Towards Autonomous Railway Operations: A Semi-Hierarchical Deep Reinforcement Learning Approach to the Vehicle Rescheduling Problem

Alberto Castagna* Stefan Zahlner* Adrian Egli† Christian Eichenberger ‡
Daniel Boos† Manuel Meyer‡ Anton Fuxjäger*
*enliteAI, AT †SBB CFF FFS, CH ‡Flatland Association, CH
Corresponding author: a.fuxjaeger@enlite.ai

***Abstract*—Managing disruptions in railway traffic management is a major challenge. Rising traffic density and infrastructure limits increase complexity, making the Vehicle Routing and Scheduling Problem (VRSP) difficult to solve reliably and in real time. While Operational Research (OR) methods are widely used, most dispatching still relies on human expertise due to the problem's exponential combinatorial complexity. Reinforcement Learning (RL) has gained attention for its potential in multi-agent coordination, but existing RL approaches often underperform OR methods and struggle to scale in dense rail networks.**

This paper addresses this gap from a machine learning perspective by introducing a semi-hierarchical RL formulation tailored to operational railway constraints. The method separates dispatching from routing through dedicated action and observation spaces, enabling policies to specialise in distinct decision scopes and addressing the imbalance between rare dispatch decisions and frequent routing updates. The approach is evaluated on the Flatland-RL simulator across five difficulty levels and 50 random seeds, with 7 to 80 trains. Results show substantially improved coordination, resource utilisation, and robustness compared with heuristic baselines and monolithic RL, nearly doubling the number of trains reaching their destinations, while keeping deadlock rates below 5% and adaptively sequencing, delaying, or cancelling trains under heavy congestion.

***Index Terms*—Multi-Agent Reinforcement Learning, Railway Traffic Management, Vehicle Routing and Scheduling, Hierarchical Control, Multi-Agent Path Finding.**

I. Introduction

Planning and rescheduling train runs for ever-increasing traffic volumes is a complex task requiring months of expert preparation. During operations, specialists in operations centres monitor traffic flow and respond to disruptions in near real time, but dispatchers must make rapid decisions within short horizons. With rising traffic density and demand for flexibility, current dispatching processes and methods are reaching their limits, leaving dispatchers little time to react.

In systems with more than 10,000 daily journeys, minor disruptions significantly impact service quality and network stability, making dynamic rescheduling as critical as initial timetable planning.

The challenge lies in developing scalable algorithms that shift focus from local dispatch areas to global network solutions. System stability depends on shrinking time buffers, making global optimisation essential since any local adjustment affects the entire network. Artificial Intelligence (AI), particularly Multi-Agent Reinforcement Learning (MARL), offers promising approaches for real-time dispatching and rescheduling. However, scalability and reliability questions in real-world environments remain unsolved.

We study these questions using Flatland-RL, an open-source railway simulator modeling multi-agent train movement with discrete time and cell-level occupancy. The environment captures core railway characteristics including shared infrastructure, bottlenecks, dispatch timing, and routing conflicts. Recent work in this environment shows that existing RL methods remain limited by monolithic decision structures, insufficient dispatch exploration, and poor scalability in dense traffic, motivating architectures that separate decision scopes and reflect real operational structures. This motivates the need for architectures that separate decision scopes and better reflect the operational structure of real rail systems.

To address these limitations, this work introduces *Maze-Flatland*, a semi-hierarchical multi-agent RL framework for the VRSP in dense rail networks. Unlike monolithic policies, Maze-Flatland separates control into two coordinated levels: dispatching (Multi-Agent Departure Scheduling (**MADS**)) and routing (Multi-Agent Path Finding (**MAPF**)). This decomposition reduces task interference and exploration imbalance, improving scalability, coordination, and learning stability. Its performance is benchmarked against heuristic baselines, including Prioritized Planning (PP) [1], and learning-based methods such as TreeLSTM [2], showing substantial gains in coordination efficiency and deadlock reduction. A train is considered deadlocked if it can not move for rest of the episode.

The main contributions are: (i) analysis of RL approaches in Flatland-RL highlighting scalability limits; (ii) a novel semi-hierarchical two-phase RL approach for VRSP in railway; and (iii) demonstration that this solution scales effectively with traffic density, enhancing performance and robustness.

The remainder of this paper is organised as follows. Section II introduces the VRSP in railway operations. Section III reviews state-of-the-art approaches. Section IV analyses agent behaviour and motivates semi-hierarchical design. Section V

presents the proposed framework with control and observation spaces. Section VI describes simulation setup and metrics. Section VII reports results, Section VIII provides ablation analysis, and Section IX summarises contributions and future directions.

II. Problem Formulation

We consider the VRSP in railway operations as the real-time process of rescheduling train movements in response to disruptions [3] such as delays, failures, or resource shortages. When the current state diverges from the initial plan, replanning is triggered and must be performed within a short time window [4] to selectively revise affected train schedules while minimising the impact on the system.

This study focuses exclusively on operational disruptions such as train malfunctions, limiting the problem scope by assuming no shortages of staff or rolling stock. Although real-world operations involve multiple optimisation objectives, here we prioritise the arrival ratio of the available trains while maintaining deadlock-free operation. Minimising total delay is desirable but not the primary target.

