AI

MDPI

Article

Two-Stage Attention-Based Insertion Policy for the Dynamic Vehicle Routing Problem

Souleymane Balde [1,*], Oussama Dribel [1,2,3] and Fouad Riane [1,2,4]

[1] Laboratoire de Recherche en Ingénierie, Innovation des Systèmes Avancés (LISA), Faculty of Sciences and Techniques, Hassan First University of Settat, Route de Casablanca, P.O. Box 577, Settat 26000, Morocco; oussama.dribel@centrale-casablanca.ma (O.D.); fouad.riane@centrale-casablanca.ma (F.R.)
[2] Complex Systems and Interactions, Ecole Centrale Casablanca, Casablanca 27187, Morocco
[3] Laboratoire d'Automatique, de Mécanique et d'Informatique Industrielles et Humaines (LAMIH), Campus Mont Houy, Université Polytechnique Hauts-de-France, UMR 8201, Centre National de la Recherche Scientifique (CNRS), F-59313 Valenciennes, France
[4] Industrial Engineering Laboratory (LGI), CentraleSupélec, Université Paris-Saclay, 3 rue Joliot-Curie, F-91192 Gif-sur-Yvette, France
* Correspondence: souleymane.balde@centrale-casablanca.ma

Abstract

This paper addresses dynamic vehicle routing with online requests, where insertion heuristics follow fixed rules ignoring the effect of each insertion on later requests. It proposes a two-stage attention-based insertion policy: a first stage selects a vehicle, and a second stage an insertion position within its route. Routes and fleet are encoded by attention, and the policy is trained with REINFORCE and a greedy rollout baseline on the final episode cost instead of immediate insertion costs. It is compared with cheapest, nearest and random insertion and with a single-stage policy over all vehicle–position pairs, on instances with 10 to 120 customers and degrees of dynamism from 10% to 90%. Both learned policies outperform nearest and random insertion and generalize to unseen sizes and degrees of dynamism. When most customers are dynamic on instances with up to 35 customers, the single-stage policy outperforms cheapest insertion by up to 3.1% and the two-stage policy matches it; otherwise, cheapest insertion remains better. The single-stage policy is cheaper in most settings, but the two-stage policy decides up to 2.7 and trains up to 5.7 times faster, for a cost at most 3.6% higher, and becomes cheaper on large, highly dynamic instances.

Keywords: dynamic vehicle routing; online insertion; attention mechanism; deep reinforcement learning; neural combinatorial optimization

Check for updates

Academic Editor: Demos T. Tsahalis

Received: 1 September 2026
Revised: 1 October 2026
Accepted: 15 October 2026
Published: 17 October 2026

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

1. Introduction

The Vehicle Routing Problem (VRP) is a central problem in combinatorial optimization with numerous applications in logistics and transportation systems. Given a set of customers with spatial locations and demands, and possibly additional constraints, the objective is to design a set of vehicle routes that minimize operational costs while satisfying all constraints. In many services, however, the information needed to plan routes is not fully available before operations begin. A courier may receive a new pickup request after leaving the depot, a same-day delivery service may accept additional orders during the day, and in intralogistics, automated guided vehicles receive new transport tasks while they are still executing previous ones [1]. These situations define the Dynamic Vehicle Routing Problem (DVRP), in which routing decisions must be revised as information is revealed [2,3]. This

paper focuses on the most common source of dynamism in such services: the online arrival of previously unknown customer requests.

Unlike the static setting, where the full set of customers is known in advance and routes are constructed from scratch, the DVRP involves modifying partially executed routes while preserving feasibility and operational efficiency. When a request is revealed, vehicles are already traveling, part of their capacity is reserved for customers they have not yet reached, and movements that have started cannot be undone. A response is required quickly, and its quality depends on requests that are not yet known. This fundamentally changes the nature of the decision process: the problem becomes one of incremental adjustment rather than global construction.

Classical approaches address this difficulty in different ways. Insertion heuristics [4] integrate each new request at the feasible position of least additional cost; they remain widely used because they are fast, simple, and return a feasible decision whenever one exists. Rolling-horizon strategies re-optimize the unexecuted part of the plan when time permits, for instance, with adaptive large neighborhood search [2,5], and scenario-based approaches sample possible future requests to evaluate current decisions [6]. These heuristics are, however, handcrafted: they apply a fixed rule designed by experts, such as placing each request at the position of least immediate additional cost. They are myopic in this sense. Each decision is evaluated in isolation, following the same rule whatever the instance, without considering how it constrains the insertion of later requests.

Reinforcement learning (RL) offers a complementary approach by learning decision rules from repeated interactions with a routing environment. A learned policy can exploit recurring patterns in customer locations, demands and fleet states. In dynamic routing with stochastic customers, Joe and Lau [7] showed that a learned value function combined with a routing heuristic improves on myopic policies, particularly for high degrees of dynamism. Recent studies nevertheless stress that performance depends on the state representation and on the learning setting, rather than on the use of neural networks alone [8,9].

In parallel, attention-based neural architectures have opened new perspectives for learning routing heuristics directly from data. Rather than explicitly designing routing rules, encoder–decoder models with attention learn constructive policies that sequentially build solutions by exploiting structural patterns in problem instances [10,11]. A seminal contribution in this field is the fully attention-based model proposed by Kool et al. [11] for the Capacitated VRP (CVRP). Attention is well suited to routing because each decision depends on relations among customers, vehicles and the current plan, which the mechanism weighs explicitly. These models, however, are mostly designed for static problems: they construct complete solutions from scratch by selecting the next customer to visit. In the DVRP, the natural decision is different. A revealed request must be assigned to a vehicle and placed within the remaining part of its route, while the relative order of the visits already planned is preserved.

In this work, we therefore consider an attention-based policy specifically designed for dynamic insertion decisions. Instead of constructing routes sequentially from scratch, decisions are triggered by the arrival of new customers. Each revealed request is handled through a two-stage decision: first selecting a vehicle, and then selecting an insertion position within the remaining route of that vehicle. This insertion-based perspective preserves the operational structure of dynamic routing and of the insertion heuristics used in practice but replaces their fixed evaluation rule with a trainable policy that scores vehicle–position decisions from the current routing state. Planned customers are contextualized within their route by self-attention, each candidate insertion position is represented by the two visits it connects, and vehicles are compared with one another through attention across

the fleet. The policy is trained with REINFORCE on the cumulative cost of all insertions of an episode, which equals the increase in total route length with respect to the initial plan.

The main contributions of this paper are as follows:

- An arrival-triggered formulation of the DVRP. Each new request is inserted into the modifiable part of an executing plan.
- A two-stage insertion policy. It first selects a vehicle, then an insertion position within the remaining route of that vehicle.
- An attention-based encoding of the vehicles, the already known customers, and the candidate insertion positions. These encodings are used both to select the vehicle and to select the insertion position.
- An experimental comparison of the proposed policy with classical insertion heuristics and a single-stage learning policy. The proposed approach is compared with the cheapest, nearest, and random insertion heuristics, as well as with a single-stage policy that jointly selects vehicle–position pairs. The comparison covers several instance sizes and degrees of dynamism.

The remainder of the paper is organized as follows. Section 2 reviews the relevant classical and learning-based approaches. Section 3 formulates the dynamic decision setting, and Section 4 presents the proposed model and its training procedure. Section 5 describes the computational setting and evaluation protocol. Section 6 reports the experimental results, which are discussed in Section 7. Finally, Section 8 concludes the paper and outlines directions for future research.

2. Related Work

In this section, we review the classical and learning-based approaches most relevant to the online insertion of customer requests. Dynamism in vehicle routing can arise from several sources [2,3]. New customers may place orders while vehicles are already on the road. The demand of a known customer may only be confirmed at the time of the visit. Travel times may also change with traffic conditions during the day. Each of these situations calls for a different decision: integrating a new visit, revising quantities, or re-timing routes. This paper considers the first source, namely requests that arrive during operations, and handles each of them by inserting it into the routes being executed. For this setting, the degree of dynamism is commonly measured as the proportion of requests revealed after operations begin [2].

Classical approaches provide the main foundations for responding to such arrivals. Insertion heuristics [4] assign each new customer to the feasible location producing the smallest immediate cost increase. For distance-based routing, this increase is obtained from the affected edges without recomputing the full route, which makes cheapest insertion a meaningful practical baseline. Metaheuristics explore more substantial changes to the plan. Adaptive large neighborhood search [5], for instance, repeatedly destroys and repairs parts of a solution and can be embedded in a rolling-horizon strategy that re-optimizes the remaining plan as requests arrive, using insertion for an immediate feasible response and additional search when time permits [2]. Montemanni et al. [12] applied this principle with an ant colony system that re-optimizes the known requests over successive time slices, and their dynamic instances remain a standard test set for the DVRP. Scenario-based planning goes further by sampling possible future customers and evaluating present decisions against these scenarios [6].

RL introduces another way to use experience in these sequential decisions. Joe and Lau [7] formulate the DVRP with stochastic customers as a route-based Markov decision process and combine a neural value function, learned by temporal-difference learning, with simulated annealing; their approach improves on myopic policies and on scenario-based

and approximate value iteration approaches, notably when the degree of dynamism is high. Konovalenko and Hvattum [8] consider the immediate acceptance or rejection of incoming requests and show that the design of the state space, including derived features and feature transformations, significantly affects the performance of deep RL. Akkerman et al. [9] compare linear and neural value-function and policy approximations on several DVRPs with stochastic requests, including a same-day delivery case. In an intralogistics application, Wang et al. [1] use a two-layer deep RL scheme to select vehicle and task dispatching rules for multi-load automated guided vehicles. In these works, the routes themselves are built or updated by a heuristic or an optimization procedure, while the learned model guides this procedure or takes higher-level decisions such as acceptance or dispatching.

Neural combinatorial optimization, in contrast, learns the routing decisions themselves. Pointer Networks [13] use attention to output a sequence of input nodes, but their recurrent encoder imposes an artificial ordering on inherently unordered node sets. Nazari et al. [10] removed the recurrent encoder, separated static customer information from state-dependent information such as remaining demand, and trained the model with RL; their study also includes an extension in which customers arrive stochastically. A major breakthrough is the fully attention-based encoder–decoder proposed by Kool et al. [11]. Their model relies on multi-head attention to produce permutation-invariant node embeddings, and the decoder constructs routes sequentially using a context vector representing the current partial solution and an attention mechanism over candidate nodes. The model is trained with REINFORCE and a greedy rollout baseline, a scheme that we also adopt. This architecture provides a useful foundation, but next-customer selection differs from choosing where to insert a request into an existing route.

Several works have introduced explicit vehicle decisions into this paradigm. For the heterogeneous capacitated VRP, Li et al. [14] decompose each construction step into two decisions. A first decoder selects the vehicle to move, and a second decoder selects the next customer appended to its route. Both decoders are trained jointly with a greedy rollout baseline, for min–max and min-sum objectives. Liu et al. [15] similarly use two decoders, for vehicle selection and passenger allocation, to schedule heterogeneous autonomous vehicles in on-demand last-mile transportation. In dynamic crowd logistics, Xiang et al. [16] combine an upper-level agent, which decides when to match batches of orders with crowd drivers, with lower-level agents that plan routes using a dual-decoder attention model. These studies show the value of structuring fleet decisions hierarchically, with a first choice at the vehicle level followed by a finer-grained action. They differ in the nature of this second action, which appends a customer to a route under construction, allocates passengers, or plans routes for a batch of orders. The present work follows this hierarchical principle and applies it to a different action: inserting a newly revealed request at a chosen position within the remaining route of the selected vehicle.

Attention has also been applied directly to dynamic and stochastic routing. In MARDAM, Bono et al. [17] represent customers and vehicle states with attention and let each vehicle choose its next customer as the environment evolves, including a variant in which part of the requests are revealed online. Pan and Liu [18] design a partially observable Markov decision process with a dynamic attention mechanism for a routing problem in which customer demands change during operations. For electric vehicle routing with pickup–delivery and time windows, Guan et al. [19] use a heterogeneous attention encoder to capture the distinct roles of depots, customers and charging stations, together with a dynamic-aware context embedding that tracks state transitions and temporal feasibility in the decoder. A related dynamic-aware architecture addresses capacitated electric vehicle routing with service time windows and time-of-use electricity prices [20]. These works are relevant to the representation of evolving vehicle states and to feasibility handling, which

also shape our vehicle context. Their notion of dynamism, however, concerns changing loads, energy or time during route execution, or requests to which a vehicle responds by choosing its next destination. Inserting a request into a previously planned sequence imposes a different action structure from successively choosing the next destination.

Learning has also been applied to insertion itself, either to repair existing solutions or to construct new ones. Hottung and Tierney [21] integrate an attention-based repair operator, trained by RL, into a large neighborhood search that repeatedly destroys and repairs routing solutions. Luo et al. [22] replace the appending paradigm of constructive neural solvers with an insertion-based one, in which unvisited nodes are inserted at any valid position of the partial solution, and propose an architecture dedicated to the prediction of insertion positions. These methods show that learned insertion predates the present work, and the second stage of our policy follows the same principle. However, they operate on static instances whose customers are all known in advance, and they either repair a solution during search or construct one from scratch. Integrating an externally arriving request during execution, with committed movements, reserved capacity, and a fleet-level vehicle choice, serves a different purpose.

In summary, the proposed approach differs from the methods reviewed above in three respects. First, decisions are triggered by the arrival of a request during route execution, rather than taken at each step of the construction of a solution or each time a vehicle becomes available. Second, the learned action is the insertion of this request at a position within the remaining route of a vehicle, which preserves the order of the visits already planned and the movements already committed. Constructive policies select the next customer to visit instead [10,11,14]. Learned insertion methods, for their part, operate on static instances [21,22]. Third, candidate insertion positions are represented by the two visits they connect, within an attention-based encoding of each route and of the fleet that is shared by both decision stages.

Table 1 summarizes the learning-based methods most closely related to this work, according to the problem they address, the source of dynamism they consider and the decision they learn.

Table 1. Learning-based routing methods most closely related to this work.

