mathematics

MDPI

Article

Real-Time Task Scheduling and Resource Planning for IIoT-Based Flexible Manufacturing with Human–Machine Interaction

Gahyeon Kwon [1], Yeongeun Shim [1], Kyungwoon Cho [2] and Hyokyung Bahn [1,*]

[1] Department of Computer Engineering, Ewha University, Seoul 03760, Republic of Korea; sonaby2@ewhain.net (G.K.); shim161030@ewhain.net (Y.S.)
[2] Embedded Software Research Center, Ewha University, Seoul 03760, Republic of Korea; cezanne@ewha.ac.kr
* Correspondence: bahn@ewha.ac.kr

Abstract: The emergence of Flexible Manufacturing Systems (FMS) presents new challenges in Industrial IoT (IIoT) environments. Unlike traditional real-time systems, FMS must accommodate task set variability driven by human–machine interaction. As such variations can lead to abrupt resource overload or idleness, a dynamic scheduling mechanism is required. Although prior studies have explored dynamic scheduling, they often relax deadlines for lower-criticality tasks, which is not well suited to IIoT systems with strict deadline constraints. In this paper, instead of treating dynamic scheduling as a prediction problem, we model it as deterministic planning in response to explicit, observable user input. To this end, we precompute feasible resource plans for anticipated task set variations through offline optimization and switch to the appropriate plan at runtime. During this process, our approach jointly optimizes processor speeds, memory allocations, and edge/cloud offloading decisions, which are mutually interdependent. Simulation results show that the proposed framework achieves up to 73.1% energy savings compared to a baseline system, 100% deadline compliance for real-time production tasks, and low-latency responsiveness for user-interaction tasks. We anticipate that the proposed framework will contribute to the design of efficient, adaptive, and sustainable manufacturing systems.

Keywords: real-time scheduling; human–machine interaction; flexible manufacturing system; edge computing; task offloading; energy-efficient computing; DVFS; evolutionary computation; industrial IoT; energy efficiency

MSC: 68U01

check for updates

Academic Editor: Zhi Pei, Zhihai Zhang, Jian Chen and Chaobo Yan

Received: 23 April 2025
Revised: 27 May 2025
Accepted: 29 May 2025
Published: 31 May 2025

Citation: Kwon, G.; Shim, Y.; Cho, K.; Bahn, H. Real-Time Task Scheduling and Resource Planning for IIoT-Based Flexible Manufacturing with Human–Machine Interaction. *Mathematics* **2025**, *13*, 1842. https://doi.org/10.3390/math13111842

Copyright: © 2025 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 (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The advent of Industrial IoT (IIoT) and Flexible Manufacturing Systems (FMS) has significantly improved flexibility and efficiency in modern manufacturing [1,2]. IIoT enables real-time monitoring and data analysis in production environments, and FMS uses this information to dynamically reconfigure manufacturing processes without system interruption [3,4]. In such environments, Human–machine Interaction (HMI) bridges the gap between human operators and machines, allowing seamless integration of user input into automated systems [5,6]. This interaction further enhances the system's ability to adapt quickly and flexibly, which is essential for achieving intelligent and sustainable manufacturing [7].

Despite these advancements, IIoT-based FMS encounters resource management challenges that are not typically addressed in traditional real-time systems. First, flexible manufacturing systems must handle both predefined production tasks and dynamically

arriving user-interaction tasks. Production tasks are hard real-time tasks that require offline schedulability analysis to ensure deadline compliance [8]. In contrast, user-interaction tasks, though latency-sensitive, are not bound by deadlines and require online scheduling due to their unpredictability [9]. Integrating these heterogeneous task types into a single system introduces complex scheduling challenges, as the resource management philosophies for real-time and non-real-time tasks differ significantly.

Second, the variability of production tasks presents another significant challenge. Unlike traditional real-time systems that assume a static task set, FMS must adapt to sudden surges or modifications in production tasks. This requires immediate rescheduling under tight resource constraints, posing challenges beyond those of traditional scheduling problems. When local resources are insufficient, offloading real-time tasks to edge/cloud servers becomes not just beneficial but imperative for maintaining system stability and deadline satisfaction [10–12]. Such dynamic changes also complicate resource planning, as resource configurations such as processor speed, memory allocation, and offloading ratio are mutually interdependent.

Although prior studies have explored dynamic task scheduling, they address uncertainty in task set variability by relaxing deadlines for some tasks, which may not be suitable for IIoT-based manufacturing systems, where strict timing constraints are required. Specifically, they permit occasional deadline misses for less critical tasks [13,14] or adopt best-effort approaches that treat deadlines as soft goals and focus instead on reducing latency [15–17].

In this paper, we propose a resource planning framework for FMS that integrates production and user-interaction tasks while dynamically adapting to workload variations. The framework jointly optimizes resource allocation and scheduling decisions across local (processor, memory, and network) and remote resources, ensuring that production tasks meet their deadlines, user-interaction tasks maintain acceptable latency, and system-wide energy consumption is minimized. It consists of three key processes:

- **Offline resource planning**: Scheduling real-time production tasks by optimizing processor speed, memory allocation, and offloading decisions to minimize energy consumption while meeting deadline constraints. To accommodate user-interaction tasks, pre-allocated resource slots are reserved within the offline schedule.
- **Online scheduling for user-interaction tasks**: When user-interaction tasks arrive, they are assigned to reserved slots and scheduled online to ensure low latency.
- **Adaptation to task set changes**: If user-interaction tasks cause modifications to the production task set, the offline resource plan is updated, and slots for user-interaction tasks are adjusted to maintain low latency.

To prepare for task set changes, the proposed framework analyzes potential variations in advance and optimizes corresponding offline resource plans, which are retained for immediate application when changes occur. This approach represents a significant advancement in enabling adaptive, sustainable, and efficient operations within IIoT-based FMS. Simulation results across diverse task sets demonstrate that the proposed framework achieves an average energy reduction of 73.1% compared to a baseline system, while ensuring zero deadline misses for production tasks and maintaining acceptable low latency for user-interaction tasks, even under significant workload variations.

The main contributions of this paper can be summarized as follows:

- **A unified scheduling framework for heterogeneous task sets**
  We propose a unified scheduling framework that supports real-time production tasks and sporadic user-interaction tasks within a single system. Unlike previous models, it ensures strict deadline compliance even when task sets are dynamically modified through human interaction.

- **Handling task dynamics through explicit input rather than uncertain prediction**
  Unlike uncertainty-driven prediction models, our framework treats task set updates as explicit, bounded, and instantly known inputs, enabling deterministic rescheduling without compromising deadline guarantees.
- **Joint optimization of interdependent resources**
  The proposed method jointly optimizes processor speed (DVFS), memory mapping (HPM/LPM), and edge/cloud offloading, considering their mutual interdependencies to minimize energy while meeting performance goals.
- **Offline plan generation and instant runtime switching**
  Feasible resource plans are precomputed for anticipated task set combinations using evolutionary computation. Task slots for user-interaction tasks are also reserved, enabling immediate reconfiguration at runtime without schedulability violations.

The remainder of this paper is structured as follows. Section 2 presents the problem definition and task execution modeling. Section 3 details the resource planning optimization based on evolutionary computation techniques. Section 4 evaluates the proposed scheme through simulation experiments. Section 5 provides a brief overview of related work. Finally, Section 6 concludes the paper.

2. Problem Definition and Modeling

Scheduling problems fundamentally involve two decision components: (1) how to assign resources and (2) how to determine the execution order of tasks. Classical scheduling approaches typically focus on ordering tasks under fixed resource conditions. In real-time systems, this is done with static task sets to ensure deadline guarantees. Non-real-time systems, on the other hand, tolerate more variability and pursue best-effort strategies to reduce latency.

With the rise of edge/cloud computing, resource provisioning decisions have become just as crucial as scheduling policies, especially under dynamic load conditions. For non-real-time workloads, adaptive provisioning—such as scale-up or scale-out—is possible by monitoring traffic and reacting accordingly. In real-time systems, similar adaptation is now increasingly required. Specifically, uncertainty in real-time task variability is usually handled by relaxing deadlines for some lower-criticality tasks [13,14]. While this strategy is effective in soft real-time systems, it may not be suitable for IIoT-based manufacturing environments, where strict deadline compliance is essential for all production tasks.

Such task set changes, however, are not arbitrary or unpredictable. They are explicitly triggered by human interaction, bounded in scope, structurally predictable, and promptly visible to the system at runtime. Therefore, this is not a prediction problem, but rather a case of deterministic rescheduling in immediate response to explicit, observable events.

2.1. Problem Definition

In traditional manufacturing systems, once a production task set is configured, it is repeatedly executed under a fixed resource configuration. Modifying the task set typically requires the system to be stopped, reconfigured offline, and then restarted. However, in modern flexible manufacturing systems equipped with HMI, human operators can make changes to the production task set at runtime without interrupting system operation. Such human-triggered modifications include (1) inserting new tasks such as quality checks or configuration overrides, (2) modifying or canceling ongoing production tasks, and (3) triggering emergency interventions or inspection-related procedures [18,19]. These events are modeled in our system as user-interaction tasks, which are not bound by hard deadlines, but are latency-sensitive and must be scheduled online due to their unpredictable nature. It is important to clarify that user-interaction tasks do not act as a managerial layer

for production tasks, but rather as externally triggered inputs that may modify them at runtime. Integrating these two types of tasks within a unified system presents a challenge, as real-time tasks require deterministic resource planning, whereas user-interaction tasks demand flexibility. If separate resources are statically reserved for each task type, resource utilization becomes inefficient.

