KEEP REHEARSING AND REFINING: LIFELONG LEARNING VEHICLE ROUTING UNDER CONTINUALLY DRIFTING TASKS

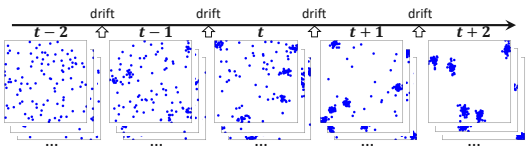
Anonymous authors
Paper under double-blind review

ABSTRACT

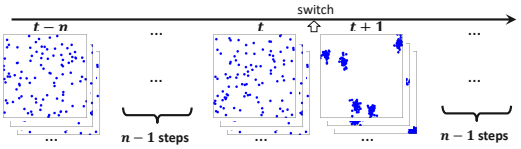
Existing neural solvers for vehicle routing problems (VRPs) are typically trained either in a one-off manner on a fixed set of pre-defined tasks or in a lifelong manner with tasks arriving sequentially, assuming sufficient training on each task. Both settings overlook a common real-world property: problem patterns may drift continually over time, yielding massive tasks sequentially arising, each with only limited training resources. In this paper, we propose a novel lifelong learning paradigm for neural VRP solvers under continual task drift over time, where each task is locally stationary at one learning time step but receives only insufficient training resources. We empirically demonstrate that such continual drift arises in practice using real-world logistics datasets. We then propose **D**ual **R**eplay with **E**xperience **E**nhancement (DREE), a general framework to improve learning efficiency and mitigate catastrophic forgetting under such drift. Extensive experiments based on real-world logistics datasets and commonly used synthetic datasets show that, under such continual drift, DREE effectively learns new tasks, preserves prior knowledge, improves generalization to unseen tasks, and can be applied to various existing neural solvers.

1 INTRODUCTION

Although neural solvers for vehicle routing problems (VRP) have achieved impressive performance by leveraging the strong capability of deep learning (Sun et al., 2024; Chen et al., 2025a; Huang et al., 2025), especially deep reinforcement learning (DRL), they are predominantly built upon a restrictive assumption: abundant training resources are available from each task (problem pattern) that is pre-defined and stationary. Training instances during learning are always independent and identically distributed, assuming patterns/tasks remain unchanged.



(a) Continually drifting tasks



(b) Periodically stationary tasks

Figure 1: Lifelong learning of VRP, with node distribution changes from uniform to clustered.

However, real-world scenarios often depart from this stationary regime (Markov et al., 2020; Liu et al., 2021; Wang et al., 2024c; Mardešić et al., 2024; Li et al., 2025b). New VRP instances arise sequentially over time, while their underlying patterns often drift continually (cf. Figure 1a). An example is daily urban delivery, where a service provider faces a stream of new VRP instances from daily orders. The scale and the underlying distribution of orders may change over time, as we observed (cf. Section 3.1) from the months of delivery records in real-world logistics datasets (Wu et al., 2024; Merchán et al., 2022) of cities with populations ranging from millions to tens of millions. Such changes can be driven by factors such as business growth, shifting consumer trends, and the emergence or relocation of customer clusters. Under such drift, one-off training becomes insufficient, as solvers trained only on historical tasks tend to suffer performance degradation when facing newly emerging patterns (Sun et al., 2018; Zhang et al., 2023). However, simply fine-tuning on each new

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

pattern may lead to catastrophic forgetting of previously learned knowledge (Wang et al., 2024b; Khetarpal et al., 2022). This gap motivates the methods that continually learn from continually drifting tasks while maintaining competence on previously encountered tasks.

Some recent studies (Li et al., 2025a; Feng et al., 2025; Pei et al., 2026) have begun exploring lifelong learning for neural VRP solvers, aiming to improve plasticity, i.e., the ability to learn new tasks, and stability, i.e., the ability to preserve knowledge of previously learned tasks. However, they mainly focus on the particular scenario considering periodically stationary tasks with sufficient per-task learning (denoted the *periodically stationary* scenario, cf. Figure 1b). Learning of each task remains stationary for a long time interval (hundreds of training steps with millions of instances) before the next task. Under this assumption, these methods typically rely on sufficient per-task training and on storing high-quality task-specific knowledge, such as model parameters (Li et al., 2025a; Feng et al., 2025) or behaviors (Pei et al., 2026) to mitigate forgetting. However, in real-world scenarios where the problem patterns drift continually, such assumptions no longer hold. Obtaining and storing high-quality per-task knowledge becomes impractical. As our experiments demonstrate (cf. Section 5.2), these methods can suffer substantial performance degradation.

In this work, we propose a novel paradigm and provide a formal formulation of lifelong learning for neural VRP solvers, where the task changes continually during learning, with sufficient per-task training not available (referred to as the *continually drifting* scenario). We propose **D**ual **R**eplay with **E**xperience **E**nhancement (DREE), a general and effective lifelong learning framework. DREE buffers encountered problem instances as well as the corresponding learned knowledge, like model behaviors, as experiences and replays them. In addition, DREE proactively enhances the quality of buffered experiences through interactions between the two replay mechanisms (*refining*), addressing the misleading issue from low-quality experiences induced by insufficient per-task training. We conduct extensive experiments on synthetic datasets, constructed from widely used instance-generation distributions, and real-world logistics datasets. Experiment results verify that DREE outperforms existing methods and has good applicability to different existing solvers.

Our main contributions are: (i) we introduce a new paradigm and formal formulation of lifelong learning under continual task drift for neural VRP solvers, motivated by observations of drift in real-world logistics data; (ii) we propose DREE, a general framework learning from this drift; and (iii) we validate DREE through extensive experiments on both synthetic continual-drift benchmarks and real-world datasets, supported by comprehensive assessment and analysis.

2 Related Work

2.1 Neural VRP Solvers

Neural VRP solvers learn from problem-solving experience, reducing human effort while achieving high performance (Bengio et al., 2021; Liu et al., 2023; Tang & Yao, 2024; Bogyrbayeva et al., 2024). They include construction and improvement methods (Bi et al., 2022; Kong et al., 2024). Construction solvers (e.g., (Huang et al., 2025; Kwon et al., 2020)) generate routes autoregressively, producing strong solutions within seconds. Improvement solvers iteratively refine complete solutions by configuring (Wu et al., 2022; Ma et al., 2023) or selecting predefined operators (Pei et al., 2025b; Guo et al., 2025). We focus on constructive neural solvers due to their broad applicability.

