Linking High-Level Synthesis with FPGA Runtime Orchestration

Despoina Tomkou
Microprocessors and Digital Systems Laboratory, ECE, NTUA, Athens, Greece

Aggelos Ferikoglou
Microprocessors and Digital Systems Laboratory, ECE, NTUA, Athens, Greece

Dimosthenis Masouros
Microprocessors and Digital Systems Laboratory, ECE, NTUA, Athens, Greece

Sotirios Xydis
Microprocessors and Digital Systems Laboratory, ECE, NTUA, Athens, Greece

Dimitrios Soudris
Microprocessors and Digital Systems Laboratory, ECE, NTUA, Athens, Greece

Abstract

FPGAs are increasingly being adopted across the edge-to-cloud continuum due to their ability to provide both high performance and energy efficiency. However, the complexity of programming FPGAs often leads to deployed designs that underutilize available resources. FPGA multi-tenancy has been proposed to enhance resource utilization, yet monolithic designs and dynamic workload demands continue to challenge efficient FPGA usage and compliance with Quality of Service requirements. To address these issues, we propose a novel framework for the optimal orchestration of FPGAs across the edge-to-cloud continuum while meeting user demands. The framework generates approximations of Pareto-optimal designs for each application, capturing trade-offs between performance and resource usage with minimal bitstream generation. This information allows the runtime orchestrator to select the most suitable design based on available PR regions and the QoS requirements of each user. Experimental results demonstrate that the proposed approach achieves an average reduction of QoS violations by a factor of 8.1× across diverse workloads and baseline configurations. Overall, the framework offers a practical and effective solution for realizing FPGA-as-a-Service across the edge-to-cloud continuum.

2012 ACM Subject Classification Computing methodologies; Computer systems organization → Cloud computing; Computer systems organization → Heterogeneous (hybrid) systems; Hardware → Emerging architectures

Keywords and phrases FPGA, Orchestration, Partial Reconfiguration, FPGAaaS

Digital Object Identifier 10.4230/OASIcs.PARMA-DITAM.2026.7

Supplementary Material
Software (Source Code): `https://github.com/aferikoglou/FPGAScheduler` [13]
archived at `swh:1:dir:2941cf80f4a5b396622da5ebb3f1fb76ffae7b8b`

Funding This work has been partially funded by the EU Horizon 2022 program under grant agreement No 101096698 REFMAP (`https://www.refmap.eu/`).

1 Introduction

The rapid growth and increasing capabilities of IoT devices, together with the extensive deployment of high-speed 5G networks, have led to a massive rise in data generation. These technological advancements are driving a diverse range of applications, including autonomous vehicles [20], smart urban infrastructures [2], and intelligent industrial systems [32]. Such

applications demand ultra-low latency, high reliability, and real-time data processing, requirements that traditional cloud-based architectures alone cannot effectively fulfill. To address these challenges, computing resources are being positioned closer to the data sources, leading to the emergence of edge computing. As a result, multi-layered architectures are being developed to distribute computational resources and applications seamlessly from the network edge to centralized cloud platforms. This integrated approach establishes the edge-cloud computing continuum, enabling efficient workload distribution, reduced latency, and improved resource utilization across a wide range of application domains [31].

Although edge-cloud architectures expand computing capabilities by providing access to a large pool of additional resources, conventional CPUs often fall short for fast, near real-time processing. To address this limitation, hardware accelerators such as GPUs, FPGAs, and ASICs are increasingly integrated across the computing continuum. GPUs, with thousands of small cores enabling massive parallelism, are particularly effective for computationally intensive tasks like Deep Learning [5], but they come with high power consumption and can be inefficient on workloads with complex parallelism [26]. In contrast, ASICs deliver outstanding performance and energy efficiency for specific tasks through customized circuit designs, though they involve high development costs and lack flexibility after fabrication [11]. FPGAs provide a middle ground between these approaches, offering hardware acceleration while retaining post-deployment reconfigurability [30]. Their flexible architecture allows custom implementations of critical operations, from general-purpose datapaths to specialized components such as attention mechanisms and matrix operations in transformer-based models [7], achieving both high performance and energy efficiency.

The increasing adoption of FPGAs is evident from their integration into the infrastructures of major cloud providers. Microsoft's Project Brainwave [25], an FPGA-based real-time AI platform, delivers pre-trained DNN models as online services and operates on Catapult-enhanced servers [27]. This integration illustrates how vendors leverage FPGAs to accelerate large-scale workloads efficiently while preserving the flexibility needed to support evolving AI applications. Beyond serving as accelerators for computationally intensive tasks within provider infrastructures, FPGAs are increasingly offered as a service. Under the FPGA-as-a-Service (FPGAaaS) paradigm, they are made available as computing resources, allowing users to allocate, program, and deploy them through existing management frameworks for on-demand hardware acceleration [1]. For instance, Amazon provides FPGAs as on-demand cloud resources [3], illustrating the growing adoption FPGAaaS.

Despite the advantages offered by the FPGAaaS paradigm, significant challenges remain in efficiently utilizing available FPGA resources. From the user's perspective, programmability continues to be a major obstacle. Unlike traditional computing resources, FPGAs require substantial manual effort and specialized expertise to achieve efficient designs [9]. Although High-Level Synthesis (HLS) tools simplify development, they still demand considerable hardware knowledge to achieve effective performance optimization [28]. As shown by Chi et al. [8], insufficient optimization can even cause FPGA accelerators to perform worse than CPUs. Moreover, exploring the vast FPGA design space is constrained by the lengthy hardware generation process, making comprehensive optimization both time-consuming and resource-intensive [10]. The use of unoptimized designs also poses challenges for providers, as inefficient workloads lead to under-utilization of FPGA resources and, consequently, effective under-provisioning. To enhance utilization, providers often employ multi-tenancy, allowing multiple users to share FPGA resources [1]. However, this approach introduces additional complexities. In a multi-tenant environment, the concurrent sharing of limited resources presents major challenges to satisfying users' Quality of Service (QoS) requirements. Efficient

orchestration of these workloads must also adapt to dynamic and fluctuating demands while maintaining effective resource allocation. Together, these factors make the delivery of FPGAaaS a highly complex and demanding task.