Reference	Problem	Source of Dynamism	Learned Decision
Nazari et al. [10]	CVRP	Construction state; stochastic arrivals in an extension	Next customer
Kool et al. [11]	TSP, CVRP	None	Next customer
Li et al. [14]	Heterogeneous CVRP	None	Vehicle, then next customer
Liu et al. [15]	Last-mile transport	Real-time passenger demand	Vehicle, then passenger allocation
Xiang et al. [16]	Crowd logistics	Arriving orders and drivers	Batch-matching timing; route planning
Joe and Lau [7]	DVRP with time windows	Customer arrivals	Value function guiding re-optimization
Konovalenko and Hvattum [8]	DVRP	Customer arrivals	Acceptance or rejection
Bono et al. [17]	Stochastic and dynamic CVRPTW	Travel times, demands, online requests	Next customer of each vehicle
Pan and Liu [18]	Dynamic and uncertain VRP	Changing demands	Next customer
Guan et al. [19]	Electric PDP with time windows	Vehicle energy and time states	Next node
Hottung and Tierney [21]	CVRP	None	Repair of a destroyed solution
Luo et al. [22]	TSP, CVRP	None	Insertion position of an unvisited node
This work	CVRP with online requests	Customer arrivals during execution	Vehicle, then insertion position in its remaining route

3. Problem Formulation

The DVRP fundamentally differs from its static counterpart. At a given decision epoch, vehicles are already executing previously constructed routes, and the decision-maker must adapt these ongoing plans in response to newly revealed customer requests. This section defines the problem considered in this paper and the insertion decision that has to be taken when a request is revealed.

3.1. Static and Dynamic Customers

We consider a depot, indexed by 0, and a set $\mathcal{N} = \{1, \dots, n\}$ of customers. Each customer i is characterized by its spatial coordinates $(x_i, y_i) \in \mathbb{R}^2$, its demand d_i and its revelation time t_i^{rev}, at which the request becomes known. The travel cost $c(i, j)$ between two locations is their Euclidean distance. A fleet of K identical vehicles of capacity Q, based at the depot, serves the customers. Vehicles travel at constant speed, so that travel times are proportional to distances, and services are instantaneous.

The customers are divided into two groups. *Static customers*, $\mathcal{N}^{\text{stat}}$, are known before operations begin ($t_i^{\text{rev}} = 0$); an initial plan serving them is built at time 0, and the vehicles start executing it. *Dynamic customers*, $\mathcal{N}^{\text{dyn}} = \mathcal{N} \setminus \mathcal{N}^{\text{stat}}$, are revealed online during an operating horizon H: their existence, location and demand are unknown until their revelation time. Following the standard definition [2], the degree of dynamism is the proportion of dynamic customers, $\delta = n^{\text{dyn}}/n$, where n^{dyn} is the number of dynamic customers. The construction of the initial plan is not the subject of this work: the decisions studied here concern the dynamic customers only. The objective is to serve all customers while minimizing the total distance traveled by the fleet over the whole operating period, although the dynamic customers are not known when the first routing decisions are made.

3.2. Dynamic Insertion Decisions

We consider a dynamic routing process evolving over continuous time $\tau \in \mathbb{R}_+$. At a given time τ, each vehicle $k \in \{1, \dots, K\}$ follows a partially executed route. A leg that has been started cannot be interrupted. The position of vehicle k, denoted $p_k(\tau)$, is therefore defined as its current location if it is waiting, and as the destination of its current leg if it is traveling: it is the first point from which its route can be modified. We denote by

$$\mathcal{R}_k(\tau) = \left(r_{k,1}, \dots, r_{k,m_k}\right) \tag{1}$$

the ordered sequence of customers that remain to be served by vehicle k after this position, excluding customers that have already been visited. The remaining route of vehicle k is thus the sequence $\left(p_k(\tau), r_{k,1}, \dots, r_{k,m_k}, 0\right)$, ending with the return to the depot.

Each customer i in a remaining route is associated with its coordinates, its demand d_i and an estimated arrival time $\eta_i(\tau)$ induced by the current route of its vehicle. The demands of the customers in $\mathcal{R}_k(\tau)$ are reserved, and the remaining capacity $\hat{D}_k(\tau)$ of vehicle k is the load available at $p_k(\tau)$ once these demands are deducted.

When a dynamic customer j is revealed, at time $\tau = t_j^{\text{rev}}$, the decision consists of determining:

- A vehicle k that will serve customer j;
- An insertion position s within the remaining route of vehicle k.

A position $s \in \{0, \dots, m_k\}$ inserts j between two consecutive elements of the remaining route, denoted $\text{prev}(s)$ and $\text{next}(s)$. Position $s = 0$ inserts j immediately after the vehicle position and position $s = m_k$ just before the return to the depot. The insertion preserves the relative order of the customers already planned. When several customers are revealed at the same time, they are inserted one after the other.

An insertion of customer j into the route of vehicle k is feasible if the vehicle can still serve it,

$$d_j \leq \hat{D}_k(\tau), \tag{2}$$

in which case every position of its remaining route is feasible. If no vehicle of the fleet can serve the request, an additional vehicle is used at a penalty; this case remains exceptional in the instances considered.

The immediate cost of inserting j at position s of vehicle k is the increase in route length caused by the insertion

$$\Delta_{j,\tau}(k,s) = c\big(\text{prev}(s), j\big) + c\big(j, \text{next}(s)\big) - c\big(\text{prev}(s), \text{next}(s)\big). \tag{3}$$

Cheapest insertion selects, for each request, the feasible pair (k,s) that minimizes this immediate cost. The objective of the problem, however, is the total distance traveled at the end of the operating period. Since each insertion modifies the routes into which later customers will be inserted, minimizing the cost of each insertion separately does not, in general, minimize this total distance. The insertion policy is therefore required to select (k,s) from the information available at time τ only, namely the current routes, the remaining capacities and the new request, so as to minimize the expected total distance over the future arrivals.

4. Two-Stage Attention-Based Insertion Policy

From a learning perspective, directly modeling the full joint action space as a single distribution over all feasible vehicle–position pairs requires comparing every candidate position of every vehicle at each decision. To structure this decision, we decompose it into two conditional choices: first selecting a vehicle, and then selecting an insertion position within the remaining route of that vehicle. The policy is factorized as

$$\pi_\theta(k, s \mid S_\tau, j) = p_\theta(k \mid S_\tau, j)\; p_\theta(s \mid S_\tau, j, k), \tag{4}$$

where S_τ denotes the routing state at time τ, that is, the information available when customer j is revealed (Section 3.2), and θ the trainable parameters. The first stage captures global assignment decisions by comparing the new customer with each vehicle; the second stage captures local structural decisions by comparing it with the insertion positions of the selected vehicle.

Figure 1 provides an overview of the proposed policy, from the routing state to the insertion decision. The left panel shows the system state at the time a request is revealed: vehicles are executing their routes from the depot, and new customers, such as A and B in the example, must be integrated into these routes. The policy first encodes this state independently of the new request (second panel). Each remaining route, including the vehicle position and the final depot, is processed separately: the coordinates, demand and expected arrival delay of each customer are embedded, a rank encoding indicates the order of the visits, and intra-route self-attention produces contextualized route tokens. The state of each vehicle, combining its position, remaining capacity and traveled distance, is encoded by a vehicle MLP, and each candidate insertion position is represented from its predecessor, its successor and the vehicle state. Each vehicle is then summarized by attention over its insertion positions, and the vehicles are compared with one another by self-attention across the fleet. The insertion decision is taken by two successive attention stages (third panel). In the first stage, the embedding of the new customer is compared with the fleet representations, and a masked softmax over the feasible vehicles selects a vehicle. In the second stage, the insertion positions of the selected vehicle are updated with

its fleet representation and compared with the new customer, and a masked softmax over the feasible positions selects the insertion position. The right panel shows the resulting routes: in the example, customer A is inserted between customers 4 and 6. When several customers are revealed at the same time, they are processed sequentially, the state being updated after each insertion. The components of the policy are detailed in Sections 4.2–4.4 and its training in Section 4.5.

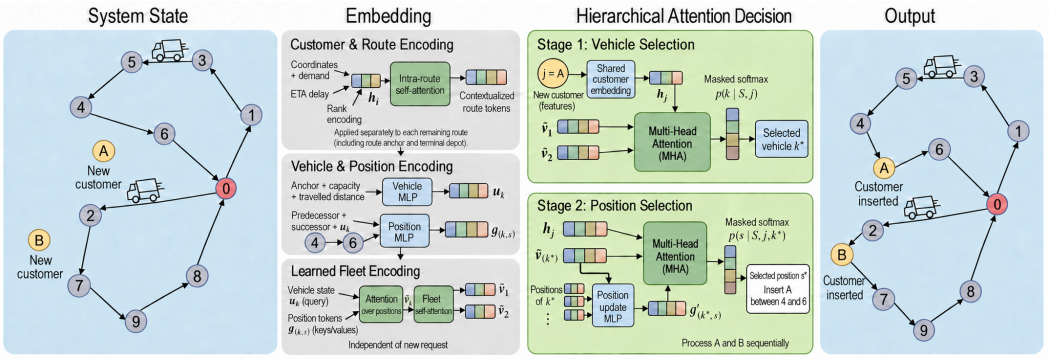


Figure 1. Overview of the proposed two-stage attention-based insertion policy. Numbered nodes are the customers already known at this time (static customers), nodes A and B are newly revealed customers, and node 0 is the depot.

4.1. Attention Blocks

All attention-based components of the policy rely on the same building block, which follows the encoder layer of the Transformer [23] as used in attention-based routing [11]. Given a set of N tokens $X \in \mathbb{R}^{N \times d_h}$, where d_h is the embedding dimension, multi-head attention (MHA) with M heads computes, for each head m, queries, keys and values $Q^{(m)} = XW_Q^{(m)}$, $K^{(m)} = XW_K^{(m)}$ and $V^{(m)} = XW_V^{(m)}$, and

$$\mathrm{MHA}(X) = \mathrm{Concat}\Big(\mathrm{softmax}\Big(\frac{Q^{(m)}K^{(m)\top}}{\sqrt{d_h/M}}\Big)V^{(m)}\Big)_{m=1}^{M} W_O. \tag{5}$$

An attention block then applies MHA and a feed-forward layer (FF), each followed by a residual connection and layer normalization (LN):

$$\mathrm{AttBlock}(X) = \mathrm{LN}\big(Y + \mathrm{FF}(Y)\big), \qquad Y = \mathrm{LN}\big(X + \mathrm{MHA}(X)\big), \tag{6}$$

where FF consists of two linear layers with a ReLU activation and a hidden dimension of $4d_h$. We write $\mathrm{AttBlock}^L$ for a stack of L such blocks with distinct parameters. Each component below has its own blocks. Similarly, $\mathrm{MLP}(\cdot)$ denotes a two-layer perceptron with a ReLU activation, and $[\cdot \,\|\, \cdot]$ denotes concatenation.

4.2. Encoding of the Routing State

4.2.1. Customer Embeddings

Static and dynamic customers share the same projection of their static features so that a customer is represented in the same way before and after its insertion. Their time information is added through separate projections, since a delay before a planned visit and the age of a new request do not have the same meaning. A customer i planned in a remaining route is embedded as

$$h_i = \mathrm{LN}\Big(W_x[x_i,\, y_i,\, d_i]^\top + W_\eta\big(\eta_i(\tau) - \tau\big)\Big), \tag{7}$$

where $\eta_i(\tau) - \tau$ is the delay before its planned visit. The vehicle position $p_k(\tau)$ and the final depot are embedded with the same projection, using their coordinates, a zero demand and their own expected arrival delay. The new customer j is embedded as

$$h_j = \mathrm{LN}\Big(W_x[x_j,\, y_j,\, d_j]^\top + W_a\big(\tau - t_j^{\mathrm{rev}}\big)\Big), \tag{8}$$

where $\tau - t_j^{\mathrm{rev}}$ is the age of the request, i.e., the time elapsed since it arrived. This age is zero when, as in this paper, each request is processed upon revelation; the term allows deferred decisions without changing the architecture.

4.2.2. Route Encoding

For each vehicle k, the embeddings of its remaining route $\big(p_k(\tau), r_{k,1}, \ldots, r_{k,m_k}, 0\big)$ are stacked, and a sinusoidal encoding $\mathrm{PE}(r)$ of the rank r of each element is added, so that the order of the planned visits is represented explicitly. The resulting sequence is processed by L_R attention blocks, whose parameters are the same for all vehicles:

$$\big(\bar{h}_{k,0}, \bar{h}_{k,1}, \ldots, \bar{h}_{k,m_k+1}\big) = \mathrm{AttBlock}^{L_R}\Big(\big(h_{p_k} + \mathrm{PE}(0),\, h_{r_{k,1}} + \mathrm{PE}(1),\, \ldots,\, h_0 + \mathrm{PE}(m_k+1)\big)\Big). \tag{9}$$

Each contextualized token $\bar{h}_{k,r}$ thus carries information about the corresponding visit and about the rest of its route. Each route is encoded separately: routes do not exchange information at this step, and no causal mask is used, since all planned visits are known and already satisfy the constraints.

4.2.3. Vehicle State and Insertion Positions

The local state of vehicle k combines the contextualized token of its position with its remaining capacity $\hat{D}_k(\tau)$ and the distance $D_k^{\mathrm{done}}(\tau)$ it has already travelled:

$$u_k = \mathrm{MLP}_u\big([\bar{h}_{k,0} \,\|\, \hat{D}_k(\tau) \,\|\, D_k^{\mathrm{done}}(\tau)]\big). \tag{10}$$

Rather than representing an insertion position by a single node, each position $s \in \{0, \ldots, m_k\}$ of the remaining route is represented by the contextualized tokens of its predecessor and successor, together with the vehicle state:

$$g_{k,s} = \mathrm{MLP}_g\big([\bar{h}_{\mathrm{prev}(s)} \,\|\, \bar{h}_{\mathrm{next}(s)} \,\|\, u_k]\big). \tag{11}$$

4.2.4. Vehicle Representation

Each vehicle is summarized by an attention over its own insertion positions, using a query derived from its state:

$$v_k = \mathrm{MLP}_v\Big(\big[u_k \,\|\, \mathrm{MHA}\big(\mathrm{MLP}_q(u_k),\, G_k,\, G_k\big)\big]\Big), \tag{12}$$

where $G_k = (g_{k,0}, \ldots, g_{k,m_k})$ and $\mathrm{MHA}(q, G, G)$ denotes multi-head attention with a single query q over the keys and values G. Unlike a mean pooling of the route, this learned summary can give more weight to the parts of the route that matter for future insertions. Finally, the vehicles are compared with one another by L_V attention blocks over the fleet:

$$(\tilde{v}_1, \ldots, \tilde{v}_K) = \mathrm{AttBlock}^{L_V}\big((v_1, \ldots, v_K)\big). \tag{13}$$

No encoding of the vehicle index is added so that the representation is equivariant to a permutation of the vehicles. The new customer is not used anywhere in this encoder:

the representations $\tilde{v}_k$ and $g_{k,s}$ describe the routing state only, and the new customer is compared with them in the two decision stages.