Traditionally, real-time tasks are executed on real-time operating systems (RTOS) to meet strict timing requirements, while non-real-time tasks, such as user-interaction tasks, are executed on general-purpose operating systems (GPOS), which employ more flexible scheduling policies. The resource management philosophies of RTOS and GPOS differ significantly. RTOS uses pre-determined resource plans for deterministic execution, while GPOS relies on dynamic online resource management to accommodate spontaneous user-generated tasks. Despite these differences, FMS should integrate both task types into a unified scheduling framework to efficiently handle production tasks as well as user interaction tasks.

Beyond the heterogeneity of task types, another major challenge is the dynamic variability of production tasks. Unlike traditional real-time systems that assume a static task set, FMS should adapt to sudden surges or modifications in production tasks. Such abrupt workload surges can lead to resource exhaustion, making it nearly impossible to guarantee real-time task deadlines. Moreover, incorporating these changes into the schedule at runtime is computationally intractable. Fortunately, the availability of edge/cloud servers allows for task offloading, enabling deadline guarantees even when local resources are exhausted.

However, dynamic offloading introduces additional complexity, as scheduling decisions must be made instantly, and execution-time uncertainty may arise due to the variable availability of remote resources [20]. Moreover, offloading affects local resource configurations, such as processor speed and memory allocation for non-offloaded tasks. Therefore, comprehensive consideration must be given to task execution timing, resource usage characteristics, data volume, and network conditions. Specifically, tasks suitable for remote execution should be identified, and their worst-case execution times should be modeled based on execution paths and the conditions of remote resources.

To handle such complexity, this study adopts evolutionary computation techniques for scheduling production tasks, while reserving pre-allocated slots for user-interaction tasks to enable timely execution. The optimization process involves processor speed scaling, memory allocation, and offloading decisions. The processor operates under dynamic voltage and frequency scaling (DVFS), allowing it to adjust its clock speed and voltage levels based on computational demand [21,22]. When the workload is light, DVFS reduces energy consumption by lowering the processor's speed; during peak demand, it restores full performance prior to offloading decisions. Similarly, memory is managed using a hybrid architecture, where low-power memory is used for energy-efficient storage, and high-performance DRAM is allocated to frequently accessed tasks [23,24].

As user-interaction behavior and production tasks change over time, the optimization process should be updated accordingly. However, recomputing the resource plan at every task set modification can introduce delays that critically affect real-time performance. To address this, the proposed framework precomputes resource plans for all plausible production task set configurations and maintains a repository of these plans for immediate deployment. This approach is based on the observation that production tasks do not change arbitrarily, but follow a limited set of possible variations that can be identified in advance.

Accordingly, in this study, production tasks are categorized into essential production tasks and temporary production tasks, and resource plans are precomputed for scenarios both with and without temporary tasks. When workload fluctuations occur due to the

addition or removal of temporary tasks, the system immediately switches to the corresponding precomputed plan, ensuring efficient resource utilization with minimal scheduling delay. Figure 1 illustrates the overall workflow of the proposed resource planning and task execution framework.

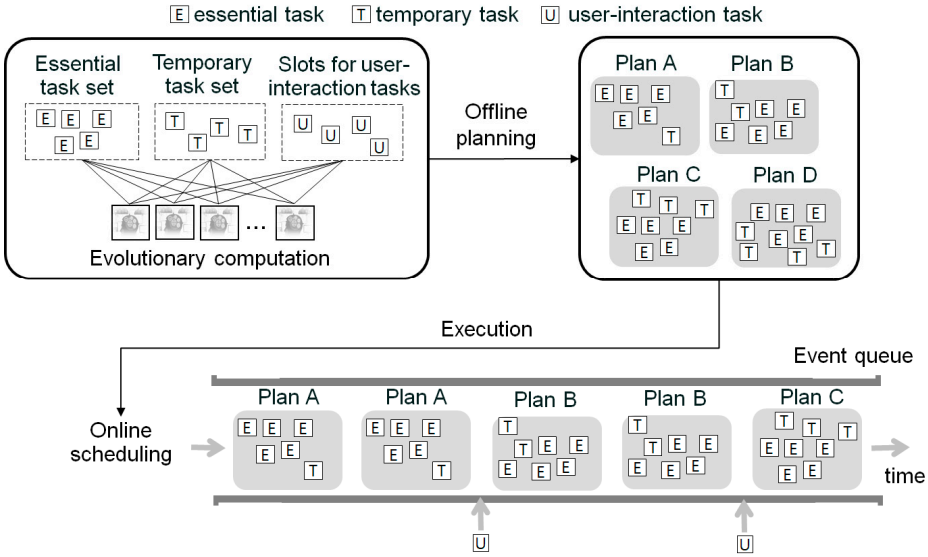


Figure 1. Overview of the proposed task execution workflow with offline and online resource planning. Offline resource plans are precomputed using evolutionary computations, while online adaptation occurs upon the arrival of user-interaction tasks during system execution.

2.2. Task Execution Model

The term "task" is often used interchangeably to refer to both physical manufacturing operations at the OT (Operational Technology) level and real-time computational tasks executed on processors at the IT (Information Technology) level. This ambiguity has grown with the rise of cyber-physical systems, where real-time computational tasks increasingly function as cyber-counterparts of physical actions. In our model, tasks refer specifically to IT-level computational units. For instance, the movement of a manufacturing robot arm is an OT task, while the calculations required to control that motion constitute the IT task targeted by our framework. These computational tasks must be scheduled over resources such as CPU cycles, memory, and network bandwidth, and can be managed using established scheduling theory and resource planning techniques.

Our task execution model extends the traditional real-time task model by incorporating optimizations such as processor speed control, low-power memory allocation, and offloading to edge/cloud servers [25]. The task set is defined as $T = \{t_1, t_2, \ldots, t_n\}$, where each task $t_i \in T$ belongs to one of the following categories: essential production tasks, temporary production tasks, or task slots for user-interaction tasks. A task slot is not an actual task, but a reserved time slot allocated for online scheduling of user-interaction tasks.

The target system consists of a processor with DVFS capabilities and a hybrid memory system comprising low-power memory (LPM) and high-performance memory (HPM) in the form of DRAM. Additionally, edge/cloud servers are available for executing offloaded tasks. Each task can either be executed locally or offloaded, leading to a division of the task set T into two subsets: ON_DEVICE and OFFLOADED. While offloading can improve performance by leveraging the higher computational power of remote servers, it should be carefully managed since offloaded tasks should complete execution and return results within their deadlines. Certain control tasks that interact with sensors or actuators should always be executed on the local processor due to predefined system constraints.

Since tasks can be stored in two different memory types, the task set T is further categorized into HPM and LPM. A task t_i is represented as Equation (1).

$$t_i =< WCET_i, Period_i, Data_i > \tag{1}$$

where $WCET_i$ is the worst-case execution time when the task executes at the default clock speed of the local processor, $Period_i$ is its execution period, and $Data_i$ is defined as Equation (2).

$$Data_i =< Size_i, Input_i, Output_i, Rd_i, Wr_i > \tag{2}$$

where $Size_i$ is the memory footprint, $Input_i$ and $Output_i$ represent the input/output data, and Rd_i and Wr_i denote the number of read and write operations in memory. For tasks $t_i \in$ OFFLOADED, $Input_i$ should be transmitted to the remote server before execution, and $Output_i$ should be received from the server after execution. This study focuses on periodic tasks, where each real-time task's deadline is defined by $Period_i$. The global scheduling period G_Period_T for all tasks $t_i \in T$ is the least common multiple of individual task periods.

When tasks execute locally without offloading, the schedulability condition of Equation (3) must be met to ensure all deadlines are satisfied at maximum processor speed:

$$Utilization = \sum_{t_i \in T} \frac{WCET_i}{Period_i} \leq 1 \tag{3}$$

If this condition holds, preemptive scheduling using the Earliest-Deadline-First (EDF) algorithm is feasible [26,27]. However, since we use DVFS, the worst-case execution time should be adjusted for different processor speeds, leading to Equation (4).

$$Utilization = \sum_{t_i \in T} \frac{DVFS(WCET_i)}{Period_i} \leq 1 \tag{4}$$

where $DVFS(WCET_i)$ represents the worst-case execution time of t_i under DVFS. Since tasks residing in HPM and LPM have different memory access times, the processor utilization must be computed separately for each memory type, as formulated in Equation (5).

$$Utilization = \sum_{t_i \in HPM} \frac{DVFS_{HPM}(WCET_i)}{Period_i} + \sum_{t_i \in LPM} \frac{DVFS_{LPM}(WCET_i)}{Period_i} \leq 1 \tag{5}$$

The terms $DVFS_{HPM}(WCET_i)$ and $DVFS_{LPM}(WCET_i)$ are calculated as Equations (6) and (7), respectively.

$$DVFS_{HPM}(WCET_i) = DVFS(WCET_i) \tag{6}$$

$$DVFS_{LPM}(WCET_i) = DVFS(WCET_i) + \Delta_{Rd} \cdot Rd_i + \Delta_{Wr} \cdot Wr_i \tag{7}$$

where Δ_{Rd} and Δ_{Wr} represent the additional memory read/write delays between HPM and LPM. Since offloading is also supported, the processor utilization constraint of Equation (8) must be met:

$$\begin{aligned} Utilization = & \sum_{t_i \in ON_DEVICE \cap HPM} \frac{DVFS_{HPM}(WCET_i)}{Period_i} + \\ & \sum_{t_i \in ON_DEVICE \cap LPM} \frac{DVFS_{LPM}(WCET_i)}{Period_i} + \sum_{t_i \in OFFLOADED} \frac{Comnd_i}{G_Period_T} \leq 1 \end{aligned} \tag{8}$$

where $Comnd_i$ represents the time required to issue an offloading command to the network module. Once execution locations and processor speeds are determined, the local processor

initiates offloading commands at the start of G_Period_T, maximizing parallel execution between local and remote tasks.

