TU Delft

Delft University of Technology

Document Version
Final published version

Licence
CC BY

Citation (APA)
Li, H., Amiri Simkooei, A., & Ribeiro, M. J. (2026). Dynamic aircraft maintenance scheduling with Graph Neural Network integrated deep reinforcement learning. *Reliability Engineering & System Safety*, *277*, Article 113392. https://doi.org/10.1016/j.ress.2026.113392

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
In case the licence states “Dutch Copyright Act (Article 25fa)”, this publication was made available Green Open Access via the TU Delft Institutional Repository pursuant to Dutch Copyright Act (Article 25fa, the Taverne amendment). This provision does not affect copyright ownership.
Unless copyright is transferred by contract or statute, it remains with the copyright holder.

Sharing and reuse
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.

Reliability Engineering and System Safety 277 (2027) 113392



Contents lists available at ScienceDirect

Reliability Engineering and System Safety

journal homepage: www.elsevier.com/locate/ress



Dynamic aircraft maintenance scheduling with Graph Neural Network integrated deep reinforcement learning

Haonan Li *, Álvaro Lanza Rausell, Alireza Amiri-Simkooei, Marta Ribeiro

Delft University of Technology, Faculty of Aerospace Engineering, Operation & Environment, The Netherlands

ARTICLE INFO

Keywords:
Aircraft maintenance policy
Maintenance scheduling
Reinforcement learning
Graph Neural Network

ABSTRACT

Industrial aircraft maintenance policies still rely on rigid, static slot structures that limit operational efficiency. Existing research has mainly addressed task (re-)allocation within a pre-defined number and duration of slots, while direct slot planning (jointly determining slot duration and allocation) remains difficult due to its combinatorial complexity. We propose a graph-based, reinforcement-learning framework that sequentially constructs maintenance task packages within a fixed planning horizon. A Graph Neural Network (GNN) encodes relations among tasks to inform a deep reinforcement learning (DRL) policy, which learns to group tasks into slots efficiently. This framework is trained and evaluated on operational data from a major European airline and benchmarked against a fixed-time Mixed-Integer Linear Programming (MILP) benchmark and the current industry practice. The results demonstrate that the model rapidly generates feasible schedules, and quantify trade-offs among interval spillage, panel reuse, and drop-out tasks relative to these benchmarks. Our model is able to decrease task spillage and increase panel reuse consistently. Finally, as the reward weights can be tuned to reflect an airline's operational preferences, this framework provides a flexible decision-support tool for dynamic aircraft maintenance task packaging.

1. Introduction

Aircraft Maintenance, Repair, and Overhaul (MRO) activities are central to airline safety, regulatory compliance, and financial performance. MRO costs account for roughly 10% of an airline's total operating costs [1], and a 10% reduction in those costs can potentially double profits [2], so even modest efficiency gains carry significant financial weight.

Maintenance policy determines the structure of MRO programs: the scheduling, frequency, and duration of aircraft checks. Many current airline policies are static and have remained unchanged for years, partly to establish stable patterns that reduce human error. This rigidity produces systematic planning inefficiencies. Optimizing the policy itself therefore addresses these inefficiencies at their source, rather than fine-tuning task scheduling within an already fixed structure, which produces more modest improvements.

Although considerable research has focused on optimizing maintenance task scheduling within predefined slots [3,4], the foundational design of the slots themselves remains largely unexplored. A further limitation of prior work is the absence of fleet-level optimization, which can create challenges in managing hangar availability [4,5]. This paper addresses both gaps.

We hypothesize that a key limitation of existing maintenance policies is their failure to exploit the complex interdependencies among maintenance tasks. Tasks are often interconnected through shared access requirements (e.g., opening the same panel), common skill sets, or synchronized maintenance intervals. To model and exploit these relational structures, we propose a framework that integrates a Graph Neural Network (GNN) with a Deep Reinforcement Learning (DRL) agent. The GNN captures the network of task relationships; the DRL agent learns a policy for grouping and scheduling those tasks dynamically. All models are trained and validated on operational data from a major European airline.

The primary contributions of this research are four-fold:

1. We demonstrate the application of DRL to achieve fast and cost-effective optimization in this complex domain.
2. We incorporate a GNN to extract and exploit relational task information, leading to more informed and efficient maintenance decisions.
3. We address multiple objectives simultaneously (i.e. spillage, panel reuse, and workload balancing) to improve practical applicability and alignment with real-world operational constraints.

* Corresponding author.
E-mail address: george.li@tudelft.nl (H. Li).

https://doi.org/10.1016/j.ress.2026.113392
Received 26 January 2026; Received in revised form 26 August 2026; Accepted 27 August 2026
Available online 3 September 2026
0951-8320/© 2026 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

4. We deliver a flexible decision-support tool that allows airlines to generate and compare alternative maintenance policies under different operational preferences (e.g. prioritizing interval spillage versus panel-sharing), enabling policy exploration prior to implementation.

The remainder of this paper is organized as follows. Section 2 provides a formal problem definition. Section 3 reviews the relevant literature. Section 4 details the proposed methodology. Sections 5, 6 and 7 introduce the case study, hypotheses, and experimental setup, respectively, followed by the training results in Section 8. Section 9 reports an illustrative exactly solved toy comparison, reward-weight sensitivity, task-edge ablation, a fleet-level fixed-time benchmark, raw-policy multi-seed stability, realistic initial-status results, maintenance slot construction quality, and stage-wise conflict-solver effects. Section 10 discusses the implications of our findings. Finally, Section 11 concludes the paper.

2. Problem definition

This section formally defines the maintenance scheduling problem addressed in this research. To establish the necessary context, in Section 2.1, we first provide a detailed characterization of the incumbent maintenance policy currently employed by the airline, which is structured around a rigid, block-based system. Following this description, in Section 2.2, we formulate the core problem, identifying the significant inefficiencies such as interval wastage that arise from this static framework. Finally, we establish the primary research objective in Section 2.2: to develop a dynamic maintenance policy that overcomes these limitations while preserving long-term planning stability.

2.1. Incumbent maintenance policy: A block-based system

Airlines employ a block-based maintenance policy to structure their routine maintenance activities. In Fig. 1, we provide the structure of types of tasks in A-check. The scope of this paper will be focusing on the repetitive tasks. Major part of repetitive maintenance tasks are aggregated into predefined sequential **"blocks"** that cycle over time. Fig. 2 illustrates a four-block cycle for one aircraft, where Block 2 must follow Block 1, Block 3 follows Block 2, and the sequence resets to Block 1 after Block 4 is completed. The used 4 blocks are only illustrative, the actual numbers of blocks may differ. The intervals between blocks are predetermined. Under an ideal scenario, each block and their due dates are known for the foreseeable future.

Additionally, within this framework, maintenance tasks are classified into two primary categories, illustrated in Fig. 2, and elaborated as follows:

1. In-block Tasks: These tasks are statically assigned to one or more blocks. Their execution is definitively initiated by the scheduled appearance of their associated block. For example, if Task 1 is assigned to Block 1, it will be performed every time Block 1 is scheduled. The system utilizes iterative and strictly ordered blocks to implement a standard operational pattern, a design choice intended to enhance procedural reliability and reduce human error.
2. Drop-Out Tasks (DO Tasks): These tasks are not tied to any specific block. Instead, they are scheduled dynamically based on their own prescribed intervals (e.g. flight hours or cycles). Drop-Out tasks exist because their intervals fundamentally misalign with the block intervals. This can lead to significant resource underutilization (i.e., waste) when attempting to allocate. DO Tasks added pressure on the planner increases the risk of errors, and can lead to a less effective maintenance schedule.

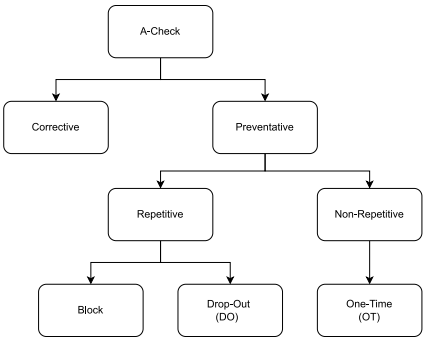


Fig. 1. Structure of types of tasks carried out in A-checks.

A maintenance check is therefore defined as a work package assigned to an aircraft's maintenance **"Slot"**. This package is comprised of three distinct components: (1) The primary scheduled **"Block"**, which encapsulates multiple In-block tasks, (2) Any Drop-Out tasks that have simultaneously become due, and (3) Any other maintenance tasks,such as modifications. One work package usually only contains one **"Block"**.

2.2. Problem formulation and research objective

The block-based system offers many operational advantages, primarily stability and predictability. This deterministic structure simplifies long-term planning for material and labour allocation and reduces the cognitive load on maintenance technicians. The rigidity of this system, however, introduces considerable inefficiency. Its primary limitation is the inability to dynamically adapt to the actual state of the aircraft or leverage the interdependencies among tasks. This rigidity leads to a phenomenon we term 'interval wastage'. For example, a task with a permissible interval of 1500 flight hours might be assigned to a block that repeats every 1000 h. As a result, the task is performed four times more frequently than required by safety regulations, resulting in a 50% loss of its potential service interval. This problem is exacerbated when unplanned maintenance events or early task completions disrupt the static schedule, since the system cannot re-optimize the plan to account for these changes.

The central problem addressed in this paper is the inherent inefficiency of the block-based static maintenance policy. Our primary objective is to deconstruct this rigid framework and develop a novel method for generating dynamic maintenance policies based on their interconnections of panels and task intervals. This method groups tasks directly into optimized work packages, without relying on a repetitive block structure. Since repeatable blocks are no longer used, the subsequent methodology discard the block structure. Instead, maintenance opportunities are represented solely as slots, and tasks are directly assigned to these specific slots. To retain the benefit of long-term predictability, the proposed policy will be designed to generate a stable maintenance plan over an extended horizon, thereby providing planning stability to the airline while maximizing operational efficiency.

3. Literature review

This chapter presents a comprehensive literature review structured as follows. Section 3.1 establishes the current state-of-the-art in Aircraft Maintenance Optimisation. Addressing the limitations inherent in existing research, Section 3.2 draws inspiration from analogous problems in related fields to identify potential solutions. Subsequently, Section 3.3 evaluates both exact and heuristic methodological approaches. The review concludes in Section 3.4 by delineating the specific research gap this study aims to bridge.

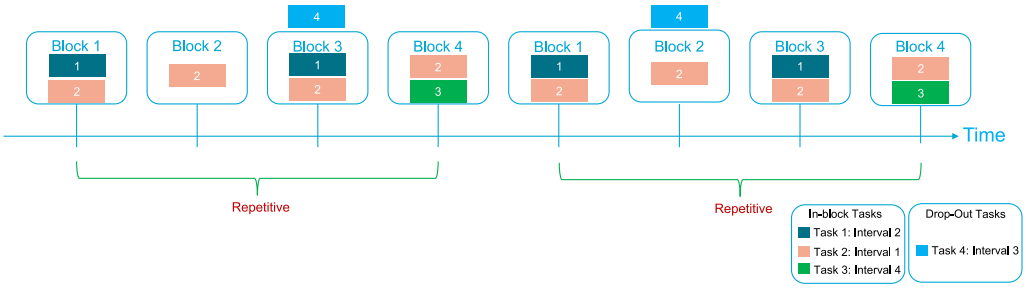


Fig. 2. Illustration of task assignment within a repetitive block system. Tasks within a specific block inherit the block's due date (Tasks 1, 2 and 3). Task 4, designated as an Drop-Out (DO) task, is scheduled in the block nearest to its specific due date.

3.1. Research in aircraft maintenance optimization

Aircraft maintenance optimization is a widely studied field, reflecting its crucial importance for aviation safety, operational reliability, and financial performance. The literature documents a clear evolution in maintenance philosophies, moving from traditional time-based Preventive Maintenance (PM) and reactive Corrective Maintenance (CM) towards more sophisticated data-driven strategies. Modern approaches such as Condition-Based Maintenance (CBM) and Predictive Maintenance (PdM) leverage real-time data and prognostic models to predict the Remaining Useful Life (RUL) of a component, aiming to perform maintenance only when there is evidence of degradation or imminent failure [6]. In existing work, we define two primary areas:

1. The development of prognostic models for PdM [1,7–12]. These models employ machine learning algorithms and condition-monitoring data to predict the RUL of components, thereby optimizing maintenance planning.
2. The optimization of short-term scheduling for pre-defined maintenance packages [3,4]. As Adimonyemma and Sun [13] highlight, very few studies address the precedent, long-term strategic problem: how to optimally group thousands of individual maintenance tasks into efficient work packages that are subsequently scheduled.

This focus on short-term challenges is evident in recent research. Many studies address maintenance task assignment by focusing on minimizing costs and interval waste among uncertain task arrivals [14–17]. Others develop models for real-time rescheduling to handle disruptions, although they often disregard wider network considerations [18]. Alternative approaches integrate tail assignment with maintenance scheduling using methods like dynamic programming to schedule tasks into nightly slots [19]. However, this can overlook resource availability. This body of work aligns with a broader trend towards "robust planning" [20], but it essentially assumes that the maintenance packages are already defined.

More limited research has explored the long-term aircraft maintenance Task Allocation Problem (TAP), which is closer to addressing the grouping gap. Early work integrated maintenance planning with flight operations, however, prioritized flight support over maintenance optimization [21]. In contrast, Steiner [22] introduced a heuristic approach aimed at minimizing maintenance actions and balancing capacity. Subsequent models focused explicitly on task-packaging strategies. For example, Muchiri and Smit [23] developed a model to cluster tasks and introduced the "de-escalation" concept to measure the cost of performing maintenance earlier than necessary. Similarly, Hölzel et al. [3] presented a cost-optimization method that prioritized tasks by labour hours, while Li et al. [5] used a fuzzy C-means clustering algorithm to group tasks based on weighted properties.

In general, the literature on this long-term TAP remains limited. Existing studies address the problem at either the aircraft level [5,23] or fleet level [3,21,22,24], but importantly, they often do not provide assessments of the optimality of the proposed groupings. This review can therefore confirm that the fundamental challenge, i.e. how to structure an optimal long-term maintenance policy by strategically grouping tasks before they become an operational scheduling problem, remains an important and largely unaddressed research area.

3.2. Inspiration from analogous combinatorial problems

Given the relative scarcity of research targeting the foundational maintenance policy problem, it is productive to seek inspiration from more extensively studied domains of combinatorial optimization. The core of the maintenance policy problem lies in efficiently grouping a large set of discrete tasks, each with its own constraints (i.e., due dates, skills, and panels), into packages to be executed at specific opportunities to minimize cost and downtime. In the literature, this is often referred to as the "task packaging", "task allocation", or "task grouping" problem. The analogy between task scheduling and the bin packing problem (BPP) has been established in previous studies [16]. Each bin represents a maintenance opportunity, whereas maintenance task represents what is inside each bin. However, this research introduces a key distinction: it operates under the constraint of a predetermined number of bins.