The procedure in real-world railway operations is twofold: first, operators generate an optimal baseline timetable and allocate resources for all train categories over a shared time window; second, they must adapt this timetable in real time by reassigning routes and resources whenever operational disruptions occur. In this work, we specifically address the second phase, which is triggered when disruptions require immediate timetable and resource adjustments to maintain system service.

While many current traffic management operations distribute control among human operators responsible for different sectors or areas, fully automated solutions rely on decentralised control distributed at the train level, which ensures scalability across network configurations and fleet sizes.

In this work, each train is managed by a single agent, and the two terms are used interchangeably. Each agent perceives a partial, agent-centered observation based on its planned path and forecasted state. This prediction may not accurately reflect future conditions, because in a distributed control setting, other trains execute independent actions that can alter the shared environment. As a result, an individual agent's anticipated future position is based only on its own estimation and local information, which may diverge from the actual state if other agents behave unexpectedly or if unforeseen interactions occur. Finally, we do not limit communication range nor consider communication, privacy, or safety issues.

The problem is modelled as a multi-agent system in which several agents operate within a shared environment under partial observability. Although each agent aims to complete its journey on time, they share limited infrastructure and may compete for the same resources, creating local conflicts. Their individual objectives are independent, but their behaviours are interdependent, since the actions of one agent can constrain or delay others. As a result, effective solutions require coordinated collaboration [5].

III. Related Work

Existing VRSP solutions can broadly be divided into two classes: *OR*-based optimisation and *RL*-based approaches. OR methods encompass both mathematical optimisation models, which formally encode objectives and constraints, and heuristics, which are typically based on domain-specific rules or simple decision strategies, and are relatively easy to implement but difficult to adapt, since operational changes require manual redesign and scalability degrades in real-time settings.

RL methods learn adaptive policies from interaction rather than rules, enabling broader generalisation. In multi-agent settings they gain scalability through decentralised decision-making. Despite this flexibility, deploying RL in distributed systems remains challenging, requiring careful reward design and coordination among agents.

Flatland-RL challenges have served as a benchmark for studying the VRSP in large-scale railway environments. In the 2020 edition, *An_Old_Driver* [6] combined PP [9], LNS [10], and Minimum Communication Policy (MCP) [11] for deadlock-free re-planning. The *JBR_HSE* team [1] proposed an RL solution using tree-based observations and neighbour communication, and introduced a *scheduler* via SL to dispatch trains based on estimated success probability. *Netcetera* [1] adopted graph-based observations but reported reduced resource utilisation and noise from deeper lookahead.

Building on these ideas, [2] introduced a TreeLSTM model to encode future paths and anticipate deadlocks, improving toward OR baselines but underusing capacity in low-density settings. More recently, [7] introduced LCPPO, a PPO extension with local critics. Although it improved success rates over earlier RL baselines, its scalability remains limited.

MARL shows promise for realistic railway scheduling [8], though coordination, observation design and dispatch control remain open challenges.

Most existing methods rely on heuristic optimisation or monolithic RL models that struggle with scalability and adaptability. To overcome these limitations, this work formulates the VRSP as a semi-hierarchical MARL problem in which a dispatching policy decides which trains to release and when, and a routing policy handles on-track path following. This decomposition captures the operational distinction between dispatching trains and routing them once active, and it forms the foundation of the Maze-Flatland framework.

Table I summarises key approaches to the VRSP in railway operations, comparing them by *Approach* (heuristic or RL-based). To provide a more precise breakdown, we distinguish two functional dimensions: *Dispatching* (disp), which governs when trains are released and how their sequencing or priorities are managed, and *Routing* (rout), which concerns how a method determines a train's path through the network. The table also reports *Novelty* (the central contribution of each method), *Local Obs.* (whether decisions rely on localised agent observations), *Rail Topology* (fixed layouts versus randomly generated networks), and *Limitations* (the method's main weaknesses).

TABLE I: Summary of related work.

Reference	Approach Disp	Approach Rout	Novelty	Local Obs	Rail Topology	Limitations
An_Old_Driver 2021 [6]	OR		PP, LNS partial re-planning	✗	✓	centralised and iterative approach
JBR_HSE 2021 [1]	SL	RL	agents share messages supervised dispatching	✓	✓	scheduler with fixed threshold
Netcetera 2021 [1]	RL		rail-as-graph priority scheduling	✓	✓	under-usage of resources
TreeLSTM 2022 [2]	RL		path encoding with TreeLSTM	✓	✓	under-usage of resources
LCPPO 2024 [7]	RL		dynamic local per-group critic	✓	✓	poor scalability and frequent deadlocks
Schneider et al. 2024 [8]	RL		high-fidelity proprietary simulator	✓	✗*	simulator-dependent and fixed rail layout
Our solution	**RL**	**RL**	**semi-hierarchical decision making**	✓	✓	**constant speed**

*Evaluated on different sections of the same network.