Existing methods typically require sufficient one-off training with millions of instances from predefined stationary tasks (Kwon et al., 2020; Fang et al., 2024). Their parameters are then frozen for inference. Generalization remains challenging (Gao et al., 2024; Zhou et al., 2025a): performance can be strong on instances i.i.d. with the training data but degrades sharply on unseen tasks (Liu et al., 2023; Jiang et al., 2022; 2023; Zhou et al., 2023). New architectures (e.g., (Fang et al., 2024; Wang et al., 2025; Xiao et al., 2025; Zheng et al., 2025)) and learning strategies (e.g., (Wang et al., 2024a; Liu et al., 2024a; Chen et al., 2025b)) improve generalization but still rely on sufficient one-off training on representative tasks, without actively adapting to emerging tasks.

2.2 Lifelong Learning Neural VRP Solvers

Fine-tuning, the most common approach, trains a pretrained solver on a new task (Zhou et al., 2023; Drakulic et al., 2025; Yu et al., 2026) but overwrites prior knowledge and causes forgetting on earlier

tasks (Wang et al., 2024b; Khetarpal et al., 2022). Stability on previous tasks and plasticity for new tasks are important when tasks arrive sequentially (Li et al., 2025a; Feng et al., 2025; Pei et al., 2026).

Buffering and replaying representative signals is widely used to address drift (Rosin & Belew, 1997; Isele & Cosgun, 2018; Buzzega et al., 2020). Existing lifelong learning solvers use experience replay to mitigate forgetting and transfer knowledge (Li et al., 2025a; Feng et al., 2025; Pei et al., 2026), with different experience representations. Feng et al. use problem instances, assuming controllable generation to revisit previous tasks. Learned parameters of sufficiently trained task-specific models also serve as knowledge (Li et al., 2025a). LLR-BC (Pei et al., 2026) stores high-quality solution trajectories as behavioral knowledge for later imitation.

Despite this progress, existing lifelong learning studies for neural VRP solvers assume periodically stationary tasks. Their methods rely on sufficient per-task training, and obtaining high-quality task-specific models (Li et al., 2025a; Feng et al., 2025) or behaviors (Pei et al., 2026) for subsequent knowledge consolidation. Issues arise in the continually drifting scenario. (i) A large number of tasks exist, each with highly limited training resources, making task-specific knowledge scarce and valuable. Using only instances or only behaviors as experience discards complementary information contained in the other. (ii) Without sufficient per-task training, the quality of buffered experience is substantially low. As a result, existing lifelong learning neural solvers are ineffective under the continually drifting scenario (cf. Section 5.2).

3 Lifelong Learning under Continually Drifting VRP Tasks

3.1 Continual Drift in VRP

Continual Drift Observed in the Real World Based on LaDe (Wu et al., 2024) and Amazon Last Mile Routing Merchán et al. (2022), real-world logistics datasets containing months of daily delivery records from multiple cities with populations ranging from millions to tens of millions, we empirically verify the existence of continual drift in real-world applications. Specifically, for a region or a delivery station, we quantify the spatial discrepancy between customers on any two days using the Gromov–Wasserstein (GW) distance (Mémoli, 2011). Then we draw the curve of the mean GW distance and the interval length between the two days to analyze the relation between them. With slight fluctuations, the GW distance curve shows an overall increasing trend as the temporal interval between two days becomes larger (Figure 2 shows an example). It reveals a clear continual drift trend in the spatial distribution of delivery customers. The problem scale also shows a continual drift trend. More details about the datasets, distance computation, and results are in Appendix C.

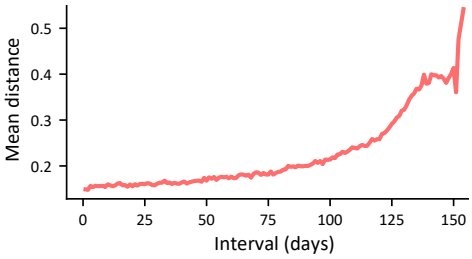


Figure 2: Mean GW distance increases with the interval length, city Jilin, region 31.

Continual Drift in the View of Tasks Following the common formulation of VRP tasks in neural solver studies, a VRP task P in this work is defined as a problem instance distribution in fixed settings, such as scale (Li et al., 2025a; Liu et al., 2024b), node-coordinate distribution (Jiang et al., 2023; Pei et al., 2026), constraints (Zhou et al., 2024; Pan et al., 2025; Luo et al., 2025), and the distance measurement (Feng et al., 2025). As observed, the VRP instance distribution would continually drift. Under a fine-grained time discretization, the task can be approximately stationary within a single step, yet two consecutive steps have related but different tasks. In this case, learning over a total of T time steps is conducted on a sequence of tasks $\{P_1, \ldots, P_T\}$, where $P_t \neq P_{t'}, \forall t \neq t'$. Training at each step on the corresponding task is inherently insufficient, since neural solvers typically require a large number of steps (epochs) to learn a single stationary task sufficiently (Li et al., 2025a; Feng et al., 2025; Kwon et al., 2020; Fang et al., 2024).

The periodically stationary scenario studied by existing works (Li et al., 2025a; Feng et al., 2025; Pei et al., 2026) assumes each task remains unchanged for n learning steps before the time step t that the next task arrives, i.e., $P_{t'} = P_t, \forall t' \in \{t-n, \ldots, t-1\}$, where $n \gg 1$ so that sufficient per-task training is available. Compared with it, the continually drifting scenario departs in that (i) no

sufficient per-task training is guaranteed, and (ii) task distributions drift continuously over time rather than remaining constant over long intervals.

While plasticity is essential for adapting to new tasks, stability is also important. A solver should preserve performance on earlier tasks, as instances of them may re-occur and newly encountered tasks could be near-variants of past ones. The overall goal is to learn a solver that can produce high-quality solutions for instances drawn from any seen task of any time step.

