future internet

MDPI

Article

Hierarchical Scheduler with Adaptive Time-Budget Reallocation for Time-Triggered Edge-Fog-Cloud Architectures

Omar Hekal *, Josepaul Paulachan, Daniel Onwuchekwa and Roman Obermaisser

Chair of Embedded Systems, University of Siegen, Hölderlinstraße 3, 57076 Siegen, Germany
* Correspondence: omar.hekal@uni-siegen.de

Abstract

The lack of determinism restricts the integration of safety-critical applications into Edge–Fog–Cloud (EFC) architectures. Existing EFC schedulers are typically designed for dynamic, best-effort operation based on unmanaged resource allocation and elastic virtualization. This paradigm introduces unbounded queueing, resource contention, and timing jitter, making standard schedulers unsuitable for hard-deadline workloads. Moreover, most approaches focus on computational placement, while communication is abstracted or treated as a secondary cost term. As a result, bounded-latency routing and deterministic task execution are rarely co-optimized under a unified timing model. This paper addresses these gaps by utilizing a managed Time-Triggered Edge–Fog–Cloud (TTEFC) architecture that supports safety-critical workloads, orchestrates IEEE Time-Sensitive Networking (TSN) for local intra-domain communication, and uses IETF Deterministic Networking (DetNet) for routed inter-domain paths. On this infrastructure, a hierarchical genetic algorithm (HGA) is proposed to jointly schedule partition-to-execution-location allocation, partition execution order, inter-partition route selection, and negotiated per-partition time budgets that act as temporal boundaries for parallel partition-level optimizers. An adaptive slack reallocation operator redistributes unused temporal slack from over-satisfied partitions to budget-violating partitions, improving feasibility convergence. Experiments on synthetic DAG workloads with 100–500 tasks compare the proposed HGA against HEFT and round-robin baselines. These baselines are included as scoped external references to contextualize the end-to-end scheduling performance of the proposed method. Ablation results show that slack reallocation improves partition-budget feasibility, reaches feasible budget assignments earlier, and produces tighter budget–makespan alignment than feedback-free and static-budget variants. An automotive-characteristic DAG case study further evaluates the method on an application-oriented workload under the same timing and communication assumptions.

Keywords: scheduling; optimization; cloud scheduling; Edge–Fog–Cloud architectures; time-triggered systems; hierarchical genetic algorithm; adaptive slack reallocation; time-budget negotiation; bounded-latency path selection

Check for updates

Academic Editors: Nane Kratzke and Eldar Sultanow

Received: 8 July 2026
Revised: 12 August 2026
Accepted: 13 August 2026
Published: 18 August 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

The Edge–Fog–Cloud (EFC) continuum is increasingly adopted as an architectural model for distributed, data-intensive applications [1]. By distributing computation, storage, and networking capabilities from centralized Cloud resources toward the network edge, EFC enables scalable processing of the massive data streams generated by the Internet of Things (IoT). In parallel, safety-critical domains such as industrial automation, automotive

systems, and avionics, which are characterized by stringent timing constraints, are also undergoing data-centric transformations [2]. Traditionally deployed on specialized platforms, these systems are facing new demands for scalability and connectivity, as well as lifecycle challenges associated with rapid hardware obsolescence. Migrating safety-critical logic onto distributed EFC infrastructures promises substantial benefits. However, it exposes a fundamental conflict between best-effort EFC design principles and the determinism required by safety-critical systems. This conflict is referred to as the determinism gap: safety-critical workloads require strict end-to-end timing guarantees that state-of-the-art EFC architectures were not originally designed to provide [3].

The primary value proposition of many Edge–Fog–Cloud (EFC) research efforts is their dynamism, including flexible resource allocation [4] and adaptation to fluctuating workloads [5]. While this flexible model is attractive for best-effort distributed applications such as streaming analytics [6], it is problematic for safety-critical applications governed by hard end-to-end deadlines. In high-integrity real-time domains, including industrial automation, automotive systems, and avionics, temporal correctness is as important as functional correctness; a computed control command delivered after its deadline may be functionally useless and potentially unsafe [7,8]. Because unmanaged public Cloud environments include resource contention and virtualization effects that are not bounded at the application level, executing safety-critical logic on standard EFC infrastructures shifts critical timing risks to runtime layers outside the scheduler's control, thereby complicating the repeatable timing behavior required for high-integrity safety-critical applications [9,10].

This determinism gap is further amplified at the communication level. Standard EFC deployments commonly rely on best-effort communication protocols and routed application-layer messaging, which do not by themselves provide strict bounds on message latency [11]. Without bounded communication delays and synchronized time references, timing guarantees across a distributed multi-tier continuum cannot be established within the scheduling model.

This work addresses the determinism gap by adopting the time-triggered design principles [8,12] within the Time-Triggered Edge–Fog–Cloud (TTEFC) architectural model introduced in our prior work [13]. The TTEFC model targets managed, deterministic EFC systems in which temporal isolation is provisioned rather than assumed. Under this model, compute resources provide bounded task execution behavior, while the network fabric exposes a Deterministic Communication Interface (DCI) for bounded-latency communication [14]. The DCI abstracts deterministic communication services that can be implemented using IEEE Time-Sensitive Networking (TSN) in local Layer 2 domains and IETF Deterministic Networking (DetNet) in routed Layer 3 segments [15,16], together with managed time-synchronization support such as emerging 5G Time Synchronization as a Service (TSaaS) [17]. DCI compliance is therefore treated as an infrastructure-level responsibility of the provider or network operator.

Within this scoped timing model, the remaining challenge is at the scheduler level: the system must synthesize an offline end-to-end allocation and execution plan that jointly accounts for task execution, inter-partition communication, and deadline feasibility. This problem is NP-hard [18]. Existing EFC and workflow schedulers, including classical baselines such as HEFT [19] and GA-based EFC schedulers [3], do not explicitly combine deterministic route selection with feedback-driven time-budget adjustment. Accordingly, two scheduler-level gaps are identified.

First, computation placement and communication routing are commonly treated as separate, often sequential, optimization problems [3]. In typical EFC baselines, optimization is primarily driven by compute-centric objectives such as makespan, while communication is represented by simplified delay terms rather than explicit deterministic route selection.

This separation can produce placements that appear computationally efficient but do not account for whether the available inter-partition communication paths satisfy the bounded-latency assumptions required during end-to-end schedule reconstruction.

Second, adaptive time-budget adjustment across partitions is not explicitly modeled. A partition may exceed its assigned budget while another retains unused slack. Yet conventional hierarchical or decoupled scheduling flows lack an explicit feedback mechanism to redistribute this slack during optimization. As a result, feasibility recovery is often left to unguided perturbations, which can slow convergence or fail to find feasible end-to-end solutions even when sufficient global timing margin exists.

This paper addresses these two gaps by focusing on offline temporal feasibility synthesis over heterogeneous compute resources and bounded TSN/DetNet communication paths. In particular, the proposed HGA extends the earlier TTEFC optimization model [13] by coupling partition allocation, partition execution order, deterministic inter-partition route selection, and per-partition time-budget adjustment into a single feasibility-aware search. At the system level, the chromosome jointly encodes partition-to-execution-location allocation, partition execution order, route-selection hints, and time budgets. At the partition level, local GAs synthesize task schedules under the assigned time-budget constraints and return realized makespans and budget violations. The adaptive slack reallocation operator then uses these feedback values to shift unused temporal budget toward partitions that require additional execution margin. The evaluation includes an ablation study of adaptive time-budget reallocation, deadline-sensitivity analysis, and an automotive-characteristic case study. Table 1 summarizes the main differences between the prior work [13] and the present method.

Table 1. Comparison between the prior TTEFC framework [13] and the present method.

Aspect	Prior TTEFC Framework [13]	Present Method
Scheduling focus	Spatial task-to-tier partitioning followed by local scheduling within each tier.	Feasibility-aware hierarchical scheduling with partition allocation, execution ordering, time-budget assignment, and route selection.
Partition coordination	Local schedules are optimized independently after partitioning.	Partition schedules are coordinated through system-level time budgets and feedback from realized partition makespans.
Communication treatment	Communication is represented through static delay costs.	Deterministic communication paths are selected and included during end-to-end schedule reconstruction.
Temporal feasibility	No explicit partition-level time-budget adaptation is used.	Partition time budgets are adapted by reallocating available slack to budget-violating partitions.
Main objective	Reduce spatial partitioning and local scheduling cost.	Synthesize schedules that satisfy both partition-level budgets and the global deadline.

The main methodological elements and experimentally evaluated contributions of the present work are summarized as follows. The two-level hierarchical structure and the bounded-latency route-selection representation are design choices of the complete scheduler; their individual effects are not isolated experimentally.

- Two-level hierarchical scheduling structure: A two-level HGA is proposed in which the system-level GA jointly optimizes partition-to-execution-location allocation, partition execution order, deterministic inter-partition route-selection hints, and per-partition time budgets. These decisions constrain parallel partition-level GAs that synthesize local task schedules under assigned time-budget constraints, enabling coordinated end-to-end optimization across system and partition layers.

- Adaptive time-budget reallocation for hierarchical feasibility: A feedback-guided slack reallocation operator is introduced to adjust partition time budgets during the system-level search. The operator uses realized partition makespans, budget violations, and available slack to transfer temporal margin from over-satisfied partitions to partitions under timing pressure, thereby improving partition-budget feasibility and convergence stability.
- Bounded-latency path-selection representation: The proposed framework embeds route-selection decisions into the system-level chromosome, linking partition placement and scheduling decisions with infrastructure-provisioned communication paths demonstrating bounded latency. This enables communication-path availability and path-dependent delay to be considered during schedule reconstruction rather than as external post-processing. The HGA does not synthesize TSN gate-control schedules or DetNet resource reservations.
- Evaluation through benchmarks, ablation, and an automotive-characteristic case study: The proposed scheduler is evaluated on scalable synthetic DAG workloads, deadline-tightening scenarios, and an ablation study comparing feedback-guided reallocation against feedback-free and static-budget variants. In addition, an automotive-characteristic case study is constructed from public benchmark characteristics to assess the method on an application-oriented workload structure.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 introduces the Time-Triggered Edge–Fog–Cloud system model and formulates the scheduling problem. Section 4 presents the proposed hierarchical scheduling framework. Section 5 evaluates the proposed approach through benchmark and sensitivity analyses. Section 6 discusses the main observations derived from the results. Section 7 concludes the paper.

2. Related Work

Modern IoT applications are increasingly being deployed across an Edge–Fog–Cloud (EFC) continuum to meet demands for low latency and high throughput while exploiting heterogeneous resources near data sources. This shift motivates architectural models that clarify the roles of Edge, Fog, and Cloud nodes, and it raises system-level challenges in coordinating computation and communication across tiers. As this work addresses scheduling and communication-aware task placement in EFC settings, the following discussion will focus on architectural concepts and abstractions that directly affect timing, resource availability, and orchestration constraints.

2.1. Edge–Fog–Cloud Architectures

EFC architectures extend Cloud-centric deployments by distributing computation across Edge and Fog layers to improve responsiveness and reduce traffic. In [20], an EFC-based framework for streaming analytics was introduced to support IoT scenarios where continuous data streams and fast reaction times make purely centralized processing impractical. Architectural resilience and operational robustness have also been emphasized in EFC ecosystems. A resilience-oriented taxonomy was provided in [21], where approaches were grouped into capacity-, trustworthiness-, and efficiency-driven mechanisms, highlighting that orchestration policies should reflect node capabilities and failure characteristics rather than relying solely on offloading models.

The increasing adoption of IoT–Fog–Cloud ecosystems was further surveyed in [22], where Fog was described as an intermediate layer that complements the Cloud by enabling localized analytics and security functions while supporting quality of service objectives. Standardization efforts have aimed to formalize these layered deployments, including

OpenFog/IEEE 1934 guidelines [23] and MEC reference models [24]. Overall, these works motivate multi-tier abstractions in which heterogeneous resources, locality, and tier-specific constraints jointly shape feasible orchestration and scheduling decisions.

2.2. Communication-Aware and Deterministic Scheduling

Communication-aware scheduling has been studied at several abstraction levels, ranging from inter-core multiprocessor communication to distributed network scheduling. A distinction exists between deterministic path selection and deterministic network scheduling. While path selection determines which pre-established route a flow uses, network scheduling additionally assigns transmission opportunities and resources, such as queuing, shaping, and transmission windows, to resolve conflicts [15,16]. The present work performs deterministic path selection at the application scheduling level. For each communicating endpoint pair, the scheduler selects a route from a catalog of feasible paths and incorporates its bounded latency during end-to-end schedule reconstruction. The lower-level configuration required to guarantee these bounds—including TSN transmission configuration and DetNet resource reservation—is assumed to be provisioned by the network infrastructure via the Network Configurator and the Deterministic Communication Interface. Thus, the proposed HGA integrates application scheduling with pre-provisioned deterministic paths rather than synthesizing complete TSN or DetNet schedules natively. While existing approaches demonstrate the value of joint scheduling, their scopes differ from the present framework. Wang et al. jointly schedule computation and inter-core communication to minimize overhead, including memory usage [25,26]. However, their focus remains on on-chip execution rather than route selection across a distributed Edge–Fog–Cloud continuum or the feedback-guided coordination of time budgets. At the network level, Zhang et al. [27] investigated routing and scheduling co-design in software-defined deterministic networks, primarily focusing on coordinating network routes and transmission schedules. Similarly, Nie et al. [28] proposed a joint communication-offloading and scheduling scheme for TSN-based Edge networks that directly addresses TSN transmission planning and incremental stream injection. In contrast, the present framework incorporates path selection into application-level computation and partition scheduling, delegating detailed network transmission schedules to the underlying infrastructure. Accordingly, the contribution of the proposed framework is an offline hierarchical scheduler that couples cross-tier partition placement, execution ordering, selection among bounded-latency deterministic paths, and adaptive, feedback-guided reallocation of partition-level time budgets under a unified end-to-end feasibility model.

2.3. Hierarchical and Time-Budget-Aware Scheduling

General Cloud and Fog scheduling methods commonly address NP-hard workflow and resource-allocation problems by optimizing makespan, cost, energy consumption, or resource utilization through heuristic and metaheuristic search [29]. Genetic and hybrid optimization methods have demonstrated their effectiveness for exploring large task-mapping spaces [30,31]. However, these general-purpose approaches typically treat communication as an aggregate cost and do not coordinate deterministic cross-tier path selection with explicit partition-level timing constraints. More closely related to the present work are reservation-based and hierarchical scheduling methods that provide temporal isolation or coordinate resource allocation across multiple scheduling levels. KubeDeadline [32] combines SCHED_DEADLINE-based reservations with node-level admission control for latency-sensitive containerized applications. Struhár et al. [33] combine offline resource placement and dimensioning with online controllers that adjust CPU reservations or trigger migration when timing requirements cannot be maintained locally. Runtime monitor-

ing mechanisms have also been used to adapt execution budgets according to observed application behavior [34].

These approaches demonstrate the benefit of explicit execution reservations and hierarchical coordination. However, their adaptation is primarily admission-driven or runtime-oriented. They do not jointly synthesize cross-tier application placement, selection among provisioned deterministic communication paths, and feedback-guided redistribution of partition-level time budgets within an offline end-to-end scheduling search.

The closest direct predecessor of the present framework is the TTEFC optimization model introduced in our prior work [13]. That work established the managed multi-tier TTEFC architecture and applied a nested optimization structure in which a global search determined spatial task-to-tier partitioning, while local schedulers minimized makespan within the individual tiers.

The prior model did not include explicit partition-level time budgets, feedback-guided temporal negotiation, or path-selection genes. Communication was represented through static delay costs rather than selection among alternative deterministic paths. Consequently, local schedules were not coordinated through temporal feedback when one partition exceeded its required execution margin while another retained unused slack.

The present work extends this model from spatial partitioning to hierarchical temporal and path-selection co-design. The system-level chromosome jointly represents partition allocation, partition execution order, deterministic path-selection hints, and partition time budgets. Partition-level schedulers return realized makespans and budget violations to the system-level search, where an adaptive slack reallocation operator transfers temporal margin from over-satisfied partitions toward partitions under timing pressure. Thus, the main extension is the offline feedback loop between system-level budget assignment and partition-level schedule realization, combined with selection among infrastructure-provisioned TSN/DetNet paths under bounded communication latencies.

3. System Model

This section defines the Time-Triggered Edge–Fog–Cloud (TTEFC) system model considered in this work and the formal abstractions used by the proposed scheduler. The model represents a managed Edge–Fog–Cloud setting in which task execution times, communication delays, and clock offsets are bounded. This defines the timing scope of the scheduling problem studied in this paper and distinguishes it from unmanaged best-effort IoT/Cloud environments. We first describe the architecture and clarify the scope of the deterministic timing model. We then define the platform, network, and application models used to formulate the end-to-end scheduling problem. The scheduling algorithms are presented in Section 4.

3.1. TTEFC Architecture and Determinism Assumptions

Our work considers a distributed application deployed across a hierarchical Edge–Fog–Cloud (EFC) continuum. This architecture is driven by the growth of IoT data and the need to meet stringent timing requirements. A Fog orchestrator coordinates the execution of application partitions across tiers and manages the timing of inter-partition communication. Cross-domain communication (including Fog–Cloud and Cloud–Cloud connectivity) is assumed to use a deterministic backbone with bounded latency, while the Fog and Edge layers operate on Time-Sensitive Networking (TSN). TSN extends standard Ethernet with time synchronization, scheduled traffic, and resource reservation, enabling bounded latency under a shared time base. Under these assumptions, the scheduler can evaluate allocation, scheduling, and path-selection decisions using bounded execution and communication

latencies, as required by the considered hard-deadline scheduling model. The reference TTEFC deployment architecture considered in this work is shown in Figure 1.