Over the years, numerous solutions have been proposed to streamline the FPGA development process and enable more efficient sharing of FPGA resources. Within the context of HLS, many studies have focused on automating the identification of optimal directive configurations for a given design. Traditional approaches explore the configuration space by evaluating Quality of Result (QoR) metrics obtained from HLS-synthesized designs [34], while modern data-driven methods leverage knowledge from previously optimized designs to guide configurations for unseen applications [15]. Other techniques aim to accelerate the design process by building predictive models that estimate QoR metrics for unexplored designs [24]. Despite these advances, most existing approaches focus on higher-level design aspects, limiting their applicability for rapid hardware generation and deployment in real-world scenarios. In the context of FPGAaaS, Partial Reconfiguration (PR) [36] has emerged as a key enabler of FPGA multi-tenancy, allowing multiple users to share resources and improve overall utilization. In this paradigm, the FPGA is partitioned into multiple PR regions, each functioning as a virtual FPGA to which a user can allocate their custom hardware design. PR-based virtualization permits idle fabric to be reassigned to other users, effectively transforming the FPGA into a multi-tenant device. PR regions may be configured as asymmetric [41], to accommodate hardware modules of varying sizes, or symmetric [6] with fixed dimensions. However, the hardware designs provided by users or obtained from FPGA-accelerated libraries are typically monolithic, requiring specific resources that may not be available at deployment, which can result in violations of user requirements until the necessary resources become available.

In this work, we propose a novel framework for the efficient utilization of multi-tenant FPGAs. Its primary objective is to minimize QoS violations for applications deployed on the FPGA while ensuring optimal use of device resources. Our framework generates, for each deployed application, an approximation of the Pareto-optimal designs representing different trade-offs between performance and resource usage. This is achieved through a modeling approach capable of constructing the Pareto front with a minimal number of hardware generations. At runtime, the orchestrator selects the most appropriate design based on the available PR regions to meet the application's QoS requirements. Experimental results demonstrate that our framework reduces QoS violations by 8.1× on average across various baseline mechanisms and workloads with differing arrival rates and resource demands, underscoring its effectiveness in realizing the FPGAaaS paradigm.

2 Related Work

Recent years have seen significant research efforts dedicated to improving FPGA utilization, addressing both the challenges faced by users in developing efficient accelerators and by providers in managing FPGAs as shared computational resources. These efforts can be broadly classified into two categories: i) *programming-oriented approaches*, which focus on automating the identification of optimal HLS directives and streamlining the design space exploration process, and ii) *orchestration-oriented approaches*, which aim to efficiently share FPGA resources among multiple users, with a particular emphasis on techniques based on partial reconfiguration.

Programming-oriented Approaches. The exploration of directive design spaces has long been a major challenge in the field of HLS. Early approaches sought to efficiently navigate the directive configuration space by analyzing QoR metrics obtained from HLS tools, employing techniques such as simulated annealing [22], evolutionary algorithms [16], and particle swarm optimization [29]. Other studies introduced specialized heuristics to enhance the search process [37]. More recent frameworks have extended these efforts by applying source-level transformations through modern compiler infrastructures such as MLIR [40] or custom compilers built on top of HLS [34]. Nevertheless, achieving accurate results in these methods typically requires invoking the HLS tool to obtain QoR metrics, which is a time-consuming process that can take several minutes per design. To address this limitation, data-driven approaches have emerged, leveraging knowledge from large collections of previously optimized designs to guide configurations for unseen applications [14, 15]. Despite these advancements, most existing methods remain focused on high-level design abstraction, reducing their applicability for rapid hardware generation and deployment in practical orchestration scenarios.

Another research direction focuses on replacing the HLS process itself by developing predictive models trained on large collections of existing designs. During exploration, these models are queried to estimate QoR metrics, greatly reducing the need for time-consuming synthesis runs. The increasing availability of design data has enabled the use of various ML methods, including XGBoost [10], ANNs [10, 23], and CNNs [21], for predicting latency, resource utilization, and power consumption on target FPGAs. Recently, GNNs [35, 33] have attracted attention for their ability to capture the structural characteristics of HLS applications when predicting QoR metrics. Despite their improved representational power, most of these approaches remain limited to HLS estimations, which often diverge from the actual hardware performance in terms of latency, resource usage, and power consumption of the implemented design [10]. Bridging this gap between high-level prediction accuracy and low-level hardware realization remains a critical challenge for enabling truly efficient and deployable FPGA design automation frameworks.

Orchestration-oriented Approaches. The increasing density and capacity of modern FPGAs have made it possible to virtualize devices effectively, enabling multiple users to share resources efficiently. Partial Reconfiguration (PR) is a widely used technique for achieving this goal, as it divides an FPGA into several reconfigurable regions, each functioning as a virtual FPGA. Research on PR-based FPGA partitioning can generally be divided into asymmetric and symmetric approaches [36]. Asymmetric PR regions provide enhanced flexibility by supporting hardware modules of different sizes without requiring full device reconfiguration. One representative work in this category is hCODE2.0 [41], an open-source FPGAaaS platform that leverages asymmetric PR to enable flexible multi-tenancy. The framework simplifies accelerator design and deployment while incorporating an intelligent scheduler that optimizes performance and utilization by considering both resource availability and communication bandwidth.