To ensure that the offloaded tasks meet their deadlines, Equation (9) must be satisfied:

$$\forall t_i \in \text{OFFLOADED},\ Trnarnd_i < Period_i \tag{9}$$

where $Trnarnd_i$ is computed as Equation (10).

$$Trnarnd_i = Up_i + Remote(WCET_i) + Dn_i \tag{10}$$

Here, $Remote(WCET_i)$ represents the worst-case execution time of the task t_i on the remote server, which depends on the difference in clock frequencies between the local and remote processors. The upload and download times are defined as Equations (11) and (12), respectively.

$$Up_i = \begin{cases} \frac{Size_i}{BW_{up}} & \text{if first execution} \\ \frac{Input_i}{BW_{up}} & \text{otherwise} \end{cases} \tag{11}$$

$$Dn_i = \frac{Output_i}{BW_{dn}} \tag{12}$$

where BW_{up} and BW_{dn} represent the uplink and downlink bandwidths, respectively.

The aforementioned model assumes a single-core processor in the local system. For multi-core processors, the right-hand side of Equations (3)–(5) and (8) can be adjusted by replacing 1 with the number of processor cores. In this case, multi-core scheduling can be performed using a P-Fair class of algorithms instead of EDF [28].

2.3. Energy Model

The objective of this study is to minimize energy consumption while ensuring performance requirements for production and user-interaction tasks. In IIoT-based manufacturing, energy consumption arises at two distinct levels: the OT level, associated with physical equipment such as actuators, motors, and conveyors, and the IT level, related to computational resources such as CPUs, memory, and network modules. This study focuses exclusively on IT-level energy consumption, namely the energy used by processors (under DVFS), hybrid memory (HPM and LPM), and network modules during the execution of both local and offloaded tasks. The energy consumed by physical machinery at the OT level is not considered, as it falls outside the scope of this computational scheduling model.

The total system energy consumption during a global scheduling period G_Period_T is computed by summing the energy consumed by the processor, memory, and network components as follow.

The processor energy consumption is defined as Equation (13).

$$\begin{aligned} Energy_{\text{proceessor}} = \quad & c\ V_i^2 f_i \Bigg\{ \sum_{t_i \in OFFLOADED} Comnd_i + \\ & \sum_{t_i \in ON_DEVICE \cap HPM} DVFS_{HPM}(WCET_i) \cdot \frac{G_Period_T}{Period_i} + \\ & \sum_{t_i \in ON_DEVICE \cap LPM} DVFS_{LPM}(WCET_i) \cdot \frac{G_Period_T}{Period_i} \Bigg\} \end{aligned} \tag{13}$$

where c is the switching capacitance, V_i is the supply voltage, and f_i is the clock frequency.

The network energy consumption is calculated as Equation (14).

$$Energy_{\text{network}} = \sum_{t_i \in OFFLOADED} Net_Power(Up_i + Dn_i) \cdot \frac{G_Period_T}{Period_i} \tag{14}$$

where Net_Power is the power consumption of the network module, while Up_i and Dn_i are the uplink and downlink transmission times, respectively.

The memory energy consumption consists of dynamic and static energy [24,29], which are defined as Equations (15) and (16), respectively.

$$\begin{aligned} Energy_{\text{memory}_{dyn}} &= \sum_{t_i \in HPM} (Rd_i \cdot HPM_E_{Rd} + Wr_i \cdot HPM_E_{Wr}) \cdot \frac{G_Period_T}{Period_i} \\ &+ \sum_{t_i \in LPM} (Rd_i \cdot LPM_E_{Rd} + Wr_i \cdot LPM_E_{Wr}) \cdot \frac{G_Period_T}{Period_i} \end{aligned} \tag{15}$$

$$\begin{aligned} Energy_{\text{memory}_{stat}} &= \sum_{t_i \in HPM} (HPM_Power \cdot Size_i \cdot G_Period_T) \\ &+ \sum_{t_i \in LPM} (LPM_Power \cdot Size_i \cdot G_Period_T) \end{aligned} \tag{16}$$

where HPM_E_{Rd} and HPM_E_{Wr} represent the energy consumption per read and write operation in HPM, while LPM_E_{Rd} and LPM_E_{Wr} denote the corresponding values for LPM. HPM_Power and LPM_Power represent power consumptions per unit memory capacity for HPM and LPM, respectively.

3. Resource Planning Optimization Based on Evolutionary Computation

In real-time systems, schedulability testing should be conducted before execution to ensure that no deadlines are missed by appropriately planning system resources for the given task set. Following conventional real-time task scheduling practices, we primarily schedule production tasks offline. This means that execution configurations—including processor speed, memory allocation, and offloading decisions—are determined before the task set runs. However, unlike traditional real-time scheduling, where task sets remain static, FMS requires adaptive scheduling since task sets may change dynamically during execution. To accommodate these variations, we generate a set of resource plans by applying evolutionary computation to task set combinations and retain them for runtime use.

Additionally, for user-interaction tasks, task slots are pre-allocated within the offline schedule during the optimization process. When a user-interaction task arrives at runtime, it is dynamically scheduled using online scheduling within these reserved slots. We employ the Round-Robin algorithm for online scheduling, a widely used approach for interactive task scheduling in time-sharing systems [30]. Figure 2 illustrates the flowchart of the proposed method, outlining how resource planning and online adaptation are integrated. The following sections elaborate on the offline optimization process using evolutionary computation.

3.1. Problem Encoding

In our problem formulation, a solution should determine the resource planning configuration for all tasks in a given task set, including processor speed, memory allocation, and offloading decisions. To represent these solutions, our evolutionary computation model encodes each individual using three chromosomes, where each chromosome corresponds to one of the three optimization parameters: processor speed, memory location, and offloading decisions. The length of each chromosome is equal to the total number of tasks in the task set, as illustrated in Figure 3.

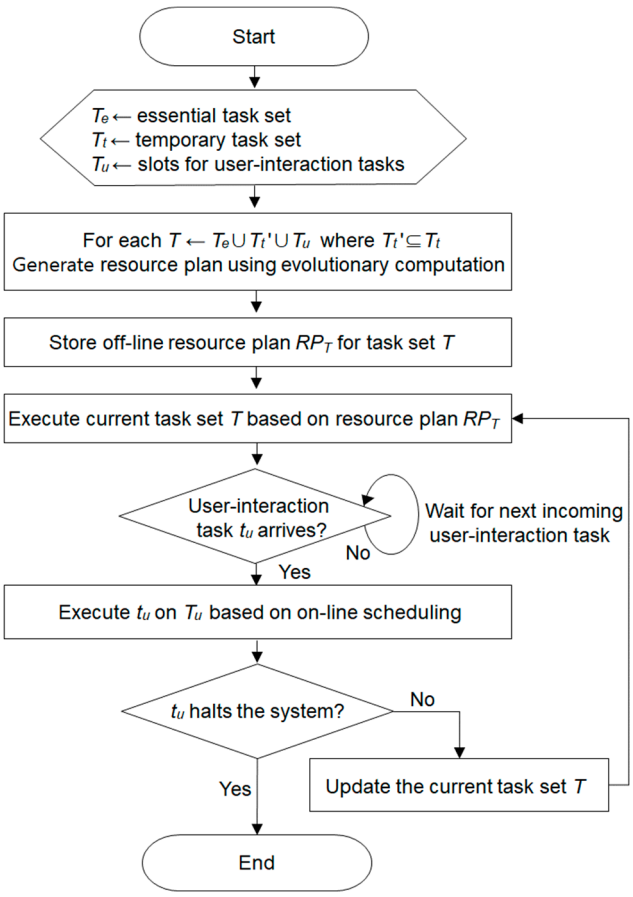


Figure 2. Flowchart of the proposed task scheduling and resource planning method.

