DPUConfig: Optimizing ML Inference in FPGAs Using Reinforcement Learning

Alexandros Patras, Spyros Lalis, Christos D. Antonopoulos, and Nikolaos Bellas
Department of Electrical and Computer Engineering, University of Thessaly, Volos, Greece
{patras, lalis, cda, nbellas}@uth.gr

***Abstract*—Heterogeneous embedded systems, with diverse computing elements and accelerators such as FPGAs, offer a promising platform for fast and flexible ML inference, which is crucial for services such as autonomous driving and augmented reality, where delays can be costly. However, efficiently allocating computational resources for deep learning applications in FPGA-based systems is a challenging task. A Deep Learning Processor Unit (DPU) is a parameterizable FPGA-based accelerator module optimized for ML inference. It supports a wide range of ML models and can be instantiated multiple times within a single FPGA to enable concurrent execution. This paper introduces DPUConfig, a novel runtime management framework, based on a custom Reinforcement Learning (RL) agent, that dynamically selects optimal DPU configurations by leveraging real-time telemetry data monitoring, system utilization, power consumption, and application performance to inform its configuration selection decisions. The experimental evaluation demonstrates that the RL agent achieves an energy efficiency that is 95% (on average) of the optimal attainable energy efficiency for several CNN models on the Xilinx Zynq UltraScale+ MPSoC ZCU102.**

***Index Terms*—FPGA, adaptive, configuration, reinforcement learning, power efficiency.**

I. Introduction

Recent ML advances have accelerated the deployment of deep learning inference on heterogeneous platforms. While CPUs and GPUs remain standard, FPGA-based MPSoCs have emerged as efficient alternatives due to their flexibility and energy efficiency. Among numerous accelerator designs [1], [2], Deep Learning Processor Units (DPUs) [3] offer a particularly promising platform.

A key challenge in leveraging FPGA-based MPSoCs for DL inference lies in workload variability and diverse configuration options. Unlike CPU scheduling—focused on core allocation to balance latency and power—FPGA platforms offer additional degrees of freedom, such as selecting DPU configurations and hardware parameters tailored to specific tasks. These choices significantly impact both performance and energy efficiency.

To address these challenges, this work proposes an adaptive scheduling framework based on Reinforcement Learning (RL), which integrates real-time telemetry data, such as CPU and memory system utilization, power dissipation, and application performance, into its decision-making process. By harnessing a custom RL agent, our system dynamically determines the optimal task-to-hardware mapping, choosing the most suitable DPU configuration for each incoming inference task. This enables the runtime system to balance the trade-offs between computation latency and power dissipation while accommodating the unique constraints imposed by FPGA architectures.

This paper makes the following key contributions:

- We introduce DPUConfig, a custom RL-based run-time system that, under stochastic variability and user constraints, consistently chooses a DPU configuration for ML inference, which is very close to the optimal one. Although prior work has applied RL-based resource allocation or configuration to optimize specific metrics, this is the first work to apply those methods to reconfigurable DPUs.
- We implement DPUConfig on the ZCU102 device and evaluate it against varied ML workloads. The analysis demonstrates that the agent's optimization benefits significantly outweigh the costs of runtime reconfiguration. This demonstrates the feasibility and practicality of our approach.

II. Deep Learning Processor Units (DPUs)

Deep Learning Processor Units (DPU) [3], released as part of the Xilinx Vitis-AI toolchain [4], have become a popular solution to deploy pre-trained ML models on FPGA devices. DPUs are programmable, featuring a CISC-style instruction set capable of supporting inference for a wide range of CNN models. DPUs are used in an increasingly large number of applications such as autonomous driving [5], object detection [6], [7], thermal imaging [8], and even semantic segmentation for space applications [9].

Multiple instances of DPUs can be used to run independent ML inferences concurrently (Table I). For example, due to resource constraints on the ZCU102 MPSoC, only up to three large B4096 DPUs can be instantiated to execute three independent ML models. Although the system supports the concurrent execution of heterogeneous CNN models, they are constrained to a unified DPU configuration, as the Vitis AI runtime precludes the simultaneous use of differing configurations. The B4096 configuration has Pixel Parallelism (PP) = 8 and Input and Output Channel Parallelism (ICP) = (OCP) = 16. Hence, the peak performance is 2048 MAC operations

The research is conducted in the operating framework of the University of Thessaly Innovation, Technology Transfer Unit and Entrepreneurship Center "One Planet Thessaly", under the "University of Thessaly Grants for Scientific Publication Support" action and is funded by the Special Account of Research Grants of the University of Thessaly. It is also funded by the European Union's Horizon Europe Programme under the "MLSysOps" Project (Grant Agreement No. 101092912).

per cycle [3]. Each MAC operation counts for two regular operations.

