AMHSTRAFFICLAB: AN INDUSTRIAL BENCHMARK FOR ROUTING AND REGIONAL TRAFFIC CONTROL IN LARGE-SCALE RESOURCE-COMPETITIVE TRANSPORT NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

In industrial settings, such as automated material handling systems (AMHS) in modern semiconductor manufacturing, transport networks must serve uneven, continuously arriving demand with large vehicle fleets and limited track space. Routing decisions interact with rules that control resource competition, including vehicle-following rules, right-of-way mechanisms, and regional admission control; together, they shape traffic flow and affect the risks of congestion, deadlock and throughput collapse. However, comparisons across industrial routing studies are difficult to reproduce when simulator implementations and accompanying control policies are incompletely documented. We introduce AMHSTrafficLab, an industrial benchmark for reproducibly evaluating and developing routing and regional traffic control in large-scale AMHS: it provides industrially grounded vehicle dynamics, track layouts, workloads, and standardized interfaces for classical, heuristic, and learning-based policies. Using the benchmark, we evaluate 13 routing methods across 7 fleet–workload scenarios and 3 regional-control settings. Our results show that without regional-control, the system deadlocks frequently regardless of the routing algorithm used, demonstrating that routing performance cannot be interpreted independently of the accompanying traffic-control policy. A proposed regional admission-control baseline reduces throughput-collapse runs from 265/910 to 9/910 and substantially suppresses local deadlocks. With regional control, we further compare different routing methods and find that learning-based methods do not consistently outperform congestion-aware heuristics, and their relative performance varies with workload and fleet density. These findings highlight the benchmark's value for evaluating routing and traffic control together and for developing future algorithms.

1 INTRODUCTION

Automated material-handling systems (AMHS) are essential to the modern semiconductor manufacturing industry. Figure 1 shows an example AMHS track network and overhead hoist transport (OHT) vehicles. In such a system, hundreds of OHTs must execute dynamically, unevenly arriving pickup-and-delivery tasks on large directed track networks. Vehicles compete for track space, merge permissions, and service resources. Waiting vehicles can obstruct upstream traffic, which allows local congestion to propagate across the network or even form circular deadlock waits.

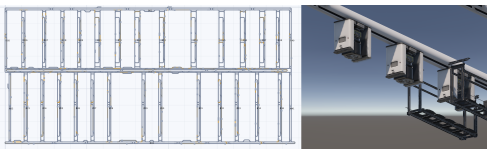


Figure 1: AMHS track network and OHTs.

Two key components integrated in the system are decisive in reducing these congestions or failures: *routing policies*, which shapes the spacial distribution of traffic; and *traffic control*, which determines how vehicles process through contested resources, including merge nodes and regional

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

control zones. Therefore, the performance of routing and control algorithms should be evaluated together under a common and controlled operating environment.

Conducting a reproducible industrial evaluation even for routing algorithms remains difficult in the existing literature. AMHS studies often rely on case-specific or incompletely documented simulators, layouts, workloads, accompanying control mechanisms, or customizable interface. On the other hand, other existing transportation benchmarks have enabled substantial progress, but they often omit one or more characteristics essential to the industrial AMHS traffic system, including large directed networks, persistent and uneven transport demand, shared-resource competition, industrially grounded workloads, and explicit collision-avoidance and deadlock-control mechanisms.

Contributions. Our main contributions are:

- **A reproducible industrial benchmark for fairly comparing and developing algorithms.** We introduce AMHSTRAFFICLAB, which combines industrially grounded AMHS dynamics, large directed layouts, uneven workloads, and standardized interfaces for classical, heuristic, and learning-based routing and regional-control policies.
- **A control-aware evaluation framework and a control baseline.** We show that without a regional-control policy, the system deadlocks frequently across all 13 tested classical, heuristic, and learning-based routing methods. This formalizes routing evaluation as conditional on the accompanying traffic-control setting. We provide a regional admission baseline that substantially reduces system collapses and suppresses small-scale local deadlocks.
- **A scenario-wide comparison of routing performance.** Under the regional admission baseline with regional empty-vehicle rebalancing on or off, we compare 13 routing methods across 7 scenarios. In terms of transport efficiency, congestion, and system collapses, learning methods do not consistently outperform congestion-aware heuristics, especially the online travel-time estimator (EDR) with best average performance.

2 RELATED WORK

In this section, we survey existing work in two sectors. We first compare related benchmarks and simulators in Table 1, then review relevant routing and traffic-control methods.

2.1 BENCHMARKS AND SIMULATORS

Traffic and railway benchmarks. Flow benchmarks mixed-autonomy vehicle and signal control (Vinitsky et al., 2018; Wu et al., 2022). CityFlow simulates large-scale microscopic traffic simulation (Zhang et al., 2019), while RESCO standardizes signal-control evaluation (Ault & Sharon, 2021). IntersectionZoo benchmarks cooperative driving at signals with data-informed traffic demand, continuous vehicle dynamics, and generalization protocols (Jayawardana et al., 2025). Flatland studies large-scale railway coordination with track transitions, occupancy constraints, and deadlocks (Mohanty et al., 2020; Laurent et al., 2021).

Warehouse benchmarks. Lifelong multi-agent pickup and delivery studies recurring tasks for a persistent robot fleet, including robot kinematics (Ma et al., 2017; 2019). RWARE and Jumanji's RobotWarehouse provide repeated shelf-delivery tasks on discrete grid (Papoudakis et al., 2021; Bonnet et al., 2024).

Benchmarks on safety and sustained operation. Safety Gym and Safety-Gymnasium combine physical simulation with explicit constraint costs (Ray et al., 2019; Ji et al., 2023). Safe-control-gym compares classical and learned controllers under constraints and disturbances (Yuan et al., 2022). These environments motivate safety evaluation. In industrial transport however, safety means more than avoiding collision: persistent mutual blocking and task throughput collapse must also be considered.

AMHS Simulators. SMAT2022 targets production control by supplying industrially calibrated semiconductor production and large-scale vehicle-simulation data (Lee et al., 2022); Sung et al.

Table 1: Representative benchmarks. ✓: documented support; ○: partial support or limited evidence; ×: absent.

Benchmark	Customizable move	Large scale	Directed network	Lifelong closed fleet	Realistic demand	Realistic motion	Collision safety	Deadlock and liveness
Flow benchmarks (2018)	✓	✓	✓	×	○	✓	✓	✓
CityFlow	✓	✓	✓	×	○	✓	✓	✓
RESCO	✓	✓	✓	×	✓	✓	✓	✓
IntersectionZoo	✓	×	✓	×	✓	✓	✓	✓
Flatland	✓	✓	✓	×	○	×	✓	✓
RWARE / RobotWarehouse	✓	×	×	✓	○	×	✓	×
Safety Gym / Safety-Gymnasium	✓	×	×	○	×	✓	✓	×
Safe-control-gym	✓	×	×	×	×	✓	✓	×
SMAT2022	×	✓	✓	✓	✓	✓	✓	○
Sung et al. (2025)	✓	×	○	✓	✓	×	✓	✓
AMHSTRAFFICLAB	✓	✓	✓	✓	✓	✓	✓	✓

Criteria. Customizable move: user-defined movement, action selection, or routing; large scale: at least 100 vehicles or robots in one network (not batches); directed network: direction-dependent movement or service graph; lifelong closed fleet: successive tasks, including completion-triggered requests, for the same fleet; realistic demand: empirically calibrated or informed by operational temporal or spatial patterns; realistic motion: continuous-position kinematics or physics; collision safety: contact, occupancy-conflict, or avoidance checks; deadlock and liveness: detection, prevention, or resolution of persistent blocking.

(2025) study OHT operation, dispatching, and routing in a small-scale discrete-event setting. However, a public, large-scale AMHS benchmark with customizable routing and regional control remains unavailable in these studies.

2.2 ROUTING AND CONTROL IN RELATED TRANSPORT SETTINGS

General routing policies. Classical shortest-distance routing uses Dijkstra's algorithm (Dijkstra, 1959). Direct variants include selecting among $k > 1$ shortest paths, or assigning higher heuristic costs to busy tracks. Q-routing learns packet-delivery times from local feedback (Boyan & Littman, 1993). DQN approximates action values with neural networks (Mnih et al., 2015). MAPPO trains decentralized cooperative policies using centralized value estimation (Yu et al., 2022).

Local intersection and regional gating control. In road networks, max-pressure selects signal stages from upstream and downstream queues (Varaiya, 2013). Accumulation-based gating regulates regional inflow (Keyvan-Ekbatani et al., 2012), while perimeter and boundary control extends to heterogeneous multi-region networks (Aboudolas & Geroliminis, 2013).

Routing and traffic control specific to the AMHS area. AMHS routing studies update costs from observed traversal times Bartlett et al. (2014), congestion monitoring Lee et al. (2018), Q(λ) feedback Hwang & Jang (2020), traffic-aware state Kang et al. (2022), or learning based cost adjustment (Kang et al., 2019). Traffic-control work addresses deadlock detection and resolution (Im et al., 2010); Park et al. (2025) combine routing-cost updates with merge sequencing for urgent lots. In AMHSTRAFFICLAB, we make both routing and regional control available for comparison within the same fixed simulator.

3 AMHSTRAFFICLAB

Grounded in AMHS scenarios from semiconductor fabs, AMHSTRAFFICLAB provides a unified simulation environment for evaluating routing and regional control.

3.1 SIMULATION ENVIRONMENT

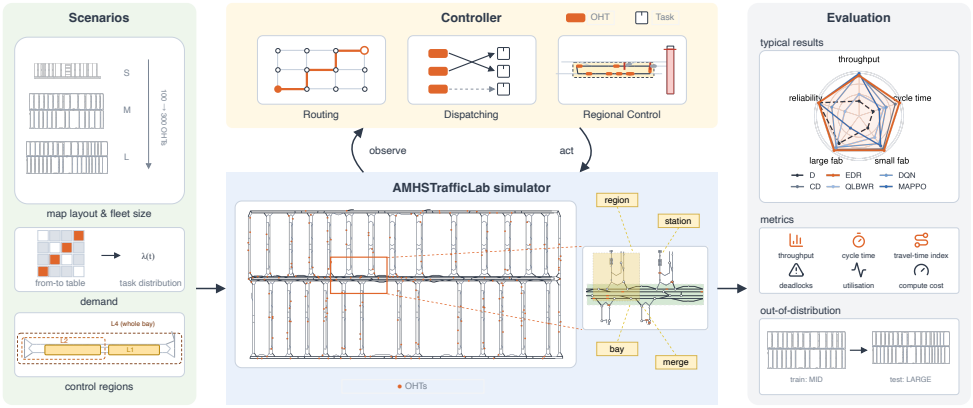


Figure 2: Architecture of AMHSTRAFFICLAB.

AMHSTRAFFICLAB represents the transport system as a directed network of track segments, intersections, and service stations. In this system, a fixed-size OHT fleet serves dynamically arriving pickup-and-delivery tasks for FOUPs (wafer carriers).

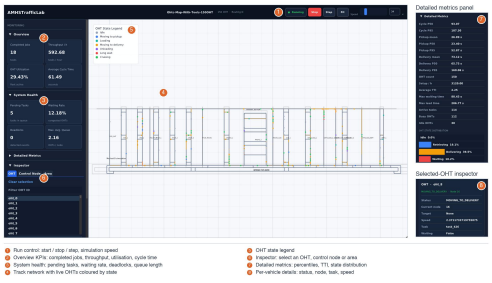


Figure 3: AMHSTRAFFICLAB GUI: track, OHT states, and metrics.

Figure 2 summarizes the platform architecture. Scenario configurations specify the network layout, fleet size, transport demand, and control regions. During operation, controllers observe the simulation state to assign tasks, plan routes, and regulate admission to controlled regions. The simulator executes these decisions and continuously updates vehicle motion, task progress, and resource occupancy. Key system states and performance metrics are visualized in real time through the graphical user interface (GUI) (See Figure 3), enabling users to monitor ongoing system behavior. In parallel, the evaluation module records and aggregates operational metrics for subsequent policy comparison and performance analysis.

To illustrate how these components work together, we follow a task that transports a FOUP from station A to station B:

1. **Task generation and vehicle assignment.** The platform generates the task according to the transport demand specified in the scenario configuration. The task waits for assignment until the dispatcher selects an available vehicle V to serve it.
2. **Route planning.** The routing policy plans a path from the current location of V to station A, guiding the vehicle toward the pickup location. The same routing policy will also be used to plan the route from station A to station B, after the FOUP has been loaded.
3. **Vehicle movement and access coordination.** Along the route, vehicle movement depends on both traffic conditions and access permissions. When V follows another vehicle, its car-following behavior is modeled using the Intelligent Driver Model (IDM) (Treiber et al., 2000), with an additional safe-distance constraint for collision avoidance. Before V enters a controlled region, a regional controller determines whether entry is permitted. At intersections, a right-of-way mechanism respects regional controllers and coordinates conflicting access requests from different vehicles.
4. **Loading and transport.** Upon reaching station A, V stops to load the FOUP and then travels to station B. The delivery trip is guided by the same routing policy and follows the same move mechanisms.

5. **Unloading and task completion.** The task ends once unloading at station B is complete. Its completion and execution time are recorded for evaluation.

In a lifelong simulation, these mechanisms operate concurrently across multiple vehicles and tasks. By jointly simulating task execution, vehicle motion, and their dynamic interactions, the platform integrates individual task lifecycles into a complete system-level transport simulation environment.

3.2 Routing and Regional Control Policies

In AMHSTrafficLab, we evaluate and provide customizing interface for two key control components: routing and regional control. Policies use available system states to make decisions.

Routing. Routing determines how a vehicle travels through the track network to its current destination. Figure 4(a) illustrates how different policies can select different paths for the same origin and destination.

The routing interface allows users to design and integrate their own policies. It provides the planning origin and destination, together with network connectivity, track status, and vehicle states. Users can select the information relevant to their policies and define how it is used to choose a path. A custom policy returns a complete route as an ordered sequence of nodes. The simulator then guides the vehicle along this route.

Regional control. When moving, vehicles may enter a circular wait, leaving the involved vehicles unable to advance. Such deadlocks can arise when queues propagate upstream until vehicles block one another around a closed loop, or when inappropriate access orders create circular waiting dependencies. The latter can occur even at relatively low overall traffic loads. Figure 4(b) illustrates deadlocks.

The platform therefore complements routing with regional control to coordinate vehicle entry and reduce the risk of deadlock. It also supports empty-vehicle rebalancing to adjust idle vehicle distribution, helping the system continue serving tasks.

For admission control, the interface provides regional capacity and occupancy, waiting vehicles, and existing entry permissions. A policy uses this information to adjust the admission mode, including whether new entry permissions should be issued. The simulator manages entry queues, permissions, and vehicle entry under the shared access rules. Figure 4(c) illustrates admission decisions at low and high occupancy.

For empty-vehicle rebalancing, the interface provides information on each region's outstanding pickup tasks, idle vehicles, loaded vehicles destined for that region, and target vehicle reserve. Users can define custom policies that output how many empty vehicles each region needs to receive or can supply. The simulator uses these decisions to arrange transfers.

Interface definitions, accessible state fields, input and output formats, and integration procedures are described in Appendix B.

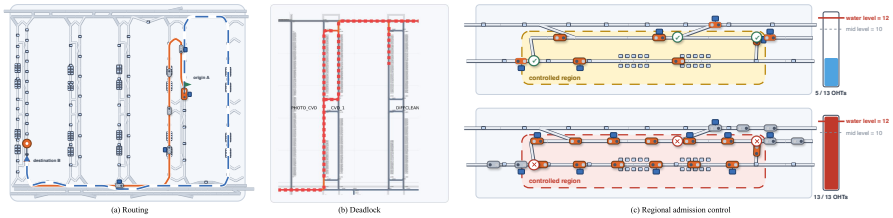


Figure 4: (a) Different routing policies select alternative paths. (b) Circular vehicle waiting dependencies. (c) Regional admission control.

4 Evaluation Settings and Protocol

Using the simulation environment described above, we construct evaluation scenarios with different track layouts, fleet sizes, and transport demands.

4.1 Scenarios Setting

The benchmark comprises seven scenarios based on three track layouts simplified from real AMHS maps, covering fleet sizes from 100 to 300 OHTs and transport demand from 2,400 to 7,350 tasks/hour. Each scenario is defined by its track network, fleet size, bay-to-bay transport demand, and regional configuration. Transport demand is specified through a From–To table, where each entry gives the task arrival rate for an origin–destination bay pair. Details of the scenes are provided in Appendix C.

4.2 Routing and Regional Control Baselines

Routing baselines. We compare 13 routing methods through a unified interface. Classical and online adaptive methods include Dijkstra routing (D, Dijkstra (1959)), a k-shortest-path method using multiple candidate paths (KD), congestion-weighted routing that prices each track by its current occupancy and downstream queue (CD), and predictive-flow routing that estimates traffic flow from the From–To demand table (PFD). In addition, EDR selects paths using travel-time estimates updated online (Bartlett et al., 2014), while CBR performs segment-level routing using moving-average congestion information (Lee et al., 2018).

Learning-based methods include QLBWR (Hwang & Jang, 2020), QECV (Kang et al., 2022), and BRQ, our upstream-residual variant of Q-routing (Boyan & Littman, 1993). Neural methods include NQ and MLP DQN, using DQN value learning (Mnih et al., 2015), GDQN with GraphSAGE-inspired neighborhood features (Hamilton et al., 2017), and MAPPO with behavior-cloning initialization (Yu et al., 2022). We adapt these methods to OHT route selection through the common interface. Their implementations, learning procedures, and parameter settings are provided in Appendix D.

Regional control baselines. We adopt a water-level regional admission-control policy motivated by accumulation-based gating (Keyvan-Ekbatani et al., 2012; Aboudolas & Geroliminis, 2013). The policy operates in three modes—*open*, *quota*, and *hold*—and switches among them according to regional vehicle occupancy and engineered predefined thresholds. We additionally implement a simple empty-vehicle rebalancing baseline that computes supply–demand imbalance for a region uses pending pickup tasks, idle vehicles, destined loaded vehicles and target empty-vehicle reserves, and cooperate inter-regional repositioning. The corresponding decision rules and parameter settings for both control mechanisms are provided in Appendix D.

We evaluate three regional-control configurations: (i) no admission control or empty-vehicle rebalancing, (ii) water-level admission control only, and (iii) water-level admission control combined with empty-vehicle rebalancing. This experimental design isolates the effects of inflow admission control and empty-vehicle redistribution on AMHS performance.

4.3 Evaluation Metrics

The evaluation focuses on transport efficiency, congestion, and liveness. Transport efficiency is measured by completed tasks, throughput, pickup, delivery, and total task cycle times, and OHT utilization. For time-based metrics, we report the mean, P50, and P95 to capture both typical and tail performance.

Traffic conditions are characterized by the travel time index (TTI) and two run-level liveness indicators: confirmed persistent circular waits (DE) and throughput collapse (DL), with their classification detailed in Appendix D.4.2. Both are retained because local circular waits need not cause system-wide collapse, while throughput can collapse without a confirmed circular wait. Detailed metric definitions and calculation procedures are provided in Appendix D.4.

4.4 Experimental Setup

Each policy configuration is run for 20,000 simulation seconds in every scenario, with ten repetitions using random seeds $0, 1, \ldots, 9$. For a given scenario and seed, all configurations use the same task stream and initial vehicle states.

5 Results and Analysis

We discuss the main experimental findings in this section; complete results are reported in Appendix E.

5.1 Effects of Regional Control

We compare three configurations: no regional control (C_0), admission control only (C_A), and admission control with empty-vehicle rebalancing (C_{AR}). The results exhibit the indispensable role of regional control policies in routing comparison and in sustained system operation.

Table 2: Runs that lose service, out of the 70 scenario–seed runs of each routing method.

	C_0		C_A		C_{AR}			C_0		C_A		C_{AR}	
Method	DL	DE	DL	DE	DL	DE	Method	DL	DE	DL	DE	DL	DE
D	25	25	1	1	0	0	QECV	17	23	2	0	0	1
KD	20	20	0	0	0	0	BRQ	25	22	0	2	0	2
CD	17	16	0	0	0	0	NQ	22	24	0	0	0	0
PFD	22	26	0	0	0	0	DQN	21	24	0	0	0	0
EDR	12	16	0	0	0	0	GDQN	21	30	0	0	1	3
CBR	21	20	1	1	0	1	MAPPO	24	25	2	1	0	0
QLBWR	18	24	3	1	2	0	**Total**	**265**	**295**	**9**	**6**	**3**	**7**

Regional admission control. Table 2 shows that, without regional control (C_0), all 13 routing methods experience frequent throughput collapse and circular waits (at least 12/70 and 16/70 runs, respectively). Admission control reduces total collapses from 265/910 to 9/910 and circular waits from 295/910 to 6/910, while adding rebalancing yields 3/910 and 7/910, respectively. This confirms that regional control, rather than routing alone, is the decisive factor in sustaining system operation.

Admission control also changes deadlock scale. Without it, 92% of confirmed events are confined to one bay with tens or twenties of vehicles, whereas the rare events under regional control are predominantly network-scale with at least 57 vehicles (the only exception with 27 vehicles in a single bay). Thus, regional admission control nearly eliminates local bay-level deadlocks (Figure 5). Appendix E.1 provides the counting rule and representative snapshots.

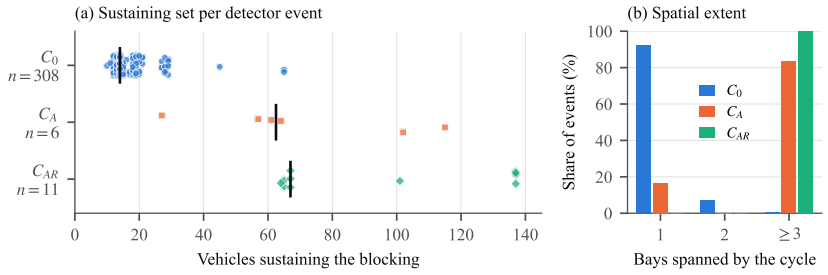


Figure 5: Scale of confirmed circular waits under the three control settings, over all 13 routing methods, seven scenarios, and seeds 0–9. (a) Vehicles sustaining each blocking, counted as in Appendix E.1; n is the number of events and vertical bars mark medians. (b) Share of events by the number of bays the waiting set spans.

Table 3: Regional-control results for Dijkstra over 20,000 simulated seconds (seeds 0–9). DL counts throughput-collapse runs; other values average non-collapse runs. Circular deadlocked events are reported in Appendix E.

Scenario	Control	DL	Throughput (tasks/h)	Mean cycle (s)	TTI	OHT util. (%)
SMALL 100	C_0	0/10	2393	123.6	2.00	79.7
	C_A	0/10	2393	120.5	1.99	78.0
	C_{AR}	0/10	2393	118.9	2.00	77.1
SMALL 150	C_0	0/10	3081	166.1	2.50	84.1
	C_A	0/10	3086	162.5	2.49	82.8
	C_{AR}	0/10	3081	174.9	2.55	86.6
MID 175	C_0	0/10	4790	90.3	2.13	67.0
	C_A	0/10	4789	89.5	2.13	66.4
	C_{AR}	0/10	4790	88.4	2.14	65.6
MID 200	C_0	1/10	5292	91.6	2.24	65.7
	C_A	0/10	5291	90.7	2.22	65.1
	C_{AR}	0/10	5291	89.9	2.25	64.5
MID 225	C_0	5/10	4924	107.4	2.34	77.1
	C_A	0/10	6011	96.5	2.35	69.8
	C_{AR}	0/10	6010	96.8	2.39	69.9
LARGE 275	C_0	9/10	3667	117.0	2.41	80.7
	C_A	0/10	6772	104.5	2.45	65.6
	C_{AR}	0/10	6776	90.1	2.52	60.2
LARGE 300	C_0	10/10	–	–	–	–
	C_A	1/10	7232	112.4	2.62	69.2
	C_{AR}	0/10	7263	93.5	2.64	61.4

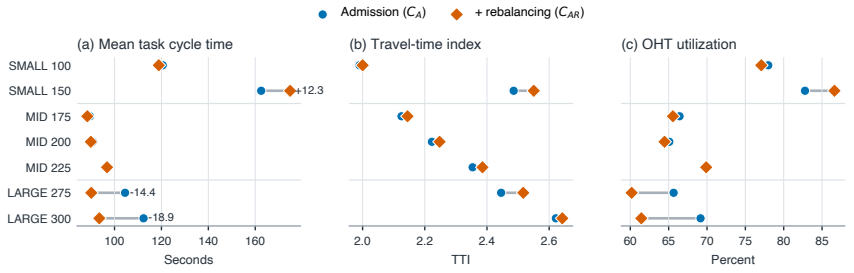


Figure 6: Effects of empty-vehicle rebalancing under Dijkstra routing. Each line connects the conditional mean under admission control (C_A) and admission with rebalancing (C_{AR}) for the same scenario. Throughput-collapse runs are excluded. Values in panel (a) report $C_{AR} - C_A$ in seconds for the three cases discussed in the text.

