Deadline-Guarded Edge Inference with FPGA Preemption

Zhuoran Chen, Jiale Chen, Rui Tan, Wenjing Yang, and Mo Li

***Abstract*—Deep neural networks (DNNs) are increasingly being employed in delay-sensitive edge applications such as autonomous driving, industrial automation, and extended reality. However, due to the use of complex computing hardware and algorithms, the execution time of a DNN is stochastic and follows a distribution usually with a long tail exceeding the specified deadline. To guard the deadline, we propose a real-time and preemptive GPU-FPGA heterogeneous computing system that continuously monitors the progress of the DNN execution on the GPU and predicts deadline miss. Upon prediction of a miss, the system activates the FPGA as a deadline guardian to run a smaller DNN for the same task to meet the deadline. We design lightweight predictors based on offline data of progress versus final completion time and hardware status readings. Moreover, to deal with the performance loss due to resource contention on the FPGA, we further propose a simulation method to find the optimal preemption plan in multitasking scenario. Extensive evaluation with a Jetson AGX Orin GPU and a Xilinx ZCU102 FPGA shows that our system achieves up to 20 times reduction on miss rate and improves effective accuracy by more than 10%.**

***Index Terms*—Edge AI, Deadline Miss Prediction, GPU-FPGA Heterogeneous Computing, Preemption, Real-Time Inference**

I. Introduction

DEEP neural networks (DNNs) are increasingly employed in delay-sensitive mobile perception applications [1]–[3], such as autonomous driving, robots, and augmented reality. For example, to achieve safe autonomous driving, the driving agent must adhere to an end-to-end processing latency requirement of less than 100 milliseconds for sensing data processing [3]. However, a known issue of the deployed DNNs is the large time jitter in DNN executions, e.g., more than 300 milliseconds in Apollo's [4] perception and planning modules [5], [6]. For a vehicle traveling at 120 kilometers per hour, every additional 0.3 seconds of delay increases its stopping distance by 10 meters, thereby degrading driving safety.

Z. Chen was with the College of Computing and Data Science, Nanyang Technological University, Singapore. He is now with the Department of Computer Science and Engineering, the Hong Kong University of Science and Technology, Hong Kong.
E-mail: zr.chen@connect.ust.hk
J. Chen and R. Tan are with the College of Computing and Data Science, Nanyang Technological University, Singapore.
E-mail: {jiale.chen, tanrui}@ntu.edu.sg
W. Yang is with the College of Computer Science and Technology, National University of Defense Technology, China.
E-mail: wenjing.yang@nudt.edu.cn
M. Li is with the Department of Computer Science and Engineering, the Hong Kong University of Science and Technology, Hong Kong.
E-mail: lim@cse.ust.hk

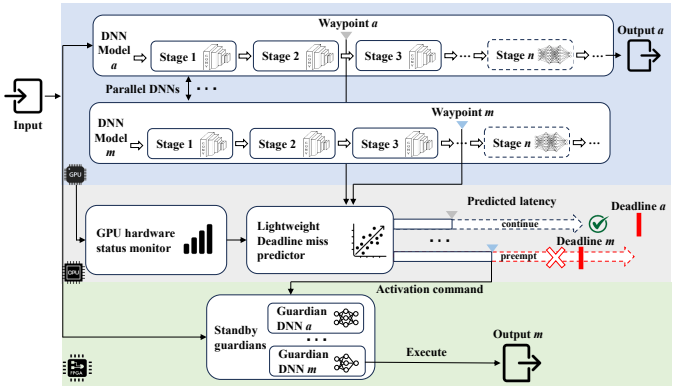


Fig. 1. Design Overview.

To reduce the impact of jitters, recent studies [6]–[9] have primarily focused on constructing the execution pipeline based on the worst-case execution time (WCET) analysis of GPU-based applications. In this approach, the WCET (e.g., 99th percentile of execution time) of each candidate model is estimated at first. The model with the WCET closest to the given deadline is deployed on the device. However, the latency profile of a DNN often exhibits long-tail behavior, i.e., the maximum latency is much larger than the average latency, while the frequency of long latency instances is low. Due to this, the WCET approach can be overly conservative and leads to low utilization of computational resources. Another approach is early exit [10]–[12], which inserts early-exit classifiers into some intermediate layers of a DNN model. Whenever the prediction confidence is high, the execution exits via an inserted classifier to reduce average latency. However, the execution time of the model with early exits is unpredictable and depends on the input data. It may still exceed the deadline, especially when no early exits are used due to not high enough confidence through the course. Essentially, early-exit strategies reduce average inference latency to maximize throughput, while it cannot guarantee the timely inference before a deadline. However, in real-time applications, timely model execution before the deadline is important for decision-making to avoid accidents. As a result, early-exit is not suitable for such real-time applications. Moreover, both WCET and early-exit approaches fall short of countermeasures to deal with potential deadline misses directly.

In this study, we propose a real-time and preemptive heterogeneous computing system that comprises a GPU-based *primary computing unit* and a Field-Programmable Gate Array (FPGA)-based *deadline guardian*. As illustrated in Fig. 1, our

system executes multiple DNNs on a primary GPU for each inference task. Concurrently, on the host CPU of the GPU platform, a lightweight prediction model works to persistently forecast potential deadline misses. This prediction is informed by a combination of inputs: timestamps reported from predefined progress waypoints within the DNNs, and integrated real-time GPU hardware status readings. If a deadline miss is forecast, the CPU sends an activation command to a standby FPGA, which serves as the deadline guardian. This guardian then executes a smaller, faster model using the same inference input, acting as a preemptive measure to ensure a result is delivered before the deadline.

Our design exploits the near-zero jitter property of FPGA computation for deterministic algorithms including DNNs. Running a small model on FPGA can ensure timely completion of the inference even when the deadline miss is predicted at the late phase of the GPU inference process. As a result, the predicted deadline miss can be avoided with the cost of reduced model accuracy. In safety-critical domains like autonomous driving, systems face strict real-time constraints where a late inference result is considered a system failure [13]. For these applications, the consequence of a missed deadline is far more severe than the potential reduction in model accuracy from using a smaller, faster model. This preemptive mechanism is therefore designed to improve the system's functional accuracy by ensuring a timely output is always available, preventing the zero-utility outcome of a late result.

The design of our system mainly faces two key challenges. The first challenge is the difficulty in real-time deadline miss prediction. Making accurate predictions on real-time DNN inference latency is hard. While existing works [14]–[17] present different methods to predict the average latency of a DNN based on the network structure, such offline latency predictors are inaccurate for online deadline miss prediction, due to the latency variance or jitters in real deployment. The increased latency variability in multitask, resource contention scenarios further exacerbates the problem. Moreover, the available time for deadline miss prediction is limited. For example, considering a real-time stream processing application operating on an image sequence captured at 30 frames per second, the per-frame inference time should be within 33 ms before the next image comes in. Under such conditions, the deadline miss predictor has a time budget of milliseconds or even less, which is very tight for sophisticated predictors.

The second challenge arises from resource contention on the FPGA when protecting multiple DNNs in parallel. Specifically, in multitask scenario, each DNN on GPU has its guardian DNN on the FPGA. Only one guardian DNN can be executed at a time. Therefore, the guardian DNN may fail to run when the FPGA is busy running another guardian DNN, leading to potential deadline miss and degradation of overall effective accuracy.

To address the first challenge, we perform offline profiling of each DNN's latency and design a lightweight online deadline miss predictor. The profiling and predictor are based on carefully chosen progress inspection waypoints for each DNN. Specifically, for each waypoint, we select a guardian model and train four prediction models of predicted inference time versus the elapsed time to reach the current waypoint. For each combination of waypoint, guardian model, and prediction model, we calculate the expected system performance with the profiling dataset and select the combination with the highest performance. As such, the waypoint, guardian model, and prediction model can be used to predict deadline miss when the inference progresses to the waypoint.

To address the second challenge, we propose a Monte Carlo-based simulation method to simulate the resource contention on FPGA and select the combination of offloading plans, consisting of a set of waypoint, guardian model and decision model that maximizes the overall effective accuracy of the multitask scenario. Specifically, first, we identify the best preemption plans at each waypoint of each DNN. Then, we simulate the parallel execution of DNNs on the GPU, with each DNN implementing one of its preemption plans. We employ the Monte Carlo method to simulate preemption operations, recording all preemption attempts and their corresponding guardian workloads on the FPGA. Following this, we pinpoint all conflicts on the FPGA and calculate the expected overall system performance, taking into account some failures in preemption attempts. Finally, we select the combination of plans that ensures the maximum overall performance.

Our main contributions are summarized as follows:

- We propose a heterogeneous edge computing system consisting of a GPU-based main computing unit and an FPGA-based deadline guardian to greatly reduce deadline misses of DNN inference.
- We design a waypoint-based DNN inference progress monitoring approach and lightweight deadline miss predictors based on the monitoring result.
- We propose a Monte Carlo-based simulation approach to address FPGA resource contention in multitasking environments by selecting optimal offloading plans, enhancing the overall effective accuracy of parallel DNN protection.
- Evaluation results show that our system achieves up to 20 times reduction in miss rate and improves effective accuracy by up to 10%.

The remainder of this paper is organized as follows. Section II reviews related works. Section III presents the background and motivation of our work. Section IV introduces our system model and formally defines the general multi-task optimization problem that we aim to solve. We then present our two-stage solution to this problem. Section V details the first stage: a methodology for generating high-quality preemption plans for individual tasks. Section VI describes the second stage, where we use a Monte Carlo simulation to solve the full multi-task problem by finding the optimal combination of these plans under resource contention. Section VII presents our comprehensive evaluation results. Finally, we conclude the paper in Section VIII.

II. Related Work

A. Real-time DNN Inference at Edge

Meeting strict real-time requirements for edge DNN inference remains challenging. Early heuristic approaches relied on

deploying models with a Worst-Case Execution Time (WCET) shorter than the deadline [6]–[9]. However, due to execution jitters, WCET-based methods are overly conservative and often lead to severe under-utilization of computational resources.

To improve computational efficiency and overall system throughput, subsequent research has focused on scheduling and resource isolation. Task-level optimizations include earliest-deadline-first scheduling [18], streaming frameworks [19], and heterogeneous CPU-GPU co-scheduling [20]. At the hardware and execution levels, proactive techniques such as spatial partitioning [21], stream-level priority scheduling [22], [23], and resource-adaptive frameworks [24] are deployed to maximize hardware utilization and optimize execution flows. While these proactive techniques successfully optimize execution, they cannot entirely eliminate the residual tail latency that persists under concurrent workloads, nor do they offer a hard safeguard against actual deadline misses for individual inferences. To address this critical gap, we propose a deadline-guarded, real-time edge inference approach utilizing a GPU-FPGA heterogeneous system. Our framework is *complementary* to existing scheduling methods: rather than just optimizing typical execution, it provides a vital, reactive safety net that explicitly guarantees execution deadlines for every single DNN task.