In contrast, symmetric PR regions, often referred to as resource slots or tiled regions, have fixed and predefined sizes that cannot be altered at runtime. Adjacent regions can be combined to host larger accelerators, helping to reduce internal fragmentation and improve overall resource utilization. Several research frameworks and commercial platforms have adopted symmetric PR partitioning to enable FPGAaaS. Chen et al. [6] proposed a cloud FPGA framework that divides devices into symmetric PR regions for accelerator deployment, achieving low virtualization latency, though limited by PCIe bandwidth constraints. The

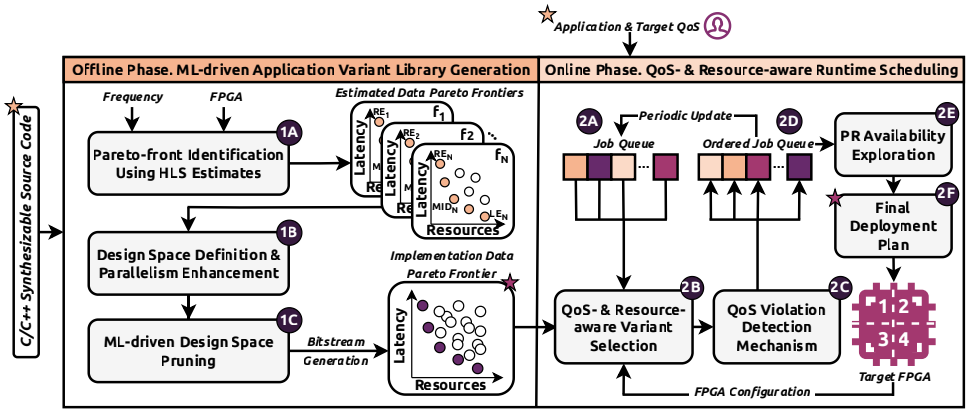

Figure 1 Overview of the Offline and Online Phases of the Proposed Framework.

RC3E FPGA hypervisor [19] allows users to deploy custom hardware designs through full or partial FPGA allocation, improving utilization while maintaining strict security controls. Similarly, a datacenter-oriented reconfigurable accelerator framework [12] connects FPGAs via PCIe and leverages a hypervisor to manage PR regions, effectively reducing redundant reconfigurations. Commercial solutions such as Amazon's EC2 F1 instances [3] extend this model by providing reusable FPGA images and development toolkits for hardware and software developers, while Accelize QuickStore introduces a pay-per-use marketplace for third-party FPGA accelerators and IP cores. Despite these advancements, most accelerators, whether user-designed or sourced from FPGA libraries, remain monolithic in nature, requiring fixed resource configurations. This inflexibility can limit deployment adaptability, potentially leading to QoS violations when suitable resources are not immediately available at runtime.

3 Proposed Scheduling Framework

The proposed methodology enables efficient orchestration of FPGA resources across the edge-to-cloud continuum. Its central objective is to satisfy application-level QoS constraints by minimizing QoS violations while simultaneously maximizing utilization of the available FPGA resources. As illustrated in Figure 1, the framework operates in two complementary phases: (i) the *ML-driven Application Variant Library Generation (Offline)* phase, detailed in Section 3.1, and (ii) the *QoS- & Resource-aware Runtime Scheduling (Online)* phase, presented in Section 3.2.

3.1 Offline Phase: ML-driven Application Variant Library Generation

The objective of the offline phase is to take a given HLS-based application with a synthesizable C/C++ kernel and generate a library of hardware variants, each offering different trade-offs between latency and resource usage. This library of bitstreams serves as a foundation for the online orchestrator, enabling informed design selection at runtime.

The process begins by approximating the Pareto frontier of the computationally intensive kernel 1A. Given a set of target operating frequencies and the chosen FPGA device, this stage generates the latency–resource Pareto frontier for each frequency configuration. This step relies on pre-synthesis information, specifically the reports generated during the HLS process. The rationale for this decision stems from the substantial body of prior work capable

of efficiently deriving Pareto-optimal configurations at the HLS level, as discussed in Section 2. Moreover, publicly available design collections provide Pareto-optimal configurations for well-studied applications, allowing us to directly leverage these results without performing full design-space exploration from scratch. We realize this step using the GNΩSIS dataset [18], the largest HLS design collection currently available, comprising implementations from over 55 applications drawn from established benchmark suites and open-source repositories. For applications not included in the dataset, we employ an NSGA-II–based optimization engine [17] that automatically derives the latency–resource Pareto front for the specified frequency and target FPGA.

The next phase of the methodology focuses on identifying the subset of designs that will be physically implemented to build the application variant library 1B. From each Pareto frontier generated across the evaluated operating frequencies, we select three representative design points: (i) the latency-optimized configuration (LE), (ii) the resource-optimized configuration (RE), and (iii) an intermediate design balancing both objectives (MID). Instead of implementing the entire Pareto set, we choose only these key configurations for two reasons. First, synthesizing and implementing every Pareto candidate would lead to excessive compilation time and computational overhead [24]. Second, offering the runtime scheduler clearly differentiated design points provides meaningful flexibility when mapping workloads to available FPGA resources. Beyond the directive-based variants, we further enrich the design space by incorporating coarse-grained parallelism. For each selected design at each operating frequency, we instantiate multiple Compute Units (CUs), meaning multiple copies of the kernel on the Programmable Logic (PL). This simple yet effective strategy can significantly reduce execution latency when sufficient hardware resources are available. To ensure feasibility, we scale the number of CUs and keep only those configurations whose estimated resource usage, as reported by the HLS tool, fits within the target FPGA.