4.3. Vehicle Selection

Let $\mathcal{K}_j(\tau)$ be the set of vehicles that can serve customer j, that is, those satisfying the capacity condition of Section 3.2. The embedding of the new customer and the representations of these vehicles form a sequence that is processed by L_D attention blocks:

$$\left(z_j,\ (\tilde{v}'_k)_{k\in\mathcal{K}_j(\tau)}\right) = \mathrm{AttBlock}^{L_D}\Big(\left(h_j,\ (\tilde{v}_k)_{k\in\mathcal{K}_j(\tau)}\right)\Big). \tag{14}$$

In this sequence, the vehicles become conditioned on the new customer and compare themselves with one another for this request, while the customer token z_j summarizes how j relates to the feasible fleet. The compatibility between the customer and each feasible vehicle is then scored as

$$u_{kj} = C\tanh\left(\frac{\left(W'_Q z_j\right)^\top \left(W'_K \tilde{v}'_k\right)}{\sqrt{d_h}}\right), \qquad k \in \mathcal{K}_j(\tau), \tag{15}$$

where the constant C bounds the scores to $[-C, C]$. This clipping prevents the probability of any feasible vehicle from vanishing early in training, so that alternative choices continue to be sampled and evaluated [11]. Vehicles that cannot serve customer j are masked, and the vehicle selection probability is obtained by a softmax over the scores of the feasible vehicles:

$$p_\theta(k \mid S_\tau, j) = \frac{\exp(u_{kj})}{\sum_{k'\in\mathcal{K}_j(\tau)} \exp(u_{k'j})}. \tag{16}$$

4.4. Insertion-Position Selection

Once a vehicle k has been selected, the second stage determines where the new customer j should be inserted within its remaining route. Let $\mathcal{S}_k$ denote the set of feasible insertion positions of vehicle k. Each position representation is first updated with the fleet-level representation of the vehicle, $g'_{k,s} = \mathrm{MLP}_{g'}\left([g_{k,s} \,\|\, \tilde{v}_k]\right)$. The new customer, the selected vehicle and its feasible positions then form a sequence processed by L_P attention blocks:

$$\left(z'_j,\ \cdot\ ,\ (\hat{g}_{k,s})_{s\in\mathcal{S}_k}\right) = \mathrm{AttBlock}^{L_P}\Big(\left(h_j,\ \tilde{v}_k,\ (g'_{k,s})_{s\in\mathcal{S}_k}\right)\Big). \tag{17}$$

The customer and vehicle tokens provide context only; scores are computed for the position tokens alone by comparing them with z'_j, the customer token contextualized by the selected vehicle and its feasible positions:

$$\ell_s = C\tanh\left(\frac{\left(W''_Q z'_j\right)^\top \left(W''_K \hat{g}_{k,s}\right)}{\sqrt{d_h}}\right), \qquad s \in \mathcal{S}_k, \tag{18}$$

with the same clipping constant C as in the first stage. The probability of inserting customer j at position s in the route of the selected vehicle k is obtained by a softmax over the scores of its feasible positions:

$$p_\theta(s \mid S_\tau, j, k) = \frac{\exp(\ell_s)}{\sum_{s'\in\mathcal{S}_k} \exp(\ell_{s'})}. \tag{19}$$

Single-Stage Comparison Policy

To assess the decomposition, we also consider a single-stage policy that shares the same encoder but defines one distribution over all feasible vehicle–position pairs. The

position tokens $g'_{k,s}$ of all vehicles are processed together with h_j by L_P attention blocks, and a single softmax is taken over all of them, with the same scoring function as in Equation (18). This policy can compare positions across vehicles directly, but its attention involves all the positions of the fleet, whereas the second stage of the two-stage policy attends only to the positions of the selected vehicle.

4.5. Training Procedure

Since the proposed policy acts on an already executing routing plan, each episode starts from the initial plan of the static customers and consists of the successive insertions of the dynamic customers. The training objective is the final cost of the episode.

4.5.1. Cost of a State and Insertion Costs

At any time, the cost of the routing state is the total distance it implies, that is, the distance already traveled by the vehicles plus the length of their remaining routes:

$$\mathcal{C}(S_\tau) = \sum_k \big(D_k^{\text{done}}(\tau) + L_k(\tau)\big) + \lambda_{\text{act}}\,\mathbf{1}_{\text{act}}, \tag{20}$$

where $L_k(\tau)$ is the length of the remaining route of vehicle k, including the end of its current leg. The last term accounts for the additional vehicle: when no vehicle of the fleet has enough capacity to serve a request, an additional vehicle can be activated once per episode, which can be seen as subcontracting the request; $\mathbf{1}_{\text{act}}$ indicates this activation and λ_{act} is its penalty. The cost of the t-th insertion is the variation of $\mathcal{C}$ caused by the decision, $\Delta_t = \mathcal{C}(S_t^+) - \mathcal{C}(S_t^-)$, which reduces to Equation (3) when the additional vehicle is not involved.

At the end of an episode ξ, the final cost $C(\xi)$ is the total distance traveled plus the activation penalty. If a request cannot be served even with the additional vehicle, the episode stops, and a failure penalty λ_{fail} and a penalty λ_{uns} per unserved customer are added. These penalties are designed to guide the policy towards good decisions. Inserting customers wherever it is immediately cheapest may exhaust the capacity of the vehicles and leave no room for later requests. Since such decisions would lead to the activation of the additional vehicle or to a failure, their cost would be reflected in the return, which should encourage the policy to preserve capacity for future requests. With C_0 the cost of the initial plan and n^{dyn} the number of dynamic customers, the insertion costs satisfy

$$\sum_{t=1}^{n^{\text{dyn}}} \Delta_t = C(\xi) - C_0, \tag{21}$$

where, in case of failure, the penalties are attached to the last decision. The return of an episode, $R(\xi) = -\sum_t \Delta_t$, is thus equal to $C_0 - C(\xi)$: maximizing it is equivalent to minimizing the final cost, whereas cheapest insertion minimizes each Δ_t separately.

4.5.2. Policy Gradient

The policy is trained using the REINFORCE estimator with a greedy rollout baseline as in [11]. At each training step, a batch of B instances is sampled. For each instance $e = 1, \ldots, B$, we call episode, denoted ξ_e, the complete simulation of this instance by the policy, from the initial plan of its static customers to the completion of the routes after all its dynamic customers have been inserted. The baseline b_e is the final cost obtained on the same instance, with the same initial plan and the same sequence of arrivals, by a frozen copy of the policy that selects its actions greedily. Future requests are revealed progressively to both policies, so that sharing the arrival sequence does not make them observable in advance. The advantage of the episode is

$$A_e = b_e - C(\xi_e) = R(\xi_e) - R^{\text{greedy}}(\xi_e), \tag{22}$$

where the second equality holds because both rollouts start from the same initial plan. The advantage is positive when the sampled episode is cheaper than the greedy rollout of the baseline policy. The same advantage is assigned to every insertion of the episode, so that each decision is credited for its effect on the final cost, including on the insertion of later customers. The advantages are normalized over the batch, and the training loss is

$$\mathcal{L}(\theta) = -\frac{1}{B}\sum_{e=1}^{B} A_e \sum_{t=1}^{n_e^{\mathrm{dyn}}} \log \pi_\theta\left(k_t, s_t \mid S_t, j_t\right), \tag{23}$$

where n_e^{dyn} is the number of dynamic customers of episode e and, by Equation (4), the log-probability of an insertion decomposes as

$$\log \pi_\theta(k, s \mid S_\tau, j) = \log p_\theta(k \mid S_\tau, j) + \log p_\theta(s \mid S_\tau, j, k), \tag{24}$$

so that both stages and the shared encoder are trained jointly from the same signal. No supervised pre-training on heuristic decisions is used, so that the policy is not driven towards imitating cheapest insertion.

The frozen baseline policy is periodically challenged: every U training steps, the current and frozen policies are evaluated greedily on fresh instances, and the frozen copy is replaced by the current policy only if the latter is significantly better according to a one-sided paired t-test. During training, the policy is also evaluated greedily on a fixed validation set, and the parameters with the lowest validation cost are retained. Algorithm 1 summarizes the training procedure.

Algorithm 1: REINFORCE Training for the Two-Stage DVRP Insertion Policy

Input : Distribution of DVRP instances $\mathcal{D}$; initial policy parameters θ; batch size B; challenge period U; significance level α_t; validation set $\mathcal{V}$.

Output: Policy parameters $\theta^\star$ with the lowest validation cost.

Initialize the frozen baseline policy $\theta^{\mathrm{BL}} \leftarrow \theta$
for *each training step* **do**
  Sample a batch of B instances from $\mathcal{D}$ and build the initial plan of their static customers
  foreach *instance* $e = 1, \ldots, B$ **do**
    `// Episode` ξ_e`: sampled insertions of all dynamic customers`
    while *dynamic customers remain* **do**
      Advance the vehicles to the next revelation time τ and reveal customer j
      Sample $k \sim p_\theta(k \mid S_\tau, j)$, then $s \sim p_\theta(s \mid S_\tau, j, k)$
      Apply the insertion (k, s) and record $\log \pi_\theta(k, s \mid S_\tau, j)$
    Complete the routes and compute the final cost $C(\xi_e)$
    Roll out θ^{BL} greedily on the same instance, initial plan and arrivals to obtain the cost b_e
    $A_e \leftarrow b_e - C(\xi_e)$
  Normalize the advantages A_e over the batch
  Compute $\mathcal{L}(\theta)$ (Equation (23)) and update θ with the gradient $\nabla_\theta \mathcal{L}(\theta)$
  if *the step is a multiple of* U **then**
    Evaluate θ and θ^{BL} greedily on fresh instances
    if *θ has a lower mean cost and the paired t-test gives* $p < \alpha_t$ **then**
      $\theta^{\mathrm{BL}} \leftarrow \theta$
    Evaluate θ greedily on $\mathcal{V}$; if its cost is the lowest so far, $\theta^\star \leftarrow \theta$

5. Computational Experiments

This section describes the generated instances, the methods compared, the training settings and the evaluation protocol.

5.1. Instances

Experiments are conducted on randomly generated DVRP instances. For each instance, the depot is located at the center of the unit square $[0,1]^2$, which is the same for all instances, and customer locations are sampled uniformly in this square. Customer demands are integers sampled uniformly between 1 and 9. A proportion of the customers, chosen at random, are static and known at time 0; the revelation times of the dynamic customers are sampled uniformly over the horizon $[0, H]$. Training, validation and test instances are generated from disjoint random streams.

All quantities are normalized when the instances are generated, and the policy uses them without further rescaling. Coordinates lie in the unit square, and distances are Euclidean distances in this square. Demands are divided by the vehicle capacity Q, so that each vehicle has a capacity of 1; the remaining capacities $\hat{D}_k(\tau)$ are thus expressed as fractions of the vehicle capacity. Since vehicles travel at unit speed, times are expressed in the same unit as distances: the horizon H, the revelation times, the delays $\eta_i(\tau) - \tau$, the age $\tau - t_j^{\text{rev}}$ of a request and the distance $D_k^{\text{done}}(\tau)$ traveled by each vehicle are all lengths of the unit square. The costs and penalties are expressed in the same unit.

To investigate the behavior of the proposed approach across different problem scales, we train three models on instances with 20, 50 and 100 customers. Their configurations are given in Table 2. In all three cases, 40% of the customers are static, which corresponds to a degree of dynamism $\delta = 0.6$.

The trained policies are then evaluated on the configurations of Table 3. For each configuration, the degree of dynamism is varied over $\delta \in \{0.1, 0.3, 0.5, 0.6, 0.75, 0.9\}$, the number of static customers being rounded down. Configurations whose size differs from the training size, as well as all values of δ other than 0.6, are generalization tests: the corresponding instances are not seen during training.

Table 2. Training configurations.

Parameter	Notation	Small	Medium	Large
Total customers	n	20	50	100
Degree of dynamism	δ	0.6	0.6	0.6
Number of vehicles	K	3	5	6
Vehicle capacity	Q	40	70	80
Time horizon	H	2.0	3.5	4.5
Maximum demand	$d_{\max}$	9	9	9
Batch size	B	512	512	256

The initial routing plan of the static customers is constructed using a parallel nearest-neighbor (NN) heuristic. This choice is deliberate: the objective of the present work is not to learn the initial route construction step but rather to study the online insertion of newly revealed customers into an already available routing plan. The same initial plan is used by all methods on a given instance.

Table 3. Test configurations. Each configuration is evaluated for $\delta \in \{0.1, 0.3, 0.5, 0.6, 0.75, 0.9\}$ on 1000 instances.

Trained Model	Customers n	Vehicles K	Capacity Q	Horizon H
Small ($n = 20$)	10	2	30	1.5
	20	3	40	2.0
	35	4	50	2.5
Medium ($n = 50$)	30	4	50	2.5
	50	5	70	3.5
	70	6	70	4.0
Large ($n = 100$)	80	6	70	4.0
	100	6	80	4.5
	120	8	90	5.0

5.2. Compared Methods

The proposed two-stage policy is compared with three insertion heuristics and with the single-stage policy of Section 4.4. All methods receive the same information at each decision, apply the same feasibility condition, and are subject to the same rules for the additional vehicle.