3.2 Problem Formulation

We formulate lifelong learning under continual drift as lifelong reinforcement learning. For an instance p, a constructive solver extends a partial solution in a stationary instance-conditioned Markov decision process (MDP) $\mathcal{M}(p) = \langle \mathcal{S}(p), \mathcal{A}(p), \mathcal{R}(p), \mathcal{T}(p) \rangle$. A state s in $S(p)$ encodes p and the partial solution, and an action a in $\mathcal{A}(p)$ selects the next node. The reward $\mathcal{R}(p)$ derives from the objective (e.g., negative tour length (Kwon et al., 2020)), and $\mathcal{T}(p)$ governs transitions under the constraints. At time step t, sampling $p \sim P_t$ jointly determines the MDP's state space, action space, rewards, and transitions. For fixed-scale tasks, instances can share a node-index action convention, but states, feasible-action masks, distance rewards, and capacity updates depend on p.

The task induces an MDP distribution $\mathbf{D}_t$, with $\mathcal{M}(p) \sim \mathbf{D}_t$. Its reward and transition marginals, $\mathbf{D}_t^{\mathcal{R}}$ and $\mathbf{D}_t^{\mathcal{T}}$, satisfy $\mathcal{R}(p) \sim \mathbf{D}_t^{\mathcal{R}}$ and $\mathcal{T}(p) \sim \mathbf{D}_t^{\mathcal{T}}$. Reward and transition functions are coupled through the sampled instance $p \sim P_t$ and remain fixed throughout each episode given p.

MDP distributions differ across time steps, i.e., $\mathbf{D}_t \neq \mathbf{D}_{t'}$, $\forall t \neq t'$. Changing distance measures (Feng et al., 2025) or objectives (Chen et al., 2025a) changes rewards, while changing constraints (Lin et al., 2024; Zhou et al., 2025b) modifies transitions and problem variants. We focus on scale and node-distribution drift, which changes the distribution of instance-conditioned MDPs, including their states, available actions, and distance-dependent rewards.

4 DREE

Experience replay preserves learned knowledge to mitigate forgetting and promote transfer (Li et al., 2025a; Feng et al., 2025; Pei et al., 2026), with its effectiveness depending on what is replayed and its quality. Existing methods use learned model parameters (Li et al., 2025a; Feng et al., 2025) or behaviors (Pei et al., 2026), typically assuming high quality when acquired. This assumption commonly fails under continual drift, where insufficient per-task training can produce low-quality knowledge that may misguide later learning.

We propose **DREE** (Figure 3), a general framework combining *problem instance replay* (PIR), *knowledge replay* (KR), and *experience enhancement* (EE). DREE buffers instances and learned knowledge, such as model parameters or behaviors, to preserve scarce per-task knowledge. PIR relearns buffered instances, while KR learns from previous solvers. We adopt LLR-BC's confidence-weighted behavior matching (Pei et al., 2026) for its performance advantage (Pei et al., 2026). EE updates an instance's buffered behaviors whenever PIR finds a better solution for it (Section 4.4), continuously improving replay targets that prior methods keep fixed (Pei et al., 2026; Isele & Cosgun, 2018; Buzzega et al., 2020). This replay–evaluate–refine process links learning on historical instances to knowledge preservation. DREE also supports model-based KR (Li et al., 2025a), which we implement and evaluate in Section 5.3. The following subsections describe its components, with further details and complexity analysis in Appendix A.

4.1 Experience Buffer

The base neural solver conducts the learning of an instance $p \sim P_t$ of a task P_t to improve the performance on the task, by minimizing a loss $\mathcal{L}_{\mathrm{DRL}}(\theta, p)$, where θ represents the solver model. The definition of $\mathcal{L}_{\mathrm{DRL}}(\theta, p)$ depends on the specific base solver used, and for different base solvers, the definition could be different. Experiences are generated while the solver learns the current task.

Each experience is defined as $e = \langle p, f, [(s, \pi)] \rangle$, where p denotes the problem instance, f is the objective value of a solution of p, and $[(s, \pi)]$ represents the corresponding solution construction trajectory, i.e., a sequence of states s and the solver's action distributions (behaviors) π on them.

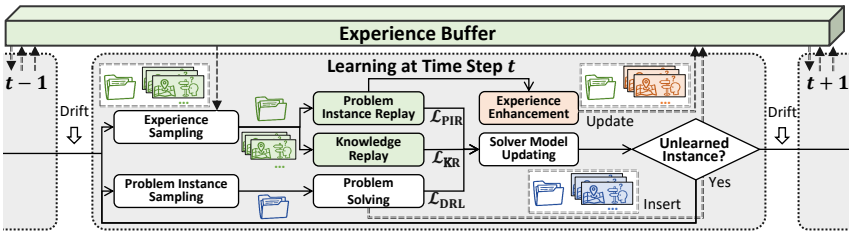


Figure 3: Lifelong learning of DREE with the task drifts between every two consecutive time steps.

DREE maintains a buffer with a fixed size $|\mathcal{B}|$ and inserts newly generated experiences using reservoir sampling (Vitter, 1985), so that each experience has an equal probability of being retained. Specifically, a new experience is inserted directly if the buffer is not full. Otherwise, it replaces a random buffered experience with probability $\frac{|\mathcal{B}|}{|E|}$, where $|E|$ denotes the number of experiences generated so far. DREE performs buffering and replay at the batch level.

4.2 Problem Instance Replay

A direct way to improve performance on previous tasks is to solve and learn instances of them again. Prior works typically implement such replay by generating new instances of previous tasks while learning a new task (Li et al., 2025a; Feng et al., 2025). In contrast, DREE reuses the buffered instances, thus is applicable when instance arriving is uncontrollable.

Specifically, for a buffered instance p_b, DREE re-solves it using the current solver to obtain a new trajectory $[(s_\theta, \pi_\theta)]$ and a new solution with a new objective value $f(\theta, p_b)$. The base solver then applies its standard training objective on this trajectory, yielding a problem instance replay (PIR) loss

$$\mathcal{L}_{\text{PIR}}(\theta, e) = \mathcal{L}_{\text{DRL}}(\theta, p_b),$$

where p_b is the buffered problem instance in e. DREE performs PIR with an adaptive interval N, allocating replay according to the observed experience-refinement yield. For a batch of M replayed instances, let M^+ be the number of instances where the new objective is better than the buffered one. DREE schedules the next PIR after learning $N \leftarrow \text{UB} - \frac{M^+}{M}(\text{UB} - \text{LB})$ batches of new-task instances, where LB and UB are hyperparameters. The ratio $\frac{M^+}{M}$ measures the observed refinement yield on the most recently replayed batch. A smaller ratio means fewer successful refinements, so a larger interval reduces computation spent seeking further improvements. KR continues to use the buffered behaviors at each learning iteration, while N remains bounded by LB and UB. $\mathcal{L}_{\text{PIR}}(\theta, e) \leftarrow 0$ when PIR is not performed.