Introducing CU exploration significantly increases the number of designs that must be physically generated. For example, consider a scenario where only a single operating frequency is evaluated. Assume the LE configuration can support 1, 2, 4, and 6 CUs, the MID configuration can support 1, 2, 4, 6, and 8 CUs, and the RE configuration can support 1, 2, 4, 6, 8, and 10 CUs on the target FPGA. In this case, the design space expands to a total of 15 variants that must be implemented. Additionally, prior research has shown that pre-synthesis estimates often differ from post-implementation results [10], meaning that the actual resource utilization of a multi-CU design may vary, and some configurations may even be not possible to generate. To address this problem, step 1C introduces two mechanisms that leverage post-implementation data from a single CU, along with a small subset of the design space, to assess the feasibility of multi-CU configurations on the FPGA and to accurately predict the design's latency as well as the utilization of BRAMs, DSPs, FFs, and LUTs. This step enables the prediction of feasibility, latency, and resource usage for the remaining configurations. Further details can be found in the paragraph titled "Feasibility Classifier and Performance–Resource Modeling". Using these predictions, the actual Pareto frontier can be identified, allowing only the Pareto-optimal designs to be implemented and thereby significantly reducing the number of required implementations. The resulting Pareto frontier subsequently serves as the application variant library for the online phase.

Feasibility Classifier & Performance–Resource Modeling. To train our models, we utilize applications from the GNΩSIS dataset [18]. Specifically, we use several applications such as KNN and KMeans and obtain their Pareto frontiers at 100, 200, and 300 MHz on the Alveo U200 FPGA. For each target frequency, we select representative designs including LE, MID,

and RE, and expand the design space by incorporating multiple CUs. Each variant is implemented, and for every valid design, we collect post-implementation metrics including latency, BRAM, DSP, FF, and LUT utilization. For each application, the collected implementation data are used to train both the feasibility classifier and the performance–resource models. We build upon established approaches from the literature [10], employing a Decision Tree Classifier (DTC) for feasibility prediction, Linear Regression (LR) for latency estimation, and XGBoost Regression (XGB) for predicting BRAMs, DSPs, FFs, and LUTs. Model performance for each application is evaluated using 5-fold cross-validation, applying the R^2 metric for regression tasks and the F_1 score for classification. After feature engineering and hyperparameter tuning, DTC achieves an average F_1 score of 0.967, while the LR model attains an R^2 of 0.97. The XGB-based models also exhibit strong predictive capability, achieving R^2 scores of 0.99 for BRAMs, 0.98 for DSPs, 0.97 for FFs, and 0.97 for LUTs. Having established high accuracy across all models, we reduce the dataset to identify the minimal portion of the design space that must be implemented beyond the single-CU configurations. The remaining points are selected via random sampling. Our results indicate that utilizing only 20% of the design space leads to an average accuracy degradation of less than 3% across all models. These findings demonstrate that the proposed models can be effectively trained using a small subset of the design space, substantially decreasing the number of bitstream generations required to construct the application variant library.

3.2 Online Phase: QoS- & Resource-aware Runtime Scheduling

The objective of the online phase is to manage a stream of FPGA-accelerated applications, each with a user-defined execution-time requirement, in order to minimize QoS violations while efficiently utilizing the available FPGA resources. The scheduler maintains a queue 2A, where each entry represents an application along with its maximum acceptable execution time. For every queued job, the system tracks both the specified execution deadline and the time elapsed since the job entered the queue. Based on this information, it computes the remaining allowable time before a QoS violation occurs, defined as the difference between the target execution time and the accumulated waiting time. As the job waits longer, this remaining time decreases, signaling increased urgency for scheduling. This parameter is continuously updated and used to guide job prioritization and reduce QoS violations.

Step 2B is responsible for selecting the most suitable application variant to meet the target application's requirements. It operates using the application variant library generated during the offline phase, along with knowledge of the current FPGA configuration. Here, the FPGA configuration refers to the number of available PR regions. In this work, we assume symmetric PR regions, in line with the prevailing practice in literature [1]. The FPGA is divided into these PR regions such that the total resources are evenly distributed, giving each region an equal share of the device's resources. To identify the best candidate, this step first verifies that the application variant can fit within the available PR regions. Specifically, it excludes any variants that require more BRAM, DSP, FF, or LUT resources than are available in a single PR region. Among the remaining bitstreams, it further eliminates any variants that cannot meet the QoS constraint based on the remaining allowable time, as these would result in a violation. From the filtered set of candidates, the scheduler then selects the variant with the shortest execution time to maximize performance while ensuring compliance with the QoS requirements.

If none of the available application variants that fit within a single PR region can satisfy the QoS constraint, the QoS violation detection mechanism is triggered 2C. This step identifies jobs that cannot be accommodated under the current conditions and temporarily

de-prioritizes them by placing them at the end of the queue, allowing feasible jobs to be scheduled first. Once the urgency scores for all applications are computed, the queue is reordered so that the most urgent jobs are processed first, forming the ordered job queue 2D. When a job becomes ready for execution, the framework explores the available PR regions 2E to determine whether a suitable region is available for allocation. If a region is available, the application is assigned to it; otherwise, the job remains in the queue until a region becomes free. Once assigned, the job is mapped to the corresponding PR region of the FPGA, finalizing the deployment plan 2F.

4 Experimental Evaluation

In this section, we present the experimental evaluation of the proposed framework. We begin by describing the experimental setup and the baseline mechanisms used for comparison, followed by a detailed presentation and analysis of the obtained results and the impact of the proposed approach.