For preparing the necessary artifacts, the Vitis AI framework reads the pre-trained ONNX [10] or PyTorch [11] representation of the ML model, optionally performs pruning and INT8 quantization, and then compiles the quantized model for a target DPU architecture using the Vitis AI compiler [4]. To exploit parallelism within the layers of the ML model, the compiler performs multiple optimizations, such as layer fusion and instruction scheduling. Once execution starts, the DPU fetches its instructions from DDR memory, while on-chip BRAM/DRAM buffers store the inputs, outputs, and intermediate data to reduce latency and minimize external memory bandwidth usage. The DPUs are invoked by the host CPU and execute the CNNs layer by layer.

TABLE I: Name, maximum number of DPU instances, and the selected configurations used in the action space of the RL agent, based on the DPUCZDX8G IP [3] for the Zynq UltraScale+ MPSoC [12]. For example, we consider 4 instances in B1600.

DPU configuration (PP*ICP*OCP)	Max. instances	Notation	Selected Configurations
B512 (4*8*8)	8	B512_8	B512_{1,4,8}
B800 (4*10*10)	7	B800_7	B800_{1,4,7}
B1024 (8*8*8)	6	B1024_6	B1024_{1,3,6}
B1152 (4*12*12)	6	B1152_6	B1152_{1,3,6}
B1600 (8*10*10)	4	B1600_4	B1600_{1,2,3,4}
B2304 (8*12*12)	4	B2304_4	B2304_{1,2,3,4}
B3136 (8*14*14)	3	B3136_3	B3136_{1,2,3}
B4096 (8*16*16)	3	B4096_3	B4096_{1,2,3}

III. Motivation

This section presents results of system characterization for realistic deep neural network (DNN) inference scenarios using DPU acceleration and the Vitis AI framework on the Xilinx Zynq Ultrascale+ FPGA board (ZCU102) [12]. Our analysis explores the design space along three key dimensions, namely latency, accuracy, and energy efficiency (measured as performance per watt, PPW)–calculated as inference throughput (FPS) divided by the average power consumption. This helps identify the most critical features that determine the optimal DPU configuration for efficient ML inference.

A. The optimal DPU configuration depends on the characteristics of the ML model.

Fig. 1 shows the energy efficiency (PPW) and the performance (in Frames Per Second) of two representative ML inference use cases over different DPU configurations when a single ML model type runs on the DPUs. Considering only DPU configurations that deliver at least 30 fps, we observe that the optimal DPU configuration depends on the characteristics of the model. For example, with ResNet152 [13], the B4096_1 configuration offers the highest energy efficiency; meanwhile, for MobileNetV2 [14], B2304_2 yields the highest efficiency.

Even if larger DPU size configurations provide the highest throughput and energy efficiency for many ML models, this is not the case for smaller models with lower arithmetic intensity and lower DPU utilization, such as MobileNetV2 (see Table III). Such models do not utilize all the resources of a larger DPU (DPU utilization is only 17.1%), but they seem to benefit from more instances of smaller DPUs. The MobileNetV2 performance in B4096_1 is only 2.6x higher than in B512_1. In contrast, ResNet152, being more compute-bound, achieves a much larger 5.8x speedup under the same settings.

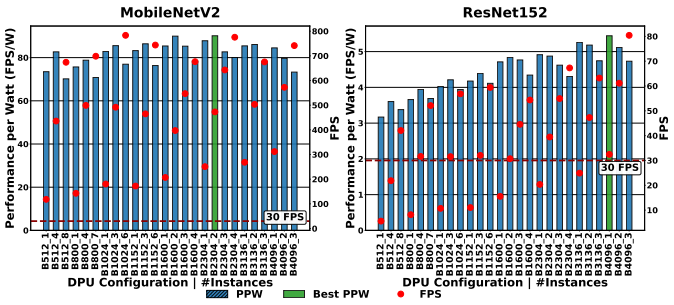

Fig. 1: The optimal execution target depends on ML characteristics. The bars (left axis) show energy efficiency in FPS per Watt, and the red points (right axis) indicate performance.

B. CPU interference from co-executing applications may alter the optimal DPU configuration.

To study concurrent workload effects, we introduced artificial CPU- and memory-intensive tasks, defining three system states: None (N), Compute-heavy (C), and Memory-heavy (M).

Fig. 2 shows that to meet 30 FPS for MobileNetV2, the energy-optimal configuration shifts from B2304_2 in state N to the smaller B1600_2 in states C and M. High memory utilization by competing workloads causes larger DPUs to stall, degrading PPW, whereas smaller DPUs utilize limited bandwidth more efficiently. Additionally, smaller models require frequent CPU coordination, increasing susceptibility to CPU contention. ResNet152 exhibits a similar trend, favoring a smaller DPU (B3136_2) under memory pressure (state M).