Unlike previous monolithic RL methods, Maze-Flatland decomposes decision-making into two specialised models, strengthening the learning signal through subtask-specific observations and reducing noise propagation. To our knowledge, no prior Flatland-RL work learns separate dispatching and routing policies as MARL agents with distinct observation spaces, although earlier work has explored heuristic dispatch–routing decompositions [1].

IV. Motivation

In Flatland-RL, at each timestep each train selects one of five actions: *NOOP* (maintain current state), *STOP* (hold the train in the current cell), or a movement action: *FW* (forward), *LEFT*, or *RIGHT* (turn).

During early experimentation, our analysis of agent behaviour in the environment revealed a strong imbalance between *OFF-MAP* (undispatched) and *ON-MAP* (active) experiences, as shown in Figure 1. Since dispatching is implicitly tied to the forward action, fewer than 15% of these actions correspond to departures. At higher densities, agents increasingly choose the *NOOP* to delay dispatch under congestion.

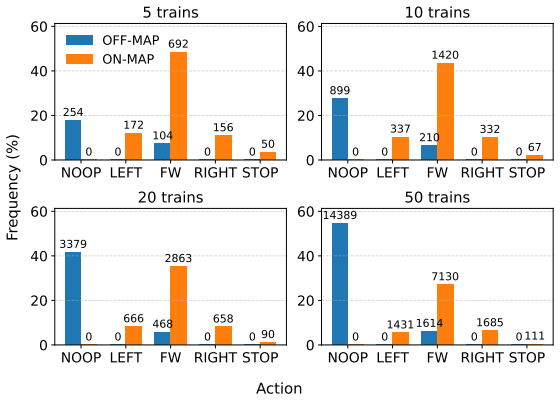

Fig. 1: Action distribution across 20 episodes for a trained agent on random maps where at least 90% of trains arrive at their target.

The observed behavioural imbalance reflects a well-known limitation of monolithic RL policies: frequent subtasks dominate learning signals, while rare but high-impact decisions (dispatching) remain underexplored. This problem is known as the Hard-Exploration Problem [12], which arises in environments with very sparse or even deceptive rewards. Random exploration is prone to failure as it will rarely discover successful states or obtain meaningful feedback from the environment. Hierarchical RL theory therefore motivates explicit modelling of rare, temporally extended decisions as separate high-level policies to stabilise exploration and value estimation. In railway operations, dispatch timing is precisely such a rare yet high-impact decision. The absence of an explicit dispatch action further reduces its exploration, reinforcing a routing-dominant bias. Our experiments indicate that this effect becomes more pronounced when using incremental training approaches, as in [1], [2]. In addition, dispatching and routing pull the policy in different directions: delaying a train may support global flow, whereas locally it appears suboptimal. A single value function must reconcile these competing signals, which complicates the learning process. These factors motivate separating the decision scopes. The semi-hierarchical framework introduced in the next section defines distinct state and action spaces for dispatching and routing, reducing interference and enabling task-specific observations.

V. Method

To address the VRSP, we adopt a semi-hierarchical MARL formulation that separates dispatching from routing. Our design follows hierarchical RL principles, decomposing the problem into subgoals and subtasks. This aligns with established hierarchical RL frameworks such as MAXQ [13], which assume that the designer specifies useful subgoals and corresponding subtasks. In contrast to fully hierarchical RL, however, our method optimizes a single global objective and enforces a strict temporal ordering between subpolicies: the dispatching policy is executed first, and the routing policy is used only after dispatching has been completed, rather than being invoked arbitrarily at any time. By imposing this structure, the designer restricts the space of admissible policies during

reinforcement learning, which typically improves tractability and sample efficiency.

We model the VRSP as a Partially Observable Markov Decision Process [14] (POMDP) with partial observability limiting agents to local information. The environment is partitioned into dispatching and routing subspaces (S_d, S_r) with action sets (A_d, A_r). Transitions T and rewards R follow standard Markov Decision Process (MDP) definitions. An agent starts in dispatching space (S_d, A_d) and transitions to routing space (S_r, A_r) after a dispatch action.

Dispatching and routing operate on different temporal and spatial scales. Dispatching occurs only once per train and determines when it enters the network, giving it a sparse but high-impact role. Routing, in contrast, involves frequent local decisions made throughout the journey. In a POMDP, combining them into a single policy forces the agent to learn a joint value function over heterogeneous state features, creating high variance and action imbalance. These structural differences motivate a decomposition in which each policy focuses on its own decision scope. Based on this formulation, we design two tailored action–observation spaces in Flatland-RL that reflect the requirements of dispatching and routing respectively.

The semi-hierarchical control loop, illustrated in Figure 2, governs both dispatching and routing. The environment provides each agent with an observation, which is processed by either **MADS** for dispatching or **MAPF** for routing. The decision controller triggers decisions only when necessary, such as at switches.

MADS — The dispatching agent selects between two actions: *Dispatch* or *Wait*. The *Wait* action delays the decision by one timestep, while *Dispatch* places the train onto the rail, enabling the use of the routing policy. **MADS** learns a decentralised policy based on current network conditions.