4.1 Experimental Overview & Baselines Description

We assess the efficiency of the proposed scheduling framework through a comprehensive set of experiments designed to capture realistic workload dynamics and system behaviors. The experimental workloads are derived from applications in the GNΩSIS dataset [18], focusing on KNN and KMeans, which are representative algorithms widely used in large-scale data analytics and real-time processing and are particularly well suited for FPGA acceleration [38]. All implementations were developed using the AMD Vitis Unified Software Platform 2021.1 [4] and executed on the Alveo U200 FPGA. The experiments were conducted on a Linux-based server equipped with an Intel Xeon Silver 4210 CPU @ 2.20 GHz, featuring 32KB of L1D/I, 1MB of L2, and 14MB of L3 cache. To evaluate Partial Reconfiguration (PR), we employed a simulation-based approach instead of deploying physical bitstreams, providing the flexibility to explore a broad range of partitioning schemes and FPGA configurations. T3he experiments involve a set of 30 applications, each defined with specific QoS constraints representing maximum allowable execution times. Application demands were modeled by aggregating the execution times of all available design variants and selecting the 25%-ile values, emulating a high-load operational scenario. Application arrivals were generated using a bursty distribution derived from Alibaba accelerator cluster traces [39], capturing realistic cloud workload patterns where requests arrive in concentrated bursts. This setup ensures a realistic yet analytically manageable representation of production-scale system conditions.

The proposed framework is evaluated along two complementary dimensions: (a) the scheduling strategy, which determines the order in which applications are executed, and (b) the management strategy, which governs bitstream selection and FPGA partitioning through PR. At the scheduling level, the proposed scheduler is compared against a conventional FIFO scheduler that serves applications strictly in their order of arrival without accounting for workload characteristics or system state. This baseline provides a simple yet informative reference point for assessing the benefits of the proposed policy under dynamic and heterogeneous workloads. At the management level, four configurations are examined to capture the effects of Pareto-awareness and FPGA partitioning. The first configuration (`M1`) represents a Pareto-agnostic manager without access to the Pareto frontier. It consistently selects a fixed bitstream that, on average across applications, is 30% slower and utilizes 24% more resources than the MID design, emulating the behavior of a non-expert user choosing a

suboptimal configuration. The FPGA operates as a single monolithic region in this setup. The second configuration (M2) introduces Pareto-awareness, enabling the manager to select implementations from the Pareto frontier while still using a single FPGA region. The third configuration (M3) combines Pareto-awareness with PR by dividing the FPGA into two reconfigurable regions, allowing concurrent execution of multiple Pareto-optimal designs. The fourth configuration (M4) further extends this concept by partitioning the FPGA into four PR regions, enhancing parallelism and flexibility. The number of PR regions is limited to four, as further partitioning would excessively constrain the available hardware area, preventing many bitstreams from fitting within smaller regions and reducing the system's practical usability. The performance of each configuration is evaluated using four key metrics: (a) total number of QoS violations, (b) average waiting time, representing the duration an application remains in the queue before execution, (c) end-to-end execution time, which includes waiting time, execution time, and scheduling overhead, and (d) average excess time, defined as the mean duration by which a job exceeds its QoS target. For jobs meeting their QoS constraints, the excess time is clipped to zero.

4.2 Detailed Analysis of Proposed Framework versus Defined Baselines

We analyze the system behavior under two workload scenarios. The first corresponds to a homogeneous case, where a single application, in this case KNN, is scaled across multiple executions, while the second represents a heterogeneous case, where multiple distinct applications are executed concurrently.

4.2.1 Homogeneous Workload Evaluation

Figure 2 presents the results of the homogeneous workload experiment. Focusing on manager M1, which represents the Pareto-agnostic configuration where the FPGA operates as a single region, we can observe the baseline impact of the proposed scheduling policy. The proposed scheduler results in 22 QoS violations compared to 27 under FIFO. In terms of end-to-end execution time, it achieves a distribution ranging from 31 seconds to 13.3 minutes, with an average of 7.1 minutes. The waiting time exhibits a range from 2 milliseconds to 12.7 minutes, averaging 6.6 minutes. Overall, the proposed scheduler achieves about a 1% reduction in both end-to-end execution and waiting times, with a modest 2% increase in average excess time. These results demonstrate that even at the scheduling level alone, prioritizing jobs with tighter deadlines effectively reduces violations and improves timing efficiency. When examining manager M2, the benefits of Pareto-awareness become more pronounced. Under this configuration, the proposed scheduler records 18 QoS violations, improving upon its performance in M1. The end-to-end execution time distribution ranges from 23 seconds to 8.9 minutes, with an average of 4.9 minutes, while waiting times span from 2.7 milliseconds to 8.6 minutes, averaging 4.5 minutes. This corresponds to a 1.45× reduction in both end-to-end and waiting times, accompanied by a 1.6× decrease in average excess time. These improvements underscore the value of Pareto-awareness, as M2 leverages latency-optimized designs from the Pareto frontier, resulting in more efficient execution and a substantial reduction in QoS violations.

Next, we evaluate the performance of the M3 and M4 managers, which leverage PR to divide the FPGA into multiple independent regions. As expected, increasing the number of PR regions reduces the number of QoS violations, since more requests can be executed concurrently. In the proposed approach, M3 results in 13 violations, whereas M4 further decreases this number to only 3, demonstrating the clear advantages of finer-grained reconfig-

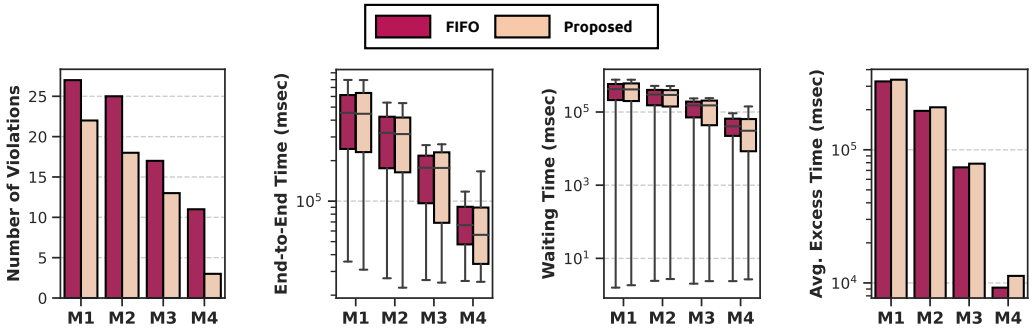

Figure 2 Evaluation of the proposed approach under homogeneous workloads, showing (Left) the number of QoS violations, (Middle Left) the distribution of end-to-end execution time, (Middle Right) the distribution of waiting time, and (Right) the average excess time.