C. The optimal DPU configuration varies with inference accuracy requirements.

Vitis AI employs channel pruning, which removes entire channels/filters in convolutional layers [15]. This reduces model size and increases performance, but at the cost of reduced accuracy. Fig. 3 presents the energy efficiency for three versions of ResNet152, corresponding to pruning ratios of 0%, 25%, and 50%. For an accuracy threshold of 60%, ResNet152 can be pruned by 25% to radically improve energy efficiency compared to the original model using a different DPU configuration (B3136_1 instead of B4096_1). Thus, the availability of differently pruned model variants, combined with a specific accuracy target, complicates dynamic selection of DPU configurations to maximize PPW while meeting accuracy requirements.

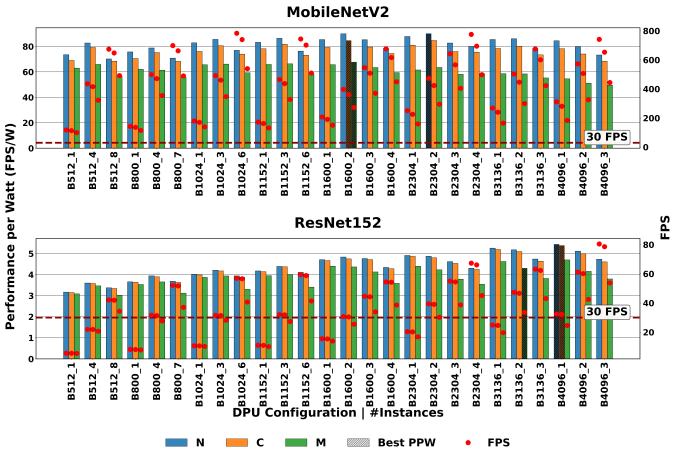

Fig. 2: PPW (left axis, bars) and performance in FPS (right axis, points) across different DPU configurations under three system states. The dark bars highlight the configuration achieving the best energy efficiency while maintaining performance above 30 FPS.

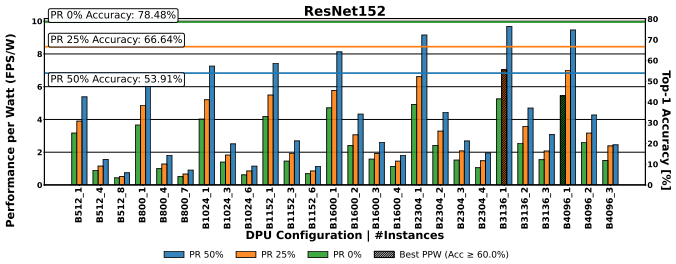

Fig. 3: PPW (left axis, bars) and accuracy (right axis, lines) across different DPU configurations under the N state. For example, the accuracy of ResNet152 when 25% of its channels are eliminated is 66.64%.

IV. Design of the DPUConfig Framework

Fig. 4 shows the overview of the DPUConfig framework. The core component of DPUConfig is the RL agent, a machine learning approach focused on learning a sequence of decisions to maximize cumulative rewards [16]. The primary objective of the RL agent is to determine an optimal DPU configuration under specific constraints. The DPUConfig framework is deployed and executed directly on the Arm CPU of the FPGA platform, and is invoked upon each new ML inference request (i.e., workload arrival). DPUConfig observes the state S of the system, including the static characteristics of the ML model and the runtime metrics from telemetry monitoring, as shown in Table II. Then, it uses the RL agent to select the next action A that will maximize the energy efficiency of the platform while satisfying the latency constraints set by the user. Action A selects the next DPU configuration (size and number of compute units). DPUConfig then dynamically reconfigures the FPGA with the new DPU configuration, loads the DPU instructions for the ML model, and executes the inference on the new configuration. Based on the measured execution metrics (fps and power), DPUConfig computes the reward function, which evaluates how action A improves the energy efficiency under the latency constraints.

A. Reinforcement Learning Agent

This section defines the three core components that formulate the optimization space of DPUConfig and the training procedure.

State. Based on the analysis of Section III, Table II summarizes the state features that capture both the runtime system status and the model characteristics, which together form the input to the RL agent. The *dynamic system features* include per-core CPU utilization, the usage of memory bandwidth across read and write memory ports, and the power consumption of the FPGA and CPU subsystems. These metrics are collected periodically at runtime.

The *model static features* describe the ML model itself, including the number of MAC operations, the memory load/store requirements, and the total number of parameters (shown in Table III). These static features are known *a priori* and are collected once when a new model is introduced into DPUConfig. Finally, each model specifies its own performance constraints.

TABLE II: State features.

State	Description
Dynamic System Features	
CPU_i	Utilization of CPU core $i \in \{0, 1, 2, 3\}$
$MEMR_j$	Memory read bandwidth (MB/s) of port $j \in \{0, \ldots, 4\}$
$MEMW_j$	Memory write bandwidth (MB/s) of port $j \in \{0, \ldots, 4\}$
P_{FPGA}	Power consumption (W) of FPGA fabric
P_{ARM}	Power consumption (W) of CPU cores
Model Static Features	
GMAC	Number of MAC operations (GMACs)
LDFM	Model load from memory (bytes)
LDWB	Model load from weight buffer (bytes)
STFM	Model store to memory (bytes)
PARAM	Number of trainable model parameters
Constraints	
C_PERF	FPS Performance constraint

Actions. The available actions correspond to 26 distinct DPU configurations, combining different DPU sizes and numbers of instances, as listed in Table I. The action space does not cover the entire design space, as certain intermediate configurations (e.g., B512_2, B800_6) were omitted. These configurations were excluded based on empirical analysis, which showed that they do not provide meaningful variation in training and are never part of the optimal configuration set.

