

SPARX: Enabling Resource-Aware Model Partitioning for Distributed Inference on the Heterogeneous AI-Native Edge

Washington R. D. Silva, Fábio L. Verdi
Department of Computer Science
Federal University of São Carlos (UFSCar)
Sorocaba, Brazil
washingtonrds@estudante.ufscar.br
verdi@ufscar.br

Luciano S. Fraga, Kleber V. Cardoso
Institute of Informatics
Universidade Federal Goiás (UFG)
Goiânia, Brazil
lucianosouza@discente.ufg.br
kleber@ufg.br

Andrew Williams
Ericsson Research
Ericsson AB
Stockholm, Sweden
andrew.williams@ericsson.com

Omer Rana
School of Computational and Mathematical Sciences
Cardiff University
Cardiff, Wales
ranaof@cardiff.ac.uk

Luiz F. Bittencourt
Institute of Computing
University of Campinas
Campinas, Brazil
bit@ic.unicamp.br

Abstract—As 6G evolves into an AI-Native ecosystem, efficiently executing compute-intensive machine learning algorithms (such as Deep Neural Networks) on edge devices is essential to meet stringent application latency constraints. Partitioning these models across a resource-constrained, heterogeneous compute continuum (e.g., CPUs, GPUs, DPUs, FPGAs) introduces severe scheduling complexity. Effectively navigating this requires capturing architectural heterogeneity without time consuming, exhaustive physical profiling. To address this, we introduce SPARX, a resource-aware model partitioning framework for heterogeneous edge infrastructure. SPARX translates AI models into a Directed Acyclic Graph (DAG), incorporating a recovery layer that reconstructs structural metadata lost during node fusion. By coupling estimated computational demands with hardware specifications, our hierarchical multi-stage optimization scheme generates deployment plans that jointly minimize energy consumption and makespan under strict latency constraints. We evaluated SPARX on a mobile Augmented Reality use case with two YOLO11 variants over a disaggregated 5G RAN topology model. Results show that explicitly modeling operation fusion reduces estimated energy consumption, and lightweight heuristics generated feasible deployment plans within milliseconds, where exact optimization approaches proved impractical.

Index Terms—Model Partitioning, Distributed Inference, Resource Allocation, Heterogeneous Devices, Object Detection.

I. INTRODUCTION

The convergence of sensing, communication and computation is a defining characteristic of the 6G era. As networks evolve into Artificial Intelligence (AI)-Native platforms, they are expected to host increasingly complex workloads, supporting time-sensitive user applications while executing the intrinsic intelligence of the network itself. This transition is already being formalized, with 3GPP Release 18 establishing the technical framework for integrating AI into the wireless air interface to enhance critical use cases such as Beam Management (BM) and Channel State Information (CSI) feedback [1]. In these scenarios, the inference engine cannot be decoupled from the data source; decisions must be executed directly at the Radio Unit (RU) or distributed across the adjacent compute

continuum to satisfy the stringent latency constraints of these applications [1], [2].

Implementing this vision, however, presents a resource paradox. AI models are highly compute-intensive, requiring processing capabilities often not available at the edge. This constraint necessitates workload distribution: rather than executing a monolithic model, a Deep Neural Network (DNN) must be partitioned into discrete components and distributed collaboratively across the edge infrastructure [3]. Yet, this distribution introduces significant complexity: the modern edge is highly heterogeneous, comprising CPUs, GPUs, and specialized accelerators like Data Processing Units (DPUs) and Field Programmable Gate Arrays (FPGAs). Managing an AI-Native system amidst this structural complexity requires careful planning and efficient resource allocation frameworks.

While model partitioning literature has evolved from simple layer-by-layer distribution to advanced Directed Acyclic Graph (DAG) scheduling [4], current frameworks exhibit fundamental gaps when applied to the 6G edge. First, existing works evaluate heterogeneity through a limited scope, typically confined to standard CPUs, GPUs, or embedded IoT devices, and consequently fail to capture the broader infrastructure diversity of the modern edge. Second, to estimate inference latency, current approaches rely on physical device profiling. While accurate for known setups, this dependency creates a severe scalability bottleneck; given the combinatorial explosion of emerging AI architectures and deployment hardware, exhaustive physical benchmarking undermines the flexibility to rapidly adapt deployments to novel models or infrastructure updates. Third, most frameworks optimize a single metric, whereas 6G demands joint optimization of latency and energy [1]. Finally, current partitioning engines ignore runtime-specific behaviors, such as compiler-level operation fusion, generating static plans that diverge from actual hardware execution.

To bridge these gaps, we introduce SPARX (Scalable Parti-

tioning for AI Resource eXploration), a resource-aware model partitioning framework designed to realize viable distributed inference across the modern heterogeneous 6G edge. To capture infrastructure diversity without exhaustive physical benchmarking, we translate the target DNN into a DAG Intermediate Representation (IR) to estimate the computational demand (in FLOPs) of each node while accounting for runtime-level graph optimizations. Additionally, we characterize hardware specifications to derive an analytical performance profile that approximates real-world execution. Based on this profile, we employ a hierarchical Mixed-Integer Linear Programming (MILP) formulation alongside greedy heuristics to generate scheduling plans that minimize overall makespan and energy consumption, strictly constrained by a 20 ms latency SLA. Finally, we demonstrate SPARX’s capabilities using a Mobile Augmented Reality (MAR) object detection use case, evaluating two model variants (YOLO11-nano and YOLO11-medium) within a disaggregated 5G hierarchical RAN topology [5], featuring varying tiers of CPUs, GPUs, DPUs, and FPGAs.

The remainder of this paper is organized as follows: Section II reviews related work; Section III details the SPARX architecture, DAG IR profiling, and the hierarchical MILP formulation; Section IV describes our experimental protocol and evaluation results; Finally, Section V concludes the paper.

II. RELATED WORK

The landscape of distributed inference at the network edge has transitioned from simple computation offloading to complex, multi-device topology optimization. Existing model partitioning frameworks can be categorized based on their structural assumptions, profiling strategies, and hardware scopes. Early paradigms, such as Neurosurgeon [6] and collaborative edge-cloud architectures [7], treated DNNs as strictly linear structures, identifying a single “split point” to offload compute-intensive tasks. To parallelize workloads across edge clusters, MoDNN [8] subsequently established the baseline for horizontal layer-by-layer model slicing.