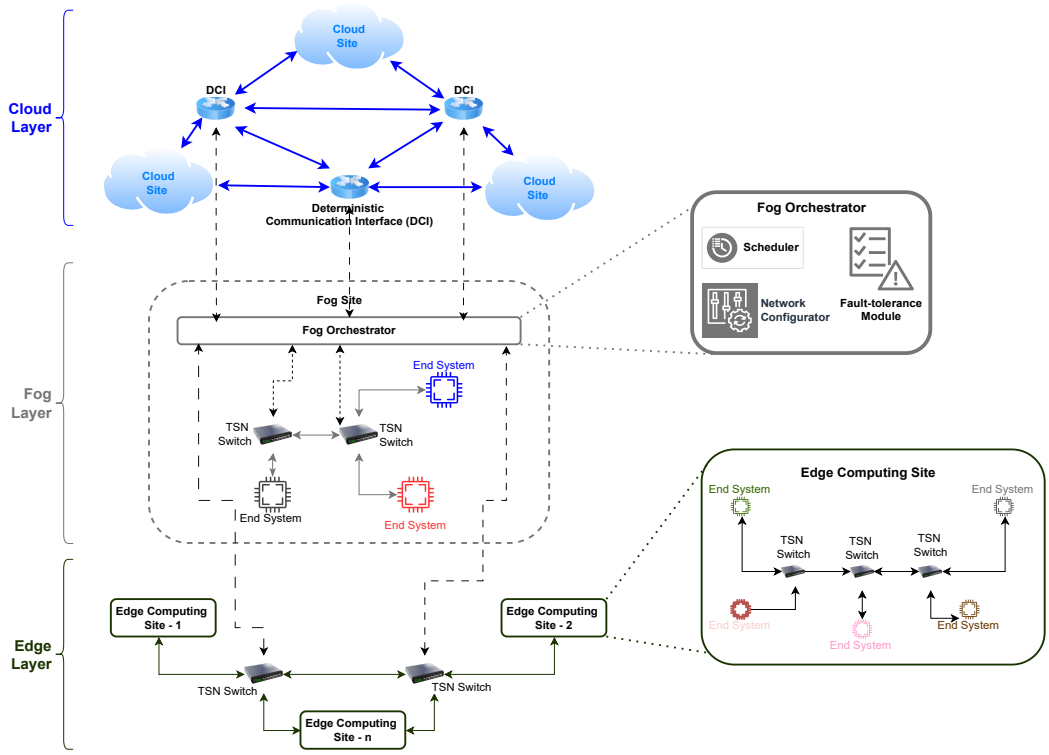


Figure 1. Reference TTEFC deployment architecture model used by the hierarchical scheduler.

3.2. Global Synchronization and TSaaS

To support time-triggered execution across the distributed continuum, a system-wide global time base is assumed. The 3GPP Time Synchronization as a Service (TSaaS) framework (Release 18) [17] facilitates this by leveraging 5G/6G infrastructure to distribute a Grandmaster clock signal with sub-microsecond accuracy to participating nodes. Any residual clock drift and distribution uncertainty are assumed to remain within a bounded error margin ensured by the synchronization service and the underlying network configuration, so that the time base can be used consistently for deterministic communication and time-triggered execution. This time base is assumed to be available to Edge, Fog, and Cloud end systems in the considered managed deployment.

3.3. Edge Layer

The Edge layer represents the physical boundary of the system and hosts the most latency-sensitive functions, such as sensing, preprocessing, and closed-loop actuation. Edge end systems are typically resource-constrained and therefore execute lightweight processing close to data sources, while compute-intensive functions may be offloaded to higher tiers when permitted by end-to-end timing constraints. The Edge layer follows the TSN assumptions stated in Section 3.1.

3.4. Fog Layer

The Fog layer is an intermediate tier between the resource-constrained Edge and the high-capacity Cloud. It hosts the Fog orchestrator, which acts as the application's coordinator by scheduling workloads, negotiating with different computational clusters, and offloading tasks to Cloud or Edge resources based on timing constraints. Positioned closer to Edge devices than the Cloud but with greater computational capacity than the

Edge, the Fog layer is well-suited to manage time-critical workloads: it can rapidly react to task arrivals, distribute processing across local nodes, and leverage Cloud resources when necessary to meet stringent deadlines. The Fog layer follows the TSN assumptions stated in Section 3.1.

3.4.1. Scheduler Module

The scheduler module implements a hierarchical scheduling mechanism that synthesizes both system-level and partition-level schedules. At the system level, it coordinates (i) the allocation of application partitions to Edge, Fog, or Cloud resources, (ii) the execution order of these partitions, (iii) the timing of inter-partition communication, and (iv) the assignment of per-partition time budgets that collectively satisfy the end-to-end deadline. At the partition level, each partition is scheduled independently within its assigned time budget by determining the internal task execution order, mapping tasks to End Systems, and enforcing intra-partition communication constraints to minimize the local makespan while preserving precedence and resource feasibility.

To improve feasibility under heterogeneous execution and communication delays, the scheduler incorporates an adaptive time-budget reallocation mechanism that redistributes slack from partitions that complete early to those that exceed their allocated budgets. The resulting local makespans are reported to the system level; when a partition exceeds its budget, feasibility is encouraged through targeted penalties and repair actions that guide the search toward schedules that satisfy the global timing constraints.

This hierarchical coordination mitigates globally infeasible solutions that may arise when task placement, timing, and communication routing are optimized independently across tiers. By negotiating time budgets based on feedback from partition schedules, end-to-end determinism and deadline compliance are maintained under varying workload characteristics.

3.4.2. Network Configurator

The network configurator provisions deterministic communication resources for application messages at both the inter- and intra-partition levels. For inter-partition messages, it maps each flow onto the deterministic communication fabric and reserves the required paths and transmission resources to maintain bounded latency. It exposes the set of feasible deterministic routes between communicating endpoints and associates each route with a bounded latency that is used during schedulability analysis. If no feasible deterministic connection exists for a required sender–receiver pair, the corresponding communication is treated as infeasible and handled as a violation in the end-to-end feasibility assessment.

For intra-partition messages, the configurator provides the bounded communication parameters required by the partition-level scheduler. When communicating tasks are placed on the same end system, the transfer is abstracted as a local delivery with negligible network delay. Otherwise, the transfer is constrained to the local deterministic domain (e.g., TSN) and is modeled with a bounded latency consistent with the provisioned local communication resources.

3.4.3. Fault-Tolerance Module

The architecture anticipates deterministic adaptation by integrating concepts from multi-schedule graphs (MSGs) and metascheduling. In time-triggered systems, metascheduling computes a schedule for each possible context event and organizes the schedules into a multi-schedule graph. Upon the occurrence of an event (e.g., a resource failure or detected slack), the system switches from the active schedule to the appropriate successor schedule in the MSG. Recent work in [2] shows that a multi-objective genetic algorithm-based metascheduler can generate MSGs while minimizing memory consumption by exploiting

structural similarity between schedules and storing only their incremental differences. By relying on pre-computed schedule variants and deterministic schedule switching, fault handling and recovery can be performed without introducing runtime uncertainty that could compromise end-to-end timing guarantees. This module is included as an architectural extension point; the evaluation in this paper focuses on baseline schedule synthesis.

3.5. Cloud Layer

Unlike standard best-effort Cloud models (e.g., AWS, GCP), the Cloud Layer in the proposed TTEFC architecture is defined as a Managed Real-Time Cloud Slice. This represents a logical partitioning of data center resources, with temporal isolation strictly enforced by real-time hypervisors and partitioning kernels. This substrate ensures a bounded worst-case execution time (WCET) for offloaded tasks. While our model uses representative resource constraints inspired by commercial platforms, the scheduler's ability to provide timing guarantees relies on this deterministic compute substrate.

3.6. Deterministic Communication Interface (DCI)

The deterministic backbone used for inter-site connectivity is modeled as a Deterministic Communication Interface (DCI). The DCI is assumed to provide bounded-latency communication for cross-domain transfers, including Fog–Cloud and Cloud–Cloud connectivity. This interface is implemented using IETF DetNet flows, which enable explicit route pinning and resource reservation to guarantee bounded latency [16]. Our model accounts for the significant propagation delay of these backbone paths by assigning higher communication costs to DCI transfers compared to local Edge/Fog links operating within TSN domains.

3.7. End Systems in the TTEFC Architecture

In the proposed architecture, the term End System (ES) is used in the strict sense defined by the avionics (TTEthernet) and industrial automation (TSN) domains. An End System is not merely a passive communication interface; it is a computational node equipped with a deterministic network interface. An End System ES_i is a tuple (C_i, N_i), where C_i represents the Computational Unit, comprising the CPU (Host), memory, and operating system capable of executing application tasks, and N_i represents the Network Unit, specifically a TSN-compliant Network Interface Card (NIC) or a specialized network controller.

The ES facilitates the interaction between the application layer and the deterministic network fabric through three primary functional roles:

- Computation: It executes the tasks assigned to it by the scheduler.
- Traffic Injection (Talker): It injects data messages into the network at precise, pre-scheduled time instants governed by the global schedule. The Network Unit N_i enforces traffic shaping (e.g., using IEEE 802.1Qbv time-aware shapers) to ensure that traffic enters the switch fabric without inducing congestion [35].
- Traffic Reception (Listener): It receives messages from the network, buffering them for consumption by the local tasks.

3.7.1. Heterogeneity of End Systems in the TTEFC Architecture

To capture the heterogeneous nature of the continuum, End Systems (ES) are differentiated across the Edge, Fog, and Cloud layers.

Edge End Systems

Edge End Systems represent the physical boundary of the system "far edge" and include smart sensors, Programmable Logic Controllers (PLCs), robotic joint controllers, and automotive Electronic Control Units (ECUs). These nodes are characterized by con-

strained computational resources. Typically based on microcontrollers (e.g., ARM Cortex-M/R) or low-power Systems-on-Chip (SoCs), they provide limited RAM and processing capability. Consequently, while they offer negligible communication latency for local I/O interactions, their processing-time values for complex tasks remain comparatively high. Their primary network role is to serve as sources of raw data streams and sinks for actuation commands.

Fog End Systems

Fog End Systems act as intermediate aggregation and processing points at the "near edge". These nodes, exemplified by Industrial PCs (IPCs), Edge gateways, and roadside units (RSUs), offer moderate computational capabilities that are significantly higher than those of Edge nodes. For the scheduler, Fog nodes represent a critical trade-off point: they provide execution speeds capable of handling tasks that are heavy for Edge nodes while maintaining lower communication latencies than the Cloud. Architecturally, they bridge local TSN domains at the edge with the routed deterministic backbone leading to the Cloud.

Cloud End Systems

Cloud End Systems constitute the high-performance tier located in data centers, comprising rack-mounted servers and bare-metal instances. These nodes provide massive memory bandwidth and server-grade processing power (e.g., Xeon and EPYC CPUs and GP-GPUs), resulting in the lowest processing-time values for computationally intensive workloads such as global path planning and model training. However, accessing these resources incurs a significant latency penalty due to physical distance and routing hops.

3.8. Platform and Network Model

The execution platform is modeled as a finite set of system-level execution locations,

$$\mathcal{C} = \{\mathrm{FE}, \mathrm{C1}, \mathrm{C2}, \mathrm{C3}\}.$$

Here, FE denotes the Edge–Fog execution location, whereas C1, C2, and C3 denote three Cloud execution locations. For each execution location $c \in \mathcal{C}$, let N_c denote the set of End Systems available at that location. The complete set of End Systems is

$$N = \bigcup_{c \in \mathcal{C}} N_c = \{ES_1, ES_2, \ldots, ES_{|N|}\}.$$

An execution location is therefore a system-level grouping of computational resources, whereas an End System is an individual computational node.

Throughout the remainder of the paper, *tier* refers to an architectural Edge, Fog, or Cloud layer; *execution location* refers to an element of $\mathcal{C}$; and *End System* refers to an individual node in N_c. The term *domain* is retained only for communication or network domains, such as a local TSN domain.

The network topology is extracted from the platform description and modeled as a graph $G_{\text{net}} = (V, L)$, where V contains End Systems and intermediate forwarding elements, and L contains the communication links between them.

For any sender–receiver pair $(s, r) \in \mathcal{N} \times \mathcal{N}$, a finite catalog of feasible deterministic paths is assumed to be provisioned by the network infrastructure, denoted by $\mathcal{P}_{s,r} = \{p_1, p_2, \ldots\}$. For a message m routed over a selected path $p \in \mathcal{P}_{s,r}$, the end-to-end communication latency is represented by the bounded cost function $L(m, p)$, which captures the propagation delay of the selected route and message-dependent transmission effects. Concurrent-flow contention is not modeled by the HGA as an additional dynamic delay. Instead, each candidate path is assumed to have been admitted and configured

by the underlying TSN/DetNet infrastructure so that its latency bound already accounts for the relevant transmission interference. In TSN domains, IEEE 802.1Qbv time-aware gates can prevent or bound contention-induced queuing for admitted scheduled traffic when the gate schedule and associated resources are correctly configured. In routed DetNet segments, explicit paths and allocated resources provide the corresponding bounded service [15,16,36]. The HGA therefore selects among precomputed bounded-latency paths but does not synthesize TSN gate-control lists, queue assignments, admission decisions, or DetNet resource reservations. Consistent with Section 3.6, paths traversing the DCI are assigned higher communication costs than local Edge/Fog paths.

3.9. Application Model

The application workload is modeled as a dependency graph, where data flow determines execution eligibility. The application $\mathcal{A}$ is represented as a Directed Acyclic Graph (DAG) $\mathcal{A} = (\mathcal{T}, \mathcal{E})$, where $\mathcal{T}$ is the task set and $\mathcal{E}$ is the set of directed dependencies.

3.9.1. Task Set ($\mathcal{T}$)

Let $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ be a finite set of tasks. Each task t_i is a non-preemptable unit of execution. Task execution times depend on the selected End System; let $PT(t_i, ES_k)$ denote the processing time (e.g., WCET) of task t_i when executed on $ES_k \in \mathcal{N}$.

3.9.2. Dependency Set ($\mathcal{E}$)

Let $\mathcal{E} \subseteq \mathcal{T} \times \mathcal{T}$ denote the set of directed edges representing precedence and data dependencies. An edge $(t_i, t_j) \in \mathcal{E}$ indicates that t_j cannot start until t_i completes and the produced data is delivered to the End System executing t_j.

3.9.3. Partition Model ($\mathcal{P}$)

To support modularity and hierarchical scheduling, the application is decomposed into a set of partitions, each grouping a subset of tasks scheduled together at the partition level. Let $\mathcal{P} = \{p_1, p_2, \ldots, p_z\}$ denote the set of partitions, and let $Part(t) \in \mathcal{P}$ return the partition to which a task $t \in \mathcal{T}$ belongs.

For a given partition $p \in \mathcal{P}$, the corresponding task subset is

$$\mathcal{T}_p = \{\, t \in \mathcal{T} \mid Part(t) = p \,\}. \tag{1}$$

The dependencies internal to partition p are

$$\mathcal{E}_p = \{\, (u, v) \in \mathcal{E} \mid u \in \mathcal{T}_p \wedge v \in \mathcal{T}_p \,\}. \tag{2}$$

Accordingly, the intra-partition workload is represented by the subgraph $\mathcal{G}_p = (\mathcal{T}_p, \mathcal{E}_p)$. In the evaluation, four application partitions are considered, $\mathcal{P} = \{p_{\mathrm{FE}}, p_{\mathrm{C1}}, p_{\mathrm{C2}}, p_{\mathrm{C3}}\}$. Each task belongs to one of these partitions according to the workload decomposition. This task-to-partition membership remains unchanged during optimization, while the system-level HGA determines the execution location assigned to each partition. The partition subscripts identify the workload partitions and do not prescribe their execution locations.

3.9.4. Message Sets ($\mathcal{M}$)

Communication is modeled explicitly as messages associated with dependency edges. For each edge $(u, v) \in \mathcal{E}$, a message $m_{u,v}$ represents the data transfer from the producer task u to the consumer task v. The set of all application messages is

$$\mathcal{M} = \{\, m_{u,v} \mid (u, v) \in \mathcal{E} \,\}. \tag{3}$$

Inter- and intra-partition messages are distinguished based on whether the communicating tasks belong to different partitions:

$$\mathcal{M}_{inter} = \{\, m_{u,v} \in \mathcal{M} \mid Part(u) \neq Part(v) \,\}, \tag{4}$$

$$\mathcal{M}_{intra} = \{\, m_{u,v} \in \mathcal{M} \mid Part(u) = Part(v) \,\}. \tag{5}$$

Messages in $\mathcal{M}_{inter}$ traverse the deterministic communication fabric across tiers, while messages in $\mathcal{M}_{intra}$ are handled within the scope of a single partition.

3.10. Problem Definition

Given an application DAG $A = (T, E)$, a partition set P, a system-level execution-location set $\mathcal{C}$, the End-System sets $\{N_c \mid c \in \mathcal{C}\}$, and deterministic path catalogs $\{\mathcal{P}_{s,r}\}$, the goal is to synthesize an end-to-end schedule that jointly accounts for (i) system-level partition allocation among execution locations and (ii) partition-level task schedules on individual End Systems under bounded execution and communication latencies. A global end-to-end deadline D is specified for the application.