4.3 Knowledge Replay

We take the model's learned behaviors as knowledge for replay due to its effectiveness and efficiency demonstrated by Pei et al.. For constructive solvers, the solution and its quality are fully determined by these behaviors. Thus, encouraging the solver to produce similar behaviors on the same (buffered) states is an effective and efficient way to preserve performance on previously learned tasks (Pei et al., 2026; Isele & Cosgun, 2018; Buzzega et al., 2020). To achieve this, the knowledge replay (KR) introduces an additional loss term, $\mathcal{L}_{\text{KR}}(\theta, e)$, which measures the discrepancy between the buffered behaviors and those produced by the current solver. By minimizing $\mathcal{L}_{\text{KR}}(\theta, e)$, DREE mitigates catastrophic forgetting when learning new tasks. Specifically, we adopt the confidence-aware weighting and reverse-KL behavior consolidation of LLR-BC (Pei et al., 2026) at the problem instance level, and define the knowledge replay loss based on a buffered experience e as

$$\mathcal{L}_{\text{KR}}(\theta, e) = \sum_{s_b \in e} \bar{w}_e(s_b) \sum_{a \in \mathcal{A}} \pi_\theta(a \mid s_b) \log \frac{\pi_\theta(a \mid s_b)}{\pi_b(a \mid s_b)},$$

where $\pi_\theta(\cdot \mid s_b)$ is the current solver's action distribution at a buffered state $s_b \in e$, $\pi_b(\cdot \mid s_b)$ is the buffered behavior stored in e corresponding to s_b, $\mathcal{A}$ is the corresponding action space, and $\bar{w}_e(s_b)$ is a confidence-based weight computed by CaEW (Pei et al., 2026).

Notably, other methods like model-based consolidation (Li et al., 2025a) can also be utilized as KR in DREE. Section 5.3 and Appendix F.3 provide further discussion and evaluation.

4.4 Experience Enhancement

Knowledge replay is effective when the target behaviors are of high quality, i.e., of solutions with good objective values. In lifelong learning of VRP solvers, buffering behaviors only after sufficiently long training on a stationary task can ensure high-quality experiences (Pei et al., 2026). However, in the continually drifting scenario, sufficient immediate per-task training is unavailable, leading to low-quality generated and buffered experiences. Since knowledge replay encourages imitation of buffered behaviors, low-quality targets may misguide learning and hurt lifelong performance.

PIR provides an opportunity to improve behavior quality after experiences have been buffered. It produces new solutions (and behaviors) whose objectives are directly comparable with the buffered ones. We thus introduce an *experience enhancement* mechanism. After performing PIR, DREE compares the new objective $f(\theta, p)$ with the buffered objective f_b for each replaced instance. Whenever $f(\theta, p)$ is better, i.e., $f(\theta, p) < f_b$, DREE updates the objective and the buffered state-behavior sequence with the newly generated ones, i.e., $f_b \leftarrow f(\theta, p)$ and $[(s_b, \pi_b)] \leftarrow [(s_\theta, \pi_\theta)]$. As a result, KR always consolidates knowledge toward the best-so-far solutions and corresponding behaviors, preserving the best performance on past tasks.

Overall, DREE optimizes the loss:

$$\mathcal{L}(\theta, p, e) = \mathcal{L}_{\mathrm{DRL}}(\theta, p) + \alpha\, \mathcal{L}_{\mathrm{KR}}(\theta, e) + \beta\, \mathcal{L}_{\mathrm{PIR}}(\theta, e),$$

where θ is the current solver, $p \sim P_t$ is a training instance of the current task P_t, and e is the experience sampled from the buffer. α and β are hyperparameters.

5 Experiments

We evaluate DREE under continual drift through five questions: (a) **Effectiveness:** Can DREE effectively and efficiently solve instances from learned tasks? (b) **Stability and plasticity:** How does DREE perform in terms of stability and plasticity? (c) **Generalization:** Can DREE generalize to unseen tasks across distributions and scales? (d) **Hyperparameter sensitivity:** How sensitive is DREE to its hyperparameters? (e) **Applicability:** Can DREE remain effective in other scenarios (e.g., periodically stationary tasks) and with other base neural solvers?

5.1 Experiment Setup

Datasets. We conduct experiments on both synthetic and real-world datasets. The synthetic dataset is constructed based on an existing one of the periodically stationary scenario (Pei et al., 2026). Specifically, for both TSP and CVRP, we consider five random orders over six principal tasks with different node distributions and scales. We take an epoch (a total of T epochs) as the unit of time steps (which is small enough, cf. Appendix D.1) and assign principal tasks evenly to the task sequence. The task of an epoch between two principal tasks is a linear mixture of them. Appendix D provides more details about principal tasks and the mixture method. In addition, TSPLIB (Reinelt, 1991) and CVRPLIB (Uchoa et al., 2017) are used for generalization ability evaluation. We further conduct an evaluation on two real-world datasets, LaDe (Wu et al., 2024) and Amazon Last Mile Routing (Merchán et al., 2022), taking each day as a time step, forming TSP and CVRP instances with all customers served each day in a given region, respectively. We involve five regions from LaDe and four from Amazon to form multiple lifelong learning sequences. The first 95% of days are used for lifelong learning, and the final 5% are left for generalization evaluation. More details are in Appendix C. For both synthetic and real-world datasets, each task (one epoch) has 4,096 training instances, significantly fewer than the millions of instances used in existing studies for one task (Kwon et al., 2020), following the insufficient per-task training assumption.

Training and Test Setting. We implement DREE[1] primarily with POMO (Kwon et al., 2020) for its concise design and broad applicability, while DREE also supports other neural solvers. We set $|\mathcal{B}| = 256$ and $\beta = 1$. $\alpha = 100$ for CVRP and 10 for TSP, as suggested in (Pei et al., 2026). The synthetic dataset has $T = 1000$ epochs, while in the real-world dataset T is determined by the valid days contained. DREE involves additional problem-solving episodes due to PIR. For a fair

[1]Code at https://anonymous.4open.science/r/Y8S75A0HNC

Table 1: Mean (std) over five task orders. MT-Ref is not lifelong learning and is trained once. Thus, ABPl/AFB/AMFB and std. are not applicable.