Empty-vehicle rebalancing. Table 3 reports the comparison under the Dijkstra routing policy. Throughput remains essentially unchanged across scenarios, while empty-vehicle rebalancing substantially reduces mean cycle time in the high-load LARGE configurations, by roughly 14–17% in LARGE 275 and LARGE 300. In both cases, OHT utilization decreases despite a slight increase in TTI, suggesting that the improvement is driven by better vehicle availability rather than faster individual trips (Figure 6). In contrast, rebalancing degrades performance in SMALL 150. Overall, its benefit is most pronounced under high-load conditions and is not uniform across scenarios.

5.2 Routing Performance across Scenarios

Common indicators for routing performance, including average cycle time, may be fairly compared only after excluding collapsed runs; thus, without regional control policy (C_0), frequently collapsed runs make such comparisons incomplete and less representative. Therefore we compare routing efficiency under the admission-control setting C_A, which suppresses collapse without adding empty-vehicle repositioning traffic. Table 4 reports throughput and mean cycle time over non-collapse runs; superscripts give the number of excluded collapse runs.

Table 4: Routing performance under admission control (C_A) over 20,000 simulated seconds and seeds 0–9. Each cell is throughput (tasks/h) / mean cycle time (s), averaged over non-collapse runs. $^{\dagger d/10}$ denotes d throughput-collapse runs.

	SMALL		MID			LARGE	
Method	100	150	175	200	225	275	300
D	2393/120.5	3086/162.5	4789/89.5	5291/90.7	6011/96.5	6772/104.5	7232/112.4$^{\dagger 1/10}$
KD	2392/135.7	3093/151.3	4788/94.7	5290/95.3	6006/106.9	6766/107.6	6983/161.2
CD	2394/118.7	3100/120.3	4789/90.4	5290/91.2	6010/95.5	6774/94.4	7265/97.1
PFD	2394/118.8	3100/117.7	4790/90.7	5291/91.4	6011/95.3	6775/94.3	7265/96.5
EDR	2394/115.7	3100/117.5	4790/88.6	5291/89.4	6011/92.7	6776/92.2	7266/94.4
CBR	2394/117.3	3100/119.8	4789/96.0	5289/96.0	6007/103.9	6773/103.4	7252/120.9$^{\dagger 1/10}$
QLBWR	2393/121.4	3098/135.8	4790/88.7	5290/90.0$^{\dagger 1/10}$	6011/94.1	6649/94.0$^{\dagger 1/10}$	7122/96.1$^{\dagger 1/10}$
QECV	2393/121.3	3091/154.3	4789/88.5$^{\dagger 2/10}$	5291/89.7	6011/94.2	6775/92.6	7260/99.2
BRQ	2393/120.6	3061/170.2	4789/93.2	5290/94.1	6005/103.0	6774/103.9	6961/133.6
NQ	2393/119.6	3099/126.0	4789/94.0	5290/91.9	6010/96.6	6775/93.5	7264/100.0
DQN	2394/118.9	3097/128.1	4789/91.3	5290/91.8	6010/96.5	6775/94.5	7265/97.4
GDQN	2393/120.4	3094/134.2	4789/91.6	5290/91.8	6010/96.5	6776/94.8	7265/96.7
MAPPO	2393/120.7	3099/124.4	4790/90.1	5290/91.8	6011/96.4	6776/93.7	7219/109.4$^{\dagger 2/10}$

At SMALL and MID loads, nearly all methods serve the offered demand, so cycle time is the main discriminator. Among methods without a collapse, EDR records the lowest mean cycle time in all five scenarios; QECV is 0.1 s lower in MID 175 only after two collapsed runs are excluded.

At LARGE loads, routing also affects stability and throughput. EDR remains collapse-free, with mean cycle times of 92.2 and 94.4 s in LARGE 275 and LARGE 300. In LARGE 300, KD and BRQ fall to 6983 tasks/h and 161.2 s, and 6961 tasks/h and 133.6 s, respectively, while D, CBR, QLBWR, and MAPPO have collapse runs. NQ, DQN, and GDQN remain stable and close to EDR in both LARGE scenarios, but no learned method consistently outperforms the strongest congestion-aware heuristic across all seven scenarios.

Detailed pickup and delivery times, congestion measures, deadlock counts, and OHT utilization are provided in Appendix E; so are full test results under the other two control baselines.

6 Conclusion

In this work, we introduce AMHSTrafficLab, an industrially grounded benchmark for routing and regional traffic control in large-scale AMHS. The platform combines AMHS simulation with unified policy interfaces to support the development, integration, and reproducible evaluation of routing and regional-control algorithms. We provide a regional admission-control baseline and compare 13 routing methods across 7 fleet–workload scenarios and 3 control settings. Our results show that admission control substantially reduces throughput collapse and suppresses small-scale local deadlocks, and that routing performance should be interpreted together with the accompanying control policy. Under admission control, learning-based methods do not consistently outperform congestion-aware heuristics, especially the EDR heuristic which achieves the lowest mean task cycle time in most scenarios.

One limitation of the current benchmark is that our task generation process uses fixed From–To tables. In real production environments, while the From–To table is statistically accurate, actual demand arrivals may vary over time based on the production status. In addition, the current region selection and rebalancing choices might not be fully engineered or optimally designed to output the best performance; clever customized policies might work better. Future work will extend the benchmark to support time-varying demand and investigate more effective region selection and control mechanisms. Overall, AMHSTrafficLab provides a common experimental foundation for comparing existing methods and developing new routing and regional-control strategies for industrial transport systems.

AI USE STATEMENT

We used generative AI tools to assist with coding during development of the simulation platform and implementation of the baselines, to draft sections of the paper, to assist with writing and language editing, and to support literature searches and the identification of relevant prior work. The authors reviewed and take responsibility for the final content of this work.

ETHICS STATEMENT

The benchmark maps used in this work are derived from real-world industrial material-handling systems and are anonymized and abstracted before publication. Information that could identify specific companies, factories, or production facilities, as well as other commercially sensitive information, is removed from the released materials. The public benchmark scenarios retain only the structural and operational characteristics required for research on routing and traffic control.

REPRODUCIBILITY STATEMENT

We plan to publicly release the packaged AMHSTrafficLab simulation platform, all benchmark maps used in this work, implementations of the evaluated algorithms, and the corresponding experimental configurations and execution scripts. The paper and supplementary material further document the simulation mechanisms, algorithm settings, hyperparameters, evaluation metrics, and statistical procedures. These materials will enable other researchers to independently run the benchmark and reproduce the reported experiments under the documented experimental settings.

REFERENCES

Konstantinos Aboudolas and Nikolas Geroliminis. Perimeter and boundary flow control in multi-reservoir heterogeneous networks. *Transportation Research Part B: Methodological*, 55:265–281, 2013. doi: 10.1016/j.trb.2013.07.003.

James Ault and Guni Sharon. Reinforcement learning benchmarks for traffic signal control. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine learning proceedings 1995*, pp. 30–37. Elsevier, 1995.

Kelly Bartlett, Junho Lee, Shabbir Ahmed, George Nemhauser, Joel Sokol, and Byungsoo Na. Congestion-aware dynamic routing in automated material handling systems. *Computers & Industrial Engineering*, 70:176–182, 2014. doi: 10.1016/j.cie.2014.02.002.

Clem Bonnet, Daniel Luo, Donal Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence Midgley, Elshadai Tegegn, Tristan Kalloniatis, Omayma Mahjoub, Matthew Macfarlane, Andries Smit, Nathan Grinsztajn, Raphael Boige, Cemlyn Waters, Mohamed Ali Mimouni, Ulrich Mbou Sob, Ruan de Kock, Siddarth Singh, Daniel Furelos-Blanco, Victor Le, Arnu Pretorius, and Alexandre Laterre. Jumanji: A diverse suite of scalable reinforcement learning environments in JAX. In *International Conference on Learning Representations*, pp. 49264–49293, 2024.

Justin A. Boyan and Michael L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in Neural Information Processing Systems*, volume 6, pp. 671–678, 1993.

Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1: 269–271, 1959. doi: 10.1007/BF01386390.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Illhoe Hwang and Young Jae Jang. Q(λ) learning-based dynamic route guidance algorithm for overhead hoist transport systems in semiconductor fabs. *International Journal of Production Research*, 58(4):1199–1221, 2020. doi: 10.1080/00207543.2019.1614692.

Kwangyoung Im, Kyungsup Kim, Youngmin Moon, Taeeun Park, and Sujeong Lee. The deadlock detection and resolution method for a unified transport system. *International Journal of Production Research*, 48(15):4423–4435, 2010. doi: 10.1080/00207540902991690.

Vindula Jayawardana, Baptiste Freydt, Ao Qu, Cameron Hickert, Zhongxia Yan, and Cathy Wu. IntersectionZoo: Eco-driving for benchmarking multi-agent contextual reinforcement learning. In *International Conference on Learning Representations*, pp. 75263–75289, 2025.

Jiaming Ji, Borong Zhang, Jiayi Zhou, Xuehai Pan, Weidong Huang, Ruiyang Sun, Yiran Geng, Yifan Zhong, Josef Dai, and Yaodong Yang. Safety Gymnasium: A unified safe reinforcement learning benchmark. In *Advances in Neural Information Processing Systems*, volume 36, pp. 18964–18993, 2023. doi: 10.52202/075280-0831. Datasets and Benchmarks Track.

Bonggwon Kang, Byeong Min Kang, and Soondo Hong. A dynamic OHT routing algorithm in automated material handling systems. *Journal of Korean Society of Industrial and Systems Engineering*, 45(3):40–48, 2022. doi: 10.11627/jksie.2022.45.3.040.

Younkook Kang, Sungwon Lyu, Jeeyung Kim, Bongjoon Park, and Sungzoon Cho. Dynamic vehicle traffic control using deep reinforcement learning in automated material handling system. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 9949–9950, 2019. doi: 10.1609/aaai.v33i01.33019949.

Mehdi Keyvan-Ekbatani, Anastasios Kouvelas, Ioannis Papamichail, and Markos Papageorgiou. Exploiting the fundamental diagram of urban networks for feedback-based gating. *Transportation Research Part B: Methodological*, 46(10):1393–1403, 2012. doi: 10.1016/j.trb.2012.06.008.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Florian Laurent, Manuel Schneider, Christian Scheller, Jeremy Watson, Jiaoyang Li, Zhe Chen, Yi Zheng, Shao-Hung Chan, Konstantin Makhnev, Oleg Svidchenko, Vladimir Egorov, Dmitry Ivanov, Aleksei Shpilman, Evgenija Spirovska, Oliver Tanevski, Aleksandar Nikov, Ramon Grunder, David Galevski, Jakov Mitrovski, Guillaume Sartoretti, Zhiyao Luo, Mehul Damani, Nilabha Bhattacharya, Shivam Agarwal, Adrian Egli, Erik Nygren, and Sharada Mohanty. Flatland Competition 2020: MAPF and MARL for efficient train coordination on a grid world. In *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*, volume 133 of *Proceedings of Machine Learning Research*, pp. 275–301, 2021.

Kwanwoo Lee, Siyong Song, Daesoon Chang, and Sangchul Park. A new amhs testbed for semiconductor manufacturing. In *2022 Winter Simulation Conference (WSC)*, pp. 3318–3325. IEEE, 2022.

Sangmin Lee, Junho Lee, and Byungsoo Na. Practical routing algorithm using a congestion monitoring system in semiconductor manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 31(4):475–485, 2018. doi: 10.1109/TSM.2018.2858013.

Hang Ma, Jiaoyang Li, T. K. Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 837–845, 2017.

Hang Ma, Wolfgang Hönig, T. K. Satish Kumar, Nora Ayanian, and Sven Koenig. Lifelong path planning with kinematic constraints for multi-agent pickup and delivery. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 7651–7658, 2019. doi: 10.1609/aaai.v33i01.33017651.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. doi: 10.1038/nature14236.

Sharada Mohanty, Erik Nygren, Florian Laurent, Manuel Schneider, Christian Scheller, Nilabha Bhattacharya, Jeremy Watson, Adrian Egli, Christian Eichenberger, Christian Baumberger, Gereon Vienken, Irene Sturm, Guillaume Sartoretti, and Giacomo Spigler. Flatland-RL: Multi-agent reinforcement learning on trains. *arXiv preprint arXiv:2012.05893*, 2020. URL https://arxiv.org/abs/2012.05893.

Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V. Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

Sung Woo Park, Sungwook Jang, and Young Jae Jang. Proactive hot lot routing for OHT systems in semiconductor fab. In *Proceedings of the Winter Simulation Conference*, 2025.

Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. Technical report, OpenAI, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Jin-Hyeon Sung, Seong-Hyeon Ju, Seung-Wan Cho, Hak-Jong Joo, Kyung-Min Seo, and Bong-Gu Kang. Modeling and exploratory analysis of discrete event simulations for optimizing overhead hoist transport systems and logistics in semiconductor manufacturing. *Mathematics*, 13(7):1167, 2025. doi: 10.3390/math13071167.

Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 62(2):1805–1824, 2000. doi: 10.1103/PhysRevE.62.1805.

Pravin Varaiya. Max pressure control of a network of signalized intersections. *Transportation Research Part C: Emerging Technologies*, 36:177–195, 2013. doi: 10.1016/j.trc.2013.08.014.

Eugene Vinitsky, Aboudy Kreidieh, Luc Le Flem, Nishant Kheterpal, Kathy Jang, Cathy Wu, Fangyu Wu, Richard Liaw, Eric Liang, and Alexandre M. Bayen. Benchmarks for reinforcement learning in mixed-autonomy traffic. In *Proceedings of the 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pp. 399–409. PMLR, 2018.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

Cathy Wu, Abdul Rahman Kreidieh, Kanaad Parvate, Eugene Vinitsky, and Alexandre M. Bayen. Flow: A modular learning framework for mixed autonomy traffic. *IEEE Transactions on Robotics*, 38(2):1270–1286, 2022. doi: 10.1109/TRO.2021.3087314.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of PPO in cooperative multi-agent games. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24611–24624, 2022. Datasets and Benchmarks Track.

Zhaocong Yuan, Adam W. Hall, Siqi Zhou, Lukas Brunke, Melissa Greeff, Jacopo Panerati, and Angela P. Schoellig. Safe-Control-Gym: A unified benchmark suite for safe learning-based control and reinforcement learning in robotics. *IEEE Robotics and Automation Letters*, 7(4):11142–11149, 2022. doi: 10.1109/LRA.2022.3196132.

Huichu Zhang, Siyuan Feng, Chang Liu, Yaoyao Ding, Yichen Zhu, Zihan Zhou, Weinan Zhang, Yong Yu, Haiming Jin, and Zhenhui Li. CityFlow: A multi-agent reinforcement learning environment for large scale city traffic scenario. In *The World Wide Web Conference*, 2019. doi: 10.1145/3308558.3314139.

APPENDIX

A System Architecture and Simulation Mechanics **15**
A.1 Overall Architecture and Module Interactions 15
A.2 Traffic Entities and Demand Modeling 16
A.3 Vehicle Motion and Simulation Execution 17
A.4 Policy Decisions and Dispatch 20
A.5 Shared Resources and Access Coordination 21
A.6 Runtime Monitoring and Statistics 22

B System State and Customizable Policies **24**
B.1 Observable System State 24
B.2 Routing-Policy Interface 28
B.3 Regional-Control Interface 29
B.4 Policy Invocation and Decision Handling 30
B.5 Minimal Policy Examples 31

C Scenario Descriptions and Construction Guide **37**
C.1 Scenario Components 37
C.2 Data Representation of Scenario Elements 37
C.3 Region Definition and Water-Level Configuration 39
C.4 Process Flows and Workload Construction 40
C.5 Equivalent Production Scale 42
C.6 Scenario Statistics 42

D Baseline Algorithms and Evaluation Protocol **45**
D.1 Common Execution Protocol 45
D.2 Routing Baselines 45
D.3 Regional Control Baselines 51
D.4 Evaluation Metrics 53
D.5 Software and Hardware 55

E Complete Experimental Results **55**
E.1 Size, Extent and Selected Snapshots of Detected Circular Waits 55
E.2 Full routing results 57
E.3 Transfer across Fleet Size and Topology 67

F Benchmark Availability and Licensing **69**

G Installation, Platform Use, and Reproduction **70**
G.1 Installation 70

G.2 Running the Platform 70
G.3 Paper Scenarios and Batch Evaluation 70
G.4 Checking Generated Results 71

A SYSTEM ARCHITECTURE AND SIMULATION MECHANICS

AMHSTRAFFICLAB models AMHS operation in semiconductor fabs while abstracting facility-specific details. This appendix describes the main objects, their relationships, and their execution mechanisms across five core modules: simulation orchestration, traffic modeling, policy decisions, resource coordination, and runtime monitoring. The platform executes different policies through common vehicle-motion, task-service, and resource-interaction mechanisms, providing a consistent simulation basis for algorithm comparison.

This section explains the simulation mechanisms used in the experiments. The four code blocks below are pseudocode: they show the order of operations and selected internal data fields of the source code, but are not commands to run or modules to import. In these blocks, `...` means that details have been omitted. The user distribution contains a compiled wheel and usage documentation; to write a policy using the wheel, follow the public interfaces and runnable examples in Appendix B.

A.1 OVERALL ARCHITECTURE AND MODULE INTERACTIONS

Core objects and modules. The platform uses `AMHSSimulation` to orchestrate simulation and `AMHS` to aggregate traffic-system state. `AMHSSimulation` handles scenario initialization, time advancement, and the order of module calls. `AMHS` stores the network, vehicles, tasks, and shared resources, and maintains their relationships. Table 5 groups the main modules by function and lists their core classes or interfaces.

Table 5: Core simulation modules, main implementations, and responsibilities.

Module	Main classes or interfaces	Responsibilities
Simulation orchestration and state management	`AMHSSimulation`, `EventEngine`, `AMHS`	Scenario assembly, state aggregation, event scheduling, and time advancement.
Traffic entities and demand modeling	`ControlNode`, `RailPath`, `OHT`, `TransportationTask`, `FromToTableGenerator`	Network, vehicles, transport tasks, and task arrivals.
Policy decisions and dispatch	`RoutingStrategy`, `AreaRebalanceController`, and dispatch and yielding functions	Task assignment, route planning, regional repositioning, and local yielding.
Shared resources and access coordination	`Resource`, `MergeResource`, `MergeClaim`, `WaterLevelArea`	Requests, queues, grants, and releases at merges and regional entrances.
Runtime monitoring and statistics	`DeadlockMonitor`, detection and statistics methods in `AMHS`, and execution records in `TransportationTask`	Persistent-wait detection, task execution records, metric aggregation, and event tracking.

Module interactions. At initialization, `AMHSSimulation` assembles the network, fleet, and control regions from the scenario configuration. During operation, task generators release transport demand, and policy modules use the current state to assign tasks, plan routes, and reposition vehicles. Vehicle objects, `OHT`, carry out these decisions and interact with resource objects at merges and regional entrances. The main loop updates vehicle, task, and resource states in a defined order, while the monitoring module records execution and outcomes. The updated state also becomes the input to subsequent decisions.

Common execution mechanisms. Policy outputs affect traffic through these objects and their interactions. For example, replacing a routing policy leaves vehicles governed by the same `OHT` motion and state-transition logic and the same request and grant procedures for shared resources. Policy performance can therefore be compared under common following, service, and access rules.

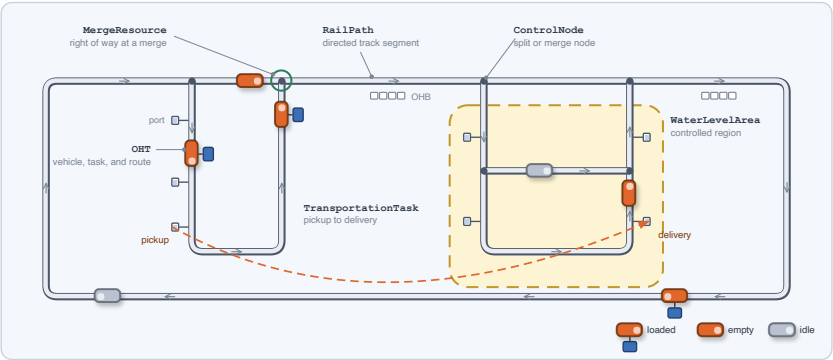


Figure 7: Simulation scenario and core objects. Vehicles execute transport tasks on a directed track network. A `ControlNode` marks a split or merge, a MergeResource arbitrates right of way at a merge, and a WaterLevelArea controls admission to a region. The dashed arrow shows one TransportationTask from a pickup port to a delivery port.

A.2 Traffic Entities and Demand Modeling

Scenario configuration and assembly. Scenario configurations specify the track network, fleet size, transport demand, and execution parameters. Scenario files, workload construction, and static statistics are described in Appendix C. `AMHSSimulation` invokes `AMHSSampleGenerator` to read scenario data and create traffic entities, then passes these objects and policy parameters to `AMHS`, establishing the environment for task generation and vehicle operation.

Track network and service locations. The track network is represented as a directed graph that provides the spatial structure for vehicle movement and task pickup and delivery. Nodes and tracks are represented by `ControlNode` and `RailPath`, respectively. `ControlNode` stores node position, type, incoming and outgoing edges, and service-location information. `RailPath` stores track length, speed limit, geometry, and the vehicles on the track.

Vehicle parameters and motion state. Each vehicle is represented by an `OHT` object with default or scenario-specific motion and service parameters. `OHT` records motion state through its current track `path`, distance along the track `x`, speed `v`, and acceleration `a`. The motion model and shared parameter settings are described in Section A.3.

Transport tasks and demand generation. Transport demand enters the simulation through individually released tasks. Each `TransportationTask` object records the pickup and delivery locations, task status, and stage timestamps for one task. For `FromToTableGenerator`, let λ_{ij} denote the arrival rate for the origin–destination partition pair (i, j) in the From–To table, measured in tasks per hour. Within a generation window $[t_0, t_0 + \Delta T)$, the number of candidate tasks is sampled as

$$N_{ij} \sim \text{Poisson}\left(\lambda_{ij} \frac{\Delta T}{3600}\right), \tag{1}$$

where t_0 is the window start time and ΔT is its duration, both measured in seconds. For candidate tasks that pass endpoint-validity checks, the generator samples scheduled release times independently and uniformly within the window:

$$t_{ij,n} = t_0 + U_{ij,n}, \qquad U_{ij,n} \overset{\text{i.i.d.}}{\sim} \text{Uniform}(0, \Delta T). \tag{2}$$

Here, n indexes candidate tasks that pass the checks, and $U_{ij,n}$ is the release offset in seconds. The corresponding release events are added to the event queue at their scheduled times. Task generation and routing use separately managed random streams, preventing a policy's internal random sampling from changing the task sequence for a given scenario and seed.

Vehicle–task relationships. After assignment, `OHT` refers to its task through `current_task` and stores its path in `routing`. The task object records transport progress, while the vehicle object records its current activity: traveling to pickup, loading, traveling to delivery, or unloading, represented by `MOVING_TO_PICKUP`, `LOADING`, `MOVING_TO_DELIVERY`, and `UNLOADING`, respectively. On task completion, the vehicle returns to `IDLE` and can receive a new task or perform an empty-vehicle movement. Task and vehicle states are maintained separately; their update timing is described in Section A.3.

A.3 Vehicle Motion and Simulation Execution

The platform executes policy decisions through common vehicle-motion and service mechanisms. OHTs travel along directed tracks, with their motion determined jointly by vehicle capabilities, track speed limits, surrounding traffic, and access permissions. Within the following range, the platform uses an Intelligent Driver Model (IDM) formulation (Treiber et al., 2000) to adjust acceleration according to spacing and relative speed.

IDM-based vehicle following. Let v and v_{lead} denote the speeds of the following and leading vehicles, respectively, and let s denote the net gap from the follower's front to the leader's rear, excluding the leader's length. The approach rate is $\Delta v = v - v_{\text{lead}}$, so a positive value indicates that the follower is closing the gap. The dynamic desired gap is

$$s^*(v, \Delta v) = s_0 + \max\left(0,\, Tv + \frac{v\Delta v}{2\sqrt{a_{\max} b}}\right), \tag{3}$$

where s_0 is the IDM minimum-gap parameter, T is the desired time headway, $a_{\max}$ is the maximum acceleration, and b is the nominal deceleration. The term Tv increases the desired gap with speed, while the relative-speed term increases it further when the follower approaches the leader. The following acceleration is

$$a_{\text{IDM}} = a_{\max}\left[1 - \left(\frac{v}{v_{\max}}\right)^{\delta} - \left(\frac{s^*}{s}\right)^2\right], \qquad \delta = 4, \tag{4}$$

where $v_{\max}$ is the vehicle speed limit. The speed term reduces the tendency to accelerate as the vehicle approaches this limit, while the gap term captures the constraint imposed by the leader. A smaller gap or a higher approach rate strengthens braking; as the leader moves away and space becomes available, the follower accelerates again. The parameter b controls the braking response and is not a hard bound on all instantaneous decelerations. Figure 8 illustrates the response to a leader that decelerates and then accelerates.