- **Cheapest insertion:** evaluates all feasible vehicle–position pairs and selects the one with minimum immediate insertion cost (Equation (3)).
- **Nearest insertion:** inserts the new customer next to the closest planned customer among feasible vehicles, at the cheaper of the two positions adjacent to it. If no feasible route contains a planned customer, the cheapest position is selected.
- **Random insertion:** selects a feasible vehicle–position pair uniformly at random. It indicates the cost of an insertion rule that uses no information on the routes.
- **Single-stage policy:** selects a vehicle–position pair from a single distribution over all feasible pairs. It shares the encoder of the proposed policy and is trained with the same procedure, on the same instances.

Cheapest insertion is used as the reference in all relative comparisons, since it is the standard insertion rule for dynamic requests and the strongest of the three heuristics.

5.3. Training Settings

The hyperparameters of the model and of the training procedure are summarized in Table 4. They are identical for the three models and for the single-stage policy. The two-stage and single-stage policies of a given size are trained with the same random seed and therefore on the same training instances. One training run is performed for each configuration.

Table 4. Hyperparameters of the model and of the training procedure.

Parameter	Notation	Value
Model		
Embedding dimension	d_h	128
Attention heads	M	8
Route encoder blocks	L_R	1
Fleet encoder blocks	L_V	1
Vehicle-selection blocks	L_D	3
Position-selection blocks	L_P	3
Clipping constant	C	10
Cost		
Activation penalty	λ_{act}	2
Failure penalty	λ_{fail}	10
Penalty per unserved customer	λ_{uns}	10

Table 4. *Cont.*

Parameter	Notation	Value
Training		
Training steps	–	150
Optimizer	–	Adam
Learning rate	–	3×10^{-4}
Gradient-norm clipping	–	10
Baseline challenge period	U	10 steps
Instances per baseline challenge	–	128
Significance level of the challenge	α_t	0.05
Validation period	–	10 steps
Validation instances	$\lvert\mathcal{V}\rvert$	100

Training and evaluation are performed on CPU, with four threads, on a workstation equipped with an Intel Core i7-14700KF processor (Intel Corporation, Santa Clara, CA, USA) and 32 GB of RAM.

5.4. Evaluation Protocol and Metrics

For each test configuration, 1000 instances are generated independently of the training and validation instances. All methods are evaluated on the same instances, with the same initial plans and the same arrival sequences, ensuring a strictly paired comparison. The learned policies are evaluated greedily by selecting the most probable vehicle and then the most probable position.

For a method m and an instance i, let C_i^m be the final cost of the episode (Section 4.5) and C_i^{CI} that of cheapest insertion. We report the following metrics:

- **Mean cost:** average final cost C_i^m over the instances;
- **Mean gap:** average relative difference $100\,(C_i^m - C_i^{\mathrm{CI}})/C_i^{\mathrm{CI}}$, in percent, where a negative gap indicates a lower cost than cheapest insertion;
- **Win, tie and loss rates:** percentages of instances for which the method obtains a lower, equal or higher final cost than cheapest insertion;
- **Confidence interval:** 95% confidence interval of the mean gap, obtained by a bootstrap with 2000 resamples;
- **Statistical significance:** p-values of a two-sided t-test and of a Wilcoxon signed-rank test on the per-instance relative gaps, testing whether the mean and the median gap differ from zero;
- **Decision time:** median time needed to take one insertion decision, including the computation of feasibility and of the policy inputs.

The heuristics are applied to one instance at a time. The learned policies process the pending decisions of all test instances in a single forward pass of the network, and their decision time is the duration of this pass divided by the number of instances. The training time of each model is also reported.

6. Results

The results are presented in the following order. We first examine the training of the learned policies (Section 6.1). We then compare all methods on the training configurations (Section 6.2), before studying how the comparison evolves with the degree of dynamism and the instance size (Section 6.3). Finally, we compare the two-stage and single-stage policies (Section 6.4) and report the computational times (Section 6.5). Tables A4–A6 in Appendix A extend the comparison of the two learned policies to all degrees of dynamism.

6.1. Training Behavior

Figure 2 shows the learning curves of the two learned policies for the three training configurations. For both policies, the validation cost decreases rapidly during the first 30 to 50 training steps, which corresponds to the acquisition of a first insertion rule that is far better than the initial random policy. The decrease then continues more slowly until the end of training. The cost of the sampled training episodes, shown by the thin lines, remains above the validation cost, since sampling keeps exploring alternative insertions whereas the validation policy selects its actions greedily.

The two policies learn at different speeds. For the medium and large models, the two-stage policy reaches a low validation cost faster: for $n = 100$, its validation cost is below 27 after 40 steps, whereas the single-stage policy needs about 70 steps to reach the same level. The single-stage policy, which must compare a larger number of candidate actions, improves more slowly at first but continues to progress, and it reaches a lower validation cost than the two-stage policy at the end of training for the three sizes. For the small model, the single-stage policy is ahead during most of the training.

For all three sizes, the validation cost is still slightly decreasing at step 150. All policies were trained with the same budget of 150 steps, and the results reported below therefore correspond to this fixed budget; longer training could further reduce their cost, particularly for the large instances, whose curves decrease the most at the end of training.

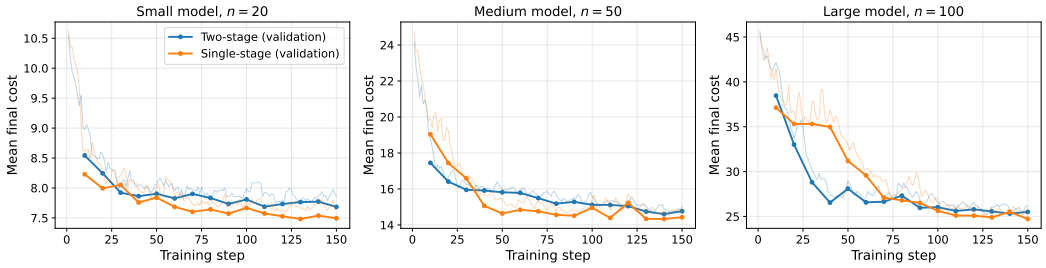


Figure 2. Learning curves of the two-stage and single-stage policies. Markers: mean final cost of the greedy policy on the validation set, evaluated every 10 steps. Thin lines: mean final cost of the sampled training episodes at each step.

Table 5 reports the training time of each model. The single-stage policy requires 1.3, 2.9 and 5.7 times more training time than the two-stage policy for the small, medium and large models, respectively. This ratio grows with the size of the instances, since the single-stage policy attends to all the insertion positions of the fleet at each decision. The training time of the large model is lower than that of the medium model because it was trained with batches of 256 instances instead of 512.

Table 5. Training time, step of the retained parameters and validation cost of the retained parameters.

	Two-Stage			Single-Stage		
Model	**Time (min)**	**Step**	**Val. Cost**	**Time (min)**	**Step**	**Val. Cost**
Small ($n = 20$)	24	150	7.684	31	130	7.483
Medium ($n = 50$)	111	140	14.602	319	140	14.333
Large ($n = 100$)	79	140	25.319	449	150	24.717

6.2. Solution Quality on the Training Configurations

Table 6 compares all methods on the configurations used for training, and Figure 3 shows the distribution of the per-instance gaps. Both learned policies obtain much lower costs than random insertion, whose mean gap ranges from 38% to 96%, and than nearest

insertion, whose mean gap ranges from 9% to 12%. The learned policies therefore extract information from the routing state that is relevant to the insertion decision, beyond the proximity used by nearest insertion.

Compared with cheapest insertion, the results depend on the size of the instances. For $n = 20$, the single-stage policy is on par with cheapest insertion, with a mean gap of -0.27% whose confidence interval contains zero and as many wins as losses, while the two-stage policy is 1.56% more expensive. For $n = 50$, the mean gaps are 2.21% and 4.23%, and for $n = 100$, they reach 5.36% and 7.79%. In all cases, the median gap is close to the mean gap so that these averages are not driven by a few extreme instances. The interquartile ranges show, however, a large variability across instances: for $n = 20$, the per-instance gap of the two-stage policy lies between -4.3% and 6.6% for half of the instances, and the policy is cheaper than cheapest insertion on 41.6% of them.

Table 6. Results on the training configurations (1000 test instances each). Gaps are relative to cheapest insertion, in percent: mean gap with its 95% bootstrap confidence interval, and median gap with its interquartile range. W/T/L: win, tie and loss rates against cheapest insertion (%). p-values of the two-sided t-test and of the Wilcoxon signed-rank test on the per-instance relative gaps.

Method	Mean Cost	Mean Gap [95% CI]	Median Gap [IQR]	W/T/L	p (t)	p (W)
Small model, $n = 20$, $\delta = 0.6$						
Cheapest	7.623	–	–	–	–	–
Nearest	8.286	9.07 [8.46, 9.67]	7.67 [1.69, 14.39]	12.3/6.2/81.5	$<10^{-3}$	$<10^{-3}$
Random	10.467	38.25 [37.23, 39.33]	36.21 [26.40, 48.24]	0.1/0.0/99.9	$<10^{-3}$	$<10^{-3}$
Single-stage	7.575	-0.27 [-0.74, 0.17]	-0.03 [-5.34, 4.33]	50.3/0.5/49.2	0.255	0.201
Two-stage	7.710	1.56 [1.03, 2.10]	1.71 [-4.28, 6.56]	41.6/0.2/58.2	$<10^{-3}$	$<10^{-3}$
Medium model, $n = 50$, $\delta = 0.6$						
Cheapest	14.149	–	–	–	–	–
Nearest	15.595	10.44 [9.93, 10.90]	9.75 [4.99, 15.52]	6.9/0.0/93.1	$<10^{-3}$	$<10^{-3}$
Random	24.491	73.72 [72.77, 74.68]	72.87 [63.49, 82.94]	0.0/0.0/100.0	$<10^{-3}$	$<10^{-3}$
Single-stage	14.433	2.21 [1.83, 2.60]	2.15 [-2.14, 6.53]	36.6/0.0/63.4	$<10^{-3}$	$<10^{-3}$
Two-stage	14.714	4.23 [3.81, 4.63]	4.02 [-0.09, 8.34]	25.4/0.0/74.6	$<10^{-3}$	$<10^{-3}$
Large model, $n = 100$, $\delta = 0.6$						
Cheapest	23.387	–	–	–	–	–
Nearest	26.170	12.05 [11.67, 12.44]	12.03 [7.62, 16.12]	1.5/0.0/98.5	$<10^{-3}$	$<10^{-3}$
Random	45.699	95.96 [95.00, 96.89]	94.69 [85.48, 105.39]	0.0/0.0/100.0	$<10^{-3}$	$<10^{-3}$
Single-stage	24.598	5.36 [5.01, 5.73]	5.19 [1.41, 9.26]	17.8/0.0/82.2	$<10^{-3}$	$<10^{-3}$
Two-stage	25.159	7.79 [7.38, 8.18]	7.45 [3.20, 12.16]	11.1/0.0/88.9	$<10^{-3}$	$<10^{-3}$

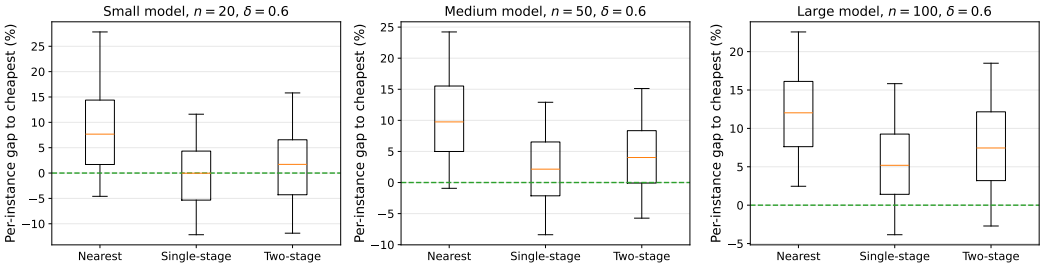


Figure 3. Distribution of the per-instance gap to cheapest insertion on the training configurations. Boxes: interquartile range and median; whiskers: 5th and 95th percentiles.

6.3. Effect of the Degree of Dynamism and of the Instance Size

Figure 4 shows the mean gap to cheapest insertion as a function of the degree of dynamism for all test configurations, and Figure 5 shows the corresponding win, tie and loss rates of the two-stage policy. Three observations can be made.

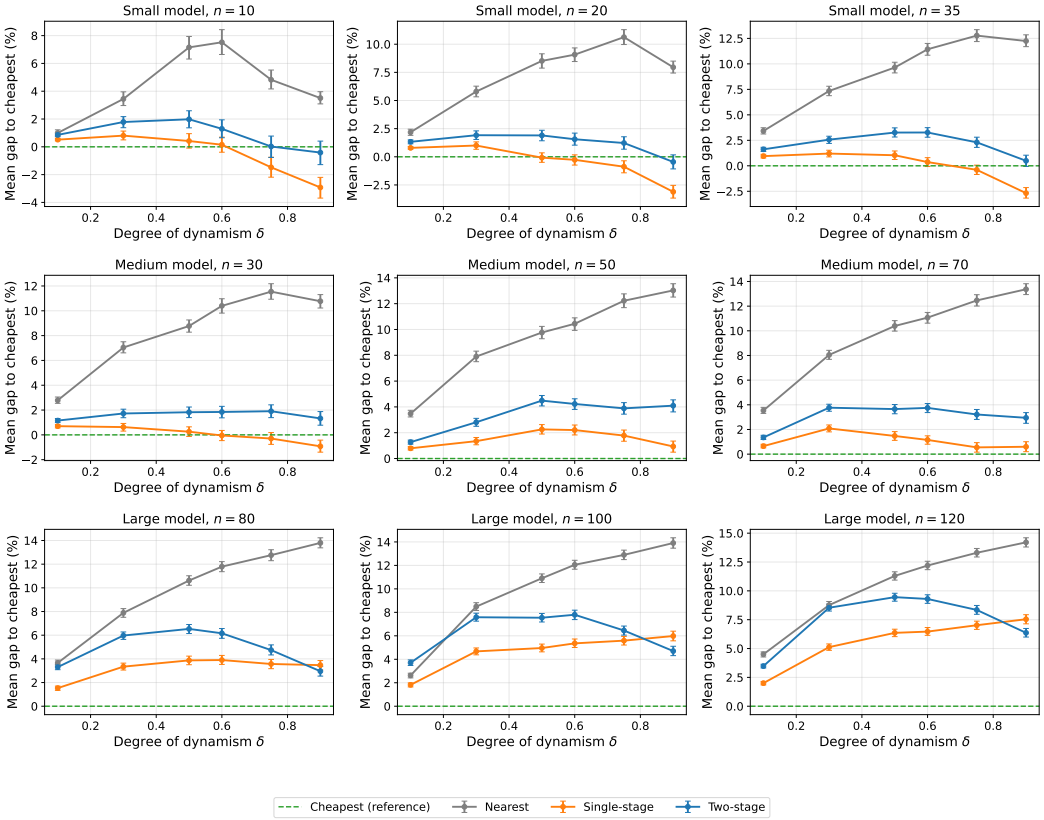


Figure 4. Mean gap to cheapest insertion as a function of the degree of dynamism δ, with 95% bootstrap confidence intervals (1000 instances per point). A negative gap indicates a lower cost than cheapest insertion. All models are trained with $\delta = 0.6$.