Since layer-wise slicing still induces memory bottlenecks within large convolutional feature maps, DeepThings [9] introduced spatial tile partitioning, which CoEdge [10] subsequently refined by mitigating redundant data overlaps during distribution. To accommodate modern AI architectures utilizing branching topologies, EdgeFlow [11] later extended these methods to natively support DAGs, an IR capable of capturing such complex model topologies. However, these frameworks struggle under strict real-time constraints: CoEdge fails to converge below 50 ms, while EdgeFlow requires approximately 2 seconds of inference time for YOLOv5.

To overcome performance bottlenecks, recent frameworks have introduced sophisticated partitioning and scheduling mechanisms. For instance, DistrEdge [12] utilizes Deep Reinforcement Learning (DRL) alongside spatial layer-volume slicing to reduce computation latency below 50 ms, though its communication overhead still violates ultra-low-latency SLAs.

Similarly, Zheng et al. [13] advance flexible workload management through a multi-objective formulation that jointly optimizes latency, device energy, and cloud rental costs. However, despite these algorithmic progressions, both frameworks share a critical shortcoming regarding real-world storage dynamics, omitting memory capacity and read/write delays from their system models. Ultimately, both approaches face significant limitations within shared 6G infrastructures, where concurrent applications induce severe resource contention.

Furthermore, the aforementioned works rely on hardware profiling to guide scheduling or to feed regression models for latency prediction. To circumvent the exhaustive and costly physical benchmarking required across diverse deployment configurations, current literature shows a growing trend toward performance modeling. However, the scope of these performance modeling frameworks has remained strictly confined to homogeneous cloud environments [14]–[16].

Finally, a deep disconnect remains between current literature and the diverse hardware fabric defining the AI-Native edge. Most partitioning engines restrict their scope to general-purpose CPUs and standard embedded modules like Raspberry Pi or NVIDIA Jetson [4]. However, modern edge deployments encompass a hybrid mix of domain-specific accelerators (e.g., discrete GPUs) and programmable network devices, such as DPUs and FPGAs [3]. Consequently, bridging these fundamentally different computing fabrics within a unified optimization framework remains a promising yet underexplored direction for current distributed inference research [17].

These limitations cause existing partitioning frameworks to fall short of meeting the specific demands of AI-Native infrastructures. SPARX directly addresses such gaps by replacing physical benchmarking with an analytical performance model, extending hardware support to encompass diverse computing units (CPUs, GPUs, DPUs, and FPGAs), incorporating runtime optimizations and memory capacity directly into its partitioning logic, and accommodating multi-objective scheduling (e.g., energy, latency, and computational load) to enforce strict SLA compliance.

III. METHODOLOGY

To realize viable deployment plans for distributed inference across the heterogeneous AI-Native edge, we map five fundamental parameters into a multidimensional search space to establish the formal constraints and coefficients utilized by the partitioning engine: (1) **AI Models**: Pre-trained DNNs exported from standard frameworks; (2) **Hardware Specifications**: Processing throughput, memory capacity, memory bandwidth, Thermal Design Power (TDP), and PCIe bandwidth derived from manufacturer datasheets; (3) **Application Constraints**: Maximum allowable end-to-end latency (e.g., 20 ms) acting as a hard boundary; (4) **Network Topologies**: Inter-edge server bandwidth and transmission latencies; and (5) **Resource Contention**: Background infrastructure utilization factors. By mathematically modeling these parameters, SPARX seamlessly supports workloads ranging from internal

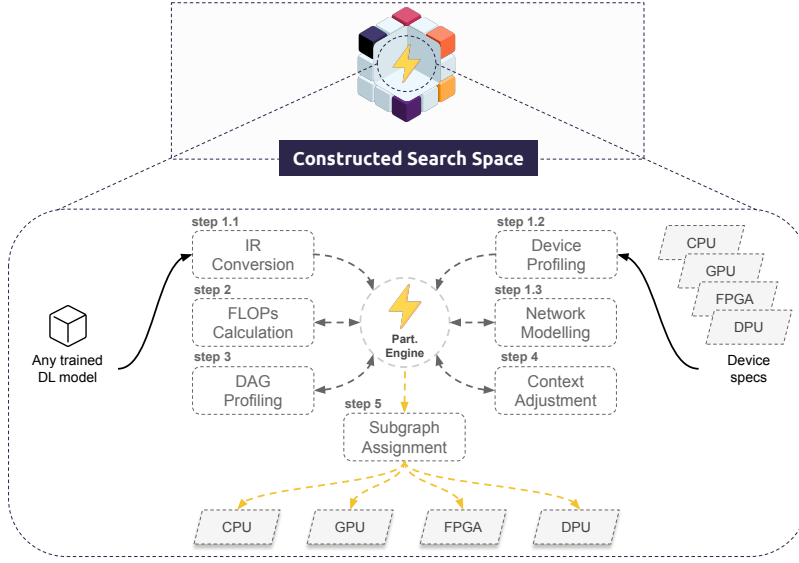


Fig. 1: SPARX workflow: (1) the AI model and edge infrastructure is abstracted into a unified performance profile; (2) baseline metrics are adjusted for operational context; (3) partitioning engine uses these to derive a suitable deployment strategy to satisfy global latency constraints.

network intelligence to distributed, user-centric edge applications.

As illustrated in Fig. 1, the framework first converts the model into an ONNX IR [18] expressed as a DAG $G = (\mathcal{V}, \mathcal{E})$. This ensures hardware interoperability by standardizing the model for a common execution engine (ONNX Runtime). Each node $v_i \in \mathcal{V}$ denotes a discrete computational operation (e.g., convolution, ReLU, GeMM) whose encoded attributes enable SPARX to statically derive compute workloads (FLOPs), offering a lightweight alternative to physical benchmarking. Furthermore, each directed edge $e_{ij} \in \mathcal{E}$ characterizes data dependencies and execution precedence, enabling SPARX to safely split complex structures (e.g., skip connections) across devices. Concurrently, the infrastructure is abstracted into a set of heterogeneous devices $\mathcal{D}$ across edge servers $\mathcal{S} = \{s_1, \dots, s_{|\mathcal{S}|}\}$. For each device, a standardized profile (Step 1.2) captures processing throughput (GFLOPs), memory capacity (GB), memory bandwidth (GB/s), PCIe bandwidth (GB/s), and power. Accordingly, we incorporate programmable networking devices as specialized compute units by parameterizing their onboard resources, such as DPU ARM processors or FPGA overlays. To estimate operational power without real-time measurements, we adopt the device’s TDP as a conservative upper-bound proxy for peak power consumption, matching established Machine Learning profiling conventions [19].