Reward. Rewards provide the feedback that guides the RL agent toward the desired goal. Each action must contribute to achieving a specific target. In our setting, however, the optimization target is inherently dynamic: the achievable performance-per-watt (PPW) depends on both the current system workload and the ML model characteristics.

This context dependency introduces two challenges. First, there is no global reward target, as an action that maximizes PPW under one workload state may be suboptimal in another.

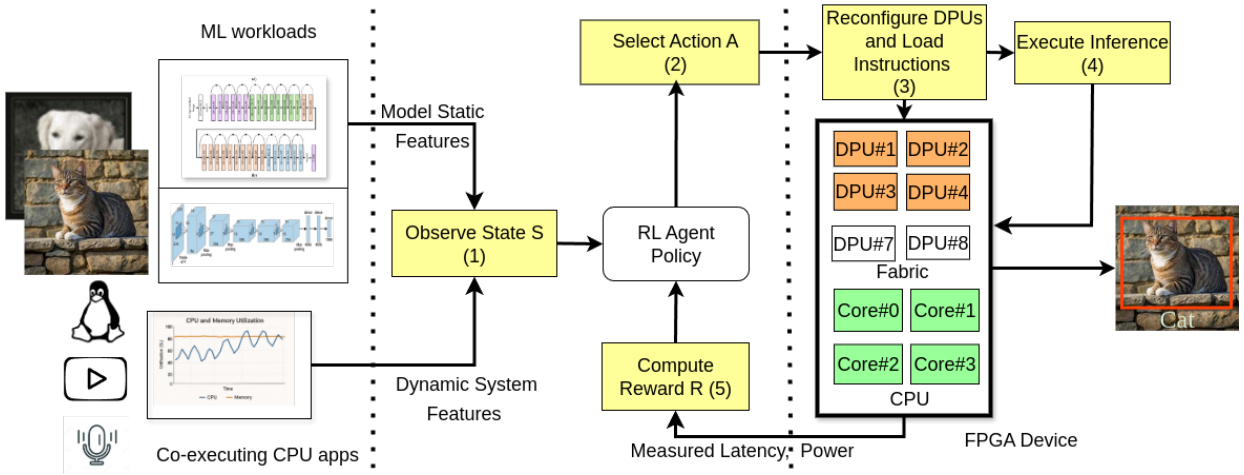

Fig. 4: High-level design of the DPUConfig framework.

Second, naive training without context awareness risks overfitting to the limited states seen during training, leading to poor generalization for unseen workloads or models. These challenges are related to moving-target problems in RL, studied in contextual bandits [17], input-driven environments [18], and state-dependent baselines [19].

To address this, we adopt a context-aware reward design. Workload-dependent state (CPU and memory port utilization, and model characteristics such as GMACs and data transfers) are included in the state. Rewards are defined relative to context-specific baselines rather than absolute PPW. This ensures that the agent evaluates performance in relation to what is achievable under the current workload and model. By combining contextual baselines with normalized reward shaping [18]–[20], the agent learns policies that generalize across workloads and models while avoiding the moving-target problem.

Algorithm 1 formalizes reward calculation using the sampled system metrics. If performance constraints are not met, a negative reward is returned. Otherwise, the reward is based on PPW, normalized against a blended baseline. This baseline is computed from two sources: a local average b_{local} from the current context bucket (defined by the current workload-dependent state) and a global average b_{global} across all contexts. A blending factor λ interpolates between them to balance local adaptation and global stability. The local average b_{local} is updated online with new PPW samples, capturing efficiency under similar conditions, while the global average b_{global} aggregates statistics across all buckets to provide a fallback when data is sparse. The factor α scales the reward to avoid extreme values. This formulation ensures bounded rewards, emphasizes relative improvements, and prevents unstable updates during training.