First, when the degree of dynamism is low ($\delta = 0.1$), all insertion methods obtain close costs. Only a few customers are inserted, and the final cost is mainly determined by the initial plan; for $n \leq 20$, a large share of the instances are ties.

Second, for the small model, the relative performance of the learned policies improves as the degree of dynamism increases, and they outperform cheapest insertion when most customers are dynamic. The two-stage policy reaches parity with cheapest insertion in the same situations: for $\delta = 0.9$ and $n \leq 35$, its mean gap lies between -0.5% and 0.5%, with a confidence interval containing zero, and it is cheaper than cheapest insertion on 48% to 56% of the instances, its highest win rates over all configurations. For $n = 20$, its median gap of -1.2% is significant according to the Wilcoxon test, but its mean gap of -0.45% does not differ significantly from zero. The single-stage policy is significantly cheaper than cheapest insertion for $\delta \geq 0.75$ with $n = 10$, 20 and 35, with mean gaps between -0.4% and -3.1%; the medium model shows the same behavior for $n = 30$. In these situations, few customers are known in advance, and each insertion has a large influence on the routes into which the following customers will be inserted. This is where a policy trained on the final cost of the episode can improve on a rule that minimizes the cost of each insertion separately.

Third, for larger instances, no configuration shows an advantage of the learned policies over cheapest insertion. For the medium model with $n = 50$ and $n = 70$, the mean gap ranges from 1.3% to 4.5% for the two-stage policy and from 0.6% to 2.3% for the single-stage policy, although the win rates increase with the degree of dynamism. The large model obtains higher gaps: for the two-stage policy, between 3.0% and 6.5% for $n = 80$, between 3.7% and 7.8% for $n = 100$, and between 3.5% and 9.5% for $n = 120$; for the single-stage policy, between 1.5% and 7.5% over the same instances. These higher gaps may stem from

the training budget, tighter for the large model (Figure 2). For the two-stage policy, the gap nevertheless decreases for the highest degrees of dynamism, as for the other models.

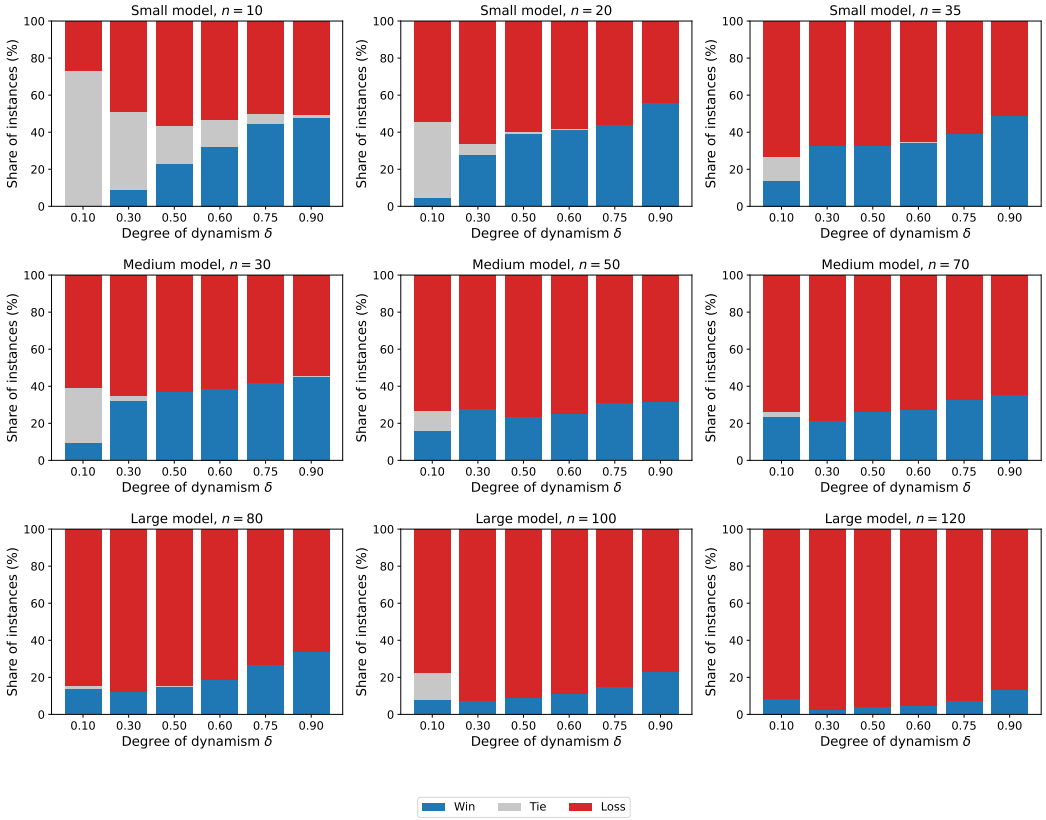


Figure 5. Win, tie and loss rates of the two-stage policy against cheapest insertion, as a function of the degree of dynamism δ (1000 instances per bar).

The models also generalize across instance sizes. The small model, trained with 20 customers, follows the same trends with 10 and 35 customers; the medium model, trained with 50 customers, follows with 30 and 70 customers; and the large model, trained with 100 customers, follows with 80 and 120 customers. Likewise, although all models are trained with $\delta = 0.6$, their behavior varies smoothly with the degree of dynamism, and the best results of the small model are obtained for $\delta = 0.9$, a value not seen during training. In all configurations except one, both learned policies remain better than nearest insertion; the exception is the two-stage policy for $n = 100$ and $\delta = 0.1$, where only 10 customers are dynamic.

6.4. Two-Stage and Single-Stage Policies

Table 7 compares the two learned policies directly. The single-stage policy obtains a lower cost than the two-stage policy in almost all configurations, with a mean gap between -0.3% and -3.1% relative to the two-stage policy, and wins on 42% to 69% of the instances for $\delta = 0.6$. The exceptions all concern the large model with $\delta = 0.9$, where the two-stage policy is cheaper for 80, 100 and 120 customers. Selecting a vehicle before seeing its insertion positions thus has a cost in solution quality: the second stage cannot correct a vehicle choice that turns out to offer only poor positions.

This advantage comes at a computational price, both at decision time and during training (Table 5).

Figure 6 summarizes the trade-off between the two learned policies over all configurations by plotting the extra cost of the two-stage policy relative to the single-stage policy against the speed-up of its decisions, defined as the ratio of the median decision times of

the single-stage and two-stage policies. The extra cost remains between 0.3% and 3.6% for all models, whereas the speed-up shifts to higher values as the instances grow: it ranges from 1.0 to 1.8 for the small model, from 1.4 to 2.7 for the medium model and from 1.7 to 2.7 for the large model. The decomposition therefore becomes more attractive as the number of candidate positions increases. For 80 to 120 customers, the two-stage policy decides about twice as fast for an extra cost of at most 3.3%, and for the highest degree of dynamism it is both faster and 0.3% to 1.0% cheaper than the single-stage policy. Its training is also 1.3, 2.9 and 5.7 times faster for the small, medium and large models (Table 5), which allows larger instances to be handled within the same training budget.

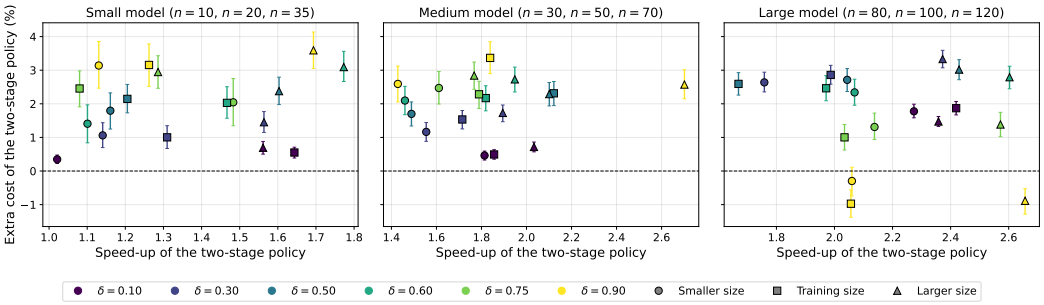


Figure 6. Extra cost of the two-stage policy relative to the single-stage policy, with its 95% bootstrap confidence interval, against the speed-up of its decisions (ratio of the median decision times of the single-stage and two-stage policies), for all test configurations (1000 instances per point). One panel per trained model; colors indicate the degree of dynamism and markers the test size relative to the training size.

Table 7. Single-stage policy against the two-stage policy on the same instances, for $\delta = 0.6$ (1000 instances per line). Gaps of the single-stage policy relative to the two-stage policy, in percent (negative: single-stage cheaper): mean gap with its 95% bootstrap confidence interval, and median gap with its interquartile range. W/T/L: win, tie and loss rates of the single-stage policy (%). p-values of the two-sided t-test and of the Wilcoxon signed-rank test on the per-instance relative gaps. Time ratio: median decision time of the single-stage policy divided by that of the two-stage policy.

Model	n	Mean Gap [95% CI]	Median Gap [IQR]	W/T/L	p (t)	p (W)	Time Ratio
Small	10	−0.64 [−1.16, −0.07]	0.00 [−4.14, 1.22]	42.3/26.8/30.9	0.022	$<10^{-3}$	1.10
Small	20	−1.46 [−1.92, −1.02]	−1.49 [−6.16, 2.63]	58.8/2.0/39.2	$<10^{-3}$	$<10^{-3}$	1.47
Small	35	−2.56 [−2.97, −2.17]	−2.56 [−7.14, 1.57]	66.5/0.0/33.5	$<10^{-3}$	$<10^{-3}$	1.77
Medium	30	−1.62 [−2.04, −1.21]	−1.63 [−5.70, 2.36]	61.8/0.0/38.2	$<10^{-3}$	$<10^{-3}$	1.46
Medium	50	−1.76 [−2.14, −1.40]	−1.75 [−5.90, 2.06]	62.7/0.0/37.3	$<10^{-3}$	$<10^{-3}$	1.82
Medium	70	−2.33 [−2.68, −1.99]	−2.26 [−5.99, 1.44]	67.4/0.0/32.6	$<10^{-3}$	$<10^{-3}$	1.95
Large	80	−1.93 [−2.29, −1.57]	−2.01 [−5.81, 2.03]	64.5/0.0/35.5	$<10^{-3}$	$<10^{-3}$	2.07
Large	100	−2.08 [−2.45, −1.72]	−2.61 [−6.09, 1.77]	66.4/0.0/33.6	$<10^{-3}$	$<10^{-3}$	1.97
Large	120	−2.44 [−2.76, −2.12]	−2.65 [−5.86, 1.04]	69.1/0.0/30.9	$<10^{-3}$	$<10^{-3}$	2.60

6.5. Computational Time

Figure 7 reports the decision times for $\delta = 0.6$. Except for $n = 100$, whose evaluation session was slower for all methods, cheapest insertion takes between 0.03 and 0.14 ms per decision, since the cost of each candidate position is obtained from three distances, and the two-stage policy takes between 0.2 and 1.0 ms. Across all sizes, the two-stage policy is 5 to 9 times slower than cheapest insertion, and the single-stage policy is slower still. All methods remain below 4 ms per decision up to 120 customers, which is compatible with real-time operation. The proposed policy does not, however, reduce the decision time compared with cheapest insertion. The configurations were evaluated in separate sessions on the same workstation, so that absolute times may vary between configurations, for instance, for $n = 100$; the ratios between methods within a configuration are more reliable.

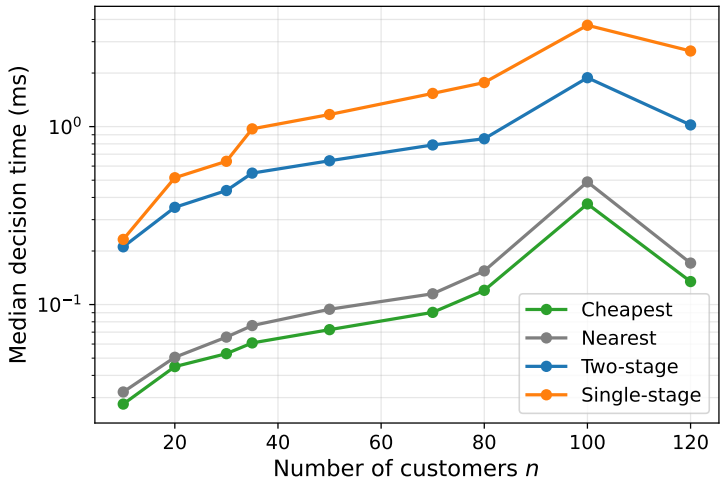


Figure 7. Median decision time as a function of the number of customers, for $\delta = 0.6$ (logarithmic scale). For each size, the model trained on the closest size is used. The time of the learned policies is amortized over the batch of test instances.

7. Discussion

7.1. When Does Learned Insertion Improve on Cheapest Insertion?

The results identify a regime in which the learned policies improve on cheapest insertion: instances in which most customers are dynamic, with up to about 35 customers. This regime corresponds to the situation in which the myopic nature of cheapest insertion is most penalizing. When few customers are known in advance, the initial routes are short, and the first dynamic customers largely determine the structure of the routes into which all subsequent customers will be inserted. A rule that places each customer at the position of least immediate cost does not take this structuring role into account. The proposed policy is trained on the final cost of the episode, and each insertion is credited for its effect on the subsequent ones; it can therefore learn insertion choices that are slightly more expensive at the time they are made but lead to cheaper routes overall. The improvement over cheapest insertion in this regime is consistent with this mechanism, as well as with the gains reported by Joe and Lau [7] over myopic policies for high degrees of dynamism. A decision-level analysis, measuring how often and by how much the learned policy deviates from the cheapest insertion and how these deviations affect the final cost, would be needed to establish it directly.