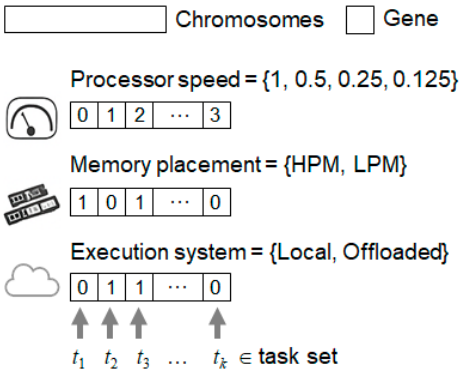


Figure 3. Chromosome encoding used in the evolutionary computation. Each of the three chromosomes encodes processor speed, memory placement, and execution location for the task set; each gene represents one task.

Although processor speed can theoretically be configured at various levels, real-world systems typically offer only a limited number of options. To reflect this, we restrict the processor speed levels to four discrete values: {1.0, 0.5, 0.25, 0.125}, where 1.0 represents the maximum processor speed. Similarly, genes in the memory location chromosome are encoded as {HPM, LPM}, and those in the offloading decision chromosome are encoded as {ON_DEVICE, OFFLOADED}. Based on this encoding, an initial population of 100 randomly generated individuals is created.

Evolutionary computation requires an objective function to assess the quality of each individual. In this study, we define a cost function, where individuals with lower cost values are considered superior. The cost function is determined by the total energy consumption when executing the task set using the resource configurations represented

by an individual's chromosomes. However, a low-cost solution may be infeasible if it fails to meet system constraints. For example, if processor utilization exceeds 1 or offloaded tasks miss their deadlines, the corresponding solution is not schedulable. To address this, a penalty function is added to the cost function, as shown in Equation (17).

$$Cost(i) = Energy_i + \ Penalty_i \tag{17}$$

where $Energy_i$ denotes the energy consumption of the task set scheduled by solution i, and $Penalty_i$ penalizes infeasible schedules. The penalty function is defined as Equation (18).

$$\begin{gathered} Penalty_i = \varepsilon \cdot \max\{0,\ Utilization_i - 1\} + \\ \gamma \cdot \sum_{t_j \in\ OFFLOADED} \max\{0,\ Trnarnd_j - Period_j\} \end{gathered} \tag{18}$$

where $Utilization_i$ represents the processor utilization under the resource plan represented by solution i, and ε and γ are weighing parameters that penalize processor overutilization and remote deadline violations, respectively.

3.2. Parent Selection

At each generation, parent selection is performed to choose two individuals from the population, which are then used to generate one or two offspring. Parent selection is typically conducted using a stochastic process that assigns a higher probability of selection to superior individuals, facilitating the evolution of the population.

In our approach, individual superiority is determined based on cost values. However, if selection probabilities are overly biased toward the best-performing individuals, the evolutionary process may prematurely converge to a local optimum, limiting diversity in the population. To mitigate this, we do not assign selection probabilities based purely on cost values. Instead, we rank individuals by their cost order and assign selection probabilities based on their rank. Specifically, we normalize the selection probabilities such that the best individual has four times the probability of selection compared to the worst individual (rank 100), a commonly used configuration in evolutionary computations [31].

3.3. Crossover and Mutation

The crossover operation generates an offspring by merging the chromosomes of two selected parents. We employ one-point crossover, where a random cut point is selected within a chromosome, and the offspring inherits genes from one parent before the cut point and from the other parent after it. Since each individual consists of three chromosomes representing processor speed, memory location, and offloading decisions, we independently select crossover points for each chromosome.

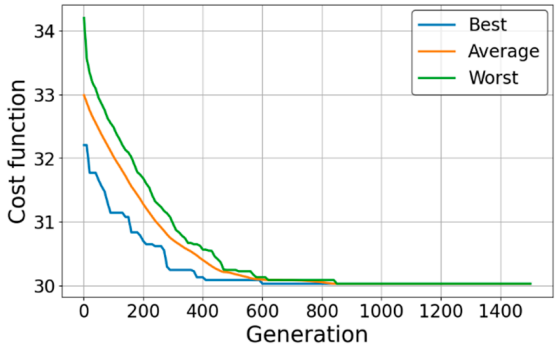
To explore a broader search space and prevent premature convergence to a local optimum, we apply mutation after crossover. Mutation alters a gene in the chromosome to preserve genetic diversity and prevent premature convergence. In our approach, mutation is performed by randomly selecting a gene position within a chromosome and replacing it with a randomly chosen valid value.

3.4. Replacement

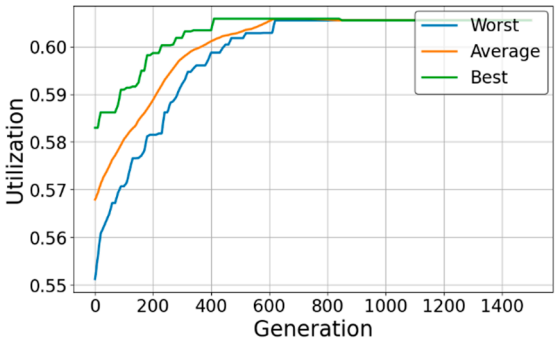
After generating an offspring through crossover and mutation, the replacement step determines how the new individual is integrated into the population. In our method, the weakest individual (i.e., the one with the highest cost) is eliminated, and the newly generated offspring is inserted into the population. This steady-state replacement strategy is widely used in evolutionary computations [32].

3.5. Convergence

Rather than fixing the number of generations, we continue the evolutionary process until population convergence is achieved. Specifically, our evolutionary computation monitors the cost values of the individuals in the population and terminates the process when all individuals' costs stabilize. Figure 4a illustrates the cost function trends of the population across generations, plotting the cost of the best individual, the worst individual, and the population average. As the figure shows, the cost function improves over time, and the population eventually converges. Importantly, the final converged population consists only of feasible schedules, ensuring that the optimization results satisfy both deadline constraints and energy efficiency. This is also confirmed by Figure 4b, where the utilization converges to a value less than 1.



(**a**) Cost function



(**b**) Utilization

Figure 4. Convergence of the evolutionary computation.

4. Performance Evaluation

To evaluate the effectiveness of the proposed resource planning framework, RP-FMS (Resource Planning for Flexible Manufacturing Systems), we conduct simulations under various workload conditions. For comparison, we also simulate Baseline, DVFS, and Offloading schemes. Baseline represents the original system configuration without DVFS or offloading. DVFS and Offloading independently optimize processor execution speed and offloading decisions, respectively, using evolutionary computation similar to RP-FMS. Thus, any improvements observed in our scheme over DVFS and Offloading stem from the refined task and resource modeling, rather than exhaustive evolutionary computation optimizations, ensuring a fair comparison.

In the experiments, both HPM and LPM have sufficient capacity to store the entire task set. Tasks allocated to LPM ensure that the corresponding HPM sections remain in low-power mode, thereby preventing static energy consumption. The access times for HPM and LPM are set to 50 ns and 100 ns for reads, and 50 ns and 350 ns for writes, respectively, following prior research [33]. The energy consumption values for HPM and LPM are

0.1 nJ/bit and 0.2 nJ/bit for reads, and 0.1 nJ/bit and 1.0 nJ/bit for writes, respectively. The static power consumption is assumed to be 1 W/GB for HPM and 0.1 W/GB for LPM.

The task set configurations follow those used in previous studies [33]. The task set consists of 100 tasks, with worst-case execution times (WCETs) randomly assigned between 500 ms and 1000 ms. Task periods are determined based on WCETs to match the desired load level. Task sizes range from 500 KB to 750 KB, input sizes from 100 KB to 500 KB, and output sizes from 100 KB to 250 KB. The network bandwidth is varied from 30 Mbps to 120 Mbps.

The load of the task set is varied from 0.05 to 0.9 in terms of processor utilization, where a load of 1.0 represents full saturation of local processing capacity.

4.1. Results for Static Task Sets

This section evaluates the effectiveness of RP-FMS under different task load conditions. Figure 5 presents the energy consumption of the four schemes as the task load varies from 0.05 to 0.9. Energy consumption values are normalized relative to the maximum energy consumption of Baseline at a load of 0.1. That is, Baseline's energy consumption at a load of 0.1 is set to 1.0, and all energy consumption values are scaled accordingly.