Finally, this task grouping and scheduling challenge also bears a strong structural resemblance to the classic Vehicle Routing Problem (VRP) or Bin Packing Problem (BPP). The VRP, in its general form, seeks to determine the optimal set of routes for a fleet of vehicles to serve a set of customers [25]. A direct analogy can thus be established:

- The maintenance opportunities (e.g., overnight hangar slots, line maintenance windows) are analogous to trips, each with a limited capacity (e.g., available man-hours).
- The individual maintenance tasks are analogous to customers, each with a specific "demand" (i.e. how many times a task must be performed in the planning horizon).
- A work package is analogous to a route, which is a sequence of customers served by a single vehicle.

3.3. Exact vs heuristic solution methods

The literature on solving VRP-like combinatorial optimization problems shows a clear evolution in methodologies. The initial approaches relied on exact methods like MILP, which are not scalable to large real-world instances [26,27]. This led to the development of classical heuristic and meta-heuristic (e.g., Genetic or Simulated Annealing) methods, which can find high-quality solutions in a reasonable amount of time, but often require extensive and problem-specific tuning and do not generalize well to new problem distributions [28,29].

The most recent and promising paradigm shift has been towards learning-based approaches, particularly Deep Reinforcement Learning (DRL) [25,30–35]. The seminal work of Nazari et al. [25] demonstrated an end-to-end framework where a neural network "agent" is trained to learn a general decision-making policy for constructing solutions.

This approach is particularly interesting for two main reasons. First, once trained, a DRL agent can rapidly construct feasible solutions, which addresses the scalability and "optimization velocity" gap. Second, a learned policy can in principle react to dynamic changes in the environment, such as an updated RUL or a schedule disruption, without re-solving the entire problem from scratch; such adaptation is not evaluated in the present fixed-horizon study.

The effectiveness of DRL critically depends on how the problem is represented to the neural network. For graph-structured problems like VRP, Graph Neural Networks (GNNs) have emerged as the state-of-the-art representation learning method. As surveyed by Cappart et al. [36], GNNs are specifically designed to operate on graph data and possess the crucial property of permutation invariance, allowing them to learn from the underlying relational structure of the problem rather than from an arbitrary ordering of inputs.

The synergy between GNNs and DRL is typically realized through an encoder–decoder architecture, a concept pioneered by the Pointer Network of Vinyals et al. [37] and later adapted for RL by Bello et al. [30]. A GNN acts as the encoder, creating powerful and context-aware embeddings of the problem state, while an attention-based decoder, such as the one proposed by Kool et al. [38], uses these embeddings to sequentially construct a solution. State-of-the-art research has focused on enhancing the GNN's ability to capture richer topological information. For example, Lei et al. [39] introduced the residual Edge-Graph Attention Network (residual E-GAT), which explicitly encodes edge features (e.g., distance or transition cost) alongside node features, leading to significant improvements in solution quality. This trend is confirmed by other recent works such as Yan et al. [40], which also integrate edge features directly into the attention mechanism. Chen and Zhou [41] also developed a GNN-boosted reinforcement learning framework for maintenance optimization in multi-dependency manufacturing
system.

3.4. Research gap

Based on the literature review, a critical research gap emerges at the intersection of strategic planning, combinatorial optimization, and methodology. This gap can be articulated in three distinct layers:

- First, a significant discrepancy exists between the focus of extant research and the fundamental strategic needs of maintenance planning. The literature is disproportionately concentrated on operational-level challenges: namely, component prognostics and the task scheduling of pre-defined maintenance policies. Consequently, the antecedent, more strategic problem of how to optimally formulate the maintenance policy itself, i.e. the optimal grouping of thousands of discrete tasks into efficient and long-term work packages, remains a foundational and largely unaddressed research area, known as policy planning.
- Second, in the limited body of work that does broach task allocation, the proposed solution methodologies exhibit a critical bottleneck. These studies predominantly rely on traditional exact methods (e.g., MILP) or classical heuristics. For a problem of this combinatorial complexity and scale, such approaches are computationally intractable or unscalable. They are not suited to the dynamic and large-scale nature of airline operations, which requires the ability to generate and adapt high-quality policies in a timely manner.
- Third, and most consequentially, no prior work has exploited the inherent relational structure of the maintenance problem to optimize the policy. Maintenance tasks are not independent entities; they are deeply interconnected by a rich topology of constraints, including intervals and panel accesses. This complex graph-like structure is ignored by existing methods. This leaves an important unexplored domain for the application of modern learning-based paradigms such as Deep Reinforcement Learning (DRL) combined with Graph Neural Networks (GNNs). These methods are specifically architected to learn from such relational data and have the potential to rapidly produce high-quality scalable solutions, thereby addressing the limitations of prior work.

The primary objective of this study is to introduce a novel decision support tool designed to assist airlines in exploring various maintenance policies, effectively bridging the aforementioned research gaps.

4. Methodology

This section outlines two approaches for optimizing aircraft maintenance policies. The first approach, defined in Section 4.1, uses a framework based entirely on Deep Reinforcement Learning (DRL). The second approach, in Section 4.2 uses a combined framework that integrates Graph Neural Networks (GNNs) with DRL.

4.1. Deep reinforcement learning framework

This section formulates the maintenance policy optimization problem as a Markov Decision Process (MDP), enabling its solution via a Deep Reinforcement Learning (DRL) framework. The formulation is defined by two primary components: i) The environment, which encapsulates the state and action spaces, and (ii) The reward function, which guides the agent's learning trajectory. The agent operates within an iterative lDO: it executes an action in a given state, receives an evaluative reward, and thus transitions to a new state. This feedback mechanism allows the agent to continuously learn and improve its policy, converging towards an optimal decision-making strategy.

4.1.1. Environment design, action, and observation space

An important step in applying DRL is the design of the environment, which serves as the interface between the optimization problem and the learning agent. This environment abstracts the real-world maintenance scheduling problem into a structured Partially Observable Markov Decision Process (POMDP). The latter is formally defined by a tuple $(\mathcal{S}, \mathcal{A}, T, R, \Omega, O, \gamma)$, representing the state space, action space, transition function, reward function, observations, observation function, and discount factor, respectively. In this subsection, we define the state space, action space, and transition dynamics. The reward function R, due to its complexity and central role in shaping the desired policy, is detailed in a subsequent dedicated section.

Observability. We consider this problem to be a POMDP, as the agent is not aware of the full queue of maintenance tasks to be planned. The *true* state would include the full predefined queue (or at least the tasks still to come), as this is information that affects future transitions and rewards. We chose not to give this information to the agent as this match the real application at deployment time, where (new) maintenance tasks can be assigned at any time.

Markov property. This property defines that the current state constitutes a sufficient statistic for all prior events when predicting future outcomes. With the current problem, one question whether information about previously planned tasks could improve future predictions, affects the future. However, since tasks are sampled without ordering constraints, consistent with the conditions of the real-world deployment setting, the sequence of past tasks carries no additional information. Consequently, we assume that the planning history does not contribute predictive value beyond the current state representation, and thus the Markov property holds.

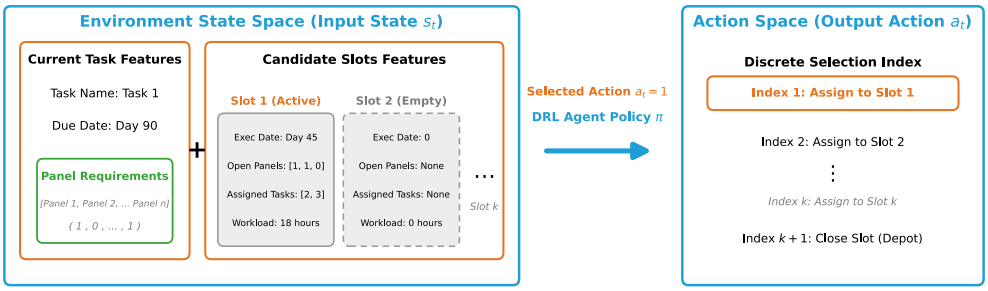


Fig. 3. Unified representation of the DRL agent's state and action spaces. The environment state space s_t (left) provides both current task features and candidate slot state features. The action space (right) is represented as a discrete choice index to assign the task to a specific slot.

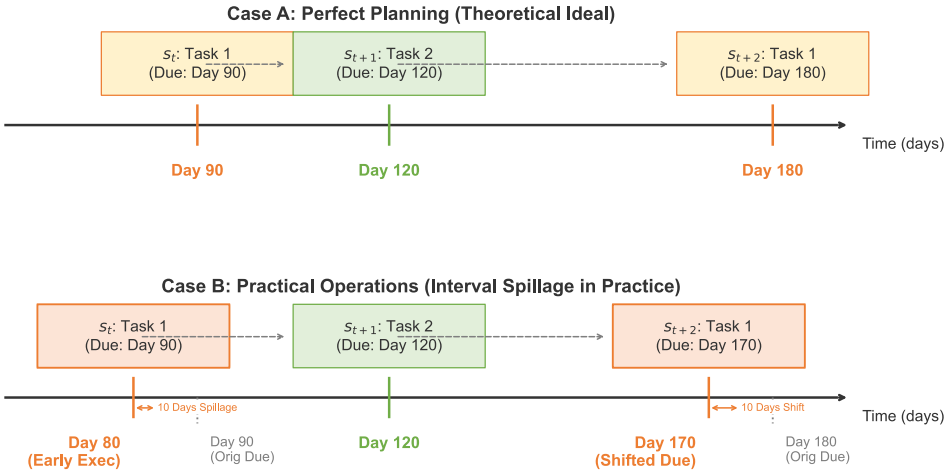


Fig. 4. Illustration of fixed-horizon slot timing. A slot is scheduled on the earliest planned due date of its constituent tasks. Tasks with later planned due dates are executed early, producing interval spillage.

Observation space ($\mathcal{S}$). The observation space $\mathcal{S}$ encompasses all possible configurations of the system observable by the agent. An observation $o_t \in \mathcal{S}$ at a given time step t provides the agent with the necessary information to select an action. For this problem, we define each observation as (1) the next sequential maintenance task requiring assignment from a predefined task list and (2) information on the existing slots. The agent goes through the complete list of tasks. As illustrated in Fig. 3, each state vector is composed of the following features: the task name, its due date (relative days from planning start date), and necessary access panels. Task name and due dates are represented as integer values, while, for panels, the categorical features are encoded using a binary representation. Additionally, information on the slots, execution day,open panels, assigned task and workload, is given.

Action space ($\mathcal{A}$). The action space $\mathcal{A}$ comprises the set of all valid actions the agent can execute from a given state. In this context, an action $a_t \in \mathcal{A}$ corresponds to assigning the current task (defined by state s_t) to a specific maintenance slot. Each slot represents a potential maintenance event and is defined by its scheduled execution date and the aggregated requirements (i.e., panels) of the tasks assigned to it. The action space can be conceptualized as a collection of these maintenance slots, which are dynamically updated as tasks are assigned. Fig. 3 provides an example, where Slot 1 represents a slot to which tasks have been assigned, thereby defining its execution date and resource requirements. If a slot is not used, all the attributes except the slot name will be empty or zero.

State transition dynamics. The reported experiments are fixed-horizon schedule-construction experiments. Each task is represented by its initial due date and maintenance interval at the beginning of an episode. A slot is dated at the earliest initial due date of its constituent tasks; consequently, every task in that slot is scheduled on or before its own planned due date, while tasks with later due dates are performed early. This early execution is termed **interval spillage**. The task demand is decremented after selection, and the transition is deterministic. The experiments do not evaluate a rolling-horizon policy that re-observes updated recurrence dates after each maintenance action (see Fig. 4).

4.1.2. Reward function engineering

The reward function, $R(s_t, a_t)$, is a crucial component that provides a scalar feedback signal to the agent, quantifying the desirability of performing action a_t in state s_t. By optimizing its policy to maximize cumulative rewards, the agent learns to achieve the desired objectives. The reward function for this research is a multi-objective formulation, tailored to the specific operational requirements of the major European airline. It is designed to incentivize three primary behaviours:

1. Minimization of interval spillage: The objective is to minimize the temporal deviation between the task's due date and its scheduled execution, thereby maximizing the utilization of the maintenance interval.
2. Reduction of panel access: To reduce the workload associated with panel removal and reinstallation, the model minimizes redundant access events. By consolidating tasks, the system ensures that a single panel opening facilitates the execution of the maximum possible number of tasks within the slot.
3. Workload balancing: The model seeks to equalize labour demand across maintenance slots, thereby facilitating optimized workforce planning and resource allocation.

An optimized maintenance policy yields concurrent economic and operational benefits. Minimizing interval spillage directly reduces costs

Table 1
Nomenclature by type.

(a) Sets and counts	
Symbol	Description
P_i	The set of all panels included in slot i.
$P_{\text{req},j}$	The set of panels required by task j.
$\lvert J_i \rvert$	The number of tasks in slot i.
$\lvert P_{\text{req},j} \cap P_i \rvert$	The number of required panels for task j included in slot i.
$\lvert P_{\text{req},j} \rvert$	The total number of panels required by task j.
$\lvert M \rvert$	The number of tasks for the whole episode.
$\lvert M_{\text{assigned}} \rvert$	The number of assigned tasks for the whole episode.
$\lvert M_{DO} \rvert$	The number of DO tasks for the whole episode.
(b) Variables	
Symbol	Description
R_{bill}	The total Spillage Reward for slot i.
S_j	The spillage metric for task j.
D_i	The execution date of slot i.
$D_{\text{due},j}$	The due date of task j.
$D_{\text{interval},j}$	The total valid time interval length for task j.
C	Task assignment completed ratio of the episode.
$\text{Var}(L)$	The variance of labour hours across all slots.
(c) Parameters and weights	
Symbol	Description
T_{spill}	The designated spillage threshold.
$T_{complete}$	The designated threshold for task complete ratio.
P_{empty}	A positive constant representing the fixed penalty for an empty slot.
α	A scaling factor (weight) for the spillage reward (α_1 for Non-GNN model, α_2 for GNN model).
β	A scaling factor (weight) for the panel reuse reward (β_1 for Non-GNN model, β_2 for GNN model).
γ	A scaling factor (weight) for the task grouping reward (γ_1 for Non-GNN model, γ_2 for GNN model).
θ	A scaling factor (weight) for the DO task penalty.(θ_1 for Non-GNN model, θ_2 for GNN model)
ω	A scaling factor (weight) for the variance penalty.(ω_1 for Non-GNN model, ω_2 for GNN model)
η	A scaling factor (weight) for the complete ratio reward.(η_1 for Non-GNN model, η_2 for GNN model)

by impact on frequency of task execution. Consolidating tasks by shared panel access significantly decreases total man-hours, mitigating the labour overhead associated with panel removal and reinstallation. Achieving equitable workload distribution across maintenance slots facilitates effective workforce planning.