Conversely, when the degree of dynamism is low, the initial plan already fixes most of the routes, few decisions remain, and all insertion rules obtain similar costs. The evolution of the gaps with δ is therefore not simply the response of the system to a larger disturbance: at a fixed number of customers, increasing δ changes the role of the insertion decisions, from local adjustments of an existing plan to the construction of most of the routes.

The difference between instance sizes, visible in Figure 4, has several explanations. First, as the number of customers increases, routes become denser, and the immediate insertion cost becomes a better indicator of the quality of an insertion, which strengthens cheapest insertion. Second, a larger number of decisions per episode makes the learning problem harder: the same advantage is assigned to all the decisions of an episode, so that the contribution of each decision to the final cost is more difficult to identify when an episode contains 60 decisions rather than 12. Third, the training budget was the same for all sizes, and the learning curves show that the large model had not converged after 150 steps. Longer training, larger batches or a curriculum over instance sizes are natural ways to address this limitation.

The learned policies also generalize beyond their training configuration. The models trained with $\delta = 0.6$ behave consistently over the whole range of degrees of dynamism, and the best results of the small model are obtained for $\delta = 0.9$. They also retain their behavior on instances with a different number of customers and vehicles. A single model can therefore be used for a range of operating conditions without retraining.

7.2. Value and Cost of the Two-Stage Decomposition

The direct comparison with the single-stage policy shows that the decomposition has a cost in solution quality. In the two-stage policy, the vehicle is selected from a summary of its route, before the new customer is compared with its insertion positions. If the selected vehicle offers only poor positions, the second stage cannot correct this choice. The single-stage policy compares all positions of all vehicles jointly and avoids this error propagation, which explains its lower costs in almost all configurations.

The decomposition, however, reduces the computational effort. The attention of the single-stage policy involves all the insertion positions of the fleet, whose number grows with the total number of planned customers, whereas the second stage of the two-stage policy only considers the positions of the selected vehicle. This difference is reflected in decision times up to 2.6 times higher and in training times up to 5.7 times longer for the single-stage policy. For the large model with $\delta = 0.9$, the two-stage policy even obtains lower costs than the single-stage policy on 80, 100 and 120 customers, which may indicate that the joint policy becomes harder to train within a fixed budget as the number of candidate positions grows. The choice between the two architectures is therefore a trade-off: the joint policy is preferable when the fleet and the routes are small, while the decomposition becomes more attractive as they grow. Measuring separately the errors made in the vehicle-selection and position-selection stages would allow the vehicle-selection stage to be improved specifically, for instance, by providing it with a richer summary of the insertion opportunities of each vehicle.

7.3. Computational Considerations

The proposed policy does not reduce the decision time compared with cheapest insertion. For a distance-based objective with a capacity constraint, the cost of a candidate position is obtained from three distances and its feasibility from a single comparison, so that exhaustive evaluation is very fast. The decision times of the learned policies, below a few milliseconds, remain compatible with real-time operation. Their interest therefore lies in the quality of the decisions in the regimes identified above, not in computational savings. This balance may change for problems in which evaluating a candidate insertion is more expensive, for instance, with time windows, where the feasibility of an insertion depends on the arrival times at all subsequent customers.

7.4. Limitations

The conclusions of this study are subject to several limitations. The instances are generated with uniformly distributed customers and release times, Euclidean distances, deterministic travel times and a capacity constraint only; clustered demand, non-uniform arrival processes, time windows and stochastic travel times are not considered. Each configuration was trained once, so that the variability due to the training seed is not measured. The comparison is limited to insertion heuristics and to the single-stage learned policy: rolling-horizon re-optimization with a metaheuristic, which could provide an absolute reference for the quality of the solutions, and other learning-based approaches to the DVRP were not evaluated on the same instances. The established dynamic test sets [12] were not used, as they include service times and would require adapting the simulator. Finally, the contribution of the individual

components of the encoder, such as the learned route summary, the rank encoding or the attention across vehicles, was not isolated by ablation.

8. Conclusions

This paper introduced a two-stage attention-based insertion policy for the Dynamic Vehicle Routing Problem with online customer requests. Each newly revealed customer is inserted into the executing plan through two successive decisions: the selection of a vehicle, and the selection of an insertion position within its remaining route. The routing state is encoded by attention within each route and across the fleet, independently of the new request, and the same representations are used by both decision stages. The policy is trained with REINFORCE and a greedy rollout baseline on the final cost of each episode, so that each insertion is evaluated for its effect on the whole operating period rather than on its immediate cost.

The computational study, conducted on generated instances with up to 120 customers and degrees of dynamism between 0.1 and 0.9, leads to three main conclusions. First, the learned policies capture insertion-relevant information: they are much better than random and nearest insertion, and they generalize to degrees of dynamism and instance sizes not seen during training. Second, when most customers are revealed during operations on small and medium instances, which is the regime in which the myopic nature of cheapest insertion is most penalizing, the single-stage policy improves on cheapest insertion and the two-stage policy matches it statistically but with a higher win rate. Third, the decomposition into two stages reduces the decision and training times compared with a single-stage policy over all vehicle–position pairs, at the price of a lower solution quality in most configurations.

Several directions emerge from this work. A first direction is to strengthen the learning on large instances, through longer training, larger batches or a curriculum over instance sizes, and to measure the variability of the results over several training seeds. A second direction concerns the analysis of the policy: separating the errors of the vehicle-selection and position-selection stages, quantifying how the learned decisions deviate from cheapest insertion, and isolating the contribution of each component of the encoder by ablation. A third direction is to compare the policy with rolling-horizon re-optimization and to evaluate it on established dynamic test sets. Finally, the framework can be extended to richer problems, such as the Dynamic Vehicle Routing Problem with Time Windows, non-uniform arrival processes and stochastic travel times, in which insertion decisions depend on more than the immediate increase in distance and in which a learned policy may offer a clearer advantage over handcrafted insertion rules.

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

Funding: This research was supported by the MILEX Project (Mobilité Intelligente et Logistique Urbaine Durable pour une Meilleure Expérience des Usagers de la Ville), supported within the APRD framework by Fondation OCP and Université Mohammed VI Polytechnique (UM6P), and by the PHC Toubkal Programme, a Morocco–France mobility and collaboration programme administered by Campus France (project no. 50000NF). The APC was funded by the MILEX Project.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No external dataset was used in this study. All training, validation and test instances are generated from fixed random seeds with the instance generator described in Section 5.1, so that every dataset used in the experiments can be regenerated exactly. The generated

data and the code supporting the reported results are available from the corresponding author upon reasonable request.

Acknowledgments: During the preparation of this manuscript, the authors used Claude (Anthropic, Claude Opus 5.5, San Francisco, CA, USA) and ChatGPT (OpenAI, GPT-6 Astra, San Francisco, CA, USA) for the purposes of drafting and editing text. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

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

Appendix A. Complete Results

Tables A1–A3 report, for every test configuration and degree of dynamism, the results of the two learned policies against cheapest insertion, for the small, medium and large models. Tables A4–A6 compare the single-stage and two-stage policies in the same configurations. Figure A1 shows the win, tie and loss rates of the single-stage policy against cheapest insertion, to be compared with those of the two-stage policy in Figure 5.

Table A1. Complete results of the learned policies against cheapest insertion for the small model (1000 test instances per line). Gaps in percent; W/T/L: win, tie and loss rates (%); p-values of the t-test and of the Wilcoxon signed-rank test on the per-instance relative gaps.

δ	Policy	Mean Gap [95% CI]	Median Gap	W/T/L	p (t)	p (W)
$n = 10$ *customers*						
0.10	Two-stage	0.86 [0.73, 0.99]	0.00	0.0/73.2/26.8	$<10^{-3}$	$<10^{-3}$
0.10	Single-stage	0.52 [0.43, 0.60]	0.00	0.0/78.4/21.6	$<10^{-3}$	$<10^{-3}$
0.30	Two-stage	1.78 [1.37, 2.17]	0.00	8.7/42.5/48.8	$<10^{-3}$	$<10^{-3}$
0.30	Single-stage	0.80 [0.50, 1.13]	0.00	9.8/50.0/40.2	$<10^{-3}$	$<10^{-3}$
0.50	Two-stage	1.98 [1.37, 2.59]	0.47	22.8/20.6/56.6	$<10^{-3}$	$<10^{-3}$
0.50	Single-stage	0.42 [−0.09, 0.95]	0.00	25.4/27.5/47.1	0.108	$<10^{-3}$
0.60	Two-stage	1.29 [0.65, 1.94]	0.34	32.4/14.2/53.4	$<10^{-3}$	$<10^{-3}$
0.60	Single-stage	0.15 [−0.39, 0.70]	0.00	31.8/19.2/49.0	0.585	0.108
0.75	Two-stage	0.02 [−0.76, 0.77]	0.00	44.6/5.7/49.7	0.952	0.984
0.75	Single-stage	−1.48 [−2.19, −0.75]	0.00	47.6/8.2/44.2	$<10^{-3}$	$<10^{-3}$
0.90	Two-stage	−0.41 [−1.28, 0.41]	0.07	47.8/1.5/50.7	0.320	0.637
0.90	Single-stage	−2.92 [−3.69, −2.20]	−1.78	55.6/2.6/41.8	$<10^{-3}$	$<10^{-3}$
$n = 20$ *customers*						
0.10	Two-stage	1.33 [1.16, 1.50]	0.12	4.5/41.1/54.4	$<10^{-3}$	$<10^{-3}$
0.10	Single-stage	0.79 [0.67, 0.91]	0.00	3.9/46.0/50.1	$<10^{-3}$	$<10^{-3}$
0.30	Two-stage	1.92 [1.54, 2.30]	1.49	28.1/5.6/66.3	$<10^{-3}$	$<10^{-3}$
0.30	Single-stage	1.00 [0.67, 1.33]	1.01	27.0/8.6/64.4	$<10^{-3}$	$<10^{-3}$
0.50	Two-stage	1.90 [1.42, 2.36]	1.80	39.3/0.8/59.9	$<10^{-3}$	$<10^{-3}$
0.50	Single-stage	−0.08 [−0.49, 0.35]	0.26	44.9/1.9/53.2	0.719	0.750
0.60	Two-stage	1.56 [1.03, 2.10]	1.71	41.6/0.2/58.2	$<10^{-3}$	$<10^{-3}$
0.60	Single-stage	−0.27 [−0.74, 0.17]	−0.03	50.3/0.5/49.2	0.255	0.201
0.75	Two-stage	1.23 [0.68, 1.78]	1.29	44.2/0.1/55.7	$<10^{-3}$	$<10^{-3}$
0.75	Single-stage	−0.88 [−1.42, −0.35]	−1.03	54.5/0.1/45.4	0.002	$<10^{-3}$
0.90	Two-stage	−0.45 [−1.08, 0.17]	−1.17	55.8/0.0/44.2	0.157	0.005
0.90	Single-stage	−3.11 [−3.67, −2.54]	−3.51	63.8/0.1/36.1	$<10^{-3}$	$<10^{-3}$
$n = 35$ *customers*						
0.10	Two-stage	1.62 [1.41, 1.82]	1.22	14.1/13.0/72.9	$<10^{-3}$	$<10^{-3}$
0.10	Single-stage	0.95 [0.76, 1.13]	0.68	15.9/16.3/67.8	$<10^{-3}$	$<10^{-3}$
0.30	Two-stage	2.57 [2.20, 2.90]	2.49	32.6/0.1/67.3	$<10^{-3}$	$<10^{-3}$
0.30	Single-stage	1.20 [0.87, 1.53]	1.37	37.3/0.4/62.3	$<10^{-3}$	$<10^{-3}$
0.50	Two-stage	3.26 [2.82, 3.72]	3.00	32.9/0.0/67.1	$<10^{-3}$	$<10^{-3}$
0.50	Single-stage	1.04 [0.62, 1.46]	0.80	44.4/0.0/55.6	$<10^{-3}$	$<10^{-3}$
0.60	Two-stage	3.27 [2.80, 3.74]	2.80	34.7/0.0/65.3	$<10^{-3}$	$<10^{-3}$
0.60	Single-stage	0.37 [−0.07, 0.80]	0.06	49.8/0.0/50.2	0.111	0.450
0.75	Two-stage	2.30 [1.81, 2.81]	2.19	39.4/0.0/60.6	$<10^{-3}$	$<10^{-3}$
0.75	Single-stage	−0.40 [−0.85, 0.07]	−0.68	53.2/0.0/46.8	0.086	0.029
0.90	Two-stage	0.49 [−0.07, 1.04]	0.19	49.0/0.0/51.0	0.084	0.258
0.90	Single-stage	−2.68 [−3.17, −2.14]	−3.02	66.1/0.0/33.9	$<10^{-3}$	$<10^{-3}$

Table A2. Complete results of the learned policies against cheapest insertion for the medium model (1000 test instances per line). Gaps in percent; W/T/L: win, tie and loss rates (%); p-values of the t-test and of the Wilcoxon signed-rank test on the per-instance relative gaps.