B. Early-exit Approach

Early-exit is a category of approaches which allows the model to terminate early if the prediction confidence is high enough [10]–[12]. BranchyNet [10] is an early study on this, which uses the cross entropy of the prediction at each classifier as the early-exit confidence. MSDNet [11] introduced a multi-scale early-exit model to maintain coarse-level features throughout the network and reduce the interference between the intermediate classifiers, using the maximum as the early-exit confidence. Different from using manually set confidence threshold, the study in [12] used a variational Bayesian approach to learn when to stop predicting during the model training. In summary, early-exit represents an opportunistic method to reduce average inference latency, but it lacks awareness of deadline misses. Differently, our work features mechanisms of deadline miss detection and prevention.

C. DNN Latency Prediction

Existing studies [14]–[17] proposed various methods for DNN latency prediction. Many studies predict the latency only based on DNN model features. For example, the studies [14], [15] simply use Floating Points Operations (FLOPs) and Multiply-accumulate Operations (MACs) of the DNN to either directly predict model latency or use these as feature inputs of regressors to make predictions. These methods are inaccurate because they ignore execution conditions and runtime implementations. Other studies [16], [17] propose to predict the latency of operators. The predicted model latency is the sum of the latency of all the operators. For instance, NeuralPower [16] predicts the model latency by summing up the latency of all sequential layers. Moreover, nn-Meter [17] predicts the latency of the kernels, i.e., the fusion of multiple operators. The model latency is estimated by summing up the latency of all kernels. However, both categories of existing prediction methods differ from our work in terms of purpose and approach of prediction. The purpose of existing methods is to estimate model latency without implementing the models; the predicted latencies are used in the early stages of model design and selection. On the other hand, ours makes online predictions on whether a specific run will miss its deadline during the serving stage. In addition, we predict by conducting profiling runs and training on historical data.

D. GPU-FPGA Heterogeneous Computing

Several studies [25]–[27] propose the use of GPU-FPGA heterogeneous computing systems to improve latency and accuracy. The study in [26] proposed a GPU-FPGA heterogeneous system that runs the fully connected layers of a DNN on FPGA and the convolutional layers on GPU to achieve faster computation and lower power usage. Study [27] proposed to implement the feature extraction on FPGA and the classification on GPU to achieve efficient data processing. The study presented in [25] capitalizes on the energy-efficiency of FPGAs, maintaining continuous FPGA operation as an auxiliary system to augment main GPU-based system's performance. When the main system's hardware components fail, the FPGA can step in as a contingency backup system. Our approach diverges in its purpose of deploying a heterogeneous system. Rather than dividing a single DNN's workloads and offloading to different devices or using extra hardware to provide auxiliary support to optimize performance, we aim to bolster reliability of a real-time system. We position the FPGA as a latency guardian by utilizing the time stability of FPGA-based DNN execution.

III. Background and Motivation

This section outlines the core problem of GPU latency variability, evaluates the practical limitations of existing software-only solutions for our target domain, and provides empirical evidence to motivate our proposed GPU-FPGA heterogeneous system.

A. The Challenge of GPU Latency Variability

GPUs are known to suffer from latency variability due to unpredictable and non-deterministic behavior in their underlying closed-source runtimes and drivers. Previous researches [6], [7], [28], [29] have identified key reasons for this, including branch divergence within single instruction, multiple data (SIMD) execution, inefficient interaction between the CPU and GPU, and dynamic memory and thread management in the CUDA driver stack. This unpredictability poses a significant risk for time-critical systems, such as those in robotics and autonomous driving, where meeting deadlines is paramount.

B. Quantifying Latency Variability on GPU and FPGA

To motivate our work, we first quantify the difference in latency variability between GPUs and FPGAs, demonstrating that the unpredictability of GPUs poses a significant risk

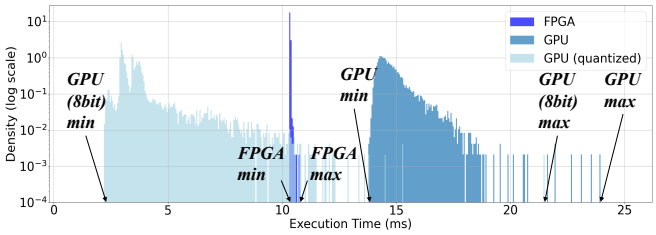


Fig. 2. Latency distribution for ResNet-50, highlighting the significantly higher variability and long-tail behavior on the Nvidia Jetson GPU compared to the highly deterministic performance of the Xilinx FPGA.

for time-critical applications. While GPUs are powerful, their performance jitter can be so severe that it undermines their suitability for tasks with strict deadlines, whereas FPGAs provide highly deterministic execution.

To understand how this problem impacts real-world applications under typical conditions, we conducted a preliminary study on two common hardware platforms. The first is Jetson AGX Orin GPU, a popular choice for mobile perception, and the second is Xilinx Zynq UltraScale+ MPSoC ZCU102 FPGA. For the GPU, we deployed ResNet-50 using PyTorch, evaluating both a full-precision model and an 8-bit quantized model using TensorRT 8.4. For the FPGA, we ran the 8-bit quantized ResNet-50 from the Vitis AI 3.5 library [30]. On both platforms, we used a common set of 10,000 inputs of size 224 × 224 and a batch size of 1, typical for streaming applications. All results were obtained under controlled ideal conditions: no resource contention, no power limits, and ideal cooling.

The results, shown in Fig. 2, expose the GPU's severe latency variability. For the full-precision model, the GPU delivered a mean latency of 17.18 ms but exhibited a long tail, with a standard deviation of 0.25 ms and a range of 7.450 ms—approximately 43.4% of the mean. The problem worsens with quantization; although the mean latency for the quantized model reduces to 5.14 ms, the standard deviation dramatically increases to 3.14 ms, and the latency range explodes to 12 ms, which is approximately 240% of the mean. As suggested in [3], tail latencies are the critical metric for evaluating time-critical applications. To better understand the severity, we consider the field of system safety, where "latency attacks" aim to compromise a system's responsiveness [31]–[34]. The scale of tail latency we observe relative to the mean is similar to what many latency attacks achieve [31], [33], rendering the system's performance profile akin to one under attack.

In contrast, the FPGA demonstrates highly predictable performance. The results for the FPGA, also shown in Fig.2, indicate a much narrower latency distribution. It achieved a mean latency of 10.36 ms with a standard deviation of just 0.009 ms, and a range of only 0.419 ms—approximately 4.04% of the mean. This makes the GPU's standard deviation over 340 times larger than the FPGA's for the same quantized model. Similar results are also found in [35] on a different FPGA platform. According to [36], the lower latency variability is due to the deterministic hardware and custom data flow architectures for DNN implementation.

This preliminary study highlights that even under ideal conditions, the GPU's long-tail latency poses a severe risk of breaching latency constraints, and its level of unpredictability can lead to dangerous situations [31]. The FPGA's deterministic, low-jitter behavior, however, makes it a more reliable platform for guaranteeing deadline adherence.

C. FPGAs as a Practical Guardian

Beyond determinism, the practical advantages of FPGA make it an ideal choice for a deadline guardian. As a guardian platform, the FPGA is only responsible for running a smaller, lightweight DNN when triggered. For this task, its high power efficiency is a major advantage; our experiments show the FPGA typically consumes 50% to 80% less energy than the GPU for a comparable inference, minimizing the overhead of adding the guardian. This cost-effectiveness extends to the budget. While the primary computing unit (the Jetson AGX Orin) is a high-performance, $2000 platform, a low-cost FPGA, at a fraction of that price ($100+), is sufficient to act as a guardian. This small incremental cost yields a significant improvement in system reliability by reducing deadline misses, making the integration of an FPGA a practical and economical solution.

TABLE I
QUALITATIVE COMPARISON OF GPU AND FPGA ARCHITECTURES IN THE PROPOSED SYSTEM

Metric/Feature	GPU	FPGA
Latency	High variability	Highly deterministic
Power Efficiency	Lower	High
Unit Cost	High	Low
Accuracy	Full precision	Quantization required
Ecosystem	Mature	Hardware-centric
Iteration Speed	Rapid	Slow
Scalability	High	Low

However, an FPGA-only system would be insufficient due to the trade-offs shown in Table I. First, FPGAs require quantization (typically INT8), lowering accuracy compared to full-precision GPU execution. Second, the hardware-centric workflow yields slow iteration speeds compared to rapid software development on GPUs. Finally, limited physical resources restrict FPGA scalability for executing multiple large DNNs simultaneously.

In summary, we propose a GPU-FPGA heterogeneous system to leverage the complementary strengths of both architectures. The GPU handles high-throughput, complex multi-DNN execution, while the FPGA serves as a reliable latency guardian, utilizing its highly deterministic execution to ensure critical deadlines are met.

IV. SYSTEM OVERVIEW AND PROBLEM FORMULATION

This section first describes the core components of our proposed GPU-FPGA system and details its operation flow. We then formally define the system's decision variables and formulate a general optimization problem. Finally, we analyze the key challenges that motivate our two-stage solution approach.

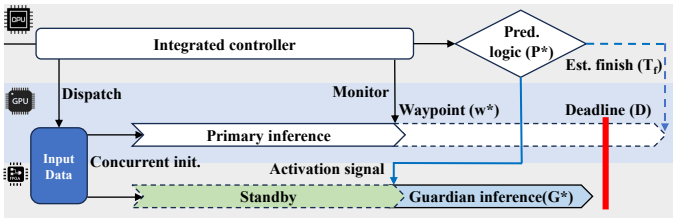


Fig. 3. The execution flow of the proposed system. An integrated controller monitors the primary GPU inference at waypoint w^*. If the predicted finish time T_f exceeds the deadline D, the standby FPGA guardian (G^*) is activated.

A. System Components

For each task k in a multi-task system, we first define a set of components that constitute the building blocks of our architecture:

Waypoints (W_k): A set of candidate waypoints, $W_k = \{w_{ik} | 1 \leq i \leq N_w^k\}$, is identified within the DNN's execution flow, where N_w^k is the total number of candidate waypoints for task k. Each w_{ik} is a specific point where we can measure the elapsed time.

Guardian Models (G_k): A corresponding set of guardian models, $G_k = \{g_{ik} | 1 \leq i \leq N_w^k\}$, is prepared on the FPGA. Each guardian model g_{ik} is uniquely optimized for and tightly coupled with its associated waypoint w_{ik}. As such, the choice of a waypoint implicitly determines the guardian model.

Prediction Models (P): A common set of candidate prediction model types, $P = \{P_j \mid 1 \leq j \leq N_p\}$, is defined, where N_p denotes the total number of available predictor types. These include lightweight models like linear regression and more complex ones like decision trees. A suite of models is prepared because the latency profiles of different DNNs exhibit varying degrees of unpredictability due to inherent DNN design and GPU operating conditions. A simple, linear model may suffice for a stable task, while a more complex, hardware-aware model is required to accurately predict the latency of a more jitter-prone task.