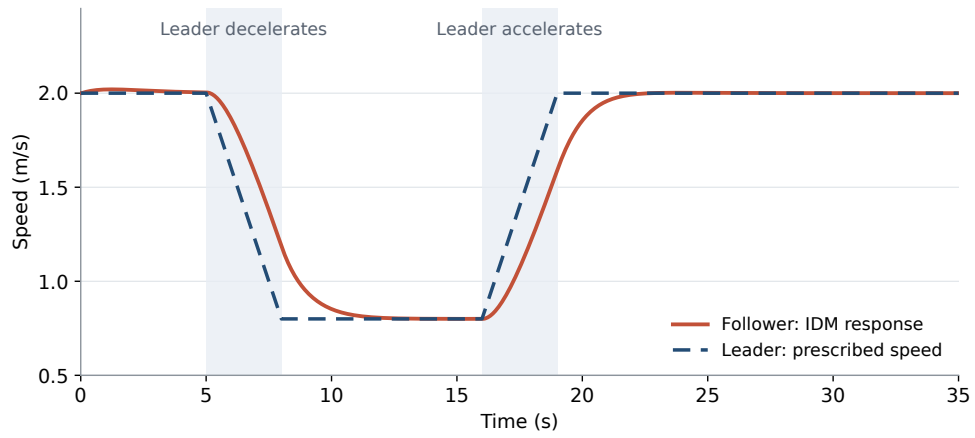


Figure 8: IDM following response. The leader decelerates from 2 to $0.8\,\mathrm{m/s}$ during 5–8 s and accelerates back during 16–19 s (shaded intervals). The follower uses the platform's motion update with the parameters listed in this subsection, an initial speed of $2\,\mathrm{m/s}$, an initial net gap of approximately $2.17\,\mathrm{m}$, and a safety stopping gap of $0.8\,\mathrm{m}$. This controlled two-vehicle example has a prescribed leader trajectory and no additional track-speed, destination-stop, or access constraints.

Other influences on vehicle motion. IDM following operates within the broader motion mechanism, which also includes:

- **Vehicle and track speed limits.** Together they determine the base target speed; when sufficient space is available, the vehicle accelerates toward this target.
- **Speed-limit and stop anticipation.** Upcoming lower-speed tracks and the current stopping target adjust the target speed within the look-ahead range.
- **Spacing and stopping constraints.** Insufficient net spacing or arrival at the relevant stopping boundary triggers a stop.
- **Right-of-way and regional admission.** Merge and region-entry permissions constrain passage; vehicles wait before the corresponding control boundary when permission is unavailable.

Shared motion and control parameters. Table 6 summarizes the settings shared across policies for a given scenario. The safety stopping gap is configured separately from the IDM minimum-gap parameter.

Table 6: Shared vehicle-motion and control parameters used in the formal experiments. Track speed limits are specified per track.

Parameter	Symbol	Unit	Value
Vehicle speed limit	$v_{\max}$	m/s	4.0
Maximum acceleration	$a_{\max}$	m/s^2	0.8
Nominal deceleration	b	m/s^2	0.8
Vehicle length	L	m	1.0
IDM minimum-gap parameter	s_0	m	0.8
IDM desired time headway	T	s	1.0
IDM speed exponent	δ	—	4
Safety stopping gap	d_{safe}	m	0.8
Track speed limit	v_{rail}	m/s	1.0, 5.0, or 5.3
Right-of-way request distance	d_{request}	m	8.0
Following and deceleration look-ahead	$d_{\text{lookahead}}$	m	8.0
Pre-merge stopping control distance	$d_{\text{merge,stop}}$	m	2.0
Post-merge resource-release distance	$d_{\text{merge,release}}$	m	1.0
Bay/Area entry control distance	d_{entry}	m	2.0
Simulation time step	Δt	s	$1/30$

Loading and unloading use fixed durations by service-location type: $13.36/11.57\,\text{s}$ at equipment ports, $9.48/6.53\,\text{s}$ at overhead buffers (OHBs), and a $10/10\,\text{s}$ fallback, respectively.

Time step and execution order. `AMHSSimulation.update()` is the entry point for one simulation step, with a base interval of $1/30\,\text{s}$. Each step first processes events due at the current time, then updates vehicle motion and discrete states, applies the relevant dispatch and regional-control operations, and finally advances the clock. Code excerpt A.1 shows this call order; the following paragraphs describe each stage.

Implementation sketch A.1. Main step order.

```
def update(self):
    self.event_engine.run_due(self.now)

    for rail_path in self.amhs.rail_paths:
        if len(rail_path.ohts) == 0:
            continue
        for oht in rail_path.ohts[:]:
            self.amhs.closest_oht_in_front(oht)
            oht.update()

    ...  # Optional following-queue statistics

    for oht in self.amhs.ohts:
        oht.switch_status()

    ...  # Immediate dispatch and rebalancing

    self.now += self.dt
    self.frame_count += 1
    self.event_engine.now = self.now
```

Scheduling due events. `EventEngine` schedules task arrivals, batch dispatch, statistical sampling, and deadlock scans through an event queue. The queue is ordered by scheduled execution time, with an increasing sequence number determining the order of events scheduled for the same time. At the start of each step, the main loop invokes the engine to process due events. An event whose scheduled time does not align with a step boundary is processed at the first step whose simulation clock reaches or exceeds that time.

Vehicle updates. After processing due events, the main loop calls `OHT.update()` sequentially in the order of tracks and their vehicle lists, advancing motion and handling access interactions along each route. Within the motion calculation, `OHT.move()` first updates position and speed using the existing acceleration, then computes the next acceleration according to the motion mechanisms described above.

Discrete state transitions. After the motion updates, the main loop traverses the fleet and calls `OHT.switch_status()`. This method uses updated positions and service timers to handle track transitions, loading, unloading, and task completion, updating task and resource relationships accordingly. Resource requests, grants, and releases can take effect immediately during motion updates and state transitions.

Control updates and clock advancement. After vehicle-state updates, the main loop processes immediate dispatch and empty-vehicle rebalancing as enabled by the configuration. Immediate dispatch handles pending tasks using the updated vehicle state; batch dispatch is a due event and executes before vehicle-motion updates. After completing the step's operations, the main loop advances the simulation clock by one step and updates the frame count and event-engine clock.

A.4 Policy Decisions and Dispatch

Policy modules read the current traffic state and produce task assignments, paths, and vehicle-repositioning decisions. Vehicle and resource objects execute these decisions. Interfaces, input and output formats, and integration procedures are described in Appendix B; baseline rules and parameter settings are given in Appendix D.

Task dispatch. `AMHS` maintains pending tasks and vehicle states and assigns tasks to available vehicles when dispatch is invoked. Batch dispatch uses `hungarian_dispatcher()` to construct a cost matrix between vehicles and tasks and solve a one-to-one matching problem. Immediate dispatch processes pending tasks individually and selects a vehicle for the current task. The invocation times of both modes follow the main loop in Section A.3. Assignment costs can use static shortest-path costs or travel-time estimates supplied by the routing policy. This setting is recorded in the experiment configuration to specify the relationship between task assignment and routing decisions.

Routing interface and policy creation. The platform uses `RoutingStrategy` as the common routing interface and supplies network queries, the clock, parameters, and random streams through a routing context. Internally, the simulator looks up the selected policy class, constructs an instance, and initializes it as illustrated below. Names such as `create_router`, `_REGISTRY`, and `register` in this sketch are internal notation. User code registers a custom class with `amhslab.register_routing_strategy` and starts it with `amhslab.run_simulation` in the same Python process (Appendix B.5.1).

Implementation sketch A.2. Routing-policy creation and registration.

```
def create_router(
    method: str, ctx: RoutingContext, **extra
) -> RoutingStrategy:
    ...  # Check that the policy name is registered
    cls = _REGISTRY[method]
    router = cls(ctx, **extra)
    router.init()
    return router

register("D", DijkstraStrategy)
register("KD", KShortestStrategy)
register("DQN", MLPDQNRoutingStrategy)
```

Path execution and runtime feedback. During task execution, `OHT` requests a path through `compute_route()` and feeds observed track-segment traversal times back through the policy's `update()` method. `RoutingStrategy` also provides entry points for travel-time estimation, replanning decisions, and optional policy-state persistence. Persistence records the algorithm's own

state, while traffic entities and the execution kernel maintain simulation state. Routing algorithms therefore share the same path-execution and feedback process.

Regional admission control. Area-control policies determine the admission state of controlled regions. `WaterLevelArea` maintains the associated queues, permissions, and occupancy information; resource interactions are described in Section A.5.

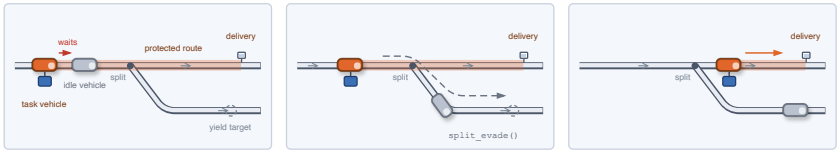
Empty-vehicle rebalancing. `AreaRebalanceController` coordinates empty-vehicle repositioning according to regional supply and demand through three components. The trigger, `RebalanceTrigger`, determines when to recompute. The count policy, `RebalanceCountScheme`, determines how many vehicles each region should receive or supply. The executor, `RepositionExecutor`, converts these counts into region matching, vehicle selection, and physical movements.

Implementation sketch A.3. Regional-rebalancing call sequence.

```
def update(self) -> Mapping[int, int] | None:
    trigger_view = RebalanceTriggerView(now=float(self.amhs.env.now))
    if not self.trigger.should_fire(trigger_view):
        return None
    snapshot = build_rebalance_snapshot(self.amhs, self.areas)
    counts = self.count_scheme.compute(snapshot)
    self.executor.apply_counts(counts)
    return counts
```

The controller first checks the trigger and traverses tasks and vehicles to construct a `RebalanceSnapshot` only when the trigger fires. The snapshot is passed to the count policy and consists of frozen data classes and tuples of regional states, providing the input for the current decision. Once the count policy returns the repositioning counts, the executor arranges vehicle movements. Repositioned vehicles remain subject to following constraints and resource permissions along their routes.

Local yielding by idle vehicles. When an idle vehicle blocks a vehicle serving a task, the platform invokes a yielding policy through `OHT.set_driveaway_target()`. The function `split_evade()` searches for a split along the task route to be protected and directs the blocking vehicle onto another available branch, clearing a passage for the task vehicle. `OHT` executes the yielding path subject to following and resource constraints along the route.



(a) An idle vehicle blocks a task vehicle (b) The idle vehicle takes the other branch (c) The task vehicle resumes its route

Figure 9: Local yielding by an idle vehicle. (a) An idle vehicle stands on the protected route of a task vehicle. (b) split_evade() sends it onto a branch outside the protected route, toward a legal yield target. (c) The task vehicle resumes its route. Both vehicles remain subject to the common following and resource constraints.

A.5 Shared Resources and Access Coordination

Resource objects and responsibilities. The platform uses `Resource` as the basis for requests, queues, grants, and releases in shared space. `MergeResource` manages directional queues and access grants at merges, while `WaterLevelArea` manages entry permissions and occupancy in controlled regions. `ControlNode` extends `MergeResource`, associating node topology with merge-resource functionality. Vehicles using shared space must satisfy both the relevant resource permissions and physical following constraints.

Merge-request representation. Each `MergeClaim` records the resource node, incoming direction, request identifier, and route generation for one merge request, together with its grant status and the vehicle's progress through the merge.

Implementation sketch A.4. Key fields of a merge request.

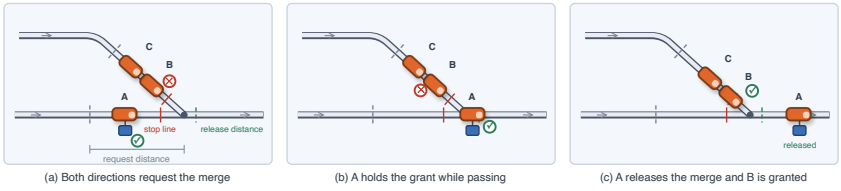
```
@dataclass
class MergeClaim:
    node_id: str
    request_id: str
    incoming_node: Optional[str]
    route_generation: int
    ...  # Request sequence, timestamp, and related fields
    active: bool = True
    granted: bool = False
    passing_distance: Optional[float] = None
    ...  # Auxiliary logging fields
```

The `request_id` identifies a request and remains unchanged throughout its lifecycle. Grant, revocation, and release callbacks use this identifier to address the corresponding request, preventing delayed callbacks from an old request from overwriting a subsequently created request. The field `incoming_node` records the request's incoming direction.

Merge grants and resource release. When processing requests, the merge resource checks request validity, available forward space, and the relevant area permissions, then selects among eligible queue heads from different directions. The default grant rule rotates across directions and prioritizes vehicles serving tasks when they are among the eligible heads. After passing the merge and reaching the configured release distance, the vehicle releases the resource for subsequent requests.

Area admission and occupancy updates. `WaterLevelArea` maintains the controlled region's nodes and boundary, physical members, entry queues, and active permissions, and coordinates entry grants according to the area-control state. After granting entry, the region object tracks the vehicle's approach, physical entry, and departure, maintaining permission and occupancy information separately. Cancellation updates the corresponding reservation and associated state. When a path involves multiple controlled regions, each region maintains its own state, and the vehicle must satisfy the respective permission requirements.

Route changes and request consistency. When replanning or yielding changes a vehicle's path, the platform removes invalidated requests through a common route-replacement and request-reconciliation process. If a vehicle changes its incoming edge to the same merge, the platform ends the old request and creates a new one so that the request record matches the actual approach direction. Valid grants needed by vehicles already inside a control zone or passing through a merge are retained, preventing route changes from interrupting an access movement already in progress.



(a) Both directions request the merge (b) A holds the grant while passing (c) A releases the merge and B is granted

Figure 10: Access coordination at a merge, with schematic distances. Vehicles request the merge within the request distance, stop at the stop line before the merge without a grant, and release it after the release distance (Table 6). (a) A holds the grant while B waits at the stop line and C queues behind B. (b) A keeps the grant while crossing. (c) A releases the merge and the grant passes to B.

A.6 Runtime Monitoring and Statistics

Task execution records. The platform updates execution records when vehicle and task states change. `TransportationTask` stores task release, assignment, vehicle arrival at pickup and

delivery locations, and loading and unloading completion times, together with the track segments actually traversed. These records follow the vehicle's actual execution and support analyses of task times, actual travel, and congestion.

Runtime metric aggregation. `AMHS` aggregates task-completion records and periodically samples vehicle motion, waiting, and queue states. Task-completion counts and stage timestamps support throughput and transport-time metrics, while state samples support vehicle-utilization and waiting statistics. Metrics and event records are associated with simulation time so that policy performance can be analyzed alongside local traffic states. Metric definitions are given in Appendix D.

Identifying wait dependencies. `AMHS.detect_deadlock_cycles()` constructs a wait-for graph from physical following and shared-resource dependencies, including waiting at merges and controlled-region entrances. The detector scans every $5\,\mathrm{s}$, selects stopped vehicles with absolute speed below $0.05\,\mathrm{m/s}$, and excludes legitimate service stops such as loading and unloading. Physical-following edges additionally require a net gap below $2\,\mathrm{m}$. The detector then extracts mutually waiting vehicle sets through strongly connected components and passes them to the monitor for persistence confirmation.

Confirming persistent circular wait. `DeadlockMonitor.observe()` tracks waiting duration by vehicle set and excludes sets whose members are all idle, without tasks or movement objectives. An event is confirmed only when the same vehicle set remains in circular wait across consecutive scans, every member satisfies the stopping condition, and the duration reaches at least $100\,\mathrm{s}$. The same persistent set is not reported repeatedly. If circular wait dissolves and later reappears, the monitor restarts timing.

Event tracking and outcome analysis. After confirmation, the platform records the relevant vehicles, paths, resources, and wait dependencies, continues the simulation, and tracks subsequent resolution of circular wait. Task-completion records are used to assess whether the event is accompanied by throughput collapse, while event records locate the formation and resolution of waiting. Task records, runtime metrics, and resource events can be exported with simulation results for subsequent evaluation and diagnosis.

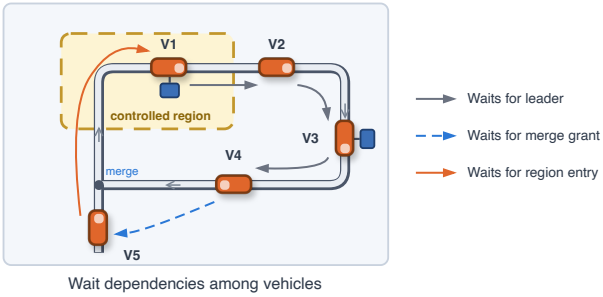


Figure 11: Wait dependencies in a circular wait. Stopped vehicles V1–V5 wait on leaders, a merge grant, and region entry, forming a cycle in the wait-for graph. The detector extracts such cycles as strongly connected components and confirms an event only after the same set remains stopped for at least $100\,\mathrm{s}$.

B System State and Customizable Policies

This appendix specifies the system state available to user-defined policies in AMHSTrafficLab, together with the inputs, outputs, and invocation rules for routing and regional-control policies. These interfaces support the design and comparison of algorithms under common vehicle-motion, task-service, and resource-management rules and can be used after installation of the compiled AMHSTRAFFICLAB wheel. Scenario files and region definitions are described in Appendix C, baselines and experimental settings in Appendix D, and installation and reproduction procedures in Appendix G.

B.1 Observable System State

User-defined policies access the current network, vehicle, task, and regional states through public system objects. Routing policies use `ctx.amhs`, whereas regional-control policies receive `amhs` as an input argument. Public fields are read-only: policies may use them to construct adjacency lists, estimate congestion, or maintain learning state, but do not directly modify vehicle positions, task assignments, resource queues, or simulation time. Route replacement and regional control are executed by the simulator through the interfaces in Sections B.2–B.4.

Object identifiers are strings. Distances are measured in meters, speeds in meters per second, and times in simulation seconds. Optional objects and event times are represented by `None` when absent. Static attributes are established when the scenario is loaded; dynamic attributes are updated as vehicles move, task events occur, or resource states change. Tables 7–9 organize the fields by their use. Sequences, sets, and mappings in these tables denote public read-only containers.

The public graph supports node and directed-adjacency queries through `node_ids()`, `has_node(u)`, `neighbors(u)`, and `has_edge(u,v)`. Edge weights are accessed through `edge_attribute(u,v,"weight")`.

Regional `occupancy` differs from `len(members)`: the former also counts vehicles that hold an entry grant and have entered the entrance control distance, counting each vehicle only once; the latter includes only vehicles physically inside the region.

Table 7: System access and network topology.

Object	Field or method	Type or unit	Meaning and updates
AMHS	sim_time	float, s	Current simulation time; advances with the simulation.
AMHS	ohts	Vehicle sequence	Current fleet; vehicles can be retrieved by ID.
AMHS	nodes, rail_paths	Node mapping; track sequence	Nodes and tracks in the scenario.
AMHS	transport_tasks	Task sequence	Tasks already generated in the current run; policies can filter pending, active, or finished tasks by status.
AMHS	areas, rebalance_areas	Region sequences	All runtime regions and the subset configured for empty-vehicle rebalancing.
AMHS	get_node, get_oht, get_area	Object queries	Retrieve an object in the current run by its ID.
AMHS	graph	Graph query object	Network nodes, directed connections, and edge weights.
ControlNode	id, position, type	str; coordinates (m); type value	Node identifier, spatial position, and type.
ControlNode	is_parking, is_join, is_split	int	Parking priority and merge/split indicators; is_parking=−1 denotes a non-parking node.
ControlNode	incoming_paths, outgoing_paths	Track sequences	Incoming and outgoing tracks.
RailPath	id, start_node, end_node	str; node objects	Track identifier and endpoints of the directed connection.
RailPath	length, max_speed	float, m and m/s	Track length and speed limit.
RailPath	weight	float	Public routing weight; its unit must match the chosen cost definition.
RailPath	type, direction	str	Track type and direction specified by the scenario.
RailPath	status, current_occupancy, ohts	Status; int; vehicle sequence	Track status, vehicle count, and vehicle order along the track.
RailPath	incoming_paths	Track sequence	Upstream tracks associated with the track entrance.

Table 8: Vehicle and transport-task state.

Object	Field	Type or unit	Meaning
OHT	id, status	str; status	Vehicle identifier and operating state.
OHT	current_node, path	Node ID; track or None	Current node record and the track occupied by the vehicle.
OHT	x, v, l	float, m, m/s, m	Position along the track, speed, and vehicle length.
OHT	current_task, preassigned_task	Task or None	Current task and a preassigned subsequent task.
OHT	target_node	Node ID or None	Current transport, cruising, or rebalancing destination.
OHT	route	Node sequence	Stored remaining route; departed nodes are removed as the vehicle advances.
OHT	route_anchor	Node ID or None	First node of the remaining route (route[0]), updated as the vehicle advances or is rerouted; None when no route is stored.
OHT	waiting	bool	Whether the vehicle is waiting.
OHT	current_area_ids	Set of IDs	Regions physically occupied by the vehicle.
OHT	requested_area_ids, granted_area_ids	Sets of IDs	Regions for which entry has been requested or granted, respectively.
Transportation Task	id, start_location, end_location	str	Task identifier, pickup node, and delivery node.
Transportation Task	status	Status	Created, preassigned, executing, completed, or failed.
Transportation Task	create_time, start_time, pickup_time, end_time	float or None, s	Task creation, start, pickup, and completion times.
Transportation Task	assigned_oht, preassigned_oht	Vehicle ID or None	Assigned and preassigned vehicles.

Table 9: Regions, waiting requests, and entry grants.

Object	Field	Type or unit	Meaning
`WaterLevelArea`	`id`, `area_index`	`str`; `int`	Region identifier and stable index within the current run.
`WaterLevelArea`	`node_ids`, `entry_edges`, `exit_edges`	Node set; directed-edge sequences	Region extent, entrances, and exits; edge endpoints are node IDs.
`WaterLevelArea`	`capacity`	`int`, vehicles	Physical capacity of the region.
`WaterLevelArea`	`low_water`, `high_water`	`int`, vehicles	Thresholds for quota control and suspension of new entry grants under automatic control.
`WaterLevelArea`	`occupancy`	`int`, vehicles	Occupancy used for admission control; updated with vehicle and grant states.
`WaterLevelArea`	`members`	Vehicle set	Vehicles physically inside the region.
`WaterLevelArea`	`controllable`	`bool`	Whether a policy may override the region's admission mode.
`WaterLevelArea`	`admission_mode`, `admission_override`	Admission enum; override may be `None`	Effective mode and persistent override set by the user.
`WaterLevelArea`	`rebalance_stock`	`int` or `None`, vehicles	Target vehicle reserve; `None` excludes the region from empty-vehicle rebalancing.
`WaterLevelArea`	`waiting_queue`, `grants`	Request and grant sequences	Requests awaiting entry grants and currently valid grants.
`WaterLevelArea`	`next_grant_entry_index`	`int` or `None`	Entrance considered first in the next allocation round under quota control.
`WaterLevelArea`	`queue_version`, `grant_version`	`int`	Version counters updated when the waiting queue or grants undergo the corresponding changes.
`AreaWaitingEntry`	`request_id`, `oht`, `entry_edge`	`int`; vehicle; directed edge	Request identifier, requesting vehicle, and entry edge.
`AreaWaitingEntry`	`request_time`, `wait_time`, `service_rank`	`float` (s); `float` (s); `int`	Request time, elapsed waiting time, and position within the entrance queue.
`AreaGrantEntry`	`grant_id`, `oht`, `entry_edge`	`int`; vehicle; directed edge	Grant identifier, holder, and entry edge.
`AreaGrantEntry`	`granted_time`, `age`, `state`	`float` (s); `float` (s); status	Grant time, elapsed grant duration, and reserved or entering state.
`AreaGrantEntry`	`rotation_deadline`	`float` or `None`, s	End of the current time slice for a grant eligible for rotation under quota control.

B.2 ROUTING-POLICY INTERFACE

```
# Interface pseudocode: the wheel supplies this context.
# The platform creates and supplies a context for each simulation.
class PublicRoutingContext:
    amhs: AMHS                        # Read-only system state
    graph: ReadOnlyGraph               # Topology and edge weights
    env: ClockView                    # Read-only clock; seconds
    routing_targets: tuple[str, ...] # Target node IDs

    alpha: float                      # Policy parameter
    gamma: float                      # Policy parameter
    tau: float                        # Policy parameter
    extra: Mapping[str, object]       # Options; containers are copied
                                      # and recursively frozen

    py_rng: random.Random             # Policy's Python RNG
    np_rng: numpy.random.Generator    # Policy's NumPy RNG

    @property
    def now(self) -> float: ...       # Current time: env.sim_time


# Constructor pattern for a user-defined class.
class MyRoutingStrategy(RoutingStrategy):
    def __init__(self, ctx: PublicRoutingContext, **options):
        super().__init__(ctx, **options)  # Stores self.ctx
```

Table 10: The four required methods of a routing policy.