Method	CVRP				TSP			
	AP	AFB	AMFB	ABPl	AP	AFB	AMFB	ABPl
MT-Ref	**3.04**	-	-	-	**0.91**	-	-	-
Fine-tuning	8.75 (4.40)	5.47 (4.27)	15.75 (6.13)	3.28 (0.30)	1.97 (0.30)	0.86 (0.07)	1.64 (0.32)	1.11 (0.28)
Li	6.16 (0.38)	**0.09** (**0.06**)	**0.10** (**0.06**)	6.07 (0.38)	4.44 (0.96)	**0.02** (**0.01**)	**0.05** (**0.05**)	4.42 (0.95)
LLR-BC	4.19 (0.25)	1.04 (0.22)	1.68 (0.18)	3.16 (0.25)	1.38 (0.30)	0.34 (0.07)	0.65 (0.20)	1.05 (0.25)
DREE	**3.11** (**0.13**)	0.23 (0.15)	0.42 (0.34)	**2.88** (**0.12**)	**1.19** (**0.27**)	0.16 (0.05)	0.23 (**0.05**)	**1.03** (**0.25**)

comparison, we assign other lifelong learning methods additional batches to match the number of episodes. For evaluation, without loss of generality, we test the solver on the test instance set of each principal task, which provides an efficient and effective evaluation. We also conducted a finer-grained test in Appendix G.1. More settings are in Appendix D.2.

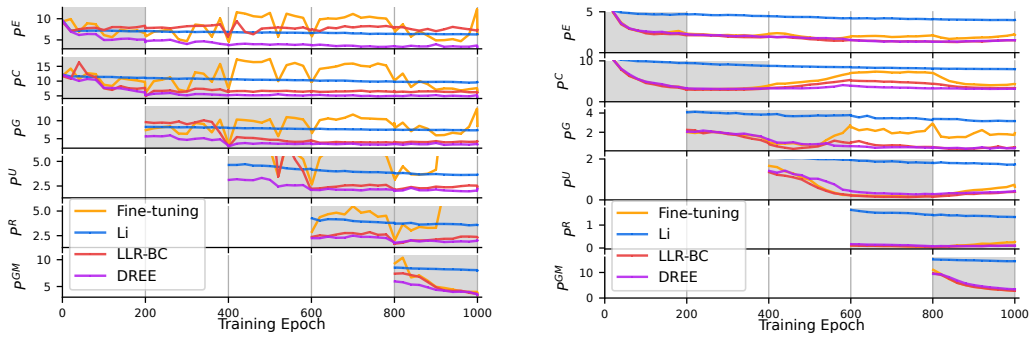
Comparison Methods. We include the commonly applied strategy in neural solver studies (Zhou et al., 2023; Yu et al., 2026), *fine-tuning*, which sequentially trains the solver only on the current task at each time step. We then include existing lifelong learning solvers, which are all designed for the periodically stationary scenario, in the continually drifting scenario, including (ii) the method of (Li et al., 2025a) (denoted *Li*) and (iii) LLR-BC (Pei et al., 2026). We also evaluate (iv) a multi-task setting, *MT-Ref*, which has access to all possible tasks and randomly selects tasks to train on in each batch. MT-Ref is an idealized offline reference with access to past and future tasks. We exclude other methods compared in (Pei et al., 2026) (e.g., methods of (Feng et al., 2025; Pei et al., 2025a)), as they store one solver per task and repeatedly compare against all stored solvers. This is prohibitively expensive in the continually drifting scenario, where massive stationary tasks are encountered over time. Appendix B presents more details about the compared methods and Appendix G provides additional experiment results of different settings of them.

Evaluation Metrics. We evaluate a lifelong learning solver with a comprehensive set of metrics, adapted from metrics designed for the periodically stationary scenario (Pei et al., 2026). Let $d_{t,i}$ denote the *test performance*, i.e., the average optimality gap (%) over all test instances, of the solver on principal task P_i after the tth training epoch. d_i^* denotes the best (smallest) test performance on a principal task P^i in one lifelong learning process, and t_i^* denotes the epoch that d_i^* is achieved (i.e., $d_i^* = d_{t_i^*,i}$). Based on this notation, after all T epochs of one lifelong learning process, we calculate (**smaller is better for all**): (a) **Average Performance (AP)**: the average performance on tasks learned, i.e., $AP = \frac{1}{K}\sum_{i=1}^{K} d_{T,i}$. (b) **Average Forgetting after Best (AFB)**: the performance decrease on previously learned tasks after the best performance of this task is achieved, i.e., $AFB = \frac{1}{K}\sum_{i=1}^{K}(d_{T,i} - d_i^*)$. (c) **Average Max Forgetting after Best (AMFB)**: the maximal forgetting of each learned task during the lifelong learning, i.e., $AMFB = \frac{1}{K}\sum_{i=1}^{K}\left(\max_{t_i^* \le t \le T} d_{t,i} - d_i^*\right)$. (d) **Average Best Plasticity (ABPl)**: the average of the best performance a solver achieves on each task, i.e., $ABPl = \frac{1}{K}\sum_{i=1}^{K} d_i^*$.

5.2 Performance on Synthetic Continual Drift

Performances on Seen Tasks. Table 1 demonstrates the metrics of each method. In terms of AP, DREE stably (smallest std.) outperforms (smaller mean) all compared lifelong learning methods over task orders on both TSP and CVRP. DREE even performs closely to *MT-Ref*, which can access all tasks simultaneously. It verifies DREE's effectiveness in solving tasks after lifelong learning of them.

Plasticity and Stability. DREE attains overall the smallest ABPl, indicating its strong plasticity. This suggests that the KR, IR, and EE mechanisms effectively transfer knowledge from previous tasks to facilitate learning on new tasks. *Fine-tuning* exhibits severe forgetting (large AFB and AMFB), as expected. *Li* strongly biases toward stability, yielding very small forgetting but substantially sacrificing plasticity (large AP and ABPl). In contrast, while maintaining promising plasticity, DREE achieves lower forgetting than all compared methods except *Li*, highlighting the effectiveness of the key modules of DREE (KR, IR, and EE) in mitigating forgetting. Figure 4 shows the test performance



(a) CVRP (b) TSP

Figure 4: Learning curve of task order 1, measured by test performance. The grey background indicates the epochs that the corresponding principal task is involved in generating intermediate tasks.

curve during lifelong learning. DREE effectively improves performance on new tasks while stably preserving its performance on previously learned tasks.