B. Execution Flow

Using a mobile perception system with K models running concurrently as an example, we now overview how our system prevents deadline misses at runtime using the components defined above. First, in an offline phase, a progress waypoint ($w \in W_k$) is determined for each DNN model on the GPU. Based on the timing constraints implied by this waypoint, a suitable guardian model is selected for the FPGA by trading off its accuracy and latency. The subsequent online phase, as illustrated in Figure 3, consists of four key steps: Concurrent Data Dispatch, Monitoring and Prediction, Decision and Preemption, and Guardian Execution.

Concurrent Data Dispatch: The host CPU orchestrates the inference process by dispatching the input data simultaneously to both the primary GPU and the guardian FPGA for each task. For the FPGA, the raw input is transmitted over a dedicated point-to-point Gigabit Ethernet link. The data volumes are modest: approximately 150 KB for a 224×224 image and 1.9 MB for a KITTI point cloud. Over the 1 Gbps link, these transfers complete in under 2 ms and approximately 16 ms, respectively—well before the earliest waypoints at 3.3 ms and 98.7 ms. The transfer latency is therefore fully masked by the GPU's primary execution and does not consume the guardian's time budget. Immediately after data dispatch, the GPU begins its inference computation. Concurrently, the FPGA receives the input and stores it in its local memory buffer. However, it does not initiate the inference computation on the hardware engine; instead, it holds the data in reserve and waits for an activation trigger. This strategy ensures that while multiple tasks can safely buffer their data on the FPGA simultaneously, the limited computational resources are only consumed if and when a specific guardian is explicitly activated. While a production deployment could employ PCIe or AXI DMA for higher bandwidth, the analysis above demonstrates that even standard Ethernet provides sufficient throughput for the input sizes in our target workloads.

Monitoring and Prediction: An integrated controller, running on the CPU of the main computing platform, monitors the GPU's progress. When the execution reaches the predetermined optimal waypoint (w^*), the controller records the elapsed time and queries the current hardware state. It feeds this information into the selected lightweight prediction model ($P^* \in P$), which also runs on the host CPU, and estimates the final completion time (T_f).

Decision and Preemption: The controller compares the predicted latency with the task's deadline (D).

- **If $T_f \leq D$ (No Miss Predicted):** The GPU continues its execution uninterrupted to provide the final, high-accuracy result. The controller sends a signal to the FPGA to abandon its standby state and discard the buffered input.
- **If $T_f > D$ (Miss Predicted):** The controller issues a guardian activation command via an SSH signal over a direct point-to-point Ethernet link. The GPU task is immediately terminated to save power and resources.

Guardian Execution: Upon receiving the guardian activation command, the FPGA transitions from standby to active and immediately begins processing the input it has already buffered, using its designated guardian model (G^*). The guardian processes the original input sample from scratch rather than receiving intermediate tensors from the GPU. While transmitting intermediate GPU results to the FPGA could theoretically save computation time, we avoid this approach because modern perception DNNs (e.g., those using encoder-decoder or Feature Pyramid Network structures [37]) maintain large intermediate tensors whose transmission latency often far exceeds the execution time of running a lightweight guardian model from scratch. Furthermore, relying on intermediate transfers imposes strict architectural coupling between the GPU and FPGA, which eliminates the flexibility to dynamically place waypoints and seamlessly swap heterogeneous guardian models.

This process ensures that the system defaults to the high-performance GPU while maintaining a robust fallback mechanism to meet every deadline.

C. Decision Variables and Problem Formulation

In a realistic mobile perception system, multiple (K) DNN tasks run concurrently on the GPU, all guarded by a single, shared FPGA. Our central goal is to select an optimal configuration for each task to maximize the overall system utility.

We define a **preemption plan** for task k, denoted O_{ijk}, which is defined by the choice of a waypoint w_{ik} (which implies a guardian g_{ik}) and a predictor type P_j. While our formulation is general, our solution methodology (detailed in Section V) will show that we first find the best predictor for each waypoint before tackling the system-wide optimization.

To evaluate system performance across heterogeneous tasks (e.g., classification, detection, segmentation), a unified utility function is required. However, the native performance metrics for these tasks, such as Top-1 accuracy for classification and mean Average Precision (mAP) for detection, are not directly comparable, making a simple weighted sum meaningless. To address this, we define a **normalized effective accuracy** ($\overline{Acc_e}$). The key to this metric is normalizing each guardian model's performance relative to its own task's high-accuracy GPU model. By dividing the guardian's accuracy ($Acc_{g_{ik}}$) by the main model's accuracy (Acc_{G_k}), we create a unitless quality score for each preemption. This allows the system's overall utility to be expressed as the expected value of this normalized accuracy, as shown below:

$$\overline{Acc_{e_{ijk}}} = (N_{F_{ijk}} \cdot \frac{Acc_{g_{ik}}}{Acc_{G_k}} + N_{G_{ijk}} \cdot 1)/N_{total\_k} \quad (1)$$

For task k using plan O_{ijk}: $N_{F_{ijk}}$ and $N_{G_{ijk}}$ are the on-time samples on FPGA and GPU, $Acc_{g_{ik}}$ and Acc_{G_k} are the accuracies of the guardian (associated with w_{ik}) and main models, and N_{total_k} is the total sample count.

The general optimization problem is to find the set of binary decision variables α_{ijk} that maximizes the weighted sum of this metric across all tasks:

$$\arg\max_{\alpha_{i,j,k}} \sum_{k=1}^{K} \sum_{i=1}^{N_w^k} \sum_{j=1}^{N_p} \alpha_{ijk} \cdot \lambda_k \cdot \overline{Acc_{e_{ijk}}} \quad (2)$$

$$\text{subject to} \sum_{i=1}^{N_w^k} \sum_{j=1}^{N_p} \alpha_{ijk} = 1 \qquad \forall k \in 1...K \quad (3)$$

$$\sum_{k=1}^{K} \lambda_k = 1 \quad (4)$$

where α_{ijk} is a binary decision variable that is 1 if the plan O_{ijk} consisting of waypoint w_{ik} and predictor P_j is selected for task k, and 0 otherwise. λ_k is a weight assigned to prioritize tasks based on their importance, and the constraint in Eq. 3 ensures exactly one plan is chosen for each task.

D. The Challenges of Optimization

*1) **Achieving Accurate Real-Time Prediction under Uncertainty**:* As highlighted in the Introduction, DNN inference latency exhibits significant jitter and relies on tight time budgets, making accurate deadline miss prediction highly difficult. Offline profiling alone is insufficient to capture runtime variance. This difficulty directly impacts the selection of the optimal preemption plan (O_{ijk}). There is a critical tension created by prediction uncertainty: predicting at an early waypoint leaves ample time to run a high-accuracy guardian on the FPGA, but the remaining GPU variance makes the prediction unreliable. Conversely, predicting at a late waypoint minimizes variance and improves prediction accuracy, but severely restricts the time budget, forcing the use of a lower-accuracy guardian. Therefore, the challenge is not just selecting a model, but determining the specific waypoint and predictor type that can reliably overcome real-time jitter to trigger the guardian only when necessary.

*2) **Quantifying Inter-Task Contention**:* Even with optimal plans for each individual task, their combined performance in a multi-task scenario is not simply the sum of their parts. Since all tasks share a single FPGA, their preemption attempts interfere with one another. A plan that preempts frequently might perform well in isolation but could fail often in a congested system, degrading its own effective accuracy and that of other tasks. Evaluating this coupled performance is intractable to profile directly due to the combinatorial explosion of multi-task plan combinations, making it impossible to measure the true values of $N_{F_{ijk}}$ and $N_{G_{ijk}}$ for the global system.

To address these challenges, we propose a two-stage approach. First, we develop a methodology to solve the prediction-preemption trade-off for single tasks (Section V). Second, we introduce an efficient simulation-based technique to resolve the inter-task contention and find the globally optimal set of plans (Section VI).

V. Single-Task Preemption Plan Generation

This section details our methodology for solving the first challenge identified in Section IV-D: finding an optimal preemption plan for a single task by systematically navigating the prediction-preemption trade-off. This trade-off is central to our system: predicting at an early waypoint leaves ample time for a high-accuracy guardian but yields unreliable predictions due to remaining GPU variance, while predicting at a late waypoint improves prediction accuracy but restricts the guardian to a smaller, less accurate model. Our approach consists of three steps: (1) generating waypoint and guardian candidates, (2) training a suite of prediction models, and (3) evaluating all combinations to select the best-performing plans. The output of this process is a set of high-quality candidate plans for each task, which serve as the input to the multi-task optimization stage described in Section VI.

A. Waypoint and Guardian Candidate Generation

To determine the set of candidate waypoints $W = \{w_i | 1 \leq i \leq N_w\}$, we adopt a filtering-based approach. Theoretically, a progress monitoring waypoint can be inserted after the execution of any computational operator. Therefore, for a DNN model comprising L layers (including convolution, pooling, normalization, etc.), we first consider all L layer boundaries as potential candidates, as shown in Step 1 of Fig. 4. However, not all layer boundaries are viable or effective for monitoring in a real-time system. We refine this initial set of L layers into

the final candidate set W by applying three sequential filtering rules.

Second, as depicted in Step 2 of Fig. 4, we filter out candidates based on compiler constraints and kernel fusion. Modern DNN compilers may optimize performance by merging consecutive operations, such as convolution, batch normalization, and activation, into a single computational kernel. Placing a waypoint within such a fused sequence is invalid, as the host CPU cannot interrupt or inspect the GPU's status inside a running kernel. Consequently, we remove all layer boundaries that are optimized away by the compiler, effectively reducing the candidate pool from the total number of logical layers to the number of executable kernels.

Third, we filter out candidates based on topological constraints (Step 3). Waypoints located inside parallel branches of the network structure are excluded. A timestamp recorded within a specific branch only reflects the completion of that path, leading to ambiguous progress measurements regarding the overall inference status. Therefore, we restrict candidates to the backbone or sequential join points of the network.

Fourth, we apply a sparsity constraint to determine the final set size N_w (Step 4). While many points may remain valid after the first two filters, dense monitoring at every kernel boundary introduces excessive communication overhead between the CPU and GPU. Instead, we select a small, representative subset from the remaining valid points. As illustrated in the final step of Fig. 4, these are chosen to be evenly distributed across the expected execution timeline (e.g., waypoints W_1, W_2, and W_3 at roughly 20%, 60%, and 80% of average latency). This distribution is designed to sample the prediction-preemption trade-off: early waypoints provide a larger time budget (allowing for higher-accuracy guardians), while late waypoints provide more accumulated runtime data (allowing for more accurate deadline miss predictions). The optimal balance between these two competing factors cannot be determined analytically and is resolved empirically in Section V-C.

For each selected waypoint $w_i \in W$, we then generate a corresponding optimal guardian model. Let T_{w_i} be the mean latency to reach waypoint w_i and D be the task deadline. The remaining time budget available for the FPGA guardian is $D - T_{w_i}$. To find the optimal guardian G_i, we generate a pool of potential guardian models (via training lightweight architectures or compressing the original model) and process them through the Xilinx Vitis AI toolchain for INT8 quantization. We deploy these candidates to the FPGA to profile their execution latency (T_g) and accuracy (Acc_g). From this pool, we filter for models that satisfy the timing constraint $T_g < D - T_{w_i}$ and select the one with the highest accuracy. This procedure yields the final set of tightly coupled waypoint-guardian pairs $\{(w_i, G_i) | 1 \leq i \leq N_w\}$.