In conventional RL, agents maximize cumulative rewards, which are often assumed to increase monotonically. However, in our case, the reward is defined as the relative improvement over a context-specific baseline, and thus it naturally fluctuates

Algorithm 1 Reward calculation based on PPW and constraints.

procedure CALCULATEREWARD(S)
  measuredFPS $\leftarrow S.fps$
  fpgaPower $\leftarrow S.fpgaPower$
  cpuUtil $\leftarrow S.cpu$, memUtil $\leftarrow S.memory$
  gmac $\leftarrow S.gmac$, modelData $\leftarrow S.modelData$
  ppw $\leftarrow \frac{measuredFPS}{fpgaPower}$
  if $(measuredFPS < FPSConstraint)$ **then**
    $r \leftarrow$ -1.0
    return r
  $contextKey \leftarrow (cpuUtil, memUtil, gmac, modelData)$
  $b_{\text{local}} \leftarrow$ CTXMEAN$[contextKey]$
  $b_{\text{global}} \leftarrow$ GLOBALMEANPPW
  baseline $\leftarrow (1-\lambda)b_{\text{local}} + \lambda b_{\text{global}}$
  $r \leftarrow \tanh\Big(\frac{\text{ppw} - \text{baseline}}{\alpha \cdot \max(1, |baseline|)}\Big)$
  Update CTXMEAN, GLOBALMEANPPW
  return r

around zero: Positive values indicate better-than-baseline performance, while negative values represent degradation. Without bounding, these fluctuations can lead to instability during learning when the difference from the baseline is extreme. Prior work has shown that reward clipping or squashing functions help stabilize training by limiting the influence of outliers [21] and preventing the policy from overemphasizing rare high-magnitude deviations [22]. As a result, the policy learns to generalize across varying workloads and model states without being destabilized by rare outliers, consistent with findings in prior work on stabilized RL training [23].

Training. Algorithm 2 summarizes the training procedure of our RL agent. Instead of running live hardware experiments during training, we rely on a large set of pre-recorded measurements. These were collected from exhaustive runs covering different DPU configurations, ML models, and workload states. At each training step, the system is initialized to an initial state according to the chosen workload mode (C, N, or M) and target model. We adopt single-step episodes:

Algorithm 2 Training the RL Agent with PPO

```
procedure TRAINRLAGENT
    for episode = 1 ... N do
        Select workload mode (N, M, C) & ML model
        Initialize system to empty state
        Observe initial system metrics and model features
        a ← Agent selects action based on current policy
        state ← Fetch telemetry data
        r ← CALCULATEREWARD (state)
        Update PPO policy parameters using (state, a, r)
    return trained RL agent
```

the agent observes the initial state, selects an action, and the outcome is retrieved from the corresponding pre-recorded experiment data. The collected system metrics and model characteristics are then passed to the reward function (Algorithm 1) to compute the training signal. Finally, the Proximal Policy Optimization (PPO) [24] backend updates the policy based on the observed reward. This process is repeated across all state-action combinations, and the resulting trained RL agent is returned. The RL agent was implemented using the OpenAI Gymnasium [25] environment and trained using Ray RLLib [26]. Training was conducted for one million episodes; only the discount factor ($\gamma = 0.995$) and learning rate (1×10^{-5}) were overridden, while all other PPO hyperparameters and the backend model architecture retained the default Ray RLLib PPO configuration.

V. Experimental Evaluation

A. Experimental Setup

We perform our evaluation on the Xilinx Zynq UltraScale+ FPGA board ZCU102 [12], using the Vitis AI v3.5 framework [4], and the DPU v4.1 [3]. Our framework is evaluated using ten CNNs for image classification and the YOLOv5 detector (Table III), which have varying compute and memory demands. For each model, two pruned variants with pruning ratios 25% and 50% were also added, for a total of 33 models. The ImageNet [30] dataset was used as input for the classification networks, and COCO [31] for the object detector. We use a PyTorch [11]-based script and the `vaitrace` [4] tool to extract static model features, and the `stress-ng` [32] utility to emulate the workload states C and M.

In total, 2574 experiments were executed, covering the space of models, configurations, and workload states (26 DPU configurations $\times$ 11 models $\times$ 3 pruned variants $\times$ 3 workload states). Each experiment was run for a fixed number of input images. An OpenTelemetry collector [33] ran on a separate machine, collecting system metrics exported by a Prometheus Node Exporter [34] instance on the ZCU102 at 3Hz, alongside application performance metrics. This sampling rate provided sufficient time resolution for accurate power measurements, while incurring less than 30% utilization of a single CPU core without affecting the results. Power measurements were obtained from the integrated sensors of the ZCU102 board.

The CNN models were split into two sets: i) 24 for training and ii) 9 for testing. The split was determined via k-means clustering on GMAC values, grouping the models into three categories: small, medium, and large. One representative model with its two pruned variants from each category was placed in the test set. All three workload modes (N, C, M) were included during training. Model and workload combinations were presented to the RL agent in round-robin order, ensuring exposure to the full range of state–action pairs.