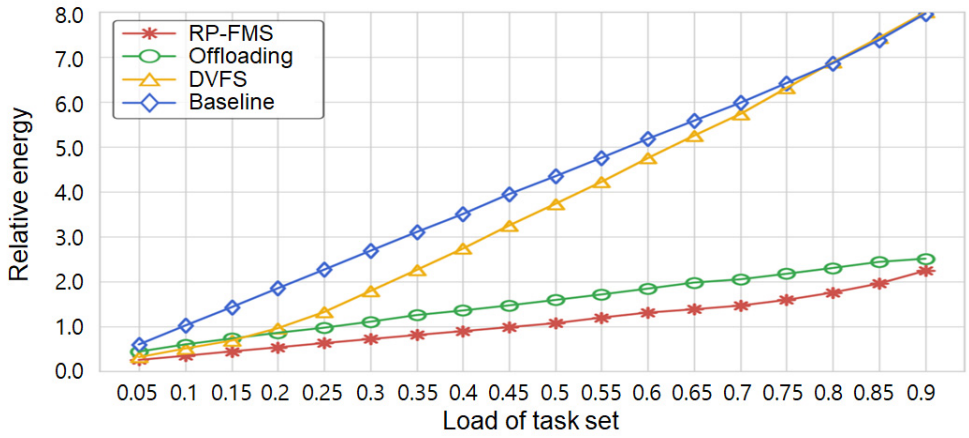
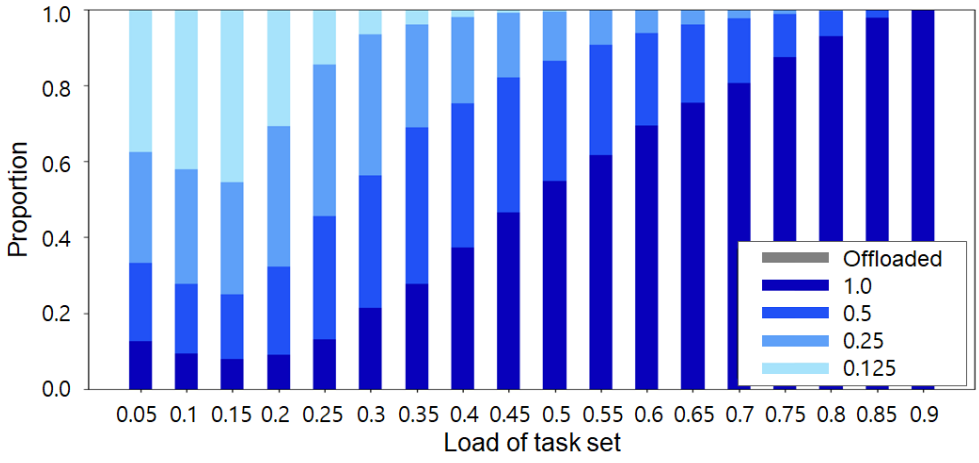


Figure 5. Comparison of energy consumption as the load of task set is varied.

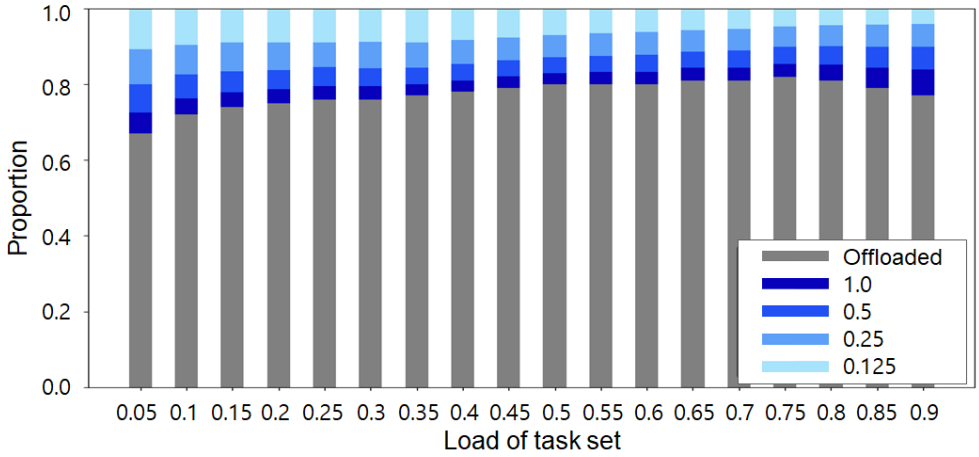
As shown in the figure, Baseline's energy consumption increases almost proportionally with the task load. DVFS and Offloading reduce energy consumption by nearly 50% in the 0.1 to 0.2 load range. However, DVFS becomes ineffective as the load increases, eventually converging to Baseline. This is because, under high-load conditions, the processor operates at full speed, leaving little room for energy savings through DVFS. In contrast, Offloading remains effective regardless of load conditions, as task execution can still be transferred to remote servers. Specifically, both DVFS and Offloading show competitive performance under light loads ranging from 0.05 to 0.2, but Offloading proves more effective at higher loads between 0.3 and 0.9, where it achieves 38.0% to 68.6% greater energy savings than DVFS. RP-FMS outperforms both DVFS and Offloading across all load conditions, as it co-optimizes processor speed adjustments and offloading decisions.

Specifically, RP-FMS achieves an average energy reduction of 61.1% compared to DVFS and 31.2% compared to Offloading.

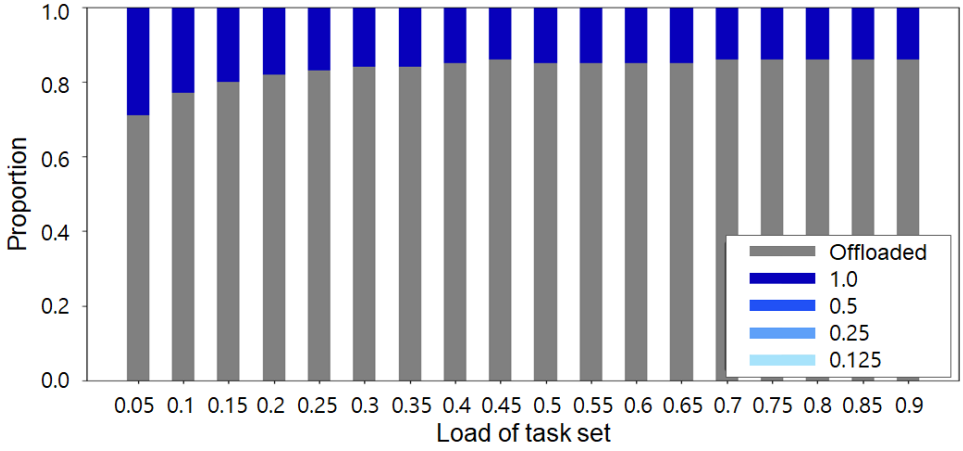
To further analyze these results, Figure 6 shows how the four schemes manage task execution in terms of processor speed and offloading decisions. DVFS efficiently utilizes all four processor speed levels when the task load is low. However, as the load increases, it predominantly sets the processor speed to 1.0, since fewer idle slots are available, making it difficult to lower processor speed.



(**a**) DVFS



(**b**) RP-FMS



(**c**) Offloading

Figure 6. Comparison of processor speed and offloading configurations as the load of the task set is varied.

In contrast, Figure 6b illustrates that RP-FMS maintains a balanced distribution of processor speed across all load levels by jointly optimizing processor speed and task placement through offloading. Specifically, RP-FMS offloads approximately 65–75% of tasks. The offloading ratio increases as the load rises, but at peak load levels, it slightly

decreases. This is because RP-FMS co-optimizes offloading and DVFS to identify the most energy-efficient configuration, rather than maximizing offloading indiscriminately.

On the other hand, Offloading in Figure 6c shows a continuous increase in the offloading ratio as the load grows. This occurs because, when the processor is operating at full speed, increasing offloading is always beneficial for energy savings.

Figure 7 shows the processor utilization of the four schemes as the task load varies. Utilization is a critical metric for evaluating how efficiently resources are managed. Low utilization indicates excessive idle time, suggesting opportunities for power-saving techniques such as reducing processor speed. However, when the task load is extremely low, even operating at the lowest processor speed may not be sufficient to achieve high utilization.

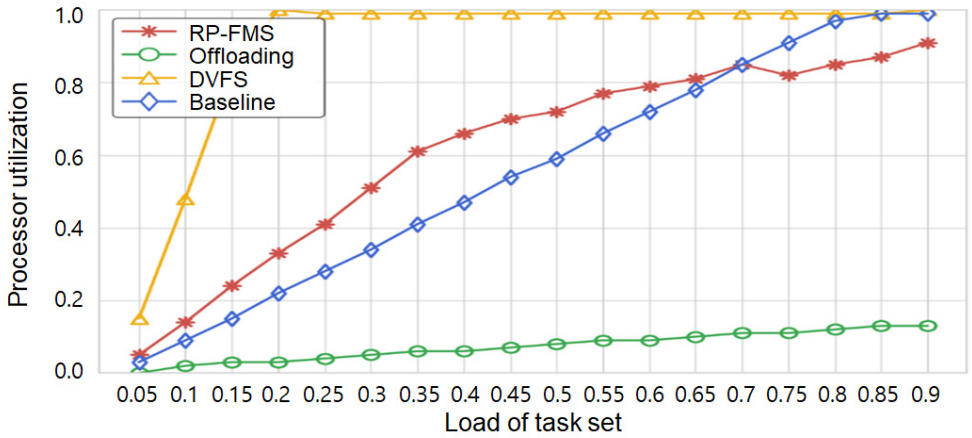


Figure 7. Comparison of processor utilization as the load of the task set is varied.

As shown in Figure 7, RP-FMS maintains higher utilization than Baseline, particularly when the load is not very high. This occurs because power-saving techniques are applied aggressively when the workload is not heavy. Offloading exhibits consistently low utilization since a large portion of tasks are executed on remote servers, although the local processor still runs at full speed. RP-FMS shows 22.9% and 770.6% higher processor utilization compared to Baseline and Offloading, respectively.

DVFS exhibits the highest utilization, reaching almost 1.0, except in very low-load conditions of 0.05 to 0.15. In such cases, even the slowest processor mode cannot fully utilize the available resources.