Performances on Unseen Benchmark Instances. Across five orders, DREE achieves the lowest gap among the compared lifelong learning methods on both CVRPLIB and TSPLIB (Table 14), and is comparable to or even better than MT-Ref. It suggests DREE's generalization ability for training a solver to solve unseen tasks effectively. Appendix F.2 provides more experimental details.

5.3 APPLICABILITY

To Different Base Solvers. To verify that the superior performance of DREE over other lifelong learning solvers does not stem from a bias toward a specific base solver (e.g., POMO), we conduct a comparison based on three other existing solvers, namely Omni (Zhou et al., 2023), INViT (Fang et al., 2024), and CoEKS (Yu et al., 2026). We implement and compare DREE with *fine-tuning* and LLR-BC on task order 1. Our goal is not to compare across base solvers but to evaluate different lifelong learning methods on the same base solver. The results across different base solvers are not directly comparable, as training settings are different (cf. Appendix F.3). As Table 2 shows, DREE outperforms *fine-tuning* and LLR-BC generally, with a similar pattern found on POMO. On CoEKS for CVRP, DREE achieves the lowest AP (3.39) and AFB (0.73), while LLR-BC has lower AMFB and fine-tuning has lower ABPl (Table 17). These results support DREE's applicability across the evaluated backbones, with different trade-offs in process-level metrics.

Table 2: Metric values with Omni or INViT as base solver.

Method	CVRP				TSP			
	AP	AFB	AMFB	ABPl	AP	AFB	AMFB	ABPl
Fine-tuning (INViT)	23.01	1.00	1.46	22.02	9.00	0.47	0.63	8.52
LLR-BC (INViT)	21.87	0.30	0.45	21.57	9.97	0.39	0.74	9.57
DREE (INViT)	**21.06**	**0.18**	**0.36**	**20.88**	**8.45**	**0.32**	**0.56**	**8.13**
Fine-tuning (Omni)	5.67	2.46	7.63	3.20	1.02	0.31	1.04	**0.71**
LLR-BC (Omni)	3.98	1.07	2.52	**2.91**	1.08	0.07	0.09	1.01
DREE (Omni)	**3.45**	**0.54**	**1.52**	**2.91**	**0.98**	**0.06**	**0.07**	0.93

To Different Knowledge Replay Methods. We further implement the idea of DREE in Li et al.'s method, which uses model buffering and distillation for knowledge consolidation, forming *DREE-Li*. On synthetic task order 1, DREE-Li reduces AP from 6.12 to 3.57 for CVRP and from 3.64 to 1.40 for TSP relative to Li, verifying DREE's effectiveness in performance improvement based on different knowledge replay methods. Appendix F.3 gives details of the implementation and results.

To Scenarios with Additional Task Change Types. We further conduct a comparison on (i) the periodically stationary scenario (Pei et al., 2026), and (ii) a variant of the continual drifting scenario with sudden changes, in which we randomly select three epoch interval lengths and swap their corresponding tasks, thereby introducing multiple abrupt changes into the task sequence (more details in Appendix G.4). Table 3 reports the results, based on task order 1. In both scenarios, DREE outperforms compared methods, with a similar pattern found in the continually drifting scenario. This demonstrates the robustness of DREE to more complex drift dynamics.

Table 3: Results on two additional scenarios.

Method	Periodically Stationary								Continual Drifting with Sudden Changes							
	CVRP				TSP				CVRP				TSP			
	AP	AFB	AMFB	ABPl	AP	AFB	AMFB	ABPl	AP	AFB	AMFB	ABPl	AP	AFB	AMFB	ABPl
Fine-tuning	6.67	4.34	6.44	2.34	2.40	1.68	4.13	0.73	6.52	3.36	8.67	3.16	1.55	0.81	0.93	0.74
LLR-BC	2.55	0.28	0.31	2.27	1.46	0.79	1.47	**0.67**	4.31	1.14	2.76	3.18	0.90	**0.11**	0.13	0.79
DREE	**2.30**	**0.04**	**0.09**	**2.26**	**0.80**	**0.10**	**0.16**	0.70	**3.19**	**0.30**	**0.51**	**2.89**	**0.81**	**0.11**	**0.11**	**0.70**

5.4 Ablation Study and Hyperparameter Analysis

To further analyze the proposed method, we conduct an ablation study and hyperparameter analysis, with Appendix F.4 and F.5 providing details of the settings and results. The ablation study verifies that removing PIR, KR, or EE leads to performance degradation across multiple metrics, confirming the contribution of each component. Varying the hyperparameter settings causes only minor fluctuations across metrics, suggesting that DREE is not overly sensitive to these settings.

5.5 Evaluation on Real-World Datasets

We further conduct lifelong learning on the real-world datasets, and Table 4 lists the overall results (more details in Appendices E and C.4, respectively) on TSP instances derived from LaDe and CVRP instances derived from the Amazon dataset. DREE still consistently outperforms the compared methods. In particular, DREE preserves the strong stability and plasticity, and also achieves the best performance (denoted *Gen*, where smaller is better) on unseen instances. Moreover, by reinforcing high-quality learned behaviors, DREE achieves better performance than MT-Ref. It demonstrates DREE's effectiveness in handling drift in real-world logistics scenarios.

Table 4: LaDe (TSP): mean (std.) over five regions. Amazon (CVRP): mean (population std.) over four stations. A dash denotes an unavailable or inapplicable result.