B. Results

Fig. 5 presents the normalized PPW achieved by DPUConfig compared against: i) an Optimal configuration, which always runs inference on the best DPU configuration, ii) the configuration with the maximum FPS, and iii) the one with minimum power dissipation. For the workload state C, DPUConfig achieves on average 97% of the optimal PPW, with two cases exactly matching the optimal configuration. In workload state M, where memory bandwidth is heavily stressed and DPU performance is further degraded, the average drops slightly to 95%, with no case reaching the exact optimum. This is a strong indication that the RL agent is tuned to generalize across different models and system states rather than overfitting to specific cases. The maximum-FPS configuration (typically B4096_1) achieves only 47% of the optimal PPW in workload state C and 35% in workload state M, showing that the largest DPU is not necessarily the most energy-efficient. Similarly, the minimum-power configuration (B512_1) consistently falls far short of optimal efficiency. These results confirm that neither extreme (largest nor smallest DPU) is efficient and that DPUConfig achieves near-optimal energy efficiency in all conditions.The results in Fig. 5 exclude DPU reconfiguration time, assuming batch sizes are large enough to amortize the reconfiguration overhead. In all evaluation experiments, the performance constraint was set to 30 FPS and was satisfied in 89% of the test cases, with violations occurring only for the demanding ResNet152 model under the M workload state.

Fig. 6 depicts the timeline for model arrival and DPU configuration. Solid and dashed lines represent observed and average PPW, respectively, for InceptionV3 and ResNext50. Shaded regions indicate ZCU102 overheads: telemetry (88 ms), RL inference (20 ms), reconfiguration (384 ms), and instruction loading (507 ms). While a DPU change incurs the full 999 ms overhead, reuse eliminates the reconfiguration and loading phases. For long-running inferences, this latency is negligible compared to runtime, enabling near-optimal efficiency.

VI. Prior work

CNN accelerators. Many works propose efficient CNN accelerators for FPGAs [35]–[37]. Specifically for DPU design, Du et al. [38] propose a framework for accelerating DNN inference on FPGAs using heterogeneous multi-DPU engines, extending Vitis AI support for homogeneous DPU deployments. Their system uses both task-level and pipelined parallelism to map CNN layers to distinct DPUs based on resource profiles. The method improves inference throughput

TABLE III: ML model characteristics. Latency and Data I/O refer to the inference of a single image using the B4096_1 configuration. The reported accuracy is for *INT8* quantized models without pruning. For YOLOv5s, accuracy refers to Mean Average Precision (mAP). Each model has also two pruned versions (25% & 50%).

Type	Model	Latency (ms)	Avg. *INT8* Accuracy	# Layers	# GMAC operations	Data I/O between DRAM–DPU (MB)	Bandwidth (GB/s)	Arithm. Intensity (MACs/Byte)	DPU Efficiency
Training	ResNet18	4.43	67.90%	18	1.82	12.13	2.03	149.83	71.90%
	ResNet50	11.72	77.60%	50	4.10	38.94	2.85	105.33	59.00%
	MobileNetV2	3.21	68.23%	53	0.30	5.74	1.49	52.49	17.10%
	DenseNet121 [27]	17.39	68.70%	98	2.86	43.74	2.93	65.28	26.90%
	InceptionV4	32.23	77.14%	150	12.3	89.00	2.54	138.23	63.00%
	RepVGG A0	4.83	72.41%	45	1.52	11.84	2.00	128.26	53.40%
	ResNext-50 32x4d [28]	27.42	76.21%	50	11.41	95.85	3.17	119.06	68.90%
	YOLOv5s	34.70	42.10%	60	8.26	159.80	3.27	51.69	42.90%
Test	RegNetX 400MF [29]	5.71	70.15%	72	1.57	24.33	3.76	64.57	47.40%
	InceptionV3	15.03	77.03%	98	5.74	43.13	2.46	133.05	63.50%
	ResNet152	30.81	78.48%	152	11.54	76.52	2.35	150.81	62.00%

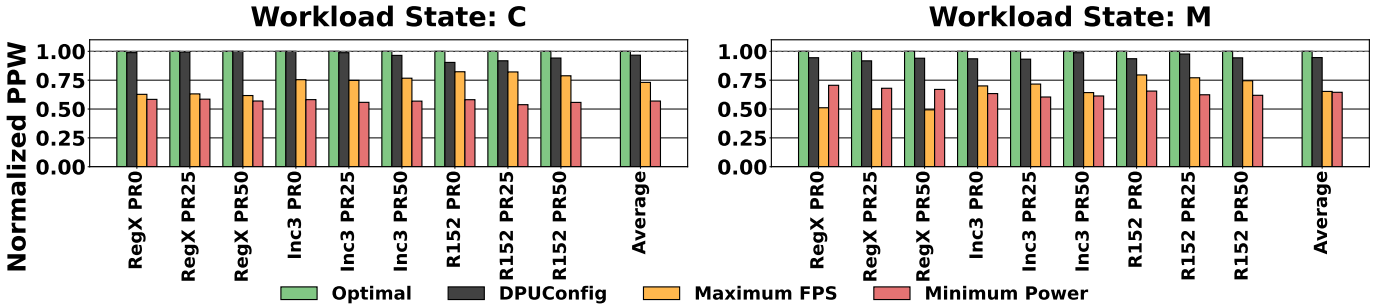

Fig. 5: Normalized PPW results of DPUConfig across two workload states (C, M). Model Abbr.: RegX = RegNetX, Inc3 = InceptionV3, R152 = ResNet152. PR0, PR25, PR50 denote pruning ratios of 0%, 25%, and 50%, respectively.