Method	Inputs	Output or purpose
`compute_route(start_node, target_node, max_hops=None)`	Planning origin, destination, and an optional search-budget hint.	Complete node list containing both endpoints.
`update(task_id, dest_nodes, from_node, to_node, observed_time, next_node=None, agent_id=None)`	Task ID, associated destinations, traversed edge, observed traversal time, and optional successor node and vehicle ID.	Update policy statistics or learned parameters.
`estimate_travel_time(start_node, end_node)`	Two nodes.	Non-negative Python `float`, or `math.inf` for an unreachable pair.
`estimate_task_time(current_node, pickup_node, dropoff_node)`	Current location, pickup node, and delivery node.	Estimated pickup-and-delivery cost, with the same return-type requirements as above.

Let s denote the planning origin, g the destination, and $p = (v_0, \ldots, v_k)$ the returned route. A valid route satisfies $v_0 = s$ and $v_k = g$, with a directed track connecting every pair of consecutive nodes. The return value must be a Python `list` whose elements are string node IDs from the map. When $s = g$, the policy returns `[s]`. The interface accepts neither a next-hop action nor a truncated route.

The platform determines the planning origin from the vehicle's physical position at the time of the decision. A vehicle at a node plans from that node; an additional replanning request issued while the vehicle is traversing a track starts from that track's endpoint. Each call to `compute_route` uses its supplied `start_node`, which must be included in the returned complete node list.

A custom policy is registered by calling `register_routing_strategy` with a `name`, a `StrategyClass`, and optional `conditions`. The simulation runner selects the registered policy in the same Python process.

B.3 REGIONAL-CONTROL INTERFACE

Regional control comprises admission control and empty-vehicle rebalancing. Both can access the state in Section B.1, but act on different decisions: admission control determines whether entry requests receive grants, whereas rebalancing determines how many vehicles each region needs to receive or can supply.

B.3.1 REGION SELECTION AND ADMISSION CONTROL

All runtime regions are accessible through `amhs.areas`. The `controllable` field indicates whether a user policy may override a region's admission mode and can be used to select the regions to control. Region geometry and base water-level parameters are specified by the scenario. At runtime, the following methods change admission behavior:

- `amhs.set_admission(area,mode)` sets a persistent admission-mode override.
- `amhs.clear_admission_override(area)` removes the override and restores automatic water-level control.

These methods may be called only during the policy stage, within `AreaControlPolicy.execute(amhs,rebalance_scheme)`. An override persists after the method returns and must be explicitly cleared to restore automatic control.

Table 11: Regional admission modes.

Python enum member	Meaning
`AdmissionMode.OPEN`	Allocates grants to eligible entrance-queue heads without using the water-level margin to limit grants outside the entrance control distance. Vehicles remain subject to the shared right-of-way and motion rules.
`AdmissionMode.QUOTA`	Limits grants outside the entrance control distance according to the remaining admission quota and rotates allocations among entrances. Its enum value is `quota`.
`AdmissionMode.HOLD_NEW_ENTRIES`	Issues no new grants. Retains qualifying grants held by queue-head vehicles near the entrance or entering the region, and withdraws other revocable grants.

Changing an admission mode does not remove vehicles or change their positions directly. The platform manages waiting requests, grant retention and withdrawal, and physical entry. Policies may use `waiting_queue`, `grants`, and grant-rotation times to define their rules, but cannot directly reorder queues, issue grants, or modify region membership.

B.3.2 EMPTY-VEHICLE REBALANCING

`RebalanceCountScheme.compute(amhs)` returns a mapping from region objects to integer vehicle counts. For a region a, a positive value $r_a > 0$ requests vehicles, a negative value $r_a < 0$ indicates vehicles available to supply, and $r_a = 0$ indicates no net rebalancing demand.

Returned regions must belong to `amhs.rebalance_areas` in the current run, namely regions whose `rebalance_stock` is not `None`.

B.3.3 MAP EDITOR

We provide a map editor for visually editing control regions. Figure 12 shows an example of the editor. The editor supports reading in an existing region list (for example, the ones provided in the baseline) and editing, or exporting to a new list. Users select tracks to specify region membership and configure the region identifier, admission water level, quota threshold, and optional rebalance threshold. The editor highlights the member tracks of the selected region and supports overlapping and nested regions.

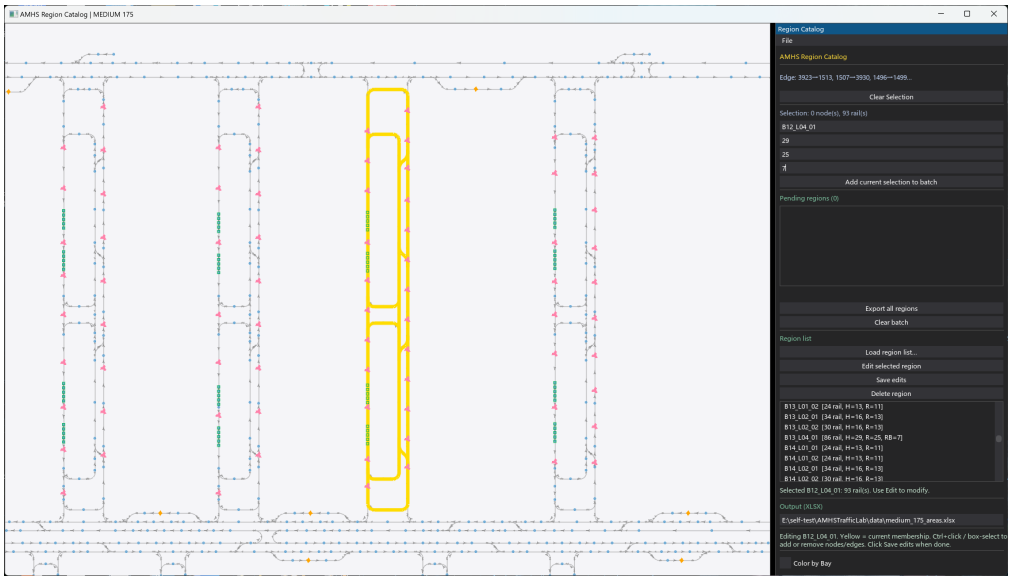


Figure 12: An example of the map editor interface using the MEDIUM 175 layout. Yellow tracks indicate the track members of the selected region, `B12_L04_01`. The side panel displays the region list and configuration fields; the selected region has a water level of 29, a quota threshold of 25, and a target reserve of 7 empty vehicles.

B.4 Policy Invocation and Decision Handling

Route computation is triggered by task execution and can additionally be triggered by user-defined conditions. The platform requests a route when needed for a new task or destination, or when the current route is empty or can no longer be used. These required calls do not wait for a user-defined condition to hold.

A routing policy may also request replanning at node departures through `should_replan_at_node(node,oht)`. Before each vehicle-specific routing decision, replanning check, or edge-feedback update, the platform supplies the corresponding read-only vehicle view through `set_decision_context(oht)`. Returning the Python value `True` from `should_replan_at_node` requests a new complete route from the current node. It does not bypass route validation, regional admission control, or the vehicle-motion rules. The default implementation returns `False`.

Learning policies that require observations beyond routing decisions may register a synchronous callback through `register_simulation_observer(callback,every_n_frames)`. The callback receives the read-only system state after every specified number of completed frames. With `every_n_frames=1`, the benchmark's 30-Hz simulation provides one post-frame observation per completed frame. Policies must copy any values that they intend to retain, because the public objects are live views of the current state.

Additional replanning conditions use one of the following signature templates:

```
def global_condition(amhs) -> bool:
    ...

def per_oht_condition(amhs, oht) -> bool:
    ...
```

The global condition is checked once at each integer simulation second and must return a Python `bool`. Each evaluation that returns `True` produces a new replanning request.

Regional-control conditions use `area_condition(amhs)->bool`. The platform checks this condition each simulation frame (30 Hz at the fixed 1/30-s step). A `True` result triggers one execution of the regional-control policy; a condition that remains true continues to trigger execution each frame.

B.5 Minimal Policy Examples

B.5.1 An Independent Dijkstra Implementation

The following example implements Dijkstra's algorithm using a priority queue and public graph queries for directed adjacency and edge weights. The base weights are non-negative track lengths, so the returned path minimizes total distance. The two estimation methods convert distance to time using a common reference speed.

```
from heapq import heappop, heappush
from math import inf, isfinite
from amhslab import RoutingStrategy


class MyDijkstra(RoutingStrategy):
    def __init__(self, ctx, **options):
        super().__init__(ctx, **options)
        self.speed = float(ctx.extra.get("nominal_speed", 4.0))
        if not isfinite(self.speed) or self.speed <= 0:
            raise ValueError("nominal_speed must be finite and positive")

    def _search(self, start, target):
        graph = self.ctx.graph
        if not graph.has_node(start) or not graph.has_node(target):
            return [], inf

        distance, parent = {start: 0.0}, {}
        heap = [(0.0, start)]
        while heap:
            cost, node = heappop(heap)
            if cost != distance[node]:
                continue
            if node == target:
                route = [target]
                while route[-1] != start:
                    route.append(parent[route[-1]])
                return route[::-1], cost

            for next_node in graph.neighbors(node):
                weight = float(
                    graph.edge_attribute(node, next_node, "weight")
                )
                candidate = cost + weight
                if candidate < distance.get(next_node, inf):
                    distance[next_node] = candidate
                    parent[next_node] = node
                    heappush(heap, (candidate, next_node))
        return [], inf

    def compute_route(self, start_node, target_node, max_hops=None):
        return self._search(start_node, target_node)[0]

    def update(self, task_id, dest_nodes, from_node, to_node,
               observed_time, next_node=None, agent_id=None):
        pass

    def estimate_travel_time(self, start_node, end_node):
        return float(self._search(start_node, end_node)[1] / self.speed)

    def estimate_task_time(
        self, current_node, pickup_node, dropoff_node
    ):
        return (self.estimate_travel_time(current_node, pickup_node)
                + self.estimate_travel_time(pickup_node, dropoff_node))
```

The following code registers the policy name and selects it for a run within the same Python process:

```
from amhslab import register_routing_strategy, run_simulation

register_routing_strategy("my-dijkstra", MyDijkstra)
result = run_simulation(
    "examples/configs/synthetic.json",
    until=60.0,
    method="my-dijkstra",
)
```

B.5.2 NODE-LEVEL REPLANNING

The preceding shortest-path policy computes a route only when the simulator requires one. The following subclass additionally requests replanning whenever a vehicle departs from a branching node. The simulator then invokes `compute_route` with the current node and destination, and validates the returned complete route before installing it.

```
class ForkReplanningDijkstra(MyDijkstra):
    def __init__(self, ctx, **options):
        super().__init__(ctx, **options)
        self.decision_oht = None

    def set_decision_context(self, oht):
        # One strategy instance serves the entire fleet. This reference is
        # replaced before the next vehicle-specific routing operation.
        self.decision_oht = oht

    def should_replan_at_node(self, node, oht):
        # The comparison returns an exact Python bool.
        return len(self.ctx.graph.neighbors(node)) > 1


register_routing_strategy(
    "fork-replanning-dijkstra",
    ForkReplanningDijkstra,
)
result = run_simulation(
    "examples/configs/synthetic.json",
    until=60.0,
    method="fork-replanning-dijkstra",
)
```

Returning `False` does not suppress replanning required by the simulator, for example when a route is absent or no longer valid. Conversely, returning `True` changes only the stored route; it does not directly move the vehicle or change its admission state.

B.5.3 AN ONLINE LEARNING ROUTING POLICY

The example below learns an exponential moving average of traversal time for each directed edge. Route search uses the learned traversal times together with the latest observed edge occupancies. Edge feedback is delivered through `update`, whereas the frame observer supplies a copied snapshot of the global traffic state. The example also separates online training from frozen checkpoint evaluation.

```
import json
from heapq import heappop, heappush
from math import inf
from pathlib import Path

from amhslab import (
    RoutingStrategy,
    register_routing_strategy,
    register_simulation_observer,
    run_simulation,
)
```

```
frame_state = {"edge_occupancy": {}}


def observe_frame(amhs):
    # Copy scalar values. Retaining a public object would retain a live view.
    frame_state["edge_occupancy"] = {
        (path.start_node.id, path.end_node.id):
            int(path.current_occupancy)
        for path in amhs.rail_paths
    }


class OnlineTravelTimeRouter(RoutingStrategy):
    training = True
    checkpoint = None
    last_instance = None

    def __init__(self, ctx):
        super().__init__(ctx)
        self.alpha = float(ctx.alpha)
        self.nominal_speed = 4.0
        self.occupancy_penalty = 0.5
        self.edge_time = {}
        self.decision_oht = None

        if self.checkpoint is not None:
            self.load_checkpoint(self.checkpoint)
        type(self).last_instance = self

    def set_decision_context(self, oht):
        self.decision_oht = oht

    def should_replan_at_node(self, node, oht):
        return len(self.ctx.graph.neighbors(node)) > 1

    def _edge_cost(self, node, next_node):
        distance = float(
            self.ctx.graph.edge_attribute(node, next_node, "weight")
        )
        free_flow_time = distance / self.nominal_speed
        travel_time = self.edge_time.get(
            (node, next_node), free_flow_time
        )
        occupancy = frame_state["edge_occupancy"].get(
            (node, next_node), 0
        )
        return travel_time + self.occupancy_penalty * occupancy

    def _search(self, start, target):
        if start == target:
            return [start], 0.0

        distance, parent = {start: 0.0}, {}
        heap = [(0.0, start)]
        while heap:
            cost, node = heappop(heap)
            if cost != distance[node]:
                continue
            if node == target:
                route = [target]
                while route[-1] != start:
                    route.append(parent[route[-1]])
                return route[::-1], cost
```

```
            for next_node in self.ctx.graph.neighbors(node):
                candidate = cost + self._edge_cost(node, next_node)
                if candidate < distance.get(next_node, inf):
                    distance[next_node] = candidate
                    parent[next_node] = node
                    heappush(heap, (candidate, next_node))
        return [], inf

    def compute_route(self, start_node, target_node, max_hops=None):
        return self._search(start_node, target_node)[0]

    def update(self, task_id, dest_nodes, from_node, to_node,
               observed_time, next_node=None, agent_id=None):
        if not self.training:
            return
        key = (from_node, to_node)
        observation = float(observed_time)
        previous = self.edge_time.get(key, observation)
        self.edge_time[key] = (
            (1.0 - self.alpha) * previous
            + self.alpha * observation
        )

    def estimate_travel_time(self, start_node, end_node):
        return float(self._search(start_node, end_node)[1])

    def estimate_task_time(
        self, current_node, pickup_node, dropoff_node
    ):
        return float(
            self.estimate_travel_time(current_node, pickup_node)
            + self.estimate_travel_time(pickup_node, dropoff_node)
        )

    def save_checkpoint(self, path):
        records = [
            [start, end, value]
            for (start, end), value in sorted(self.edge_time.items())
        ]
        Path(path).write_text(json.dumps(records), encoding="utf-8")

    def load_checkpoint(self, path):
        records = json.loads(Path(path).read_text(encoding="utf-8"))
        self.edge_time = {
            (start, end): float(value)
            for start, end, value in records
        }
```

Training and frozen evaluation use distinct registered policy names. The observer must be registered in the same Python process as `run_simulation`; its registration is removed after each run.

```
def run_with_observer(method):
    frame_state["edge_occupancy"] = {}
    unsubscribe = register_simulation_observer(
        observe_frame,
        every_n_frames=1,
    )
    try:
        return run_simulation(
            "examples/configs/synthetic.json",
            until=600.0,
            method=method,
        )
    finally:
```

```
        unsubscribe()


register_routing_strategy(
    "online-travel-time-train",
    OnlineTravelTimeRouter,
)
run_with_observer("online-travel-time-train")
OnlineTravelTimeRouter.last_instance.save_checkpoint(
    "online-travel-time.json"
)


class FrozenOnlineTravelTimeRouter(OnlineTravelTimeRouter):
    training = False
    checkpoint = "online-travel-time.json"


register_routing_strategy(
    "online-travel-time-eval",
    FrozenOnlineTravelTimeRouter,
)
run_with_observer("online-travel-time-eval")
```

The checkpoint contains only algorithm-owned parameters. During frozen evaluation, edge-feedback calls are still delivered, but the policy does not update its learned values. The observer is read-only and cannot directly change vehicle, task, or network state.

B.5.4 Regional Admission and Empty-Vehicle Rebalancing

The regional policy below uses a simple high/low-water rule: it suspends new entry grants when occupancy reaches the high threshold, restores automatic admission when occupancy falls below the low threshold, and retains the existing control state between the thresholds. The rebalancing scheme computes each region's net demand by subtracting the observed number of candidate idle vehicles from its target reserve.

```
from amhslab import (
    AdmissionMode, AreaControlPolicy, RebalanceCountScheme,
)


class IdleStockScheme(RebalanceCountScheme):
    def compute(self, amhs):
        counts = {}
        for area in amhs.rebalance_areas:
            idle_count = sum(
                oht.status == "IDLE"
                and oht.current_task is None
                and oht.preassigned_task is None
                and oht.target_node is None
                and not oht.waiting
                for oht in area.members
            )
            counts[area] = int(area.rebalance_stock) - idle_count
        return counts


class WaterLevelPolicy(AreaControlPolicy):
    def execute(self, amhs, rebalance_scheme):
        for area in amhs.areas:
            if not area.controllable:
                continue
            if area.occupancy >= area.high_water:
                amhs.set_admission(area, AdmissionMode.HOLD_NEW_ENTRIES)
```

```
            elif area.occupancy < area.low_water:
                amhs.clear_admission_override(area)
        amhs.request_rebalance(rebalance_scheme.compute(amhs))


def area_condition(amhs):
    return True
```

C SCENARIO DESCRIPTIONS AND CONSTRUCTION GUIDE

The benchmark builds on transport between process areas, reentrant processing, and local material handling in wafer manufacturing. It comprises seven evaluation scenarios across three layout families: SMALL, MID, and LARGE. The families represent different transport demand structures through their process routes and route proportions. Scenarios within a family share track topology and route configuration, with fleet size and total demand defining different operating conditions.

C.1 SCENARIO COMPONENTS

Each scenario is defined by map information, a workload matrix, and region definitions. Map information describes the transport network and service locations, the workload matrix describes transport demand between bays, and region definitions describe local control regions and their configurations.

Map information. Map information comprises control nodes, tracks, bays, and service locations. Control nodes specify spatial coordinates and node types, while directed tracks specify connections between nodes and track lengths. Equipment, buffer, and storage locations are associated with control nodes and organized by bay. Together, these data determine network topology, geometric layout, and the distribution of service locations.

Workload matrix. Process routes and their proportions determine the relative numbers of transfers. Associating processing stages with hosting bays gives origin–destination demand, and production volume determines demand rates. The workload matrix uses bays in the map as origin and destination units. Each entry gives the nominal transport demand from one bay to another in moves/h. The matrix includes both intra-bay and inter-bay demand, describing total workload and its spatial distribution. In the current scenarios, it is represented by origin-bay, destination-bay, and demand-rate records in `FromToTable`. Sections C.4 and C.5 describe the associated process flows, workload construction, and wafer-volume conversion.

Region definitions. Region definitions specify local regions through track and node membership, together with region hierarchy, upper and lower water levels, and empty-vehicle target reserves. Regions may be nested or overlapping. Their members reference tracks and nodes in the map, linking regional configurations to the network topology.

The three components are linked through node, track, and bay identifiers; Sections C.2 and C.3 specify the fields and relationships. When extending a scenario, these relationships should be checked after changes to the layout, workload, or regional configuration.

C.2 DATA REPRESENTATION OF SCENARIO ELEMENTS

Map information is organized into four worksheets describing control nodes, directed tracks, service locations, and bay definitions. A fifth worksheet specifies the workload matrix. The following tables present each worksheet separately using consistent English field names.

C.2.1 CONTROL NODES: CONTROLNODE

The `ControlNode` worksheet records network nodes, their coordinates, and node attributes (Table 12).

Table 12: Fields of the `ControlNode` worksheet.

Field	Meaning
`id`	Control-node identifier.
`type`	Node category.
`x`, `y`	Spatial coordinates (mm).
`IsParking`	Parking-point attribute.

C.2.2 DIRECTED TRACKS: RAIL

The Rail worksheet records directed connections between control nodes, together with track categories, bay membership, and lengths (Table 13).

Table 13: Fields of the Rail worksheet.

Field	Meaning
id	Directed-track identifier.
Type	Track category.
start_node	Start node; references ControlNode.id.
end_node	End node; references ControlNode.id.
BayCode	Bay containing the track.
Length	Track length (mm).

C.2.3 SERVICE LOCATIONS: MAP_STATION_DATA

The map_station_data worksheet associates equipment, buffer, and storage locations with control nodes (Table 14). Multiple service-location records can reference the same control node, so service locations and control nodes are counted separately.

Table 14: Fields of the map_station_data worksheet.

Field	Meaning
StationCode	Service-location identifier.
PointCode	Associated node; references ControlNode.id.
BayCode	Bay annotation of the service location.
Type	Location category: equipment (EQ), overhead buffer (OHB), or stocker (STK).

C.2.4 BAY DEFINITIONS: BAY_DEFINITIONS

The Bay_Definitions worksheet identifies Bays and their categories (Table 15). These definitions and track-level Bay identifiers jointly determine Bay membership of control nodes.

Table 15: Fields of the Bay_Definitions worksheet.

Field	Meaning
BayCode	Bay identifier.
BayType	Bay category: INTRABAY or INTERBAY.

C.2.5 WORKLOAD MATRIX: FROMTOTABLE

The FromToTable worksheet specifies the workload of each scenario. Each row records one origin–destination bay pair and its nominal transport demand rate (Table 16). Bay identifiers in this matrix must be consistent with those in the map.

Records with identical FromBay and ToBay values represent intra-bay demand; other records represent inter-bay demand. Total workload and both fractions are computed by summing TaskArrive.

Table 16: Fields of the FromToTable worksheet.

Field	Meaning
FromBay	Origin Bay; references a bay identifier in the map.
ToBay	Destination Bay; references a bay identifier in the map.
TaskArrive	Nominal transport demand for this bay pair (moves/h).

C.3 Region Definition and Water-Level Configuration

Regional configuration comprises region attributes and their membership in the transport network. The Areas worksheet records region attributes, while AreaTracks and AreaNodes associate regions with tracks and nodes, respectively. At least one membership worksheet is required.

C.3.1 Region Attributes: Areas

The Areas worksheet defines each region's hierarchy, water levels, and empty-vehicle reserve (Table 17).

Table 17: Fields of the Areas worksheet.

Field	Meaning
AreaID	Region identifier, referenced by the membership worksheets.
Enabled	Whether the region is enabled.
Layer	Region hierarchy level.
WaterLevel	Configured upper water level (vehicles).
ReopenLevel	Configured lower water level (vehicles).
RebalanceStock	Target reserve of empty vehicles, used when regional rebalancing is enabled.

Enabled regions require both water levels, with the lower value no greater than the upper one. A blank reserve excludes a region from the rebalancing target set; an explicit zero specifies a target reserve of zero empty vehicles. Regions may be nested or overlapping, and water levels and reserves are recorded separately for each region. Section C.6 reports enabled-region counts, water-level ranges, and configured reserves.

C.3.2 Track Membership: AreaTracks

Each row in AreaTracks associates a region with a member track in the map (Table 18).

Table 18: Fields of the AreaTracks worksheet.

Field	Meaning
AreaID	Region; references Areas.AreaID.
RailID	Member track; references Rail.id.

C.3.3 Node Membership: AreaNodes

Each row in AreaNodes associates a region with a member control node in the map (Table 19).

Table 19: Fields of the `AreaNodes` worksheet.

Field	Meaning
`AreaID`	Region; references `Areas.AreaID`.
`NodeID`	Member node; references `ControlNode.id`.

C.4 Process Flows and Workload Construction

All three layout families use the same workload construction: specify process routes, assign a proportion to each route, distribute its processing stages across the bays that host them, and aggregate transfers between successive stages into a From–To matrix. Each family's workload composition is specified by its route set and route proportions.

C.4.1 Process Routes

A process route describes successive transfers between processing stages. Routes include transfers between process areas, reentry into previously visited areas, and local equipment–buffer circulation. Routes can be concatenated or repeated during production, with every transport leg contributing to the workload.

SMALL. SMALL represents transfers between deposition, photolithography, cleaning, and implantation. Its routes identify individual process units, including three representative routes:

- S_1: PVD_2 → PVD_PTCH → PTCH_2 → PTCH_PHOTO → PHOTO_2 → PHOTO_CVD → CVD_1 → DIFFCLEAN_3_SINGLE → DIFFCLEAN_IMPT → IMPT_1 → PVD_2;
- S_2: PVD_1 → IMPT_1 → PVD_OUT → PVD_1;
- S_3: PHOTO_2 → PHOTO_CVD → CVD_1 → DIFFCLEAN_4 → PHOTO_2.