3.10.1. Decision Variables

The following coupled decisions determine the end-to-end schedule:

- System-level partition allocation: a mapping

$$\Phi : \mathcal{P} \to \mathcal{C}$$

that assigns each partition $p \in \mathcal{P}$ to one system-level execution location $\Phi(p) \in \mathcal{C}$. In the evaluated configuration, $\mathcal{C} = \{\text{FE}, \text{C1}, \text{C2}, \text{C3}\}$, and Φ is restricted to a one-to-one allocation between the four application partitions and these four execution locations. Consequently, no two partitions share the same execution location. The allocation selects the group of End Systems available for scheduling the tasks of a partition, rather than an individual End System. Accordingly, Φ is encoded as a permutation of $\mathcal{C}$, and the system-level genetic operators preserve this constraint.
- Partition-level task mapping: for each partition $p \in P$, a local mapping

$$\phi_p : T_p \to N_{\Phi(p)}$$

that assigns each task of partition p to an individual End System belonging to the execution location selected by $\Phi(p)$. This mapping is produced by the partition-level scheduler, subject to the time-budget and feasibility constraints imposed by the system-level scheduler.
- System-level execution order: an ordered sequence π over $\mathcal{P}$ that defines the system-level order in which partitions are integrated into the global schedule.
- Deterministic routing for inter-partition messages: for each inter-partition message $m_{u,v} \in \mathcal{M}_{inter}$, a deterministic path $\rho(m_{u,v}) \in \mathcal{P}_{s,r}$ is selected, where $s = \phi_{Part(u)}(u)$ and $r = \phi_{Part(v)}(v)$ denote the End Systems executing the sender task u and the receiver task v, respectively.
- Per-partition time budgets: a budget vector $TB = \{TB(p) \mid p \in \mathcal{P}\}$ that assigns an execution-time budget constraint to each partition. The budgets are negotiated so they support the feasibility of the global deadline.

3.10.2. Constraints

Feasibility of an end-to-end schedule requires:

- **Precedence and data availability:** For every dependency edge $(u, v) \in \mathcal{E}$, task v must not start before task u completes and the associated message $m_{u,v}$ is delivered.
- **Resource feasibility:** Tasks mapped to the same End System must not overlap in time on the same processing resource, and each task executes with processing time $PT(t_i, ES_k)$ on its assigned End System.
- **Bounded communication:** Each inter-partition message $m_{u,v} \in \mathcal{M}_{inter}$ must be assigned a feasible deterministic path $\rho(m_{u,v})$ with bounded latency $L(m_{u,v}, \rho(m_{u,v}))$; if no feasible path exists, the solution is infeasible.
- **Partition time-budget compliance:** For each partition $p \in \mathcal{P}$, the makespan of its partition-level schedule, denoted by $MS(p)$, must satisfy $MS(p) \leq TB(p)$.
- **End-to-end deadline:** The global completion time (global makespan) MS_{global} must satisfy $MS_{global} \leq D$.

3.10.3. Optimization Objectives

The primary system-level objective is to synthesize an end-to-end schedule that satisfies the partition time-budget constraints and the global application deadline under the bounded execution and communication assumptions of the considered model. The system-level search is therefore formulated as a bi-objective feasibility problem.

For a system-level individual ind, let $MS_p(\text{ind})$ denote the makespan returned by the partition-level scheduler for partition p, let $TB_p(\text{ind})$ denote the time budget assigned to that partition, and let $MS_{\text{global}}(\text{ind})$ denote the makespan of the reconstructed end-to-end schedule. The first system-level objective is the cumulative partition time-budget violation:

$$F_{\text{viol}}(\text{ind}) = \sum_{p \in P} \max\big(0, MS_p(\text{ind}) - TB_p(\text{ind})\big). \tag{6}$$

The second system-level objective is the positive global lateness:

$$F_{\text{late}}(\text{ind}) = \max\Big(0, MS_{\text{global}}(\text{ind}) - D\Big). \tag{7}$$

The resulting system-level fitness vector is

$$\mathbf{F}_{\text{sys}}(\text{ind}) = (F_{\text{viol}}(\text{ind}), F_{\text{late}}(\text{ind})), \tag{8}$$

where both components are minimized jointly using NSGA-II [37]. Let $IPC_{\text{viol}}(\text{ind})$ denote the set of inter-partition messages for which no feasible deterministic path exists. A system-level individual is fully feasible if and only if

$$F_{\text{viol}}(\text{ind}) = 0, \qquad F_{\text{late}}(\text{ind}) = 0, \qquad IPC_{\text{viol}}(\text{ind}) = \emptyset. \tag{9}$$

Raw global makespan is not included as an additional system-level fitness component. Instead, end-to-end deadline compliance is represented by F_{late}, which decreases as MS_{global} approaches the deadline and becomes zero once $MS_{\text{global}} \leq D$. Beyond this point, the global-lateness objective no longer provides direct selection pressure for further makespan reduction, although the evolutionary search continues for the configured number of generations.

The reconstructed global makespan is nevertheless retained as an evaluation metric and is used to distinguish multiple fully feasible individuals during the final representative-selection step described in Section 4.2.

At the partition level, the local schedulers minimize their partition makespans while accounting for time-budget overruns, as defined in Section 4.4. Consequently, partition execution times, placement decisions, and route selections affect the reconstructed global makespan, while the system-level fitness remains explicitly focused on partition time-budget feasibility and end-to-end deadline feasibility.

4. Scheduling Framework

This section presents the proposed two-level scheduler for the problem defined in Section 3.10.

4.1. Framework Overview

The proposed scheduler is organized into a *system-level scheduler* and multiple parallel *partition-level schedulers*. The system-level scheduler determines (i) partition allocation across execution domains in $\mathcal{C}$, (ii) the system-level execution order of partitions, (iii) deterministic route selections for inter-partition messages, and (iv) per-partition time budgets that define offline temporal budget constraints. Given $(\Phi(p), TB(p))$, each partition-level scheduler constructs a local schedule *within the assigned time budget* by selecting a precedence-feasible task order and mapping tasks to End Systems in $N_{\Phi(p)}$, while enforcing intra-partition communication constraints. The resulting partition schedules are integrated through an end-to-end reconstruction step that inserts inter-partition transmissions using the selected deterministic paths with bounded latency; the reconstructed schedule is then evaluated to obtain the global makespan and feasibility violations for each system-level solution.

For clarity, Table 2 summarizes the principal system-level and partition-level decision variables used by the hierarchical scheduler.

Table 2. Main notation used by the hierarchical scheduler.

Symbol	Description
π	System-level permutation defining the order in which partitions are integrated into the global schedule.
Φ	System-level mapping $\Phi : P \to C$ assigning each partition to an execution location in $C = \{FE, C1, C2, C3\}$.
ρ	System-level route-selection vector. For each inter-partition message, $\rho[m]$ stores an index into the candidate-path set and thereby determines the selected deterministic path.
TB	Vector containing the time budget assigned to each partition.
σ^p	Precedence-feasible task order within partition p.
φ_p	Mapping $\varphi_p : T_p \to N_{\Phi(p)}$ assigning the tasks of partition p to End Systems within its selected execution location.
η^p	Priority values used to order intra-partition messages.
ξ^p	Candidate-path indices used for intra-partition messages.

Illustrative End-to-End Example

Consider an application with two partitions, $P = \{p_1, p_2\}$, where $T_{p_1} = \{t_1, t_2\}$, $T_{p_2} = \{t_3, t_4\}$, and $m_{2,3}$ is an inter-partition message from t_2 to t_3. Assume that a system-level individual specifies $\pi = [p_1, p_2]$, $\Phi(p_1) = FE$, $\Phi(p_2) = C1$, and initial time budgets $TB = [20, 14]$ TU. The selected deterministic path for $m_{2,3}$ has a bounded latency of 4 TU.

Suppose that the partition-level schedulers return $MS(p_1) = 15$ TU and $MS(p_2) = 17$ TU. Partition p_1 therefore has 5 TU of unused budget, whereas p_2 exceeds its budget by 3 TU, giving $F_{\text{viol}} = 3$ TU. Using this feedback, the time-budget operator progressively adjusts the budgets; an illustrative later budget vector is $TB' = [16, 18]$ TU, for which $F_{\text{viol}} = 0$.

During end-to-end reconstruction, if t_2 finishes at 15 TU, the message arrives at $15 + 4 = 19$ TU. Partition p_2 is consequently shifted according to this arrival constraint, producing a global makespan of 36 TU. For $D = 40$ TU, $F_{\text{late}} = \max(0, 36 - 40) = 0$; therefore, the reconstructed individual is fully feasible. This example is included only to illustrate the scheduling workflow and does not represent an additional experiment.

4.2. System-Level Scheduler

The system-level scheduler evolves a population of candidate end-to-end solutions, where each individual jointly encodes (i) a partition allocation among the execution locations in $\mathcal{C}$, (ii) a system-level partition order, (iii) deterministic route selections for inter-partition messages, and (iv) per-partition time budgets. The system-level search aims to achieve end-to-end schedulability under the global deadline and per-partition time-budget constraints. Partition budget overruns are represented by F_{viol}, while missing feasible deterministic routes are recorded as communication violations and excluded from the fully feasible solution set.

The exploration covers trade-offs between computation and communication across tiers. For example, allocating a partition to a higher tier may reduce its processing time while increasing the latency of its inter-partition communications, whereas allocating it closer to the edge may reduce communication delay at the cost of longer execution times. Deterministic route selections are enforced by assigning each inter-partition message to a specific candidate path from the path catalog, thereby avoiding runtime route-selection variability. The resulting system-level schedule specifies where partitions execute, in what order they are integrated, how inter-partition messages are routed over deterministic paths, and which time budgets constrain the partition-level schedulers. The system-level scheduling procedure is summarized in Algorithm 1.

The system-level search treats cumulative partition time-budget violation F_{viol} and positive global lateness F_{late} as joint objectives throughout the NSGA-II evolution. After the final generation, fully feasible individuals, satisfying $F_{\text{viol}} = F_{\text{late}} = 0$, are considered first. If multiple fully feasible individuals are available, the individual with the smallest reconstructed global makespan is returned.

If the final population contains no fully feasible individual, a representative is selected from the first non-dominated front. Since the two objectives are measured on potentially different numerical scales, they are first normalized within that front. The individual with the smallest Euclidean distance to the ideal normalized point $(0, 0)$ is then returned as the best available compromise. This procedure does not impose a fixed priority between partition-budget compliance and global deadline compliance.

Each system-level individual is evaluated by invoking the partition-level scheduler for every partition and reconstructing an end-to-end schedule to compute feasibility violations and the global makespan. This evaluation procedure is summarized in Algorithm 2.

The system-level search relies on genetic operators that preserve the validity of the permutation-based genome components and maintain the alignment between the partition order and the time-budget vector. The system-level crossover uses partially mapped crossover (PMX) [38] for permutation-encoded components and two-point crossover for the route-selection component, as summarized in Algorithm 3.

Algorithm 4 defines the structure-preserving mutation operator applied to the system-level genome, with repairs to maintain feasibility under the model constraints.

Algorithm 1 System-level scheduler

Input: Per-partition data $(PT^p, \mathcal{M}^p_{\text{intra}}, JD^p)$; inter-partition messages $\mathcal{M}_{\text{inter}}$; path catalogs $\{\mathcal{P}_{s,r}\}$; global deadline D; GA parameters $(N_{\text{pop}}, N_{\text{gen}}, p_{\text{cx}}, p_{\text{mut}})$
Output: Representative evaluated individual $ind^\star$

Genome: $ind = [\,\pi \mid \Phi \mid \rho \mid TB\,]$
$Pop \leftarrow \textsc{InitPopulation}(N_{\text{pop}}, D)$
for all $ind \in Pop$ **do**
  $ind.\text{fit} \leftarrow \textsc{Evaluate}(ind)$
  $ind.\text{fit} = \big(F_{\text{viol}}(ind), F_{\text{late}}(ind)\big)$
end for
$Pop \leftarrow \text{NSGAIISelect}(Pop, N_{\text{pop}})$
for $gen = 1$ to N_{gen} **do**
  $Parents \leftarrow \textsc{CrowdedTournament}(Pop)$
  $Offspring \leftarrow \textsc{Clone}(Parents)$
  for all pairs (c_1, c_2) in $Offspring$ **do**
    if $rand < p_{\text{cx}}$ **then**
      $(c_1, c_2) \leftarrow \textsc{CrossoverSys}(c_1, c_2)$
    end if
  end for
  for all $child \in Offspring$ **do**
    if $rand < p_{\text{mut}}$ **then**
      $child \leftarrow \textsc{MutateSys}(child)$
    end if
    $child \leftarrow \textsc{MutateTimeBudget}(child)$
    $child.\text{fit} \leftarrow \textsc{Evaluate}(child)$
    $child.\text{fit} = \big(F_{\text{viol}}(child), F_{\text{late}}(child)\big)$
  end for
  $Pop \leftarrow \text{NSGAIISelect}(Pop \cup Offspring, N_{\text{pop}})$
end for
$Pop_{\text{feas}} \leftarrow \{ind \in Pop \mid F_{\text{viol}}(ind) = 0 \wedge F_{\text{late}}(ind) = 0 \wedge IPC_{\text{viol}}(ind) = \emptyset\}$
if $Pop_{\text{feas}} \neq \emptyset$ **then**
  $ind^\star \leftarrow \arg\min_{ind \in Pop_{\text{feas}}} MS_{\text{global}}(ind)$
else
  $Front_1 \leftarrow \textsc{FirstNonDominatedFront}(Pop)$
  $v_{\min} \leftarrow \min_{ind \in Front_1} F_{\text{viol}}(ind)$
  $v_{\max} \leftarrow \max_{ind \in Front_1} F_{\text{viol}}(ind)$
  $\ell_{\min} \leftarrow \min_{ind \in Front_1} F_{\text{late}}(ind)$
  $\ell_{\max} \leftarrow \max_{ind \in Front_1} F_{\text{late}}(ind)$
  for all $ind \in Front_1$ **do**
    if $v_{\max} > v_{\min}$ **then**
      $\widehat{F}_{\text{viol}}(ind) \leftarrow \dfrac{F_{\text{viol}}(ind) - v_{\min}}{v_{\max} - v_{\min}}$
    else
      $\widehat{F}_{\text{viol}}(ind) \leftarrow 0$
    end if
    if $\ell_{\max} > \ell_{\min}$ **then**
      $\widehat{F}_{\text{late}}(ind) \leftarrow \dfrac{F_{\text{late}}(ind) - \ell_{\min}}{\ell_{\max} - \ell_{\min}}$
    else
      $\widehat{F}_{\text{late}}(ind) \leftarrow 0$
    end if
    $d(ind) \leftarrow \sqrt{\widehat{F}_{\text{viol}}(ind)^2 + \widehat{F}_{\text{late}}(ind)^2}$
  end for
  $ind^\star \leftarrow \arg\min_{ind \in Front_1} d(ind)$
end if
return $ind^\star$

Algorithm 2 Evaluate

Input: Individual $ind = [\,\pi \mid \Phi \mid \rho \mid TB\,]$; partition data $(PT^p, \mathcal{M}^p_{\text{intra}}, JD^p)$
inter-partition messages $\mathcal{M}_{\text{inter}}$; path catalogs $\{\mathcal{P}_{s,r}\}$; global deadline D
Output: Fitness vector $ind.\text{fit} = \big(F_{\text{viol}}(ind), F_{\text{late}}(ind)\big)$ and evaluation artifacts attached to ind

Build $triple[p] = (\text{alloc} = \Phi[i], \text{tb} = TB[i])$ for each partition $p = \pi[i]$
for all partitions p in order π **do**
  $(sched^p, ms^p, \phi_p) \leftarrow \textsc{PartitionLevelScheduler}\big(p, PT^p, \mathcal{M}^p_{\text{intra}}, JD^p, triple[p], \{\mathcal{P}_{s,r}\}\big)$
  $results[p] \leftarrow sched^p; \quad maps[p] \leftarrow \phi_p; \quad makespans[p] \leftarrow ms^p$
  $\text{viol}[p] \leftarrow \max\big(0, ms^p - triple[p].\text{tb}\big)$
end for
$(globalSched, MS_{\text{global}}, IPC_{\text{viol}}) \leftarrow$ $\textsc{SystemLevelReconstruction}\big(results, maps, \mathcal{M}_{\text{inter}}, \rho, \{\mathcal{P}_{s,r}\}\big)$
$latenessSigned \leftarrow MS_{\text{global}} - D$
$F_{\text{late}}(ind) \leftarrow \max\big(0, latenessSigned\big)$
$F_{\text{viol}}(ind) \leftarrow \sum_{p \in P} \text{viol}[p]$
Attach to ind: $eval_\pi$, $eval_\Phi$, $eval_\rho$, $eval_TB$, $results$, $maps$, $makespans$, and viol MS_{global}, $latenessSigned$, and IPC_{viol}
$ind.\text{fit} \leftarrow \big(F_{\text{viol}}(ind), F_{\text{late}}(ind)\big)$
return ind.fit

Algorithm 3 CrossoverSys

Input: Parent chromosomes c_1, c_2 with genome $[\,\pi \mid \Phi \mid \rho \mid TB\,]$
Output: Offspring (c_1', c_2') with valid π, Φ and aligned TB

Apply PMX on π to obtain valid partition-order permutations
Apply PMX with repair on Φ to preserve a valid allocation structure
Realign TB to the new π so that budgets remain attached to partition IDs
Apply two-point crossover on ρ
return updated (c_1, c_2)

Algorithm 4 MutateSys

Input: Individual $child = [\,\pi \mid \Phi \mid \rho \mid TB\,]$
Output: Mutated individual with valid π, Φ and aligned TB

Perform a synchronized swap on (π, Φ, TB) at two positions
Optionally swap two entries in Φ as an allocation tweak
Repair π and Φ if necessary to preserve the valid partition ordering and one-to-one partition-to-execution-location allocation
return $child$

Deterministic routing decisions are realized by selecting a concrete path for each inter-partition message from its candidate path set. The path selection function used during evaluation and reconstruction is given in Algorithm 5.