To translate these objectives into a learnable signal, we design a composite reward function comprising immediate and terminal rewards. The precise magnitudes of these reward components are treated as hyper parameters and tuned during experimentation. The structure is defined by the following positive (incentivize) and negative (penalise) reward components. All notations are described in Table 1.

Immediate rewards. An immediate reward is issued after each action taken by the agent.

+/- **Spillage Reward:** A positive or negative reward is granted if a task assignment results in an interval spillage below or over a predefined threshold T_{spill} (e.g., 20%), with the reward magnitude inversely proportional to the spillage. The threshold also reflects the buffer reserved for uncertainties. The equation of spillage reward is shown in Eq. (1).

$$R_{\text{spill}} = \alpha_1(T_{\text{spill}} - S_j) = \alpha_1(T_{\text{spill}} - \frac{D_i - D_{\text{due},j}}{D_{\text{interval},j}}) \tag{1}$$

\+ **Panel Reuse Bonus:** A positive reward is given for assigning a task to a slot that already contains tasks requiring common panels. The metric for panel reuse reward is given as:

$$R_{\text{panel}} = \beta_1 \frac{\lvert P_{\text{req},j} \cap P_i \rvert}{\lvert P_{\text{req},j} \rvert} \tag{2}$$

\+ **Task Grouping Bonus:** To encourage consolidation, a reward is provided for adding a task to a non-empty slot, proportional to the number of tasks already present. The rewarding metric of task grouping is presented as follows:

$$R_{\text{group}} = \gamma_1 \lvert J_i \rvert \tag{3}$$

\- **New Slot Penalty:** A minor penalty is incurred for assigning a task to an empty slot, discouraging the unnecessary creation of new maintenance events. The metric of a new slot penalty is given as follows:

$$R_{\text{new_slot}} = \begin{cases} -P_{\text{empty}} & \text{if } \lvert J_i \rvert = 0 \\ 0 & \text{if } \lvert J_i \rvert > 0 \end{cases} \tag{4}$$

The total immediate reward after each action is the result of the sum of the previous components:

$$R_i = R_{\text{spill}} + R_{\text{panel}} + R_{\text{group}} + R_{\text{new_slot}} \tag{5}$$

Scaling factors $\alpha_1, \beta_1,$ and γ_1 are applied to weigh the reward components, ensuring the agent respects the relative importance of the defined objectives. The optimal values for these weights were established via empirical experimentation.

Terminal rewards. At the conclusion of each episode (i.e., after all tasks in the planning horizon have been assigned), a terminal reward is calculated to evaluate the quality of the overall schedule.

\- **Drop-Out Task Penalty:** A substantial penalty is applied for each slot containing only a single task, as these "Drop-Out (DO)" tasks represent inefficient planning. The equation for DO task penalty is presented as follows:

$$R_{\text{DO}} = \theta_1 \lvert M_{DO} \rvert \tag{6}$$

\- **Workload Variance Penalty:** A penalty proportional to the variance of labour hours across all valid (non-drop-out) slots is applied to promote balanced workloads. The metric for variance penalty is given as

$$R_{\text{variance}} = -\omega_1 Var(L) \tag{7}$$

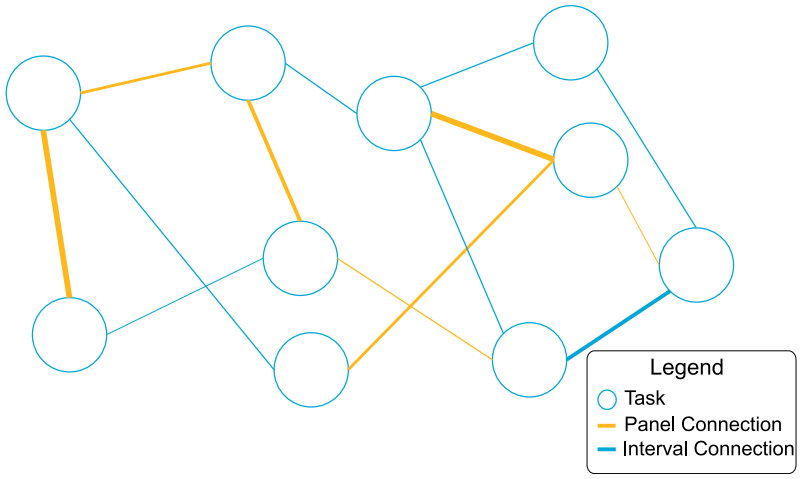


Fig. 5. Visualization of the maintenance task network. Nodes denote tasks and directed edges connect temporally proximate tasks in the initial planning horizon.

The final reward of the episode is presented as follows:

$$R_{\text{final}} = \sum R_i + R_{\text{DO}} + R_{\text{variance}} \tag{8}$$

The immediate rewards provide a denser feedback signal than the terminal rewards, allowing for more frequent behavioural adjustments. Consequently, a hybrid reward structure combining both elements guides the agent towards faster convergence.

4.2. Hybrid GNN-DRL framework

The DRL formulation from Section 4.1, while effective, does not explicitly encode temporal proximity between maintenance tasks. To represent this relation, we introduce a hybrid framework that integrates a Graph Neural Network (GNN) with the DRL agent. The GNN encodes the initial due-date neighbourhood of each task before the policy selects tasks sequentially. We first introduce graph construction for maintenance-policy optimization in Section 4.2.1; the environment design, including observation, action space, and reward function, is defined in Section 4.2.2.

4.2.1. Graph construction

For the reported GNN-SAC implementation, a graph node represents one task in the fixed planning horizon. Its input features are task identifier, maintenance interval I_i, labour requirement L_i, skill category, initial due date d_i, and the dynamic remaining demand $D_{i,t}$. Panel identifiers are retained by the environment to calculate the panel-reuse reward and reported KPI; they are not used as an edge-weight formula.

The task graph is directed. Each task node i is linked to its $K = 50$ nearest task nodes by absolute initial due-date distance,

$$e_{ij} = |d_i - d_j|, \qquad j \in \mathcal{N}_{50}(i). \tag{9}$$

The scalar e_{ij} is the edge attribute. A virtual depot node has zero features and is connected bidirectionally to task nodes with zero-valued edge attributes. Thus, the graph encodes temporal proximity in the initial planning snapshot; it does not use a hand-designed Jaccard or harmonic-interval edge weight.

The encoder first maps the concatenated node and demand features to $\mathbf{h}_i^0$. At each of three residual edge-attention layers, the edge attribute is transformed to $\mathbf{z}_{ij}$ and the complete implemented node update is

$$\begin{aligned}
\tilde{\boldsymbol{\alpha}}_{ij}^l &= \text{LeakyReLU}\left(\mathbf{W}_\alpha^l[\mathbf{h}_i^l \,\|\, \mathbf{h}_j^l \,\|\, \mathbf{z}_{ij}]\right), \\
\boldsymbol{\alpha}_{ij}^l &= \text{softmax}_{j\in\mathcal{N}(i)}\left(\tilde{\boldsymbol{\alpha}}_{ij}^l\right), \\
\mathbf{m}_i^l &= \sum_{j\in\mathcal{N}(i)} \boldsymbol{\alpha}_{ij}^l \odot \mathbf{h}_j^l, \qquad \mathbf{h}_i^{l+1} = \mathbf{h}_i^l + \text{ReLU}(\mathbf{m}_i^l).
\end{aligned} \tag{10}$$

Here $\boldsymbol{\alpha}_{ij}^l$ is a learned, channel-wise attention vector and $\odot$ denotes element-wise multiplication. The final node embeddings are supplied to the decoder and critic as described below.

Fig. 5 provides a conceptual visualization of the temporal task graph. Nodes represent tasks and edges represent initial due-date proximity.

4.2.2. Environment design

Here, an important different from the DRL formulation in Section 4.1: this graph representation has knowledge on all tasks to be planned. Therefore, we consider this approach to be an MDP. To take advantage of the graph representation, the MDP is redefined to operate directly on the graph structure. The latter is formally defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, representing the state space, action space, transition dynamics, reward function, and discount factor, respectively.

Decision process. The GNN-SAC policy observes the fixed task graph, the current remaining demand, the current capacity state, and the feasibility mask. At each step, it selects a task node to add to the open slot or the depot node to close that slot. Note that the term *sequential construction* is used to describe the order in which the policy forms slots; it does not imply rolling-horizon re-optimization of future due dates.

State space ($\mathcal{S}$). In this framework, the entire graph $\mathcal{G}$ constitutes the state $s_t \in \mathcal{S}$. The size of the state space is decided by nodes and edges of the graph. To process this high-dimensional non-Euclidean data, we employ a GNN as the agent's policy and value network. Specifically, we adapt the Residual Edge-Graph Attention (Residual-EGAT) model [39] taking into consideration the edge information and residual connections between layers to capture information of graph structures directly. This architecture processes the node and edge features to generate rich node embeddings. The attention mechanism within the GNN learns to weigh the importance of different neighbouring nodes, allowing the agent to dynamically focus on the most relevant task relationships when making a decision, which significantly improves sample efficiency.

Action space ($\mathcal{A}$). The action space $\mathcal{A}$ is the set of all task nodes in the graph, $\mathcal{A} = \mathcal{V}$. At each step within the construction of a maintenance slot, the action a_t corresponds to selecting the next task node to add to the current sequence of tasks in this slot. Selecting a non-depot node signifies the inclusion of an additional task in the current slot. Conversely, selecting the depot node terminates the current slot and triggers the initiation of a new one.

Actor-critic GNN connection. To translate the learned graph structure into decisions, the output node embeddings $\mathbf{h}_i^L \in \mathbb{R}^D$ from the final GNN layer L are supplied to an attention-based actor together with the current remaining demand and feasibility mask. The critic pools the node embeddings to obtain a graph-level representation:

1. **Actor Network (Policy Head):** A decoder query is formed from the current node embedding and the pooled graph embedding. Attention between this query and each candidate node embedding yields a raw action logit u_i:

$$u_i = \mathrm{MLP}_{\mathrm{actor}}(\mathbf{h}_i^L) \tag{11}$$

The probability $P(a_t = v_i \mid s_t)$ of selecting task v_i is computed via a masked softmax:

$$P(a_t = v_i \mid s_t) = \frac{\exp(u_i) \cdot M_i(s_t)}{\sum_{k \in \mathcal{V}} \exp(u_k) \cdot M_k(s_t)} \tag{12}$$

where $M_i(s_t) \in \{0, 1\}$ is a binary feasibility mask.

2. **Critic Network (Value Head):** To estimate the state value $V(s_t)$, we apply a global mean pooling over all node embeddings to obtain a graph-level representation $\mathbf{h}_{\mathrm{graph}}$, which is then processed by a value MLP:

$$V(s_t) = \mathrm{MLP}_{\mathrm{critic}}\left(\frac{1}{N}\sum_{i=1}^{N} \mathbf{h}_i^L\right) \tag{13}$$

Depot node representation. The special depot node $v_{\mathrm{depot}} \in \mathcal{V}$ is appended to the graph as a virtual task node with zero static features and zero demand. Selecting it closes the current slot and starts a new one. In the reference graph, it is linked bidirectionally to task nodes with zero edge attributes; in the no-edge ablation these links are removed together with all task edges.

Action masking and slot timing ($M_i(s_t)$). At each decision step, the environment excludes a candidate task if it has no remaining demand, has already been selected in the currently open slot, or would make the span between the earliest and latest initial due dates in that slot exceed the configured limit $\Delta_{\max} = 40$ days. $d_{\min,t}$ and $d_{\max,t}$ denote the smallest and largest due dates already present in the open slot, respectively. The relevant timing test is

$$M_i(s_t) = \begin{cases} 1 & \text{if } D_{i,t} > 0,\ i \notin J_t,\ \text{and } \max(d_{\max,t}, d_i) - \min(d_{\min,t}, d_i) \le \Delta_{\max}, \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

where J_t is the set of tasks already in the open slot. A task is also masked when an optional limit on the number of multi-task slots has been reached. Invalid logits are set to $-\infty$. When the depot closes a slot, its preferred execution date is set to $\min_{j \in J_t} d_j$. Therefore every task in the slot is executed no later than its initial planned due date; the timing rule controls how early a later-due task may be brought forward. Labour capacity and other operational constraints are not hard constraints in the reported implementation and remain subject to planner review.

Transition dynamics. The decision-making process is structured as the sequential construction of maintenance slots. We introduce a special **depot node** to the graph, which serves as the starting and ending point for each slot's task sequence. A single slot is formed by the agent executing a trajectory that starts at the depot node, visits a series of task nodes, and finally returns to the depot node to signify the slot's completion, as illustrated in Fig. 6.

After a task is selected, its remaining demand is decremented by one. On selecting the depot, the current slot is closed and the agent begins a new slot. The episode terminates when the remaining demands of all task nodes are zero (or, in experiments with an explicit slot budget, when that budget is exhausted). The initial due dates remain the timing reference throughout the reported fixed-horizon episode.

Algorithm 1 Fixed-horizon GNN-SAC slot construction.

Require: Task graph with initial due dates d_i, intervals I_i, demands D_i, and limit $\Delta_{\max}$
Initialize an empty open slot J, its due-date span, panel set, and labour total.
while some task has remaining demand and the optional slot budget is not exhausted **do**
  Construct the feasibility mask M using Equation (14); apply M to the actor logits.
  Select a feasible task node or the depot node.
  if a task i is selected **then**
    Add i to J; decrement D_i; update the due-date span and panel set; assign the task-selection reward.
  else if $J \neq \varnothing$ **then**
    Close J at execution date $\min_{j \in J} d_j$; assign the depot-return reward; reset the open-slot summary.
  else
    Assign the empty-return penalty.
  end if
end while
Add the completion and workload-variance terms; return the closed depot-to-depot slots.

At the end of an episode, all tasks must be assigned (i.e. zero remaining demand). The final output is structured as a series of depot-to-depot sequences, where each sequence constitutes a single maintenance slot.

4.2.3. Reward function and construction procedure

The GNN-SAC reward is implemented at the task-selection and depot-return events. Selecting a task gives a positive base reward. A positive timeliness bonus is added when the candidate's due date is within $0.2I_i$ of the earliest due date already present in the open slot, and a positive panel-reuse bonus is added when it shares an access panel with a task already in that slot. Returning to the depot closes a slot: a multi-task slot receives a positive density reward proportional to its number of tasks, whereas a one-task slot receives a negative drop-out penalty and an empty return receives a negative penalty. An additional negative penalty applies when the configured multi-task-slot limit is exceeded.