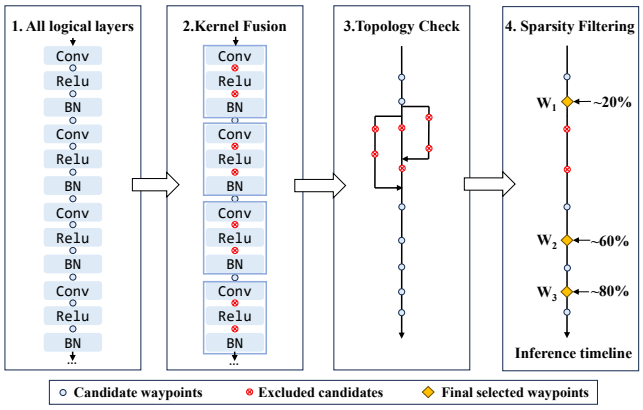


Fig. 4. Overview of the filtering-based waypoint selection process. The initial candidates from all logical layers are sequentially filtered by kernel fusion constraints, topological checks, and sparsity requirements to yield the final set W.

B. Prediction Model Training

1) Data Collection: With the waypoint candidates defined, we collect a dataset for training our prediction models under dynamic resource contention. We execute the target DNN concurrently with a secondary background DNN to simulate the resource contention characteristic of multi-task environments. The purpose of this background setup is to introduce controlled system-level variability, ensuring our dataset captures latency profiles across a wide spectrum of contention states, from idle to saturation. To generate diverse interference levels, we periodically vary the computational load of this background DNN. Specifically, we cyclically adjust the neuron count in its first layer through a wide range (e.g., from 1 to 32,768), holding each configuration for a fixed period (e.g., 30 seconds). This approach creates sustained epochs of low, medium, and high GPU load, allowing us to effectively profile the target DNN's behavior under varying interference.

In each profiling run, we record the inference time T_{w_i} at each waypoint candidate w_i, and the total inference time T_f. Moreover, we also record the platform's real-time hardware state S_{w_i}, which includes GPU and CPU workload, memory workload, GPU and CPU temperature, and CPU frequency, using Nvidia Jetson's `tegrastats` tool. The query of these readings has a sub-millisecond overhead and does not affect the overall DNN latency. After profiling, data tuples of (T_{w_i}, S_{w_i}, T_f) for each waypoint are stored in a dataset. This final, aggregated dataset is then partitioned into training and validation sets.

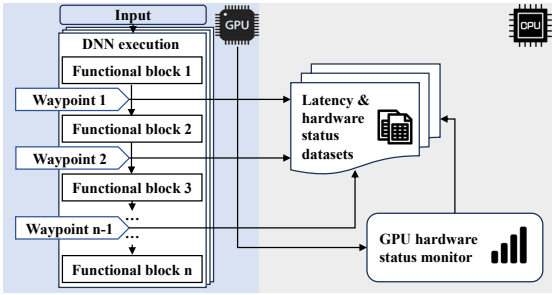


Fig. 5. Schematic of gathering training dataset at different waypoints.

2) Model Training: The choice of prediction model is not incidental: it directly affects preemption quality. An inaccurate predictor either triggers unnecessary preemptions (reducing effective accuracy by substituting the guardian for the GPU) or

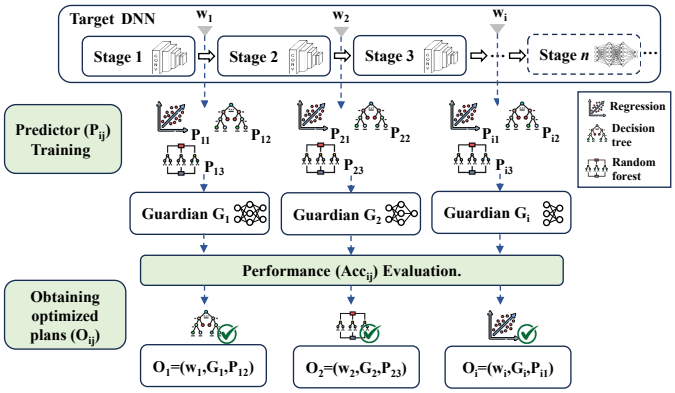


Fig. 6. Selecting the optimal preemption plan for a single DNN task by evaluating a set of waypoint candidates and their associated prediction models.

fails to detect true deadline misses (leaving them unguarded). Because the latency profiles of different DNNs exhibit varying degrees of unpredictability depending on DNN design and GPU operating conditions, no single predictor type is universally best. We therefore train a suite of $N_p = 4$ models for each waypoint candidate, spanning a range of complexity, and let the evaluation in Section V-C select the best one per waypoint.

The simplest model is a **linear regression** of T_f on T_{w_i}. While it has minimal overhead, its accuracy degrades at early waypoints or under high contention, where latency standard deviation can increase up to 5 times. To capture the non-linear hardware–software interactions that drive this variance, we also train three **hardware-aware models** that use the real-time hardware state S_{w_i} in addition to T_{w_i}: (1) multiple regression, which learns a linear mapping over the combined feature set; (2) a decision tree, which performs implicit feature selection to identify the most influential hardware metrics; and (3) a random forest, which aggregates multiple trees for improved robustness. All four models run in sub-millisecond time on the host CPU, ensuring negligible impact on overall inference latency.

C. Optimal Candidate Plan Selection

After generating all waypoint-guardian pairs and training the prediction models, we conduct a two-step process to find the best candidate plans for a single task. This selection process is visualized in Fig. 6, which shows how each waypoint candidate placed between the DNN's execution stages is associated with a suite of prediction models on the CPU and a dedicated guardian model on the FPGA.

This evaluation step is where the prediction-preemption trade-off is concretely resolved. First, for each waypoint candidate w_i, we determine its single best prediction model, P^*_{ij}. We do this by evaluating all N_p predictors for that waypoint using the effective accuracy metric, $Acc_{e_{ij}}$, on our held-out validation dataset:

$$Acc_{e_{ij}} = (N_{F_{ij}} \cdot Acc_{g_i} + N_{G_{ij}} \cdot Acc_G)/N_{total} \quad (5)$$

where $N_{F_{ij}}$ and $N_{G_{ij}}$ are the number of on-time samples executed on the FPGA and GPU, respectively, based on the predictions of P_{ij}. The predictor that yields the highest effective accuracy is selected for that waypoint.

This results in a set of optimized waypoint plans, $\{O^*_i = (w_i, G_i, P^*_{ij}) | 1 \leq i \leq N_w\}$. Each O^*_i now represents the best possible configuration for a given waypoint. Comparing across waypoints then reveals the optimal operating point on the trade-off curve: for example, in our ResNet-50 evaluation (Section VII-C1), Waypoint 3 with a simple regression predictor outperforms earlier waypoints paired with more complex models, because the improved prediction reliability at that later stage more than compensates for the smaller guardian.

Finally, we rank these optimized waypoint plans based on their effective accuracy scores. This process is repeated for each task k in the system, yielding a ranked list of high-quality, optimized candidate plans. This list serves as the direct input for the final multi-task optimization stage.

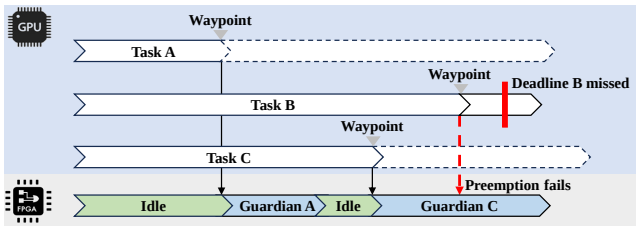


Fig. 7. Resource contention on the shared FPGA guardian.

VI. Multi-Task Preemption Optimization

This section addresses the challenge of inter-task contention on the shared FPGA guardian. As illustrated in Fig. 7, when multiple tasks run concurrently, a preemption attempt by one task acts as a resource block for others, potentially preventing a critical guardian execution if the FPGA is already busy. Consequently, the individually optimal plans derived in Section V may underperform in a global context, and the single-task profiling metrics ($N_{F_{ijk}}$ and $N_{G_{ijk}}$) become inaccurate as preemption success now depends on the coupled behavior of all tasks. Since profiling every possible combination of plans is computationally intractable due to the combinatorial explosion, we propose an efficient simulation-based approach to estimate global system performance and select the optimal set of plans.

To overcome this intractability, we introduce an efficient, simulation-based methodology. This approach allows us to accurately model the sequential execution and potential queuing delays on the shared FPGA, providing a fast and reliable estimation of the true, coupled performance metrics for any given set of preemption plans. With this simulation framework established, we can then proceed to solve the general optimization problem from Section IV.

A. Simulation-Based Optimization

To efficiently solve the general optimization problem, we introduce a simulation algorithm based on the Monte Carlo method. This simulation allows us to rapidly evaluate the performance of any given combination of preemption plans without costly real-world profiling.

The inputs to our optimization process are the ranked lists of optimized waypoint plans $\{O_i^*\}$ generated for each task in Section V. For each of these optimized plans, we compute its empirical preemption rate (P_{p_i}) from the validation dataset. This rate is the ratio of preemption attempts made by its chosen predictor (P_{ij}^*) to the total number of samples, and it serves as a statistical proxy for the plan's behavior.

The simulation algorithm, detailed in Algorithm 1, then models the parallel execution of all K tasks, each configured with a specific preemption plan. It simulates preemption outcomes over a large number of time steps, using the preemption rate P_{p_i} to stochastically model preemption attempts. By tracking the FPGA's availability, the simulation calculates the crucial **preemption success rate (P_{ps}^k)** for each task, which is the ratio of successful preemptions to total attempts.

Algorithm 1: Monte Carlo simulation for calculating preemption success rate under contention.

Input: A set of K preemption plans, one for each task.
For each task k: Deadline D_k, waypoint time T_{w_k}, guardian latency L_{G_k}, and preemption rate P_{p_k}.
T_{steps}: number of simulation steps; t_l: simulation step length.
Output: Preemption success rates $\{P_{ps}^1, ..., P_{ps}^K\}$.