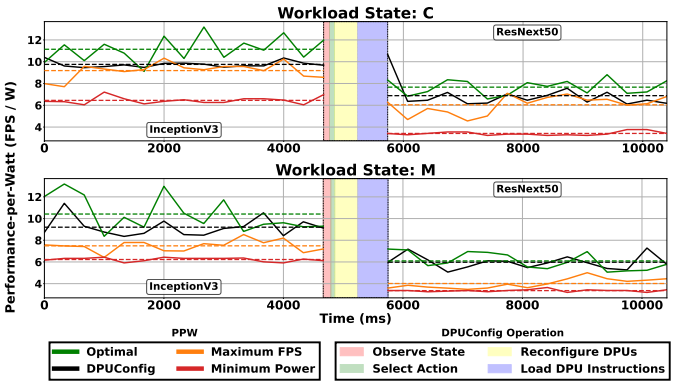

Fig. 6: Timeline of DPUConfig operation during inference of InceptionV3 and ResNext50, where a reconfiguration takes place.

by up to 19% on the ZCU104 platform, improving resource utilization and scheduling efficiency for CNN workloads. Unlike DPUConfig, this work does not employ RL methods for optimization. Also, emphasis is purely on improving performance, rather than achieving energy efficiency.

EXPRESS [39] and its successor, EXPRESS-2.0, are ML-based frameworks that predict execution time for concurrent CNNs on Xilinx DPUs. By incorporating hardware, model, and interconnect features, they achieve prediction errors as low as 0.7%, facilitating design space exploration. However, these studies focus primarily on latency prediction rather than optimizing DPU configurations for energy efficiency.

RL for system optimization. Recent work leverages RL for system optimization on heterogeneous platforms. ConfuciuX [40] uses RL and genetic algorithms for resource allocation within DNN accelerators, converging 4.7–24× faster than prior methods. BAND [41] and HERTI [42] employ RL schedulers for fine-grained subgraph placement across CPUs, GPUs, and NPUs to minimize latency. While recent studies extend scheduling to FPGAs [43], these approaches focus on task scheduling rather than dynamic DPU configuration, which is the focus of DPUConfig.

VII. Conclusions

To support energy-efficient ML inference in an FPGA-based MPSoC platform, we introduce DPUConfig, a custom RL-based agent that continuously learns and selects near-optimal DPU configurations by taking into account the features of the ML model and dynamically adapting to the stochastic runtime variance. The evaluations conducted on a ZCU102 MPSoC show that, on average, DPUConfig selects configurations achieving 95% of the optimal PPW, proving the effectiveness of our framework.

References

[1] S. Mittal, "A survey of FPGA-based accelerators for convolutional neural networks," *Neural Computing and Applications*, vol. 32, no. 4, pp. 1109–1139, 2020.

[2] F. Yan, A. Koch, and O. Sinnen, "A survey on FPGA-based Accelerator for ML Models," 2024. [Online]. Available: https://arxiv.org/abs/2412.15666

[3] Xilinx Inc., *DPUCZDX8G for Zynq UltraScale+ MPSoCs: Product Guide*, Xilinx, 2021, pG338 (v3.2), Accessed: 2025-06-29.

[4] AMD/Xilinx Inc., *Vitis AI User Guide (UG1414, Version 3.5)*, Sep. 2023.

[5] T. Wu, W. Liu, and Y. Jin, "An End-to-End Solution to Autonomous Driving Based on Xilinx FPGA," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 427–430.

[6] C. Li, R. Xu, Y. Lv, Y. Zhao, and W. Jing, "Edge Real-Time Object Detection and DPU-Based Hardware Implementation for Optical Remote Sensing Images," *Remote Sensing*, vol. 15, no. 16, p. 3975, 2023. [Online]. Available: https://doi.org/10.3390/rs15163975

[7] R. A. Amin, M. Hasan, V. Wiese, and R. Obermaisser, "FPGA-based Real-Time Object Detection and Classification System using YOLO for Edge Computing," *IEEE Access*, vol. PP(99), 2024.

[8] M. S. Hussein, M. A. Soliman, M. Elhoseny, A. Khalil, and K. Wahba, "Implementation of a DPU-Based Intelligent Thermal Imaging Hardware Accelerator on FPGA," *Electronics*, vol. 11, no. 1, p. 105, 2022.

[9] N. Perryman, S. Sabogal, C. Wilson, and A. D. George, "Dependable DPU Architectures on AMD-Xilinx Versal Adaptive SoCs for Space Applications," *IEEE Transactions on Aerospace and Electronic Systems*, 2025.

