] W. Xiao *et al.*, "Adaptive compression offloading and resource allocation for edge vision computing," *IEEE Trans. on Cognitive Comm. and Netw.*, vol. 10, 2024.

[15] C. Puligheddu, J. Ashdown, C. F. Chiasserini, and F. Restuccia, "SEM-O-RAN: Semantic O-RAN slicing for mobile edge offloading of computer vision tasks," *IEEE Trans. on Mobile Computing*, 2023.

[16] C. Puligheddu, N. Varshney, T. Hassan, J. Ashdown, F. Restuccia, and C. F. Chiasserini, "OffloaDNN: Shaping DNNs for scalable offloading of computer vision tasks at the edge," in *IEEE ICDCS*, 2024.

[17] W. Seo, S. Cha, Y. Kim, J. Huh, and J. Park, "SLO-aware inference scheduler for heterogeneous processors in edge platforms," *ACM Trans. on Archit. Code Optim.*, vol. 18, 2021.

[18] Y. Wang *et al.*, "Balancing energy efficiency and real-time performance in GPU scheduling," in *IEEE RTSS*, 2021.

[19] J. Cai, Y. Wei, Z. Wu, S. Peng, and K. Ma, "Inter-layer scheduling space definition and exploration for tiled accelerators," in *ACM Annual Intern. Symp. on Computer Architecture*, 2023.

[20] J. Zhang *et al.*, "INDM: Chiplet-based interconnect network and dataflow mapping for DNN accelerators," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Sys.*, vol. 43, 2024.

[21] H. Sharma, P. Dhingra, J. Doppa, U. Ogras, and P. P. Pande, "A heterogeneous chiplet architecture for accelerating end-to-end transformer models," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 30, no. 3, 2025.

[22] X. Wang *et al.*, "On task mapping in multi-chiplet based many-core systems to optimize inter- and intra-chiplet communications," *IEEE Trans. on Computers*, vol. 74, 2025.

[23] A. Das, E. Russo, and M. Palesi, "Multi-objective hardware-mapping co-optimisation for multi-DNN workloads on chiplet-based accelerators," *IEEE Trans. on Computers*, vol. 73, 2024.

[24] L. L. Schiavo, J. A. Ayala-Romero, A. Garcia-Saavedra, M. Fiore, and X. Costa-Perez, "Yinyangran: Resource multiplexing in GPU-accelerated virtualized RANs," in *IEEE INFOCOM*, 2024.

[25] J. A. Ayala-Romero, L. L. Schiavo, A. Garcia-Saavedra, and X. Costa-Perez, "Mean-field multi-agent contextual bandit for energy-efficient resource allocation in vrans," in *IEEE INFOCOM*, 2024.

[26] L. L. Schiavo *et al.*, "CloudRIC: Open Radio Access Network (O-RAN) virtualization with shared heterogeneous computing," in *ACM MobiCom*, 2024.

[27] L. Lo Schiavo *et al.*, "Datasets used in https://doi.org/10.1145/3636534.3649381," 2024.

[28] C. Singhal, Y. Wu, F. Malandrino, M. Levorato, and C. F. Chiasserini, "Resource-aware deployment of dynamic DNNs over multi-tiered interconnected systems," in *IEEE INFOCOM*, 2024.

[29] H. Rahmath P, V. Srivastava, K. Chaurasia, R. G. Pacheco, and R. S. Couto, "Early-exit deep neural network - a comprehensive survey," *ACM Comput. Surv.*, vol. 57, 2024.

[30] C. Singhal, Y. Wu, F. Malandrino, M. Levorato, and C. F. Chiasserini, "Distributing inference tasks over interconnected systems through dynamic DNNs," *IEEE Trans. on Networking*, 2025.

[31] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: Extending MNIST to handwritten letters," in *IEEE IJCNN*, 2017.

[32] S. Teerapittayanon *et al.*, "Branchynet: Fast inference via early exiting from deep neural networks," in *IEEE ICPR*, 2016.

[33] Y. Liu *et al.*, "Dynamic token-pass transformers for semantic segmentation," 2023.

[34] G. Xu *et al.*, "Lgvit: Dynamic early exiting for accelerating vision transformer," in *ACM Intern. Conf. Multimedia*, 2023.

[35] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, "Heterogeneous dataflow accelerators for multi-DNN workloads," in *IEEE HPCA*, Los Alamitos, CA, USA, 2021.

[36] H. Kwon *et al.*, "MAESTRO: A data-centric approach to understand reuse, performance, and hardware cost of DNN mappings," *IEEE Micro*, vol. 40, 2020.

[37] A. Brandt and M. Brandt, "Waiting times for M/M systems under state-dependent processor sharing," *Queueing Systems*, vol. 59, 2008.

[38] Y. Zhang, W. Zheng, X. Dong, and S. Gan, "A performance analytical approach based on queuing model for network-on-chip," in *Intern. Symp. on Parallel Arch., Alg. and Programming*, 2010.

[39] R. Egorova, B. Zwart, and O. Boxma, "Sojourn time tails in the M/D/1 processor sharing queue," *Probability in the Engineering and Informational Sciences*, vol. 20, 2006.

[40] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.

[41] S. Boyd, S.-J. Kim, L. Vandenberghe, and A. Hassibi, "A tutorial on geometric programming," *Optimization and engineering*, 2007.

[42] R. J. Duffin and E. L. Peterson, "Geometric programming with signomials," *J. Optimization Theory and App.*, 1973.

[43] R. K. Iyer and J. A. Bilmes, "Submodular optimization with submodular cover and submodular knapsack constraints," *Advances in neural information processing systems*, 2013.

[44] C. Singhal, Y. Wu, F. Malandrino, S. L. G. Contreras, M. Levorato, and C. F. Chiasserini, "Resource-efficient sensor fusion via system-wide dynamic gated neural networks," in *IEEE SECON*, 2024.

[45] F. Jalali *et al.*, "Fog computing may help to save energy in cloud computing," *IEEE JSAC*, 2016.