Consequently, SPARX is inherently agnostic to both DNN architectures and physical topologies. Since the core partitioning logic relies on standardized ONNX graphs and parameterized device profiles, extending the framework to accommodate novel AI models or custom infrastructure requires only updating the input parameters.

SPARX performs analytical performance estimation by pro-

filting each node $v_i \in \mathcal{V}$ via static shape inference using *onnx_tool* [20], calculating tensor boundaries based on operator attributes (kernel size, stride, padding) without execution. However, default upstream graph optimizations in ONNX Runtime compress topologies prior to execution via node fusion [21], introducing non-standard compound kernels (e.g., *FusedMatMul*, *QuickGelu*, fused *Split*) that lack native registration in standard profiling tools. To circumvent this, we introduce a recovery layer that reconstructs the structural metadata and computational demands of these fused primitives in two stages.

First, the recovery layer inspects the optimized DAG to identify unregistered nodes and extracts their underlying Protobuf attributes. For *FusedMatMul*, output boundaries are dynamically resolved by evaluating Boolean transpose flags to maintain spatial consistency. Similarly, activation bounds for *QuickGelu* are identity-mapped from input dimensions, and fused *Slice* operations are parsed to explicitly allocate matching tensor shapes to downstream nodes. Second, we calculate the Multiply-Accumulate (MAC) counts based on specific operator behaviors. For *FusedMatMul*, spatial dimensions (M rows, N columns, and shared inner dimension K) are extracted from input tensors and scaled by higher-order batch sizes to capture the General Matrix Multiply (GEMM) volume:

$$\text{MACs}_{\text{FusedMatMul}} = \text{Batch} \times M \times N \times K \quad (1)$$

For *QuickGelu* nodes, the computational volume is calculated by multiplying total tensor elements by a base cost of 34 MACs, mirroring *onnx_tool*’s implementation for standard *Gelu* operators, allocating 32 operations for exponential curve approximation (EXP_{MACs}) and 2 for scaling multiplications

(MUL_{MACs}) [20]:

$$\text{MACs}_{\text{QuickGelu}} = \text{Total}_{\text{elements}} \times (\text{EXP}_{\text{MACs}} + 2 \times \text{MUL}_{\text{MACs}}) \quad (2)$$

The fidelity of this estimation was validated by verifying that the reconstructed total matches the aggregate MAC count of the constituent operators prior to fusion.

From the resolved MAC counts, the total floating-point operations (Step 2) for any node v_i is derived as:

$$\text{FLOPs}(v_i) = 2 \times \text{MACs}(v_i) \quad (3)$$

The raw memory footprint is simultaneously estimated by multiplying intermediate tensor elements by the data type byte size (e.g., 4 bytes for 32-bit precision), establishing a deterministic bound for device memory capacity requirements.

SPARX integrates these workloads and hardware profiles to formulate a comprehensive performance model (Step 3), defining three primary latency components (in ms) for any node v_i mapped to device d_k :

Computation Latency: The raw execution time derived from the ratio of scaled GigaFLOPs to hardware processing throughput:

$$T_{\text{comp}}(v_i, d_k) = \left(\frac{\text{FLOPs}(v_i)/10^9}{\text{Throughput}(d_k)} \right) \times 1000 \quad (4)$$

Memory I/O Overhead: Bounded by tensor read/write cycles to local memory, evaluated by scaling the tensor volume (GB) against internal device memory bandwidth:

$$T_{\text{mem}}(v_i, d_k) = \left(\frac{(\text{Data}(v_i)/10^9) \times 2}{\text{BW}_{\text{mem}}(d_k)} \right) \times 1000 \quad (5)$$

Communication Latency: Dictated by data transfers across the local PCIe bus during host-to-device/device-to-host boundaries $\mathcal{B}$, or dynamically evaluated during partitioning when consecutive nodes span separate hardware:

$$T_{\text{pcie}}(v_i, d_k) = \begin{cases} \frac{\text{Data}(v_i)}{10^6 \times \text{BW}_{\text{PCIE}}} & \text{if } v_i \in \mathcal{B} \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

If the partition boundary crosses separate servers s_k and s_l , inter-server network communication costs $T_{\text{inter-server}}(s_k, s_l)$ are modeled using a hierarchical 5G RAN topology [5]. Wireless transmission delays between the user device and the base station (BS) are defined as $T_{\text{wireless}} = \frac{F^{\text{ul}}}{r_{bs}^{\text{ul}}} + \frac{F^{\text{dl}}}{r_{bs}^{\text{dl}}}$, where F^{ul} and F^{dl} denote upload and download sizes, respectively. Uplink and downlink transmission rates, r_{bs}^{ul} and r_{bs}^{dl} , follow the Shannon capacity formula: $r_{bs}^{\text{ul}} = B \log_2(1 + \frac{p_u g}{\sigma^2})$ and $r_{bs}^{\text{dl}} = B \log_2(1 + \frac{p_{bs} g}{\sigma^2})$, where B is the channel bandwidth, p_u and p_{bs} represent transmission powers, σ^2 is noise power, and the channel gain g is modeled as a Rician random variable to abstract dominant Line-of-Sight (LoS) paths and inherently support user mobility.

Operational energy efficiency (in Joules) is evaluated by multiplying a node's execution runtime by the hardware device's TDP:

$$E_{\text{comp}}(v_i, d_k) = \left(\frac{T_{\text{comp}}(v_i, d_k)}{1000} \right) \times \text{TDP}(d_k) \quad (7)$$

To capture multi-tenant operational realities across a disaggregated 5G edge, a resource contention model adjusts execution profiles based on a device's background utilization rate $U(d_k) \in [0, 1)$ (Step 4). The effective processing capacity $(1 - U(d_k))$ scale adjusts base latencies to mathematically project delayed completion times caused by processor sharing and preemption:

$$T_{\text{adj}}(v_i, d_k) = \frac{T_{\text{comp}}(v_i, d_k)}{1 - U(d_k)} \quad (8)$$

The resource contention rate proportionally scales energy consumption to enforce a conservative worst-case upper bound, accounting for baseline power draw during preempted execution spans:

$$E_{\text{adj}}(v_i, d_k) = E_{\text{comp}}(v_i, d_k) \times \left(\frac{T_{\text{adj}}(v_i, d_k)}{T_{\text{comp}}(v_i, d_k)} \right) \quad (9)$$

Finally, the partitioning engine (Step 5) leverages this comprehensive performance profile to solve the resource allocation task. Our resource-aware model partitioning problem generalizes the classical Multiprocessor Scheduling with Precedence Constraints (MPSPC), a well-known NP-complete challenge. However, since our framework incorporates heterogeneous energy and communication costs, finding optimal schedules for large-scale DAGs is strictly NP-hard. The search space expands exponentially over node-device assignments, resulting in a worst-case computational complexity of $O(|\mathcal{D}|^{|\mathcal{V}|})$. To maintain mathematical tractability, we restrict the formulation exclusively to mixed-integer linear programming (MILP), utilizing the SCIP solver to navigate the search space bounded by five critical constraints: consistency, precedence, global latency, non-parallelism, and memory.