LaDe (TSP)					
Method	AP	AFB	AMFB	ABPl	Gen
MT-Ref	1.06 **(0.43)**	-	-	-	1.44 **(0.43)**
Fine-tuning	4.63 (3.79)	3.81 (3.18)	12.90 (13.88)	0.81 (0.65)	1.89 (1.53)
Li	10.01 (5.23)	4.47 (5.25)	8.31 (8.26)	5.55 (1.81)	12.64 (8.99)
LLR-BC	1.34 (1.29)	1.03 (1.03)	2.59 (1.97)	0.31 (0.26)	1.53 (1.14)
DREE	**1.03 (0.80)**	**0.73 (0.58)**	**2.06 (1.89)**	**0.30 (0.23)**	**1.32 (0.64)**
Amazon (CVRP)					
Method	AP	AFB	AMFB	ABPl	Gen
MT-Ref	2.08 (0.61)	–	–	–	2.24 (0.61)
Fine-tuning	2.25 (0.59)	1.08 (0.44)	3.42 (1.41)	**1.18** (0.36)	2.74 (0.88)
Li	4.16 (1.59)	0.65 (0.25)	1.39 (0.22)	3.51 (1.46)	4.84 (1.87)
LLR-BC	1.93 (0.41)	0.42 (0.13)	0.82 (0.10)	1.51 (0.45)	2.28 (0.51)
DREE	**1.77** (0.57)	**0.29** (0.13)	**0.73** (0.53)	1.49 (0.49)	**2.18** (0.56)

6 Conclusions

As observed in the real-world logistics datasets, new VRP instances commonly sequentially arrive with the problem pattern/distribution continually drifting. We propose lifelong learning for neural VRP solvers under this continual task drift, and formulate it as a lifelong RL problem with a drifting MDP distribution. A major challenge in this case is the low experience quality due to the frequent task change and insufficient per-task learning. To address it, we propose DREE, which integrates problem instance replay, knowledge replay, and experience enhancement to continually acquire, rehearse, and refine experience. Experiments on synthetic and real-world datasets verify that DREE can achieve superior performance to existing lifelong learning solvers under the continually drifting scenario, while remaining effective on more types of task drifting. This highlights that, in lifelong learning of VRP solvers, both problem instances and behaviors constitute critical experience, and enhancing the quality of buffered experience is crucial for effective learning.

Despite its strong results, DREE treats tasks of all time steps equally. A potential future direction is to selectively emphasize steps that accumulate significant drift and are most distinct from others for better efficiency. Incorporating strategies from dynamic optimization (Yang, 2015) or online learning (Zhang et al., 2023) also has potential for improvement. Extending DREE to scenarios with drifting constraints, so as to perform lifelong learning on diverse problem variants, also constitutes a promising future direction.

AI USE STATEMENT

We used GPT-5.6 Sol to assist with language editing and proofreading and to help check the correctness and consistency of mathematical expressions, pseudocode, and other formal descriptions in the manuscript. The authors reviewed and independently verified all adopted AI-assisted revisions and take full responsibility for the accuracy, originality, and integrity of the final manuscript.

REPRODUCIBILITY STATEMENT

We describe the method in Section 4 and Appendix A, with pseudocode in Algorithm 1. Parameter settings and training and test protocols are provided in Section 5.1 and Appendicesd D.2. For the base solvers, we follow the settings in their original papers (Kwon et al., 2020; Zhou et al., 2023; Fang et al., 2024; Yu et al., 2026) and use their open-source implementations (Section 5.1 and Appendix F.3). Data generation procedures and source references are given in Section 5.1 and Appendices D and C. Code and data are available at the anonymous link provided in Section 5.1.

REFERENCES

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

Jieyi Bi, Yining Ma, Jiahai Wang, Zhiguang Cao, Jinbiao Chen, Yuan Sun, and Yeow Meng Chee. Learning generalizable models for vehicle routing problems via knowledge distillation. In *NeurIPS*, volume 35, pp. 31226–31238, 2022.

Aigerim Bogyrbayeva, Meraryslan Meraliyev, Taukekhan Mustakhov, and Bissenbay Dauletbayev. Machine learning to solve vehicle routing problems: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 25(6):4754–4772, 2024.

Jakob Bossek, Pascal Kerschke, Aneta Neumann, Markus Wagner, Frank Neumann, and Heike Trautmann. Evolving diverse tsp instances by means of novel and creative mutation operators. In *ACM/SIGEVO Conference on Foundations of Genetic Algorithms*, pp. 58–71. Association for Computing Machinery, 2019.

Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: A strong, simple baseline. In *NeurIPS*, volume 33, pp. 15920–15930, 2020.

Jinbiao Chen, Zhiguang Cao, Jiahai Wang, Yaoxin Wu, Hanzhang Qin, Zizhen Zhang, and Yue-Jiao Gong. Rethinking neural multi-objective combinatorial optimization via neat weight embedding. In *ICLR*, 2025a.

Yuanyao Chen, Rongsheng Chen, Fu Luo, and Zhenkun Wang. Improving generalization of neural combinatorial optimization for vehicle routing problems via test-time projection learning. In *NeurIPS*, 2025b.

Darko Drakulic, Sofia Michel, and Jean-Marc Andreoli. GOAL: A generalist combinatorial optimization agent learner. In *ICLR*, 2025.

Han Fang, Zhihao Song, Paul Weng, and Yutong Ban. INViT: A generalizable routing problem solver with invariant nested view transformer. In *ICML*, 2024.

Shaodi Feng, Zhuoyi Lin, Jianan Zhou, Cong Zhang, Jingwen Li, Kuan-Wen Chen, Senthilnath Jayavelu, and Yew-Soon Ong. Lifelong learner: Discovering versatile neural solvers for vehicle routing problems. *arXiv preprint arXiv: 2508.11679*, 2025.

Chengrui Gao, Haopu Shang, Ke Xue, Dong Li, and Chao Qian. Towards generalizable neural solvers for vehicle routing problems via ensemble with transferrable local policy. In *International Joint Conference on Artificial Intelligence*, 2024.

Tong Guo, Yi Mei, Mengjie Zhang, Haoran Zhao, Kaiquan Cai, and Wenbo Du. Learning-aided neighborhood search for vehicle routing problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(7):5930–5944, 2025.

Ziwei Huang, Jianan Zhou, Zhiguang Cao, and Yixin Xu. Rethinking light decoder-based solvers for vehicle routing problems. In *ICLR*, 2025.

David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *AAAI*, volume 32, 2018.

Yuan Jiang, Yaoxin Wu, Zhiguang Cao, and Jie Zhang. Learning to solve routing problems via distributionally robust optimization. In *AAAI*, volume 36, pp. 9786–9794, 2022.