[10] J. Bai, F. Lu, K. Zhang *et al.*, "ONNX: Open Neural Network Exchange," https://github.com/onnx/onnx, 2019.

[11] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *PyTorch: an imperative style, high-performance deep learning library*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[12] Xilinx Inc., "Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit," https://www.xilinx.com/products/boards-and-kits/zcu102.html, 2021.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun. 2016, pp. 770–778.

[14] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun. 2018, pp. 4510–4520.

[15] B. Li, B. Wu, J. Su, G. Wang, and L. Lin, "EagleEye: Fast Sub-net Evaluation for Efficient Neural Network Pruning," *CoRR*, vol. abs/2007.02491, 2020. [Online]. Available: https://arxiv.org/abs/2007.02491

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: A Bradford Book, 2018.

[17] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A Contextual-Bandit Approach to Personalized News Article Recommendation," in *Proceedings of the 19th International Conference on World Wide Web*, Apr. 2010, pp. 661–670.

[18] H. Mao, S. B. Venkatakrishnan, M. Schwarzkopf, and M. Alizadeh, "Variance Reduction for Reinforcement Learning in Input-Driven Environments," in *7th International Conference on Learning Representations, ICLR, New Orleans, LA, USA, May 6-9*, 2019.

[19] G. Tucker, S. Bhupatiraju, S. Gu, R. E. Turner, Z. Ghahramani, and S. Levine, "The Mirage of Action-Dependent Baselines in Reinforcement Learning," Nov. 2018.

[20] A. Y. Ng, D. Harada, and S. J. Russell, "Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping," in *16th International Conference on Machine Learning (ICML)*, ser. ICML '99, San Francisco, CA, USA, Jun. 1999, pp. 278–287.

[21] V. Mnih *et al.*, "Human-Level Control through Deep Reinforcement Learning," vol. 518, no. 7540, pp. 529–533, 2015.

[22] J. Fu, X. Zhao, C. Yao, H. Wang, Q. Han, and Y. Xiao. Reward Shaping to Mitigate Reward Hacking in RLHF. [Online]. Available: http://arxiv.org/abs/2502.18770

[23] M. Hessel *et al.*, "Rainbow: Combining improvements in deep reinforcement learning," in *32nd Conference on Artificial Intelligence*. AAAI Press, 2018, pp. 3215–3222.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. [Online]. Available: http://arxiv.org/abs/1707.06347

[25] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.

[26] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica, "Ray: a distributed framework for emerging ai applications," in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'18. USA: USENIX Association, 2018, p. 561–577.

[27] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2018. [Online]. Available: https://arxiv.org/abs/1608.06993

[28] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," 2017. [Online]. Available: https://arxiv.org/abs/1611.05431

[29] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," 2020. [Online]. Available: https://arxiv.org/abs/2003.13678

[30] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun. 2009, pp. 248–255.

[31] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2015. [Online]. Available: https://arxiv.org/abs/1405.0312

[32] "stress-ng," https://github.com/ColinIanKing/stress-ng.

[33] "Open Telemetry," https://opentelemetry.io.

[34] "Prometheus Node Exporter," https://github.com/prometheus/node_exporter.

[35] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[DL] A survey of FPGA-based neural network inference accelerators," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 1, pp. 1–26, 2019.

[36] Umuroglu, Yaman and others, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM, 2017, pp. 65–74.

[37] F. Fahim *et al.*, "hls4ml: An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices," *CoRR*, vol. abs/2103.05579, 2021. [Online]. Available: https://arxiv.org/abs/2103.05579

[38] Z. Du, W. Zhang, Z. Zhou, Z. Shao, and L. Ju, "Accelerating DNN Inference with Heterogeneous Multi-DPU Engines," in *Proceedings of the 60th Design Automation Conference (DAC)*, 2023, pp. 1–6.

[39] S. Goel, R. Kedia, R. Sen, and M. Balakrishnan, "EXPRESS: A Framework for Execution Time Prediction of Concurrent CNNs on Xilinx DPU Accelerator," *ACM Transactions on Embedded Computing Systems (TECS)*, 2024.

[40] S. Kao, G. Jeong, and T. Krishna, "ConfuciuX: Autonomous Hardware Resource Assignment for DNN Accelerators using Reinforcement Learning," in *Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 622–636.

[41] X. Li, Y. Zhu, and Y. Zhang, "BAND: Coordinated Multi-DNN Inference on Heterogeneous Mobile Processors," in *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*. ACM, 2022, pp. 175–188.

[42] Z. Liu, X. Wu, and K. Huang, "HERTI: A Reinforcement Learning-Augmented System for Efficient Real-Time Inference on Heterogeneous Embedded Systems," in *Proceedings of the 2021 International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2021, pp. 195–207.

[43] Y. Wu, G. Wu, Y. Wang, and X. Liao, "Reinforcement Learning-Based Task Scheduling for Heterogeneous Computing in End-Edge-Cloud Environment," *Cluster Computing*, 2025.