The observation combines global metrics, such as total trains, map occupancy, episode length, and remaining time, with conflict information assessing whether the network can accommodate another train. At dispatch time, active trains share their top-k paths, which are compared with those of the waiting agent to compute overlapping cells (*N_conflict_cells*)

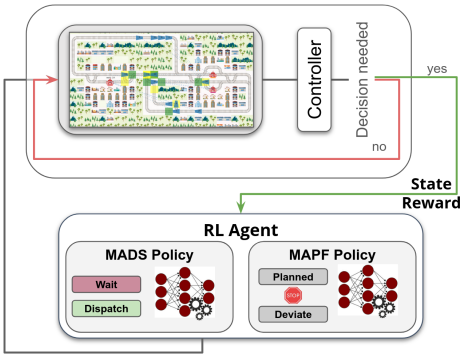

Fig. 2: Semi-hierarchical control loop with decision controller for skipping.

and distances to potential clashes (*Dist_conflict*) as reported in Figure 4.

MAPF — For routing, the action space reflects real-world constraints. At each switch, a train may face up to two possible directions and can follow the *Planned* path, *Deviate* to an alternative route, or *Stop* until a new movement signal is given. Deviations trigger replanning along the shortest path to the target, ignoring obstacles. Each agent follows a predefined route until deviation occurs, mirroring real-world operations where dispatchers assign fixed paths.

The observation refines the tree-based structure (*TreeObsForRailEnv*) from the Flatland-RL repository [15]. Cells up to the next Decision Point (DP), defined as cells where an on-map train has one or more routing options, are collapsed; each path is encoded using a binary representation distinguishing the *shortest* and *alternative* paths. For branches beyond the first decision point, only deadlock presence, distance, and travel cost are recorded.

To enable decentralised coordination, each agent also observes the observation of the first encountered agent along its projected path at depth 1, where the projected path includes both the shortest and the alternative branch. The observation is depicted in Fig. 3.

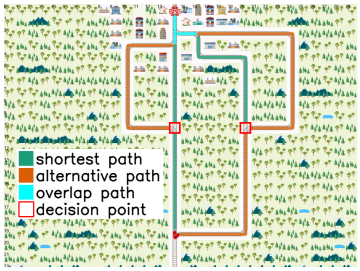

Fig. 3: **MAPF** — representation of an observation.

Each agent is equipped with a **decision controller** that prevents unnecessary decisions [16] under masking conditions. For **MADS**, actions are skipped if the train has not yet reached its departure time or if no valid path exists to the target within the episode horizon. For **MAPF**, masking limits available actions to those feasible at the current cell, considering route divergence and nearby conflicts.

The semi-hierarchical MARL framework is outlined in Alg. 1. The environment and the observation spaces for dispatching (Obs_{disp}) and routing (Obs_{rout}) are initialised in lines 1–2, followed by policy initialisation at line 3. For each timestep t_i in an episode, the joint action vector $\mathcal{A}_i$ is computed from the two policies. If an agent has not yet departed, its observation from Obs_{disp} is passed to **MADS** to select an action (lines 7–9); otherwise, Obs_{rout} and **MAPF** are used (lines 10–12). Finally, the environment steps with $\mathcal{A}_i$ (line 15).

The semi-hierarchical decomposition isolates two decision layers with different scopes: dispatching controls global traffic density, whereas routing handles local conflict resolution by selecting deviations or stops.

Algorithm 1 Maze-Flatland decision process: Semi-hierarchical agent–environment interaction.

initialise environment E
initialise observation builders for dispatching (Obs_{disp}) and routing (Obs_{rout})
initialise policies **MADS** (π_{mads}) and **MAPF** (π_{mapf})
for each timestep t_j **do**
  Compute $\mathcal{A}_t \leftarrow \langle a_t^0, a_t^1, \ldots, a_t^n \rangle$
  for each agent α^i **do**
    if α^i is outside of the map **then**
      $s_t^i \leftarrow Obs_{disp}(E)$
      $a^i \leftarrow \pi_{mads}(s_t^i)$
    else
      $s_t^i \leftarrow Obs_{rout}(E)$
      $a^i \leftarrow \pi_{mapf}(s_t^i)$
    end if
  end for
  step E with $\mathcal{A}_i$
end for

VI. Simulation Setup

All experiments were conducted on the Flatland-RL environment [15], using versions 4.0.4 and 3.0.15 (the latter for backward compatibility with existing baseline implementations). Aside from the random-disruption generator, the two versions are identical; we therefore used the same timetables, rail configurations, and disruption profiles in all experiments, varying only the sampled malfunction timings.

We adopt a high-concurrency evaluation regime. To create this setting, we run all trains at a constant speed of 1, which causes many trains to be simultaneously active on the network. This produces denser traffic, more frequent conflicts, and a substantially harder dispatch-routing problem. Fractional-speed profiles, in contrast, slow down part of the fleet and effectively stretch the operational window, lowering concurrent activity and reducing congestion. As our goal is to assess scalability under dense traffic, all baselines and Maze-Flatland are evaluated using the constant-speed configuration.