Algorithm 5 SelectPath

Input: Route-index hint h; candidate paths $\mathcal{P}_{s,r}$ for endpoints (s, r)
Output: Chosen path $p^\star$

$C \leftarrow \mathcal{P}_{s,r}$
if $C = \varnothing$ **then**
  return penalized virtual path
end if
if $0 \le h < |C|$ **then**
  return $C[h]$
else
  return $\arg\min_{p \in C} \text{latency}(p)$
end if

The minimum-latency branch in Algorithm 5 is included only as a defensive implementation safeguard to prevent schedule reconstruction from failing if an out-of-range route index is encountered. Under normal operation, route-index genes are initialized and processed as indices of the available candidate-path catalog, and the selected path is therefore determined by the evolved hint. The fallback is not intended as an additional greedy routing policy or as a competing optimization mechanism; it only ensures that an evaluable schedule can still be constructed in the presence of an unexpected invalid index.

4.3. End-to-End Reconstruction

This subsection describes the reconstruction step that integrates partition schedules and inserts inter-partition transmissions under deterministic routing. Algorithm 6 summarizes this procedure.

Algorithm 6 SystemLevelReconstruction

Input: Per-partition schedules $\{sched^p\}$ with task times and task mappings $\{\phi_p\}$
inter-partition messages $\mathcal{M}_{\text{inter}}$; route-index hints ρ; path catalogs $\{\mathcal{P}_{s,r}\}$
Output: $globalSched$ containing tasks and inter-partition transmissions; MS_{global}; set $IPCviol$

Assemble $globalSched$ from $\{sched^p\}$
$globalSched$.tasks $\leftarrow \bigcup_p sched^p$
$globalSched$.transmissions $\leftarrow \varnothing$
$M \leftarrow \mathcal{M}_{\text{inter}}$; $IPCviol \leftarrow \varnothing$
Initialize arrival constraints:
for each task v, set ArriveAfter$[v] \leftarrow start(v)$
for all $m_{u,v} \in M$ **do**
  $s \leftarrow \phi_{\text{Part}(u)}(u)$; $r \leftarrow \phi_{\text{Part}(v)}(v)$
  $C \leftarrow \mathcal{P}_{s,r}$
  if $C = \varnothing$ **then**
    $IPCviol \leftarrow IPCviol \cup \{m_{u,v}\}$
    $arr(m_{u,v}) \leftarrow +\infty$
  else
    $idx \leftarrow \rho[m_{u,v}]$
    if $0 \le idx < |C|$ **then**
      $path(m_{u,v}) \leftarrow C[idx]$
    else
      $path(m_{u,v}) \leftarrow \arg\min_{p \in C} \text{latency}(p)$
    end if
    $arr(m_{u,v}) \leftarrow end(u) + \text{latency}(path(m_{u,v}), m_{u,v})$
    Add $(m_{u,v}, tx_s{=}end(u), tx_e{=}arr(m_{u,v}), path(m_{u,v}))$ to $globalSched$.transmissions
    ArriveAfter$[v] \leftarrow \max(\text{ArriveAfter}[v], arr(m_{u,v}))$
  end if
end for
EnforceArrivalConsistency (*globalSched*.tasks, ArriveAfter)
AlignProcessorQueues (*globalSched*.tasks)
$MS_{\text{global}} \leftarrow$ maximum end-time over all tasks in $globalSched$.tasks
return $globalSched$, MS_{global}, $IPCviol$

For each inter-partition message $m_{u,v} \in \mathcal{M}_{inter}$, the sender and receiver End Systems are identified from the task mappings, and a feasible candidate path is selected using the route-index hint $\rho[m_{u,v}]$. If the hinted index is invalid, the path with minimum latency is selected. The message arrival time is computed as $end(u)$ plus the bounded latency of the selected path, and the consumer task is constrained to start no earlier than this arrival time. Messages for which no feasible deterministic path exists are recorded as inter-partition communication violations. The global makespan MS_{global} is computed as the latest completion time over all tasks after enforcing these arrival constraints.

Here, $IPCviol$ denotes the set of inter-partition communication violations, i.e., messages in $\mathcal{M}_{inter}$ for which no feasible deterministic path exists.

Two consistency operations in Algorithm 6 are implemented. EnforceArrivalConsistency enforces dependency timing: a task may start only after all of its predecessors have completed and, for communicated dependencies, after the corresponding messages have arrived at the receiving End System. If necessary, the task start time is delayed to satisfy the latest of these requirements. AlignProcessorQueues enforces execution-resource exclusivity. For each End System, the assigned tasks are considered in their current execution order, and any task that overlaps with the preceding task is delayed until that task is completed. Because such timing adjustments may affect downstream dependencies, both conditions are re-evaluated until no further task needs to be delayed.

4.4. Partition-Level Scheduler

Once the system-level scheduler fixes the execution-location allocation $\Phi(p)$ and time budget $TB(p)$ for a partition p, a local optimization is performed to construct a feasible schedule using the End Systems in $N_{\Phi(p)}$. Each partition-level scheduler operates independently and can be executed in parallel for different partitions. Its role is to determine a precedence-feasible task order, a task-to-End-System mapping within $N_{\Phi(p)}$, and the corresponding intra-partition communication decisions while respecting the local time budget.

For each partition p, the local optimization problem is to minimize the partition makespan subject to the assigned scheduling time-budget constraint:

$$\min\ (ms^p) \qquad \text{s.t.} \qquad ms^p \leq TB(p). \tag{10}$$

ms^p denotes the completion time of the last task in partition p, whereas $TB(p)$ is the time budget assigned by the system-level scheduler. Schedules that exceed the assigned budget are treated as locally infeasible and contribute a violation term during the system-level evaluation.

To solve this problem, a genetic search is applied at the partition level. Each local individual is encoded as

$$g^p = [\,\sigma^p \mid \phi_p \mid \eta^p \mid \xi^p\,], \tag{11}$$

where σ_p is a precedence-feasible task order, ϕ_p maps tasks to individual End Systems in $N_{\Phi(p)}$, η_p defines local message priorities, and ξ_p stores the candidate-path indices used for intra-partition communications.

$$fit(g^p) = (ms^p,\ \max(0, ms^p - TB(p))), \tag{12}$$

which simultaneously favors short makespans and penalizes budget overruns.

Algorithm 7 summarizes the partition-level scheduler. The partition-level search begins by generating an initial population of valid local individuals. The task-order component σ^p is initialized using random topological orders derived from the precedence relations in JD^p, thereby ensuring that all local individuals satisfy intra-partition dependencies from the outset. The mapping component ϕ_p assigns tasks only to End Systems belonging to the execution location selected by $\Phi(p)$, while η^p and ξ^p initialize message priorities and local path selections for intra-partition communications.

At each generation, every local individual is evaluated through a deterministic reconstruction step that converts the encoded order, mapping, and communication choices into a concrete schedule. The resulting local makespan ms^p is then combined with the budget-overrun term $\max(0, ms^p - TB(p))$ to compute the local fitness according to (12). The evolutionary process then applies selection, crossover, and mutation operators that preserve precedence feasibility and allocation validity. Once the local search terminates,

the best reconstructed schedule is returned to the system-level scheduler, together with its realized task mapping and makespan.

For a fixed genome, the reconstruction procedure deterministically produces a unique local schedule. The start time of each task is computed solely from predecessor completion times, communication latencies along the selected paths, and the availability of the assigned End System. Consequently, the resulting makespan and any budget violation depend only on the decisions encoded in the genome. This determinism ensures consistent evaluation of individuals at the system level and allows scheduling decisions to be compared reliably during the evolutionary search.

Algorithm 7 PARTITIONLEVELSCHEDULER

Input: Partition p; data $(PT^p, \mathcal{M}^p_{\text{intra}}, JD^p)$; (alloc=$\Phi(p)$, tb=$TB(p)$)
path catalogs $\{\mathcal{P}_{s,r}\}$; inner GA parameters $(N^{(p)}_{\text{pop}}, N^{(p)}_{\text{gen}}, p^{(p)}_{\text{cx}}, p^{(p)}_{\text{mut}})$
Output: Local schedule $sched^p$, local makespan ms^p, and task mapping ϕ_p

Genome: $g^p = [\,\sigma^p \mid \phi_p \mid \eta^p \mid \xi^p\,]$
$Pop^p \leftarrow$ INITLOCALPOPULATION$(p, \Phi(p))$
$best^p \leftarrow \varnothing$; $bestMS^p \leftarrow +\infty$
for $gen = 1$ to $N^{(p)}_{\text{gen}}$ **do**
  for all $g^p \in Pop^p$ **do**
    $(sched^p_g, ms^p_g, \phi^g_p) \leftarrow$ RECONSTRUCTPARTITION $(JD^p, g^p, PT^p, \mathcal{M}^p_{\text{intra}}, \{\mathcal{P}_{s,r}\})$ ▷ Algorithm 8
    $fit(g^p) \leftarrow (ms^p_g, \max(0, ms^p_g - TB(p)))$
    if $ms^p_g < bestMS^p$ **then**
      $best^p \leftarrow g^p$; $bestMS^p \leftarrow ms^p_g$
    end if
  end for
  $Parents^p \leftarrow$ TOURNAMENTSELECT(Pop^p, fit)
  $Offspring^p \leftarrow$ CROSSOVERLOCAL$(Parents^p)$
  $Offspring^p \leftarrow$ MUTATELOCAL$(Offspring^p)$
  $Pop^p \leftarrow$ SELECTNEXTLOCAL$(Parents^p \cup Offspring^p, N^{(p)}_{\text{pop}})$
end for
$(sched^p, ms^p, \phi_p) \leftarrow$ RECONSTRUCTPARTITION $(JD^p, best^p, PT^p, \mathcal{M}^p_{\text{intra}}, \{\mathcal{P}_{s,r}\})$
return $(sched^p, ms^p, \phi_p)$

4.5. Local Schedule Reconstruction

Given a local genome $g^p = [\,\sigma^p \mid \phi_p \mid \eta^p \mid \xi^p\,]$, the reconstruction procedure deterministically constructs the schedule of partition p by assigning start and completion times to all tasks and by inserting the corresponding intra-partition transmissions.

The reconstruction must satisfy three conditions: (i) precedence feasibility between tasks, (ii) exclusive use of each End System, and (iii) causal consistency of intra-partition communications.

Tasks are processed according to the precedence-feasible order encoded in σ^p. For each task t, the earliest feasible start time is computed as the maximum of processor availability and the arrival time of all incoming messages. The assigned End System is determined by $\phi_p(t)$, while the communication latency of each incoming message is derived from the candidate path selected through the path-index vector ξ^p.

Once a task is scheduled, all outgoing intra-partition messages are inserted according to the message-priority vector η^p. The resulting message transmission times are then propagated to successor tasks to ensure that no task starts before all required data has arrived.

The partition makespan ms^p is finally obtained as the maximum completion time among all tasks in the reconstructed schedule. Algorithm 8 summarizes this procedure.

Algorithm 8 RECONSTRUCTPARTITION (Local Reconstruction)

Input: JD^p; genome $g^p = \langle \sigma^p, \phi_p, \eta^p, \xi^p \rangle$; PT^p; $\mathcal{M}^p_{\text{intra}}$; path catalogs $\{\mathcal{P}_{s,r}\}$
Output: sched_p containing task and intra-partition message times; local makespan ms_p; φ_p

Initialize $next_free[q] \leftarrow 0$ for every end system q used in ϕ_p
for all task t in σ^p **do**
  $q \leftarrow \phi_p[t]$
  $\text{predReady} \leftarrow \max_{u \in Pred(t)} (end(u) + \text{latency}(path_{u,t}, m_{u,t}))$ where $path_{u,t} = \text{SELECTPATH}(\xi^p[m_{u,t}], \mathcal{P}_{\phi_p(u),q})$ ▷ 0 if $Pred(t) = \varnothing$
  $\text{procReady} \leftarrow next_free[q]$
  $start(t) \leftarrow \max(\text{predReady}, \text{procReady})$
  $end(t) \leftarrow start(t) + PT^p(t, q)$
  $next_free[q] \leftarrow end(t)$
  for all intra-partition messages $m : t \to v$ in $\mathcal{M}^p_{\text{intra}}$ **do** ordered by $\eta^p[m]$
    $q_v \leftarrow \phi_p[v]$
    $path \leftarrow \text{SELECTPATH}(\xi^p[m], \mathcal{P}_{q,q_v})$ ▷ Algorithm 5
    $tx_s(m) \leftarrow end(t)$
    $tx_e(m) \leftarrow tx_s(m) + \text{latency}(path, m)$
    enforce $start(v) \leftarrow \max(start(v), tx_e(m))$ ▷ propagate to successors if needed
  end for
end for
$ms^p \leftarrow \max_t end(t)$
return $\text{sched}_p, \text{ms}_p, \varphi_p$

4.6. Time-Budget Mutation and Slack Distribution

The system-level GA includes the time-budget mutation operator described in Algorithm 9, which dynamically adjusts the partition time budgets during the search. Instead of applying blind mutation to the time-budget vector, the operator adjusts the budgets based on the feedback obtained from the previous evaluation. The numerical settings used by the operator are reported in Table 3.

During mutation, partitions that finish significantly earlier than their assigned budget act as *donors*, whereas partitions whose execution time approaches or exceeds their allocated budget act as *receivers*. The budgets of donor partitions are progressively reduced, while the budgets of receiver partitions are increased according to their observed timing deficits. This adjustment shifts temporal margin toward partitions that require additional execution time while reducing unnecessary surplus in over-provisioned partitions.

By continuously balancing slack among partitions, the system-level search can escape stagnation caused by poor initial budget assignments. Over successive generations, the mutation operator progressively stabilizes the budget distribution so that partitions neither suffer from excessive deadline pressure nor accumulate unnecessary slack. Empirically, this results in faster convergence toward feasible end-to-end schedules because the evolutionary search avoids exploring obviously infeasible budget allocations.

Table 3. Parameters used by the time-budget mutation operator in Algorithm 9.

Parameter	Algorithm Notation	Value
Safety-margin ratio	δ	0.03
Lower stability-band limit	$band_{\text{lo}}$	$target + 0.006\,TU$
Upper stability-band limit	$band_{\text{hi}}$	$target + 0.018\,TU$
Proportional budget-increase factor	Increase toward $target$	0.33
Proportional budget-decrease factor	Decrease toward $band_{\text{lo}}$	0.22
Perturbation fraction	Small bounded perturbation	0.007

Algorithm 9 MUTATETIMEBUDGET

Input: Individual $ind = [\,\pi \mid \Phi \mid \rho \mid TB\,]$ with stored partition makespans $makespans[p]$ and violations $\mathrm{viol}[p]$
Output: Updated TB aligned with permutation π

Parameters: use the time-budget mutation settings reported in Table 3.
for $i = 0$ to $|\pi| - 1$ **do**
  $p \leftarrow \pi[i]$
  $tb \leftarrow TB[i]$
  $ms \leftarrow makespans[p]$
  $v \leftarrow \mathrm{viol}[p]$
  $target \leftarrow ms(1 + \delta)$ ▷ safety margin
  Define stability band $(band_{\mathrm{lo}}, band_{\mathrm{hi}})$ around $target$
  if $v > 0$ **or** $tb < ms$ **then** ▷ deficit
    Increase $TB[i]$ proportionally toward $target$
  else if $tb \geq band_{\mathrm{hi}}$ **then** ▷ surplus
    Decrease $TB[i]$ proportionally toward $band_{\mathrm{lo}}$
  else if $tb < target$ **then**
    Slightly increase $TB[i]$ toward $band_{\mathrm{lo}}$
  else
    Keep $TB[i]$ unchanged
  end if
  Optionally apply a small bounded perturbation
end for
return TB

5. Performance Evaluation

This section evaluates the proposed hierarchical genetic algorithm (HGA) with respect to end-to-end schedulability, global makespan, and convergence behavior. The evaluation comprises baseline benchmarking, runtime and convergence analyses, deadline-sensitivity experiments, a controlled ablation study, and an automotive-characteristic case study.

5.1. Evaluation Setup

The proposed scheduler was implemented in Python 3.10 using the DEAP evolutionary computation framework [39]. The evaluation used synthetic Directed Acyclic Graph (DAG) workloads with 100, 250, and 500 tasks. The use of synthetic DAGs is well established in scheduling research, where generators such as TGFF [40] and SDF3 [41] are widely used to enable controlled and repeatable benchmarking across diverse application structures. Following this methodology, the DAGs in this work were generated using the Stanford Network Analysis Platform (SNAP), which supports systematic variation in graph size and structural properties, including dependency density and in- and out-degree distributions. This enables reproducible evaluation across heterogeneous workload patterns while preserving comparability with prior DAG-based scheduling studies [42].

These workloads were evaluated on a fixed Edge–Fog–Cloud platform model represented as a graph of compute and communication resources.

The compute infrastructure comprises four system-level execution locations, $\mathcal{C} = \{\mathrm{FE}, \mathrm{C1}, \mathrm{C2}, \mathrm{C3}\}$. The location FE contains the evaluated Edge–Fog resources, whereas C1, C2, and C3 contain the evaluated Cloud resources. Each execution location c provides a set N_c of individual End Systems available to the partition-level scheduler. For the communication infrastructure, a finite set of deterministic candidate routes was computed offline on the platform graph using a K-shortest-path method for each relevant source–destination pair [43]. During scheduling, routing decisions were then made by selecting from this precomputed path catalog. The same platform model was used across all workload sizes and deadline settings, so differences between the workload results are not caused by changes in the underlying execution infrastructure.

Within this model, application DAG vertices represent tasks and DAG edges represent inter-task messages. Tasks are associated with execution-time parameters, such as worst-case execution times, while messages incur deterministic communication costs once routing paths are selected from the precomputed path catalog. This allows both computation placement and communication-path selection to be evaluated consistently within the same scheduling framework. The main platform and workload parameters used in the evaluation are summarized in Table 4.

Table 4. Platform and workload parameters used in the evaluation.