δ	Policy	Mean Gap [95% CI]	Median Gap	W/T/L	p (t)	p (W)
$n = 30$ *customers*						
0.10	Two-stage	1.15 [0.99, 1.32]	0.45	9.7/29.5/60.8	$<10^{-3}$	$<10^{-3}$
0.10	Single-stage	0.70 [0.56, 0.84]	0.05	9.2/37.8/53.0	$<10^{-3}$	$<10^{-3}$
0.30	Two-stage	1.72 [1.38, 2.07]	1.71	32.3/2.6/65.1	$<10^{-3}$	$<10^{-3}$
0.30	Single-stage	0.63 [0.30, 0.92]	0.76	37.2/3.0/59.8	$<10^{-3}$	$<10^{-3}$
0.50	Two-stage	1.82 [1.39, 2.24]	1.95	37.3/0.0/62.7	$<10^{-3}$	$<10^{-3}$
0.50	Single-stage	0.26 [−0.13, 0.65]	0.43	47.0/0.7/52.3	0.187	0.180
0.60	Two-stage	1.85 [1.37, 2.30]	1.79	38.6/0.0/61.4	$<10^{-3}$	$<10^{-3}$
0.60	Single-stage	−0.05 [−0.46, 0.36]	0.01	50.0/0.0/50.0	0.820	0.615
0.75	Two-stage	1.91 [1.39, 2.42]	1.66	42.1/0.0/57.9	$<10^{-3}$	$<10^{-3}$
0.75	Single-stage	−0.30 [−0.77, 0.19]	0.05	49.4/0.1/50.5	0.227	0.234
0.90	Two-stage	1.33 [0.78, 1.88]	0.89	45.5/0.0/54.5	$<10^{-3}$	$<10^{-3}$
0.90	Single-stage	−0.92 [−1.39, −0.42]	−1.13	55.5/0.0/44.5	$<10^{-3}$	$<10^{-3}$
$n = 50$ *customers*						
0.10	Two-stage	1.27 [1.11, 1.43]	0.90	15.7/11.3/73.0	$<10^{-3}$	$<10^{-3}$
0.10	Single-stage	0.79 [0.66, 0.92]	0.51	16.7/14.1/69.2	$<10^{-3}$	$<10^{-3}$
0.30	Two-stage	2.81 [2.50, 3.11]	2.67	27.8/0.1/72.1	$<10^{-3}$	$<10^{-3}$
0.30	Single-stage	1.35 [1.06, 1.63]	1.49	37.5/0.1/62.4	$<10^{-3}$	$<10^{-3}$
0.50	Two-stage	4.49 [4.08, 4.88]	4.58	23.7/0.0/76.3	$<10^{-3}$	$<10^{-3}$
0.50	Single-stage	2.26 [1.90, 2.63]	2.12	34.3/0.0/65.7	$<10^{-3}$	$<10^{-3}$
0.60	Two-stage	4.23 [3.81, 4.63]	4.02	25.4/0.0/74.6	$<10^{-3}$	$<10^{-3}$
0.60	Single-stage	2.21 [1.83, 2.60]	2.15	36.6/0.0/63.4	$<10^{-3}$	$<10^{-3}$
0.75	Two-stage	3.89 [3.44, 4.33]	3.63	31.2/0.0/68.8	$<10^{-3}$	$<10^{-3}$
0.75	Single-stage	1.78 [1.34, 2.21]	1.62	42.0/0.0/58.0	$<10^{-3}$	$<10^{-3}$
0.90	Two-stage	4.10 [3.61, 4.55]	3.69	31.8/0.0/68.2	$<10^{-3}$	$<10^{-3}$
0.90	Single-stage	0.93 [0.49, 1.35]	0.70	46.2/0.0/53.8	$<10^{-3}$	$<10^{-3}$
$n = 70$ *customers*						
0.10	Two-stage	1.35 [1.19, 1.53]	1.13	23.3/3.2/73.5	$<10^{-3}$	$<10^{-3}$
0.10	Single-stage	0.65 [0.51, 0.80]	0.47	28.6/5.3/66.1	$<10^{-3}$	$<10^{-3}$
0.30	Two-stage	3.77 [3.48, 4.05]	3.59	21.4/0.0/78.6	$<10^{-3}$	$<10^{-3}$
0.30	Single-stage	2.08 [1.81, 2.37]	1.87	33.5/0.0/66.5	$<10^{-3}$	$<10^{-3}$
0.50	Two-stage	3.65 [3.27, 4.02]	3.59	26.3/0.0/73.7	$<10^{-3}$	$<10^{-3}$
0.50	Single-stage	1.47 [1.12, 1.83]	1.50	39.3/0.0/60.7	$<10^{-3}$	$<10^{-3}$
0.60	Two-stage	3.75 [3.37, 4.11]	3.52	27.5/0.0/72.5	$<10^{-3}$	$<10^{-3}$
0.60	Single-stage	1.15 [0.81, 1.49]	1.20	42.0/0.0/58.0	$<10^{-3}$	$<10^{-3}$
0.75	Two-stage	3.22 [2.81, 3.62]	2.84	32.8/0.0/67.2	$<10^{-3}$	$<10^{-3}$
0.75	Single-stage	0.55 [0.16, 0.93]	0.53	45.9/0.0/54.1	0.006	0.021
0.90	Two-stage	2.95 [2.50, 3.37]	2.64	35.3/0.0/64.7	$<10^{-3}$	$<10^{-3}$
0.90	Single-stage	0.60 [0.20, 1.01]	0.42	46.8/0.0/53.2	0.003	0.012

Table A3. Complete results of the learned policies against cheapest insertion for the large model (1000 test instances per line). Gaps in percent; W/T/L: win, tie and loss rates (%); p-values of the t-test and of the Wilcoxon signed-rank test on the per-instance relative gaps.

δ	Policy	Mean Gap [95% CI]	Median Gap	W/T/L	p (t)	p (W)
$n = 80$ *customers*						
0.10	Two-stage	3.31 [3.09, 3.53]	3.02	14.0/1.3/84.7	$<10^{-3}$	$<10^{-3}$
0.10	Single-stage	1.53 [1.34, 1.70]	1.22	24.2/2.4/73.4	$<10^{-3}$	$<10^{-3}$
0.30	Two-stage	5.97 [5.65, 6.27]	5.88	12.5/0.0/87.5	$<10^{-3}$	$<10^{-3}$
0.30	Single-stage	3.34 [3.06, 3.64]	2.99	24.9/0.0/75.1	$<10^{-3}$	$<10^{-3}$
0.50	Two-stage	6.53 [6.14, 6.90]	6.68	15.3/0.0/84.7	$<10^{-3}$	$<10^{-3}$
0.50	Single-stage	3.87 [3.51, 4.23]	3.86	25.1/0.0/74.9	$<10^{-3}$	$<10^{-3}$
0.60	Two-stage	6.16 [5.74, 6.56]	6.07	18.6/0.0/81.4	$<10^{-3}$	$<10^{-3}$
0.60	Single-stage	3.90 [3.53, 4.29]	3.82	26.7/0.0/73.3	$<10^{-3}$	$<10^{-3}$
0.75	Two-stage	4.74 [4.33, 5.18]	4.15	26.9/0.0/73.1	$<10^{-3}$	$<10^{-3}$
0.75	Single-stage	3.57 [3.17, 3.95]	3.34	28.6/0.0/71.4	$<10^{-3}$	$<10^{-3}$
0.90	Two-stage	2.96 [2.54, 3.36]	2.60	33.9/0.0/66.1	$<10^{-3}$	$<10^{-3}$
0.90	Single-stage	3.47 [3.07, 3.86]	3.46	29.2/0.0/70.8	$<10^{-3}$	$<10^{-3}$

Table A3. *Cont.*

δ	Policy	Mean Gap [95% CI]	Median Gap	W/T/L	p (t)	p (W)
$n = 100$ *customers*						
0.10	Two-stage	3.70 [3.47, 3.95]	3.35	8.0/14.8/77.2	$<10^{-3}$	$<10^{-3}$
0.10	Single-stage	1.82 [1.64, 1.99]	1.26	15.6/15.1/69.3	$<10^{-3}$	$<10^{-3}$
0.30	Two-stage	7.58 [7.25, 7.93]	7.42	7.5/0.0/92.5	$<10^{-3}$	$<10^{-3}$
0.30	Single-stage	4.67 [4.38, 4.97]	4.57	16.4/0.0/83.6	$<10^{-3}$	$<10^{-3}$
0.50	Two-stage	7.54 [7.18, 7.90]	7.16	9.2/0.0/90.8	$<10^{-3}$	$<10^{-3}$
0.50	Single-stage	4.96 [4.63, 5.29]	4.64	18.8/0.0/81.2	$<10^{-3}$	$<10^{-3}$
0.60	Two-stage	7.79 [7.38, 8.18]	7.45	11.1/0.0/88.9	$<10^{-3}$	$<10^{-3}$
0.60	Single-stage	5.36 [5.01, 5.73]	5.19	17.8/0.0/82.2	$<10^{-3}$	$<10^{-3}$
0.75	Two-stage	6.45 [6.06, 6.83]	6.33	15.2/0.0/84.8	$<10^{-3}$	$<10^{-3}$
0.75	Single-stage	5.59 [5.21, 5.95]	5.22	17.4/0.0/82.6	$<10^{-3}$	$<10^{-3}$
0.90	Two-stage	4.70 [4.31, 5.11]	4.52	23.2/0.0/76.8	$<10^{-3}$	$<10^{-3}$
0.90	Single-stage	5.98 [5.57, 6.40]	5.74	19.1/0.0/80.9	$<10^{-3}$	$<10^{-3}$
$n = 120$ *customers*						
0.10	Two-stage	3.48 [3.32, 3.64]	3.53	8.4/0.1/91.5	$<10^{-3}$	$<10^{-3}$
0.10	Single-stage	2.00 [1.85, 2.15]	1.95	18.3/0.0/81.7	$<10^{-3}$	$<10^{-3}$
0.30	Two-stage	8.54 [8.25, 8.85]	8.18	2.7/0.0/97.3	$<10^{-3}$	$<10^{-3}$
0.30	Single-stage	5.12 [4.85, 5.40]	4.88	11.9/0.0/88.1	$<10^{-3}$	$<10^{-3}$
0.50	Two-stage	9.45 [9.10, 9.79]	9.26	4.3/0.0/95.7	$<10^{-3}$	$<10^{-3}$
0.50	Single-stage	6.36 [6.03, 6.68]	6.07	10.1/0.0/89.9	$<10^{-3}$	$<10^{-3}$
0.60	Two-stage	9.30 [8.92, 9.66]	9.12	5.0/0.0/95.0	$<10^{-3}$	$<10^{-3}$
0.60	Single-stage	6.47 [6.15, 6.82]	6.31	11.3/0.0/88.7	$<10^{-3}$	$<10^{-3}$
0.75	Two-stage	8.35 [7.97, 8.73]	7.91	7.4/0.0/92.6	$<10^{-3}$	$<10^{-3}$
0.75	Single-stage	7.02 [6.66, 7.37]	6.72	11.8/0.0/88.2	$<10^{-3}$	$<10^{-3}$
0.90	Two-stage	6.36 [6.00, 6.74]	5.89	13.4/0.0/86.6	$<10^{-3}$	$<10^{-3}$
0.90	Single-stage	7.54 [7.14, 7.93]	7.05	11.3/0.0/88.7	$<10^{-3}$	$<10^{-3}$

Table A4. Single-stage policy against the two-stage policy for the small model, for all degrees of dynamism (1000 instances per line). Same columns as Table 7.

δ	Mean Gap [95% CI]	Median Gap [IQR]	W/T/L	p (t)	p (W)	Time Ratio
$n = 10$ *customers*						
0.10	−0.31 [−0.43, −0.20]	0.00 [0.00, 0.00]	12.4/81.8/5.8	$<10^{-3}$	$<10^{-3}$	1.02
0.30	−0.73 [−1.08, −0.36]	0.00 [−0.80, 0.00]	28.3/54.2/17.5	$<10^{-3}$	$<10^{-3}$	1.14
0.50	−1.11 [−1.59, −0.63]	0.00 [−3.19, 0.22]	40.2/33.3/26.5	$<10^{-3}$	$<10^{-3}$	1.16
0.60	−0.64 [−1.16, −0.07]	0.00 [−4.14, 1.22]	42.3/26.8/30.9	0.022	$<10^{-3}$	1.10
0.75	−0.95 [−1.57, −0.30]	0.00 [−4.85, 2.17]	48.1/16.4/35.5	0.003	$<10^{-3}$	1.48
0.90	−1.89 [−2.57, −1.21]	−1.42 [−6.86, 1.59]	56.4/11.4/32.2	$<10^{-3}$	$<10^{-3}$	1.13
$n = 20$ *customers*						
0.10	−0.49 [−0.64, −0.34]	0.00 [−0.16, 0.00]	26.2/57.0/16.8	$<10^{-3}$	$<10^{-3}$	1.64
0.30	−0.71 [−1.03, −0.37]	0.00 [−3.15, 1.38]	47.3/16.7/36.0	$<10^{-3}$	$<10^{-3}$	1.31
0.50	−1.65 [−2.04, −1.24]	−1.14 [−5.28, 2.19]	57.5/4.4/38.1	$<10^{-3}$	$<10^{-3}$	1.21
0.60	−1.46 [−1.92, −1.02]	−1.49 [−6.16, 2.63]	58.8/2.0/39.2	$<10^{-3}$	$<10^{-3}$	1.47
0.75	−1.72 [−2.22, −1.20]	−1.81 [−7.34, 3.39]	59.3/0.6/40.1	$<10^{-3}$	$<10^{-3}$	1.08
0.90	−2.19 [−2.74, −1.66]	−1.99 [−8.39, 3.31]	60.0/0.3/39.7	$<10^{-3}$	$<10^{-3}$	1.26
$n = 35$ *customers*						
0.10	−0.61 [−0.78, −0.43]	0.00 [−1.57, 0.14]	43.1/29.6/27.3	$<10^{-3}$	$<10^{-3}$	1.56
0.30	−1.18 [−1.48, −0.87]	−1.17 [−4.22, 1.88]	60.1/2.0/37.9	$<10^{-3}$	$<10^{-3}$	1.56
0.50	−1.93 [−2.30, −1.54]	−2.04 [−6.23, 2.09]	64.0/0.1/35.9	$<10^{-3}$	$<10^{-3}$	1.60
0.60	−2.56 [−2.97, −2.17]	−2.56 [−7.14, 1.57]	66.5/0.0/33.5	$<10^{-3}$	$<10^{-3}$	1.77
0.75	−2.31 [−2.78, −1.86]	−2.65 [−6.95, 1.93]	64.4/0.0/35.6	$<10^{-3}$	$<10^{-3}$	1.29
0.90	−2.79 [−3.27, −2.27]	−3.06 [−7.91, 2.39]	65.6/0.0/34.4	$<10^{-3}$	$<10^{-3}$	1.69

Table A5. Single-stage policy against the two-stage policy for the medium model, for all degrees of dynamism (1000 instances per line). Same columns as Table 7.