The complete route mixture also includes return trips between process units, transfer segments connecting processing stages, and local circulation within individual units.

MID and LARGE. MID and LARGE cover wet processing (WET), diffusion and thermal treatment (DF), thin-film deposition (TF), photolithography (PHOTO), etching (ETCH), and chemical mechanical planarization (CMP). They use four cross-process routes:

- R_1: WET → DF → WET;
- R_2: WET → TF → PHOTO → ETCH → WET;
- R_3: WET → TF → CMP → WET;
- R_4: WET → PHOTO → ETCH → WET.

These routes capture transfers associated with thermal treatment, deposition and pattern transfer, deposition and planarization, and photolithography and etching, respectively. Each process category also has a local equipment → buffer → equipment route. MID and LARGE use different route proportions to describe the relative intensity of these process connections and local circulation.

C.4.2 Route Proportions

Let $\mathcal{R}$ denote the route set, q_r the proportion of route occurrences assigned to route r, and ℓ_r its number of transport legs. The proportions satisfy

$$q_r \geq 0, \qquad \sum_{r \in \mathcal{R}} q_r = 1, \qquad \bar{\ell} = \sum_{r \in \mathcal{R}} q_r \ell_r.$$

Route r therefore contributes a fraction $q_r \ell_r / \bar{\ell}$ of all moves, accounting for both its frequency and its length.

The complete SMALL configuration contains 82 routes, with a mean of 2.467063 transport legs per route occurrence. The accompanying route configuration specifies every process-unit sequence

Table 20: Route proportions in SMALL. Grouped rows sum the proportions of their constituent routes.

Route or route group	Routes	Proportion
S_1: deposition, lithography, cleaning, and implantation cycle	1	1.404855%
S_2: deposition and implantation branch	1	1.709812%
S_3: lithography, deposition, and cleaning reentry	1	1.154723%
Other closed routes between process units	53	70.705340%
Local circulation within process units	14	24.997858%
Transfer segments between processes	12	0.027412%

and its precise proportion. Weighting by route length gives 20.265278% intra-bay moves and 79.734722% inter-bay moves.

Table 21: Route proportions in MID and LARGE. Each local route contains two transport legs.

Route	Legs ℓ_r	MID q_r	LARGE q_r
R_1: WET $\to$ DF $\to$ WET	2	$1/18$	$1/18$
R_2: WET $\to$ TF $\to$ PHOTO $\to$ ETCH $\to$ WET	4	$1/18$	$1/18$
R_3: WET $\to$ TF $\to$ CMP $\to$ WET	3	$1/18$	$1/18$
R_4: WET $\to$ PHOTO $\to$ ETCH $\to$ WET	3	$1/18$	$1/18$
WET local circulation	2	$28/108$	$504/4779$
DF local circulation	2	$7/108$	$294/4779$
TF local circulation	2	$14/108$	$770/4779$
PHOTO local circulation	2	$14/108$	$1015/4779$
ETCH local circulation	2	$14/108$	$889/4779$
CMP local circulation	2	$7/108$	$245/4779$

Both MID and LARGE have $\bar{\ell} = 20/9$. The four cross-process routes contribute 30% of all moves, and the six local routes contribute 70%. The distribution of local circulation among processes follows their respective route proportions.

C.4.3 Bay Allocation and the From–To Matrix

Processing stages are associated with hosting bays through allocation groups. Let $\mathcal{B}_u$ denote the bay set for allocation group u, and let e_i be the equipment-location count in bay i. Its allocation weight is

$$a_{ui} = \frac{e_i}{\sum_{j \in \mathcal{B}_u} e_j}, \qquad i \in \mathcal{B}_u.$$

Equipment counts are obtained from EQ records in `map_station_data`. A stage assigned to a specific process unit has a singleton bay set and weight one. A stage hosted by multiple bays is expanded using these weights. Stages in the same allocation group share one bay within a route; local processing and buffer stages belong to the same group. Table 22 lists the hosting sets for MID and LARGE; individual process units in SMALL routes map to their configured bays.

Table 22: Process categories and hosting bays in MID and LARGE.

Process	MID	LARGE
WET	B06, B07, B12, B13, B23, B29, B30, B36	B12, B13, B29, B30
DF	B17, B24	B10, B31
TF	B14, B18, B26, B31	B08, B11, B16, B25, B32, B35
PHOTO	B10, B11, B27, B28, B33	B07, B15, B18, B23, B26, B33, B36
ETCH	B09, B15, B32, B34	B06, B14, B17, B24, B27, B34
CMP	B08, B16, B25	B09, B28

Let $A_{ij}^{(r)}$ be the weighted number of $i \to j$ moves contributed by route r after bay allocation, so that $\sum_{i,j} A_{ij}^{(r)} = \ell_r$. Transfers between distinct allocation groups are allocated by the product of origin and destination bay weights; local circulation within one bay contributes to the corresponding diagonal entry. For total demand Λ, the final matrix is

$$\lambda_{ij} = \frac{\Lambda}{\bar{\ell}} \sum_{r \in \mathcal{R}} q_r A_{ij}^{(r)}, \qquad \sum_{i,j} \lambda_{ij} = \Lambda.$$

All three layout families use this expression. The SMALL, MID, and LARGE route mixtures cover 18, 26, and 27 demand bays, yielding 87, 238, and 223 positive-demand bay pairs, respectively. Substituting each scenario's total demand and storing λ_{ij} in `TaskArrive` of `FromToTable` gives the scenario workload.

C.5 Equivalent Production Scale

Production volume determines how often the route mixture repeats. Across all seven scenarios, we use 25 wafers per lot, 720 hours per month, and an average of 1,000 moves per lot as the production-scale conversion convention. For m average moves per lot, route r occurs $mq_r/\bar{\ell}$ times per lot on average. Weighting these occurrences by route length gives m moves, linking route proportions, per-lot handling, and monthly wafer starts.

For w wafers/lot, m moves/lot, H hours/month, and total transport demand Λ in moves/h,

$$\Lambda = \frac{Q_{\text{eq}}}{wH}\, m, \qquad Q_{\text{eq}} = \frac{\Lambda w H}{m},$$

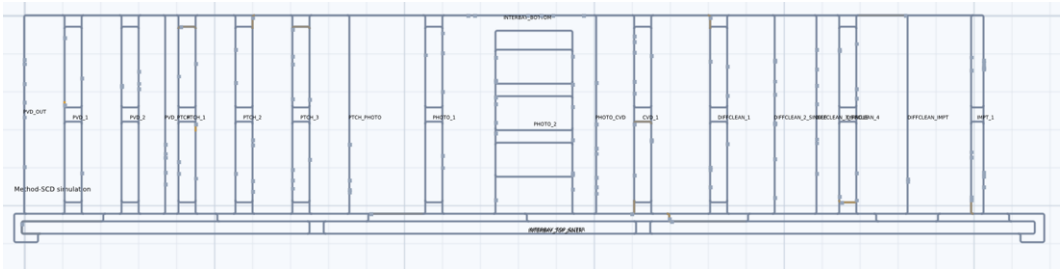
where Q_{eq} is measured in wafers/month. Substituting $w = 25$, $m = 1000$, and $H = 720$ gives $Q_{\text{eq}} = 18\Lambda$.

Table 23: Transport demand and equivalent monthly wafer starts.

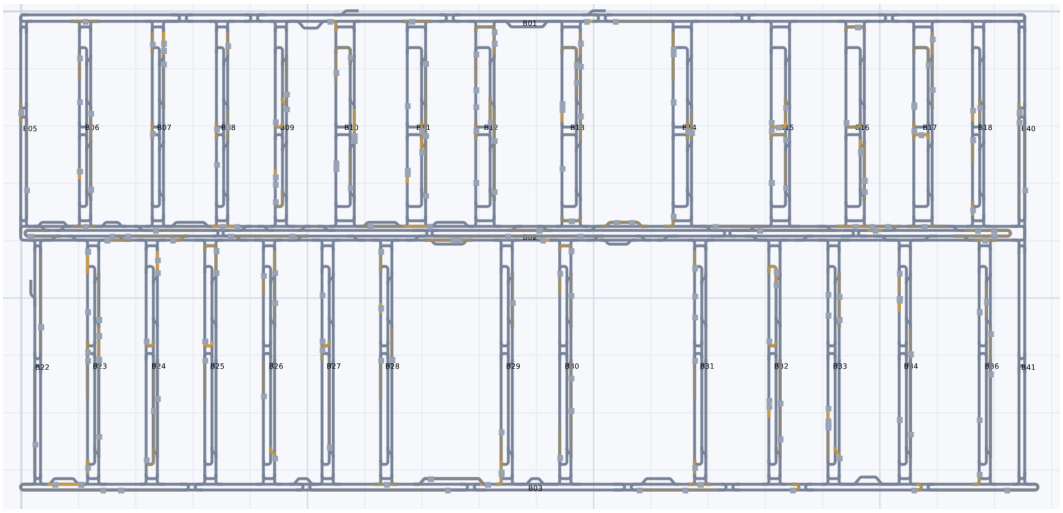
Scenario	Demand (moves/h)	Equivalent wafer starts (wafers/month)
SMALL 100	2,400	43,200
SMALL 150	3,120	56,160
MID 175	4,850	87,300
MID 200	5,350	96,300
MID 225	6,100	109,800
LARGE 275	6,850	123,300
LARGE 300	7,350	132,300

C.6 Scenario Statistics

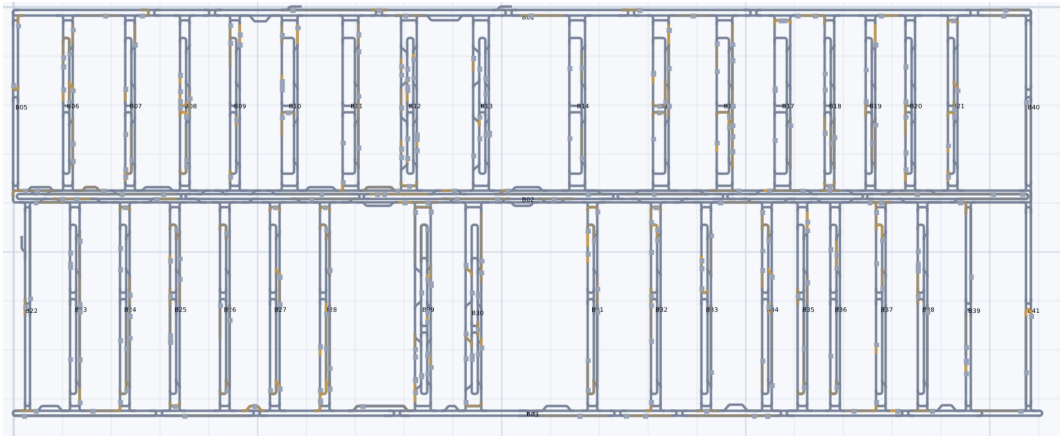
Figure 13 shows the track layouts of the SMALL, MID, and LARGE families. Scenarios within each family share the same layout.



(a) SMALL



(b) MID



(c) LARGE

Figure 13: Track layouts of the three benchmark families, shown in the simulator: (a) SMALL, (b) MID, and (c) LARGE. Bay labels identify the process areas and transport corridors.

C.6.1 TOPOLOGY AND SERVICE LOCATIONS

Table 24: Static structural statistics of the three layout families.

Family	Control nodes	Directed tracks	Total track length (m)	Bays	Service-location records	Equipment EQ / buffer OHB / storage STK
SMALL	3,334	3,419	3,012.289	21	3,239	314 / 2,925 / 0
MID	2,782	3,174	5,755.836	33	1,586	532 / 1,038 / 16
LARGE	3,522	4,014	7,114.642	40	2,246	531 / 1,698 / 17

Total track length is the sum of directed-track lengths in `Rail`, converted from mm to m. Service locations are counted as valid records; multiple records can reference the same control node. Scenarios in a family share this static structure.

C.6.2 DEMAND STRUCTURE AND SPATIAL LOAD

Table 25: Normalized demand structure of each layout family.

Family	Bay pairs with positive rates	Intra-bay task fraction	Inter-Bay task fraction
SMALL	87	20.27%	79.73%
MID	238	70.00%	30.00%
LARGE	223	70.00%	30.00%

Demand fractions are computed by summing bay-pair rates in `FromToTable`. Fleet density is fleet size divided by total track length in km; nominal demand per vehicle is total transport demand divided by fleet size.

Table 26: Fleet density and nominal task intensity.

Scenario	Fleet density (vehicles/km)	Nominal task intensity (tasks/vehicle/h)
SMALL 100	33.20	24.00
SMALL 150	49.80	20.80
MID 175	30.40	27.71
MID 200	34.75	26.75
MID 225	39.09	27.11
LARGE 275	38.65	24.91
LARGE 300	42.17	24.50

C.6.3 WATER-LEVEL REGION STATISTICS

The SMALL, MID, and LARGE scenarios contain 56, 186, and 224 enabled regions, respectively. Table 27 summarizes their configured water-level ranges and reserves.

Region counts include enabled records, and the ranges give their minimum and maximum configured water levels. Reserve statistics cover enabled regions with a specified `RebalanceStock`. Overlapping regions are counted separately, and the reserve sum aggregates their configured values.

Table 27: Configured water-level ranges and reserves for each scenario. Water levels and reserves are measured in numbers of vehicles.

Scenario	Configured lower levels	Configured upper levels	Enabled regions with reserves	Sum of configured reserves
SMALL 100	18–96	22–100	18	115
SMALL 150	18–96	22–100	18	115
MID 175	7–122	9–126	30	164
MID 200	7–122	9–126	30	183
MID 225	7–122	9–126	30	203
LARGE 275	5–146	7–150	37	240
LARGE 300	5–146	7–150	37	257

D Baseline Algorithms and Evaluation Protocol

This appendix describes the decision mechanisms of the routing and regional-control baselines, together with the common evaluation protocol.

D.1 Common Execution Protocol

All routing methods operate under the same vehicle dynamics, service rules, and resource constraints. Task assignment is fixed: every five simulated seconds, minimum-cost bipartite matching assigns available OHTs to pending tasks. The assignment cost is the sum of the shortest physical distances from the vehicle to the pickup location and from pickup to delivery. Routing methods determine the paths used to execute these assignments.

Routes are requested for pickup, delivery, and cruising movements. EDR, CBR, QLBWR, NQ, DQN, GDQN, and MAPPO additionally replan at branching nodes, while QECV replans at every node with an outgoing edge. KD, CD, PFD, and BRQ retain their selected routes unless a new route is required.

The simulator keeps one weighted copy of the track graph. D and KD select routes on this graph, while NQ, DQN, GDQN, and MAPPO use it to complete a route beyond their own next hop, and every method falls back to it when its returned route does not end at the requested target.

The routing comparison fixes regional control to admission only (C_A); the regional-control comparison fixes routing to D and varies C_0, C_A, and C_{AR} as described in Section 4.2. Alternating-direction merge grants and a minimum inter-vehicle safety distance of $0.8\,\mathrm{m}$ are shared across these comparisons.

D.2 Routing Baselines

D.2.1 k-Shortest-Path Routing (KD)

KD spreads vehicles over several near-shortest paths instead of concentrating them on one. For an origin–destination pair it enumerates the k shortest loopless paths under the shared weights and keeps those within a stretch bound,

$$\mathcal{P}(u,v) = \big\{p \in \mathrm{KSP}_k(u,v) :\ L(p) \le \eta\, L(p_1)\big\}, \tag{5}$$

where $L(p)$ is the static length of path p and p_1 is the shortest candidate. The path set is computed once per origin–destination pair and then reused for the rest of the run.

A vehicle is assigned a path by a hash of the run seed, the vehicle identity, and the pair,

$$\mathrm{index} = \mathrm{blake2b}\big(\texttt{seed}\,|\,\texttt{vehicle}\,|\,u\,|\,v\big) \bmod |\mathcal{P}(u,v)|, \tag{6}$$

so the assignment is fixed within a run, reproducible across replays, and different across seeds. Because it depends on the vehicle rather than on the request time, two vehicles travelling between the same stations keep different paths, while one vehicle keeps its path whenever it repeats a trip.

KD uses $k = 3$ and $\eta = 1.25$. It tests whether static path diversity alone relieves congestion, without any traffic measurement.

D.2.2 CONGESTION-WEIGHTED DIJKSTRA (CD)

CD is a state-based heuristic that reads live traffic at the moment a route is requested. Unlike the methods that smooth measurements over time, it recomputes every edge cost from the current simulator state, which makes it the most reactive and the most expensive of the classical baselines.

Edge cost. For track $e = (u, v)$ of length ℓ_e, let $\mathcal{V}_e$ be the vehicles currently on it and $\mathrm{cap}_e = \max(1, \lfloor \ell_e/\delta \rfloor)$ its nominal capacity at spacing δ. With occupancy ratio $\rho_e = |\mathcal{V}_e|/\mathrm{cap}_e$, queue length n_v at the downstream node, and h_v the saturation score of the area entered at v, the cost is

$$c_e = \ell_e \Big[1 + w_\rho \, \rho_e^{\beta} + w_q \, \frac{n_v}{s_q} + w_z \mathbb{1}\big[|\mathcal{V}_e| \geq 1\big] + w_h \max(0, h_v) + P \mathbb{1}\big[e \text{ not normal}\big] \Big]. \quad (7)$$

The occupancy term grows superlinearly, so a track close to capacity is penalized far more than a lightly used one; the presence term adds a fixed cost to any occupied track; the queue term anticipates waiting at the merge ahead; and the saturation term raises the cost of tracks that lead into a congested area.

Settings and cost. CD uses $w_\rho = 2$, $\beta = 2$, $w_q = 1$, $s_q = 5$, $w_z = 1$, $w_h = 1$, $\delta = 2\,\mathrm{m}$, and $P = 10^6$. A route is computed only when one is requested; CD does not replan at branching nodes. Every edge relaxation evaluates equation 7 against the live vehicle and queue state.

D.2.3 PREDICTED-FLOW DIJKSTRA (PFD)

PFD is a demand-driven fixed-route baseline. Given the rail network and task arrival rates in the From–To table, it predetermines a fixed path for each reachable origin–destination (OD) pair to distribute expected traffic while controlling travel distance. Its central mechanism is sequential assignment: high-demand OD groups choose routes first, and their assigned flow increases the cost seen by subsequent groups.

Demand representation. Let $G = (V, E)$ be the directed rail network, with physical edge lengths ℓ_e. To group OD pairs with similar routing choices, one-in/one-out chains are compressed into macro segments between junctions or endpoints, called anchors. Interior nodes of each segment are divided into at most p contiguous ranges of approximately equal size; each anchor forms a singleton range. An OD class $k = (c(u), c(v))$ groups origins and destinations by their ranges. Its route consists of a compulsory origin suffix to its exit anchor, a chosen macro path, and a compulsory prefix from the destination's entry anchor.

For bay-pair demand λ_{ab}, let V_a and V_b be the control nodes in the two bays. The predictor assigns each node pair the rate

$$d_{uv} = \frac{\lambda_{ab}}{|V_a|\,|V_b|}, \qquad u \in V_a, \quad v \in V_b. \quad (8)$$

Let $\mathcal{R}$ contain reachable pairs with distinct endpoints and let $\Lambda = \sum_{(u,v)\in\mathcal{R}} d_{uv}$. Excluded demand is not redistributed. Pairs that can travel directly along their common macro segment contribute only to fixed flow. For the other pairs, compulsory origin and destination portions also contribute to fixed edge flow F_e, while their rates are summed into class demand D_k. Thus F_e accounts for traffic that cannot be diverted by choosing a different macro path.

Flow-aware route assignment. With mean physical edge length $\bar{\ell} = |E|^{-1} \sum_{e\in E} \ell_e$, the cost of macro segment m, whose physical edges are E_m, is

$$C_m(f) = w_\ell \sum_{e\in E_m} \frac{\ell_e}{\bar{\ell}} + w_f \frac{\sum_{e\in E_m} f_e}{\Lambda}. \quad (9)$$

The first term penalizes distance, and the second penalizes predicted traffic along the segment. Both are dimensionless. Planned flow is initialized to $f_e = F_e$. Classes are processed in decreasing

demand D_k, with a fixed order for ties. Each class selects a minimum-cost macro path under the current flow, then adds its entire demand to that path. This single-pass assignment encourages later classes to use less-loaded alternatives while limiting detours through the distance term.

Algorithm D.1. Predicted-flow routing.

1. Construct the macro network and OD ranges; distribute bay-pair demand using equation 8.
2. Aggregate compulsory traversals into F_e and the remaining OD demand into D_k; initialize $f_e \leftarrow F_e$.
3. For each class k in decreasing D_k:
   (a) Compute segment costs $C_m(f)$ using equation 9.
   (b) Select a minimum-cost macro path P_k between the class's exit and entry anchors.
   (c) For every physical edge traversed by P_k, increase f_e by D_k.
4. Fix all selected paths. For OD classes without assigned demand, select paths under the final predicted-flow costs.
5. For each vehicle request, combine its class path with the compulsory origin and destination portions; use the direct segment path when applicable.

After preprocessing, the OD-to-path mapping remains fixed throughout the run, and each routing request retrieves its predetermined path. PFD thus provides a reference for assessing how far demand-informed planning can support routing, and what additional benefits learning or online adaptation can offer. The baseline uses $p = 4$, $w_\ell = 1$, and $w_f = 20$.

D.2.4 Exponentially Averaged Delay Routing (EDR)

EDR follows the adaptive routing of Bartlett et al. (2014), which replaces planned travel times with times measured on the network. Each track carries an estimate in seconds,

$$\hat{t}_e \leftarrow (1 - \lambda)\, \hat{t}_e + \lambda\, \tau_e, \tag{10}$$

updated whenever a vehicle completes track e in observed time τ_e, and initialized with the free-flow time $\ell_e / v_e^{\max}$. Routing does not read $\hat{t}_e$ directly. A routing table T_e is published from the current estimates every Δ seconds and used as the edge cost until the next publication, with unavailable tracks excluded. The delay between measuring and using an estimate keeps a single slow vehicle from redirecting the fleet.

EDR uses $\lambda = 0.1$ and $\Delta = 1\,\mathrm{s}$, and it recomputes a route at every branching node, so a vehicle can react to an estimate published after it started its trip. Relative to Bartlett et al. (2014), we compute a shortest path on demand instead of maintaining a full origin–destination table; for a given table the chosen next hops are the same, and the routing time is charged to the benchmark. EDR is the online-adaptive reference of the matrix: it reacts within one second of a measurement, without any model of demand or of the layout.

D.2.5 Congestion-Based Routing (CBR)

CBR follows Lee et al. (2018), which routes on section-level congestion measured over a long window rather than on instantaneous state. Sections are the maximal runs of track between branch, merge, or station nodes, so one index describes a whole corridor.

Congestion index. Every simulation frame, CBR records the speed of each moving vehicle. At the end of a window of W seconds, section σ receives the absolute congestion index

$$A_\sigma = \frac{1}{N_\sigma V_\sigma} \sum_{\text{frames}} \sum_{i \in \sigma} \min(v_i, V_\sigma) \in (0, 1], \tag{11}$$

where V_σ is the section speed limit and N_σ the number of vehicle samples. A section with no samples in the window is treated as free-flowing, $A_\sigma = 1$. The cost of a track in section σ combines distance and congestion,

$$c_e = \ell_e \Big(\alpha_d + \frac{\alpha_c}{\max(\underline{a}, A_\sigma)} \Big), \qquad \alpha_d + \alpha_c = 1, \tag{12}$$

so a free-flowing network gives pure distance routing and a fully congested section is penalized by at most $1/\underline{a}$.

CBR uses $W = 600\,\mathrm{s}$, $\alpha_d = 0.825$, $\alpha_c = 0.175$, and $\underline{a} = 0.01$, and it replans at branching nodes. CBR therefore represents the slow end of the adaptation spectrum, against which EDR at $1\,\mathrm{s}$ and CD at request time can be compared.

D.2.6 Learning-Based Methods: Common Setup

The seven learning-based methods share the route interface of Section D.1 and differ in what they learn and when they decide. Consider a vehicle at node i whose current leg ends at target t, which is a pickup, delivery, or cruising node. Let $N^+(i)$ be the successors of i, ℓ_{ij} the length of track (i, j), $d(j, t)$ the shortest distance from j to t, and $T^0(j, t)$ the corresponding free-flow travel time. When a vehicle leaves track (i, j), the simulator reports its traversal time τ_{ij}, which includes any waiting caused by following, merge grants, or admission. Every method learns from this observed time; none receives a separate congestion reward.

Decision points and route completion. QLBWR and BRQ store a value for every node–target–successor triple and assemble the whole route hop by hop. The other five methods choose only the next hop $c \in N^+(i)$ and append a shortest path from c to t. For NQ, DQN, GDQN, and MAPPO, this tail uses the simulator's shared routing weights of Section D.1; QECV uses free-flow travel times. QLBWR, NQ, DQN, GDQN, and MAPPO decide again at every branching node, QECV at every node with a successor, and BRQ only when a new route is requested.