Parameter	Evaluated Configuration
Execution locations	FE, C1, C2, C3
End Systems per execution location	FE: 19; C1: 25; C2: 25; C3: 25
Candidate-path catalog	$K = 4$ per relevant End-System pair
Local TSN forwarding hop	1 TU
DCI forwarding hop	10 TU
Workload generation	SNAP-generated DAGs and execution-time values
100T workload	100 tasks; 212 dependencies/messages
250T workload	250 tasks; 523 dependencies/messages
500T workload	500 tasks; 1038 dependencies/messages

For the evaluated synthetic workloads, the generated execution-time values were used directly; no fixed tier-to-tier execution-time ratio was imposed.

The baseline application deadlines were derived from the makespan of a separate initial list-scheduling heuristic applied to each workload and correspond to the 100% deadline setting used in the main benchmark comparison. This heuristic greedily assigns topologically sorted tasks to the available processors according to earliest finish time. It was used only to generate the reference deadline and is distinct from the HEFT implementation evaluated in the comparison. Because the two procedures used different placement and routing configurations, their resulting makespans are not expected to coincide.

The baseline deadlines were set to 2600, 2700, and 4300 TU for the 100T, 250T, and 500T workloads, respectively. Additional experiments with tightened deadlines of 90%, 80%, and 70% of these baseline values are presented later in the sensitivity analysis to evaluate the robustness of the proposed approach under reduced timing slack.

At the system level, the optimization jointly minimizes total partition budget violations and global lateness, where lateness is defined as the difference between the global makespan and the application deadline. At the partition level, the local scheduler minimizes partition makespan while penalizing partition budget overruns. This objective hierarchy aligns with the hierarchical feasibility model, in which partition-level schedules are evaluated against local time-budget constraints within the end-to-end reconstruction process.

Tables 5 and 6 summarize the parameters used for the system-level and partition-level genetic algorithms, respectively.

The GA parameters in Tables 5 and 6 were fixed across all workload sizes and deadline settings to maintain a consistent computational budget. The system-level population size of 16 was selected as a practical trade-off between population diversity and the computational cost of the nested optimization, since evaluating each system-level individual requires executing a partition-level GA for every partition. Similarly, 60 system-level generations were used to provide sufficient search and stabilization time under the evaluated configuration. As reported in Section 5.5, the cumulative partition-budget violation reached zero by generation 31 at the latest in the reported convergence scenarios, leaving additional generations for stabilization and end-to-end objective improvement. These settings are not claimed to be globally optimal.

Table 5. System-level GA parameters.

Parameter	Value
Population size	16
Generations	60
Crossover probability	0.80
Mutation probability	0.70
Fitness objective 1	Cumulative partition budget violations
Fitness objective 2	Positive global lateness
Selection strategy	NSGA-II

Table 6. Partition-level GA parameters.

Parameter	Value
Population size	16
Generations	20
Crossover probability	0.70
Mutation probability	0.70
Fitness objective 1	Partition makespan
Fitness objective 2	Partition budget overruns

HEFT and Round-Robin are used as scoped external reference schedulers for heterogeneous task scheduling. Since they do not include hierarchical partition-level time budgets, feedback-guided budget adaptation, or route-selection genes, they are not used to isolate the contribution of individual HGA mechanisms. The controlled variants in Section 5.7 specifically isolate the contribution of adaptive time-budget reallocation while retaining the remaining hierarchical scheduling structure.

5.2. Scoped Reference Schedulers

This subsection specifies the evaluated HEFT and Round-Robin reference configurations.

5.2.1. Heterogeneous Earliest-Finish-Time (HEFT)

HEFT [19] prioritizes tasks according to upward rank and assigns each task to the processing element that minimizes its earliest finish time. It is precedence-aware and communication-aware at the cost-model level. In the present evaluation, HEFT was applied using static, predefined partition-to-execution-location assignments. Therefore, each task could be assigned only to processing elements belonging to the End-System set of the execution location predefined for its partition, and HEFT did not dynamically change the partition allocation among FE, C1, C2, and C3.

HEFT evaluated communication using the same bounded cost function $L(m, p)$ as the proposed framework. For each communicating endpoint pair, one feasible path was randomly selected from the corresponding precomputed path catalog before scheduling and was then kept fixed throughout the HEFT evaluation. HEFT therefore accounted for the bounded latency of the selected path but did not optimize path selection among the alternative catalog paths. It also did not perform hierarchical system–partition co-design or adaptive time-budget reallocation.

5.2.2. Round-Robin (RR)

Round-Robin dispatches ready tasks cyclically according to a fixed service order. Although simple and robust, it does not exploit critical-path structure, does not coordinate decisions across partitions, and does not adapt time budgets based on feasibility feedback.

5.3. Benchmark Results Under Baseline Deadlines

Table 7 summarizes the final results obtained for the three workload sizes under the baseline deadline setting.

Table 7. Global schedulability results under the baseline deadline setting. HGA values are reported as mean ± standard deviation over ten completed independent runs; all timing values are in TU.

Workload	Deadline	HGA Makespan	Global Lateness	Violation Sum
100T	2600	2408.6 ± 87.1	0.0 ± 0.0	0.0 ± 0.0
250T	2700	2271.7 ± 242.0	0.0 ± 0.0	0.0 ± 0.0
500T	4300	4016.9 ± 153.2	0.0 ± 0.0	0.0 ± 0.0

Across the 10 independent runs launched for each workload, all 10 completed successfully, and the proposed HGA satisfies both the application deadline and all partition time-budget constraints. The mean global makespans are 2408.6 ± 87.1 TU, 2271.7 ± 242.0 TU, and 4016.9 ± 153.2 TU for the 100T, 250T, and 500T workloads, respectively. The corresponding mean timing margins relative to the baseline deadlines are 191.4 TU, 428.3 TU, and 283.1 TU. It is important to note that the three workloads are distinct SNAP-generated DAG instances, not scaled versions of the same graph. Their dependency structures and execution-time distributions therefore differ, so the task count alone does not monotonically determine the global makespan. Accordingly, the lower mean makespan of the 250T instance is not inconsistent with its larger task count.

To further examine the behavior of the adaptive slack reallocation mechanism, Table 8 reports the final partition budgets and realized partition makespans under the baseline experiments.

Table 8. Partition budgets and realized partition makespans at convergence under the baseline deadlines. Values are reported as mean ± standard deviation over ten completed independent runs; all values are in TU.

Workload	p_{FE}		p_{C1}		p_{C2}		p_{C3}	
	Budget	MS	Budget	MS	Budget	MS	Budget	MS
100T	1995.9 ± 1.8	1988.6 ± 3.9	148.0 ± 0.0	129.0 ± 0.0	252.6 ± 0.9	243.1 ± 0.3	976.2 ± 2.0	941.4 ± 2.0
250T	1137.4 ± 16.4	1119.8 ± 19.8	404.0 ± 5.5	346.1 ± 1.5	826.7 ± 2.0	799.3 ± 2.9	444.0 ± 3.8	400.2 ± 0.8
500T	2363.1 ± 43.3	2297.3 ± 46.8	908.4 ± 1.2	873.6 ± 3.2	426.2 ± 3.9	409.2 ± 1.2	437.9 ± 2.3	419.8 ± 0.8

As shown in Table 8, the partition time budgets converge toward the corresponding realized partition makespans across the evaluated workloads. The adaptive slack reallocation mechanism progressively adjusts each partition budget based on the execution behavior returned by the partition-level scheduler. As a result, the remaining gap between assigned budgets and realized makespans is reduced across partitions, indicating reduced under-provisioning and over-provisioning at convergence.

Table 9 reports the HGA makespans together with the scoped HEFT and Round-Robin reference values. The HGA values are reported as mean ± standard deviation over 10/10 launched and completed independent runs. The reference configurations, described in Section 5.2, do not share the allocation and routing decision space of the HGA. Their values are therefore reported descriptively and are not used for percentage-reduction or statistical-superiority claims.

Table 9. Global makespan under the evaluated HGA and scoped reference configurations. HGA values are reported as mean ± standard deviation over 10/10 launched and completed independent runs.

Workload	HGA	HEFT	RR	Deadline
100T	2408.6 ± 87.1	4329	2714	2600
250T	2271.7 ± 242.0	3848	2698	2700
500T	4016.9 ± 153.2	4726	4958	4300

The observed makespan differences are consistent with the broader decision space available to the proposed HGA, which jointly searches partition allocation, partition execution order, route-selection hints, and partition time budgets. However, the external baseline comparison does not by itself attribute the improvement to any single mechanism.

5.4. Scheduler Runtime and Scalability

The computational cost of the proposed scheduler was evaluated using wall-clock measurements under the baseline deadline setting. Each workload was executed using ten independent random seeds on the high-performance computing infrastructure, with 16 CPU cores and 128 GB of memory allocated to each run. Runtime was measured from scheduler initialization until completion of the optimization and generation of the final output files. Waiting time in the batch-system queue was excluded. In addition to the total runtime, the elapsed time at which the first fully feasible schedule was identified was recorded.

Table 10 shows mean runtimes of 15.28 ± 0.25 h, 14.02 ± 0.17 h, and 28.4 ± 9.67 h for the 100T, 250T, and 500T workloads, respectively. Fully feasible schedules were first found after 9.68 ± 0.17 h, 3.87 ± 0.27 h, and 10.50 ± 1.39 h. The 500T workload required approximately twice the runtime of the 250T workload. However, the difference between 100T and 250T shows that runtime also depends on workload structure and feasibility difficulty, not only on task count. These results confirm the non-trivial computational cost of the nested HGA and support its use for offline schedule synthesis.

Table 10. Measured wall-clock runtime of the proposed HGA under the baseline deadline setting. Values are reported as mean ± standard deviation over ten independent runs. Batch-system queue waiting time is excluded.

Workload	Tasks	Messages	Runtime (h)	Time to First Fully Feasible Schedule (h)
100T	100	212	15.28 ± 0.25	9.68 ± 0.17
250T	250	523	14.02 ± 0.17	3.87 ± 0.27
500T	500	1038	28.4 ± 9.67	10.50 ± 1.39

5.5. Convergence and Adaptive Budget Behavior

Table 11 reports the evolution of the cumulative partition violation sum and global lateness for one representative run of each evaluated workload–deadline scenario, illustrating the convergence behavior of the HGA. In all cases, the partition violation sum converges to zero, indicating that partition-level feasibility is eventually achieved. The table also shows whether this local feasibility is accompanied by zero final global lateness or by a residual end-to-end timing gap at convergence.

With respect to the end-to-end deadline, the results show a more selective behavior. Zero global lateness is achieved in all baseline cases and in several tightened-deadline scenarios, namely 100T at 90%, 250T at 90%, 80%, and 70%, and 500T at 90%. In contrast, the 100T–80%, 100T–70%, 500T–80%, and 500T–70% cases converge with residual lateness values of 228, 488, 115, and 607 TU, respectively. These cases show that local partition

feasibility does not necessarily imply end-to-end feasibility when the imposed deadline becomes too restrictive. Instead, the search converges to a stable compromise solution under the available computation and communication resources.

Table 11. Convergence summary for one representative run of each evaluated scenario.

Workload Size	Deadline Setting	Initial Viol.	Final Viol.	First Gen. Zero Viol.	Initial Late.	Final Late.
100T	100%	1544	0	25	0	0
100T	90%	1688	0	27	149	0
100T	80%	1824	0	29	447	228
100T	70%	1960	0	31	650	488
250T	100%	520	0	10	0	0
250T	90%	649	0	12	0	0
250T	80%	725	0	15	0	0
250T	70%	929	0	15	36	0
500T	100%	1215	0	24	0	0
500T	90%	1239	0	23	65	0
500T	80%	1292	0	30	1026	115
500T	70%	1514	0	31	976	607

To quantify the adaptive budget behavior directly, Table 12 reports the aggregated budget–makespan mismatch, measured as $\sum_p |TB_p - MS_p|$ across all partitions.

Table 12. Reduction in total budget–makespan gap across partitions for the same representative runs reported in Table 11.

Workload Size	Deadline Setting	Initial $\sum_p \lvert TB_p - MS_p \rvert$	Final $\sum_p \lvert TB_p - MS_p \rvert$	Reduction (%)
100T	100%	2394	58	97.6
100T	90%	2416	54	97.8
100T	80%	2430	87	96.4
100T	70%	2441	79	96.8
250T	100%	1044	149	85.7
250T	90%	1047	73	93.0
250T	80%	999	85	91.5
250T	70%	1086	61	94.4
500T	100%	2581	114	95.6
500T	90%	2304	119	94.8
500T	80%	2079	166	92.0
500T	70%	2096	150	92.8

This quantity decreases substantially in every evaluated scenario, with reductions ranging from 85.7% to 97.8%. The reduction confirms that the adaptive slack reallocation mechanism progressively aligns the system-level time budgets with the realized execution demand returned by the partition-level schedulers. In other words, the mutation operator not only repairs violating partitions, but also reduces both over-provisioning and under-provisioning, thereby strengthening the temporal alignment between the system level and the partition level.

Figures 2 and 3 provide a generation-wise view of this adaptation process for representative baseline and tightened-deadline cases. In both cases, the assigned partition budgets progressively move toward the realized partition makespans returned by the partition-level schedulers. For the baseline 100T case, this alignment coincides with zero final global lateness. For the tightened 500T–80% case, the same budget-alignment behavior

is observed even though a residual global lateness remains. This indicates that the adaptive mutation operator stabilizes partition-level budget assignments even in scenarios where the evaluated search retains residual global lateness.

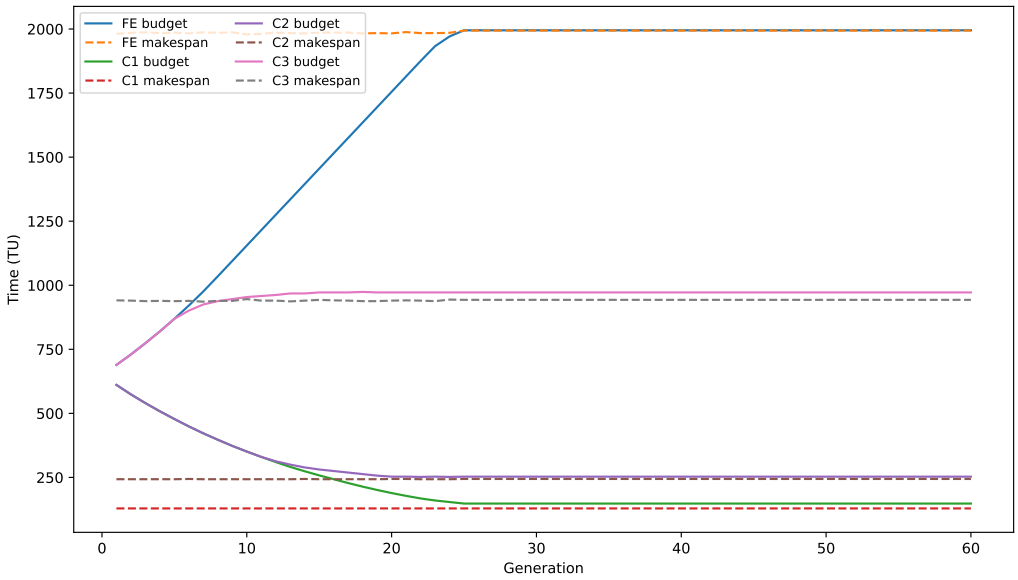


Figure 2. Generation-wise evolution of partition budgets and realized partition makespans for the 100T workload under the baseline deadline setting. The progressive alignment between the assigned budgets and the partition makespans indicates convergence of the partition-level temporal budgets.

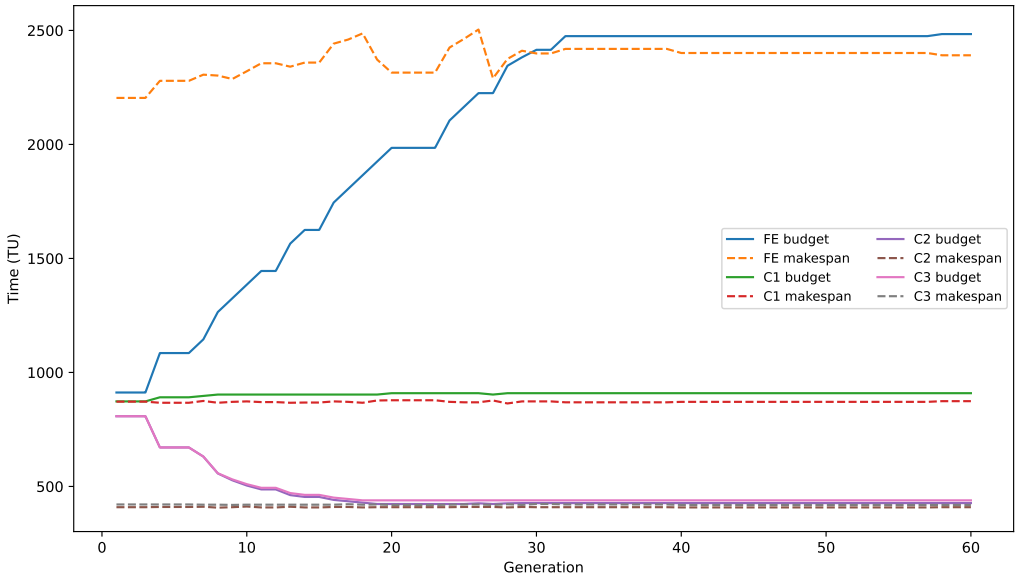


Figure 3. Generation-wise evolution of partition budgets and realized partition makespans for the 500T workload under the 80% tightened deadline setting. Although a residual global lateness remains in this case, the partition budgets still converge toward the realized partition makespans.

To complement the partition-level budget-alignment trajectories, Figures 4 and 5 show the system-level fitness evolution and global makespan for a 500T baseline run.