A. Problem Definition

We model resource allocation as a lexicographic two-stage optimization problem. Stage 1 minimizes total energy consumption subject to a global end-to-end latency budget Δ , while Stage 2 shifts to minimizing the overall execution makespan while locking the optimal energy found in Stage 1. Compressing these potential idle periods is crucial to reducing overall inference latency and freeing computational resources earlier, which is particularly beneficial in shared server environments where resource contention may occur.

To formally model this scheduling problem, we define the primary binary decision variable $x_{i,k} \in \{0, 1\}$, where $x_{i,k} = 1$ if node $v_i \in \mathcal{V}$ is assigned to device $d_k \in \mathcal{D}$, alongside a continuous variable $s_i \in \mathbb{R}^+$ denoting a node's execution start time. To minimize energy, we account for both active hardware processing costs ($E_{i,k}^{\text{dyn}}$) and baseline static energy consumption, which is governed by a static power rate (P_k^{sta}) modeled as 10% of the device's TDP and scaled by the node's execution latency ($T_{i,k}$). By incorporating the wireless network energy overhead (E^{wireless}) as a constant offset, Stage 1 is formalized as:

$$\text{minimize} \quad \sum_{v_i \in \mathcal{V}} \sum_{d_k \in \mathcal{D}} x_{i,k} \left(E_{i,k}^{\text{dyn}} + \frac{T_{i,k}}{1000} P_k^{\text{sta}} \right) + E^{\text{wireless}} \quad (10)$$

where $P_k^{\text{sta}} = 0.10 \times \text{TDP}_k$ represents the static power rate of device d_k .

This optimization is bound by five linear constraints. First, the *consistency constraint* ensures each node is allocated to exactly one device:

$$\sum_{d_k \in \mathcal{D}} x_{i,k} = 1, \quad \forall v_i \in \mathcal{V} \quad (11)$$

Second, the *precedence constraint* preserves DAG dataflow dependencies, preventing a child node v_j from executing until its parent v_i completes and inter-device communication $C_{i,j,p,c}$ is resolved, enforced via a Big-M linearization:

$$s_j \geq s_i + T_{i,p} + C_{i,j,p,c} - M \cdot (2 - x_{i,p} - x_{j,c}) \quad (12)$$

where $T_{i,p}$ is the parent execution time on device d_p , M is a large worst-case latency constant, and the communication cost function is defined as:

$$C_{i,j,p,c} = \begin{cases} 0 & \text{if } d_p = d_c \\ \frac{1000 \cdot \text{Data}_i}{\min(\text{BW}_p, \text{BW}_c)} + \mathbb{I}_{p,c}(\mathcal{L}_{\text{path}} \cdot \Delta_{\text{net}}) & \text{if } d_p \neq d_c \end{cases} \quad (13)$$

Here, Data_i is the parent's output tensor size, BW_k is device PCIe bandwidth, and $\mathbb{I}_{p,c}$ is an indicator variable applying a network latency expansion penalty (Δ_{net}) over shortest-path distances ($\mathcal{L}_{\text{path}}$) if devices reside on distinct servers.

Third, to guarantee that the distributed inference meets strict performance deadlines, we establish a *global latency constraint*. The DAG's total makespan $\mathcal{M}$ must not exceed the application latency requirement L_{req} . We define the makespan as the maximum completion time across all nodes, which includes a node's start time, its execution latency on the selected device, and the network delay required to transmit the final results back to the base station (BS):

$$\mathcal{M} \geq s_i + \sum_{d_k \in \mathcal{D}} x_{i,k} \cdot (T_{i,k} + R_k), \quad \forall v_i \in \mathcal{V} \quad \mathcal{M} \leq L_{\text{req}} \quad (14)$$

where $T_{i,k}$ is the processing time of node v_i on device d_k , and R_k represents the routing delay to the base station. Similar to the inter-device communication penalty, this return delay is calculated by scaling the shortest-path distance from the selected device to the base station by the network latency expansion factor: $R_k = \mathcal{L}_{\text{path}}(k, \text{BS}) \cdot \Delta_{\text{net}}$.

Fourth, to enforce the sequential execution of independent nodes assigned to the same device, we introduce a *non-parallelism constraint*, which dictates that topologically independent nodes sharing a device must not overlap. Let $\mathcal{I}$ represent the set of all independent node pairs (v_i, v_j) in the DAG. For every $(v_i, v_j) \in \mathcal{I}$ and device $d_k \in \mathcal{D}$, we introduce a binary decision variable $b_{i,j,k} \in \{0, 1\}$ to determine their

execution order. We apply a Big-M disjunctive formulation to prevent simultaneous processing:

$$s_i + T_{i,k} + \epsilon \leq s_j + M \cdot (3 - x_{i,k} - x_{j,k} - b_{i,j,k}) \quad (15)$$

$$s_j + T_{j,k} + \epsilon \leq s_i + M \cdot (2 - x_{i,k} - x_{j,k} + b_{i,j,k}) \quad (16)$$

where $T_{i,k}$ and $T_{j,k}$ are the execution latencies of nodes v_i and v_j on device d_k , ϵ is a sufficiently small constant to ensure strict temporal separation, and M remains the worst-case latency constant defined previously.

These constraints activate only when both nodes share the same device ($x_{i,k} = x_{j,k} = 1$). Under this condition, $b_{i,j,k}$ forces a strict sequential order: if $b_{i,j,k} = 1$, Eq. 15 is enforced (v_i executes before v_j) and Eq. 16 is relaxed. Conversely, if $b_{i,j,k} = 0$, Eq. 16 is enforced (v_j before v_i) and Eq. 15 is relaxed. If placed on different devices, both equations are safely relaxed.