Stretch-bounded action sets. Unrestricted exploration of a learned policy can send vehicles on long detours that no dispatcher cost anticipates. QECV, NQ, DQN, GDQN, and MAPPO therefore choose among

$$\mathcal{A}_s(i, t) = \Big\{ c \in N^+(i) : \ \ell_{ic} + d(c, t) \le s \min_{c' \in N^+(i)} \big(\ell_{ic'} + d(c', t)\big) \Big\}, \tag{13}$$

where $s \ge 1$ is the stretch bound. QECV applies the same rule to free-flow times. The shortest successor always remains admissible, so the constraint never leaves a vehicle without an action.

Training and evaluation. Each learned method uses one frozen checkpoint for each scenario, and the same checkpoint is evaluated under C_0, C_A, and C_{AR}. In the main comparison, each checkpoint is trained on its corresponding scenario and then held fixed throughout evaluation. Parameters are not updated and exploration is disabled, except that QLBWR retains its Boltzmann action rule. Neural methods are initialized by imitating the shortest-distance next hop before temporal-difference or policy-gradient training, so that online training starts from a feasible routing policy. Cross-scenario transfer is evaluated separately in Appendix E.3, where a checkpoint trained on a source scenario is applied to a target scenario without further adaptation.

Figure 14 shows how the TD loss of the three neural-based routers evolves during online training on L275 scenario. All three losses gradually drop within the first $5\,\mathrm{ks}$ and remain close to 2×10^{-2} for the rest of the $20\,\mathrm{ks}$ run.

D.2.7 Q(λ)-Based Routing (QLBWR)

QLBWR follows the Q(λ) route guidance of Hwang & Jang (2020), proposed for OHT systems. It learns $Q(i, t, j)$, the expected travel time from i to t when leaving through j, and initializes it with $T^0_{ij} + T^0(j, t)$. After traversing (i, j), the update uses potential-based shaping with $\Phi(x) = T^0(x, t)$:

$$\delta = \tau_{ij} + \rho\big(\gamma\Phi(j) - \Phi(i)\big) + \gamma \min_{k \in N^+(j)} Q(j, t, k) - Q(i, t, j), \tag{14}$$

with the bootstrap term omitted when $j = t$. Accumulating eligibility traces decay by $\gamma\lambda$ and every traced pair moves by $\alpha\delta e$; traces are cleared when the vehicle leaves the greedy path or reaches t, as in Watkins' Q(λ) (Watkins & Dayan, 1992). Actions follow a normalized Boltzmann rule, $p(j) \propto \exp\big(-Q(i, t, j)/(\tau\,\bar{Q}_{i,t})\big)$, where $\bar{Q}_{i,t}$ is the mean value over $N^+(i)$. We use $\alpha = 0.1$, $\gamma = 0.9$, $\lambda = 0.8$, $\rho = 1$, and $\tau = 0.03$. Two adaptations differ from the original: learning runs on the physical track graph rather than an aggregated decision graph, and only the current leg's target is updated, because updating all targets after every traversal is too slow for fleets of 100–300 vehicles.

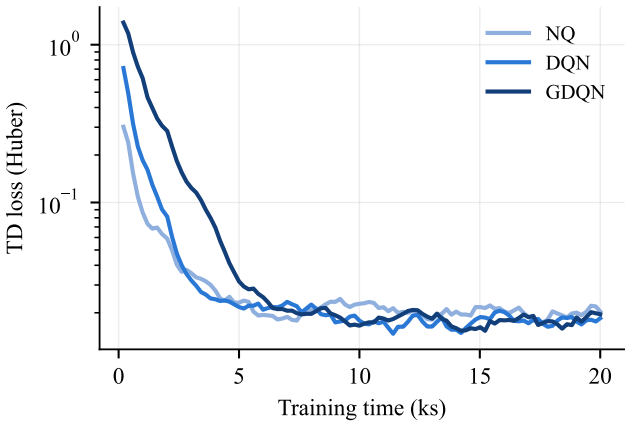


Figure 14: TD loss of NQ, DQN, and GDQN during online training on L275 scene. Curves show a moving average over 2 ks, computed in log space.

D.2.8 Q-Learning with Expected Congested Vehicles (QECV)

QECV follows Kang et al. (2022), which augments the routing state with the number of vehicles competing for the same destination. Its state is (i, t, M), where M counts other vehicles with target t whose remaining distance to t is shorter than the ego vehicle's. Transitions of one leg are buffered and applied when the leg ends, with step size $\alpha_n = 1/(1 + n(s, a))$ for the n-th visit and target $\tau_{ij} + \gamma \min_{a'} Q(s', a')$. During training, actions are drawn with probability proportional to $\rho^{1/Q(s,a)}$. We use $\gamma = 0.9$ and $\ln \rho = 320$. Two adaptations apply at evaluation: actions are greedy rather than sampled, and they are restricted to $\mathcal{A}_{1.5}$ under free-flow times (equation 13).

D.2.9 Bellman-Residual Q-Routing (BRQ)

BRQ is our extension of Q-routing (Boyan & Littman, 1993). In Q-routing, a delay observed on one track changes only the value of that track, so upstream decisions leading to it learn about the delay only after vehicles traverse them again. BRQ passes the same temporal-difference error to those upstream decisions immediately.

Value and update. BRQ keeps $Q(i, t, j)$ and initializes it with the shortest distance $\ell(i, j) + d(j, t)$. After traversing (i, j), it computes

$$\delta = \tau_{ij} + \gamma \min_{k \in N^+(j)} Q(j, t, k) - Q(i, t, j), \qquad Q(i, t, j) \leftarrow Q(i, t, j) + \alpha\delta. \tag{15}$$

A breadth-first search over reversed tracks then collects every node l within H hops upstream of i, together with its next node m toward i. With h the hop count from l to i, each such decision is updated as

$$Q(l, t, m) \leftarrow Q(l, t, m) + \alpha\delta\, \kappa^{h+1}, \qquad 1 \le h \le H. \tag{16}$$

The decay $\kappa < 1$ weakens the correction with distance from the observed delay.

Algorithm D.2. Bellman-residual Q-routing.

1. Initialize $Q(i, t, j) \leftarrow \ell(i, j) + d(j, t)$ for all reachable triples.
2. When a route to t is requested at i, build it hop by hop: at each node, move to $\arg\min_j Q(\cdot, t, j)$ among unvisited successors; if t is not reached, use the shortest-distance route.
3. During training, when a vehicle leaves (i, j):
   (a) Compute δ and update $Q(i, t, j)$ using equation 15.
   (b) Update the upstream decisions within H hops using equation 16.
4. During evaluation, keep Q fixed and return the greedy route.

The evaluated configuration uses $\alpha = 0.1$, $\gamma = 0.1$, $H = 4$, and $\kappa = 0.65$; training actions follow a mean-centred Boltzmann rule with temperature 0.3. The name refers to the propagated Bellman

residual δ; BRQ is not the residual-gradient method of Baird (1995). BRQ isolates the effect of faster upstream credit assignment while keeping the tabular state of Q-routing.

D.2.10 NEURAL Q-ROUTING (NQ)

NQ is our neural value-based router. It replaces the per-triple table with a network shared by all vehicles and nodes, so a decision can use local traffic around each candidate successor.

Features and network. For each candidate $c \in N^+(i)$, NQ builds a 24-dimensional vector $\phi(i, t, c)$ with four groups: (i) normalized indices of i, t, and c, together with in- and out-degrees and a station indicator; (ii) the normalized distances $d(i, t)$, ℓ_{ic}, and $d(c, t)$, and the distance progress $d(i, t) - d(c, t)$; (iii) the vehicle's task phase and accumulated congestion time; and (iv) traffic around the candidate, namely occupancy and waiting share of track (i, c), the queue at c, the maximum and mean occupancy on tracks one and two hops downstream, and two scores derived from them for bottlenecks and spillback. Distances are divided by $\max(1, \bar{\ell}\sqrt{|V|})$ and traffic signals are clipped to $[0, 1]$. A multilayer perceptron with two hidden layers of 64 ReLU units maps ϕ to a scalar $Q_\theta(i, t, c)$, trained with Adam (Kingma & Ba, 2014) and the Huber loss. Because group (i) identifies nodes, NQ is specific to one track topology.

Semi-Markov learning. Decisions occur only at branching nodes, so the time between decisions varies. NQ accumulates the traversal times between two consecutive decisions into a reward $r = -\sum \tau/30\,\mathrm{s}$ and regresses Q_θ toward

$$y = r + \gamma \max_{c' \in N^+(i')} Q_\theta(i', t, c'), \tag{17}$$

where i' is the next decision node and $y = r$ when t is reached. Transitions enter a replay buffer of 5,000 samples; after 256 transitions, one minibatch of 64 is drawn every four transitions. NQ uses no target network.

Algorithm D.3. Neural Q-routing.

1. Pretrain Q_θ on sampled branching nodes and targets by regressing a one-hot score on the shortest-distance successor.
2. At each branching node i, compute $\mathcal{A}_s(i, t)$ using equation 13 and select $c = \arg\max_{c \in \mathcal{A}_s(i,t)} Q_\theta(i, t, c)$; during training, explore with probability ε.
3. Return the route $[i, c]$ followed by the shortest path from c to t.
4. During training, store the semi-Markov transition at the next decision node and update Q_θ toward equation 17.

Training uses $\gamma = 0.99$ per decision, learning rate 10^{-5}, and $\varepsilon = 0.01$; pretraining uses 5,000 samples for 100 epochs. Evaluation uses $s = 1.05$ in SMALL and MID scenarios and $s = 1.02$ in LARGE scenarios. NQ tests whether local traffic features, rather than a larger routing table, improve next-hop selection.

D.2.11 DEEP Q-NETWORK ROUTING (DQN)

DQN keeps the features, network, and decision points of NQ and adds the stabilizers of deep Q-learning (Mnih et al., 2015). A target network Q_{θ^-} replaces Q_θ in the bootstrap term of equation 17 and tracks the online network by Polyak averaging, $\theta^- \leftarrow (1-\eta)\theta^- + \eta\theta$ with $\eta = 0.005$ after every update. The target uses a single network for both action selection and evaluation, not the Double DQN estimator. Exploration decays linearly from $\varepsilon = 0.05$ to 0.01 over 50,000 decisions. All other settings, including pretraining and the stretch bound, match NQ, so the NQ–DQN comparison isolates the target network and exploration schedule.

D.2.12 GRAPH-AGGREGATED DEEP Q-NETWORK ROUTING (GDQN)

GDQN is our topology-agnostic variant of DQN. It removes node identities from the input and describes each candidate by traffic aggregated over its downstream neighborhood, following the mean aggregator of GraphSAGE (Hamilton et al., 2017).

For a node n, GDQN defines a seven-dimensional signal $g(n)$: normalized distance to t, in- and out-degree, mean and maximum occupancy of the tracks leaving n, queue length at n, and the share of stopped vehicles on those tracks. The candidate representation is

$$\phi_G(i,t,c) = \Big[\, \phi_{\mathrm{loc}}(i,t,c),\ g(c),\ \tfrac{1}{|\mathcal{N}_1|}\textstyle\sum_{n\in\mathcal{N}_1} g_{1:4}(n),\ \tfrac{1}{|\mathcal{N}_2|}\sum_{n\in\mathcal{N}_2} g_{1:3}(n) \Big], \tag{18}$$

where ϕ_{loc} keeps the ten NQ features that contain no node index, $\mathcal{N}_1$ contains the successors of c, and $\mathcal{N}_2$ their successors other than c. The aggregation is fixed and parameter-free; the Q-network and training procedure are those of DQN. Evaluation uses $s = 1.10$ in all scenarios. The design permits cross-topology transfer. In the main routing comparison, each checkpoint is evaluated on its training topology; Appendix E.3 separately evaluates frozen source-scenario checkpoints on different layouts.

D.2.13 MAPPO WITH BEHAVIOR-CLONING INITIALIZATION

MAPPO is our multi-agent policy-gradient router, initialized by behavior cloning. It follows centralized training with decentralized execution as in MAPPO (Yu et al., 2022): all vehicles share one actor that sees local information, while a critic sees a system-level state during training.

Actor and critic. The actor scores each admissible candidate $c \in \mathcal{A}_s(i,t)$ with a network on the GDQN features ϕ_G (two hidden layers of 64 tanh units) and selects among candidates with a softmax. The critic receives a 16-dimensional global state: normalized simulation time; the fractions of idle, pickup-bound, loading, delivery-bound, and unloading vehicles; mean, maximum, and standard deviation of track occupancy; the share of stopped vehicles; pending and in-progress tasks per vehicle; and mean and maximum bay fill and node queue length.

Asynchronous policy optimization. Vehicles reach branching nodes at different times, so each transition spans a variable interval Δt. The reward is $-\Delta t$ in seconds, and discounting follows elapsed time rather than decision count:

$$\hat{A}_k = \delta_k + \gamma^{\Delta t_k}\lambda_{\mathrm{GAE}}\hat{A}_{k+1}, \qquad \delta_k = -\Delta t_k + \gamma^{\Delta t_k} V_\psi(x_{k+1}) - V_\psi(x_k), \tag{19}$$

where x_k is the global state and advantages are computed within each vehicle's sequence of decisions. The actor maximizes the clipped PPO objective (Schulman et al., 2017), and the critic minimizes a squared value error.

Algorithm D.4. MAPPO with behavior-cloning initialization.

1. Behavior cloning: sample branching nodes and targets, and train the actor with cross-entropy to select the shortest-distance successor.
2. At each branching node, sample $c \sim \pi(\cdot \mid i,t)$ over $\mathcal{A}_s(i,t)$ and return $[i,c]$ followed by the shortest path from c to t.
3. Close a transition when the vehicle reaches its next branching node or t, and append it to a shared rollout.
4. After every 64 transitions, compute advantages using equation 19 and run four epochs of PPO updates.
5. During evaluation, select $\arg\max_c \pi(c \mid i,t)$ and do not update.

Behavior cloning uses 5,000 samples for 10 epochs with learning rate 10^{-3}. PPO uses $\gamma = 0.99$ per second, $\lambda_{\mathrm{GAE}} = 0.95$, clipping range 0.2, value coefficient 0.5, entropy coefficient 0.01, learning rate 10^{-5}, and gradient-norm clipping at 0.5. Evaluation uses $s = 1.05$. Relative to MAPPO, the adaptations are a variable candidate set, time-based discounting for asynchronous decisions, and behavior-cloning initialization.

D.3 REGIONAL CONTROL BASELINES

Regional control decides which vehicles may enter a congested part of the network and, in one setting, where idle vehicles wait. Three settings are compared: C_0 applies neither policy, C_A adds water-level region admission, and C_{AR} adds empty-vehicle rebalancing on top of it. The admission machinery of C_A and C_{AR} is identical, so their difference isolates repositioning.

D.3.1 WATER-LEVEL REGION ADMISSION (C_A)

A controlled region is a set of nodes declared in the scenario workbook, together with a close water level C and a quota level $R < C$. Regions may overlap or be nested within one another, and a vehicle may lie inside multiple regions at once, and each region counts it independently. Entry points are derived from the layout as the tracks that cross into the region, and every entry point keeps its own queue. We compute a reference water level from the tracks within the region,

$$C_{\mathrm{ref}} = |M| + \left\lfloor \frac{\Lambda - \Lambda_M}{\ell_{\mathrm{veh}} + s} \right\rfloor, \tag{20}$$

where Λ is the total track length inside the region, M its merge nodes, Λ_M the length of their control zones, ℓ_{veh} the vehicle length, and s the safety spacing. This formula provides a reference for setting regional water levels. Starting from this reference, we make adjustments based on operational experience to determine the final close water level C and quota level $R < C$. Table 28 summarizes the regions of each map.

Table 28: Controlled regions of the three maps. Water level is the final configured high-water level C; reference water level is the value C_{ref} computed from track length by equation 20. Rebalancing regions are those carrying a target reserve, which only C_{AR} reads.

Map	Regions	Nesting layers	Water level (med.)	Reference water level (med.)	Rebalancing regions
SMALL	56	1–6	28	30	18
MID	186	1–11	19	21	30
LARGE	224	0–11	17	21	37

Occupancy. A region counts a vehicle when it occupies one of its nodes, and also when it holds an entry permission and has come within the $2\,\mathrm{m}$ control line of an entry point. Counting approaching vehicles prevents a region from admitting more traffic than it can absorb while earlier entrants are still crossing the boundary. A permission held by a vehicle that is still far away does not count, so it does not reserve space it may not use for some time.

Admission bands. Let O be the occupancy. The region operates in three bands.

- **Open** ($O < R$). The head of every entry queue receives a permission in each round.
- **Quota** ($R \le O < C$). At most $C - O - F$ further permissions are issued, where F is the number outstanding. Permissions rotate: a holder that has not entered within $15\,\mathrm{s}$ yields to the head of another entry queue if one is waiting, and otherwise keeps its turn.
- **Hold** ($O \ge C$). Permissions that are not already committed are withdrawn and none are issued. A vehicle that has crossed the control line or entered keeps its permission.

Rounds start from a rotating cursor over the entry points and grant at most one vehicle per entry point, which keeps a single busy entrance from consuming the whole quota. Within an entry queue, vehicles are ordered by their remaining distance to that entry.

Execution. A vehicle without a permission decelerates to the control line and waits there, so a blocked region forms a queue outside its boundary rather than inside it. Admission also constrains right of way: a vehicle queued for a region cannot take a merge grant at a merge node inside that region, which stops it from entering through the intersection it is waiting at. Permissions are re-evaluated whenever occupancy changes, and a periodic sweep every $3\,\mathrm{s}$ revisits regions with non-empty queues.

D.3.2 EMPTY-VEHICLE REBALANCING (C_{AR})

Admission control keeps vehicles out of saturated regions but does not decide where idle vehicles wait. Rebalancing adds that decision. Each participating region declares a target reserve F in the scenario workbook; the remaining regions do not take part.

Every $5\,\mathrm{s}$, the controller evaluates each participating region:

$$\Delta = F + Q - I - A, \tag{21}$$

where Q counts tasks whose pickup lies in the region and has not yet been served, I the idle vehicles inside it, and A the loaded vehicles already bound for it. A positive Δ is a deficit and a negative one a surplus. Commitments from earlier rounds are subtracted before matching, so a vehicle already on its way is not requested twice.

Each reserve follows the origin demand of the region's bay, $\lambda_i = \sum_j \mathrm{TaskArrive}(i, j)$, normalized across the participating bays and scaled to a scenario-level cap U:

$$F_i = \min\Big\{U,\ \Big\lfloor U\,\frac{\lambda_i - \lambda_{\min}}{\lambda_{\max} - \lambda_{\min}} + \tfrac{1}{2}\Big\rfloor\Big\}. \tag{22}$$

The bay with the highest origin demand therefore holds U idle vehicles and the one with the lowest holds none. Each F_i is a local target for online repositioning rather than a static allocation of vehicles.

Transfers are then assigned in two stages, both by minimum-cost matching. The first matches surplus regions to deficit regions using the static distance between their boundaries; the second matches individual idle vehicles in each surplus region to the assigned destinations using their current positions. Only vehicles that are idle, taskless, and not already repositioning are eligible, and a matched vehicle is sent to the nearest entry node of its destination region, where it waits for its next task. Distances in both stages come from the static layout rather than from the routing method under test, so the repositioning decision is the same for all 13 routing methods.

D.4 Evaluation Metrics

D.4.1 Metric Definitions

Every metric is first computed for one run over the full window $[0, T]$ with $T = 20{,}000\,\mathrm{s}$; no warm-up period is removed. For a task k, let t_k^{rel} be its release time, t_k^{asg} its assignment to a vehicle, t_k^{load} and t_k^{end} the completion of loading and unloading, and $\mathcal{K}$ the set of tasks completed by T.

Transport efficiency. Throughput is $3600\,|\mathcal{K}|/T$ tasks per hour. Three task times are recorded:

$$\underbrace{t_k^{\mathrm{end}} - t_k^{\mathrm{rel}}}_{\text{cycle}}, \qquad \underbrace{t_k^{\mathrm{load}} - t_k^{\mathrm{asg}}}_{\text{pickup}}, \qquad \underbrace{t_k^{\mathrm{end}} - t_k^{\mathrm{load}}}_{\text{delivery}}. \tag{23}$$

Pickup time covers the empty trip and loading, and delivery time covers the loaded trip and unloading. Cycle time additionally contains the dispatch wait $t_k^{\mathrm{asg}} - t_k^{\mathrm{rel}}$, so it exceeds the sum of the other two. For each run we report the mean, P50, and P95 over $\mathcal{K}$, with percentiles obtained by linear interpolation. Tasks still open at T do not enter these statistics; task times must therefore be read together with throughput, especially when a run approaches collapse.

OHT utilization is the share of fleet time spent serving tasks,

$$U = \frac{1}{N\,T}\sum_{v=1}^{N}\int_0^T \mathbb{1}\big[v \text{ is pickup-bound, loading, delivery-bound, or unloading at } s\big]\,ds, \tag{24}$$

where N is the fleet size. Vehicle states are sampled every simulated second. Idle waiting, cruising, and rebalancing moves are not counted as service. Higher utilization therefore means that more of the fleet is occupied by tasks; at a fixed throughput, it indicates longer service per task rather than better performance.

Travel time index. TTI measures delay along the path a vehicle actually drove. For task k, let P_k be the sequence of tracks traversed during its pickup and delivery trips, including repeated tracks and tracks added by replanning. With $v_e^{\max}$ the smaller of the vehicle and track speed limits,

$$\mathrm{TTI}_k = \frac{\big(t_k^{\mathrm{arr,p}} - t_k^{\mathrm{asg}}\big) + \big(t_k^{\mathrm{arr,d}} - t_k^{\mathrm{load}}\big)}{\sum_{e\in P_k} \ell_e / v_e^{\max}}, \tag{25}$$

where $t_k^{\mathrm{arr,p}}$ and $t_k^{\mathrm{arr,d}}$ are the arrival times at the pickup and delivery nodes. The numerator includes waiting caused by following, merge grants, and admission, but excludes loading, unloading, and the dispatch wait. The denominator assumes constant maximum speed, so $\mathrm{TTI}_k > 1$ even on an empty network because of acceleration and deceleration. Because both terms refer to the same driven path, TTI does not penalize a detour that is traversed without delay; such detours appear in pickup and delivery times instead. The run-level TTI is the mean of TTI_k over $\mathcal{K}$.

Routing latency. Route ms is the mean wall-clock time of a top-level routing call, measured with a monotonic high-resolution clock. It includes work done inside the call, such as refreshing a traffic snapshot or evaluating a network, and excludes learning updates, periodic traffic observation outside the call, and the fixed dispatcher. The number of calls differs by method: D, KD, CD, PFD, and BRQ are called once per route request, methods that replan at branching nodes about five times as often, and QECV at every node. Route ms therefore measures the latency of one decision, not the total routing cost of a run. All methods were timed under the same 32-worker configuration, so the values are comparable across methods.

D.4.2 Failure Classification

Three run-level indicators describe loss of liveness. They answer different questions and are reported separately.

Detector events (DE). The online detector of Appendix A.6 confirms a circular wait when the same set of stopped vehicles remains mutually blocked for at least $100\,\mathrm{s}$. The simulation continues after confirmation, and no intervention is applied. A run counts toward DE if at least one event is confirmed.

Terminal gridlock. A run ends in terminal gridlock if tasks remain open at T and no task has been completed during the final $600\,\mathrm{s}$.

Throughput collapse (DL). A run is a throughput-collapse run if it ends in terminal gridlock or its throughput falls below half of a stable reference. For scenario σ, method m, and seed z, the reference is

$$\Theta^{\mathrm{ref}}_{\sigma,m,z} = \max\Big(\Theta^{C_A}_{\sigma,m,z},\ \Theta^{C_A}_{\sigma,\mathrm{D},z},\ \operatorname*{med}_{z'}\Theta^{C_A}_{\sigma,m,z'},\ \operatorname*{med}_{z'}\Theta^{C_A}_{\sigma,\mathrm{D},z'}\Big), \tag{26}$$

where Θ^{C_A} is throughput under admission control. The reference is taken from C_A, the setting with the fewest failures, and is used for all three control settings. Taking the maximum over four candidates prevents a method's own degraded run from lowering its threshold. In the final matrix, throughput-to-reference ratios have a clear gap around the threshold: no run has a ratio between 0.498 and 0.501.

Why DE and DL differ. Table 29 cross-tabulates the two indicators. DE records a structural event, whereas DL records a system-level loss of service, and neither implies the other. Of the 103 runs with DE but without DL, 102 still contain an unresolved event at T: a circular wait can persist locally while the rest of the fleet continues to serve at least half of the reference throughput. Conversely, 72 runs collapse without a confirmed event, because the detector confirms only a circular wait that keeps the same stopped vehicles for $100\,\mathrm{s}$, whereas a loss of service can also follow from shifting blocking patterns or from severe congestion.