Figure 4 shows that the cumulative partition-budget violation progressively decreases to zero, while the positive global lateness also becomes zero and remains zero during the later generations. Figure 5 provides the corresponding global view: after fluctuations during the initial exploration phase, the global makespan stabilizes below the $D = 4300$ TU application deadline. For this representative run, feasibility is reached well before the

termination of the 60-generation search, and the recorded solution remains feasible during the remaining generations.

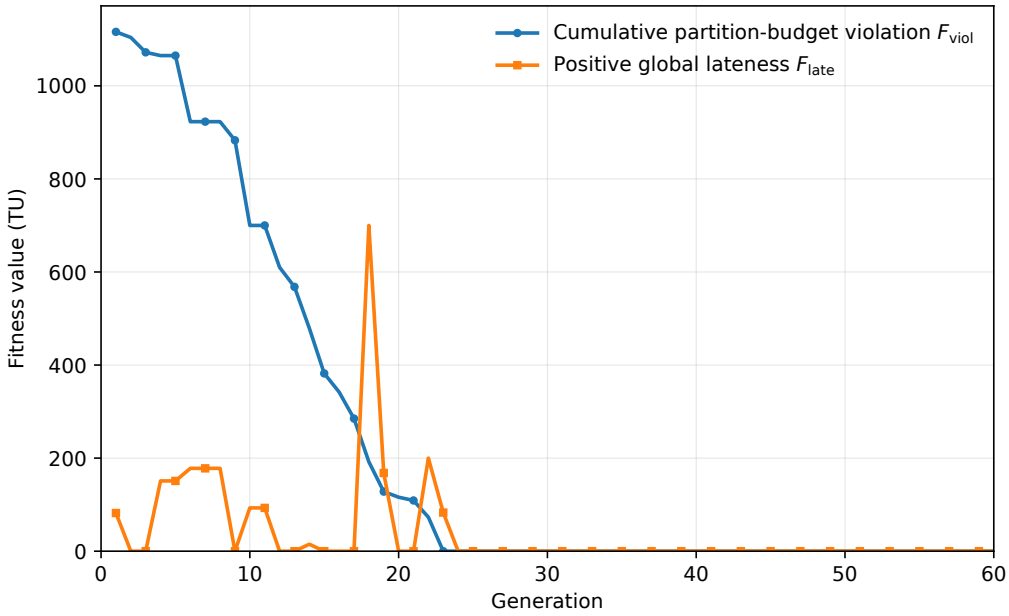


Figure 4. Generation-wise evolution of the system-level fitness objectives for the representative 500T run under the baseline deadline $D = 4300$ TU.

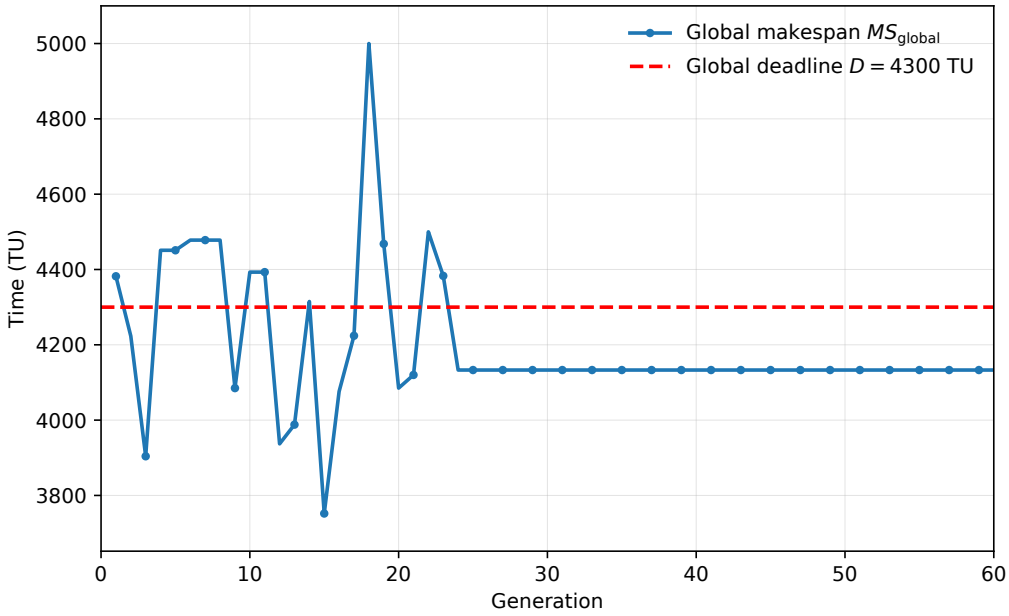


Figure 5. Generation-wise evolution of the global makespan relative to the application deadline $D = 4300$ TU for the representative 500T run.

Taken together, Tables 11 and 12 and Figures 2–5 show that the adaptive slack reallocation mechanism reduces partition-budget violations and aligns the assigned budgets with the realized partition makespans, while achieving system-level feasibility in the representative baseline run.

5.6. Sensitivity to Deadline Tightening

The results further indicate that the proposed HGA is sensitive to progressively tightened end-to-end deadlines. As the deadline is reduced from the baseline setting to 90%,

80%, and 70%, the feasible region of the search space shrinks, and satisfying both partition-level feasibility and global end-to-end feasibility becomes increasingly challenging.

For moderate tightening, the proposed approach remains robust. Several 90% cases and all evaluated 250T cases still achieve zero final global lateness, indicating that the adaptive time-budget negotiation mechanism can absorb a substantial reduction in the global timing margin while preserving end-to-end schedulability. This behavior suggests that, when sufficient structural slack is still available across partitions and communication paths, the HGA can successfully redistribute time budgets and recover a globally feasible solution.

Under stronger tightening, however, the feasible region shrinks significantly. In particular, the 100T–80%, 100T–70%, 500T–80%, and 500T–70% scenarios converge with nonzero residual lateness. This behavior indicates that the algorithm is no longer limited by poor budget assignment alone, but by the reduced global temporal margin imposed by the end-to-end deadline itself. In such cases, the adaptive mutation operator still eliminates partition-level violations and stabilizes the partition budgets, but the evaluated search does not fully remove the remaining timing gap. Further improvements may require additional search effort, alternative parameter settings, deadline relaxation, or changes to the available compute and communication resources.

An additional consequence of the feasibility-oriented objective is that tighter deadlines can lead to lower final makespans. Since F_{late} remains positive while $MS_{\text{global}} > D$, a tighter value of D maintains selection pressure toward shorter end-to-end schedules until the deadline is reached. Consequently, some cases with tighter deadlines obtain substantially shorter schedules than under the baseline deadline. For example, for the 250T workload, the mean makespan decreases from 2271.7 TU under the baseline deadline to 1825.1 TU under the 70% deadline setting. A similar effect is observed in the representative 500T run at the 80% deadline setting, which reaches a final makespan of 3555 TU. This behavior results from the feasibility-oriented formulation of the global-lateness objective rather than from early termination of the evolutionary search.

An important observation is that the performance degradation is gradual rather than abrupt as the deadline is tightened. Even in scenarios where a nonzero residual lateness remains, the search continues to reduce both the violation term and the budget–makespan mismatch, converging to stable compromise solutions. Such behavior is particularly important for hierarchical scheduling, as it shows that the proposed HGA remains stable and effective even when full end-to-end feasibility cannot be achieved.

5.7. Ablation Study of Adaptive Time-Budget Reallocation

The controlled ablation is designed specifically to isolate the effect of adaptive time-budget reallocation. Accordingly, the hierarchical optimization structure and deterministic route-selection mechanism are retained unchanged across all variants. They are treated as fixed design elements of the complete scheduler rather than as independently evaluated mechanisms.

The ablation is restricted to the 100T and 250T workloads to evaluate the mechanism at two distinct problem sizes without repeating the scalability analysis of the complete HGA. The 500T workload is already evaluated in the main benchmark, runtime, convergence, and deadline-sensitivity analyses. Moreover, the complete HGA requires 28.4 ± 9.67 h per run for the 500T workload under the baseline configuration, making a full ablation at this workload size substantially more computationally expensive.

For the 100T and 250T workloads, the ablation is performed at 70%, 80%, 90%, and 100% of the baseline deadline, corresponding to $D = \{1820, 2080, 2340, 2600\}$ TU for 100T and $D = \{1890, 2160, 2430, 2700\}$ TU for 250T. Each workload–deadline–variant

combination is evaluated using 10 independent random seeds, and all 10 launched runs completed successfully; therefore, no runs were excluded because of non-completion.

The ablation study provides the controlled mechanism-level comparison for the proposed HGA. Three scheduler variants are compared, as summarized in Table 13. *Proposed-HGA* denotes the complete method, where partition time budgets are adapted using the proposed slack reallocation technique. This technique uses feedback from the observed partition makespans, budget violations, and available slack to adjust the time-budget distribution during the system-level search. *HGA-NoFeedback* retains the same hierarchical GA structure and keeps mutable time-budget genes but replaces feedback-guided budget adaptation with bounded random time-budget mutation. This means that the time budgets can still evolve, but their changes are not guided by partition slack or violation feedback. For *HGA-StaticBudget*, the time budget of each partition is computed once before the system-level evolutionary search as $TB(p) = D|\mathcal{T}_p|/|\mathcal{T}|$; that is, each partition receives a share of the global deadline proportional to the number of tasks it contains. The resulting budgets remain fixed throughout the run, while the partition-to-execution-location allocation remains optimized by the system-level HGA.

Table 13. Scheduler variants used in the ablation study.

Variant	TB Genes	TB Mutation	Partition Feedback
Proposed-HGA	Yes	Yes	Yes
HGA-NoFeedback	Yes	Yes	No
HGA-StaticBudget	Fixed	No	No

A run is considered deadline-feasible if $MS_{\text{global}} \leq D$. It is considered partition-time-budget feasible if

$$\sum_{p \in \mathcal{P}} \max(0, MS_p - TB_p) = 0.$$

A run is fully feasible only when both conditions are satisfied. This distinction is important because a schedule may satisfy the global end-to-end deadline while still violating one or more partition time budgets.

Table 14 reports the feasibility rates and the generation at which all partition time budgets are first satisfied. The results show that fixed workload-aware time budgets are not sufficient for hierarchical schedulability in this setting. HGA-StaticBudget meets the global deadline in some cases, especially under relaxed deadlines, but it fails to meet partition time budgets in any evaluated setting. For example, in the 250T workload at $D = 2430$ and $D = 2700$, HGA-StaticBudget meets the global deadline in 100% of the runs, but the partition time budgets are never all satisfied. This confirms that global deadline feasibility alone does not guarantee feasibility with respect to the partition-level timing constraints used by the hierarchical scheduler.

HGA-NoFeedback improves over HGA-StaticBudget because its time-budget genes remain mutable. In several relaxed cases, it reaches the same fully feasible rate as Proposed-HGA. This indicates that time-budget mutability is useful in itself. However, the comparison also shows that feedback-free mutation is less reliable under tighter timing conditions. In the 100T workload at $D = 1820$, HGA-NoFeedback does not satisfy the partition time budgets in any run, whereas Proposed-HGA satisfies them in all runs. Similarly, at $D = 2080$, HGA-NoFeedback satisfies the partition time budgets in only 40% of the runs, while Proposed-HGA reaches 100%. For the 250T workload at the tightest deadline, $D = 1890$, Proposed-HGA is fully feasible in all runs, whereas HGA-NoFeedback is fully feasible in 50% of the runs. These cases indicate that the proposed feedback mechanism is

most beneficial when the timing margin is limited and budget changes must be directed toward partitions with observed timing pressure.

Table 14. Feasibility and partition-budget convergence results for the ablation study over 10 launched and completed independent runs per workload–deadline–variant combination. The first partition-budget-feasible generation is computed over the runs in which all partition time budgets were satisfied.

Workload	Scheduler Variant	Deadline Value (TU)	Launched/ Completed	Deadline Met (%)	Budgets Met (%)	Fully Feasible (%)	First Feasible Generation
100T	Proposed-HGA	1820	10/10	0.0	100.0	0.0	31.0 ± 0.0
100T	HGA-NoFeedback	1820	10/10	0.0	0.0	0.0	–
100T	HGA-StaticBudget	1820	10/10	0.0	0.0	0.0	–
100T	Proposed-HGA	2080	10/10	0.0	100.0	0.0	29.0 ± 0.0
100T	HGA-NoFeedback	2080	10/10	0.0	40.0	0.0	59.0 ± 1.4
100T	HGA-StaticBudget	2080	10/10	0.0	0.0	0.0	–
100T	Proposed-HGA	2340	10/10	100.0	100.0	100.0	26.9 ± 0.3
100T	HGA-NoFeedback	2340	10/10	100.0	100.0	100.0	54.0 ± 2.5
100T	HGA-StaticBudget	2340	10/10	30.0	0.0	0.0	–
100T	Proposed-HGA	2600	10/10	100.0	100.0	100.0	25.0 ± 0.0
100T	HGA-NoFeedback	2600	10/10	100.0	100.0	100.0	45.0 ± 2.4
100T	HGA-StaticBudget	2600	10/10	90.0	0.0	0.0	–
250T	Proposed-HGA	1890	10/10	100.0	100.0	100.0	16.6 ± 1.3
250T	HGA-NoFeedback	1890	10/10	50.0	100.0	50.0	50.0 ± 2.8
250T	HGA-StaticBudget	1890	10/10	0.0	0.0	0.0	–
250T	Proposed-HGA	2160	10/10	100.0	100.0	100.0	13.4 ± 0.9
250T	HGA-NoFeedback	2160	10/10	100.0	100.0	100.0	35.5 ± 3.7
250T	HGA-StaticBudget	2160	10/10	50.0	0.0	0.0	–
250T	Proposed-HGA	2430	10/10	100.0	100.0	100.0	11.3 ± 0.8
250T	HGA-NoFeedback	2430	10/10	100.0	100.0	100.0	27.0 ± 3.5
250T	HGA-StaticBudget	2430	10/10	100.0	0.0	0.0	–
250T	Proposed-HGA	2700	10/10	100.0	100.0	100.0	9.4 ± 0.7
250T	HGA-NoFeedback	2700	10/10	100.0	100.0	100.0	23.3 ± 3.1
250T	HGA-StaticBudget	2700	10/10	100.0	0.0	0.0	–

The convergence behavior further supports this interpretation. Whenever both Proposed-HGA and HGA-NoFeedback eventually satisfy all partition time budgets, Proposed-HGA reaches this condition earlier. For the 100T workload, the first partition-feasible generation is reached by Proposed-HGA between generations 25 and 31 on average, compared with generations 45 to 59 for HGA-NoFeedback. For the 250T workload, Proposed-HGA reaches partition-time-budget feasibility between generations 9.4 and 16.6 on average, while HGA-NoFeedback requires between generations 23.3 and 50. This suggests that slack and violation feedback serve as a convergence aid by guiding the search toward useful budget changes rather than relying solely on undirected random budget perturbations.

Table 15 provides a second view of the same behavior by reporting the final per-partition budget–makespan difference, computed as $\Delta TB_p = TB_p - MS_p$. Smaller positive values indicate tighter alignment between the assigned time budget and the realized partition makespan, while negative values indicate a partition time-budget violation. The results show that HGA-NoFeedback often reaches feasibility by leaving substantially larger unused time budgets. For example, in the 100T workload at $D = 2600$, HGA-NoFeedback leaves large positive differences in C1 and C2, whereas Proposed-HGA keeps all partition differences comparatively small. A similar pattern appears in the 250T workload at $D = 2700$, where HGA-NoFeedback leaves considerably larger positive differences in C1 and C3 than Proposed-HGA. Therefore, HGA-NoFeedback can be effective in relaxed cases, but its final budget distribution is less tightly aligned with the realized partition makespans.

Table 15. Mean ± standard deviation of the final per-partition budget–makespan difference after convergence. The value is computed as $\Delta TB_p = TB_p - MS_p$, where smaller positive values indicate tighter alignment and negative values indicate partition time-budget violations.

Workload	Scheduler	Deadline D	p_{FE} ΔTB	$p_{C1}\Delta TB$	$p_{C2}\Delta TB$	$p_{C3}\Delta TB$
100T	Proposed-HGA	1820	29.4 ± 9.0	5.0 ± 0.0	9.3 ± 1.0	36.8 ± 4.0
100T	HGA-NoFeedback	1820	-190.8 ± 42.9	296.5 ± 115.4	204.0 ± 91.0	121.0 ± 91.2
100T	Proposed-HGA	2080	43.3 ± 12.0	5.0 ± 0.0	9.9 ± 0.9	35.3 ± 3.0
100T	HGA-NoFeedback	2080	-9.9 ± 35.1	386.1 ± 131.9	164.0 ± 116.1	194.7 ± 202.2
100T	Proposed-HGA	2340	9.7 ± 3.8	5.0 ± 0.0	9.1 ± 0.6	34.8 ± 2.5
100T	HGA-NoFeedback	2340	27.4 ± 11.7	475.4 ± 137.2	294.4 ± 110.3	119.1 ± 68.8
100T	Proposed-HGA	2600	7.3 ± 4.7	19.0 ± 0.0	9.4 ± 1.0	34.8 ± 2.3
100T	HGA-NoFeedback	2600	12.2 ± 11.1	466.4 ± 159.9	372.2 ± 110.6	150.6 ± 126.5
250T	Proposed-HGA	1890	56.8 ± 39.0	13.3 ± 1.7	28.8 ± 5.6	17.1 ± 0.9
250T	HGA-NoFeedback	1890	30.0 ± 8.5	101.0 ± 114.6	34.5 ± 43.1	112.0 ± 45.3
250T	Proposed-HGA	2160	19.6 ± 21.2	15.4 ± 1.6	26.0 ± 2.3	17.4 ± 1.3
250T	HGA-NoFeedback	2160	32.9 ± 34.0	148.9 ± 58.8	57.5 ± 88.4	124.5 ± 97.2
250T	Proposed-HGA	2430	26.8 ± 24.4	32.0 ± 5.5	25.0 ± 5.3	22.1 ± 4.5
250T	HGA-NoFeedback	2430	27.8 ± 30.5	193.4 ± 77.4	46.5 ± 72.8	105.5 ± 64.5
250T	Proposed-HGA	2700	17.7 ± 16.0	57.9 ± 6.7	27.3 ± 3.6	43.8 ± 3.5
250T	HGA-NoFeedback	2700	42.1 ± 33.4	275.4 ± 81.3	98.7 ± 52.1	270.8 ± 64.7

The final timing statistics in Table 16 should therefore be interpreted together with the feasibility and budget-alignment results. HGA-NoFeedback can occasionally obtain a similar or even lower mean makespan in relaxed cases because its feedback-free budget mutation can still explore feasible allocations. However, makespan alone does not capture whether the partition time budgets are satisfied or whether the assigned budgets are tightly aligned with partition execution demand. Across the evaluated settings, Proposed-HGA consistently eliminates partition time-budget violations, reaches partition-time-budget feasibility earlier, and produces tighter budget–makespan alignment.