uration. For `M3`, the end-to-end execution time ranges from 24.8 seconds to 4.4 minutes, with an average of 2.6 minutes, while the waiting time spans from 2 milliseconds to 4 minutes, averaging 2.2 minutes. In the case of `M4`, the end-to-end time varies between 25 seconds and 2.8 minutes, averaging 1.5 minutes, and the waiting time ranges from 2 milliseconds to 2.4 minutes, with an average of 43 seconds. When compared to `M2`, `M3` achieves a 1.9× improvement in end-to-end execution time, while `M4` achieves a 4.3× improvement. Similarly, the waiting time is reduced by 2.1× with `M3` and by 6.4× with `M4`. The average excess time, which quantifies how long jobs exceed their QoS targets, also decreases significantly – by 2.7× for `M3` and 18.5× for `M4` compared to `M2`. These results demonstrate that increasing the number of PR regions consistently enhances overall system performance. They further highlight the effectiveness of the proposed scheduling framework in exploiting PR to achieve high resource utilization and strong QoS compliance. An additional observation is that the proposed scheduler shows an 8% higher average excess time, even though it achieves fewer QoS violations. This behavior is attributed to the *QoS Violation Detection Mechanism*, which de-prioritizes jobs predicted to miss their deadlines. Consequently, while this strategy helps reduce the number of QoS violations, it also increases the waiting time for certain jobs, slightly raising the average excess time.

Ablation Study. In this section, we conduct an ablation study based on Figure 2 (Left) to examine how each mechanism introduced in the proposed framework contributes to reducing QoS violations. The FIFO scheduler paired with the `M1` manager serves as the baseline for this analysis. Introducing the proposed scheduler, which prioritizes applications that are close to violating their QoS constraints and de-prioritizes those that have already missed them, leads to a 19% reduction in violations compared to the baseline. Adding Pareto-awareness, which enables the manager to identify and select latency-efficient configurations from the Pareto frontier, provides a further 20% decrease in violations. The most substantial improvement is achieved with the `M4` configuration, which incorporates both Pareto-awareness and PR by dividing the FPGA into four independent regions. This setup results in a total reduction of 77% in violations, demonstrating the combined benefits of adaptive scheduling, Pareto-aware management, and fine-grained FPGA partitioning. As expected, exploiting the full potential of partial reconfiguration delivers the greatest performance gains by enabling concurrent execution and improving resource utilization.

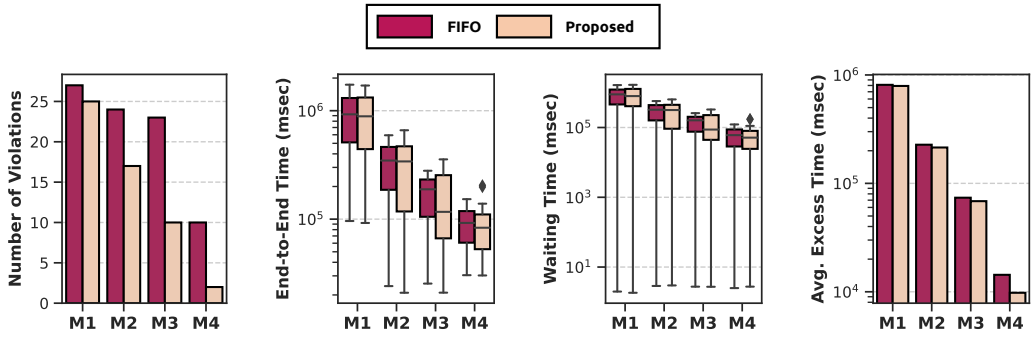

Figure 3 Evaluation of the proposed approach under heterogeneous workloads, showing (Left) the number of QoS violations, (Middle Left) the distribution of end-to-end execution time, (Middle Right) the distribution of waiting time, and (Right) the average excess time.

4.2.2 Heterogeneous Workload Evaluation

For the heterogeneous workload, we evaluate a diverse set of applications, each generating requests with distinct execution completion requirements, providing a more realistic representation of deployment scenarios. Figure 3 presents the results of this experiment. The observed results align with the trends seen in the homogeneous workload, where the proposed scheduler combined with the `M4` manager achieves the fewest QoS violations. Specifically, it results in only 2 violations. The end-to-end execution time distribution ranges from 30 seconds to 3.4 minutes, with an average of 1.5 minutes, while the waiting time varies between 2.7 seconds and 2.9 minutes, averaging 57.3 seconds. The average excess time is measured at 9.8 seconds. In terms of violations, this configuration achieves substantial improvements, showing 13.5×, 8.5×, and 5× fewer violations compared to the FIFO scheduler with `M1` and the proposed scheduler with `M2` and `M3`, respectively. It also outperforms the same baselines in terms of average end-to-end execution time, achieving improvements of 10.6×, 3.6×, and 1.8×, respectively. In terms of waiting time, it demonstrates reductions of 14.9× and 2.3× compared to the corresponding configurations. Furthermore, the average excess time shows an order-of-magnitude reduction relative to the FIFO scheduler with `M1` and `M2`, and it remains 7.6× lower than that of `M3`. These findings indicate that the proposed scheduler, when combined with Pareto-awareness and full utilization of four PR regions, consistently surpasses all baseline approaches, even under realistic heterogeneous workload conditions.