$t \leftarrow 0$; $\gamma \leftarrow$ zero-array of length T_{steps};
for $k \leftarrow 1$ **to** K **do**
  $N_{attempts}^k \leftarrow 0$; $N_{success}^k \leftarrow 0$;
end for
for $i \leftarrow 1$ **to** T_{steps} **do**
  $t \leftarrow t + t_l$
  for $k \leftarrow 1$ **to** K **do**
    if *($t \geq T_{w_k}$) && (($(t - T_{w_k}) \bmod D_k = 0$) && (rand() $\leq P_{p_k}$)* **then**
      $N_{attempts}^k \leftarrow N_{attempts}^k + 1$;
      if $\gamma[i] = 0$ **then**
        $N_{success}^k \leftarrow N_{success}^k + 1$;
        Set $\gamma[i : i + L_{G_k}/t_l]$ to 1;
      end if
    end if
  end for
end for
for $k \leftarrow 1$ **to** K **do**
  $P_{ps}^k \leftarrow N_{success}^k / N_{attempts}^k$;
end for

The logic of Algorithm 1 is as follows. The algorithm first initializes a timeline vector γ to all zeros, representing an idle FPGA, and sets statistical counters for attempts and successes for each task to zero. The main loop then iterates through time in discrete steps t_l. In each time step, it checks every task to see if a preemption event should occur. The core logic resides in the conditional check: the condition $(t - T_{w_k}) \bmod D_k = 0$ simulates a periodic task arrival, triggering only at the precise moment a task's waypoint is reached in each of its execution cycles. A preemption is then attempted stochastically based on the plan's preemption rate P_p. If an attempt occurs, the algorithm increments the task's attempt counter. Crucially, contention is modeled by checking if the FPGA is idle ($\gamma[i] == 0$). If it is, the preemption is deemed successful, the success counter is incremented, and the γ vector is marked as busy (set to 1) for the duration of that guardian model's latency. If the FPGA is already busy, the attempt fails. After the simulation completes, the final preemption success rate for each task is calculated.

With the preemption success rate P_{ps}^k for a given set of plans, we can accurately calculate the expected values of $N_{F_{ijk}}$ and $N_{G_{ijk}}$ and thus compute the overall system objective function (Eq. 2). We can then embed this simulation within a search algorithm. Given the manageable number of high-quality candidate plans per task from Section V, an exhaustive search over the top few candidates for each task becomes feasible. This allows us to find the combination of plans $\{\alpha_{ijk}^*\}$ that maximizes the global system performance for final deployment.

VII. Evaluation

In this section, we present the evaluation results of our proposed system under different scenarios.

A. Evaluation Metrics

To evaluate the performance of our framework for a single DNN task, we use two key metrics. The primary indicator is the **Effective Accuracy** (Acc_e), as defined in Eq. 5. This metric correctly captures the combined utility of the system, rewarding high-accuracy GPU completions and appropriately valuing the slightly lower-accuracy (but on-time) guardian completions. Additionally, to directly measure the system's reliability, we report the **Miss Rate**, defined as the percentage of total inferences that fail to meet their deadline.

For the multi-task scenario, performance is assessed using a single, comprehensive system-wide metric: the **Weighted Mean Normalized Effective Accuracy**. This directly corresponds to our objective function from Eq. 2. The contribution of each task's Normalized Effective Accuracy ($\overline{Acc_e}$) to the final score is scaled by a weight, λ_k. We assign this weight to be proportional to a task's average execution latency on the GPU. This design choice is critical for system balance. Without it, a simple average would be dominated by high-frequency, low-latency tasks. Our weighting scheme ensures that less frequent but potentially more critical, long-running tasks are given appropriate priority in the optimization, ensuring the final score reflects true utility across the entire heterogeneous workload. In addition to this primary metric, we also report per-task miss rates and resource utilization statistics for a more detailed analysis.

B. Experiment Setup

In terms of software and hardware configurations of DNN implementations, we follow the same setup outlined in Section III. We implement the monitoring and preemption control modules using Python 3.8 on a Jetson AGX Orin running Jetpack L4T 35.1.0.

The selected deadline prediction model and its associated control logic are executed on the Arm Cortex-A78AE CPUs of the Jetson AGX Orin. Importantly, this lightweight model does not involve any deep learning components and runs entirely on the CPU, ensuring that it does not interfere with GPU resources or the execution of the primary inference task. Communication between the FPGA and GPU is established via a lightweight SSH connection. Instead of transmitting any intermediate results, we send only a compact control signal to trigger preemption. This design choice minimizes overhead, and detailed breakdown study is provided in evaluations.

On the FPGA side, we employ a Xilinx ZCU102 board configured with the DPUCZDX8G (B4096) overlay. This implementation consumes 52,161 LUTs, 98,246 registers, 255 BRAMs, and 710 DSP slices. While the ZCU102 is a high-end edge development board, our single-DPU configuration occupies only a fraction of its available fabric, and comparable resource budgets are available on lower-cost edge FPGAs, ensuring the design generalizes beyond this prototype platform.

C. Evaluation of Protection for a Single DNN Task

1) Example on ResNet-50: Firstly, we evaluate the performance of our preemption method protecting a single DNN task on GPU. We first focus on the popular ResNet-50 [38] as our GPU side example. The nominal latency requirement for a typical 60 FPS streaming task is approximately 16 ms. To evaluate performance under varying levels of deadline strictness, we define four deadlines based on their Z-scores relative to this mean. The Z-score is computed as: $Z = (X - \mu)/\sigma$, where X is the latency deadline, $\mu = 16$ ms is the mean latency, and $\sigma = 0.5$ ms is the standard deviation. We select Z-scores of -2, 0, 2, and 4, which correspond to latency deadlines of 15 ms, 16 ms, 17 ms, and 18 ms, respectively.

We test our method and two baseline methods with different deadline settings and compare their performances. Our approach's implementation includes GPU and FPGA profiling, preemption decision-making, and online deployment. Initially, we identify 4 monitoring points within the ResNet-50 model and select corresponding guardian models for each waypoint, as described in Section V-A. Then, we generate training datasets by profiling on 10,000 images in ILSVRC dataset, under varied GPU loads. Next, we train different models to predict deadline misses. The online deployment stage involves a preemption controller on Jetson, managing GPU-FPGA communication and executing tests to collect data for evaluation.

2) Baseline Methods: Our work introduces an application-level reactive controller designed to guard standard DNN deployments against unpredictable, dynamic latency jitter. For a fair comparison, we select baselines that represent alternative application-level strategies. We therefore distinguish our approach from the broad class of proactive system-level optimizations. This class includes everything from static enhancements (e.g., pruning, custom accelerators) to dynamic hardware schedulers (e.g., those that partition GPU resources). These powerful techniques are **complementary** to our work, not directly competitive. They aim to *reduce* the sources of latency jitter, creating a better-behaved baseline system. Our controller, in contrast, is designed to *react* to the *remaining*, unmitigated jitter that persists even in such optimized systems. Given this focus on application-level control, we selected two representative baselines that embody the most relevant alternative philosophies:

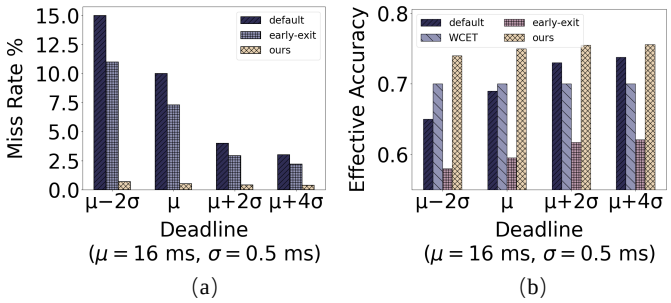


Fig. 8. System performance using different protection methods with different deadline settings. (a) Miss Rate. (b) Effective Accuracy. WCET method's miss rate is 0 at all deadlines and is not shown in (a).

WCET: This baseline represents the common strategy of ensuring deadlines by choosing a model that is statically provisioned to be fast enough. It embodies the philosophy of all static enhancements when applied to hard deadlines. To implement this, we selected the PyTorch pretrained ResNet-18 model, as its 99.9th percentile inference latency of 14.6 ms fits within our target deadlines, making it a robust representative of this "safe-by-design" approach.

Early-Exiting: While our method adapts dynamically to *system latency*, the most established alternative adapts to *input difficulty*. Early-exiting DNNs are the primary example of this philosophy. They are not designed to guard against system jitter, but they represent the state-of-the-art in dynamic accuracy-latency trade-offs, making them the most relevant conceptual baseline. To implement this, we trained an early-exit enabled ResNet-50 following standard procedures [39]. During inference, this results in a dynamic latency profile where approximately 26% of inputs exit early across its two exit branches.

3) Result Evaluation: We identify that our method performs best when making predictions with simple regression model at Waypoint 3 and using MobileNetV1 as guardian. As shown in Fig 8(a), in general, all methods provide certain level of miss rate reduction compared to the default situation in which no protection is provided. Our method outperforms early-exit method by reducing miss rate to under 1% under all deadline settings. WCET methods have a constant zero miss rate, however, it is achieved with compromised effective accuracies. As shown in Fig 8(b), our preemption based approach significantly outperforms the two baselines on effective accuracy. Our method has effective accuracies ranging from 0.74 to 0.756, which is very close to the ideal accuracy of 0.76 on GPU without any deadline requirements. In contrast, WCET has constant effective accuracy of 0.70, while early-exit causes significant accuracy drop to around 0.6. Although the baseline methods both achieve miss rate reductions, they offer little protection on maintaining the system's effective accuracies.

Specifically, for WCET method, although it achieves 0 miss rate, it is achieved by deploying a much lighter-weight and

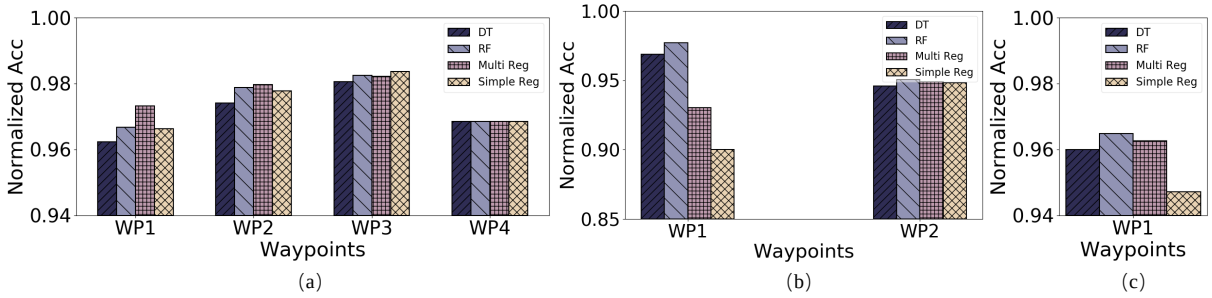


Fig. 9. System performance on different DNNs (a) ResNet-50. (b) Faster-RCNN. (c) PointPillars.

less accurate DNN. As explained in Section I and II, due to the long-tail distribution of DNN execution latency, the WCET of a DNN is much longer than its median latency. In our implementation, although the WCET of the deployed ResNet-18 is close to the deadlines, its median execution time is only 6.2 ms. This difference causes under-utilization of GPU resources and leads to compromised accuracy.