Table 16. Mean ± standard deviation of final timing metrics in the ablation study over 10 independent runs.

Workload	Scheduler	Deadline D	Final Makespan	Partition Time-Budget Violation	Positive Global Lateness
100T	Proposed-HGA	1820	2305.6 ± 4.9	0.0 ± 0.0	485.6 ± 4.9
100T	HGA-NoFeedback	1820	2475.5 ± 139.7	190.8 ± 42.9	655.5 ± 139.7
100T	HGA-StaticBudget	1820	2394.6 ± 112.1	1469.3 ± 1.5	574.6 ± 112.1
100T	Proposed-HGA	2080	2307.7 ± 1.5	0.0 ± 0.0	227.7 ± 1.5
100T	HGA-NoFeedback	2080	2379.3 ± 103.7	27.9 ± 38.7	299.3 ± 103.7
100T	HGA-StaticBudget	2080	2394.6 ± 112.1	1209.3 ± 1.5	314.6 ± 112.1
100T	Proposed-HGA	2340	2324.0 ± 10.3	0.0 ± 0.0	0.0 ± 0.0
100T	HGA-NoFeedback	2340	2323.2 ± 4.9	0.0 ± 0.0	0.0 ± 0.0
100T	HGA-StaticBudget	2340	2446.6 ± 136.7	948.6 ± 1.9	116.6 ± 125.9
100T	Proposed-HGA	2600	2408.6 ± 87.1	0.0 ± 0.0	0.0 ± 0.0
100T	HGA-NoFeedback	2600	2382.6 ± 53.7	0.0 ± 0.0	0.0 ± 0.0
100T	HGA-StaticBudget	2600	2430.6 ± 112.4	784.7 ± 0.5	2.4 ± 7.6
250T	Proposed-HGA	1890	1825.1 ± 46.6	0.0 ± 0.0	0.0 ± 0.0
250T	HGA-NoFeedback	1890	1857.5 ± 65.8	0.0 ± 0.0	7.0 ± 9.9
250T	HGA-StaticBudget	1890	2293.1 ± 118.7	1027.2 ± 0.8	403.1 ± 118.7

Table 16. *Cont.*

Workload	Scheduler	Deadline D	Final Makespan	Partition Time-Budget Violation	Positive Global Lateness
250T	Proposed-HGA	2160	2021.4 ± 99.2	0.0 ± 0.0	0.0 ± 0.0
250T	HGA-NoFeedback	2160	2038.6 ± 67.2	0.0 ± 0.0	0.0 ± 0.0
250T	HGA-StaticBudget	2160	2229.4 ± 209.3	954.6 ± 0.5	109.8 ± 174.7
250T	Proposed-HGA	2430	2161.8 ± 129.4	0.0 ± 0.0	0.0 ± 0.0
250T	HGA-NoFeedback	2430	2194.4 ± 136.9	0.0 ± 0.0	0.0 ± 0.0
250T	HGA-StaticBudget	2430	2139.1 ± 119.9	881.7 ± 0.7	0.0 ± 0.0
250T	Proposed-HGA	2700	2271.7 ± 242.0	0.0 ± 0.0	0.0 ± 0.0
250T	HGA-NoFeedback	2700	2181.6 ± 169.1	0.0 ± 0.0	0.0 ± 0.0
250T	HGA-StaticBudget	2700	2310.8 ± 199.4	808.2 ± 0.9	0.0 ± 0.0

The ablation results show that adaptive reallocation improves the reliability of partition-time-budget satisfaction. Its principal benefits are reflected in the earlier achievement of partition-budget feasibility and in the tighter alignment between assigned budgets and realized partition makespans. Final makespan is reported as a complementary outcome, but it is not expected to improve uniformly relative to the controlled variants.

Statistical Significance of the Reallocation Effects

To determine whether the observed ablation differences were consistent across random seeds, statistical tests were applied to three metrics that directly characterize the intended effect of adaptive time-budget reallocation. The first metric is the aggregate budget–makespan mismatch,

$$G = \sum_{p \in P} \left| TB_p - MS_p \right|, \tag{13}$$

where a smaller value indicates tighter alignment between the system-level partition budgets and the makespans realized by the partition-level schedulers. The second metric is final partition-time-budget feasibility, defined by

$$PF = \begin{cases} 1, & \text{if } MS_p \leq TB_p \text{ for all } p \in P, \\ 0, & \text{otherwise.} \end{cases} \tag{14}$$

The third metric is the first generation FG at which all partition time budgets are satisfied.

The comparisons used successfully completed runs matched by workload, deadline, and random seed. Since the compared observations were paired and the limited sample sizes did not justify assuming normally distributed differences, the continuous metrics G and FG were evaluated using two-sided Wilcoxon signed-rank tests. This nonparametric paired procedure is recommended for comparisons of evolutionary optimization algorithms [44]. Final partition feasibility is a paired binary outcome and was therefore evaluated using the exact two-sided McNemar test [45].

Holm's step-down procedure [46] was used to control the family-wise error associated with multiple comparisons. For G and PF, the correction was applied to the two planned comparisons within each workload–deadline setting: Proposed-HGA versus HGA-NoFeedback and Proposed-HGA versus HGA-StaticBudget. Since HGA-StaticBudget did not reach partition-time-budget feasibility, the FG analysis was restricted to Proposed-HGA versus HGA-NoFeedback, with correction across the workload–deadline cases for which paired feasible-generation values were available. Statistical significance was assessed at $\alpha = 0.05$.

Table 17 confirms the mechanism-level effects observed in Tables 14–16. Proposed-HGA achieves a significantly lower aggregate budget–makespan mismatch than the corresponding controlled variant in all planned comparisons except the comparison with HGA-NoFeedback for the 250T workload at $D = 1890$, where the difference is not statistically significant.

Proposed-HGA also achieves significantly higher final partition-time-budget feasibility than HGA-StaticBudget in every evaluated setting and than HGA-NoFeedback for the 100T workload at $D = 1820$ and $D = 2080$. In the remaining comparisons with HGA-NoFeedback, both methods attain final partition-time-budget feasibility in the same matched runs, resulting in $p_{\mathrm{PF}} = 1.000$. When both Proposed-HGA and HGA-NoFeedback reach partition-time-budget feasibility, Proposed-HGA reaches this condition significantly earlier for 100T at $D = 2340$ and $D = 2600$, and for 250T at $D = 2160$, $D = 2430$, and $D = 2700$. These results indicate that adaptive reallocation primarily improves budget alignment, feasibility reliability, and convergence speed rather than uniformly reducing the final global makespan.

Table 17. Statistical significance of the adaptive time-budget reallocation effects. The reported values are Holm-adjusted p-values. p_G evaluates the aggregate budget–makespan mismatch, p_{PF} evaluates final partition-time-budget feasibility, and p_{FG} evaluates the first partition-feasible generation. Bold values indicate statistical significance at $\alpha = 0.05$.

Workload	Deadline	Comparison	p_G	p_{PF}	p_{FG}
100T	1820	Proposed vs. NoFeedback	**0.016**	**0.004**	–
100T	1820	Proposed vs. StaticBudget	**0.008**	**0.004**	–
100T	2080	Proposed vs. NoFeedback	**0.016**	**0.031**	0.250
100T	2080	Proposed vs. StaticBudget	**0.004**	**0.004**	–
100T	2340	Proposed vs. NoFeedback	**0.004**	1.000	**0.027**
100T	2340	Proposed vs. StaticBudget	**0.004**	**0.004**	–
100T	2600	Proposed vs. NoFeedback	**0.008**	1.000	**0.039**
100T	2600	Proposed vs. StaticBudget	**0.004**	**0.004**	–
250T	1890	Proposed vs. NoFeedback	0.500	1.000	0.500
250T	1890	Proposed vs. StaticBudget	**0.004**	**0.004**	–
250T	2160	Proposed vs. NoFeedback	**0.016**	1.000	**0.047**
250T	2160	Proposed vs. StaticBudget	**0.016**	**0.004**	–
250T	2430	Proposed vs. NoFeedback	**0.008**	1.000	**0.039**
250T	2430	Proposed vs. StaticBudget	**0.008**	**0.004**	–
250T	2700	Proposed vs. NoFeedback	**0.004**	1.000	**0.027**
250T	2700	Proposed vs. StaticBudget	**0.004**	**0.004**	–

The tests use completed runs matched by random seed. For G and PF, all 10 matched seed pairs were available for every comparison. For FG, only matched seeds for which both compared variants reached partition-time-budget feasibility were included. This resulted in $n = 4$ matched pairs for the 100T case at $D = 2080$ and $n = 10$ matched pairs for the remaining reported FG comparisons. For G and PF, Holm's correction was applied separately within each workload–deadline setting to the two planned comparisons: Proposed-HGA versus HGA-NoFeedback and Proposed-HGA versus HGA-StaticBudget. A dash indicates that the first feasible generation could not be compared because the corresponding variant did not reach partition-time-budget feasibility.

5.8. Automotive-Characteristic DAG Case Study

To complement the scalable synthetic DAG experiments, an application-oriented case study based on automotive benchmark characteristics is included [47]. The reference benchmark does not provide a ready-to-run input model for the proposed scheduler. Instead, it reports characteristics of automotive software, including runnable activation patterns, label-based communication, execution-time distributions, and cause-and-effect chain timing requirements. These characteristics were used to construct a scheduler-compatible automotive-characteristic workload instance for the present evaluation.

The constructed workload contains 100 jobs, 212 message dependencies, and 10 end-to-end chains. Its purpose is not to reproduce a proprietary vehicle application but to introduce an automotive-oriented dependency and timing structure into the evaluation.

The automotive-characteristic workload is evaluated using the same proposed HGA pipeline as the previous experiments. The system-level search determines allocation, execution order, deterministic route choices, and time-budget assignment, while the partition-level schedulers synthesize local task schedules under the assigned budgets. The global application deadline is set to $D = 2600$ TU. This global deadline applies to the completion of the entire workload and is distinct from the individual deadline associated with each cause-and-effect chain.

HEFT is evaluated using the same implementation and configuration employed to generate the HEFT results provided in Section 5.3. Its different makespan in the present case study results from applying that implementation to the automotive-characteristic workload rather than the synthetic 100-task workload.

The chain deadlines are assigned according to the construction rule described by Kramer et al. [47]: the end-to-end constraint of a chain is obtained by adding the periods of its involved activation patterns. Let $\mathcal{A}_c$ denote the activation patterns involved in chain c. Its deadline is therefore

$$D_c = \sum_{a \in \mathcal{A}_c} T_a, \tag{15}$$

where T_a is the normalized period assigned to activation pattern a. In the constructed scheduler input, the 10 ms, 20 ms, and 100 ms patterns are represented by 1000, 2000, and 10,000 TU, respectively. The angle-synchronous pattern is represented by 1000 TU at the selected operating point. Consequently, a chain involving only a 20 ms pattern receives a deadline of 2000 TU, whereas chains involving 10 ms and 100 ms patterns, or 10 ms, 20 ms, and 100 ms patterns, receive deadlines of 11,000 and 13,000 TU, respectively.

For a cause-and-effect chain $c = (v_1, \ldots, v_k)$, its scheduled latency is measured as

$$L_c = F(v_k) - S(v_1), \tag{16}$$

where $S(v_1)$ is the scheduled start time of the first chain job and $F(v_k)$ is the scheduled finish time of the last chain job. Chain c satisfies its deadline when $L_c \leq D_c$. The ratio L_c / D_c is reported to allow comparison between chains having different deadlines. Total chain tardiness is defined as $\sum_c \max(0, L_c - D_c)$.

The chain metrics are computed after schedule construction. Neither the chain types nor the chain deadlines are used in the HGA fitness function or by HEFT when constructing the evaluated schedules. The chains are classified as sensor-to-actuator or fault-reaction chains, following the two cause-and-effect-chain categories described in the automotive benchmark.

As shown in Table 18, the proposed HGA obtains a global makespan of 2457 TU, compared with 3494 TU for HEFT. The HGA meets the global deadline with a margin of 143 TU, whereas HEFT exceeds it by 894 TU. The HGA schedule also has zero cumulative partition time-budget violation. The first HGA schedule satisfying both the global deadline and all partition-budget constraints is found at generation 18 and is retained until convergence.

Table 18. Global and chain-level results for the automotive-characteristic workload. Global slack is calculated as $D - MS_{\text{global}}$. All timing values are in TU.

Method	Makespan	Global Slack	Global Met	Chains Met	Max. *L*/*D*	Total Chain Tardiness
Proposed HGA	2457	143	Yes	9/10	1.748	748
HEFT	3494	−894	No	9/10	1.883	883

At the chain level, as can be shown in Table 19, both methods satisfy 9 of the 10 deadlines. HGA nevertheless produces a lower latency for 9 of the 10 chains and reduces the maximum latency-to-deadline ratio from 1.883 to 1.748. The exception is chain c_1, for which HEFT obtains a lower latency, although both schedules remain well within its 2000-TU deadline. Chain c_4 is the only chain that misses its deadline under both methods. HGA produces a latency of 1748 TU and HEFT produces 1883 TU, compared with the chain deadline of 1000 TU. HGA therefore reduces the deadline overrun from 883 TU to 748 TU but does not eliminate it.

These findings distinguish global application feasibility from cause-and-effect-chain feasibility. The HGA schedule satisfies the global deadline and all partition-budget constraints, but it does not satisfy every chain-specific deadline. The case study therefore demonstrates global-deadline and partition-budget feasibility and generally lower chain latencies relative to HEFT, rather than complete chain-level feasibility.

The chain-level latency and deadline-satisfaction results constitute the primary automotive-specific metrics in this case study, while global makespan is retained as a complementary system-level metric. Because the present evaluation contains one constructed workload instance and one deterministic scheduled occurrence of each chain, the results should be interpreted as an illustrative application-oriented case study rather than a statistical evaluation of automotive workloads. Occurrence-to-occurrence jitter cannot be meaningfully estimated without repeated chain activations and a model of execution- or communication-time variability; such an evaluation remains outside the scope of the present work.

Table 19. Per-cause-and-effect-chain latency and deadline satisfaction for the automotive-characteristic workload.

Chain	Chain Type	Deadline	HGA Latency	HGA *L*/*D*	HGA Met	HEFT Latency	HEFT *L*/*D*	HEFT Met
c_1	Sensor–actuator	2000	389	0.195	Yes	243	0.122	Yes
c_2	Fault reaction	10,000	1217	0.122	Yes	1404	0.140	Yes
c_3	Sensor–actuator	1000	333	0.333	Yes	747	0.747	Yes
c_4	Fault reaction	1000	1748	1.748	No	1883	1.883	No
c_5	Sensor–actuator	10,000	1638	0.164	Yes	2702	0.270	Yes
c_6	Fault reaction	10,000	471	0.047	Yes	1258	0.126	Yes
c_7	Sensor–actuator	2000	598	0.299	Yes	951	0.476	Yes
c_8	Fault reaction	3000	966	0.322	Yes	2579	0.860	Yes
c_9	Sensor–actuator	11,000	2329	0.212	Yes	2457	0.223	Yes
c_{10}	Fault reaction	13,000	2217	0.171	Yes	3396	0.261	Yes

6. Discussion

The results show that the proposed HGA with adaptive time-budget reallocation improves the alignment between assigned partition budgets and the partition makespans realized during scheduling. In the ablation study, the proposed HGA reaches partition-budget feasibility more consistently and in fewer generations than the HGA-NoFeedback and HGA-StaticBudget variants.

An important insight is that partition-level feasibility and end-to-end feasibility must be treated as related but distinct objectives. The experiments show that adaptive time-budget reallocation is effective in repairing local infeasibility and aligning partition budgets with realized execution demand. However, once the global deadline becomes too restrictive, improving local feasibility alone is not always sufficient to remove the remaining end-to-end timing gap.

Beyond feasibility repair, the observed convergence behavior suggests that adaptive time-budget reallocation improves the temporal coordination between the system-level

and partition-level schedulers by reducing the mismatch between assigned budgets and realized execution demand. This effect is also observed in cases with residual global lateness, indicating that improved partition-level coordination does not necessarily imply end-to-end feasibility under strongly tightened deadlines.

Limitations and Practical Considerations

The measured wall-clock results show that the nested HGA incurs a non-trivial offline synthesis cost. Since the current runtime evaluation is limited to workloads containing up to 500 tasks, the computational behavior beyond this range has not been empirically established. The proposed method should therefore be regarded as an offline schedule-synthesis approach under the evaluated workload sizes, while scalability to larger application models requires further investigation.