To support the semi-hierarchical decision structure, we built a Flatland-RL wrapper enabling action masking and step-skipping. Implementation is open-source.[1]

Agents exchange observations under the assumption of flawless communication; both communication failures and adversarial messaging are out of scope. For the **MAPF** policy, we limit the observation depth to 2 (full segment after the next DP), as shown in Figure 3. Increasing this depth would raise computation time while adding superfluous and inaccurate information, since observation relies on projected rather than actual future train positions.

We evaluated Maze-Flatland across multiple instances to assess the robustness of the algorithm under increasing complexity and denser schedules. Inspired by past Flatland-RL challenges, we ran our agent over 50 seeds in a setup similar to the competition setting [1]. We simulated five levels with an increasing number of trains, ranging from 7 to 80.

[1] https://github.com/enlite-ai/maze-flatland

During evaluation, all trains move at a constant speed of 1, traversing one cell per timestep, and malfunctions last between 20 and 50 timesteps. The maximum number of rails between cities and within each city is fixed at 2. The number of trains, map size, number of cities, and malfunction probability for each test level are given in Table II.

TABLE II: Instance complexities used for evaluation.

Test	Trains	Map size	Cities	π Malf.
0	7	30×30	2	1/540
1	10	30×30	2	1/900
2	20	30×30	3	1/1,800
3	50	30×35	3	1/4,500
4	80	35×30	5	1/7,200

Agent Architecture

Figure 4 depicts the shared architecture for both **MAPF** and **MADS**.

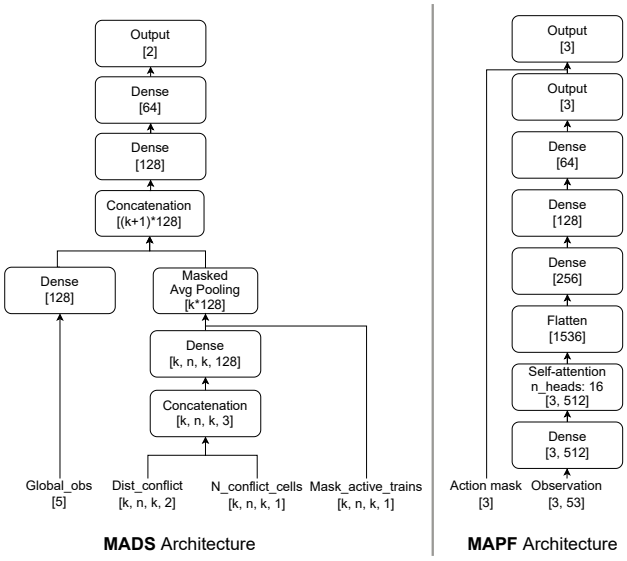

Fig. 4: Semi-hierarchical agent architecture.

For **MADS**, input comprises four blocks: a 5-element global-feature tuple, the *Dist_conflict* vector, the *N_conflict_cells* (depending on active-train count n), and top-k path comparisons, plus a binary mask identifying active trains at each timestep. We fix n to enable mini-batch learning and use the mask to indicate active trains. *Dist_conflict* and *N_conflict_cells* are embedded from 3 dimensions to 128, and an average pooling across the masked n trains and their top-k paths yields a $k \times 128$ vector. The pooling compresses the features associated with the top-k candidate paths of the off-map train, yielding a single summary vector used for the dispatch decision.

Concurrently, *Global_obs* is embedded to 128-dim and concatenated with the pooled conflict features. The combined vector passes through dense layers to produce a binary output. Masking is redundant due to binary action space and the *Decision Controller*. All experiments use $k = 2$ (top-2

shortest paths). Increasing k offered negligible benefits while substantially enlarging the observation, since the environment must compute path conflicts against all k alternatives for each on-map train.

For **MAPF**, the input is a $[3, 53]$ tensor (three 53-dimensional vectors: the current train plus two candidates for conflicts), with an action mask to block infeasible moves. Each 53-dim vector is linearly projected to 512 features, processed through a 16-head self-attention module equivalent to a single Transformer encoder layer, added back to its input and passed through a 10% dropout (disabled at inference). The output is concatenated into a vector of size $1,536$, which is finally run through three dense layers and combined with the mask.

Model hyperparameters are selected through a grid search: **MADS** uses *Tanh*, while **MAPF** uses *ReLU* activations.

Training

While specific implementation details are documented in the repository, this section summarises the training process.

MADS and **MAPF** are trained independently through an iterative Behavioural Cloning (BC) procedure. Trajectories are generated using a fixed-budget Monte Carlo Tree Search (MCTS) planner following [17]. The search uses policy logits as priors and Monte-Carlo returns as value estimates, with depth limits and simulation budgets chosen to keep rollout time tractable in Flatland-RL. BC updates are performed with minibatched training and early stopping based on validation loss. As BC quality depends on trajectory quality rather than their origin, we pre-filter the collected data by discarding samples from trains that fail to reach their destination.

Training for **MAPF** follows a curriculum that increases network complexity and train density. The refinement cycle is terminated once the agent reaches a satisfactory performance threshold. We stop training once the success rate exceeds *50% for 15 trains on a* 30×30 *map with 3 cities*, after which additional experience provides diminishing returns. The resulting **MAPF** policy is then frozen.