For early-exit method, the effective accuracy is much lower compared to others. The reason is twofold. First, the fundamental concept of early-exit is to trade accuracy for decreased latency. Specifically, the accuracy of the main branch of an early-exit enabled model is affected by its early-exit branches. During the training of an early-exit enabled model, the overall loss is calculated by the linear combination of each early exit branch's loss and the main branch's loss. Studies in [40] show that assigning higher weights to early exits can result in more assertive early decisions, but it may compromise overall accuracy. Fine-tuning of these parameters is required to find the best setting that mitigates the accuracy loss, but it requires much extra training efforts. Specifically, we set the early exits with default weights of 0.1 and 0.3 as in [39], and the final exit is weighted as 1.0 - (0.1 + 0.3) = 0.6. Second, although similar to our deadline-guarding methods, early-exiting methods are developed with the aim to improve system throughput by reducing average latency of processing one input. It is less effective in streaming scenario that inputs arrive at constant rate and the throughput requirement of the system is fixed, in which reducing average latency by early-exiting a proportion of inputs does not provide direct protection against deadline misses. As illustrated in Fig. 8(a), when the task is subject to a deadline, the miss rate of early-exiting remains high compared to other methods, resulting in even lower effective accuracy. Finally, if we integrate our deadline miss decision model with the early-exit model, the inference will be forced to exit when we decide a run will miss the deadline even if the confidence threshold is not met. Under those circumstances, it is obvious that there is no performance improvement forcing the model to exit without a reliable prediction result.

4) Protection Performance on Different DNNs: We also benchmark the performance of our preemption method when protecting other types of DNNs. In addition to ResNet-50, we introduce two other DNNs as examples. Specifically, for the common tasks of 2D detection and 3D point cloud detection, we select Faster R-CNN [41] and PointPillars [42] to evaluate the generalizability of our methods across different tasks. These models mirror typical tasks in mobile perception, and also represent light, medium, and heavy computational workloads. Using the same procedure, we develop preemption plans for each DNN and evaluate the protection performances. Specifically, each DNN is executed individually on the GPU. We use COCO [43] dataset for 2D detection, and KITTI [44] dataset for 3D point cloud detection. Deadlines are set to allow 10%–20% of samples to miss the deadline, balancing loose and strict conditions. Loose deadlines focus on identifying rare long-tail latency outliers, while strict deadlines require models to capture finer distinctions in input features for precise predictions.

TABLE II
PREEMPTION PROTECTION PERFORMANCE FOR DIFFERENT GPU DNN MODELS

GPU DNN	$\overline{Acc}$		Miss Rate%	
	Protected	Default	Protected	Default
ResNet-50	0.98	0.90	0.5	10.0
Faster-RCNN	0.97	0.85	1.7	15.0
PointPillars	0.96	0.80	1.0	20.0

Fig. 9 shows the performance of our system when using different prediction models to control preemption at different waypoints for the three different DNNs. The y-axis is normalized effective accuracy ($\overline{Acc}$) described in Section VI. The figure shows that hardware-aware decision models (multiple regression, decision tree, and random forest) are able to outperform simple regression model. This proves that using real-time hardware status information to help with deadline miss prediction is effective. For ResNet-50, multiple regression has the best performance at waypoints 1 and 2; for Faster-RCNN and PointPillars, random forest excels.

Table II shows the overall protection performance compared with default after selecting the best waypoint and prediction model for each DNN. Specifically, according to Fig. 9 we use simple regression at waypoint 3 for ResNet-50, random forest at waypoint 1 for Faster-RCNN and PointPillars. It is obvious that our method creates significant performance improvement with preemption compared with the default situation for all DNNs. It offers 9 to 20 times reduction on miss rate and 0.08 to 0.16 increase on $\overline{Acc}$ across three DNNs.

D. Evaluation of Protection for Concurrent Real-world Tasks

1) Experimental Setup and Compared Strategies: This experiment evaluates our system's ability to find a globally

optimal preemption strategy in a realistic, multi-task scenario, demonstrating its superiority over naive or heuristic approaches.

Setup: We create a representative real-world environment by running three distinct DNN tasks concurrently on the GPU, all protected by a single FPGA guardian. The tasks, detailed in Table III, are chosen for their relevance to robotics and autonomous systems [29], [45], covering a variety of workloads from 2D classification to 3D detection. Table III also summarizes the design space, including the candidate waypoints and their corresponding guardian models for each task.

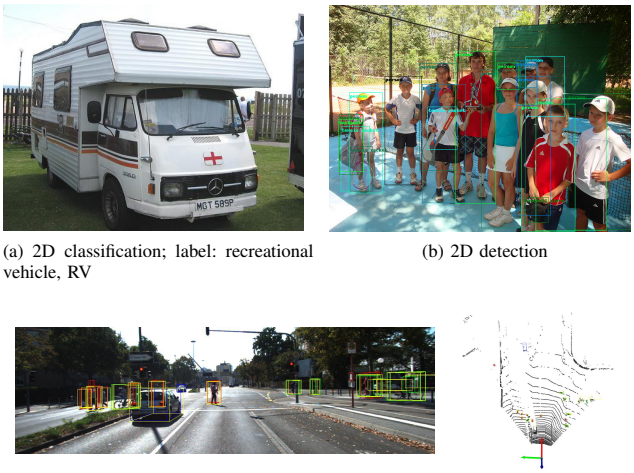
To complement the design space overview, Table IV provides a detailed characterization of the *optimized waypoint plans* generated for each task, as described in Section V. For each waypoint candidate, this table shows the specific prediction model that was selected after our evaluation, along with its key performance metrics.

We evaluate the quality of each predictor using the Negative Predictive Value (NPV). NPV is critical for our system as it measures the reliability of a "no preemption" decision. It is calculated as NPV = TN / (TN + FN), where TN (True Negatives) is the number of times the model correctly predicted that no deadline miss would occur, and FN (False Negatives) is the number of times it incorrectly predicted no deadline miss when one was actually imminent. A high NPV, therefore, directly corresponds to a low rate of un-guarded deadline misses.

The table also reports the resulting empirical preemption rate. This rate, the percentage of inferences the predictor flags for preemption on the validation dataset, serves as the key input (P_p) for our multi-task contention simulation in Section VI. Notably, for earlier waypoints with higher uncertainty (e.g., Task 1 at 3.3ms), more complex hardware-aware models like Multiple Regression were chosen, while for later, more predictable waypoints, simpler models were sufficient.

Compared Strategies: We implement and empirically evaluate the performance of four distinct preemption strategies. Each strategy represents a different philosophy for selecting the combination of waypoint plans for the three tasks. The four strategies are:

1) **Individually Optimal (Greedy Approach):** This baseline represents the most intuitive but naive strategy. It combines the best-performing preemption plan for each task as if it were running in isolation. Based on our single-task analysis, this corresponds to the waypoint combination **(wp3, wp1, wp1)**. This strategy ignores the potential for inter-task contention on the FPGA.
2) **Simulation-Optimized (Our Method):** This is the strategy identified by our proposed methodology. We use the Monte Carlo simulation from Section VI to analyze the full design space of candidate plans, accounting for FPGA contention. The simulation identified the combination **(wp4, wp1, wp1)** as the one that maximizes the expected overall system performance ($\overline{Acc_e}$).
3) **Earliest-Preemption (Heuristic 1):** This strategy forces all tasks to use their earliest possible waypoint: **(wp1, wp1, wp1)**. It serves as a boundary case evaluation, representing a highly aggressive preemption policy.
4) **Latest-Preemption (Heuristic 2):** This strategy forces all tasks to use their latest possible waypoints: **(wp4, wp2, wp1)**. This represents the opposite boundary case: a conservative policy that minimizes preemption attempts.



(a) 2D classification; label: recreational vehicle, RV

(b) 2D detection

(c) KITTI road image and 3D point cloud detection

Fig. 10. Inference examples of three tasks. (a) 2D image classification. (b) 2D detection. (c) KITTI road image and point cloud detection.

2) Results and Analysis: The empirical results of deploying these four strategies are presented in Table V (for effective accuracy) and Table VI (for miss rate and resource utilization). The analysis clearly demonstrates the effectiveness of our system-level optimization.

Overall Effectiveness: First, all three preemption-based strategies dramatically outperform the "Default" (no protection) configuration. The mean $\overline{Acc_e}$ improves from 0.823 to over 0.95, and the deadline miss rates for all tasks are reduced from a baseline of 10-20% to below 3.5%. This confirms that our fundamental approach of using an FPGA guardian is highly effective.

Analysis of Heuristics: The two heuristic strategies reveal the critical trade-offs. The "Earliest-Preemption" (wp1, wp1, wp1) policy, while providing the most time for accurate guardians, suffers from high FPGA contention. This is evidenced by its relatively low preemption success rate (P_{ps}) for Task 1 (86.7%) and consequently higher miss rate (3.4%). Conversely, the "Latest-Preemption" (wp4, wp2, wp1) policy minimizes contention, achieving the highest P_{ps} and lowest miss rates. However, its reliance on less accurate, last-minute guardian models results in a lower overall $\overline{Acc_e}$ than the optimized approaches. This demonstrates that a simple heuristic is insufficient to find the optimal balance.

Benefit of Simulation-Based Optimization: The core result is the comparison between the "Individually Optimal" (greedy) and our "Simulation-Optimized" strategies. While both perform very well, our simulation-optimized plan **(wp4, wp1, wp1)** achieves the highest mean $\overline{Acc_e}$ of 0.963, surpassing the greedy approach.

TABLE III
MULTITASK SYSTEM OVERALL DESIGN SPACE

Task	GPU DNN	Acc_G	D (ms)	Default mr%	WP (ms)	Guardian	Acc_g	$\overline{Acc_g}$	FPGA Latency (ms)
2D classification	ResNet-50	0.762 (Top1 acc)	16	10	3.3	ResNet-50(8-bit)	0.756	0.99	11.7
					6.6	MobileNetv2	0.719	0.95	4.5
					11.6	MobileNetv1	0.678	0.89	2.6
					13.8	MobileNetv1'	0.519	0.68	0.7
2D detection	Faster-RCNN	0.311 (mAP)	111	15	27.0	YoloV3	0.279	0.90	68.0
					48.2	SSDMobileNet	0.220	0.71	6.3
3D detection	PointPillars	0.790 (car3D)	180	20	98.7	PointPillars	0.691	0.87	34.0

Note: Acc_G denotes GPU-side DNN model's accuracy. D denotes deadline. Acc_g and $\overline{Acc_g}$ denote guardian model's accuracy and its normalized value (Acc_g/Acc_G), respectively. Default mr% denotes the default miss rate. MobileNetv1' is a further compressed version of MobileNetv1 to reduce latency.

TABLE IV
CHARACTERIZATION OF OPTIMIZED WAYPOINT PLANS: BEST PREDICTORS AND PERFORMANCE METRICS

Task	**GPU DNN**	**Waypoint (ms)**	**Selected Predictor Model**	**Predictor NPV**[a]	**Preemption Rate (%)**[b]
2D Classification	ResNet-50	3.3	Multiple Regression	0.97	11.3
		6.6	Simple Regression	0.98	9.7
		11.6	Simple Regression	0.99	9.9
		13.8	Simple Regression	1.00	9.9
2D Detection	Faster-RCNN	27.0	Random Forest	0.99	15.6
		48.2	Random Forest	0.99	15.6
3D Detection	PointPillars	98.7	Random Forest	0.99	20.3

[a] Negative Predictive Value (NPV) measures the reliability of a "no miss" prediction.
[b] The Empirical Preemption Rate is the probability (P_p) of a preemption attempt, used in our Monte Carlo simulation.