At termination, a positive completion reward is given for near-complete demand satisfaction, while incomplete termination receives a lower or negative reward. A small negative term penalises variance in the labour totals of completed multi-task slots. Per-step rewards are clipped to $[-5, 5]$. Thus, the signs are unambiguous: task grouping, timely proximity, panel reuse, and high completion are rewarded; singleton slots, empty returns, excess slots, incomplete termination, and workload imbalance are penalised. Interval spillage is evaluated as a reported schedule metric and is encouraged indirectly through the timing bonus and the date-span mask.

Numerical timing example. Consider two tasks with initial due dates $d_1 = 100$ and $d_2 = 120$ days, and intervals $I_1 = 90$ and $I_2 = 180$ days. Their 20-day separation is below $\Delta_{\max} = 40$, so the second task remains feasible after the first task has opened the slot. The completed slot is dated at day 100. Task 1 has zero spillage and task 2 is executed 20 days early, with spillage $20/180 = 11.1\%$; neither task is late.

4.3. Post-processing: Fleet-level conflict resolution

Both framework generates an optimized set of maintenance slots for a single aircraft. To scale this solution to the fleet level, a deterministic post-processing step is applied to resolve scheduling conflicts.

The initial execution date for each generated slot is set to the earliest due date among its constituent tasks. This can lead to resource

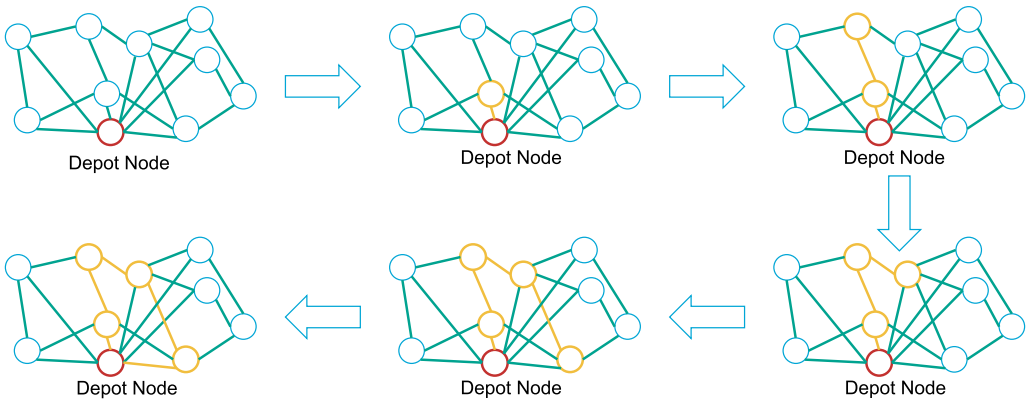


Fig. 6. Illustration of slot construction as a closed sequence. The path originates and terminates at the depot node. The agent selects task nodes step-by-step (highlighted in yellow) and finalizes the slot construction by re-selecting the depot The nodes in yellow will form a slot.

conflicts (e.g., hangar availability) if multiple slots across the fleet are scheduled concurrently. A heuristic-based conflict resolution algorithm is employed to produce a feasible fleet-wise schedule. The algorithm iteratively adjusts slot execution dates based on the following rules, prioritizing minimal deviation from the optimized plan:

1. **Slot Merging:** Slots of the same aircraft scheduled within a narrow time window (e.g., 7 days) are merged into a single visit. This consolidation is applied per aircraft, prior to fleet-level pooling.
2. **Slot Preponing:** To resolve overlaps, slots are shifted to earlier available dates. Postponement is disallowed to ensure no task exceeds its due date, thereby preserving schedule validity, although this may increase interval spillage.

The objective of this stage is to translate the optimized task groupings into a conflict-resolved operational maintenance schedule for the entire fleet.

The conflict-resolution heuristic is a deterministic post-processing component, not part of the learned policy. Slot merging can change the number of multi-task slots and panel reuse; slot preponing addresses same-day fleet conflicts while potentially increasing spillage. Because preponing only moves slots earlier, it does not create lateness relative to the initial due dates. The separate contribution of these stages is quantified in Section 9.8; the results are interpreted only for the reported metrics.

4.4. Algorithm selection

For this research, we conducted a comparative analysis of three distinct reinforcement learning (RL) algorithms across two architectural settings to identify the optimal configuration. The selected algorithms, i.e. Asynchronous Advantage Actor-Critic (A3C), Soft Actor-Critic (SAC), and Proximal Policy Optimization (PPO), were chosen for their attributes that are highly relevant in an industrial context. A3C was selected for its reputation for high training speed, SAC for its sample efficiency and robust exploration capabilities, and PPO for its stability and consistently reliable performance. Each algorithm was implemented in two configurations: one incorporating a Graph Neural Network (GNN) integration, and one using a conventional baseline architecture, to systematically evaluate the impact of the GNN architecture.

4.5. Hyper parameters

Table 2 presents the hyper parameters used in this study. These hyper parameters were initially based on those reported by Lei et al. [39], and subsequently optimized based on empirical tuning for this application.

Table 2
Hyper parameter settings for GNN and Non-GNN variants.

Category	Parameter	Value
Common Parameters		
	Optimizer	Adam
	Learning Rate	3×10^{-4}
	Discount Factor (γ)	0.96
	Rollout Rate	1×10^{-4}
Architecture Specifics		
Non-GNN	Hidden Layers	4
	Activation	ReLU
GNN-Based	Graph Layers	4
	Heads of Attention	8
	Node Embedding Dim	128
	Edge Embedding Dim	64
Algorithm Specifics		
PPO Only	Clip Ratio (ϵ)	0.25
	PPO epochs	4
	Update batch size	4
	Experience buffer size	8
SAC Only	Entropy Coefficient (α)	0.2
	Replay Buffer Size	50,000
	Soft Update (τ)	0.005

5. Case study

This research was conducted in collaboration with a major European airline. The case study encompasses the exhaustive list of tasks required for scheduled A-check maintenance across a fleet of 30+ aircraft of the same aircraft type. A planning horizon of one year (365 days) was established for the maintenance program, serving as the foundational basis for all experimental scenarios and baseline comparisons. Within a 365-day planning horizon for a single aircraft, the set of maintenance tasks comprises approximately 400 distinct tasks. Accounting for repetitions, the total task instance count is between 500 and 700. The specified intervals for these maintenance tasks exhibit significant variance, spanning a frequency spectrum from 90 days to greater than 270 days. To ensure operational relevance, validation was performed at the fleet level, encompassing all 30+ aircraft. This scope aligns with the standard industry practice for airline maintenance planning.

Table 3
Comparison of reward parameters (Non-GNN vs. GNN).

Non-GNN model			GNN model		
Name	Symbol	Value	Name	Symbol	Value
Spillage threshold	T_{spill}	20%	Spillage threshold	T_{spill}	20%
Completion threshold	$T_{complete}$	90%	Completion threshold	$T_{complete}$	90%
Empty-slot penalty	P_{empty}	0.1	Empty-slot penalty	P_{empty}	0.5
Spillage reward weight	α_1	1	Spillage reward weight	α_2	1.5
Panel-reuse reward weight	β_1	0.5	Panel-reuse reward weight	β_2	0.3
Task-grouping reward weight	γ_1	1	Task-grouping reward weight	γ_2	0.5
DO-task penalty weight	θ_1	−2.0	DO-task penalty weight	θ_2	1
Variance penalty weight	ω_1	0.1	Variance penalty weight	ω_2	0.1
Completion reward weight	–	–	Completion reward weight	η_2	20

5.1. Reward parameters

Table 3 details the parameters utilized to generate rewards for both the Non-GNN and GNN models. These values were empirically determined to optimize the model for its best performance. The difference in environment design cause their different parameters to achieve best performance.

5.2. Industry baseline

The industry baseline is derived from a major international airline currently employing a block maintenance strategy. Their operations typically schedule four A-check slots per aircraft annually, resulting in an average task spillage of approximately 29%. We note that historical metrics regarding Drop-Out (DO) tasks and panel reuse percentages were unavailable for this dataset.

6. Hypotheses

Based on the described methodologies, the following hypotheses are posited for experimental validation.

- **H1:** Within a limited processing time (e.g. 5 hr), the DRL framework will produce solutions of comparable quality to the MILP model, particularly for large-scale problem instances.
- **H2:** The hybrid GNN-DRL framework will converge faster than the stand-alone DRL approach. By explicitly encoding relational information between tasks, the GNN will enable the agent to learn a more effective policy, resulting in satisfying final schedule quality while using significantly less training time.
- **H3:** By leveraging task dependencies, the GNN-integrated architecture learns a robust policy that reflects the underlying problem structure. Conversely, standard DRL models often fail to capture these complex relationships, resulting in myopic or superficial strategies.

7. Experimental setup

All experiments were performed on a GPU server equipped with 96 GB of VRAM. For the experimental procedure, each model was evaluated on a dataset consisting of a 128-instance training partition and a 32-instance validation partition. A critical initial parameter, the due date of each task, was randomized for all instances prior to training. We begin by outlining the evaluation metrics in Section 7.1, followed by a detailed description of the MILP baseline model in Section 7.2. For GNN + SAC model, validation is set for deterministic to achieve better performance.

7.1. Evaluation metrics

In consultation with the major European airline, the following Key Performance Indicators (KPIs) have been identified and prioritized for the evaluation of maintenance schedules:

1. **Spillage:** Defined as the premature execution of a maintenance task relative to its specified interval. It is quantified as the percentage of the maintenance interval that is effectively lost due to early execution of the task. For example 10 days in advance of a 90 day interval task, has a spillage of 11%. Minimizing spillage is the primary objective, as it directly impacts the frequency of task execution and consequently operational costs.
2. **Panel Reuse:** Many maintenance tasks require the removal of access panels, a time-consuming step that contributes significantly to the technician's workload. By grouping tasks that require access to the same panels within a maintenance slot, the cumulative time spent on panel removal and re-installation can be substantially reduced. This metric is evaluated as the percentage of accessed panels that are shared by more than one task. This is the secondary objective. For reference, opening up a panel can cost from 5 min to 1 h. Note that we assume all panels require the same time to open.
3. **Workload Balancing:** This metric aims to distribute the total maintenance workload as evenly as possible across all available maintenance slots. A balanced workload facilitates more efficient and predictable workforce planning. This metric is evaluated by the range of slot labour and constitutes the tertiary objective.

Among these three metrics, the highest priority is assigned to **Spillage** then **Panel Reuse**, as they have the most direct and significant impact on overall maintenance costs.

7.2. MILP baseline

This section explains the Multi-objective Mixed-Integer Linear Programming (MILP) baseline model. The model's primary goal is to assign a set of n maintenance tasks to a fixed number of b time-limited bins (slots). This assignment is optimized against three, often competing, objectives:

1. **Spillage:** To minimize the "waste time" between a task's individual deadline and the effective deadline of the slot it is placed in (which is set by the task with the earliest deadline in that bin).
2. **Panel Reuse:** To minimize the total number of distinct access panels that must be opened. This encourages grouping tasks that use the same panels into the same bin.
3. **Workload balancing:** To balance the workload across all bins, minimizing the deviation in total labour-hours from the fleet-wide average.

A weighted-sum approach is used to combine these three objectives into a single objective function, allowing for a trade-off based on the relative importance assigned to each goal. The weights correspond to the priorities discussed in Section 7.1: $w_1 = 0.5$, $w_2 = 0.3$, and $w_3 = 0.2$. The objective categories overlap with those represented in the GNN-SAC reward, so the resulting operational KPIs can be compared. However, the terms are aggregated and scaled differently in the two

Table 4
Model nomenclature.

Parameters	
n	Total number of tasks (indexed by $i = 1, \dots, n$)
b	Total number of bins (indexed by $j = 1, \dots, b$)
K	Total number of distinct panels (indexed by $k = 1, \dots, K$)
d_i	Duration (in hours) required for task i
D_i	Deadline (in hours) for task i
L_i	Labour-hours required for task i
P_{ik}	Binary parameter: 1 if task i requires panel k, 0 otherwise
M	A large constant (e.g., $M = \max(D_i) + 1$)
$\bar{S}$	Average labour per bin, calculated as $\left(\sum_{i=1}^{n} L_i\right)/b$
w_1, w_2, w_3	Non-negative weights for the three objective terms (spillage, panels, variance)
Decision variables	
$x_{ij} \in \{0, 1\}$	Binary variable: 1 if task i is assigned to bin j, 0 otherwise
$m_j \geq 0$	Continuous variable: The minimum deadline of all tasks assigned to bin j
$z_{jk} \in \{0, 1\}$	Binary variable: 1 if panel k must be opened for bin j, 0 otherwise
$S_j \geq 0$	Continuous variable: The total labour-hours assigned to bin j
$v_j \geq 0$	Continuous variable: The absolute deviation of bin j's labour from the mean $\bar{S}$

formulations; their scalar objective values and optimality gaps are therefore not directly comparable. The model's parameters and decision variables are defined in Table 4.

We establish a fixed total number of bins to guarantee a feasible solution. Since this baseline relies solely on static attributes such as intervals, required panels, and labour estimates, the task initialization is unnecessary. Furthermore, the model does not explicitly penalize Drop-Out (DO) tasks. Instead, any bin containing a single task is treated as an DO event, mirroring the logic applied in the DRL models.

The complete MILP is formulated as follows:

Objective function. The objective is to minimize the weighted sum of the three objectives:

$$\min \quad (w_1 \cdot O_{\text{waste}} + w_2 \cdot O_{\text{panels}} + w_3 \cdot O_{\text{variance}}) \tag{15}$$

Where the components are defined as:

$$O_{\text{waste}} = \sum_{i=1}^{n} \sum_{j=1}^{b} x_{ij}(D_i - m_j) \quad \text{(Total Spillage)}$$

$$O_{\text{panels}} = \sum_{j=1}^{b} \sum_{k=1}^{K} z_{jk} \quad \text{(Total distinct panels)}$$

$$O_{\text{variance}} = \sum_{j=1}^{b} v_j \quad \text{(Total labour variance)}$$

Constraints. The minimization of the objective function is subject to the following constraints:

$$\sum_{j=1}^{b} x_{ij} = 1 \qquad \forall i = 1, \dots, n \tag{16}$$

$$m_j \leq D_i + M \cdot (1 - x_{ij}) \qquad \forall i, j \tag{17}$$

$$z_{jk} \geq x_{ij} \cdot P_{ik} \qquad \forall i, j, k \tag{18}$$

$$S_j = \sum_{i=1}^{n} L_i \cdot x_{ij} \qquad \forall j = 1, \dots, b \tag{19}$$

$$v_j \geq S_j - \bar{S} \qquad \forall j = 1, \dots, b \tag{20}$$

$$v_j \geq \bar{S} - S_j \qquad \forall j = 1, \dots, b \tag{21}$$

Constraint explanations.