Next, **MADS** is trained with the same MCTS/BC loop, but its initialisation requires extra care because dispatch decisions are sparse. We first generate a dataset using a simple heuristic dispatcher that delays any train whose top-k paths indicate an immediate conflict, as defined in the observation space. This provides clean examples of safe dispatch behaviour, from which an initial **MADS** policy is learned via BC. We then collect a heterogeneous dataset by running scenarios with 10, 20, and 50 trains, exposing **MADS** to a wider range of congestion patterns. The combined dataset is used for refinement through additional MCTS rollouts and BC updates, improving generalisation across densities and map configurations.

During evaluation and rollout, the hierarchical model is reinstated across all agents, with each agent holding an independent copy of the two policies. Performance is measured using the metrics of Table III, capturing both solution quality and scalability.

Evaluation metrics include the rate of trains that successfully reach their target stations (*Success rate*), the proportion of trains permanently blocked by circular resource dependencies (*Deadlock rate*), and the average delay between scheduled and actual arrival times (*Arrival delay*). We also report the portion of trains that remain undispatched by the end of the episode (*Cancelled rate*), either because the agent chose not to dispatch them or due to a lack of available resources. The maximum possible delay is bounded by the episode length $T_{\max}$.

TABLE III: Evaluation metrics used in experiments.

Metric name	Interval	Notes
Success rate	[0, 1]	Higher is better
Deadlock rate	[0, 1]	Lower is better
Arrival delay	[0, $T_{\max}$]	Lower is better
Cancelled rate	[0, 1]	Lower is better

VII. Results

A. Speed Impact in Flatland-RL

Before evaluating performance, we analysed how fractional and constant speed affect timetable generation in Flatland-RL, with constant-speed trains advancing one cell per timestep and fractional-speed trains moving only every few timesteps.

While fractional speeds were framed in the challenge documentation as an added source of difficulty, our results suggest the opposite. Fractional speeds reduce motion per timestep of certain trains, increasing the overall episode horizon. This wider temporal window spreads departures and arrivals, reducing concurrency and resource conflict. In contrast, constant-speed profiles shorten the episodes, significantly increasing congestion. To quantify this effect, we reproduced the 10 seeds of Test 4 from the challenge setup [1] and evaluated the operational window, defined as the time difference between the latest arrival and the earliest departure.

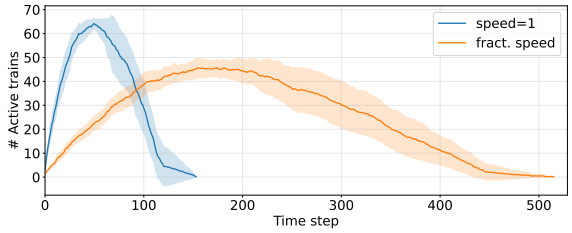

Fig. 5: Average active-train concurrency over time for 80 trains over 10 episodes on Test 4 (95% Confidence Interval).

Figure 5 shows the distribution of active trains, meaning the number of trains simultaneously within their operational window. Constant speed (speed=1) produces much shorter episodes and pushes concurrency to about 80% of all trains (around 64 active at once), creating strong competition for limited resources. Fractional speeds (fract. speed) lengthen the episode horizon, reducing peak simultaneous activity to roughly 46 trains and easing congestion across the network.

B. Performance Evaluation of Maze-Flatland

To benchmark the hierarchical approach presented in this paper we compare Maze-Flatland against four different

baselines: (i) *Greedy*, a distance-minimising agent that selects locally optimal actions; (ii) *PP*, the *An_Old_Driver*'s method [6], a heuristic planner that computes collision-free paths through sequential prioritisation; (iii) *Deadlock Avoidance*[2], a heuristic method that generates collision-free paths with minimal delay and serves as a strong benchmark for fixed-path scenarios without rerouting; and (iv) *TreeLSTM* [2], an RL method that achieves competitive performance relative to OR baselines. Each method is evaluated over 50 independent episodes across the test levels defined in Table II.

As shown in Figure 6, for test levels *0* and *1*, all methods show comparable performance across the evaluated metrics, except for *Greedy*, which suffers from a higher deadlock rate, resulting in a lower success rate. PP attains the highest Success rate, successfully dispatching nearly all trains on the map (Cancelled rate is 0% for *Test 0* with 7 trains and below 5% for *Test 1* with 10 trains). However, dispatching all trains simultaneously increases the likelihood of deadlocks, where a subset of trains block one another's paths (Deadlock rate).

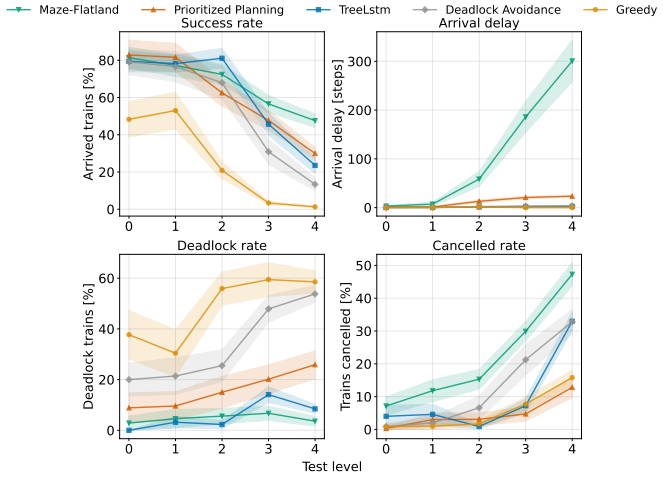

Fig. 6: Mean performance over 50 episodes across 5 levels (95% Confidence Interval).