Fifth, to guarantee that every device operates within its physical hardware limits, we introduce a *memory constraint*. For every device $d_k \in \mathcal{D}$, the total memory footprint (combining local processing footprint and incoming tensor buffering) must not exceed its maximum capacity W_k :

$$\sum_{v_i \in \mathcal{V}} x_{i,k} \cdot \mu_i + \sum_{(v_i, v_j) \in \mathcal{E}} m_{i,j,k} \cdot \mu_i \leq W_k \quad (17)$$

where μ_i represents the memory demand of node v_i . The first summation calculates the inherent memory footprint of nodes assigned to d_k . The second summation uses a binary auxiliary variable $m_{i,j,k} \in \{0, 1\}$, which equals 1 if child node v_j is assigned to d_k while its parent v_i is executed on a different device, requiring v_i 's output tensor to be buffered on d_k . To preserve linearity, this condition is enforced for every $(v_i, v_j) \in \mathcal{E}$ and $d_k \in \mathcal{D}$ using the following linearization constraints:

$$m_{i,j,k} \leq x_{j,k} \quad (18)$$

$$m_{i,j,k} \leq 1 - x_{i,k} \quad (19)$$

$$m_{i,j,k} \geq x_{j,k} - x_{i,k} \quad (20)$$

While Stage 1 provides minimum energy-cost assignments, the resulting node allocation may introduce idle gaps that prolong overall inference latency. To resolve this, Stage 2 compresses the schedule by minimizing the makespan $C_{\text{max}} \in \mathbb{R}^+$ while converting the optimal Stage 1 energy (E^* , from Eq. 10) into a strict equality constraint. The Stage 2 optimization problem is formulated as follows:

$$\text{minimize} \quad C_{\text{max}} \quad (21)$$

$$\text{subject to} \quad C_{\text{max}} \geq s_i + \sum_{d_k \in \mathcal{D}} x_{i,k} T_{i,k}, \quad \forall v_i \in \mathcal{V} \quad (22)$$

$$\sum_{v_i \in \mathcal{V}} \sum_{d_k \in \mathcal{D}} x_{i,k} \left(E_{i,k}^{\text{dyn}} + \frac{T_{i,k}}{1000} P_k^{\text{sta}} \right) + E^{\text{wireless}} = E^* \quad (23)$$

Equations (11)-(20).

where Eq. 22 bounds C_{max} by the completion time of each node $v_i \in \mathcal{V}$ (encompassing its start time s_i plus its execution

latency $T_{i,k}$), and Eq. 23 acts as the lexicographic anchor, restricting the solver to the energy-optimal baseline E^* . Finally, the five constraints established in Stage 1 remain fully active to guarantee a viable schedule in Stage 2.

IV. EXPERIMENTAL EVALUATION

We evaluated SPARX within a MAR use case. As a prominent user-centric application, MAR perfectly highlights the conflict between computing resource limitations and stringent application demands; it relies on compute-intensive DL models for object detection while requiring an end-to-end latency budget of 20 ms to guarantee an optimal Quality of Experience (QoE) [2], [22], [23]. To validate SPARX’s scheduling capabilities, we adopted the nano and medium variants of the state-of-the-art YOLO11 object detection model. Evaluating the transition between these variants provides clear intuition on how SPARX scales with increasing computational complexity and memory footprints, as the underlying partitioning dynamics remain consistent for larger architectures. To partition these models within SPARX’s engine, we investigated two distinct paradigms: the multi-stage exact solver defined in the previous section (bounded by a 24-hour runtime or a relative optimality gap less than 1%) to establish an optimal theoretical baseline, and lightweight heuristic alternatives to reduce solution time. Specifically, we integrated the classical Heterogeneous Earliest Finish Time (heft) DAG scheduling algorithm [24] and its Look-Ahead variant (heft_la) [25], alongside greedy constructive heuristics configured to minimize energy consumption (heu_es) and node completion times (heu_ct).

We modeled our infrastructure in accordance with the disaggregated RAN topology introduced in Section III. This architecture comprises three distinct servers, each provisioned with a dedicated computing unit: an NVIDIA A30 (NV-A30) GPU, an NVIDIA BlueField 2 (NV-BF2) DPU, and an Alveo U55C FPGA configured with two Deep Learning Processing Units (AL-U2). To represent a realistic shared infrastructure (Step 4, Fig. 1), we restricted the GPU to 30% of its total computational capacity while leaving the DPU and FPGA fully available. We considered this utilization rate based on the assumption that in a shared infrastructure, the device with the highest computational capacity would be the one most contested by applications.

The proposed framework was implemented in Python 3.12. The optimization problems were modeled using Google OR-Tools and solved with the SCIP backend. The DAG profiling pipeline was implemented using ONNX, ONNX Runtime, and onnx-tool. The complete source code is publicly available in a GitHub repository.¹

A. SPARX Evaluation

We first evaluated how operation fusion affected the total energy consumption and the solution time when partitioning the YOLO11-nano model. Table I reports the execution times and relative optimality gaps of all partitioning approaches, revealing that exact optimization is computationally prohibitive.

For both the fused and non-fused model variants, the exact solver exhausted the standard 24-hour runtime limit (approx. 86,400 seconds) in Stage 1 without reaching the 1% optimality gap. In Stage 2, the fused version similarly timed out at the 24-hour threshold (86,401.3 seconds). For the non-fused DAG, we extended the timeout budget to over 28 hours (101,781.6 seconds) to observe whether additional computing time would allow the exact solver to surpass the heuristic solutions. Despite this extension, it still failed to converge to a provably optimal bound. In contrast, all evaluated heuristic alternatives successfully navigated the search space and returned feasible deployment strategies within milliseconds (rendered as 0.01–0.03 seconds).

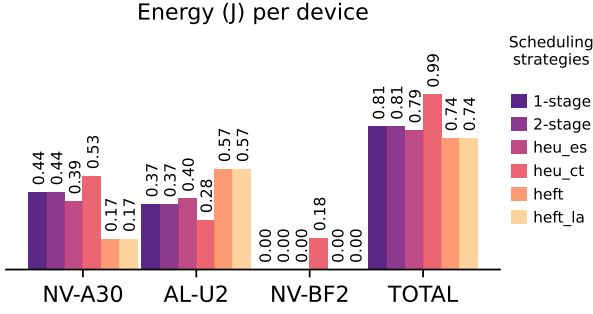
TABLE I: Solver Execution Times and Optimality Gaps

Configuration / Variant	Time (s)	Gap (%)
<i>Stage 1: Energy Optimization</i>		
Non-fused	86457	2.61
Fused	86412	3.89
Fused + heft_la	20.43	0.32
<i>Stage 2: Makespan Compression</i>		
Non-fused	101781	300.55
Fused	86401	299.86
Fused + heft_la	86413	215.22
<i>Heuristics</i>		
heu_es (non-fused)	0.01	–
heu_es (fused)	0.01	–
heu_ct (non-fused)	0.01	–
heu_ct (fused)	0.02	–
heft (non-fused)	0.01	–
heft (fused)	0.02	–
heft_la (non-fused)	0.01	–
heft_la (fused)	0.03	–