- **Constraint** (16): Ensures every task is assigned to exactly one bin.
- **Constraint** (17): The "Big-M" formulation that sets m_j to be less than or equal to the deadline D_i of any task i assigned to bin j. The objective function's minimization of $(D_i - m_j)$ implicitly pushes m_j to equal the minimum D_i among all tasks in that bin.
- **Constraint** (18): The panel activation constraint. If a task i in bin j ($x_{ij} = 1$) requires panel k ($P_{ik} = 1$), then the panel variable z_{jk} is forced to 1.
- **Constraint** (19): Defines the total labour S_j for bin j as the sum of labour-hours of all tasks assigned to it.
- **Constraints** (20) & (21): These two constraints linearize the absolute value function $v_j = |S_j - \bar{S}|$. Since the objective function minimizes v_j, it will be driven to the smallest possible non-negative value that satisfies both, which is precisely the absolute difference.

8. Results

This section details the experimental framework for evaluating the proposed model. We begin by presenting training results of three DRL algorithms in two settings (with or without GNN integration), resulting in six cases in total, and conduct a KPI comparative analysis on the models that are successfully trained. In this section, all the results are produced from 1 aircraft. In a stepwise manner, we demonstrate the superior performance of the GNN-integrated methods, showcasing that these models consistently outperform the other methods. Section 8.1 presents the scheduling results, specifically identifying which models yielded feasible solutions. Subsequently, Section 8.2 provides a comparative analysis of the models based on the selected evaluation metrics.

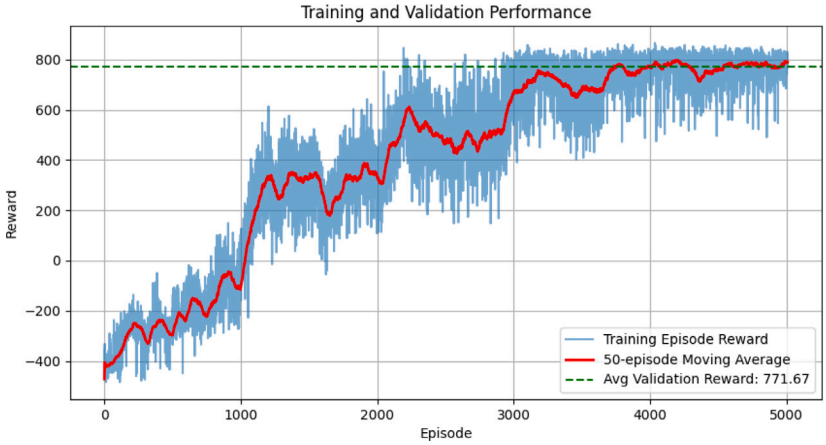
8.1. Scheduling results

The initial evaluation assessed each model's ability to schedule all tasks (approximately 350 unique tasks, or 655 in total when including repetitions) within the given planning horizon. As detailed in Table 5, only three out of the six models evaluated (i.e. A3C, GNN+SAC and GNN+PPO) could successfully generate a feasible schedule(Noted as Pass, otherwise as Fail). Consequently, only these three successful models were included in the subsequent comparison of evaluation metrics.

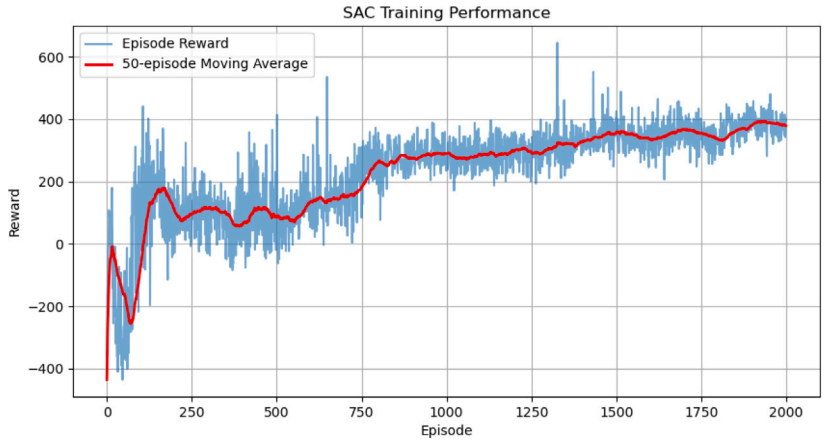
Model convergence is characterized by a plateau in episodic or epoch-based rewards, when rewards difference between episodes or epochs is less than 10. A schedule is considered feasible when all maintenance tasks, within the planning horizon, are allocated to specific maintenance slots, subject to a capacity constraint of fewer than 12 slots per aircraft per year. This constraint ensures that hangar capacity remains sufficient, and the threshold was established in consultation with the major European airline. It was tailored for the dataset used in this study, which covers a one-year planning period for a fleet of 30+ aircraft.

Environment steps are reported to enable a unified comparison across episode- and epoch-based schedules. For epoch-based methods one epoch corresponds to 512 episodes (128 instances × 4 passes); steps are approximate because per-episode trajectory length varies with the number of slots constructed.

Regarding the convergence (Fig. 7), the graph displays only two algorithms. This is due to a fundamental difference between our episode-based and epoch-based training methodologies. To better exploit the computational power of our 96 GB VRAM GPU, we designed some algorithms to be epoch-based. This enabled us to process a very large sample size of 128 instances, training them 4 times per epoch, equivalent to 512 episodes of training within that single epoch. In short,an epoch contains multiple episodes. While highly efficient, this epoch-based method generates data points far less frequently than a standard episode-based algorithm. As a result, the convergence plots for these epoch-based models are less granular and do not offer the same smooth visualization as their episode-based counterparts.



(a) A3C Convergence Graph



(b) SAC Convergence Graph

Fig. 7. Convergence graph of episode based algorithm.

Table 5
Scheduling results of the models.

Model	Scheduling results	Episodes/epochs	Approx. env. steps	Training time
A3C	Pass	5000	2.0M	5 hr
SAC	Fail	2000	0.8M	6 hr
PPO	Fail	14 epochs	2.9M	6 hr
GNN+A3C	Fail	2000	0.8M	55 min
GNN+SAC	Pass	8 epochs	1.6M	40 min
GNN+PPO	Pass	7 epochs	1.4M	38 min

8.2. Evaluation metrics comparison

In post-scheduling, tasks are classified into two categories: optimally grouped tasks and Drop-Out (DO) tasks. A primary objective of the scheduling models is to minimize the number of DO tasks. This objective is balanced against a critical operational constraint: the number of maintenance slots must be limited due to finite hangar capacity.

Fig. 8 presents the number of maintenance slots allocated and the resulting count of DO tasks for each of the three models. The averages are computed from the validation dataset of 32 instances. All three models utilized the maximum available slots (12 per year), ensuring feasibility of their schedules. With a fleet of 30+ aircraft, a maximum of 12 slots per aircraft per year is a safe number to avoid running into hanger capacity issues. In terms of performance, all models achieved a low incidence of DO tasks, a notable outcome considering approximately 655 tasks were scheduled over the period. However, the GNN+SAC model resulted in a marginally higher number of DO tasks compared to both A3C and GNN+PPO, in a sense of assigning more than 600 tasks in total. While additional DO tasks can reduce the spillage, they increase the workload for maintenance planner, who must schedule them promptly.

Further analysis, shown in Fig. 9, compares the models in terms of average spillage and panel reuse percentage. The GNN+SAC model demonstrates the lowest spillage (10.65%), followed by A3C (15.48%), while GNN+PPO exhibits the highest spillage at 22.22%. We also observe that models achieving lower spillage rates tend to exhibit a higher number of DO tasks (c.f. Fig. 8). This inverse relationship indicates that minimizing spillage and managing DO tasks are conflicting objectives. In contrast, all three models achieved comparable panel reuse rates of approximately 13%. This indicates, on average, that 13% of the unique panels in a maintenance slot are reused more than once.

Finally, Table 6 details the variability in labour hours for each model's schedule. This variability is quantified using the range defined as the difference between the maximum and minimum labour hours for model's schedule. The GNN+PPO model exhibits the largest range at 258.62 h, followed by GNN+SAC at 231.8 h, and A3C at 197.9 h. We also examine the composition of the allocated slots to evaluate the degree of similarity between the solutions generated by different models. Due to the stochastic nature of the dataset, distinct DRL models demonstrate a tendency to converge upon different local optima.

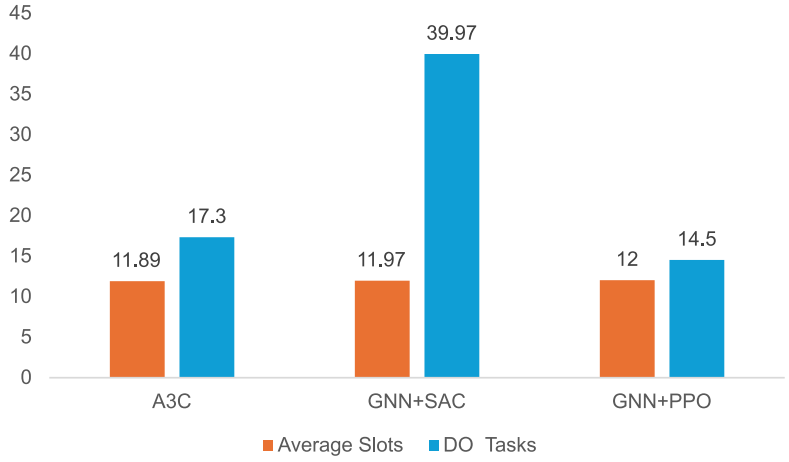


Fig. 8. Slot count vs. DO tasks: single-aircraft training results.

Table 6
Slot labour range: Single-aircraft training results.

Model	Slot Labour Range (hrs)
A3C	197.90
GNN+SAC	231.80
GNN+PPO	258.62

9. Validation

In this section, we validate the proposed GNN-SAC model from several complementary angles. Section 9.1 presents an illustrative comparison with an exactly solved MILP toy instance. Section 9.2 analyses sensitivity to the reward-weight configuration. Section 9.3 examines the effect of removing task-edge message passing. We subsequently scale the evaluation to the fleet level: Section 9.4 reports a fixed-time benchmark against the MILP formulation, Section 9.5 establishes stability across multiple random seeds before conflict-solver post-processing, Section 9.6 benchmarks performance under realistic industry-practice initial conditions, Section 9.7 assesses the quality of the generated maintenance slots, and Section 9.8 reports the stage-wise effect of fleet-level post-processing.

9.1. Illustrative comparison with an exactly solved MILP toy instance

The preceding analyses evaluate the proposed DRL and GNN-DRL models in a large-scale scenario where obtaining an exact mathematical optimum is computationally intractable. We construct a small-scale, simplified task network (40 tasks with 8 slots to force grouping) on which the MILP formulation reaches a 0% solver gap within 30 min. We then run GNN-SAC on the same instance and report the operational metrics separately. Although the two formulations express their objectives differently, i.e. the MILP uses a weighted objective function, whereas GNN-SAC represents the objectives through its reward function, they solve the same task-grouping problem and pursue the same principal operational goals: low interval spillage, high panel reuse, balanced workload, and efficient task grouping. They also use the same task data and eight-slot budget, require all tasks to be assigned, and date each slot at the earliest due date of its constituent tasks, so no task is executed late. The resulting schedules can therefore be compared directly using the common KPIs of slot count, DO tasks, average spillage, panel reuse, and workload distribution. The 0% solver gap applies specifically to the MILP objective, but the KPI comparison remains meaningful because both methods are evaluated on the same instance and operational criteria. The comparative performance is summarized in Table 7.

As shown in Table 7, both methods produce 8 slots and 0 DO tasks in this illustrative instance. Because the same input data, slot budget, feasibility requirements, and KPI definitions are used, their spillage, panel reuse, and solution characteristics provide a meaningful direct comparison. The small differences in how the objectives are represented do not prevent comparison of the final schedules. The results therefore demonstrate the computational efficiency of GNN-SAC while showing that it produces a solution of comparable operational quality to the exactly solved MILP result.

9.2. Reward function sensitivity analysis

Since the multi-objective reward weights are initialized empirically, we perform a sensitivity analysis to evaluate the policy's robustness under different reward configurations, focusing primarily on the two conflicting cost-driven objectives of spillage and panel reuse:

1. **Balanced (Default):** Uses standard weights balancing spillage minimization, panel sharing, labour load, and slot count.
2. **Spillage Focus:** Redefines weights to prioritize interval spillage reduction (α_2 scaling factor doubled).
3. **Panel Sharing Focus:** Prioritizes access panel reuse (β_2 scaling factor doubled).

The resulting trade-offs are summarized in Table 8 and visualized in Figs. 10 and 11.

Table 8 reveals a clear, mathematically sound trade-off between competing maintenance objectives:

- Prioritizing spillage (Spillage Focus) drops the spillage to its lowest value of 9.80%, but at the cost of a higher slot count (12.00) and an increase in deferred opportunity (DO) tasks (50.00) due to over-frequent scheduling. Please note that this change seems trivial is due to the safety constraints that leaves very little room to prioritize spillage.
- Prioritizing panel sharing (Panel Sharing Focus) successfully reduces slots to 8.40 and more than doubles panel reuse to 30.16%, but results in a significant spillage penalty of 20.75% as tasks are delayed to share setup times.
- The Balanced configuration successfully identifies a stable operational equilibrium, maintaining low spillage (10.65%) and low slots while avoiding severe objective degradation.

This sensitivity analysis confirms that the GNN-SAC framework successfully responds to reward weight shifts, allowing airlines to easily tailor the policy to their current operational priorities.

Table 7
Illustrative comparison on the small-scale maintenance toy problem. The MILP reaches a 0% gap for its formulation, while both methods are compared using the same operational KPIs.

Model	Average slots	DO tasks	Average spillage (%)	Panel reuse (%)	Solving time (min)
MILP (0% solver gap)	8.00	0.0	17.96%	50.83%	12
GNN-SAC (Ours)	8.00	0.0	18.54%	47.50%	9

Table 8
Multi-objective sensitivity of GNN-SAC to reward weight configurations.

Configuration	Average slots	DO tasks	Average spillage (%)	Panel reuse (%)
Balanced (Default)	11.97	39.97	10.65%	13.22%
Spillage Focus	12.00	50.00	9.80%	10.56%
Panel Sharing Focus	8.40	15.60	20.75%	30.16%

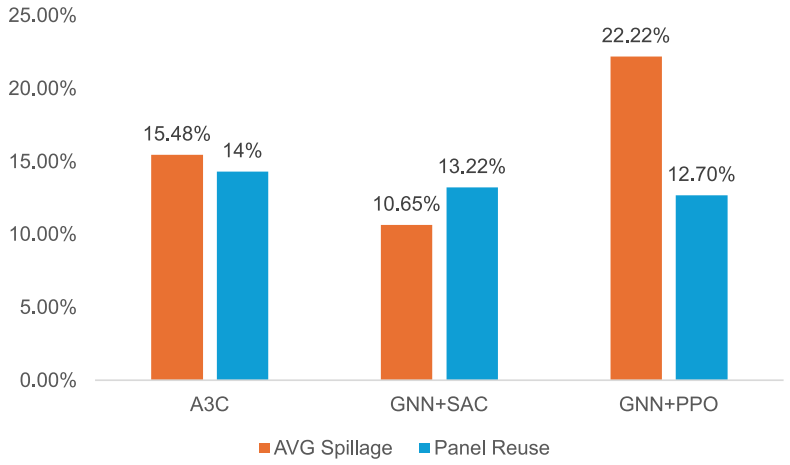


Fig. 9. Spillage vs. panel reuse: Single-aircraft training results.