As the load increases, Maze-Flatland consistently delivers more trains to their destinations, with about half completing their journeys in the 80-train scenario (*Test 4*). Although it shows a higher cancellation rate under extreme load, Maze-Flatland maintains near deadlock-free operation, averaging no more than four trains stuck in deadlock per episode ($\leq 5\%$).

Similarly, the *Deadlock Avoidance policy* reduces deadlock occurrence but at the cost of a higher cancellation rate compared to the *Greedy* baseline. Overall, Maze-Flatland achieves the best trade-off by minimising the average number of trains trapped in deadlock, despite a higher and growing arrival delay as traffic density increases.

The cancellation rate stems from the conservative behaviour of the MADS policy, which must accommodate as many trains as possible within the limited episode horizon imposed by the Flatland-RL environment and its constant speeds, as discussed in Section VII-A. Cancellations are not explicit **MADS** decisions; they arise when delayed trains lack time to reach their destinations before the episode ends.

These results confirm that the hierarchical structure of Maze-Flatland enhances robustness and scalability. The **MADS** component dynamically delays train departures according to on-map conditions (though it is overly conservative in smaller instances), while the **MAPF** component ensures deadlock-free planning and rescheduling under malfunctions. Together, these mechanisms enable the hierarchical design to balance efficiency and stability across varying conditions.

The reproduced results for both baselines differ from those reported in the respective papers for *PP* [6] and *TreeLSTM* [2]. This variation arises from the different speed profile used at initialisation. This work prioritises maximising the success rate and reducing deadlocks, and therefore employs constant speed profiles, whereas the original baseline implementations were designed and evaluated using fractional speeds. As noted in Section VII-A, the operation time window increases inversely with the maximum train speed. Consequently, the increased competition for shared resources substantially degraded performance. Nevertheless, Maze-Flatland demonstrates the most effective trade-off, maximising the number of trains successfully reaching their destinations.

To further assess the contribution of each module, we perform an ablation study analysing the independent effects of the **MADS** and **MAPF** components.

VIII. Ablation Study

In addition to the Maze-Flatland setups, we evaluate two hybrid configurations combining or replacing the **MADS** and **MAPF** modules with greedy counterparts to assess their individual contributions to overall performance. The results, shown in Fig. 7, illustrate the effect of each component across all test levels. The complete Maze-Flatland configuration clearly outperforms both hybrid and greedy variants, particularly under dense traffic conditions.

The two hybrid configurations perform better than the *Greedy* setup. ***MADS**-Greedy*, which combines the intelligent dispatcher with a greedy routing policy, behaves similarly to the greedy baseline but mitigates deadlocks by proactively cancelling trains. Conversely, *Greedy-**MAPF*** matches *Maze-Flatland* performance in low-density scenarios (up to 10 trains, *Test 1*), where all trains can typically be dispatched. As traffic increases, however, its performance declines due to congestion, resulting in a deadlock rate comparable to the *Greedy* setup.

Overall, the ablation confirms that both components are essential for robust performance. The **MADS** policy reduces deadlocks through adaptive departure scheduling, especially in dense scenarios with strong resource competition, while the **MAPF** module maintains efficient routing under congestion. Their joint operation enables the system to preserve stability and throughput as network density increases.

[2]Provided by the Flatland Association at: https://github.com/flatland-association/flatland-baselines.

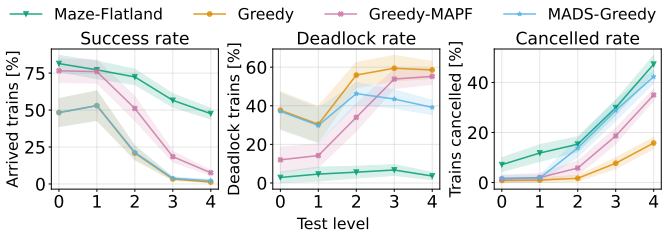

Fig. 7: Ablation study showing performance of hybrid configurations combining **MADS** and **MAPF** with greedy modules.

IX. Conclusion

This paper highlights an imbalance in agent behavior during training, where on-map actions dominate while dispatch decisions remain under-explored. The action used for dispatching coincides with the most frequent action in low-density scenarios (up to 10 trains), further amplifying this imbalance. These under-explored state–action pairs strongly influence future performance and may even lead to non-recoverable states, explaining the difficulties faced by RL methods.

To address this issue, we present Maze-Flatland, a semi-hierarchical MARL framework for the VRSP in railway operations. The proposed design separates decision-making into two complementary components: **MADS**, which governs decentralised dispatching through adaptive scheduling, and **MAPF**, which handles routing, prioritisation through stopping, and replanning under congestion and malfunctions.