TABLE V
$\overline{Acc_e}$ RESULTS FOR DIFFERENT PREEMPTION STRATEGIES

Strategy	**Mean $\overline{Acc_e}$**	$\overline{Acc_e}$ per Task		
		Task 1	Task 2	Task 3
Default (No Protection)	0.823	0.900	0.850	0.800
Earliest-Preemption *(wp1, wp1, wp1)*	0.959	0.960	0.968	0.954
Latest-Preemption *(wp4, wp2, wp1)*	0.958	0.965	0.948	0.967
Individually Optimal (Greedy) *(wp3, wp1, wp1)*	0.960	0.970	0.968	0.954
Simulation-Optimized (Ours) *(wp4, wp1, wp1)*	**0.963**	0.957	0.970	0.959

The reason for this subtle but important victory is revealed in Table VI. The greedy approach selects "wp3" for Task 1. Our simulation, however, recognized that Task 1 is the most frequent and chose the even later "wp4". This strategic choice has a key consequence: it drastically reduces the FPGA utilization from Task 1 (from 3.2% down to 0.9%). By intentionally using a less aggressive plan for the most common task, our method frees up the FPGA. This, in turn, slightly increases the preemption success rate (P_{ps}) for the other, longer-running tasks (e.g., for Task 2, from 95.1% to 95.9%). This small gain in reliability for the other tasks, enabled by a system-level view, is what leads to the higher overall system utility. This result validates our central thesis: a global, contention-aware optimization finds superior solutions that a local, greedy approach cannot.

E. Overhead Analysis

Our framework is designed to impose minimal overhead during online operation. We analyze the latency and power consumption overheads when a preemption event is triggered.

1) Online Latency and Power Overhead: Fig. 11 illustrates a detailed breakdown of system overhead for the ResNet-50 example. As shown in Fig. 11(a), the control mechanism's latency impact is negligible. This overhead is composed of two main parts: the inference time of the prediction model and the communication latency for the preemption signal. To isolate and quantify the communication overhead, we conducted a separate experiment measuring the Round-Trip Time (RTT) of the SSH signal over our direct, point-to-point Ethernet link across 1,000 trials. The one-way trigger latency is then estimated as half of this RTT. While this RTT/2 approximation may not be perfectly accurate, it provides a reliable and conservative estimate for a dedicated link. Our measurements yielded a mean one-way latency of 0.58 ms with a low standard deviation of 0.11 ms. This empirically measured sub-millisecond and stable signaling overhead, combined with the prediction model's own sub-millisecond latency, constitutes the negligible control overhead of less than 6%, as shown in Fig. 11(a). This confirms its suitability for our system, especially as this relative overhead becomes even smaller when serving heavier DNN models. We also acknowledge that while SSH was chosen for prototyping flexibility, production

TABLE VI
DETAILED PERFORMANCE METRICS (MISS RATE, FPGA UTILIZATION, PREEMPTION SUCCESS RATE) FOR DIFFERENT STRATEGIES

Strategy	Miss Rate (%)			FPGA Utilization (%)			P_{ps} (%)		
	Task 1	Task 2	Task 3	Task 1	Task 2	Task 3	Task 1	Task 2	Task 3
Earliest-Preemption	3.4	1.7	2.1	14.7	48.7	36.5	86.7	93.3	93.8
Latest-Preemption	0.4	0.8	1.0	1.9	10.6	87.5	95.2	97.3	99.3
Individually Optimal	1.8	1.5	1.9	3.2	55.4	41.4	85.8	95.1	95.0
Simulation-Optimized	1.4	1.5	1.7	0.9	56.9	42.2	85.7	95.9	95.3

deployments could use GPIO or PCIe interrupts for sub-microsecond signaling, further reducing overhead.

Fig. 11(b) demonstrates the energy efficiency of our approach. When activated, the FPGA guardian consumes only 12 W of power, a stark contrast to the GPU's 50 W. By terminating the power-hungry GPU task early and delegating to the efficient FPGA, our system not only guarantees deadlines but also provides significant energy savings in cases where preemption is necessary.

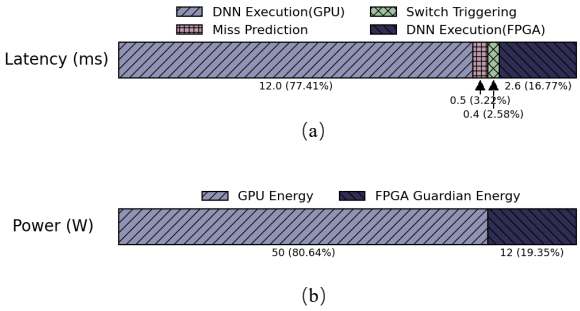


Fig. 11. Breakdown of (a) latency and (b) power consumption overhead in the proposed framework when the FPGA guardian is triggered in the ResNet example.

2) GPU Termination Overhead: A natural concern is whether aborting the GPU inference introduces jitter or stalls that could affect the guardian or subsequent frames. To quantify this, we measured the termination process over 500 trials on the Jetson AGX Orin using ResNet-50. The CPU-side preemption decision, i.e., setting the abort flag and dispatching the FPGA trigger, takes approximately 0.7 μs. This is the only step on the critical path, as the FPGA is activated immediately after on independent hardware. All GPU cleanup proceeds in the background: stream synchronization ($\sim$0.03 ms), cache clearing ($\sim$0.27 ms), and full model teardown ($\sim$4.32 ms, p99: 5.8 ms). Since even the worst-case teardown completes well within a single frame period (16 ms at 60 FPS), it neither introduces jitter to the current guardian execution nor causes cascading delays on subsequent frames.

3) Profiling and Migration Overhead: The offline overhead of our system primarily arises from three components: training guardian models (Section V-A), conducting data collection and training deadline prediction models (Section V-B), and performing Monte Carlo-based simulations (Section VI).

Among these, training guardian models is the most time-consuming, with durations ranging from 1 to 10 GPU hours depending on dataset size and model complexity. While we evaluate multiple architectures for comprehensiveness, in practice, several guardian models can be derived from a single base model using compression techniques such as quantization and pruning. For example, the MobileNetv1 and its compressed variant MobileNetv1' (Table III) are generated through such transformations. Once the base model is trained, additional variants require only minimal calibration using a small sample set.

In contrast, the overhead for data collection, deadline model training, and simulation is significantly lower. For each GPU-side DNN, data across all sequential waypoints can be collected in a single profiling run taking less than 10 minutes. Deadline prediction models are lightweight and train in under 1 minute on a standard desktop CPU. Though device-specific, these tasks adapt quickly with little extra cost when transitioning to new device combinations. Similarly, Monte Carlo simulations are computationally light—100 million steps can be executed in under 5 minutes on a desktop CPU.

When migrating to new hardware or application domains, the overhead remains manageable. Guardian model adaptation relies on compression and calibration rather than retraining. Deadline prediction models remain efficient to retrain. While data collection time scales with device speed, measurements on a constrained platform like Jetson Orin Nano show that profiling time only doubles (to 20 minutes), with no significant increase in total overhead. Public benchmarks such as those from NVIDIA [46] further confirm that system setup remains efficient even on resource-limited devices.

F. Complementarity with GPU Scheduling Methods

We conducted additional experiments to characterize GPU latency variability under several GPU-side optimization strategies that are available on our platform. The purpose of these experiments is not to replicate the full evaluation of our system, but to demonstrate a **general property of GPU execution**: that tail latency persists across scheduling configurations and cannot be eliminated by GPU-side methods alone.

DNN models tested: On the same Jetson platform, we select three models of increasing computational weight to represent the light (ResNet18), medium (ResNet50), and heavy (ResNet101), workload tiers typical in mobile perception pipelines. These models are not identical to those in our main evaluation; the purpose of this experiment is to characterize a *general property of GPU execution behavior* rather than to replicate task-specific results. The solo-execution mean la-

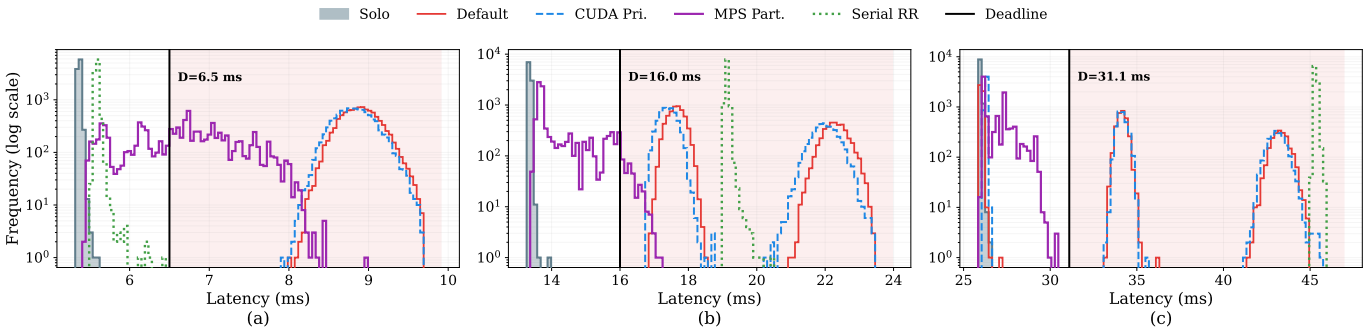


Fig. 12. Latency distributions for each task under five GPU scheduling strategies (overlaid, color-coded). Dashed vertical lines mark deadlines (1.2× solo mean). No single strategy meets all deadlines: concurrent strategies suffer jitter, MPS starves light tasks, and serial execution introduces fatal queuing delays.

TABLE VII
GPU LATENCY STATISTICS AND MISS RATES UNDER FIVE SCHEDULING STRATEGIES.

Strategy	**Task**	**Mean** (ms)	**Std** (ms)	**P99.9** (ms)	**Jitter**[a] (%)	**Miss** (%)
Solo	T1	5.4	0.02	5.5	6	0.0
	T2	13.4	0.03	13.5	5	0.0
	T3	26.0	0.05	26.2	3	0.0
Default	T1	8.9	0.24	9.7	21	100
	T2	19.5	2.31	23.3	34	100
	T3	32.2	6.73	44.6	60	51.9
CUDA Pri.	T1	8.8	0.26	9.8	64	100
	T2	19.3	2.29	24.0	48	100
	T3	32.2	6.56	45.1	71	51.2
MPS Part.	T1	6.7	0.66	8.3	53	63.0
	T2	14.2	0.84	16.9	26	3.6
	T3	26.9	0.85	29.6	16	0.0
Serial RR	T1	5.6	0.03	5.9	16	0.0
	T2	19.1	0.06	19.7	7	100
	T3	45.3	0.10	46.1	5	100

[a] Jitter is calculated as $\frac{\max - \text{mean}}{\text{mean}}$.

tencies are 5.38 ms (Task1_Light), 13.35 ms (Task2_Medium), and 25.96 ms (Task3_Heavy).

Deadline setting: For each task, the deadline is set to 1.2× its solo mean latency, yielding deadlines of 6.5,ms, 16.0,ms, and 31.1,ms, respectively. These settings correspond roughly to real-world execution rates of 150, 60, and 30 frames per second (FPS). This represents a moderate deadline that is easily met in isolation but becomes challenging under contention, a realistic scenario for edge systems.