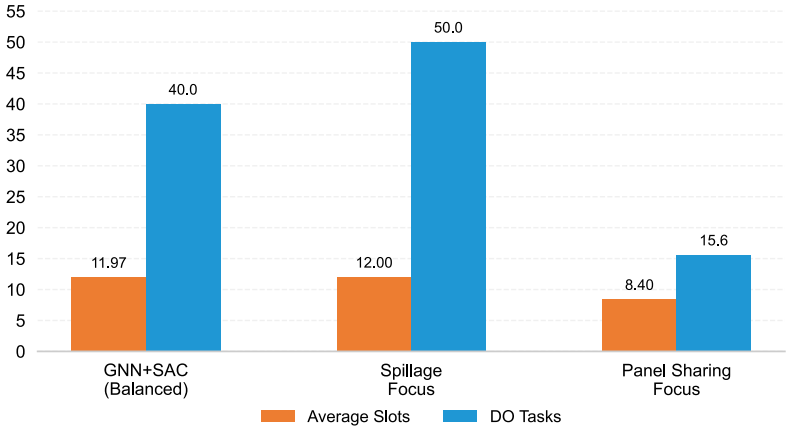


Fig. 10. Reward function sensitivity analysis: Average slots vs. DO tasks.

Note that the final adopted weights were obtained empirically and in cooperation with the use case airline. Other use-cases may justify a different weight balance according to preferences.

9.3. Ablation study: Effect of task-edge message passing

We conduct a controlled ablation under identical state formulation, action space, reward parameters, and training configuration. The experiment isolates task-edge message passing in the encoder; it does not establish superiority on every scheduling objective or isolate every element of the graph design. We evaluate GNN-SAC under two topologies:

1. **50-NN task graph:** The reference graph with directed temporal-neighbour edges, enabling encoder message passing.
2. **Edge-free graph:** The ablated setup with an empty edge index. All task-to-task *and* depot edges are removed. Formally, $\mathcal{N}(i) = \emptyset$, so the message term in Eq. (10) equals 0 and the encoder reduces to its node-wise input transformation plus residual identity path. Thus, this condition cuts off all encoder-side message passing (task-to-task and depot) while leaving the rest of the reported architecture and experimental setting unchanged.

The comparative results are summarized in Table 9.

Table 9 shows a trade-off rather than a uniform advantage. Removing task-edge messages increases average spillage from 10.65% to 16.07%, slightly reduces panel reuse from 13.22% to 12.47%, which are negative outcomes. However, one positive outcome is the reduction of DO tasks from 39.97 to 20.70. Reducing the primary objectives of average spillage and panel reuse has allowed for prioritization of DO

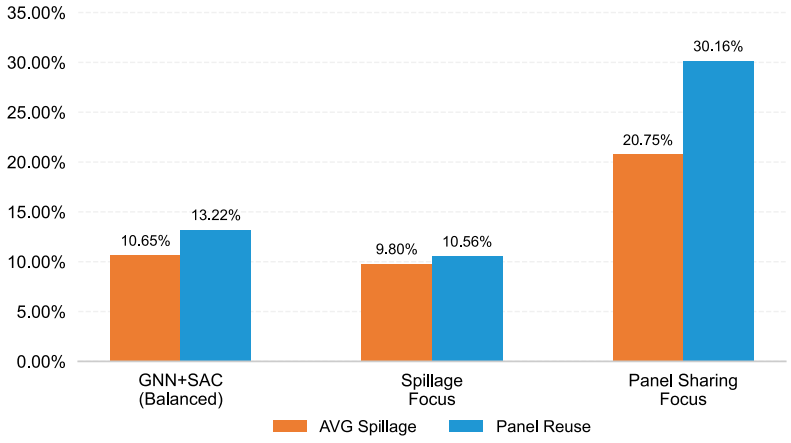


Fig. 11. Reward function sensitivity analysis: Average spillage (%) vs. panel reuse (%).

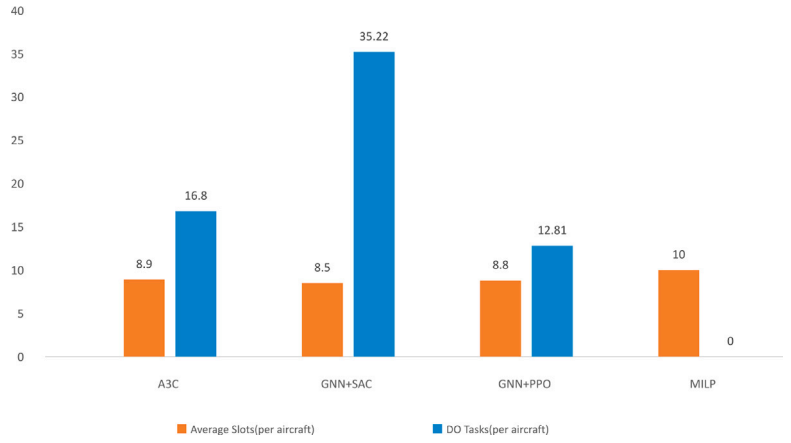


Fig. 12. Slot count vs. DO tasks — fleet-level fixed-time benchmark.

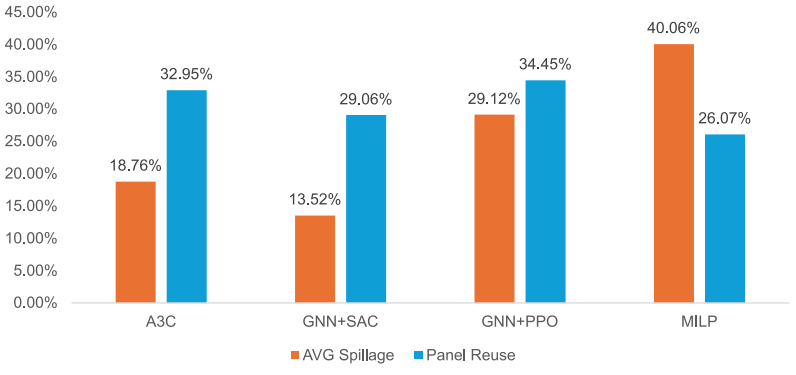


Fig. 13. Spillage vs. panel reuse — fleet-level fixed-time benchmark.

Table 9
Ablation of task-edge message passing. All non-edge settings are held fixed.

Topology	Average slots	DO tasks	Average spillage (%)	Panel reuse (%)
50-NN task graph	11.97	39.97	10.65%	13.22%
Edge-free graph	11.92	20.70	16.07%	12.47%

Table 10
Slot labour range — fleet-level fixed-time benchmark.

Model	Slot Labour Range (hrs)
A3C	186.5
GNN+SAC	225.7
GNN+PPO	249.5
MILP	88.4

tasks. In conclusion, task-edge message passing allows for improvement in two main objectives. However, the reduction of the number of DO tasks, is a welcome surprise.

9.4. Large-scale fixed-time fleet benchmark

This section compares the three proposed reinforcement-learning models with a benchmarking MILP formulation in a large-scale scenario. The MILP was allowed a maximum runtime of five hours and ended with a 102% optimality gap. It is therefore treated as a fixed-time benchmark, not as an exact optimum. The MILP uses a static task-assignment formulation with a fixed number of bins, whereas the DRL policy sequentially constructs slots from the same fixed-horizon task snapshot. Slot count, DO tasks, spillage, panel reuse, and runtime are comparable solution characteristics; they do not imply a common objective value or general superiority of one formulation. All initial task statuses are randomized before validation starts.

Fig. 12 illustrates the average number of maintenance slots utilized and the resulting Drop-Out (DO) tasks per aircraft for each model. The three reinforcement learning models allocated an average of 8-9 slots, a slight reduction compared to the single-aircraft scenario, which is attributable to the conflict solver. The A3C model generated 16.8 DO tasks per aircraft, while the GNN+SAC and GNN+PPO models generated 35.22 and 12.81 DO tasks per aircraft, respectively. In comparison, the MILP benchmark utilized 10 slots and generated zero DO tasks.

A comparison of average spillage and panel reuse percentage is provided in Fig. 13. All three reinforcement learning models exhibit a marginal increase in spillage relative to the single-aircraft case. However, they show a substantial increase in panel reuse percentage, an improvement driven by the conflict solver. The MILP model achieved 40.06% spillage and 26.07% panel reuse. These values are reported as solution characteristics of formulations with different objectives and constraints, not as a general performance ranking.

Finally, Table 10 summarizes the average range of labour hours per aircraft. While the conflict solver contributed to a slight reduction in this range for the reinforcement learning models, their labour hour variability remains considerably higher than that of the MILP model.

In this fixed-time comparison, the learned policies generate feasible schedules more quickly than the five-hour MILP run. The results support their use for rapid scenario generation in decision support.

9.5. Statistical significance and stability analysis

Building on the fleet-level results of Section 9.4, we evaluate the raw GNN-SAC policy across three random seeds (42, 43, and 44), before any fleet-level conflict-solver post-processing. We focus on spillage and access-panel reuse. The results are plotted in Figs. 14 and 15, and the aggregated statistical indicators are presented in Table 11.

As detailed in Table 11, the GNN-SAC policy exhibits exceptionally low variance across independent initializations. The standard deviation for the average slot count is only 0.04 (representing a coefficient of variation of 0.3%), while average spillage and panel reuse exhibit low standard deviations of 0.35% and 0.98%, respectively. The tight confidence intervals around the overall mean suggest high policy stability across independent training initializations. This minimal variance confirms that the GNN-SAC model's fleet-level scheduling capability is highly robust to stochastic training dynamics.

Table 11
Statistical robustness and uncertainty analysis of the raw GNN-SAC policy over multiple random seeds, before conflict-solver post-processing.

Metric	Mean	Standard deviation (SD)	95% confidence interval
Average Slots	11.92	0.04	[11.83, 12.00]
DO Tasks	43.10	2.36	[37.24, 48.96]
Average Spillage	15.00%	0.35%	[14.14%, 15.86%]
Panel Reuse	26.89%	0.98%	[24.46%, 29.31%]

Table 12
Slot labour range — fleet level (industry practice).

Model	Slot Labour Range (hrs)
A3C	274.6
GNN+SAC	289.5
GNN+PPO	315.6
MILP	102.4
Industry	66.7

9.6. Industry practice

The preceding analyses utilized randomly generated initial task statuses to ensure a large and stochastically independent dataset. However, this approach deviates from operational reality, where maintenance tasks within a block system often exhibit temporal interdependencies. Historically, tasks were executed using a block system. Consequently, the current due dates inherently reflect this structure, appearing in clustered groups rather than being evenly distributed. The complex nature of these correlations makes them difficult to model directly. Therefore, this section evaluates the models' performance, trained on randomized data, when applied to a validation set with realistic initial statuses. The performance of the RL models is benchmarked against the same validation data consisting of a snapshot of initial task statuses extracted directly from the industry's operational database.

Fig. 16 presents the average slot usage and the count of DO tasks. When using the provided initial statuses, there was a marginal reduction in slot usage across all models. This behaviour suggests that, because all models successfully met the slot capacity constraints, there was no algorithmic incentive to further minimize slot usage. Consequently, the optimization focus shifted to other objectives. Notably, the GNN-integrated models utilized this stability to significantly reduce the number of DO tasks.

An analysis of efficiency metrics is shown in Fig. 17. Substantial performance shifts are observed. While the A3C (converged) model resulted in minor improvements in spillage and panel reuse, the GNN-integrated models demonstrated dramatic enhancements. The spillage for GNN + SAC decreased from 13.52% to 5.42%, and for GNN+PPO, it dropped from 29.12% (a value close to the industry baseline) to just 7.95%. Panel reuse rates also improved considerably for both models: GNN + SAC increased from 29.06% to 33.71%, and GNN+PPO rose from 34.45% to 47.08%.

As shown in Table 12, these improvements in slot and material efficiency coincided with an increase in the labour hour range for all reinforcement learning models, indicating a trade-off between scheduling efficiency and workload variability.

9.7. Slot quality check

Initial observations indicate that the industry baseline utilizes significantly fewer slots, which necessitates task bundling and directly correlates with increased task spillage. To isolate the effect of optimization strategy, the experimental setup was modified by constraining slot availability for the three DRL algorithms (i.e. A3C, GNN+SAC, and GNN+PPO) to match the industry baseline, equivalent to approximately four slots per aircraft per year. This test was designed to ascertain

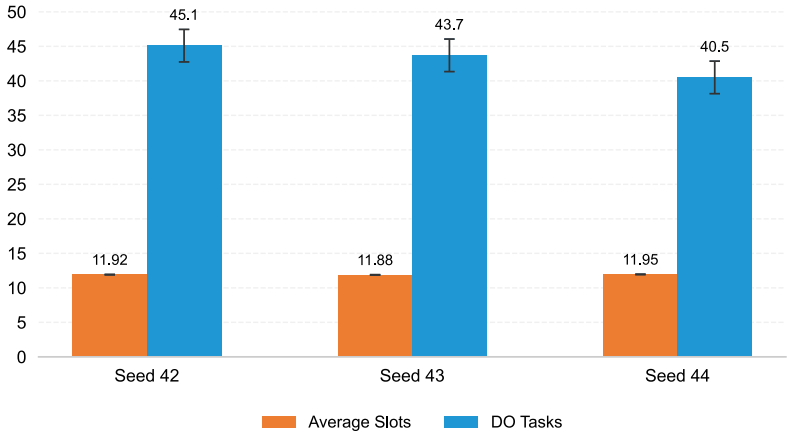


Fig. 14. Statistical robustness over multiple seeds: Average slots vs. DO tasks.