Table 29: Overlap of detector events (DE), throughput collapse (DL), and terminal gridlock over the 910 runs of each control setting.

Control	DL	DE	DL and DE	DL only	DE only	Terminal gridlock
C_0	265	295	200	65	95	31
C_A	9	6	4	5	2	1
C_{AR}	3	7	1	2	6	1

D.4.3 Aggregation across Seeds

Each table cell summarizes the ten runs of one scenario, control setting, and routing method. DL and DE are counts out of ten. All other values are unweighted means of the run-level statistics over the runs not classified as DL; runs with DE but without DL are included. Percentiles are therefore averages of per-run percentiles, not percentiles of pooled tasks. A dash marks a cell in which all ten runs collapse.

D.5 Software and Hardware

The final matrix used Python 3.12.13 on Linux 6.8 (x86-64). Runs were CPU-only on a server with two AMD EPYC 9534 64-core processors (256 hardware threads) and 251 GiB RAM. Classical and learned matrices each used 32 parallel workers. Every run is isolated and owns its random-number streams.

E Complete Experimental Results

This appendix reports the complete $13 \times 7 \times 3$ routing–scenario–control matrix. Every cell contains ten $20{,}000$-s runs with evaluation seeds 0–9. DL denotes throughput-collapse runs and DE denotes runs with at least one confirmed circular-wait event, both defined in Appendix D.4.2. The two indicators differ in both directions: a circular wait can persist locally without halving throughput, and throughput can collapse without a confirmed event (Table 29). All remaining metrics are means over non-collapse runs only. 'Route ms' means routing call latency in milliseconds.

Across all 910 runs per control setting, C_0 has 265 collapse runs, 295 runs with detector events, and 31 terminal-gridlock runs. The corresponding counts are 9, 6, and 1 for C_A, and 3, 7, and 1 for C_{AR}. Thus admission control nearly eliminates the prevalent failure mode, reducing collapse incidence by more than 96%. Under C_0, every routing method collapses in at least 12 of its 70 scenario–seed runs, showing that the high-load instability is not confined to one routing family.

Under the common C_A comparison, EDR has the lowest conditional mean cycle time in six of seven scenarios and is only 0.1 s behind QECV in MID 175; QECV, however, collapses in two MID 175 seeds. No learned method consistently dominates EDR across scenarios. Several learning-based configurations match the strongest heuristics, but QLBWR and MAPPO each retain isolated high-load collapses and their ranking changes with scenario. These observations support our conclusion that routing results must be interpreted jointly with the accompanying control setting.

E.1 Size, Extent and Selected Snapshots of Detected Circular Waits

Regional control changes more than the number of failures: it changes their scale. Each detector event is recorded with a snapshot that contains the confirmed waiting set, its wait-for edges, and the state of every controlled region. We use these snapshots to measure how many vehicles hold a blocking in place.

Counting the vehicles that sustain a blocking. Let $\mathcal{C}$ be the confirmed waiting set. When a vehicle in $\mathcal{C}$ waits on a closed region R, that region stays closed as long as its occupancy is not below the close threshold, so the vehicles inside R are part of what keeps the blocking alive even when they are not in $\mathcal{C}$ themselves. We therefore report

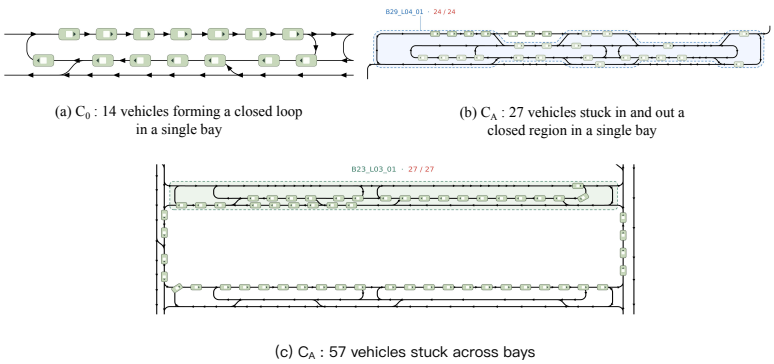
$$N = |\mathcal{C}| + \max_{R \in \mathcal{R}} \big(\mathrm{occ}(R) - |\mathcal{C} \cap R|\big)^+, \tag{27}$$

where $\mathcal{R}$ is the set of closed regions that appear as blocking resources in the wait-for edges, $\mathrm{occ}(R)$ is the number of vehicles inside R, and $(\cdot)^+$ denotes the positive part. Because controlled regions are nested, taking the maximum over $\mathcal{R}$ rather than a sum avoids counting a vehicle twice; N is thus a lower bound on the vehicles involved. With $\mathcal{R} = \emptyset$, N reduces to the size of the waiting cycle.

Without regional control, failures are local. Under C_0, 308 events occur in 295 runs, and every routing method contributes between 16 and 30 of them. Their wait-for edges consist almost entirely of vehicle following and merge grants, and no controlled region is active under C_0, so the waiting

Table 30: Detected circular waits by control setting over all 13 routing methods, seven scenarios, and seeds 0–9 (910 runs per setting). Sustaining vehicles follow equation 27; the two size columns coincide unless a closed region takes part in the blocking.

Control	Runs	Events	Waiting cycle median [min, max]	Sustaining vehicles median [min, max]	≤ 20 vehicles	One bay	Resolved before T
C_0	295	308	14 [10, 65]	14 [10, 65]	287	284	8
C_A	6	6	62.5 [7, 111]	62.5 [27, 115]	0	1	1
C_{AR}	7	11	66 [64, 114]	67 [64, 137]	0	0	4



(a) C_0 : 14 vehicles forming a closed loop in a single bay

(b) C_A : 27 vehicles stuck in and out a closed region in a single bay

(c) C_A : 57 vehicles stuck across bays

Figure 15: Snapshot of three typical deadlock cases. (a) Under regional control policy C_0, 14 vehicles formed a closed loop in a single bay. (b) Under regional control policy C_A, 27 vehicles are stuck in and out a closed region in a single bay. Locations of light-colored vehicles are inferred and might not be accurate, as these are absent from the deadlock report. (c) Under regional control policy C_A, 57 vehicles are stuck across bays.

cycle and the sustaining set coincide throughout. Both are small and uniform, with a median of 14 vehicles; 287 events (93%) involve at most 20 vehicles and 284 (92%) lie within a single bay. Medians are 13 or 14 vehicles in every scenario except SMALL 150, where 12 of the 14 sets contain 28 or 29 vehicles. The same local pattern appears across routing families, which agrees with the collapse counts in Table 29: routing alone does not remove the failure mode that dominates C_0.

With regional control, residual failures are larger and structural. Admission control removes the local waiting sets almost entirely. The 17 events that remain under C_A and C_{AR} occur in 13 of 1,820 runs, every one of them waits on at least one closed region, and none involves fewer than 27 vehicles. Sixteen of the 17 span four bays. The exception occurs in a rare case of bay B29 in the LARGE 300 scene deadlocked with 27 vehicles. In the remaining events, the waiting cycle already contains most of the occupants of the closed region, and the two counts differ by at most 23 vehicles. The large events also recur at the same locations: all nine LARGE 300 events include bay B02, and all seven SMALL 150 events involve the same four bays, two of which are interbay tracks.

Fig. 15 shows three typical cases of deadlocks. Fig. 15 (a) illustrates a deadlock occurring within a single bay. The 14 vehicles form a loop based on car-following and merge requests without the involvement of any zones, as regional control policy hasn't been applied. Fig. 15 (b) is a case where 7 vehicles are in the loop, but the loop passes through two nested closed zones (zone B29 L03 02 and B29 L04 01). The leading vehicle in the queueing entry is waiting for region B29 L04 01 to reopen. However, the number of vehicles inside this zone (24) has reached the water level, and the zone will only reopen once the OHT count drops to the safe level (21). The case shown in Fig 15 (c) is a large, network-level loop comprising four bays (two interbay and two intrabay segments). The vehicles that follow other vehicles, request merge, and queue for entering zones form a continuous loop.

The final results support the role of regional control in the main comparison. We observe 308 events in 910 runs under C_0, compared with 6 under C_A and 11 under C_{AR}, each also evaluated over 910 runs. Regional admission nearly eliminates the small waiting sets that account for most C_0 events. The residual events involve at least 27 vehicles and concentrate at a few locations, predominantly spanning multiple bays. Every residual event includes a wait on a closed region, pointing to interactions between region occupancy and entry queues as a target for further control design.

E.2 Full routing results

E.2.1 SMALL 100

Table 31: SMALL 100 under C_0. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	0/10	0/10	2393	123.6	115.2	217.8	45.6	73.8	2.00	79.7	0.34
KD	0/10	0/10	2392	140.4	127.5	258.8	53.7	77.4	1.98	87.5	54.77
CD	0/10	0/10	2393	120.4	113.2	209.3	43.7	73.2	1.98	78.1	2.31
PFD	0/10	0/10	2393	120.7	113.5	210.4	43.7	73.5	1.92	78.2	0.02
EDR	0/10	0/10	2393	117.6	111.2	201.1	42.2	72.2	1.95	76.4	0.43
CBR	0/10	0/10	2393	119.1	112.3	205.7	43.1	72.7	1.98	77.3	0.71
QLBWR	0/10	0/10	2393	124.4	115.7	220.0	46.1	74.0	2.00	80.2	0.46
QECV	0/10	0/10	2393	123.7	115.3	217.8	45.5	74.1	2.00	79.8	0.33
BRQ	0/10	0/10	2393	123.3	114.9	217.2	45.3	73.7	2.00	79.5	0.37
NQ	0/10	0/10	2393	121.7	113.9	213.1	44.3	73.6	2.00	78.8	0.66
DQN	0/10	0/10	2393	121.3	113.5	211.7	44.2	73.4	1.99	78.6	0.51
GDQN	0/10	0/10	2393	123.1	115.0	216.2	45.2	73.7	2.00	79.4	0.67
MAPPO	0/10	0/10	2393	123.3	115.1	216.9	45.3	73.8	2.00	79.5	3.71

Table 32: SMALL 100 under C_A. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	0/10	0/10	2393	120.5	112.5	211.7	43.6	73.2	1.99	78.0	0.36
KD	0/10	0/10	2392	135.7	124.1	247.3	51.2	76.7	1.97	85.4	55.39
CD	0/10	0/10	2394	118.7	111.7	205.7	42.7	72.7	1.97	77.1	2.39
PFD	0/10	0/10	2394	118.8	111.5	207.2	42.5	73.1	1.91	77.2	0.02
EDR	0/10	0/10	2394	115.7	109.6	197.4	41.0	71.7	1.94	75.3	0.44
CBR	0/10	0/10	2394	117.3	110.6	202.3	41.9	72.2	1.97	76.2	0.72
QLBWR	0/10	0/10	2393	121.4	113.5	213.1	44.3	73.3	1.99	78.5	0.49
QECV	0/10	0/10	2393	121.3	113.3	212.6	44.1	73.4	1.99	78.5	0.35
BRQ	0/10	0/10	2393	120.6	112.8	210.5	43.7	73.1	1.99	78.0	0.37
NQ	0/10	0/10	2393	119.6	112.2	208.5	43.0	72.9	1.98	77.4	0.69
DQN	0/10	0/10	2394	118.9	111.6	206.2	42.7	72.8	1.98	77.1	0.54
GDQN	0/10	0/10	2393	120.4	112.6	211.3	43.7	73.1	1.99	78.0	0.69
MAPPO	0/10	0/10	2393	120.7	112.7	211.4	43.9	73.1	1.99	78.1	4.01

Table 33: SMALL 100 under C_{AR}. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	0/10	0/10	2393	118.9	111.0	208.8	42.1	73.3	2.00	77.1	0.36
KD	0/10	0/10	2392	134.2	122.8	245.9	50.3	76.7	1.97	84.8	55.05
CD	0/10	0/10	2393	117.7	110.4	204.7	41.6	72.9	1.98	76.4	2.36
PFD	0/10	0/10	2394	117.2	109.7	205.1	40.9	73.2	1.92	76.2	0.02
EDR	0/10	0/10	2394	114.5	108.2	196.1	39.7	71.8	1.95	74.5	0.44
CBR	0/10	0/10	2394	115.7	108.9	200.5	40.4	72.4	1.98	75.3	0.71
QLBWR	0/10	0/10	2393	119.3	111.3	209.8	42.3	73.4	2.00	77.3	0.48
QECV	0/10	0/10	2393	119.9	111.6	211.1	42.7	73.6	2.00	77.7	0.35
BRQ	0/10	0/10	2393	119.0	110.9	209.3	42.1	73.3	2.00	77.1	0.37
NQ	0/10	0/10	2393	118.5	110.7	207.6	41.9	73.1	1.99	76.8	0.69
DQN	0/10	0/10	2393	117.9	110.3	206.0	41.6	73.0	1.99	76.5	0.55
GDQN	0/10	0/10	2393	118.6	110.7	207.9	41.9	73.3	2.00	76.9	0.68
MAPPO	0/10	0/10	2393	118.9	111.1	208.5	42.0	73.3	2.00	77.0	3.97

E.2.2 SMALL 150

Table 34: SMALL 150 under C_0. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	0/10	0/10	3081	166.1	138.3	331.5	53.9	92.4	2.50	84.1	0.37
KD	0/10	0/10	3092	153.0	134.0	296.5	50.5	91.5	2.39	81.8	55.00
CD	0/10	1/10	3057	121.9	112.7	217.3	35.4	82.7	2.28	69.1	2.23
PFD	3/10	5/10	2924	120.2	111.2	210.6	34.3	80.5	2.19	69.2	0.01
EDR	2/10	3/10	2991	121.6	111.2	213.4	34.3	80.1	2.21	68.4	0.43
CBR	0/10	0/10	3099	121.0	112.8	216.8	35.6	82.9	2.30	68.4	0.69
QLBWR	0/10	0/10	3098	136.1	122.5	258.6	43.5	87.5	2.40	75.6	0.47
QECV	0/10	0/10	3088	160.9	135.9	322.4	52.4	91.7	2.49	82.9	0.36
BRQ	0/10	1/10	3032	171.5	140.8	346.3	55.9	92.6	2.52	86.0	0.37
NQ	0/10	0/10	3099	125.4	115.1	230.1	37.6	85.0	2.34	70.7	0.68
DQN	1/10	1/10	3096	132.2	120.2	248.4	41.6	86.6	2.38	73.9	0.51
GDQN	1/10	1/10	3096	139.3	125.0	268.5	45.1	88.1	2.41	76.8	0.66
MAPPO	0/10	2/10	2911	130.4	115.6	240.4	37.6	83.5	2.32	73.3	4.09

Table 35: SMALL 150 under C_A. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	0/10	0/10	3086	162.5	136.0	319.8	52.5	91.5	2.49	82.8	0.39
KD	0/10	0/10	3093	151.3	131.5	295.9	49.0	90.8	2.37	80.5	54.58
CD	0/10	0/10	3100	120.3	112.0	214.9	35.2	82.6	2.27	68.0	2.29
PFD	0/10	0/10	3100	117.7	110.2	207.0	34.1	81.1	2.18	66.4	0.02
EDR	0/10	0/10	3100	117.5	110.7	205.2	34.2	80.8	2.21	66.4	0.43
CBR	0/10	0/10	3100	119.8	111.8	214.1	35.0	82.3	2.28	67.6	0.71
QLBWR	0/10	0/10	3098	135.8	121.4	257.1	42.9	87.0	2.39	74.9	0.48
QECV	0/10	0/10	3091	154.3	131.9	304.3	50.2	90.3	2.46	80.9	0.37
BRQ	0/10	1/10	3061	170.2	137.6	341.1	53.9	91.9	2.50	84.2	0.37
NQ	0/10	0/10	3099	126.0	115.7	232.0	38.2	84.8	2.33	70.9	0.70
DQN	0/10	0/10	3097	128.1	116.9	238.0	39.4	85.4	2.36	72.0	0.54
GDQN	0/10	0/10	3094	134.2	120.9	254.0	42.4	87.1	2.40	74.7	0.68
MAPPO	0/10	0/10	3099	124.4	114.1	228.3	37.0	84.4	2.32	70.1	4.15

Table 36: SMALL 150 under C_{AR}. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	0/10	0/10	3081	174.9	142.9	353.1	56.6	94.2	2.55	86.6	0.39
KD	0/10	0/10	3091	148.6	130.3	291.1	48.1	91.6	2.41	80.5	53.99
CD	0/10	0/10	3099	118.8	109.6	216.4	32.9	83.3	2.30	67.1	2.27
PFD	0/10	0/10	3101	115.0	107.1	205.7	31.1	81.4	2.20	64.9	0.02
EDR	0/10	0/10	3100	117.0	108.4	210.1	32.3	81.8	2.24	65.8	0.44
CBR	0/10	0/10	3101	117.9	109.1	213.9	32.4	82.9	2.31	66.6	0.70
QLBWR	0/10	0/10	3093	141.4	124.8	278.1	45.1	89.5	2.45	77.6	0.49
QECV	0/10	1/10	3059	163.0	135.9	329.8	52.7	92.5	2.52	83.8	0.38
BRQ	0/10	2/10	2848	184.0	145.6	382.1	58.3	93.4	2.56	89.7	0.37
NQ	0/10	0/10	3099	122.3	111.5	227.9	34.4	85.4	2.36	69.1	0.70
DQN	0/10	0/10	3099	127.3	115.0	240.9	37.5	86.6	2.40	71.6	0.53
GDQN	1/10	3/10	2946	143.0	124.6	279.6	44.5	88.5	2.45	79.0	0.69
MAPPO	0/10	0/10	3099	123.3	112.4	230.4	35.1	85.4	2.36	69.5	4.25

E.2.3 MID 175

Table 37: MID 175 under C_0. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	0/10	0/10	4790	90.3	80.0	171.2	35.0	52.8	2.13	67.0	0.23
KD	0/10	0/10	4788	96.0	85.8	181.3	38.4	55.0	2.12	71.3	8.32
CD	0/10	0/10	4789	91.0	80.6	171.7	35.4	53.1	2.09	67.5	0.91
PFD	0/10	0/10	4789	91.4	80.9	173.1	35.6	53.3	2.10	67.8	0.02
EDR	1/10	1/10	4792	89.3	80.1	164.7	34.8	52.0	2.08	66.3	0.24
CBR	0/10	0/10	4788	96.7	87.2	180.3	39.0	55.2	2.09	71.9	0.37
QLBWR	0/10	0/10	4790	89.4	79.4	167.9	34.6	52.3	2.12	66.3	0.35
QECV	0/10	0/10	4789	89.3	79.5	167.4	34.6	52.1	2.12	66.3	0.22
BRQ	0/10	0/10	4788	94.0	82.9	180.3	36.9	54.6	2.12	69.8	0.12
NQ	0/10	0/10	4788	95.3	83.5	184.8	37.6	55.1	2.07	70.7	0.53
DQN	0/10	1/10	4571	107.7	82.6	255.6	35.3	52.7	2.09	70.6	0.38
GDQN	0/10	0/10	4789	92.3	81.4	176.1	36.0	53.8	2.08	68.6	0.55
MAPPO	0/10	0/10	4789	90.7	80.3	171.4	35.2	52.9	2.10	67.3	3.76

Table 38: MID 175 under C_A. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	0/10	0/10	4789	89.5	79.5	168.7	34.6	52.4	2.13	66.4	0.24
KD	0/10	0/10	4788	94.7	84.8	177.7	37.6	54.5	2.11	70.3	8.34
CD	0/10	0/10	4789	90.4	80.2	170.3	35.1	52.8	2.09	67.1	0.93
PFD	0/10	0/10	4790	90.7	80.4	171.2	35.2	52.9	2.09	67.3	0.02
EDR	0/10	0/10	4790	88.6	79.6	162.6	34.5	51.6	2.08	65.7	0.25
CBR	0/10	0/10	4789	96.0	86.6	178.5	38.6	54.8	2.08	71.3	0.38
QLBWR	0/10	0/10	4790	88.7	79.0	165.6	34.3	51.9	2.11	65.8	0.36
QECV	2/10	0/10	4789	88.5	78.8	165.0	34.3	51.7	2.11	65.6	0.24
BRQ	0/10	0/10	4789	93.2	82.2	177.8	36.4	54.2	2.12	69.2	0.12
NQ	0/10	0/10	4789	94.0	82.4	181.5	36.8	54.7	2.06	69.8	0.58
DQN	0/10	0/10	4789	91.3	80.6	173.3	35.3	53.4	2.08	67.7	0.41
GDQN	0/10	0/10	4789	91.6	80.8	174.0	35.6	53.4	2.07	68.0	0.58
MAPPO	0/10	0/10	4790	90.1	79.9	169.7	35.0	52.6	2.10	66.8	4.01

Table 39: MID 175 under C_{AR}. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	0/10	0/10	4790	88.4	78.1	167.5	33.4	52.5	2.14	65.6	0.24
KD	0/10	0/10	4789	93.4	83.3	176.5	36.3	54.5	2.12	69.3	8.11
CD	0/10	0/10	4790	89.2	78.7	168.6	33.8	52.8	2.11	66.1	0.92
PFD	0/10	0/10	4789	89.5	79.0	169.9	34.1	53.0	2.11	66.4	0.02
EDR	0/10	0/10	4790	87.2	78.1	160.4	33.2	51.5	2.10	64.6	0.24
CBR	0/10	0/10	4789	94.2	84.4	176.0	36.9	54.7	2.09	69.9	0.37
QLBWR	0/10	0/10	4790	87.7	77.7	164.7	33.2	52.0	2.13	65.0	0.36
QECV	0/10	0/10	4788	87.6	77.6	164.3	33.2	51.9	2.13	65.0	0.24
BRQ	0/10	0/10	4789	91.9	80.6	176.6	35.1	54.2	2.13	68.2	0.12
NQ	0/10	0/10	4789	92.7	80.7	180.1	35.4	54.7	2.07	68.8	0.58
DQN	0/10	0/10	4790	90.2	79.2	172.2	34.2	53.5	2.10	66.9	0.41
GDQN	0/10	0/10	4789	90.1	79.2	172.2	34.2	53.4	2.09	66.9	0.57
MAPPO	0/10	0/10	4789	89.0	78.6	168.5	33.8	52.7	2.12	66.0	4.01

E.2.4 MID 200

Table 40: MID 200 under C_0. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	1/10	1/10	5292	91.6	80.6	175.7	34.2	54.9	2.24	65.7	0.23
KD	0/10	0/10	5290	96.3	85.4	184.3	36.8	57.0	2.22	69.2	8.26
CD	0/10	0/10	5290	91.8	80.8	175.5	34.3	55.0	2.18	65.9	0.93
PFD	0/10	0/10	5291	92.2	81.2	176.4	34.5	55.1	2.19	66.1	0.01
EDR	0/10	0/10	5291	89.9	80.2	167.0	33.8	53.6	2.17	64.5	0.24
CBR	0/10	0/10	5289	97.0	86.8	183.3	37.4	57.1	2.18	69.7	0.38
QLBWR	0/10	1/10	5139	93.2	80.4	175.6	33.7	53.9	2.21	66.3	0.35
QECV	0/10	3/10	4952	97.1	80.6	184.6	33.5	53.2	2.21	67.9	0.23
BRQ	0/10	1/10	5154	97.8	83.4	188.9	35.7	56.1	2.22	69.3	0.12
NQ	1/10	3/10	4918	105.1	82.2	215.3	34.3	54.3	2.18	69.6	0.53
DQN	1/10	3/10	5060	96.9	81.8	184.3	34.6	54.4	2.18	68.5	0.38
GDQN	2/10	5/10	4798	108.1	82.6	245.3	34.0	53.6	2.16	70.8	0.54
MAPPO	0/10	3/10	4949	101.3	82.2	195.3	34.4	54.1	2.19	69.4	4.53

Table 41: MID 200 under C_A. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	0/10	0/10	5291	90.7	80.0	173.2	33.8	54.4	2.22	65.1	0.24
KD	0/10	0/10	5290	95.3	84.7	181.3	36.3	56.5	2.21	68.4	8.28
CD	0/10	0/10	5290	91.2	80.4	173.6	34.1	54.6	2.17	65.4	0.95
PFD	0/10	0/10	5291	91.4	80.6	174.4	34.2	54.7	2.18	65.6	0.02
EDR	0/10	0/10	5291	89.4	79.9	165.6	33.6	53.3	2.16	64.1	0.26
CBR	0/10	0/10	5289	96.0	85.9	180.8	36.7	56.7	2.17	68.9	0.39
QLBWR	1/10	0/10	5290	90.0	79.7	170.1	33.6	53.8	2.21	64.5	0.40
QECV	0/10	0/10	5291	89.7	79.4	169.2	33.6	53.6	2.21	64.3	0.30
BRQ	0/10	0/10	5290	94.1	82.4	182.2	35.4	56.2	2.21	67.6	0.14
NQ	0/10	0/10	5290	91.9	80.7	176.4	34.4	55.0	2.18	65.9	0.72
DQN	0/10	0/10	5290	91.8	80.9	175.5	34.5	54.8	2.19	65.8	0.53
GDQN	0/10	0/10	5290	91.8	80.6	176.0	34.3	55.0	2.18	65.9	0.71
MAPPO	0/10	0/10	5290	91.8	80.9	175.1	34.4	54.8	2.19	65.8	6.12