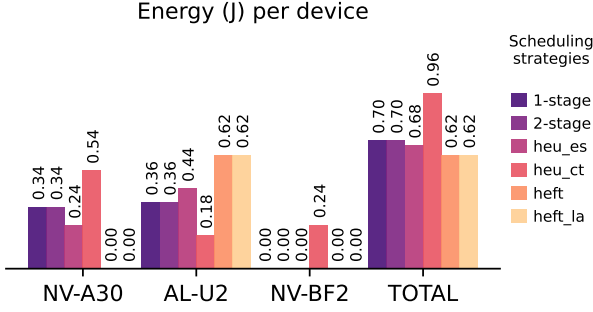
As a direct consequence of the exact solver’s premature termination in Stage 1, it was forced to output its best-found feasible energy solution, yielding globally suboptimal profiles compared to the heuristic baselines (Fig. 2). However, operation fusion still provided a measurable advantage: by reducing the DAG’s topological complexity, it allowed the solver to explore the search space more effectively and secure a significantly better energy footprint (0.70 J) within the 24-hour limit compared to the non-fused variant. Despite this improvement, the lightweight heuristics still prevailed. Under the fused configuration, heft and heft_la bypassed this computational bottleneck entirely, achieving a minimal footprint of 0.62 J—an 11.4% improvement over the exact solver’s best-effort fused solution. Conversely, heu_ct consumed the most energy across both variants (0.99 J non-fused, 0.96 J fused) owing to its aggressive selection of high-capacity, power-intensive accelerators to minimize execution delay. Overall, these results demonstrate that while operation fusion aids exact optimization, lightweight heuristics provide vastly superior energy efficiency when strict runtime limits are involved.

Fig. 3 illustrates how the computational workload is dis-

¹<https://github.com/LABORA-INF-UFG/DAG-PART>

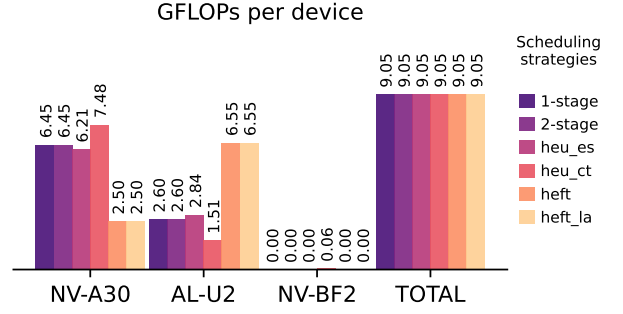


(a) Non-fused nodes.

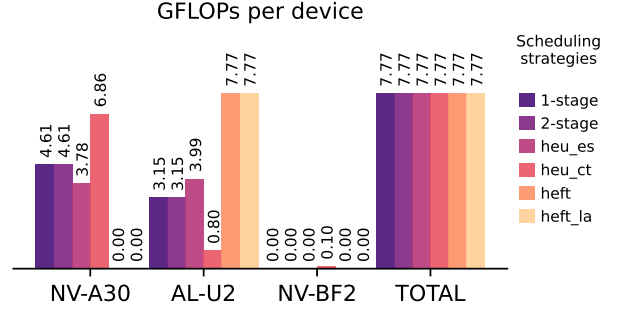


(b) Fused nodes.

Fig. 2: Estimated energy footprint of YOLO11-nano.



(a) Non-fused nodes.

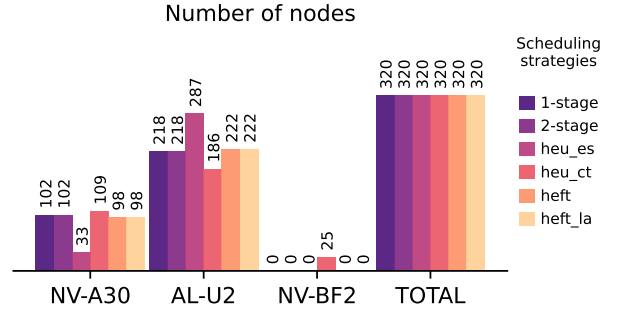


(b) Fused nodes.

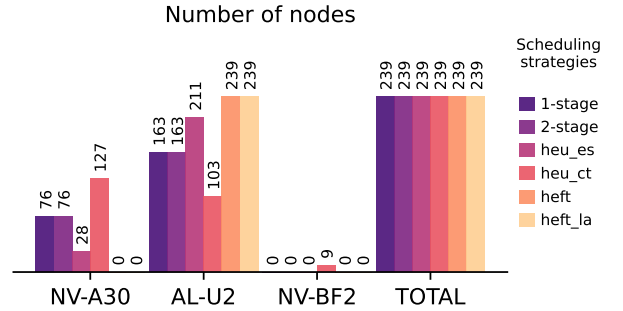
Fig. 3: Estimated computational footprint of YOLO11-nano.

tributed across the heterogeneous devices for each scheduling strategy. The exact solver and the energy-oriented heuristic (heu_es) balance the workload between the GPU and FPGA to minimize energy consumption while satisfying the latency constraint. In contrast, heu_ct prioritizes the highest-performance device, assigning most computations to the GPU, which explains its higher energy consumption. Conversely, heft and heft_la allocate nearly the entire workload to the FPGA. Although the FPGA provides lower computational throughput than the GPU, this mapping eliminates inter-device communication overhead, resulting in a more energy-efficient schedule. The benefits of operation fusion are also evident, as the total workload decreases from 9.05 GFLOPs to 7.77 GFLOPs, directly reducing the computational demand across all scheduling strategies.

The number of nodes assigned to each device is shown in Fig. 4. For YOLO11-nano (without operation fusion), the exact formulations (1-stage and 2-stage) and the energy-oriented heuristic (heu_es) distribute the workload between the NV-A30 GPU (102 nodes) and the AL-U2 FPGA (218 nodes). Conversely, the completion-time-oriented heuristic (heu_ct) leverages all available resources to minimize latency, allocating nodes across the GPU (109), FPGA (186), and DPU (25). Under this same configuration, the list-based methods (heft and heft_la) split the workload by assigning 98 nodes to the GPU and 222 to the FPGA. With operation fusion, the DAG complexity drops from 320 to 239 nodes. In this context, the exact formulations and heu_es exhibit a similar offloading pattern (GPU: 76, FPGA: 163). Meanwhile, heu_ct prioritizes the high-capacity GPU (127 nodes) and reduces DPU utilization



(a) Non-fused nodes.