1) **Solo:** Each task runs alone (zero contention, upper bound).
2) **Default:** All three tasks run concurrently with no isolation.
3) **CUDA priorities:** Tasks assigned stream priorities by deadline tightness (−3 to 0), following the mechanism underlying Pantheon/Orion-style approaches.
4) **MPS partitioning:** NVIDIA MPS enabled with SMs equally partitioned (33% each), emulating GSLICE/MIG-style resource isolation.
5) **Serial round-robin:** Tasks execute sequentially, eliminating all contention at the cost of queuing delay.

Table VII and Fig. 12 reveal a consistent finding: no GPU-only strategy simultaneously eliminates jitter and meets all deadlines. CUDA stream priorities produce distributions virtually identical to the unscheduled default—once large DNN kernels saturate the SMs, dispatch ordering has no effective lever to pull. MPS partitioning is the strongest concurrent option: it tightens Task 2 and Task 3 distributions substantially (Task 3 miss rate drops from 51.9% to 0%). However, the benefit is zero-sum across the fixed SM budget: Task 1, starved to one-third of the SMs, sees its mean rise from 5.4 to 6.7 ms with a 63% miss rate. Serial round-robin eliminates contention-induced jitter entirely, but queuing delays push Task 2 and Task 3 far beyond their deadlines (100% miss rate for both). These five strategies span the GPU-only design space and expose an inescapable trade-off: concurrency introduces jitter or starvation, while serialization cannot meet concurrent deadlines.

These results confirm that our FPGA guardian addresses a fundamentally different layer of the problem. Rather than reshaping the GPU's latency distribution, it monitors each inference and intervenes only when the tail threatens a miss. The two approaches are complementary: deploying our guardian atop MPS would retain MPS's benefits for Task 2 and Task 3 while providing essential protection for Task 1 and for the residual tail events that MPS cannot eliminate.

VIII. CONCLUSION

In conclusion, this paper presents a deadline-guarded real-time and preemptive GPU-FPGA heterogeneous computing approach. To deal with the deadline miss caused by the long-tail latency of DNN execution time, we integrate an FPGA-based computing unit, running a guardian model when the deadline miss is predicted on GPU. We introduce a set of procedures including waypoint and guardian selection, decision model training, and optimal plan searching for online deployment. Moreover, a simulation-based method is further proposed to help select the overall best offloading plan with the multitasking scenario on GPU and potential resource contention on FPGA. Extensive evaluation with a Jetson AGX Orin GPU and a Xilinx ZCU102 FPGA shows that our system achieves up to 20 times reduction on miss rate and improves effective accuracy by more than 10%.

REFERENCES

[1] J. Yi and Y. Lee, "Heimdall: mobile gpu coordination platform for augmented reality applications," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–14.

[2] P. De Backer, C. Van Praet, J. Simoens, M. P. Lores, H. Creemers, K. Mestdagh, C. Allaeys, S. Vermijs, P. Piazza, A. Mottaran *et al.*, "Improving augmented reality through deep learning: real-time instrument delineation in robotic renal surgery," *European Urology*, 2023.

[3] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 751–766.

[4] "Apolloauto/apollo," GitHub, 11 2022. [Online]. Available: https://github.com/ApolloAuto/apollo

[5] Baidu, "Baidu apollo team (2017), apollo: Open source autonomous driving," 2017. [Online]. Available: https://github.com/ApolloAuto/apollo

[6] M. Alcon, H. Tabani, L. Kosmidis, E. Mezzetti, J. Abella, and F. J. Cazorla, "Timing of autonomous driving software: Problem analysis and prospects for future solutions," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 267–280.

[7] I. S. Olmedo, N. Capodieci, and R. Cavicchioli, "A perspective on safety and real-time issues for gpu accelerated adas," in *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2018, pp. 4071–4077.

[8] K. Berezovskyi, F. Guet, L. Santinelli, K. Bletsas, and E. Tovar, "Measurement-based probabilistic timing analysis for graphics processor units," in *Architecture of Computing Systems–ARCS 2016: 29th International Conference, Nuremberg, Germany, April 4–7, 2016, Proceedings 29*. Springer, 2016, pp. 223–236.

[9] S. Heo, S. Cho, Y. Kim, and H. Kim, "Real-time object detection system with multi-path neural networks," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 174–187.

[10] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd international conference on pattern recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.

[11] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger, "Multi-scale dense networks for resource efficient image classification," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:3475998

[12] X. Chen, H. Dai, Y. Li, X. Gao, and L. Song, "Learning to stop while learning to predict," in *International conference on machine learning*. PMLR, 2020, pp. 1520–1530.

[13] A. Zou, Y. Xu, Y. Ni, J. Chen, Y. Ma, J. Li, C. Gill, X. Zhang, and Y. Jin, "A survey of real-time scheduling on accelerator-based heterogeneous architecture for time critical applications," 2025. [Online]. Available: https://arxiv.org/abs/2505.11970

[14] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[15] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[16] E. Cai, D.-C. Juan, D. Stamoulis, and D. Marculescu, "*NeuralPower*: Predict and deploy energy-efficient convolutional neural networks," in *Proceedings of the Ninth Asian Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M.-L. Zhang and Y.-K. Noh, Eds., vol. 77. Yonsei University, Seoul, Republic of Korea: PMLR, 15–17 Nov 2017, pp. 622–637. [Online]. Available: https://proceedings.mlr.press/v77/cai17a.html

[17] L. L. Zhang, S. Han, J. Wei, N. Zheng, T. Cao, Y. Yang, and Y. Liu, "nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices," in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. New York, NY, USA: ACM, 2021, p. 81–93. [Online]. Available: https://doi.org/10.1145/3458864.3467882

[18] N. Capodieci, R. Cavicchioli, M. Bertogna, and A. Paramakuru, "Deadline-based scheduling for gpu with preemption support," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 119–130.

[19] H. Zhou, S. Bateni, and C. Liu, "Sˆ 3dnn: Supervised streaming and scheduling for gpu-accelerated real-time dnn workloads," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 190–201.

[20] N. Ling, K. Wang, Y. He, G. Xing, and D. Xie, "Rt-mdl: Supporting real-time mixed deep learning tasks on edge platforms," in *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, 2021, pp. 1–14.

[21] A. Dhakal, S. G. Kulkarni, and K. K. Ramakrishnan, "Gslice: controlled spatial sharing of gpus for a scalable inference platform," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, ser. SoCC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 492–506. [Online]. Available: https://doi.org/10.1145/3419111.3421284

[22] L. Han, Z. Zhou, and Z. Li, "Pantheon: Preemptible multi-dnn inference on mobile edge gpus," in *Proceedings of the 22nd Annual International Conference on Mobile Systems, Applications and Services*, ser. MOBISYS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 465–478. [Online]. Available: https://doi.org/10.1145/3643832.3661878

[23] F. Strati, X. Ma, and A. Klimovic, "Orion: Interference-aware, fine-grained gpu sharing for ml applications," in *Proceedings of the Nineteenth European Conference on Computer Systems*, ser. EuroSys '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1075–1092. [Online]. Available: https://doi.org/10.1145/3627703.3629578

[24] S. Subramaniyan, R. Joshi, X. Wang, and M. Brocanelli, "Seeb-gpu: Early-exit aware scheduling and batching for edge gpu inference," in *Proceedings of the Tenth ACM/IEEE Symposium on Edge Computing*, ser. SEC '25. New York, NY, USA: Association for Computing Machinery, 2025. [Online]. Available: https://doi.org/10.1145/3769102.3772715

[25] C. Hao, A. Sarwari, Z. Jin, H. Abu-Haimed, D. Sew, Y. Li, X. Liu, B. Wu, D. Fu, J. Gu, and D. Chen, "A hybrid gpu + fpga system design for autonomous driving cars," in *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*, 2019, pp. 121–126.

[26] Y. Tu, S. Sadiq, Y. Tao, M.-L. Shyu, and S.-C. Chen, "A power efficient neural network implementation on heterogeneous fpga and gpu devices," in *2019 IEEE 20th international conference on information reuse and integration for data science (IRI)*. IEEE, 2019, pp. 193–199.

[27] S. Bauer, S. Köhler, K. Doll, and U. Brunsmann, "Fpga-gpu architecture for kernel svm pedestrian detection," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, 2010, pp. 61–68.

[28] A. J. Calderón, L. Kosmidis, C. F. Nicolás, F. J. Cazorla, and P. Onaindia, "Understanding and exploiting the internals of gpu resource allocation for critical systems," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019.

[29] L. Han, Z. Zhou, and Z. Li, "Pantheon: Preemptible multi-dnn inference on mobile edge gpus," 2024.

[30] Advanced Micro Devices, Inc., "Vitis AI User Guide (UG1414)," Advanced Micro Devices, Inc., Tech. Rep., July 2023, corresponds to Vitis AI version 3.5.

[31] C. Ma, N. Wang, Q. A. Chen, and C. Shen, "Slowtrack: Increasing the latency of camera-based perception in autonomous driving using adversarial examples," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

[32] H. Liu, Y. Wu, Z. Yu, Y. Vorobeychik, and N. Zhang, "Slowlidar: Increasing the latency of lidar-based detection using adversarial examples," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 5146–5155.

[33] E.-C. Chen, P.-Y. Chen, I. Chung, C.-R. Lee *et al.*, "Overload: Latency attacks on object detection for edge devices," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[34] C. Ma, N. Wang, Z. Zhao, Q. A. Chen, and C. Shen, "Slowperception: Physical-world latency attack against visual perception in autonomous driving," *arXiv preprint arXiv:2406.05800*, 2024.

[35] F. Restuccia and A. Biondi, "Time-predictable acceleration of deep neural networks on fpga soc platforms," in *2021 IEEE Real-Time Systems Symposium (RTSS)*, 2021, pp. 441–454.

[36] A. Boutros and V. Betz, "Fpga architecture: Principles and progression," *IEEE Circuits and Systems Magazine*, 2021.

[37] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the*

IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 2117–2125.

[38] 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*, 2016, pp. 770–778.

[39] N. Zmora, G. Jacob, L. Zlotnik, B. Elharar, and G. Novik, "Neural network distiller: A python package for dnn compression research," October 2019. [Online]. Available: https://arxiv.org/abs/1910.12232

[40] M. Wang, J. Mo, J. Lin, Z. Wang, and L. Du, "Dynexit: A dynamic early-exit strategy for deep residual networks," *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*, pp. 178–183, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:212646543

[41] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf

[42] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 12 697–12 705.

[43] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Doll'a r, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[44] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[45] N. Rane, "Yolo and faster r-cnn object detection for smart industry 4.0 and industry 5.0: applications, challenges, and opportunities," *Available at SSRN 4624206*, 2023.

[46] N. Corporation, "Nvidia jetson benchmarks," 2024, accessed: 2024-12-09. [Online]. Available: https://developer.nvidia.com/embedded/jetson-benchmarks