4.2. Results for Dynamic Task Sets

This section evaluates the system's performance when the task set dynamically evolves over time. Specifically, we simulate four workload scenarios that reflect various production task variations. Table 1 summarizes these scenarios, listing the sequence of workload changes over time. For instance, in Scenario 1, the task load is 0.2 for the first 40% of the execution time, then increases to 0.8 for the next 20%, and finally settles at 0.4 for the remaining 40%.

Table 1. Workload scenarios.

Scenario	Sequence of Workloads Load of Task Set (% of Execution Time)
1	0.2 (40%) – 0.8 (20%) – 0.4 (40%)
2	0.4 (60%) – 0.9 (10%) – 0.6 (30%)
3	0.1 (75%) – 0.4 (15%) – 0.7 (10%)
4	0.4 (70%) – 0.7 (30%)

Figure 8 presents the energy consumption of the four schemes, Baseline, DVFS, Offloading, and RP-FMS, across different workload scenarios. In this experiment, the Offloading and DVFS schemes are also adapted to workload variations through dynamic resource plan optimization, ensuring a fair comparison with RP-FMS. Similar to previous graphs, energy consumption is normalized relative to Baseline's energy consumption at a load of 0.1.

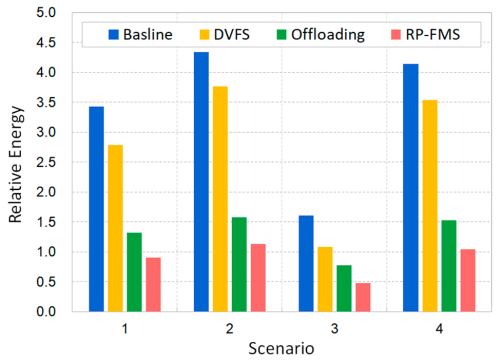


Figure 8. Comparison of energy consumption for four scenarios.

As shown in Figure 8, RP-FMS consistently achieves the lowest energy consumption, significantly outperforming DVFS, Offloading, and Baseline across all workload scenarios. On average, RP-FMS reduces energy consumption by 73.1% compared to Baseline, 65.9% compared to DVFS, and 32.3% compared to Offloading.

DVFS shows limited energy savings, as it is only effective when the task load is sufficiently low, allowing for processor slowdown without violating deadlines. In contrast, Offloading achieves substantial energy savings, as it remains beneficial regardless of task load conditions. Additionally, when Offloading is used, it creates idle time slots in the local processor, further increasing the opportunities for DVFS utilization. This synergy between DVFS and Offloading is the key factor enabling RP-FMS to achieve superior energy efficiency.

Figure 9 illustrates processor utilization across the four workload scenarios. As seen in Figure 9, DVFS maintains the highest resource utilization across all scenarios, as it dynamically adjusts processor speed to match workload conditions. Meanwhile, Offloading exhibits consistently low utilization since it does not adjust the processor speed for local tasks after migration, limiting further improvements in resource utilization. In contrast, RP-FMS maintains consistently high utilization by lowering the processor speed when idle slots remain after offloading. The observed improvements in processor utilization are 25.4% compared to Baseline and 820.7% compared to Offloading.

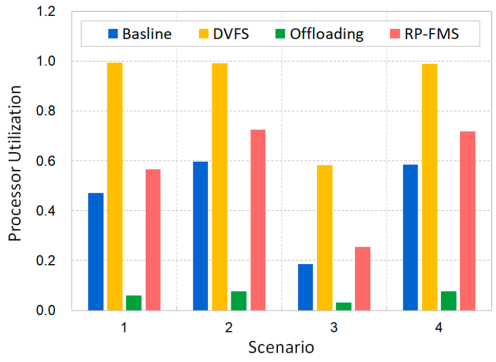
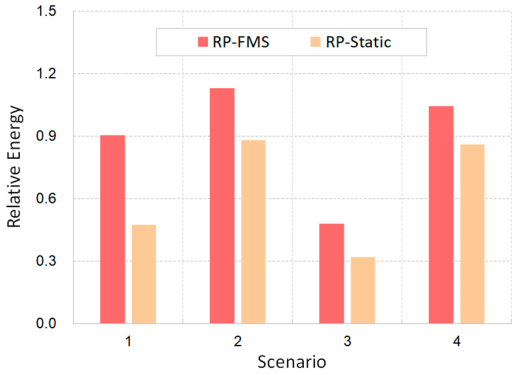


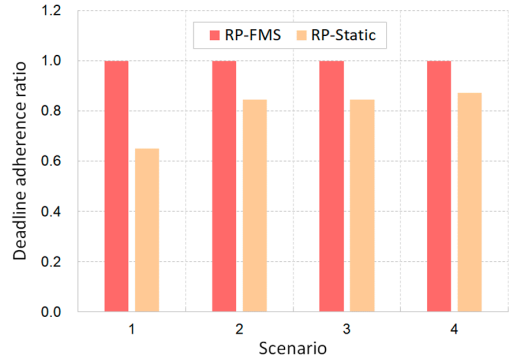
Figure 9. Comparison of processor utilization for four scenarios.

Before concluding this section, we compare RP-FMS with its static counterpart, RP-Static. RP-Static also optimizes processor speed, memory allocation, and offloading deci-

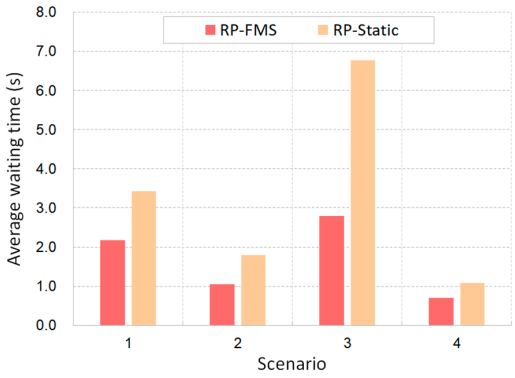
sions through evolutionary computation like RP-FMS, but it does not adapt to workload changes. Figure 10a compares the energy consumption of RP-FMS and RP-Static. As shown, RP-Static saves more energy than RP-FMS in all cases, particularly in Scenarios 1 and 3, where the initial task load is low but undergoes large variations. However, RP-Static fails to meet deadline constraints when the task set changes.



(**a**) Energy consumption



(**b**) Deadline adherence ratio



(**c**) Average waiting time

Figure 10. Comparison of RP-FMS with RP-Static.

Figure 10b illustrates the deadline-adherence ratio of both schemes. RP-FMS maintains a 100% deadline-adherence ratio across all workload scenarios, while RP-Static exhibits significant degradation in deadline adherence. The worst case occurs in Scenario 1, where the initial task load is low, but increases significantly for most of the execution time, leading to frequent deadline violations.

To further assess system performance, we analyze the responsiveness of user-interaction tasks. Figure 10c presents the average waiting time for user-interaction tasks in RP-FMS and RP-Static. As shown in the figure, the average waiting time in RP-FMS is 1.6 s, with a maximum waiting time of 2.8 s. This indicates that RP-FMS ensures user-interaction task's responsiveness within 3 s across all workload conditions. In contrast, RP-Static ex-

hibits significant variations in waiting time. Specifically, in Scenario 3, the average waiting time increases to 6.8 s, demonstrating that RP-Static fails to guarantee timely execution of user-interaction tasks. This occurs because RP-Static maintains preallocated task slots regardless of workload fluctuations, causing a queue buildup when user-interaction tasks suddenly increase, leading to excessive delays.

In summary, the adaptive version of the proposed scheme, RP-FMS, reduces energy consumption while ensuring deadline adherence and minimizing latency, even under dynamic workload conditions. When the task load is low, RP-FMS lowers processor speed to maximize energy savings. Conversely, when the task load becomes heavy, RP-FMS prioritizes time constraints by increasing processor speed and offloading ratio, ensuring a flexible and sustainable system.

5. Related Works

Research in real-time task scheduling has explored various directions, ranging from classical utilization-based models to adaptive and learning-based approaches for dynamic environments. Our work integrates scheduling under strict deadline guarantees with human–machine interaction and resource optimization in IIoT-enabled flexible manufacturing. This section categorizes prior work across three themes: scheduling under dynamic task sets, cloud-edge collaboration, and human–machine interaction.

5.1. Dynamic Task Scheduling and Learning-Based Methods

Recent studies have addressed dynamic scheduling through fairness mechanisms or machine learning. For instance, Sun et al. [34] proposed a task scheduler called Proportional Fairness-Aware Auction with Proximal Policy Optimization (PFAPPO) for Space-Air-Ground Integrated Networks (SAGIN). Their scheme allocates computing resources reasonably to each UAV (Unmanned Aerial Vehicle), which then makes intelligent offloading decisions using the distributed deep reinforcement learning Proximal Policy Optimization (PPO) algorithm. While effective in multi-resource balancing and adaptive task allocation across heterogeneous platforms, the model operates under soft deadline assumptions and does not explicitly address HMI or energy-related aspects.

Reinforcement learning has also been applied to job shop scheduling. Li et al. [35] developed a Deep Q-Network (DQN)-based scheduler for automated guided vehicles (AGVs), optimizing task assignment in real time. However, this model focuses on discrete job transitions and does not incorporate hard real-time constraints or heterogeneous resource management as primary design goals.