Table 42: MID 200 under C_{AR}. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	0/10	0/10	5291	89.9	78.8	172.7	32.8	54.5	2.25	64.5	0.24
KD	0/10	0/10	5290	94.3	83.4	180.5	35.2	56.6	2.23	67.7	8.03
CD	0/10	0/10	5291	90.3	79.4	172.9	33.1	54.7	2.20	64.8	0.92
PFD	0/10	0/10	5291	90.4	79.3	173.4	33.1	54.8	2.20	64.9	0.02
EDR	0/10	0/10	5292	88.4	78.9	163.9	32.6	53.3	2.18	63.4	0.24
CBR	0/10	0/10	5290	94.5	84.0	179.0	35.4	56.6	2.19	67.8	0.37
QLBWR	0/10	0/10	5292	89.3	78.8	170.0	32.8	54.0	2.23	64.0	0.42
QECV	0/10	0/10	5292	89.0	78.4	168.8	32.7	53.8	2.23	63.8	0.33
BRQ	0/10	0/10	5290	93.1	81.2	181.3	34.4	56.3	2.23	66.9	0.14
NQ	0/10	0/10	5291	91.0	79.6	175.7	33.3	55.2	2.20	65.3	0.72
DQN	0/10	0/10	5291	91.0	79.7	175.1	33.5	54.9	2.21	65.2	0.53
GDQN	0/10	0/10	5291	90.9	79.4	175.6	33.2	55.1	2.19	65.2	0.71
MAPPO	0/10	0/10	5291	90.8	79.7	174.3	33.3	54.9	2.21	65.1	6.19

E.2.5 MID 225

Table 43: MID 225 under C_0. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	5/10	9/10	4924	107.4	85.8	202.0	36.1	56.0	2.34	77.1	0.23
KD	3/10	6/10	5601	112.7	96.5	220.6	43.8	59.8	2.33	80.4	8.63
CD	1/10	4/10	5431	111.4	86.1	248.3	35.4	55.3	2.27	74.1	0.87
PFD	4/10	8/10	5470	100.7	85.4	192.0	36.0	56.1	2.29	73.7	0.02
EDR	0/10	2/10	5925	93.9	83.5	175.9	35.1	55.7	2.27	68.4	0.25
CBR	2/10	6/10	4925	135.4	97.4	302.8	41.8	57.0	2.27	82.4	0.39
QLBWR	2/10	6/10	4938	122.5	86.3	273.3	35.0	54.3	2.31	76.2	0.42
QECV	1/10	6/10	5232	108.6	85.2	229.7	35.3	55.3	2.33	74.2	0.32
BRQ	6/10	9/10	4590	124.2	89.5	244.1	37.4	56.9	2.31	80.8	0.14
NQ	3/10	6/10	5379	110.0	86.8	235.6	36.2	56.0	2.28	75.4	0.67
DQN	4/10	7/10	5693	99.3	85.6	192.4	36.6	56.9	2.29	73.2	0.48
GDQN	2/10	8/10	4772	126.9	87.5	277.6	35.4	55.0	2.26	78.7	0.64
MAPPO	4/10	7/10	5344	116.8	87.2	257.7	35.9	55.6	2.28	75.3	4.91

Table 44: MID 225 under C_A. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	0/10	0/10	6011	96.5	84.5	188.0	36.4	57.3	2.35	69.8	0.27
KD	0/10	0/10	6006	106.9	93.9	209.4	42.7	60.2	2.33	76.6	9.67
CD	0/10	0/10	6010	95.5	84.2	184.1	36.0	57.0	2.28	69.3	1.13
PFD	0/10	0/10	6011	95.3	84.0	183.6	35.8	57.0	2.30	69.1	0.02
EDR	0/10	0/10	6011	92.7	83.0	173.4	34.8	55.4	2.26	67.2	0.34
CBR	0/10	0/10	6007	103.9	92.6	199.5	41.4	59.4	2.28	75.1	0.50
QLBWR	0/10	0/10	6011	94.1	83.1	180.6	35.3	56.3	2.33	68.3	0.39
QECV	0/10	0/10	6011	94.2	83.1	180.9	35.3	56.3	2.33	68.3	0.28
BRQ	0/10	0/10	6005	103.0	89.6	203.9	40.2	59.5	2.33	74.2	0.13
NQ	0/10	0/10	6010	96.6	84.6	187.4	36.4	57.6	2.29	70.0	0.60
DQN	0/10	0/10	6010	96.5	84.8	187.0	36.5	57.5	2.29	70.0	0.43
GDQN	0/10	0/10	6010	96.5	84.6	187.6	36.3	57.7	2.29	70.0	0.59
MAPPO	0/10	0/10	6011	96.4	84.7	187.0	36.3	57.5	2.30	69.9	4.23

Table 45: MID 225 under C_{AR}. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	0/10	0/10	6010	96.8	84.0	190.7	36.0	57.8	2.39	69.9	0.28
KD	0/10	0/10	6004	106.5	93.1	209.6	41.9	60.3	2.35	76.1	9.67
CD	0/10	0/10	6011	94.3	82.8	182.3	34.7	57.1	2.31	68.4	1.11
PFD	0/10	0/10	6011	94.7	83.1	183.2	34.9	57.2	2.32	68.6	0.02
EDR	0/10	0/10	6011	91.9	81.9	172.6	33.9	55.5	2.29	66.6	0.33
CBR	0/10	0/10	6006	102.3	90.8	197.3	39.8	59.3	2.30	73.9	0.50
QLBWR	1/10	0/10	6011	93.9	82.5	181.5	34.6	56.7	2.36	68.1	0.37
QECV	0/10	0/10	6011	93.4	82.1	180.2	34.3	56.5	2.35	67.7	0.25
BRQ	0/10	0/10	6004	100.5	87.2	199.1	38.1	59.4	2.36	72.6	0.13
NQ	0/10	0/10	6010	95.4	83.2	186.5	35.2	57.7	2.31	69.2	0.58
DQN	0/10	0/10	6011	95.5	83.5	185.9	35.4	57.6	2.31	69.3	0.42
GDQN	0/10	0/10	6010	95.3	83.1	186.4	35.0	57.8	2.31	69.1	0.58
MAPPO	0/10	0/10	6010	95.4	83.4	186.4	35.3	57.6	2.32	69.2	4.28

E.2.6 LARGE 275

Table 46: LARGE 275 under C_0. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	9/10	6/10	3667	117.0	83.1	191.3	33.5	55.8	2.41	80.7	0.32
KD	8/10	4/10	3627	137.3	90.1	217.6	36.3	58.0	2.40	83.4	11.36
CD	7/10	6/10	4836	116.0	84.7	198.3	34.7	56.2	2.35	74.8	1.28
PFD	6/10	7/10	4515	122.3	85.8	215.1	34.5	54.1	2.32	78.2	0.02
EDR	3/10	3/10	5878	104.8	83.9	187.6	34.6	55.0	2.32	67.8	0.38
CBR	9/10	8/10	3907	150.1	93.8	234.8	39.2	58.8	2.38	83.9	0.54
QLBWR	7/10	8/10	4845	117.3	83.5	192.9	34.0	55.3	2.40	74.4	0.36
QECV	9/10	8/10	3626	124.0	85.9	200.3	34.7	55.4	2.39	81.8	0.31
BRQ	10/10	5/10	–	–	–	–	–	–	–	–	–
NQ	8/10	7/10	3490	162.9	87.1	334.3	33.0	52.2	2.33	83.3	0.64
DQN	6/10	6/10	3763	136.7	85.7	233.2	34.1	54.8	2.38	81.6	0.46
GDQN	6/10	9/10	4618	125.9	85.9	225.5	34.4	55.0	2.34	77.7	0.63
MAPPO	10/10	5/10	–	–	–	–	–	–	–	–	–

Table 47: LARGE 275 under C_A. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	0/10	0/10	6772	104.5	86.1	206.0	37.5	58.1	2.45	65.6	0.33
KD	0/10	0/10	6766	107.6	90.5	215.3	39.3	60.5	2.43	68.5	12.15
CD	0/10	0/10	6774	94.4	83.1	182.0	34.9	57.0	2.33	63.1	1.18
PFD	0/10	0/10	6775	94.3	83.1	181.4	34.9	56.8	2.34	63.0	0.02
EDR	0/10	0/10	6776	92.2	82.4	172.9	34.3	55.4	2.31	61.6	0.31
CBR	0/10	0/10	6773	103.4	90.8	201.8	39.3	61.1	2.37	69.0	0.45
QLBWR	1/10	1/10	6649	94.0	82.7	181.2	34.9	56.2	2.38	63.6	0.38
QECV	0/10	0/10	6775	92.6	81.9	176.9	34.2	55.8	2.37	61.9	0.30
BRQ	0/10	0/10	6774	103.9	88.7	208.3	38.9	60.1	2.43	67.9	0.13
NQ	0/10	0/10	6775	93.5	82.4	180.4	34.2	56.8	2.38	62.5	0.70
DQN	0/10	0/10	6775	94.5	82.9	184.0	34.7	57.3	2.39	63.2	0.48
GDQN	0/10	0/10	6776	94.8	83.1	184.6	34.9	57.4	2.35	63.4	0.65
MAPPO	0/10	0/10	6776	93.7	82.7	180.7	34.6	56.6	2.39	62.7	5.32

Table 48: LARGE 275 under C_{AR}. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	0/10	0/10	6776	90.1	78.3	177.2	29.7	57.9	2.52	60.2	0.29
KD	0/10	0/10	6775	94.2	82.3	185.0	31.6	60.1	2.49	63.0	10.32
CD	0/10	0/10	6776	90.6	78.7	177.3	30.0	58.0	2.44	60.5	1.10
PFD	0/10	0/10	6777	90.3	78.6	176.2	30.0	57.8	2.46	60.3	0.02
EDR	0/10	0/10	6776	88.1	78.0	166.8	29.5	56.0	2.41	58.8	0.29
CBR	0/10	0/10	6774	97.3	84.7	192.1	33.2	61.5	2.45	65.1	0.44
QLBWR	0/10	0/10	6776	89.5	78.1	174.4	29.7	57.3	2.49	59.8	0.36
QECV	0/10	0/10	6776	89.0	77.8	172.6	29.5	56.9	2.49	59.4	0.28
BRQ	0/10	0/10	6775	93.4	80.5	185.9	31.1	59.8	2.50	62.4	0.12
NQ	0/10	0/10	6776	90.1	78.4	176.6	29.8	57.8	2.49	60.2	0.57
DQN	0/10	0/10	6776	90.9	78.7	179.8	30.1	58.3	2.50	60.8	0.38
GDQN	0/10	0/10	6776	91.1	78.9	180.1	30.2	58.4	2.46	60.9	0.53
MAPPO	0/10	0/10	6776	90.0	78.5	176.0	29.8	57.7	2.51	60.1	3.62

E.2.7 LARGE 300

Table 49: LARGE 300 under C_0. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	10/10	9/10	–	–	–	–	–	–	–	–	–
KD	9/10	10/10	3663	163.5	91.2	258.4	35.3	57.6	2.47	84.6	10.20
CD	9/10	5/10	4752	123.8	88.0	216.9	34.2	56.8	2.40	78.3	1.08
PFD	9/10	6/10	7280	97.7	85.7	190.6	35.8	59.4	2.45	64.4	0.02
EDR	6/10	7/10	6588	100.5	84.7	185.0	34.7	57.1	2.40	66.3	0.29
CBR	10/10	6/10	–	–	–	–	–	–	–	–	–
QLBWR	9/10	9/10	3643	159.8	86.7	227.9	34.0	54.7	2.45	82.9	0.32
QECV	7/10	6/10	4625	118.0	85.1	200.5	34.3	56.8	2.48	77.4	0.22
BRQ	9/10	6/10	3635	148.0	88.2	241.6	34.6	56.9	2.48	83.8	0.11
NQ	10/10	8/10	–	–	–	–	–	–	–	–	–
DQN	9/10	6/10	3635	132.2	86.9	217.7	35.2	56.8	2.41	83.6	0.35
GDQN	10/10	7/10	–	–	–	–	–	–	–	–	–
MAPPO	10/10	8/10	–	–	–	–	–	–	–	–	–

Table 50: LARGE 300 under C_A. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	1/10	1/10	7232	112.4	90.6	241.8	41.0	61.8	2.62	69.2	0.30
KD	0/10	0/10	6983	161.2	111.8	369.3	55.2	69.8	2.81	83.5	12.09
CD	0/10	0/10	7265	97.1	85.1	189.3	35.5	59.0	2.42	63.9	1.11
PFD	0/10	0/10	7265	96.5	84.8	187.3	35.4	58.6	2.43	63.5	0.02
EDR	0/10	0/10	7266	94.4	84.2	179.0	34.7	57.2	2.39	62.1	0.32
CBR	1/10	1/10	7252	120.9	101.7	250.0	47.5	65.9	2.52	76.5	0.47
QLBWR	1/10	0/10	7122	96.1	84.2	186.7	34.9	58.1	2.48	63.8	0.33
QECV	0/10	0/10	7260	99.2	85.3	198.0	36.3	58.6	2.50	64.0	0.23
BRQ	0/10	1/10	6961	133.6	97.9	296.9	46.3	64.7	2.69	75.3	0.12
NQ	0/10	0/10	7264	100.0	85.8	201.6	36.2	61.2	2.47	65.8	0.56
DQN	0/10	0/10	7265	97.4	84.6	192.2	35.0	59.9	2.43	64.1	0.38
GDQN	0/10	0/10	7265	96.7	84.7	188.9	35.3	58.9	2.46	63.6	0.54
MAPPO	2/10	1/10	7219	109.4	91.9	224.6	41.2	62.8	2.53	70.4	3.73

Table 51: LARGE 300 under C_{AR}. DL is throughput-collapse runs; DE is runs with at least one detector event. Means exclude DL runs.

Method	DL	DE	Throughput	Cycle	P50	P95	Pickup	Delivery	TTI	Util.	Route ms
D	0/10	0/10	7263	93.5	80.4	187.4	30.4	60.6	2.64	61.4	0.30
KD	0/10	0/10	6982	108.2	88.9	228.0	36.2	64.0	2.66	68.2	10.72
CD	0/10	0/10	7267	92.9	80.5	183.9	30.3	60.1	2.54	61.1	1.13
PFD	0/10	0/10	7266	92.7	80.4	182.6	30.2	59.9	2.57	60.9	0.02
EDR	0/10	0/10	7267	90.2	79.6	172.7	29.7	58.0	2.50	59.3	0.29
CBR	0/10	1/10	7011	118.8	96.7	248.3	42.2	66.3	2.59	73.7	0.43
QLBWR	1/10	0/10	7265	98.1	83.3	201.0	33.2	60.7	2.64	63.4	0.33
QECV	0/10	0/10	7267	91.5	79.6	179.8	29.8	59.2	2.61	60.1	0.23
BRQ	0/10	0/10	7251	100.9	83.6	208.7	33.2	62.3	2.66	64.4	0.11
NQ	0/10	0/10	7266	95.4	81.0	194.8	30.7	62.2	2.58	62.8	0.56
DQN	0/10	0/10	7266	93.9	80.5	188.5	30.3	61.1	2.55	61.8	0.37
GDQN	0/10	0/10	7266	93.4	80.5	185.5	30.5	60.3	2.59	61.3	0.53
MAPPO	0/10	0/10	7081	98.3	83.5	200.5	32.5	62.8	2.62	65.4	3.37

E.3 Transfer across Fleet Size and Topology

The main comparison evaluates each learned method using the checkpoint trained for the corresponding scenario. We assess transfer separately by freezing a checkpoint trained on a source scenario and evaluating it on an unseen target scenario without further adaptation. All transfer experiments use admission control (C_A), run for $20{,}000$ simulated seconds with seeds 0–9, and are compared against the same method using the checkpoint trained for the target scenario. Two axes are covered. Cross-fleet transfer keeps the layout and changes fleet size and offered load, and covers all seven learned methods over six ordered pairs, for 420 runs. Cross-topology transfer changes the layout, and covers the four neural methods over six ordered pairs, for 240 runs. Table 52 reports the change in mean cycle time relative to the target scenario's own checkpoint.

Table 52: Change in mean cycle time when a checkpoint trained on the source scenario is evaluated on the target scenario, in percent, relative to the checkpoint trained for the target scenario. Both sides average non-collapse runs. $^{\dagger d}$ marks d throughput-collapse runs among the ten transfer runs of that cell.

(a) Cross-fleet transfer: same layout, different fleet size and load

Transfer	QLBWR	QECV	BRQ	NQ	DQN	GDQN	MAPPO
S100 → S150	$+21.9^{\dagger 1}$	$+3.2$	$-1.2^{\dagger 1}$	$+0.9$	-2.2	$+1.8$	$+30.5$
S150 → S100	-0.8	-0.6	$+0.0$	-0.6	$+0.6$	-0.3	-0.6
M175 → M225	$+0.0$	$-0.3^{\dagger 1}$	-1.8	$+24.5$	$+0.3$	$+1.1$	-1.2
M225 → M175	$+11.4$	$+0.1$	$+0.4$	-2.9	-0.3	-0.5	$+0.9$
L275 → L300	$+5.2^{\dagger 2}$	-4.1	$+16.1$	-3.7	$+3.2$	$+2.2$	-10.8
L300 → L275	$+1.1^{\dagger 1}$	-0.1	$+1.0$	$+2.7$	$+0.2$	-1.1	$+4.7$

(b) Cross-topology transfer: different layout

Transfer	NQ	DQN	GDQN	MAPPO
S150 → M200	$+1.3$	$+1.6$	$+0.5$	$+0.1$
S150 → L300	$+0.1^{\dagger 1}$	$+1.8$	$+1.0$	-10.7
M200 → S150	$+0.9$	$+5.2$	$+0.1$	$+38.5^{\dagger 1}$
M200 → L300	-2.4	-0.0	$+0.0$	-10.3
L300 → S150	-2.6	-1.2	-2.3	$+20.2^{\dagger 1}$
L300 → M200	$+2.0$	$+1.6$	-0.6	$+1.3$

Transfer does not cost liveness. Six of the 420 cross-fleet transfer runs end in throughput collapse, which is exactly the number recorded by the corresponding locally trained checkpoints, and three of the 240 cross-topology runs collapse. Served throughput is likewise unchanged, with a median difference of 0.0% on both axes and a worst case of -4.8% across fleets and -10.7% across layouts. Cycle time moves little in most cells: the median change is $+0.2\%$ across fleets and $+0.3\%$ across layouts, and 35 of 42 and 19 of 24 cells stay within $\pm 5\%$. A checkpoint taken from another scenario is therefore usable, and the risk of transfer shows up as efficiency loss in specific cells rather than as loss of service.

Direction matters more than distance. Across fleets, the transfers that cost something are those into a denser scenario. Moving a checkpoint to a larger fleet gives a median change of $+0.9\%$ and the three largest regressions of panel (a): MAPPO $+30.5\%$ and QLBWR $+21.9\%$ from SMALL 100 to SMALL 150, and NQ $+24.5\%$ from MID 175 to MID 225. Moving to a smaller fleet on the same layout is close to free, with a median of 0.0% and only one cell above 5%. This asymmetry is what the underlying decision problem suggests: a policy trained at low density has not met the congestion it will face at high density, while a policy trained at high density is conservative when it is moved to a sparser network.

Layout transfer separates the methods. Across layouts, GDQN is the most stable, with a median change of $+0.1\%$ and no cell above $+1.0\%$, consistent with a representation built from aggregated neighborhood traffic rather than from node identities. NQ and DQN, whose features include node indices, nevertheless stay within $+5.2\%$ in every cell. MAPPO carries both large regressions, $+38.5\%$ from MID 200 and $+20.2\%$ from LARGE 300, and both are transfers into SMALL 150; its two transfers into LARGE 300 instead improve on the locally trained checkpoint by about 10%. Transfers into MID 200 and LARGE 300 are otherwise uniformly small, so the sensitivity of layout transfer depends on the target rather than on the distance between source and target.

These results give the shipped checkpoints a second use beyond the main comparison: they are a starting point for studying transfer, and the matrix above provides the reference against which a transfer-oriented method can be measured.

F BENCHMARK AVAILABILITY AND LICENSING

We plan to publicly release the packaged AMHSTRAFFICLAB simulation platform, all benchmark maps and their demand and region configurations, implementations of the evaluated algorithms, and the experimental configurations and scripts needed to reproduce the reported experiments. The materials will be available at `https://github.com/AMHSTrafficLab/AMHSTrafficLab`. Industrial information incorporated into the benchmark has been anonymized. Scenario construction is described in Appendix C, and installation and experiment instructions are provided in Appendix G.

We also plan to provide a project website at `https://amhstrafficlab.github.io` to introduce the benchmark and showcase the simulation environment. The website will provide links to the released materials and documentation.

The simulation platform and algorithm implementations will be distributed under the MIT License, and the maps and scenario configurations under the Creative Commons Attribution 4.0 International License (CC BY 4.0). The release identifier corresponding to the experiments in this paper will be provided with the reproduction materials.

G INSTALLATION, PLATFORM USE, AND REPRODUCTION

This appendix describes how to install the platform, run a simulation, and evaluate the paper scenarios. The release provides the platform wheel, documentation, dependencies, licenses, example maps and profiles, and separate experiment materials comprising paper scenarios, evaluation configurations, batch-running tools, and checkpoint files.

G.1 INSTALLATION

Obtain the release from the project location in Appendix F. Use Linux x86_64 with Python 3.11. The native runtime requires glibc ≥ 2.34 and a libstdc++ exporting `GLIBCXX_3.4.30`. GUI use additionally requires a working display and OpenGL.

From the extracted release directory, create an environment and install the platform:

```
python3.11 -m venv .venv
. .venv/bin/activate
python -m pip install --only-binary=:all: \
  -c platform/requirements-runtime.lock \
  platform/amhslab-0.1.1-cp311-cp311-manylinux_2_34_x86_64.whl
python -m pip check
amhslab environment_check
```

Keep this environment active for subsequent commands. Installation details and environment troubleshooting are provided in the accompanying documentation.

G.2 RUNNING THE PLATFORM

The example materials contain maps and configuration profiles. From the release directory, validate the synthetic profile and run a 30-second headless simulation:

```
amhslab config_validate --config examples/configs/synthetic.json
amhslab headless --config examples/configs/synthetic.json \
  --until 30 --out outputs/smoke
```

Check that the run completes, reports the requested simulation duration, and writes its exports to the specified output directory.

Graphical interface. Launch the same scenario in a session with a display:

```
amhslab gui --config examples/configs/synthetic.json
```

Click `Run` to start, `Stop` to pause, and `Run` again to continue. Close the window to exit. The map editor can be opened with:

```
amhslab map_editor --map examples/maps/synthetic.xlsx
```

Save edited maps separately and update the corresponding profile paths.

For custom routing and regional policies, follow the interfaces and code examples in Appendix B. Register a custom routing strategy and invoke `run_simulation` in the same Python process.

G.3 PAPER SCENARIOS AND BATCH EVALUATION

The experiment materials include all seven paper maps and their paired region workbooks, identified in Appendix C. Use the supplied evaluation configurations to select the map, workload, fleet size, routing method, and regional-control setting. The main experiments use a horizon of 20,000 simulated seconds and seeds 0–9.

The batch-running tools are supplied with the experiment materials. Install their Python 3.11 dependencies, then run the matrices from the research directory:

```
cd reproduction
python -m benchmarks.matrix_benchmark \
  --spec experiments/routing_iclr_main13_classical_20000s_seeds0_9.yaml
python -m benchmarks.matrix_benchmark \
  --spec experiments/routing_iclr_main13_learned_20000s_seeds0_9.yaml
```

Add `--dry-run` to inspect the expanded experiment plan. Add `--resume` to continue a batch using its completed run records. Keep the inputs unchanged when resuming; use a separate output directory for changed configurations.

Checkpoint loading. Learned-policy weights are supplied as separate checkpoint files with their method and training-scenario correspondence. Set the corresponding loading option in `routing_options` to the selected file and retain the frozen-evaluation settings in the supplied YAML. Each run initializes a fresh simulation and loads its checkpoint. Transfer evaluation uses the source-scenario checkpoint specified for the target experiment.

G.4 Checking Generated Results

The batch runner writes results locally, including per-run `result.json`, `runs.csv`, and `aggregate.csv`. Check completion status, simulation duration, seed coverage, and the effective configuration for each run.

Use the metric definitions and statistical procedure in Appendix D.4.3 when comparing newly generated results with the paper. In particular, compute DL and DE over all seeds and apply the stated collapse filter before averaging the remaining metrics. Record the platform version, Python 3.11 patch version, dependencies, hardware, and worker count with the generated results.