By disentangling dispatching and routing, the semi-hierarchical approach enables each policy to focus on task-relevant observations and act over a minimal action space, improving coordination, scalability, and stabilising learning.

Experimental results confirmed that the hierarchical structure enhances robustness across different traffic densities. Maze-Flatland outperforms heuristic and monolithic RL baselines, achieving higher throughput and maintaining near deadlock-free operation even under heavy load. The ablation study further confirmed the complementary roles of the two modules: **MADS** prevents network saturation through controlled dispatching, while **MAPF** sustains efficient routing in dense traffic.

The modular design is expected to improve computational efficiency by reducing the cost of exploration through action-space dimensionality reduction.

Future work will optimise Maze-Flatland for scenarios with fractional train speeds to verify performance trends remain consistent under variable dynamics. Additionally, current agents prioritise deadlock-free operation and maximising arrival rate over minimising delays. We plan to explore multi-objective optimisation strategies to jointly optimise delay, throughput, and resource utilisation, bringing the framework closer to real-world railway operations.

Acknowledgments

The authors acknowledge the AI4REALNET project, funded by the European Union's Horizon Europe Research and Innovation Programme under Grant Agreement No. 101119527, and the Swiss State Secretariat for Education, Research and Innovation (SERI). The views and opinions expressed are solely those of the authors and do not necessarily reflect those of the European Union or SERI; neither the EU nor the granting authority can be held responsible for them.

References

[1] F. Laurent, M. Schneider, C. Scheller, J. Watson, J. Li, Z. Chen, Y. Zheng, S.-H. Chan, K. Makhnev, O. Svidchenko *et al.*, "Flatland competition 2020: Mapf and marl for efficient train coordination on a grid world," in *NeurIPS 2020 Competition and Demonstration Track*. PMLR, 2021, pp. 275–301.

[2] Y. Jiang, K. Zhang, Q. Li, J. Chen, and X. Zhu, "Multi-agent path finding via tree lstm," *arXiv preprint arXiv:2210.12933*, 2022.

[3] J.-Q. Li, P. B. Mirchandani, and D. Borenstein, "The vehicle rescheduling problem: Model and algorithms," *Networks: An International Journal*, vol. 50, no. 3, pp. 211–229, 2007.

[4] L. Lindenmaier, I. F. Lövétei, and S. Aradi, "Efficient real-time rail traffic optimization: Decomposition of rerouting, reordering, and rescheduling problem," *arXiv preprint arXiv:2209.12689*, 2022.

[5] J. Ferber and G. Weiss, *Multi-agent systems: an introduction to distributed artificial intelligence*. Addison-wesley Reading, 1999, vol. 1.

[6] J. Li, Z. Chen, Y. Zheng, S.-H. Chan, D. Harabor, P. J. Stuckey, H. Ma, and S. Koenig, "Scalable rail planning and replanning: Winning the 2020 flatland challenge," in *Proceedings of the international conference on automated planning and scheduling*, vol. 31, 2021, pp. 477–485.

[7] Y. Zhang, U. Deekshith, J. Wang, and J. Boedecker, "Improving the efficiency and efficacy of multi-agent reinforcement learning on complex railway networks with a local-critic approach," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 34, 2024, pp. 698–706.

[8] S. Schneider, A. Ramesh, A. Roets, C. Stirbu, F. Safaei, F. Ghriss, J. Wülfing, M. Güra, N. Sibon, R. Gentry *et al.*, "Intelligent railway capacity and traffic management using multi-agent deep reinforcement learning," in *2024 IEEE 27th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2024, pp. 1743–1748.

[9] D. Silver, "Cooperative pathfinding," in *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment*, vol. 1, 2005, pp. 117–122.

[10] P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems," in *International conference on principles and practice of constraint programming*. Springer, 1998, pp. 417–431.

[11] H. Ma, T. S. Kumar, and S. Koenig, "Multi-agent path finding with delay probabilities," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, 2017.

[12] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "Go-explore: A new approach for hard-exploration problems. arxiv. 2019," 1901.

[13] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *Journal of artificial intelligence research*, vol. 13, pp. 227–303, 2000.

[14] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[15] Flatland Association, "flatland-rl: Openai gym environment for railway management," https://github.com/flatland-association/flatland-rl, 2025, gitHub repository, accessed 2025-07-22.

[16] D. Roost, R. Meier, S. Huschauer, E. Nygren, A. Egli, A. Weiler, and T. Stadelmann, "Improving sample efficiency and multi-agent communication in rl-based train rescheduling," in *2020 7th Swiss Conference on Data Science (SDS)*. IEEE, 2020, pp. 63–64.

[17] Y. El Manyari, A. R. Fuxjaeger, S. Zahlner, J. van Dijk, A. Castagna, D. Barbieri, J. Viebahn, and M. Wasserer, "Efficient multi-objective optimisation for real-world power grid topology control," in *Proceedings of the 16th ACM International Conference on Future and Sustainable Energy Systems*, ser. E-Energy '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 614–621. [Online]. Available: https://doi.org/10.1145/3679240.3734659