Similarly, Zhang et al. [36] proposed a heterogeneous graph-based reinforcement learning method for surface-mount technology (SMT) workshops, showing improved scheduling performance under uncertain task loads. Although effective, the approach is not designed for hard real-time systems and does not accommodate user-triggered task set changes.

Other approaches have explored optimization frameworks such as fuzzy modeling. Wei et al. [37] introduced a multi-objective fuzzy optimization model for flight scheduling in complex airspace scenarios. The focus lies in prioritizing competing mission goals rather than accommodating dynamic task sets or real-time responsiveness.

In contrast, our framework supports runtime task set modifications triggered by human interactions while maintaining strict real-time guarantees. Table 2 summarizes the comparative characteristics of RP-FMS and recent task scheduling frameworks, highlighting differences in task adaptability, real-time guarantees, energy efficiency, and support for human–machine interaction.

Table 2. Comparison of RP-FMS with recent scheduling models across key functional dimensions.

Approach	Task Set Adaptability	Real-Time Deadline Guarantee	HMI Support	Energy Optimization	Methodology
PFAPPO (Sun et al., 2024) [34]	Fairness-based adaptation	Soft real-time	N/A	Partially via resource balancing	Auction-based + PPO
DQN-AGV Scheduler (Li et al., 2025) [35]	Allowing dynamic task insertion	No deadline model	N/A	N/A	Online DQN learning
Graph-RL SMT Scheduler (Zhang et al., 2025) [36]	Predictive workload handling	N/A	N/A	Implicitly via rewards	GNN + RL
Cloud-Edge Collaboration (Li et al., 2025) [38]	Rolling-horizon rescheduling	N/A	N/A	Through cloud-edge efficiency	Scalable neurocomputing
Voice-Controlled HMI (Mnassri et al., 2024) [39]	N/A	N/A	Voice-based HMI	N/A	Embedded system design
Omnidirectional Acoustic Sensor (Qiao et al., 2024) [40]	N/A	N/A	Acoustic-based HMI	N/A	Triboelectric sensor
HeadTrack (Hu et al., 2023) [41]	N/A	N/A	HCI via earphones	N/A	Audio-based tracking
AR Teleoperation (Wang et al., 2024) [42]	N/A	N/A	AR-based interaction	N/A	Augmented reality interface
Hypergraph Q-Learning (Shi et al., 2024) [43]	Implicitly through theoretical model	N/A	N/A	N/A	Hypergraph-based Q-learning
RP-FMS	Explicitly through precomputed plan switching	100% deadline compliance	Dedicated slots + task set adaptation	Explicitly optimized	Hybrid offline planning + online rescheduling

5.2. Cloud-Edge Collaboration and Energy-Aware Scheduling

Several works have investigated cloud-edge coordination to optimize computational performance in IIoT environments. Li et al. [38] introduced a neurocomputing-based rolling-horizon optimization model for industrial scheduling. Their framework dynamically allocates workloads across cloud and edge layers, improving scalability. However, it does not include dedicated mechanisms for latency-sensitive human interaction tasks and assumes soft deadline conditions.

Our approach complements such models by combining offline planning and online task reassignment, while enforcing hard real-time schedulability and minimizing energy consumption through DVFS, memory allocation, and offloading control.

5.3. Human–Machine Interaction and Intelligent Interfaces

Advances in HMI technologies have enabled intuitive control interfaces across robotics, assistive devices, and cyber-physical systems. For example, Mnassri et al. [39] designed a real-time, voice-controlled interface for wheelchairs using a Raspberry Pi. Qiao et al. [40] proposed an omnidirectional acoustic sensor to enhance auditory responsiveness in HMI systems, while Hu et al. [41] developed HeadTrack, a real-time HCI framework using wireless earphones for gesture-free control.

In robotic systems, Augmented Reality (AR) has been used to bridge human feedback and robot actuation. Wang et al. [42] presented a teleoperation interface using AR overlays for natural human–robot interaction. Although these systems advance interactive responsiveness, they focus on interface modality rather than integration into real-time production environments.

From a learning-based perspective, Shi et al. [43] introduced a hypergraph-based model to represent Q-learning dynamics in multi-agent scenarios. While not directly applicable to real-time scheduling, the model demonstrates how learning and structural task dynamics can be jointly modeled—a concept we address in deterministic form via precomputed resource plans.

Our framework draws inspiration from these efforts by not only reserving slots for HMI tasks but also allowing such inputs to reconfigure production task sets with immediate resource reassignment, a feature not addressed in prior HMI-integrated scheduling models.

6. Conclusions

This paper presented a resource planning framework for IIoT-based flexible manufacturing systems, addressing the challenge of integrating real-time production tasks and sporadic user-interaction tasks into a unified scheduling model. Unlike prior approaches that rely on uncertainty-driven prediction to manage task set variability, our method explicitly responds to user-triggered task changes that are bounded, structurally predictable, and immediately available to the system. This enables the framework to handle task dynamics with precise, real-time adjustments rather than reactive estimations. To support this, we precompute optimized resource plans for anticipated task set variations using evolutionary computation and store them for instant application at runtime. We also reserve task slots for user-interaction tasks during offline scheduling to ensure timely responsiveness. Furthermore, the framework jointly optimizes processor speed, memory allocation, and edge/cloud offloading by considering their mutual interdependencies. Simulation results show that the proposed method achieves up to 73.1% energy savings compared to a baseline system, guarantees 100% deadline compliance for all production tasks, and maintains low latency for user-interaction tasks.

As future work, we aim to extend our framework to multi-agent and distributed manufacturing settings, incorporate predictive models to anticipate user-interaction patterns,

and validate the system on physical testbeds or digital twin platforms. These directions will further enhance the scalability and industrial applicability of the proposed approach.

Author Contributions: Conceptualization, Y.S., K.C. and H.B.; Software, G.K., Y.S. and K.C.; Validation, Y.S. and K.C.; Formal analysis, G.K. and Y.S.; Resources, H.B.; Data curation, K.C.; Writing—original draft, G.K. and Y.S.; Writing—review & editing, K.C. and H.B.; Visualization, G.K.; Supervision, H.B.; Project administration, H.B.; Funding acquisition, H.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported in part by the National Research Foundation of Korea (NRF) under Grant RS-2024-00461678 and the Institute of Information & Communications Technology Planning & Evaluation (IITP) under Grant RS-2022-00155966 (Artificial Intelligence Convergence Innovation Human Resources Development (Ewha University)) funded by the Korean government (MSIT).

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Pavel, M.; Stamatescu, G. Flexible Manufacturing System for Enhanced Industry 4.0 and Industry 5.0 Applications. In Proceedings of the 20th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT), Abu Dhabi, United Arab Emirates, 29 April–1 May 2024; pp. 483–490.
2. Ma, X.; Xu, H.; Gao, H.; Bian, M.; Hussain, W. Real-Time Virtual Machine Scheduling in Industry IoT Network: A Reinforcement Learning Method. *IEEE Trans. Ind. Inform.* **2023**, *19*, 2129–2139. [CrossRef]
3. Tariq, A.; Khan, S.A.; But, W.H.; Javaid, A.; Shehryar, T. An IoT-Enabled Real-Time Dynamic Scheduler for Flexible Job Shop Scheduling (FJSS) in an Industry 4.0-Based Manufacturing Execution System (MES 4.0). *IEEE Access* **2024**, *12*, 49653–49666. [CrossRef]
4. Moon, S.; Lee, S.; Park, K. Learning-enabled Flexible Job-shop Scheduling for Scalable Smart Manufacturing. *J. Manuf. Syst.* **2024**, *77*, 356–367. [CrossRef]
5. Zhang, S.; Chen, Y.; Zhang, J.; Jia, Y. Real-Time Adaptive Assembly Scheduling in Human-Multi-Robot Collaboration According to Human Capability. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 31 May–31 August 2020; pp. 3860–3866.
6. Pupa, A.; Van Dijk, W.; Secchi, C. A Human-Centered Dynamic Scheduling Architecture for Collaborative Application. *IEEE Robot. Autom. Lett.* **2021**, *6*, 4736–4743. [CrossRef]
7. Yang, J.; Liu, Y.; Morgan, P. Human–machine interaction towards Industry 5.0: Human-centric smart manufacturing. *Digit. Eng.* **2024**, *2*, 100013. [CrossRef]
8. Ayvaz, S.; Alpay, K. Predictive maintenance system for production lines in manufacturing: A machine learning approach using IoT data in real-time. *Expert Syst. Appl.* **2021**, *173*, 114598. [CrossRef]
9. Park, S.; Bahn, H. WANE: Workload Adaptive Neuro-Genetic Engine for Container Usage Prediction. In Proceedings of the 5th IEEE International Conference on Advances in Electrical Engineering and Computer Applications, Dalian, China, 16–18 August 2024; pp. 1–8.
10. Song, M.; Lee, Y.; Kim, K. Reward-Oriented Task Offloading under Limited Edge Server Power for Multiaccess Edge Computing. *IEEE Internet Things J.* **2021**, *8*, 13425–13438. [CrossRef]
11. Bi, J.; Yuan, H.; Duanmu, S.; Zhou, M.; Abusorrah, A. Energy-Optimized Partial Computation Offloading in Mobile-Edge Computing with Genetic Simulated-Annealing-Based Particle Swarm Optimization. *IEEE Internet Things J.* **2021**, *8*, 3774–3785. [CrossRef]
12. Panda, S.; Lin, M.; Zhou, T. Energy Efficient Computation Offloading with DVFS Using Deep Reinforcement Learning for Time-Critical IoT Applications in Edge Computing. *IEEE Internet Things J.* **2023**, *10*, 6611–6621. [CrossRef]
13. Barletta, M.; Cinque, M.; Simone, L.; Corte, R. Criticality-Aware Monitoring and Orchestration for Containerized Industry 4.0 Environments. *ACM Trans. Embed. Comput. Syst.* **2024**, *23*, 5. [CrossRef]
14. Khan, A.N.; Iqbal, N.; Rizwan, A.; Malik, S.; Ahmad, R.; Kim, D.H. A Criticality-Aware Dynamic Task Scheduling Mechanism for Efficient Resource Load Balancing in Constrained Smart Manufacturing Environment. *IEEE Access* **2022**, *10*, 50933–50946. [CrossRef]
15. Lu, Y.; Yang, L.; Yang, S.; Hua, Q.; Sangaiah, A.; Guo, T.; Yu, K. An Intelligent Deterministic Scheduling Method for Ultralow Latency Communication in Edge Enabled Industrial Internet of Things. *IEEE Trans. Ind. Inform.* **2023**, *19*, 1756–1767. [CrossRef]