(b) Fused nodes.

Fig. 4: Node distribution of YOLO11-nano.

to just 9 nodes to optimize end-to-end execution.

As shown in Fig. 5, greedy heuristics exhibit significant limitations: they make allocation decisions based solely on immediate task completion times, ignoring inter-task dependencies and communication overhead. Although computationally effi-

cient, this local perspective often produces globally suboptimal schedules in heterogeneous distributed environments, resulting in makespans of 44.0 ms and 56.5 ms for the non-fused model, and 36.3 ms and 39.7 ms after operation fusion. While fusion reduces the overall computational workload and consequently improves latency, the resulting schedules still fail to satisfy the 20 ms application constraint.

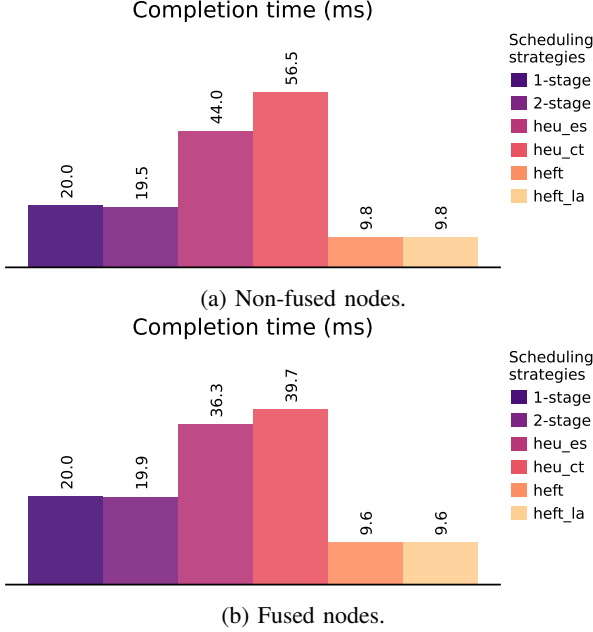


Fig. 5: Estimated makespan of YOLO11-nano inference.

The heft algorithm alleviates this limitation by prioritizing tasks according to their position on the DAG critical path before assigning them to devices, leading to more balanced schedules with lower overall makespan. Furthermore, heft_la extends this strategy through a look-ahead mechanism that evaluates the impact of each allocation on successor tasks, reducing future communication overhead and improving end-to-end execution time. As a result, both heuristics achieve the lowest completion times, approximately 9.6–9.8 ms, significantly outperforming both the greedy approaches and the exact optimization model. Interestingly, the exact solver remains close to the 20 ms latency threshold because its first stage prioritizes energy minimization and the second stage is constrained to preserve the same energy-optimal solution, limiting its ability to further reduce the makespan.

B. Hybrid Approach Evaluation

To mitigate this computational bottleneck, we evaluated a hybrid approach by introducing the solution from heft_la as a warm start to the exact solver (considering the fused DAG variant). For Stage 1, this approach allowed the solver to prune the search space so aggressively that the relative optimality gap reduced from **3.89%** to **0.32%**. Consequently, Stage 1 successfully achieved early convergence, reducing its execution time from **24 hours** to **20 seconds**. This tightened optimality gap directly affected the total energy consumption,

which decreased to 0.68 J, matching heu_es and drawing closer to the 0.62 J bound discovered by heft and heft_la (Fig. 6).

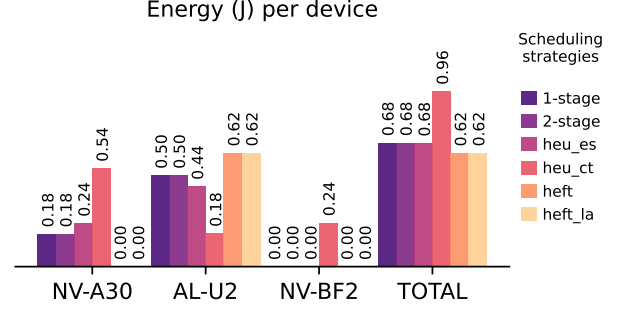


Fig. 6: Estimated energy footprint of YOLO11-nano with fused nodes and heft_la as a warm start.

Conversely, Stage 2 still exhausted its full 24-hour runtime allocation. This behavior stems from the sequential dependencies in our multi-stage optimization strategy; Stage 2 uses the Stage 1 solution as a strict baseline and operates under a hard constraint that forbids violating the energy footprint established in the first stage. Despite hitting the timeout threshold, the warm start fundamentally improved the solver’s exploration of the remaining search space, as evidenced by the Stage 2 optimality gap dropping significantly from 299.86% to 215.22%.

We next evaluated SPARX’s behavior when partitioning the larger YOLO11-medium model. Under the initial 70% GPU contention rate, the increased computational demands of the medium variant rendered the problem infeasible; neither the exact solver nor the heuristics could identify a viable deployment strategy. To establish a feasible search space, we reduced the contention rate to 35%, thereby increasing the available GPU capacity to 65%.

In this context, total energy consumption exhibited a behavior similar to YOLO11-nano: the heft and heft_la heuristics achieved the highest energy efficiency at 2.45 J, and heu_ct again consumed the most energy (3.27 J). A notable distinction from the YOLO11-nano results was that the exact solver and heu_es performed similarly, showing energy footprints of 2.69 J and 2.67 J, respectively.

Despite the increased complexity of YOLO11-medium, Stage 1 achieved native early convergence in just 1.8 seconds. This rapid termination occurred because the solver successfully reached a relative optimality gap of 0.82%, satisfying the threshold of less than 1% without requiring a heuristic warm start. As expected, the lightweight heuristics continued to find solutions within milliseconds. However, consistent with our previous observations, Stage 2 still exhausted the full 24-hour runtime limit due to the strict energy boundary imposed by the Stage 1 baseline.

The results demonstrate that relying solely on exact optimization approaches is inherently impractical due to the structural complexity of modern DNNs. Although introducing

a heuristic as a warm start successfully mitigated this computational bottleneck, the exact solver still failed to find a provably optimal solution. Increasing computational intractability due to model complexity suggests future work on approximation algorithms for resource-aware model partitioning.

V. CONCLUSION

In this paper, we introduced SPARX, a resource-aware model partitioning framework for distributed inference across the heterogeneous edge. SPARX addresses key gaps in existing approaches by abstracting both AI models and diverse edge hardware into a unified analytical performance profile, thereby avoiding the scalability bottleneck of exhaustive physical benchmarking. By capturing runtime-level optimizations such as operation fusion, SPARX generates deployment plans optimized for both energy consumption and makespan while guaranteeing strict latency SLAs.