δ	Mean Gap [95% CI]	Median Gap [IQR]	W/T/L	p (t)	p (W)	Time Ratio
$n = 30$ *customers*						
0.10	−0.42 [−0.54, −0.29]	0.00 [−0.88, 0.00]	35.3/41.0/23.7	$<10^{-3}$	$<10^{-3}$	1.81
0.30	−0.96 [−1.24, −0.67]	−0.74 [−3.34, 1.11]	59.1/4.3/36.6	$<10^{-3}$	$<10^{-3}$	1.55
0.50	−1.34 [−1.71, −1.01]	−1.26 [−4.83, 1.73]	60.5/0.2/39.3	$<10^{-3}$	$<10^{-3}$	1.49
0.60	−1.62 [−2.04, −1.21]	−1.63 [−5.70, 2.36]	61.8/0.0/38.2	$<10^{-3}$	$<10^{-3}$	1.46
0.75	−1.85 [−2.31, −1.37]	−1.93 [−6.49, 2.70]	61.6/0.0/38.4	$<10^{-3}$	$<10^{-3}$	1.61
0.90	−1.87 [−2.40, −1.37]	−2.31 [−7.37, 3.32]	60.9/0.0/39.1	$<10^{-3}$	$<10^{-3}$	1.43

Table A5. *Cont.*

δ	Mean Gap [95% CI]	Median Gap [IQR]	W/T/L	p (t)	p (W)	Time Ratio
$n = 50$ *customers*						
0.10	−0.44 [−0.57, −0.30]	0.00 [−1.43, 0.33]	45.8/22.2/32.0	$<10^{-3}$	$<10^{-3}$	1.86
0.30	−1.33 [−1.59, −1.07]	−1.53 [−4.00, 1.40]	64.0/0.7/35.3	$<10^{-3}$	$<10^{-3}$	1.71
0.50	−1.95 [−2.29, −1.61]	−1.85 [−5.78, 1.46]	65.0/0.0/35.0	$<10^{-3}$	$<10^{-3}$	2.12
0.60	−1.76 [−2.14, −1.40]	−1.75 [−5.90, 2.06]	62.7/0.0/37.3	$<10^{-3}$	$<10^{-3}$	1.82
0.75	−1.81 [−2.19, −1.41]	−2.13 [−6.20, 2.05]	63.0/0.0/37.0	$<10^{-3}$	$<10^{-3}$	1.79
0.90	−2.74 [−3.19, −2.30]	−3.27 [−7.65, 1.92]	65.7/0.0/34.3	$<10^{-3}$	$<10^{-3}$	1.84
$n = 70$ *customers*						
0.10	−0.66 [−0.80, −0.53]	−0.29 [−1.73, 0.39]	57.3/8.6/34.1	$<10^{-3}$	$<10^{-3}$	2.03
0.30	−1.54 [−1.77, −1.29]	−1.66 [−4.23, 1.05]	64.9/0.0/35.1	$<10^{-3}$	$<10^{-3}$	1.89
0.50	−1.95 [−2.28, −1.64]	−1.91 [−5.72, 1.55]	64.3/0.0/35.7	$<10^{-3}$	$<10^{-3}$	2.10
0.60	−2.33 [−2.68, −1.99]	−2.26 [−5.99, 1.44]	67.4/0.0/32.6	$<10^{-3}$	$<10^{-3}$	1.95
0.75	−2.39 [−2.77, −2.03]	−2.47 [−6.42, 1.70]	67.0/0.0/33.0	$<10^{-3}$	$<10^{-3}$	1.77
0.90	−2.02 [−2.42, −1.58]	−2.13 [−6.56, 2.15]	62.8/0.0/37.2	$<10^{-3}$	$<10^{-3}$	2.70

Table A6. Single-stage policy against the two-stage policy for the large model, for all degrees of dynamism (1000 instances per line). Same columns as Table 7.

δ	Mean Gap [95% CI]	Median Gap [IQR]	W/T/L	p (t)	p (W)	Time Ratio
$n = 80$ *customers*						
0.10	−1.64 [−1.84, −1.46]	−1.47 [−3.39, 0.03]	69.7/4.9/25.4	$<10^{-3}$	$<10^{-3}$	2.27
0.30	−2.38 [−2.64, −2.12]	−2.41 [−5.42, 0.50]	69.4/0.0/30.6	$<10^{-3}$	$<10^{-3}$	1.76
0.50	−2.34 [−2.67, −2.00]	−2.46 [−6.00, 1.15]	67.4/0.0/32.6	$<10^{-3}$	$<10^{-3}$	2.04
0.60	−1.93 [−2.29, −1.57]	−2.01 [−5.81, 2.03]	64.5/0.0/35.5	$<10^{-3}$	$<10^{-3}$	2.07
0.75	−0.90 [−1.28, −0.51]	−1.10 [−5.31, 3.20]	57.2/0.0/42.8	$<10^{-3}$	$<10^{-3}$	2.14
0.90	0.74 [0.34, 1.15]	0.35 [−3.72, 4.90]	47.6/0.0/52.4	$<10^{-3}$	0.005	2.06
$n = 100$ *customers*						
0.10	−1.74 [−1.92, −1.57]	−0.94 [−3.39, 0.00]	62.6/16.0/21.4	$<10^{-3}$	$<10^{-3}$	2.42
0.30	−2.58 [−2.86, −2.32]	−2.60 [−5.37, 0.21]	73.6/0.0/26.4	$<10^{-3}$	$<10^{-3}$	1.99
0.50	−2.26 [−2.59, −1.97]	−2.47 [−5.69, 0.89]	67.8/0.0/32.2	$<10^{-3}$	$<10^{-3}$	1.67
0.60	−2.08 [−2.45, −1.72]	−2.61 [−6.09, 1.77]	66.4/0.0/33.6	$<10^{-3}$	$<10^{-3}$	1.97
0.75	−0.63 [−0.99, −0.26]	−0.91 [−4.76, 3.63]	55.5/0.0/44.5	$<10^{-3}$	$<10^{-3}$	2.03
0.90	1.42 [1.00, 1.82]	1.09 [−3.31, 6.20]	44.5/0.0/55.5	$<10^{-3}$	$<10^{-3}$	2.06
$n = 120$ *customers*						
0.10	−1.40 [−1.53, −1.27]	−1.37 [−2.81, 0.04]	73.6/0.5/25.9	$<10^{-3}$	$<10^{-3}$	2.36
0.30	−3.06 [−3.29, −2.80]	−3.07 [−5.80, −0.38]	77.0/0.0/23.0	$<10^{-3}$	$<10^{-3}$	2.37
0.50	−2.69 [−2.98, −2.39]	−2.91 [−5.89, 0.35]	72.5/0.0/27.5	$<10^{-3}$	$<10^{-3}$	2.43
0.60	−2.44 [−2.76, −2.12]	−2.65 [−5.86, 1.04]	69.1/0.0/30.9	$<10^{-3}$	$<10^{-3}$	2.60
0.75	−1.06 [−1.39, −0.72]	−1.23 [−4.96, 2.67]	58.4/0.0/41.6	$<10^{-3}$	$<10^{-3}$	2.57
0.90	1.29 [0.89, 1.67]	1.32 [−3.32, 5.80]	42.1/0.0/57.9	$<10^{-3}$	$<10^{-3}$	2.66

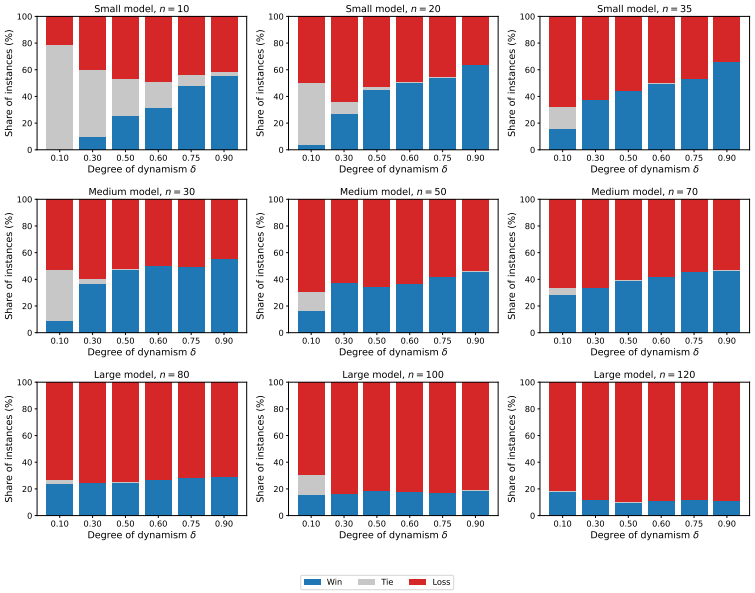


Figure A1. Win, tie and loss rates of the single-stage policy against cheapest insertion, as a function of the degree of dynamism δ (1000 instances per bar).

References

1. Wang, G.; Zou, Y.; Yang, Y.; Li, S. Dynamic Scheduling Based on Two-Layer Deep Reinforcement Learning for Multi-load AGVs. *Circuits Syst. Signal Process.* **2025**, *44*, 6445–6466. [CrossRef]
2. Pillac, V.; Gendreau, M.; Guéret, C.; Medaglia, A.L. A review of dynamic vehicle routing problems. *Eur. J. Oper. Res.* **2013**, *225*, 1–11. [CrossRef]
3. Zhang, J.; Van Woensel, T. Dynamic vehicle routing with random requests: A literature review. *Int. J. Prod. Econ.* **2023**, *256*, 108751. [CrossRef]
4. Solomon, M.M. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Oper. Res.* **1987**, *35*, 254–265. [CrossRef]
5. Ropke, S.; Pisinger, D. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transp. Sci.* **2006**, *40*, 455–472. [CrossRef]
6. Bent, R.W.; Van Hentenryck, P. Scenario-Based Planning for Partially Dynamic Vehicle Routing with Stochastic Customers. *Oper. Res.* **2004**, *52*, 977–987. [CrossRef]
7. Joe, W.; Lau, H.C. Deep Reinforcement Learning Approach to Solve Dynamic Vehicle Routing Problem with Stochastic Customers. *Proc. Int. Conf. Autom. Plan. Sched.* **2020**, *30*, 394–402. [CrossRef]
8. Konovalenko, A.; Hvattum, L.M. Optimizing a Dynamic Vehicle Routing Problem with Deep Reinforcement Learning: Analyzing State-Space Components. *Logistics* **2024**, *8*, 96. [CrossRef]
9. Akkerman, F.; Mes, M.; van Jaarsveld, W. A comparison of reinforcement learning policies for dynamic vehicle routing problems with stochastic customer requests. *Comput. Ind. Eng.* **2025**, *200*, 110747. [CrossRef]
10. Nazari, M.; Oroojlooy, A.; Takáč, M.; Snyder, L.V. Reinforcement Learning for Solving the Vehicle Routing Problem. In *Proceedings of the Advances in Neural Information Processing Systems*; Curran Associates Inc.: Red Hook, NY, USA, 2018; Volume 31.
11. Kool, W.; van Hoof, H.; Welling, M. Attention, Learn to Solve Routing Problems! In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
12. Montemanni, R.; Gambardella, L.M.; Rizzoli, A.E.; Donati, A.V. Ant Colony System for a Dynamic Vehicle Routing Problem. *J. Comb. Optim.* **2005**, *10*, 327–343. [CrossRef]
13. Vinyals, O.; Fortunato, M.; Jaitly, N. Pointer Networks. In *Proceedings of the Advances in Neural Information Processing Systems*; Curran Associates Inc.: Red Hook, NY, USA, 2015; Volume 28.
14. Li, J.; Ma, Y.; Gao, R.; Cao, Z.; Lim, A.; Song, W.; Zhang, J. Deep Reinforcement Learning for Solving the Heterogeneous Capacitated Vehicle Routing Problem. *IEEE Trans. Cybern.* **2022**, *52*, 13572–13585. [CrossRef] [PubMed]
15. Liu, Y.; Xie, B.; Long, Y.; Chen, J.; Xu, G. Learning-Based Heterogeneous Autonomous Vehicles Scheduling for On-Demand Last-Mile Transportation. *IEEE Trans. Autom. Sci. Eng.* **2025**, *22*, 21195–21211. [CrossRef]
16. Xiang, C.; Wu, Z.; Zhou, Y.; Tu, J. Dual-decoder attention model in hierarchical reinforcement framework for dynamic crowd logistics problem with batch-matching. *Transp. Res. Part C Emerg. Technol.* **2023**, *157*, 104417. [CrossRef]
17. Bono, G.; Dibangoye, J.S.; Simonin, O.; Matignon, L.; Pereyron, F. Solving Multi-Agent Routing Problems Using Deep Attention Mechanisms. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 7804–7813. [CrossRef]
18. Pan, W.; Liu, S.Q. Deep reinforcement learning for the dynamic and uncertain vehicle routing problem. *Appl. Intell.* **2023**, *53*, 405–422. [CrossRef]
19. Guan, Q.; Cao, H.; Niu, T.; Jia, L.; Yan, D.; Chen, B. Heterogeneous attention-driven deep reinforcement learning for solving EVRPs with pickup-delivery and time windows. *Appl. Energy* **2026**, *407*, 127355. [CrossRef]
20. Guan, Q.; Cao, H.; Zhuang, J.; Ou, B.; Chen, B. Efficient dynamic-aware deep reinforcement learning for solving capacitated electric vehicle routing problems with service time windows and time-of-use electricity prices. *Appl. Energy* **2026**, *423*, 128361. [CrossRef]
21. Hottung, A.; Tierney, K. Neural large neighborhood search for routing problems. *Artif. Intell.* **2022**, *313*, 103786. [CrossRef]
22. Luo, F.; Lin, X.; Zhong, M.; Liu, F.; Wang, Z.; Sun, J.; Zhang, Q. Learning to Insert for Constructive Neural Vehicle Routing Solver. In Proceedings of the Advances in Neural Information Processing Systems, San Diego, CA, USA, 2–7 December 2025; Volume 38.
23. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention Is All You Need. In *Proceedings of the Advances in Neural Information Processing Systems*; Curran Associates Inc.: Red Hook, NY, USA, 2017; Volume 30.

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