16. Kumar, M.; Walia, G.K.; Shingare, H.; Singh, S.; Gill, S. AI-Based Sustainable and Intelligent Offloading Framework for IIoT in Collaborative Cloud-Fog Environments. *IEEE Trans. Consum. Electron.* **2024**, *70*, 1414–1422. [CrossRef]
17. Cui, L.; Xu, C.; Yang, S.; Huang, J.; Li, J.; Wang, X.; Ming, Z.; Lu, N. Joint Optimization of Energy Consumption and Latency in Mobile Edge Computing for Internet of Things. *IEEE Internet Things J.* **2019**, *6*, 4791–4803. [CrossRef]
18. Kumar, N.; Lee, S. Human-Machine Interface in Smart Factory: A Systematic Literature Review. *Technol. Forecast. Soc. Change* **2022**, *174*, 121284. [CrossRef]
19. Guo, J.; Ma, S.; Zeng, S.; Che, H.; Pan, X. A Risk Evaluation Method for Human-Machine Interaction in Emergencies Based on Multiple Mental Models-Driven Situation Assessment. *Reliab. Eng. Syst. Saf.* **2024**, *252*, 110444. [CrossRef]
20. Oikonomou, P.; Kolomvatsos, K.; Anagnostopoulos, C. A Proactive and Uncertainty Driven Management Mechanism for Workflows of Processing Tasks. *Computing* **2025**, *107*, 112. [CrossRef]
21. Choi, K.; Lee, W.; Soma, R.; Pedram, M. Dynamic Voltage and Frequency Scaling under a Precise Energy Model Considering Variable and Fixed Components of the System Power Dissipation. In Proceedings of the IEEE International Conference on Computer Aided Design (ICCAD), San Jose, CA, USA, 7–11 November 2004; pp. 29–34.
22. Wang, Y.; Sheng, M.; Wang, X.; Wang, L.; Li, J. Mobile-Edge Computing: Partial Computation Offloading Using Dynamic Voltage Scaling. *IEEE Trans. Commun.* **2016**, *64*, 4268–4282. [CrossRef]
23. Wang, G.; Guan, Y.; Wang, Y.; Shao, Z. Energy-Aware Assignment and Scheduling for Hybrid Main Memory in Embedded Systems. *Computing* **2016**, *98*, 279–301. [CrossRef]
24. Zhan, J.; Zhang, Y.; Jiang, W.; Yang, J.; Li, L.; Li, Y. Energy-Aware Page Replacement and Consistency Guarantee for Hybrid NVM–DRAM Memory System. *J. Syst. Archit.* **2018**, *89*, 60–72. [CrossRef]
25. Kwon, G.; Bahn, H. Quantifying the Effectiveness of Cloud and Edge Offloading: An Optimization Study on Energy Efficiency of Mobile Real-Time Systems. In Proceedings of the 7th IEEE International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Fukuoka, Japan, 18–21 February 2025; pp. 1–6.
26. Lee, Y.; Doh, Y.; Krishna, C. EDF Scheduling Using Two-Mode Voltage Clock Scaling for Hard Real-Time Systems. In Proceedings of the ACM International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES), Atlanta, GA, USA, 16–17 November 2001; pp. 221–228.
27. Chetto, H.; Chetto, M. Some Results of the Earliest Deadline Scheduling Algorithm. *IEEE Trans. Softw. Eng.* **1989**, *15*, 1261–1269. [CrossRef]
28. Anderson, J.; Srinivasan, A. Mixed Pfair/ERfair Scheduling of Asynchronous Periodic Tasks. *J. Comput. Syst. Sci.* **2004**, *68*, 157–204. [CrossRef]
29. Nam, S.A.; Bahn, H. Adaptive Swapping for Variable Workloads in Real-Time Task Scheduling. In Proceedings of the IEEE International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI), Chongqing, China, 18–20 October 2023; pp. 1–6.
30. Silberschatz, A.; Galvin, P.B.; Gagne, G. *Operating System Concepts*, 9th ed.; Wiley: Hoboken, NJ, USA, 2014.
31. Goldberg, D. *Genetic Algorithms in Search, Optimization, and Machine Learning*; Addison-Wesley: Boston, MA, USA, 1989.
32. Kwon, S.; Bahn, H. Evolutionary Computation-Based Scheduling of Machine Learning Workloads for GPU Clusters. In Proceedings of the 5th IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA), Dalian, China, 16–18 August 2024; pp. 1–5.
33. Ki, S.; Byun, G.; Cho, K.; Bahn, H. Co-Optimizing CPU Voltage, Memory Placement, and Task Offloading for Energy-Efficient Mobile Systems. *IEEE Internet Things J.* **2023**, *10*, 9177–9192. [CrossRef]
34. Sun, G.; Wang, Y.; Yu, H.; Guizani, M. Proportional Fairness-Aware Task Scheduling in Space-Air-Ground Integrated Networks. *IEEE Trans. Serv. Comput.* **2024**, *17*, 4125–4137. [CrossRef]
35. Li, Z.; Gu, W.; Shang, H.; Zhang, G.; Zhou, G. Research on Dynamic Job Shop Scheduling Problem with AGV Based on DQN. *Clust. Comput.* **2025**, *28*, 236. [CrossRef]
36. Zhang, B.; Sang, H.; Lu, C.; Meng, L.; Song, Y.; Jiang, X. Integrated Heterogeneous Graph and Reinforcement Learning Enabled Efficient Scheduling for Surface Mount Technology Workshop. *Inf. Sci.* **2025**, *708*, 122023. [CrossRef]
37. Wei, M.; Yang, S.; Wu, W.; Sun, B. A Multi-Objective Fuzzy Optimization Model for Multi-Type Aircraft Flight Scheduling Problem. *Transport* **2024**, *39*, 313–322. [CrossRef]
38. Li, Q.; Li, L.; Liu, Z.; Sun, W.; Li, W.; Li, J.; Zhao, W. Cloud-Edge Collaboration for Industrial Internet of Things: Scalable Neurocomputing and Rolling-Horizon Optimization. *IEEE Internet Things J.* **2025**. *preprint*. [CrossRef]
39. Mnassri, A.; Nasri, S.; Boussif, M.; Cherif, A. Real-Time Voice-Controlled Human Machine Interface System for Wheelchairs Implementation Using Raspberry Pi. *Int. J. Veh. Inf. Commun. Syst.* **2024**, *9*, 81–102. [CrossRef]
40. Qiao, W.; Zhou, L.; Zhang, J.; Liu, D.; Gao, Y.; Liu, X.; Zhao, Z.; Guo, Z.; Li, X.; Zhang, B.; et al. A Highly-Sensitive Omnidirectional Acoustic Sensor for Enhanced Human–Machine Interaction. *Adv. Mater.* **2024**, *36*, 2413086. [CrossRef]
41. Hu, J.; Jiang, H.; Xiao, Z.; Chen, S.; Dustdar, S.; Liu, J. HeadTrack: Real-Time Human–Computer Interaction via Wireless Earphones. *IEEE J. Sel. Areas Commun.* **2024**, *42*, 990–1002. [CrossRef]

42. Wang, X.; Guo, S.; Xu, Z.; Zhang, Z.; Sun, Z.; Xu, Y. A Robotic Teleoperation System Enhanced by Augmented Reality for Natural Human–Robot Interaction. *Cyborg Bionic Syst.* **2024**, *5*, 0098. [CrossRef] [PubMed]
43. Shi, J.; Liu, C.; Liu, J. Hypergraph-Based Model for Modeling Multi-Agent Q-Learning Dynamics in Public Goods Games. *IEEE Trans. Netw. Sci. Eng.* **2024**, *11*, 6169–6179. [CrossRef]

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.