5 Conclusion

This work presented a novel framework designed to enhance the efficiency and flexibility of multi-tenant FPGA utilization across the edge-to-cloud continuum. By integrating Pareto-optimal design approximation and intelligent runtime orchestration, the proposed approach effectively balances performance and resource usage while minimizing QoS violations. Unlike conventional static or monolithic deployment strategies, our framework dynamically adapts to workload and resource variability, leveraging partial reconfiguration to ensure efficient sharing of FPGA resources among concurrent applications. Experimental evaluation shows an average 8.1× reduction in QoS violations across diverse workloads and baseline mechanisms, confirming the effectiveness of the proposed framework in satisfying user requirements under dynamic operating conditions.

References

1 Lamees M Al Qassem, Thanos Stouraitis, Ernesto Damiani, and Ibrahim M Elfadel. Fpgaaas: A survey of infrastructures and systems. *IEEE Transactions on Services Computing*, 15(2):1143–1156, 2020.

2 Fadi Al-Turjman, Hadi Zahmatkesh, and Ramiz Shahroze. An overview of security and privacy in smart cities' iot communications. *Transactions on Emerging Telecommunications Technologies*, 33(3):e3677, 2022. doi:10.1002/ETT.3677.

3 Amazon Web Services. Ec2 instances (f1) with programmable hardware. https://aws.amazon.com/blogs/aws/developer-preview-ec2-instances-f1-with-programmable-hardware/, 2016. Accessed: 2025-10-25.

4 AMD Vitis Unified Software Platform. https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis.html, 2025. Accessed: 2025-11-10.

5 Ebubekir Buber and DIRI Banu. Performance analysis and cpu vs gpu comparison for deep learning. In *2018 6th International Conference on Control Engineering & Information Technology (CEIT)*, pages 1–6. IEEE, 2018.

6 Fei Chen, Yi Shan, Yu Zhang, Yu Wang, Hubertus Franke, Xiaotao Chang, and Kun Wang. Enabling fpgas in the cloud. In *Proceedings of the 11th ACM Conference on Computing Frontiers*, pages 1–10, 2014. doi:10.1145/2597917.2597929.

7 Hongzheng Chen, Jiahao Zhang, Yixiao Du, Shaojie Xiang, Zichao Yue, Niansong Zhang, Yaohui Cai, and Zhiru Zhang. Understanding the potential of fpga-based spatial acceleration for large language model inference. *ACM Transactions on Reconfigurable Technology and Systems*, 18(1):1–29, 2024. doi:10.1145/3656177.

8 Yuze Chi, Weikang Qiao, Atefeh Sohrabizadeh, Jie Wang, and Jason Cong. Democratizing domain-specific computing. *Communications of the ACM*, 66(1):74–85, 2022. doi:10.1145/3524108.

9 Jason Cong, Bin Liu, Stephen Neuendorffer, Juanjo Noguera, Kees Vissers, and Zhiru Zhang. High-level synthesis for fpgas: From prototyping to deployment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):473–491, 2011. doi:10.1109/TCAD.2011.2110592.

10 Steve Dai, Yuan Zhou, Hang Zhang, Ecenur Ustun, Evangeline FY Young, and Zhiru Zhang. Fast and accurate estimation of quality of results in high-level synthesis with machine learning. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 129–132. IEEE, 2018.

11 Wei Dai and Daniel Berleant. Benchmarking contemporary deep learning hardware and frameworks: A survey of qualitative metrics. In *2019 IEEE First International Conference on Cognitive Machine Intelligence (CogMI)*, pages 148–155. IEEE, 2019. doi:10.1109/COGMI48466.2019.00029.

12 Suhaib A Fahmy, Kizheppatt Vipin, and Shanker Shreejith. Virtualized fpga accelerators for efficient cloud computing. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 430–435. IEEE, 2015. doi:10.1109/CLOUDCOM.2015.60.

13 Aggelos Ferikoglou. FPGAScheduler. Software, swhId: swh:1:dir:2941cf80f4a5b396622da5ebb3f1fb76ffae7b8b (visited on 2026-02-26). URL: https://github.com/aferikoglou/FPGAScheduler, doi:10.4230/artifacts.25581.

14 Aggelos Ferikoglou, Andreas Kakolyris, Vasilis Kypriotis, Dimosthenis Masouros, Dimitrios Soudris, and Sotirios Xydis. Data-driven hls optimization for reconfigurable accelerators. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6, 2024.

15 Aggelos Ferikoglou, Andreas Kakolyris, Dimosthenis Masouros, Dimitrios Soudris, and Sotirios Xydis. Collectivehls: A collaborative approach to high-level synthesis design optimization. *ACM Transactions on Reconfigurable Technology and Systems*, 18(1):1–32, 2024. doi:10.1145/3702005.

16 Fabrizio Ferrandi, Pier Luca Lanzi, Daniele Loiacono, Christian Pilato, and Donatella Sciuto. A multi-objective genetic algorithm for design space exploration in high-level synthesis. In *2008 IEEE Computer Society Annual Symposium on VLSI*, pages 417–422. IEEE, 2008. doi:10.1109/ISVLSI.2008.73.

17 GenHLSOptimizer GitHub. https://github.com/aferikoglou/GenHLSOptimizer, 2025. Accessed: 2025-11-01.

18 GNWSIS Dataset Hugging Face. https://huggingface.co/datasets/aferikoglou/GNWSIS, 2025. Accessed: 2025-11-01.

19 Oliver Knodel and Rainer G Spallek. Rc3e: provision and management of reconfigurable hardware accelerators in a cloud environment. *arXiv preprint arXiv:1508.06843*, 2015. arXiv:1508.06843.

20 Xhafer Krasniqi and Edmond Hajrizi. Use of iot technology to drive the automotive industry from connected to full autonomous vehicles. *IFAC-PapersOnLine*, 49(29):269–274, 2016.