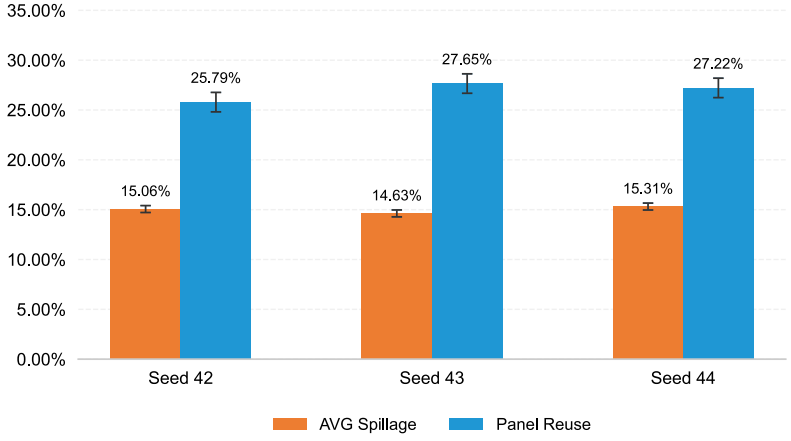


Fig. 15. Statistical robustness over multiple seeds: Average Spillage (%) vs. Panel Reuse (%).

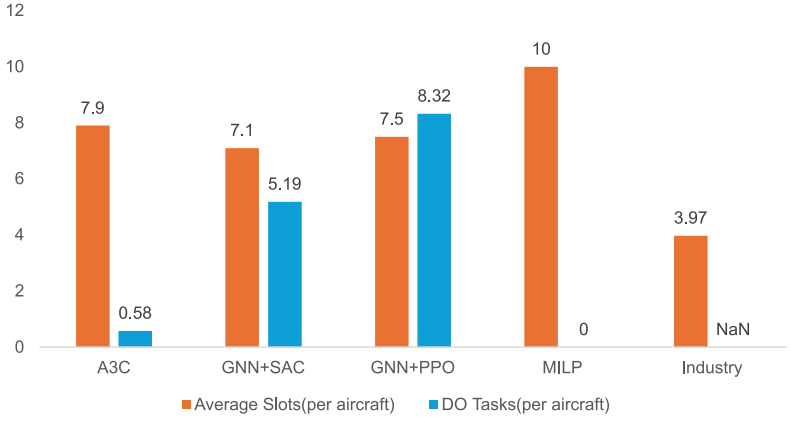


Fig. 16. Slot count vs. DO tasks — fleet level (industry practice).

whether the DRL algorithms primarily optimize task bundling efficiency or simply take advantage of increased slot availability.

The results under this constraint revealed significant divergences in model performance. As illustrated in Fig. 18, the A3C model exhibited no significant change in DO tasks. Conversely, the GNN+SAC model demonstrated a significant reduction in DO tasks, while the GNN+PPO model showed a slight increase. Analysis of spillage in Fig. 19 provided the clearest differentiation: the A3C model experienced a substantial degradation in performance, with spillage increasing from 14.74% to 33.40%. In sharp contrast, the GNN-integrated models displayed only moderate increases: GNN+SAC rose from 5.42% to 10.29%, and GNN+PPO increased from 7.95% to 10.53%. While secondary metrics such as panel reuse exhibited only marginal changes across all models, the updated labour range data confirmed an increase for all models (Table 13).

These findings strongly indicate that the GNN-integrated models (GNN+SAC and GNN+PPO) successfully developed effective strategies for efficient task bundling. Their ability to maintain relatively low spillage, even when slot availability was restricted, highlights this advanced scheduling capability. Conversely, the A3C model, lacking GNN

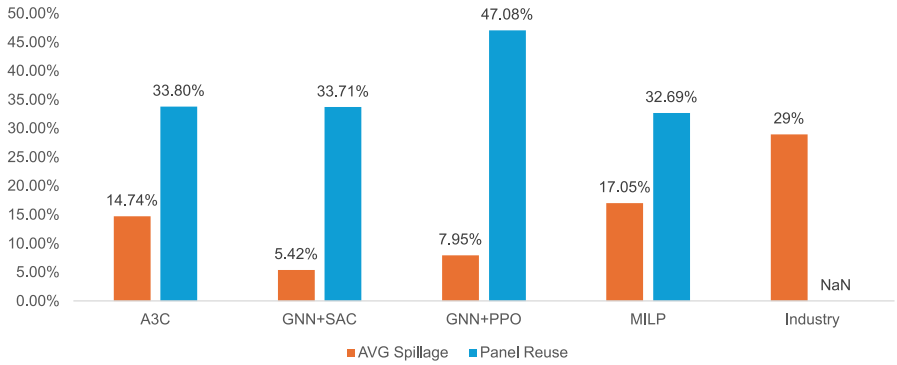


Fig. 17. Spillage vs. panel reuse — fleet level (industry practice).

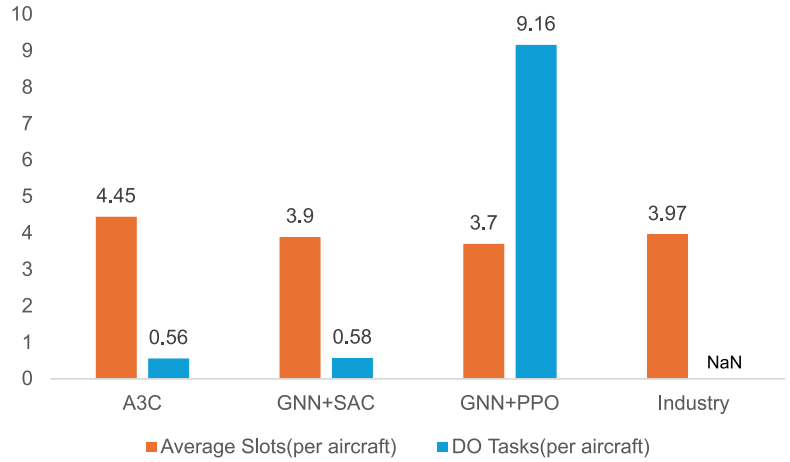


Fig. 18. Slot count vs. DO tasks — fleet level (slot quality).

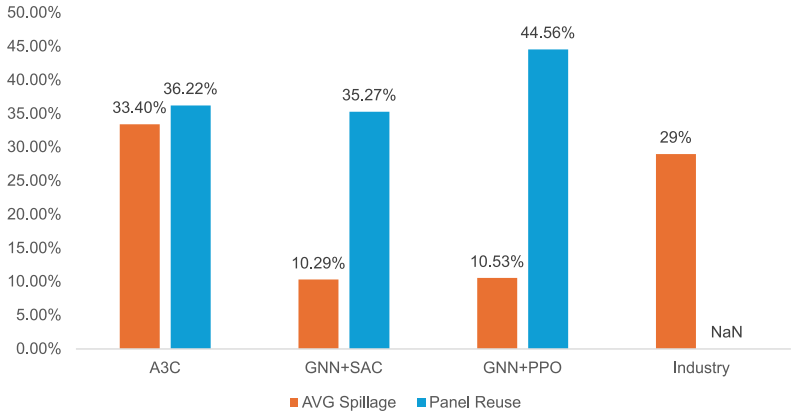


Fig. 19. Spillage vs. panel reuse — fleet level (slot quality).

Table 13
Slot labour range — fleet level (slot quality).

Model	Slot Labour Range (hrs)
A3C	310.8
GNN+SAC	302.6
GNN+PPO	359.4
Industry	66.7

integration, appears heavily reliant on increased slot availability as its primary mechanism for spillage reduction; when this mechanism was inhibited, its inability to optimize task bundling resulted in significantly higher spillage.

9.8. Conflict solver contribution

To quantify the reported effect of fleet-level post-processing, we run a standalone stage-wise analysis using a trained GNN-SAC model, seed 42, and a fleet of 31 aircraft. The analysis is separate from the multi-seed, raw-policy results in Section 9.5. Slot counts and DO-task counts are reported per aircraft; spillage and panel reuse are fleet-level KPIs.

Starting from the raw GNN output, we apply the two post-processing stages of Section 4.3 cumulatively and measure the fleet-level KPIs after

Table 14
Conflict-solver stage-wise analysis (single model, seed 42, 31 aircraft). Stages are applied cumulatively to the same raw GNN output. Slots and DO tasks are averages per aircraft.

Stage	Avg. slots	DO tasks	Spillage (%)	Panel reuse (%)
A: Raw GNN output	11.93	35.22	11.31%	25.06%
B: A + Slot Merging	8.50	35.22	11.89%	26.68%
C: B + Slot Preponing	8.50	35.22	14.38%	26.68%

each, using the same metric definitions as the rest of the paper. The results are reported in Table 14.

For this particular instance, slot merging, as expected, reduces the average number of multi-task slots from 11.93 to 8.50 per aircraft and increases panel reuse by 1.62 percentage points, while DO-task counts remain unchanged. This stage-wise analysis uses the revised solver configuration, which merges slots and moves slots earlier but does not reassign single-task slots. In turn, slot preponing addresses same-day fleet conflicts without changing slot or DO-task counts, but increases spillage from 11.89% to 14.38% as slots are shifted to earlier available dates.

10. Discussion

This section synthesizes the experimental findings, beginning with a confirmation of the hypotheses presented in Section 10.1. Subsequently, an in-depth analysis of the results is performed, exploring the underlying factors that derive the observed model performance and discussing their broader implications. Section 10.2 examines the compatibility challenges between GNN architectures and RL algorithms, while Section 10.3 highlights the specific advantages of GNN integration. Section 10.4 addresses the issue of conflicting objectives and proposes strategies to manage them. Finally, Sections 10.5 and 10.6 present the industrial implications, managerial insights, and directions for future research.

10.1. Hypotheses evaluation

- **H1:** The learned policies generate schedules more quickly than the MILP in the reported five-hour fixed-time benchmark. The MILP ended with a 102% gap, whereas the GNN-integrated models converged within one hour and the A3C model within three hours. For validation, computation time is approximately 10 min. This comparison supports a computational-efficiency advantage under the stated time budget; it does not establish an optimality comparison. This hypothesis is **supported within the fixed-time benchmark**.
- **H2:** GNN-integrated reinforcement learning models exhibit different trade-offs from the non-integrated counterpart in the reported settings. Under realistic initial task statuses, the GNN-integrated models produced favourable results on several reported metrics relative to the A3C baseline. However, in certain scenarios (Section 9.1), the converged A3C model achieved performance comparable to GNN+SAC. This evidence is limited to the evaluated task setting and does not establish overall superiority across objectives. This hypothesis is therefore **partially confirmed**.
- **H3:** GNN-integrated models may better exploit the structured task representation under the reported restricted-slot evaluation. The results in Section 9.7 show that, when slot quantity constraints are added, the GNN-integrated models retain favourable trade-offs on the reported metrics relative to A3C. This evidence does not establish transfer to other task sets, graph structures, aircraft types, or fleet sizes. This hypothesis is **partially confirmed within the evaluated setting**.

10.2. Compatibility of GNN and RL algorithms

The results in Section 8.1 suggest that the performance of a reinforcement learning agent is highly dependent on the compatibility between the algorithm's complexity and the environment's informational richness. A3C, an algorithm characterized by its aggressive policy updates, appears ill-suited for the high-dimensional state–action space introduced by the GNN, leading to policy instability rather than convergence. Conversely, in a simpler environment without the GNN, A3C eventually identifies an effective policy by processing a linear list of tasks, albeit at the cost of losing explicit relational information.

On the other hand, more complex algorithms like SAC and PPO were designed for information-rich environments. In the simpler, non-GNN setting, they failed to converge, likely because the environment lacked sufficient state information to guide their sophisticated exploration mechanisms. The integration of the GNN provides the necessary structured high-dimensional input that these algorithms require, enabling them to develop stable and effective policies.

Based on the validation of the results (Section 9), GNN+SAC is identified as the most promising model for practical implementation. It achieves the lowest spillage while maintaining competitive performance across other objectives. In contrast, competing algorithms necessitate significant trade-offs to achieve similar levels of performance.

10.3. Highlights of GNN architecture

The primary advantage of the GNN architecture lies in its ability to encode the complex inter-task relationships into a structured representation for the agent. This holistic view manifests in three key areas:

- It enhances computational efficiency. Although the converged A3C model eventually achieved comparable results, it required three hours of training. Both GNN-integrated models reached convergence in just one hour, demonstrating a significant reduction in training time. This efficiency gain is likely attributable to the GNN's ability to represent task connections, which implicitly guides the agent's exploration towards higher rewards.
- It can exploit temporal relations in the evaluated task snapshot. In the single-aircraft scenario (Section 8), all three models yielded similar results. Under the realistic initial-status evaluation (Section 9.6), the GNN-integrated models showed improved performance relative to the reported baselines. Noe that this is evidence within the evaluated data setting, not a validation of transfer across task sets, graph structures, aircraft types, or fleet sizes.
- It has a structural capacity to process graph inputs. The encoder is not intrinsically tied to a fixed node count, unlike a fixed-dimension MLP. However, the present experiments use one task setting and fleet configuration; empirical transfer across task-set sizes, graph topologies, and fleet compositions remains future work (Section 10.6).

10.4. Analysis of conflicting objectives

The optimization process involves navigating a complex landscape of competing objectives. The primary metrics considered, i.e. Spillage, Panel Reuse, and Workload Balancing, are often in conflict. For example, minimizing spillage could be trivially achieved by scheduling every task individually (increase of DO tasks), but this would be operationally infeasible. Conversely, grouping tasks to minimize DO tasks and maximize panel reuse can lead to highly unbalanced workloads across maintenance slots, i.e. task due dates are not uniformly distributed.

In this study, priority was given to spillage and panel reuse due to their direct impact on maintenance costs. The resulting imbalance

in workload, reflected in the large labour hour range, was deemed an acceptable trade-off. This decision is contextualized by the one-year planning horizon of the maintenance policy, which is assumed to provide sufficient lead time for airlines to manage workforce allocation effectively. However, the objective prioritization can be readily adapted based on specific operational requirements. The tool's capabilities can be further refined by investigating different reward functions that capture a wider range of operational preferences.

10.5. Industrial implications, deployment, and modelled constraints

The proposed framework addresses a long-standing challenge in the aviation industry: the optimization of rigid and historically derived maintenance schedules. The current paradigm is widely acknowledged to be suboptimal, but the operational complexity of enacting changes has hindered progress. Our models demonstrate not only the tangible benefits of optimization but also provide a flexible tool for simulating and evaluating different policies, thereby reducing the risks in the transition to a more efficient system.

The model is to be deployed in airlines, and run as new maintenance tasks to be performed arise. Note, however, that rewards were validated empirically. Each airline should tune the rewards formulation up to their internal preferences. Finally, the tool is intended as decision-support, leaving final acceptance to maintenance engineers.

This study provides several actionable insights for airline maintenance management. First, at the operational level, the proposed tool enables the optimization of slot planning within existing capacity constraints. As demonstrated in our experiments with fixed slot limits (e.g., four slots), the model can assist planners in maximizing the utility of restricted hangar resources, ensuring efficient task organization even under tight capacity caps. Second, the framework allows airlines to quantitatively evaluate the strategic shift from legacy rigid block systems to more flexible scheduling paradigms, highlighting potential reduction in spillage and improvements in fleet availability. Finally, the model is idealized to function as a robust decision support tool, empowering airlines to simulate various optimization objectives and receive immediate feedback on different scheduling scenarios, thereby facilitating policy design prior to implementation. Further development of the tool, including defence against historical delays and including the forecast of extra necessary labour hours, will further extend this tool as a decision-support mechanism that will help airlines balance costs and robustness.