To evaluate this approach, we employed a two-stage MILP formulation alongside lightweight heuristics to partition two variants of YOLO11 (nano and medium) over a disaggregated 5G RAN topology model. Our evaluation showed that explicitly modeling operation fusion enables deployment plans with reduced estimated energy consumption across all paradigms. Furthermore, partitioning modern DNNs using exact optimization proved impractical due to the structural complexity of these models; even when using a heuristic warm start, the solver failed to reach a provably optimal solution. In contrast, lightweight heuristics consistently produced feasible, energy-efficient deployment plans within milliseconds. Consequently, as the complexity of AI models scales, such computational bottlenecks will intensify, motivating future work on advanced approximation algorithms for resource-aware model partitioning at the modern heterogeneous edge.

ACKNOWLEDGMENTS

This work was supported by Ericsson Telecomunicações Ltda., and by the Sao Paulo Research Foundation (FAPESP), grant 2021/00199-8, CPE SMARTNESS. This research was partially funded by CAPES, Brazil - Finance Code 001.

REFERENCES

- [1] Next G Alliance, "AI-Native Wireless Networks," Alliance for Telecommunications Industry Solutions (ATIS), Tech. Rep., 2023, available at: https://nextgalliance.org/white_papers/ai-native-wireless-networks/ Accessed: 2026-01-11.
- [2] K. Chuang *et al.*, "From Dusk to Dawn: An Industry Perspective on the Transition From 5G to 6G Wireless Communications Infrastructure," *IEEE Microw. Mag.*, vol. 26, no. 12, pp. 44–63, 2025, doi:10.1109/MMM.2025.3602738.
- [3] C. Zheng *et al.*, "In-network machine learning using programmable network devices: A survey," *IEEE Commun. Surv. Tutor.*, vol. 26, no. 2, pp. 1171–1200, 2024.
- [4] M. M. H. Shuvo *et al.*, "Efficient acceleration of deep learning inference on resource-constrained edge devices: A review," *Proc. IEEE*, vol. 111, no. 1, pp. 42–91, 2023, doi:10.1109/JPROC.2022.3226481.
- [5] PASSION Project, "Photonic technologies for programmable transmission and switching modular systems based on Scalable Spectrum/space aggregation for future agile high capacity metro Networks," 2020, available at <http://www.passion-project.eu>. Accessed 05/07/2026.
- [6] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems*. Xi'an, China: ACM, 2017, p. 615–629, doi:10.1145/3093337.3037698.
- [7] P. Ren *et al.*, "Edge-assisted distributed dnn collaborative computing approach for mobile web augmented reality in 5g networks," *IEEE Network*, vol. 34, no. 2, pp. 254–261, 2020, doi:10.1109/MNET.011.1900305.
- [8] J. Mao *et al.*, "Modnn: Local distributed mobile computing system for deep neural network," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017. Lausanne, Switzerland: IEEE, 2017, pp. 1396–1401, doi:10.23919/DATE.2017.7927211.
- [9] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2348–2359, 2018, doi:10.1109/TCAD.2018.2858384.
- [10] L. Zeng *et al.*, "CoEdge: Cooperative DNN Inference With Adaptive Workload Partitioning Over Heterogeneous Edge Devices," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 595–608, 2021.
- [11] C. Hu and B. Li, "Distributed Inference with Deep Learning Models across Heterogeneous Edge Devices," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. London, United Kingdom: IEEE, 2022, pp. 330–339.
- [12] X. Hou *et al.*, "DistrEdge: Speeding up Convolutional Neural Network Inference on Distributed Edge Devices," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Los Alamitos, CA, USA: IEEE, 2022, pp. 1097–1107, doi:10.1109/IPDPS53621.2022.00110.
- [13] X. Zheng *et al.*, "Adaptive dnn partitioning strategy for optimized user fitness in edge computing networks," *IEEE Trans. Consum. Electron.*, vol. 72, no. 2, pp. 3722–3731, 2026, doi:10.1109/TCE.2026.3663558.
- [14] S. Kumar *et al.*, "Simulating llm training workloads for heterogeneous compute and network infrastructure," in *Proceedings of the 2nd Workshop on Networks for AI Computing*. Coimbra, Portugal: ACM, 2025, p. 105–107, doi:10.1145/3748273.3749212.
- [15] S. Ko *et al.*, "Dfmodel: Design space optimization of large-scale systems exploiting dataflow mappings," *arXiv preprint arXiv:2412.16432*, pp. 1–22, 2024.
- [16] S. Lee *et al.*, "Forecasting GPU Performance for Deep Learning Training and Inference," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. Rotterdam, Netherlands: ACM, 2025, pp. 493–508.
- [17] G. Sun *et al.*, "Toward ai-native task orchestration for collaborative computing in sagsins," *IEEE Commun. Mag.*, vol. 63, no. 12, pp. 112–118, 2025, doi:10.1109/MCOM.004.2400304.
- [18] ONNX, "Open Neural Network Exchange (ONNX)," <https://github.com/onnx/onnx>, 2024, version: 1.16.2.
- [19] P. Henderson *et al.*, "Towards the systematic reporting of the energy and carbon footprints of machine learning," *J. Mach. Learn. Res.*, vol. 21, no. 1, pp. 1–44, 2020, doi:10.5555/3455716.3455964.
- [20] T. Shinji, "onnx-tool: A tool for parsing, editing, and profiling onnx models," <https://github.com/ThanatosShinji/onnx-tool>, 2022, accessed: July 2026.
- [21] Microsoft, "ONNX Runtime Graph Optimizations," <https://onnxruntime.ai/docs/performance/model-optimizations/graph-optimizations.html>, 2026, accessed: July 2026.
- [22] J.-P. Stauffert *et al.*, "Latency and Cybersickness: Impact, Causes, and Measures. A Review," *Front. Virtual Real.*, vol. 1, no. 582204, pp. 1–10, 2020.
- [23] Y. Mao *et al.*, "A Survey on Mobile Edge Computing: The Communication Perspective," *IEEE Commun. Surv. Tutor.*, vol. 19, no. 4, pp. 2322–2358, 2017.
- [24] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, 2002, doi:10.1109/71.993206.
- [25] L. F. Bittencourt, R. Sakellariou, and E. R. M. Madeira, "Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm," in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, 2010, pp. 27–34, doi:10.1109/PDP.2010.56.