21 Zhe Lin, Jieru Zhao, Sharad Sinha, and Wei Zhang. Hl-pow: A learning-based power modeling framework for high-level synthesis. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 574–580. IEEE, 2020. doi:10.1109/ASP-DAC47756.2020.9045442.

22 Anushree Mahapatra and Benjamin Carrion Schafer. Machine-learning based simulated annealer method for high level synthesis design space exploration. In *Proceedings of the 2014 Electronic System Level Synthesis Conference (ESLsyn)*, pages 1–6. IEEE, 2014.

23 Hosein Mohammadi Makrani, Hossein Sayadi, Tinoosh Mohsenin, Setareh Rafatirad, Avesta Sasan, and Houman Homayoun. Xppe: cross-platform performance estimation of hardware accelerators using machine learning. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pages 727–732, 2019. doi:10.1145/3287624.3288756.

24 Dimosthenis Masouros, Aggelos Ferikoglou, Georgios Zervakis, Sotirios Xydis, and Dimitrios Soudris. Late breaking results: Language-level qor modeling for high-level synthesis. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–2, 2024. doi:10.1145/3649329.3663500.

25 Microsoft Research. Project brainwave: A deep learning platform for real-time ai inference in the cloud and on the edge. https://www.microsoft.com/en-us/research/project/project-brainwave/, 2018. Accessed: 2025-10-25.

26 Sparsh Mittal and Jeffrey S Vetter. A survey of methods for analyzing and improving gpu energy efficiency. *ACM Computing Surveys (CSUR)*, 47(2):1–23, 2014. doi:10.1145/2636342.

27 Andrew Putnam, Adrian M Caulfield, Eric S Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, et al. A reconfigurable fabric for accelerating large-scale datacenter services. *ACM SIGARCH Computer Architecture News*, 42(3):13–24, 2014.

28 Benjamin Carrion Schafer and Zi Wang. High-level synthesis design space exploration: Past, present, and future. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2628–2639, 2019. doi:10.1109/TCAD.2019.2943570.

29 Anirban Sengupta and Saumya Bhadauria. User power-delay budget driven pso based design space exploration of optimal k-cycle transient fault secured datapath during high level synthesis. In *Sixteenth International Symposium on Quality Electronic Design*, pages 289–292. IEEE, 2015. doi:10.1109/ISQED.2015.7085441.

30 Ahmad Shawahna, Sadiq M Sait, and Aiman El-Maleh. Fpga-based accelerators of deep learning networks for learning and classification: A review. *ieee Access*, 7:7823–7859, 2018. doi:10.1109/ACCESS.2018.2890150.

31 Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016. doi:10.1109/JIOT.2016.2579198.

32 Fadi Shrouf, Joaquin Ordieres, and Giovanni Miragliotta. Smart factories in industry 4.0: A review of the concept and of energy management approached in production based on the internet of things paradigm. In *2014 IEEE international conference on industrial engineering and engineering management*, pages 697–701. IEEE, 2014.

33 Atefeh Sohrabizadeh, Yunsheng Bai, Yizhou Sun, and Jason Cong. Robust gnn-based representation learning for hls. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2023. doi:10.1109/ICCAD57390.2023.10323853.

34 Atefeh Sohrabizadeh, Cody Hao Yu, Min Gao, and Jason Cong. Autodse: Enabling software programmers to design efficient fpga accelerators. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 27(4):1–27, 2022. doi:10.1145/3494534.

35 Ecenur Ustun, Chenhui Deng, Debjit Pal, Zhijing Li, and Zhiru Zhang. Accurate operation delay prediction for fpga hls using graph neural networks. In *Proceedings of the 39th international conference on computer-aided design*, pages 1–9, 2020. doi:10.1145/3400302.3415657.

36 Anuj Vaishnav, Khoa Dang Pham, and Dirk Koch. A survey on fpga virtualization. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 131–1317. IEEE, 2018.

37 Sotirios Xydis, Gianluca Palermo, Vittorio Zaccaria, and Cristina Silvano. SPIRIT: spectral-aware pareto iterative refinement optimization for supervised high-level synthesis. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 34(1):155–159, 2015. doi:10.1109/TCAD.2014.2363392.

38 Feng Yan, Andreas Koch, and Oliver Sinnen. A survey on fpga-based accelerator for ml models. *arXiv preprint arXiv:2412.15666*, 2024. doi:10.48550/arXiv.2412.15666.

39 Lingyun Yang, Yongchen Wang, Yinghao Yu, Qizhen Weng, Jianbo Dong, Kan Liu, Chi Zhang, Yanyi Zi, Hao Li, Zechao Zhang, Nan Wang, Yu Dong, Menglei Zheng, Lanlan Xi, Xiaowei Lu, Liang Ye, Guodong Yang, Binzhang Fu, Tao Lan, Liping Zhang, Lin Qu, and Wei Wang. Gpu-disaggregated serving for deep learning recommendation models at scale. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, USENIX NSDI '25. USENIX Association, 2025. URL: https://www.usenix.org/conference/nsdi25/presentation/yang.

40 Hanchen Ye, Cong Hao, Jianyi Cheng, Hyunmin Jeong, Jack Huang, Stephen Neuendorffer, and Deming Chen. Scalehls: A new scalable high-level synthesis framework on multi-level intermediate representation. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 741–755. IEEE, 2022. doi:10.1109/HPCA53966.2022.00060.

41 Qian Zhao, Motoki Amagasaki, Masahiro Iida, Morihiro Kuga, Toshinori Sueyoshi, et al. hcode 2.0: An open-source toolkit for building efficient fpga-enabled clouds. In *2017 International Conference on Field Programmable Technology (ICFPT)*, pages 267–270. IEEE, 2017.