Although the proposed HGA computes feasible end-to-end schedules for all baseline cases and several tightened-deadline cases, its runtime temporal guarantees are entirely dependent on the enforcement capabilities of the underlying deterministic network. Establishing the practical limits of this work requires distinguishing between infrastructure assumptions that reflect current industry capabilities and those that remain future architectural requirements.

Currently, IEEE Time-Sensitive Networking (TSN) is an established, deployable standard for providing bounded latency and synchronization at the local Edge and Fog layers [15]. However, extending determinism across wide-area networks via IETF DetNet [16], maintaining global synchronization via 3GPP TSaaS (Release 18) [17], and utilizing managed real-time Cloud slices are currently realistic only within private, operator-managed networks. In large-scale, public deployments today, these remain future architectural requirements.

This reliance explicitly defines the boundaries of the scheduler's capabilities. The HGA treats deterministic communication as an infrastructure-provided service exposed via the Deterministic Communication Interface (DCI). Consequently, if the DCI assumptions are violated—for example, because routing delays become unbounded or synchronization error and clock drift exceed the tolerated bounds—the synthesized timing bounds can no longer be guaranteed.

To mitigate sensitivity to such timing uncertainties, the TTEFC architecture is designed with a fault-tolerance extension point (Section 3.4.3). Rather than relying on unpredictable runtime reactive scheduling, severe violations of synchronization or communication bounds could be handled through deterministic switching among precomputed schedules in Multi-Schedule Graphs (MSGs) [2]. Ultimately, the proposed framework successfully solves the offline scheduling problem but explicitly separates the synthesis of these temporal budget constraints from the physical infrastructure hardware required to enforce them.

7. Conclusions

This paper presented a hierarchical deterministic scheduling framework for managed Time-Triggered Edge–Fog–Cloud (TTEFC) architectures under bounded execution-time and communication-latency assumptions. The proposed approach integrates computation and communication decisions within a unified optimization model, combining partition allocation, execution ordering, deterministic routing, and time-budget assignment into a single co-design problem.

To address this NP-hard problem, a two-level hierarchical genetic algorithm (HGA) was introduced. At the system level, global scheduling decisions are optimized, while parallel partition-level schedulers construct locally feasible schedules under assigned time budgets. A key contribution of this work is the adaptive slack reallocation operator, which

enables dynamic redistribution of temporal slack across partitions, thereby improving convergence toward feasible solutions without relaxing the global deadline constraint.

Across the 10/10 launched and completed baseline runs, the proposed method found schedules satisfying the modeled global deadline and partition time-budget constraints. The HEFT and Round-Robin values are retained only as scoped external references and are not used for superiority claims. In the controlled ablation, adaptive reallocation improves budget alignment and feasibility reliability and reaches partition-budget feasibility earlier than HGA-NoFeedback in the settings where both variants reach that condition.

The sensitivity analysis further revealed that the proposed framework remains robust under moderate deadline tightening, successfully maintaining feasibility in several constrained scenarios. Even when the evaluated runs do not achieve full end-to-end feasibility under tighter deadlines, the scheduler converges to stable solutions with zero partition-budget violations and reduced global lateness. This behavior highlights the ability of the hierarchical approach to provide meaningful and predictable solutions under constrained conditions.

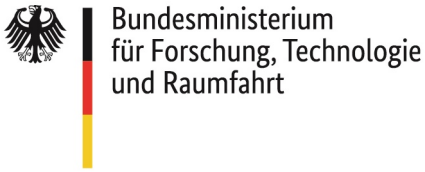
The results show that the proposed two-level HGA provides an effective integrated mechanism for coordinating partition allocation, execution ordering, bounded-latency path selection, and partition-level time budgets within the evaluated managed multi-tier model. Across the evaluated workload sizes, the complete scheduler produced schedules satisfying the modeled global deadline and partition time-budget constraints, while the controlled ablation demonstrated the contribution of adaptive time-budget reallocation to feasibility reliability, budget alignment, and convergence. The proposed framework therefore offers a systematic offline scheduling approach for managed distributed systems with stringent timing requirements.

Future work will extend the proposed framework to support runtime adaptation and fault-aware scheduling by incorporating incremental metascheduling and multi-schedule graph (MSG) techniques. Additional directions include energy-aware optimization, support for heterogeneous accelerators, and validation on real TSN/DetNet platforms to strengthen the practical relevance of the approach for cyber-physical and software-defined vehicle systems.

Author Contributions: Conceptualization, O.H., J.P. and R.O.; methodology, O.H.; software, O.H.; validation, O.H., J.P. and R.O.; formal analysis, O.H.; investigation, O.H.; resources, O.H. and J.P.; data curation, O.H. and J.P.; writing—original draft preparation, O.H.; writing—review and editing, O.H., D.O. and R.O.; visualization, J.P.; supervision, D.O. and R.O.; project administration, R.O.; funding acquisition, R.O. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the EdgeAI-Trust project, which is funded within the Chips Joint Undertaking (Chips JU), a public–private partnership in collaboration with the European Union's Horizon Europe research and innovation program and National Authorities under grant agreement number 101139892.

Gefördert durch:



Data Availability Statement: The scheduler code, synthetic SNAP-generated DAG workloads, platform and partition input files, dependency specifications, and execution instructions are openly available in Zenodo at https://doi.org/10.5281/zenodo.21622701 (accessed on 12 August 2026).

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Fernando, N.; Shrestha, S.; Loke, S.W.; Lee, K. On Edge–Fog–Cloud Collaboration and Reaping Its Benefits: A Heterogeneous Multi-Tier Edge Computing Architecture. *Future Internet* **2025**, *17*, 22. [CrossRef]
2. Hekal, O.; Onwuchekwa, D.; Obermaisser, R. Incremental Multi-Schedule Graph for Memory Optimization in Adaptive Time-Triggered Systems. In Proceedings of the 2025 14th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 10–14 June 2025; pp. 1–8. [CrossRef]
3. Alsadie, D. Advancements in Heuristic Task Scheduling for IoT Applications in Fog-Cloud Computing: Challenges and Prospects. *PeerJ Comput. Sci.* **2024**, *10*, e2128. [CrossRef] [PubMed]
4. Zhang, P.; Chen, N.; Xu, G.; Kumar, N.; Barnawi, A.; Guizani, M.; Duan, Y.; Yu, K. Multi-Target-Aware Dynamic Resource Scheduling for Cloud-Fog-Edge Multi-Tier Computing Network. *IEEE Trans. Intell. Transp. Syst.* **2023**, *25*, 3885–3897. [CrossRef]
5. Dechouniotis, D.; Athanasopoulos, N.; Leivadeas, A.; Mitton, N.; Jungers, R.; Papavassiliou, S. Edge Computing Resource Allocation for Dynamic Networks: The DRUID-NET Vision and Perspective. *Sensors* **2020**, *20*, 2191. [CrossRef] [PubMed]
6. Oukissou, Y.; Elhaou, H.; Ait Omar, D.; Zougagh, H.; Elouaham, S. Optimized Task Scheduling in Fog-Cloud Environments Using a Cost-Aware Genetic Algorithm. *J. Theor. Appl. Inf. Technol.* **2025**, *103*, 4844–4858. Available online: https://www.jatit.org/volumes/Vol103No11/24Vol103No11.pdf (accessed on 12 August 2026).
7. Lala, J.; Harper, R. Architectural principles for safety-critical real-time applications. *Proc. IEEE* **1994**, *82*, 25–40. [CrossRef]
8. Kopetz, H.; Bauer, G. The Time-Triggered Architecture. *Proc. IEEE* **2003**, *91*, 112–126. [CrossRef]
9. Wang, S. Edge Computing: Applications, State-of-the-Art and Challenges. *Adv. Netw.* **2019**, *7*, 8. [CrossRef]
10. Gala, G.; Castillo Rivera, J.; Fohler, G. Work-in-Progress: Cloud Computing for Time-Triggered Safety-Critical Systems. In Proceedings of the 2021 IEEE Real-Time Systems Symposium (RTSS), Dortmund, Germany, 7–10 December 2021; pp. 516–519. [CrossRef]
11. Dizdarević, J.; Carpio, F.; Jukan, A.; Masip-Bruin, X. A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration. *ACM Comput. Surv.* **2019**, *51*, 116. [CrossRef]
12. Obermaisser, R. *Event-Triggered and Time-Triggered Control Paradigms*; Real-Time Systems Series; Springer: New York, NY, USA, 2005; Volume 22, pp. 1–153. [CrossRef]
13. Paulachan, J.; Hekal, O.; Onwuchekwa, D.; Obermaisser, R. A Time-Triggered Edge–Fog–Cloud Architecture with Hierarchical Genetic Optimization for Safety-Critical Applications. *IEEE Access* **2025**, *13*, 217406–217422. [CrossRef]
14. Ahmed, Z.E.; Hashim, A.A.; Saeed, R.A.; Saeed, M.M. Deterministic Networking (DetNet): Architecture, Protocols, and Advanced Applications. In Proceedings of the 2024 9th International Conference on Mechatronics Engineering (ICOM), Kuala Lumpur, Malaysia, 13–14 August 2024; pp. 213–218. [CrossRef]
15. Zhang, T.; Wang, G.; Xue, C.; Wang, J.; Nixon, M.; Han, S. Time-Sensitive Networking (TSN) for Industrial Automation: Current Advances and Future Directions. *ACM Comput. Surv.* **2025**, *57*, 30. [CrossRef]
16. Finn, N.; Thubert, P.; Varga, B.; Farkas, J. Deterministic Networking Architecture. In *RFC 8655*; Internet Engineering Task Force: Fremont, CA, USA, 2019. Available online: https://datatracker.ietf.org/doc/rfc8655/ (accessed on 12 August 2026).
17. 3rd Generation Partnership Project. System Architecture for the 5G System (5GS); Stage 2. Technical Specification 3GPP TS 23.501, Version 18.0.0, Release 18, 3rd Generation Partnership Project. 2024. Available online: https://www.3gpp.org/dynareport/23501.htm (accessed on 27 July 2026).
18. Avan, A.; Azim, A.; Mahmoud, Q.H. A State-of-the-Art Review of Task Scheduling for Edge Computing: A Delay-Sensitive Application Perspective. *Electronics* **2023**, *12*, 2599. [CrossRef]
19. Topcuoglu, H.; Hariri, S.; Wu, M.Y. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *IEEE Trans. Parallel Distrib. Syst.* **2002**, *13*, 260–274. [CrossRef]
20. Cao, H.; Wachowicz, M. An Edge-Fog-Cloud Architecture of Streaming Analytics for Internet of Things Applications. *Sensors* **2019**, *19*, 3594. [CrossRef] [PubMed]
21. Prokhorenko, V.; Babar, M.A. Architectural Resilience in Cloud, Fog and Edge Systems: A Survey. *IEEE Access* **2020**, *8*, 28078–28095. [CrossRef]
22. Alli, A.A.; Alam, M.M. The Fog Cloud of Things: A Survey on Concepts, Architecture, Standards, Tools, and Applications. *Internet Things* **2020**, *9*, 100177. [CrossRef]

23. OpenFog Consortium Architecture Working Group. *OpenFog Reference Architecture for Fog Computing*; Technical Report OPFRA001.020817; OpenFog Consortium. 2017. Available online: https://inf.mit.bme.hu/sites/default/files/materials/category/kateg%C3%B3ria/oktat%C3%A1s/msc-t%C3%A1rgyak/kiberfizikai-rendszerek/17/07_OpenFog.pdf (accessed on 12 August 2026).
24. ETSI. *Mobile Edge Computing (MEC)*; Technical Requirements. ETSI Group Specification ETSI GS MEC 002 V1.1.1; European Telecommunications Standards Institute. 2016. Available online: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/002/01.01.01_60/gs_MEC002v010101p.pdf (accessed on 27 July 2026).
25. Wang, Y.; Liu, D.; Wang, M.; Qin, Z.; Shao, Z. Optimal Task Scheduling by Removing Inter-Core Communication Overhead for Streaming Applications on MPSoC. In Proceedings of the 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Stockholm, Sweden, 12–15 April 2010; pp. 195–204. [CrossRef]
26. Wang, Y.; Liu, D.; Qin, Z.; Shao, Z. Memory-Aware Optimal Scheduling with Communication Overhead Minimization for Streaming Applications on Chip Multiprocessors. In Proceedings of the 2010 IEEE 31st Real-Time Systems Symposium (RTSS), San Diego, CA, USA, 30 November 2010–3 December 2010; pp. 350–359. [CrossRef]
27. Zhang, J.; Zhu, F.; Yang, Z.; Chen, C.; Tian, X.; Guan, X. Routing and scheduling co-design for holistic software-defined deterministic network. *Fundam. Res.* **2024**, *4*, 25–34. [CrossRef] [PubMed]
28. Nie, H.; Wu, Y.; Zhu, W.; Zhong, J.; Yang, H.; Zhou, Y. Time-Triggered Task Offloading Scheduling in TSN-Based Edge Computing Power Networks. *IEEE Access* **2025**, *13*, 85979–85996. [CrossRef]
29. Al-Arasi, R.; Saif, A. Task Scheduling in Cloud Computing Based on Metaheuristic Techniques: A Review Paper. *EAI Endorsed Trans. Cloud Syst.* **2018**, *6*, 162829. [CrossRef]
30. Kakkottakath Valappil Thekkepuryil, J.; Suseelan, D.P.; Keerikkattil, P.M. An Effective Meta-Heuristic Based Multi-Objective Hybrid Optimization Method for Workflow Scheduling in Cloud Computing Environment. *Clust. Comput.* **2021**, *24*, 2367–2384. [CrossRef]
31. Aktan, M.N.; Bulut, H. Metaheuristic Task Scheduling Algorithms for Cloud Computing Environments. *Concurr. Comput. Pract. Exp.* **2022**, *34*, e6513. [CrossRef]
32. Samimi, N.; Abeni, L.; Casini, D.; Marinoni, M.; Basten, T.; Nasri, M.; Geilen, M.; Biondi, A. Enabling Containerisation of Distributed Applications with Real-Time Constraints. In *Proceedings of the 37th Euromicro Conference on Real-Time Systems (ECRTS 2025)*; Mancuso, R., Ed.; Leibniz International Proceedings in Informatics (LIPIcs): Dagstuhl, Germany, 2025; Volume 335, pp. 1–29. [CrossRef]
33. Struhár, V.; Craciunas, S.S.; Ashjaei, M.; Behnam, M.; Papadopoulos, A.V. Hierarchical Resource Orchestration Framework for Real-Time Containers. *ACM Trans. Embed. Comput. Syst.* **2024**, *23*, 4. [CrossRef]
34. Casini, D.; Abeni, L.; Marinoni, M.; Biondi, A. Runtime Monitoring for Edge Applications. In Proceedings of the 2025 21st International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT), Lucca, Italy, 9–11 June 2025; pp. 405–412. [CrossRef]
35. Gavrilut, V.M. Design Optimization of IEEE Time-Sensitive Networks (TSN) for Safety-Critical and Real-Time Applications. Ph.D. Thesis, Technical University of Denmark, Kongens Lyngby, Denmark, 2018.
36. *802.1Qbv-2015*; IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 25: Enhancements for Scheduled Traffic. IEEE: New York, NY, USA, 2016. [CrossRef]
37. Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197. [CrossRef]
38. Goldberg, D.E.; Robert, L. Alleles, Loci, and the Traveling Salesman Problem. In *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*; Grefenstette, J.J., Ed.; Lawrence Erlbaum: Pittsburgh, PA, USA, 1985; pp. 154–159.
39. De Rainville, F.M.; Fortin, F.A.; Gardner, M.A.; Parizeau, M.; Gagné, C. DEAP: A Python Framework for Evolutionary Algorithms. In Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation, New York, NY, USA, 7–11 July 2012; GECCO '12; pp. 85–92. [CrossRef]
40. Dick, R.P.; Rhodes, D.L.; Wolf, W. TGFF: Task Graphs for Free. In Proceedings of the Sixth International Workshop on Hardware/Software Codesign (CODES/CASHE'98), Seattle, WA, USA, 15–18 March 1998; pp. 97–101. [CrossRef]
41. Stuijk, S.; Geilen, M.; Basten, T. SDF3: SDF for Free. In Proceedings of the Sixth International Conference on Application of Concurrency to System Design (ACSD'06), Turku, Finland, 28–30 June 2006; pp. 276–278. [CrossRef]
42. Leskovec, J.; Sosič, R. SNAP: A General-Purpose Network Analysis and Graph-Mining Library. *ACM Trans. Intell. Syst. Technol.* **2016**, *8*, 1–20. [CrossRef] [PubMed]
43. Yen, J.Y. Finding the *K* Shortest Loopless Paths in a Network. *Manag. Sci.* **1971**, *17*, 712–716. [CrossRef]
44. Derrac, J.; García, S.; Molina, D.; Herrera, F. A Practical Tutorial on the Use of Nonparametric Statistical Tests as a Methodology for Comparing Evolutionary and Swarm Intelligence Algorithms. *Swarm Evol. Comput.* **2011**, *1*, 3–18. [CrossRef]
45. McNemar, Q. Note on the Sampling Error of the Difference between Correlated Proportions or Percentages. *Psychometrika* **1947**, *12*, 153–157. [CrossRef] [PubMed]

46. Holm, S. A Simple Sequentially Rejective Multiple Test Procedure. *Scand. J. Stat.* **1979**, *6*, 65–70. [CrossRef]
47. Kramer, S.; Ziegenbein, D.; Hamann, A. Real World Automotive Benchmarks for Free. In Proceedings of the 6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems (WATERS 2015), Lund, Sweden, 7 July 2015. Available online: https://rtn.ecrts.org/forum/download/WATERS15_Real_World_Automotive_Benchmark_For_Free.pdf (accessed on 27 July 2026).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.