Modelled timing constraints, failure modes, and mitigation. Aircraft maintenance scheduling is a safety-critical process governed by strict regulatory limits (e.g., FAA and EASA directives). Introducing neural-network-driven policies into such operational environments raises valid concerns regarding constraint satisfaction and predictability. Because the tool is deployed as decision support, with the maintenance engineer retaining final acceptance, these concerns are further addressed through three core system features:

1. **Modelled timing constraint:** Eq. (14) prevents an eligible task from entering an open slot when its initial due-date span would exceed the configured 40-day limit. Completed slots are assigned their earliest planned due date, and fleet preponing only moves them earlier. This establishes no-lateness with respect to the modelled initial due dates in the evaluated fixed-horizon setting; it is not a guarantee for unmodelled labour, disruption, or regulatory constraints.
2. **Inspection via Attention Weights:** The learned attention weights α_{ij}^{l} can be visualized to inspect which temporal task-neighbour relations were emphasized by the encoder. Note that they are a diagnostic aid for planners, not a complete causal explanation of a scheduling decision.
3. **Performance Failure Detection:** A model failure may appear as excessive slots, spillage, or an infeasible schedule under constraints not represented in the current model. These outputs should be reviewed by maintenance engineers before adoption; disruption and out-of-distribution robustness were not experimentally evaluated.

10.6. Future work

We highlight four key directions of improvement for the current model:

- Improving robust scheduling of the model: First, the model should incorporate differential panel types, as removal and reinstallation time vary across different panel categories; different reuse combinations yield different time savings. This granularity should be addressed in the model. Second, the model should be extended to include non-routine tasks, which can significantly impact labour-hour projections and material inventory. Third, the model should also include historical data on delays, workforce prediction and material prediction to enable disruption-aware scheduling, including appropriately sized slot buffers, to mitigate potential disruption to the maintenance.
- Inclusion of additional real-world maintenance costs: As shown in the results, the model prefers the creation of a higher number slots which decreases spillage. Nevertheless, this has the direct consequence of adding more variability to the maintenance work. Additionally, less spillage reduces robustness in the case of disruptions, when maintenance activities have to be postponed. All these extra considerations should be discretized into a cost value and added to the model.
- Integration of this optimization framework into a comprehensive airline operations simulator should be explored: The integration of a comprehensive simulator enables the observation of cross-sectoral impacts resulting from maintenance policy adjustments. This offers airlines substantial value by providing a holistic view of operational changes, allowing them to anticipate outcomes and proactively mitigate risks.
- Formal evaluation of cross-instance generalization and scalability: As discussed in Section 10.3, the Residual-EGAT architecture is structurally not constrained to a fixed task-count, which is a necessary precondition for transfer across problem instances of varying size and topology. Future work should empirically validate this property by testing the trained policy on task sets of different sizes, alternative graph topologies, and different fleet configurations — experiments that are beyond the scope of the current study.

11. Conclusion

This study successfully demonstrates the novel application of Deep Reinforcement Learning (DRL), both as a stand-alone method and in a hybrid framework with Graph Neural Networks (GNNs), to optimize long-term aircraft maintenance policy. By moving beyond the rigid, inefficient block-based paradigm, our approach directly addresses a critical research gap and provides a computationally tractable solution to a large-scale combinatorial problem. The novelty of this work lies in formulating the policy design as a dynamic task-grouping problem and applying advanced machine learning techniques to solve it.

The GNN encoder provides a mechanism for incorporating temporal task-neighbour relations into the DRL policy. In the evaluated industrial case study, the GNN-integrated models produced favourable trade-offs on several reported metrics under realistic initial statuses under the tested task setting and fleet configuration. This framework serves as a robust decision support tool, enabling airlines to evaluate the impact of various maintenance policies under their specific

business constraints. It also facilitates the development of a multi-stage roadmap for gradually transitioning towards a fully optimized maintenance system.

Future research should first explore the quantification of additional maintenance limitations from real-world operations, e.g. maintenance shifts, robustness, cost of moving an aircraft to the hangar. Additionally, the potential integration of this optimization framework into a comprehensive airline operations simulator should be explored. Such an integration would enable a holistic analysis of how dynamically optimized maintenance policies impact other critical factors, such as fleet availability, crew scheduling, and network reliability, thereby providing a unified platform for strategic decision-making.

CRediT authorship contribution statement

Haonan Li: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Álvaro Lanza Rausell:** Validation, Methodology, Formal analysis. **Alireza Amiri-Simkooei:** Writing – review & editing, Supervision, Project administration. **Marta Ribeiro:** Writing – review & editing, Supervision, Resources, Project administration, Funding acquisition, Data curation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

References

[1] de Pater I, Reijns A, Mitici M. Alarm-based predictive maintenance scheduling for aircraft engines with imperfect remaining useful life prognostics. Reliab Eng Syst Saf 2022;221:108341. http://dx.doi.org/10.1016/j.ress.2022.108341.

[2] Costanza D, Prentice B, Poitras M, Bardygula O, Fasano B, Sargent S, Hayes L, Franzoni C. Not Enough Aviation Mechanics - AMT Report. Technical Report, Oliver Wyman; 2022.

[3] Hölzel NB, Schröder C, Schilling T, Gollnick V. A Maintenance Packaging and Scheduling Optimization Method for Future Aircraft. Technical Report, 2012, Air transport and operations symposium.

[4] Filho JNdM, de Mesquita ACP, Abrahão FTM, Rocha GC. An innovative method to solve the maintenance task allocation and packing problem. J Qual Maint Eng 2024;30(1). http://dx.doi.org/10.1108/JQME-08-2023-0069.

[5] Li H, Zuo H, Lei D, Liang K, Lu T. Optimal combination of aircraft maintenance tasks by a novel simplex optimization method. Math Probl Eng 2015;2015. http://dx.doi.org/10.1155/2015/169310.

[6] Tseremoglou I, van Kessel PJ, Santos BF. A comparative study of optimization models for condition-based maintenance scheduling of an aircraft fleet. Aerospace 2023;10(2). http://dx.doi.org/10.3390/aerospace10020120.

[7] Mitici M, de Pater I, Barros A, Zeng Z. Dynamic predictive maintenance for multiple components using data-driven probabilistic RUL prognostics: The case of turbofan engines. Reliab Eng Syst Saf 2023;234. http://dx.doi.org/10.1016/j.ress.2023.109199.

[8] Lee J, Mitici M. Deep reinforcement learning for predictive aircraft maintenance using probabilistic Remaining-Useful-Life prognostics. Reliab Eng Syst Saf 2023;230. http://dx.doi.org/10.1016/j.ress.2022.108908.

[9] Lee J, Mitici M. An integrated assessment of safety and efficiency of aircraft maintenance strategies using agent-based modelling and stochastic Petri nets. Reliab Eng Syst Saf 2020;202. http://dx.doi.org/10.1016/j.ress.2020.107052.

[10] Lee J, Mitici M. Multi-objective design of aircraft maintenance using gaussian process learning and adaptive sampling. Reliab Eng Syst Saf 2022;218. http://dx.doi.org/10.1016/j.ress.2021.108123.

[11] de Pater I, Mitici M. Predictive maintenance for multi-component systems of repairables with Remaining-Useful-Life prognostics and a limited stock of spare components. Reliab Eng Syst Saf 2021;214. http://dx.doi.org/10.1016/j.ress.2021.107761.

[12] Duan S, Sun J, Yu ZQ, Liu SQ. Uncertain data driven predictive maintenance: A cost-oriented implementation method on aircraft system. Reliab Eng Syst Saf 2025;264. http://dx.doi.org/10.1016/j.ress.2025.111278.

[13] Adimonyemma J, Sun Y. Optimization model for large-scale long-term aircraft maintenance scheduling and station assignment. Transp Res Part E: Logist Transp Rev 2025;202:104302. http://dx.doi.org/10.1016/j.tre.2025.104302.

[14] Papakostas N, Papachatzakis P, Xanthakis V, Mourtzis D, Chryssolouris G. An approach to operational aircraft maintenance planning. Decis Support Syst 2010;48(4). http://dx.doi.org/10.1016/j.dss.2009.11.010.

[15] Shaukat S, Katscher M, Wu CL, Delgado F, Larrain H. Aircraft line maintenance scheduling and optimisation. J Air Transp Manag 2020;89. http://dx.doi.org/10.1016/j.jairtraman.2020.101914.

[16] Witteman M, Deng Q, Santos BF. A bin packing approach to solve the aircraft maintenance task allocation problem. European J Oper Res 2021;294(1). http://dx.doi.org/10.1016/j.ejor.2021.01.027.

[17] Tseremoglou I, Santos BF. Condition-based maintenance scheduling of an aircraft fleet under partial observability: A deep reinforcement learning approach. Reliab Eng Syst Saf 2024;241:109582. http://dx.doi.org/10.1016/j.ress.2023.109582.

[18] van Kessel PJ, Freeman FC, Santos BF. Airline maintenance task rescheduling in a disruptive environment. European J Oper Res 2023;308(2). http://dx.doi.org/10.1016/j.ejor.2022.11.017.

[19] Lagos C, Delgado F, Klapp MA. Dynamic optimization for airline maintenance operations. Transp Sci 2020;54(4). http://dx.doi.org/10.1287/TRSC.2020.0984.

[20] Clausen J, Larsen A, Larsen J, Rezanova NJ. Disruption management in the airline industry-Concepts, models and methods. Comput Oper Res 2010;37(5). http://dx.doi.org/10.1016/j.cor.2009.03.027.

[21] Van Buskirk C, Dawant B, Karsai G, Sprinkle J, Szokoli G, Suwanmongkol K, Currer R. Computer-aided Aircraft Maintenance Scheduling. Technical Report, Institute for Software Integrated Systems; 2002.

[22] Steiner A. A heuristic method for aircraft maintenance scheduling under various constraints. In: 6th swiss transport research conference. 2006.

[23] Muchiri AK, Smit K. Application of maintenance interval de-escalation in base maintenance planning optimization. Enterp Risk Manag 2009;1(2). http://dx.doi.org/10.5296/erm.v1i2.179.

[24] Zhang D, Hu Y, Zhang S, Zhang Y. Distributed hierarchical reinforcement learning for dynamic maintenance scheduling of large-scale airline fleets. Reliab Eng Syst Saf 2026;112249. http://dx.doi.org/10.1016/j.ress.2026.112249.

[25] Nazari M, Oroojlooy A, Takáč M, Snyder LV. Reinforcement learning for solving the vehicle routing problem. In: Advances in neural information processing systems, vol. 2018-December, 2018.

[26] Louati A, Lahyani R, Aldaej A, Mellouli R, Nusir M. Mixed integer linear programming models to solve a real-life vehicle routing problem with pickup and delivery. Appl Sci (Switzerland) 2021;11(20). http://dx.doi.org/10.3390/app11209551.

[27] Guo T, Mei Y, Zhang M, Zhao H, Cai K, Du W. Learning-aided neighborhood search for vehicle routing problems. IEEE Trans Pattern Anal Mach Intell 2025;47(7):5930–44. http://dx.doi.org/10.1109/TPAMI.2025.3554669.

[28] Barros-Everett T, Montero E, Rojas-Morales N. Parameter prediction for meta-heuristic algorithms solving routing problem instances using machine learning. Appl Sci 2025;15(6):2946. http://dx.doi.org/10.3390/app15062946.

[29] Mahmudy WF, Widodo AW, Haikal AH. Challenges and opportunities for applying meta-heuristic methods in vehicle routing problems: A review. In: The 7th mechanical engineering, science and technology international conference. Basel Switzerland: MDPI; 2024, p. 12. http://dx.doi.org/10.3390/engproc2024063012.

[30] Bello I, Pham H, Le QV, Norouzi M, Bengio S. Neural combinatorial optimization with reinforcement learning. In: 5th international conference on learning representations, ICLR 2017 - workshop track proceedings. 2017.

[31] Zhu Y, Zheng M, Su Z, Xia T, Lin J, Pan E. Deep reinforcement learning for joint optimization of maintenance and spare parts ordering considering spare parts supply uncertainty. Reliab Eng Syst Saf 2025;264. http://dx.doi.org/10.1016/j.ress.2025.111385.

[32] Takabatake T, Nagashima W, Hasegawa N. Reinforcement learning-based optimization of tsunami evacuation paths: effectiveness and robustness in two coastal areas in Japan. Reliab Eng Syst Saf 2026;266. http://dx.doi.org/10.1016/j.ress.2025.111594.

[33] Yuan J, Lei Y, Li N, Yang B, Li X, Chen Z, Han W. A framework for modeling and optimization of mechanical equipment considering maintenance cost and dynamic reliability via deep reinforcement learning. Reliab Eng Syst Saf 2025;264. http://dx.doi.org/10.1016/j.ress.2025.111424.

[34] Geurtsen M, Leenen C, Adan I, Atan Z. Deep reinforcement learning for optimal planning of production line maintenance with deterioration. Reliab Eng Syst Saf 2026;266. http://dx.doi.org/10.1016/j.ress.2025.111767.

[35] Dong C, Li D, Karimi HR. Reinforcement learning driven adaptive graph construction for fault diagnosis of chemical processes. Reliab Eng Syst Saf 2026;266. http://dx.doi.org/10.1016/j.ress.2025.111781.

[36] Cappart Q, Chételat D, Khalil EB, Lodi A, Morris C, Velickovic P. Combinatorial optimization and reasoning with graph neural networks. In: IJCAI international joint conference on artificial intelligence. 2021, http://dx.doi.org/10.24963/ijcai.2021/595.

[37] Vinyals O, Fortunato M, Jaitly N. Pointer networks. In: Advances in neural information processing systems, vol. 2015-January, 2015.
[38] Kool W, Van Hoof H, Welling M. Attention, learn to solve routing problems! In: 7th international conference on learning representations. 2019.
[39] Lei K, Guo P, Wang Y, Wu X, Zhao W. Solve routing problems with a residual edge-graph attention neural network. Neurocomputing 2022;508. http://dx.doi.org/10.1016/j.neucom.2022.08.005.
[40] Yan D, Ou B, Guan Q, Zhu Z, Cao H. Edge-driven multiple trajectory attention model for vehicle routing problems. Appl Sci 2025;15(5):2679. http://dx.doi.org/10.3390/app15052679.
[41] Chen J, Zhou X. A GNN-boosted reinforcement learning framework for maintenance optimization in multi-dependency manufacturing system. Reliab Eng Syst Saf 2026;269. http://dx.doi.org/10.1016/j.ress.2025.112087.