Yuan Jiang, Zhiguang Cao, Yaoxin Wu, Wen Song, and Jie Zhang. Ensemble-based deep reinforcement learning for vehicle routing problems under distribution shift. In *NeurIPS*, volume 36, pp. 53112–53125, 2023.

Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 75:1401–1476, 2022.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

Detian Kong, Yining Ma, Zhiguang Cao, Tianshu Yu, and Jianhua Xiao. Efficient neural collaborative search for pickup and delivery problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):11019–11034, 2024.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. In *NeurIPS*, volume 33, pp. 21188–21198, 2020.

Jingwen Li, Zhiguang Cao, Yaoxin Wu, and Tang Liu. Enhancing the cross-size generalization for solving vehicle routing problems via continual learning. *arXiv preprint arXiv:2510.10262*, 2025a.

Junru Li, Hongchang Li, and Xiaojun Zhao. Spatial–temporal evolution theory and influencing mechanisms of the express delivery network: A case on YTO express in China. *Transport Policy*, 171:408–424, 2025b.

Zhuoyi Lin, Yaoxin Wu, Bangjian Zhou, Zhiguang Cao, Wen Song, Yingqian Zhang, and Senthilnath Jayavelu. Cross-problem learning for solving vehicle routing problems. In *International Joint Conference on Artificial Intelligence*, pp. 6958 – 6966, 2024.

Fei Liu, Xi Lin, Weiduo Liao, Zhenkun Wang, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. Prompt learning for generalized vehicle routing. In *International Joint Conference on Artificial Intelligence*, pp. 6976–6984, 2024a.

Fei Liu, Xi Lin, Zhenkun Wang, Qingfu Zhang, Tong Xialiang, and Mingxuan Yuan. Multi-task learning for routing problem with cross-problem zero-shot generalization. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD, pp. 1898–1908, 2024b.

Jialin Liu, Ke Tang, and Xin Yao. Robust optimization in uncertain capacitated arc routing problems: Progresses and perspectives. *IEEE Computational Intelligence Magazine*, 16(1):63–82, 2021.

Shengcai Liu, Yu Zhang, Ke Tang, and Xin Yao. How good is neural combinatorial optimization? A systematic evaluation on the traveling salesman problem. *IEEE Computational Intelligence Magazine*, 18(3):14–28, 2023.

Fu Luo, Yaoxin Wu, Zhi Zheng, and Zhenkun Wang. Rethinking neural combinatorial optimization for vehicle routing problems with different constraint tightness degrees. In *NeurIPS*, 2025.

Yining Ma, Zhiguang Cao, and Yeow Meng Chee. Learning to search feasible and infeasible regions of routing problems with flexible neural k-opt. In *NeurIPS*, volume 36, pp. 49555–49578, 2023.

Nikola Mardešić, Tomislav Erdelić, Tonči Carić, and Marko Đurasević. Review of stochastic dynamic vehicle routing in the evolving urban logistics environment. *Mathematics*, 12(1), 2024.

Iliya Markov, Michel Bierlaire, Jean-François Cordeau, Yousef Maknoon, and Sacha Varone. Waste collection inventory routing with non-stationary stochastic demands. *Computers & Operations Research*, 113:104798, 2020.

Daniel Merchán, Jatin Arora, Julian Pachon, Karthik Konduri, Matthias Winkenbach, Steven Parks, and Joseph Noszek. 2021 Amazon Last Mile Routing Research Challenge: Data Set. *Transportation Science*, 58(1):8–11, 2022. doi: 10.1287/trsc.2022.1173.

Facundo Mémoli. Gromov–Wasserstein distances and the metric approach to object matching. *Foundations of Computational Mathematics*, 11(4):417–487, April 2011. ISSN 1615-3383.

Wenzheng Pan, Hao Xiong, Jiale Ma, Wentao Zhao, Yang Li, and Junchi Yan. UniCO: On unified combinatorial optimization via problem reduction to matrix-encoded general TSP. In *ICLR*, 2025.

Jiyuan Pei, Yi Mei, Jialin Liu, and Mengjie Zhang. LiBOG: Lifelong learning for black-box optimizer generation. In *International Joint Conference on Artificial Intelligence*, pp. 8912–8920, 2025a.

Jiyuan Pei, Yi Mei, Jialin Liu, Mengjie Zhang, and Xin Yao. Adaptive operator selection for meta-heuristics: A survey. *IEEE Transactions on Artificial Intelligence*, 6(8):1991–2012, 2025b.

Jiyuan Pei, Yi Mei, Jialin Liu, Mengjie Zhang, and Xin Yao. Lifelong learning with behavior consolidation for vehicle routing. In *ICLR*, 2026.

Gerhard Reinelt. TSPLIB—A traveling salesman problem library. *ORSA Journal on Computing*, 3 (4):376–384, 1991.

Christopher D. Rosin and Richard K. Belew. New methods for competitive coevolution. *Evolutionary Computation*, 5(1):1–29, 1997.

Rui Sun, Zhi Zheng, and Zhenkun Wang. Learning encodings for constructive neural combinatorial optimization needs to regret. In *AAAI*, volume 38, pp. 20803–20811, 2024.

Yu Sun, Ke Tang, Zexuan Zhu, and Xin Yao. Concept drift adaptation by exploiting historical knowledge. *IEEE Transactions on Neural Networks and Learning Systems*, 29(10):4822–4832, 2018.

Ke Tang and Xin Yao. Learn to optimize – A brief overview. *National Science Review*, 11(8): nwae132, 2024.

Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

Jeffrey S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, 1985.

Chenguang Wang, Zhouliang Yu, Stephen McAleer, Tianshu Yu, and Yaodong Yang. ASP: Learn a universal neural solver! *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(6): 4102–4114, 2024a.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(8):5362–5383, 2024b.

Shaolin Wang, Yi Mei, and Mengjie Zhang. Explaining genetic programming-evolved routing policies for uncertain capacitated arc routing problems. *IEEE Transactions on Evolutionary Computation*, 28(4):918–932, 2024c.

Yang Wang, Ya-Hui Jia, Wei-Neng Chen, and Yi Mei. Distance-aware attention reshaping for enhancing generalization of neural solvers. *IEEE Transactions on Neural Networks and Learning Systems*, 36(10):18900–18914, 2025.

Lixia Wu, Haomin Wen, Haoyuan Hu, Xiaowei Mao, Yutong Xia, Ergang Shan, Jianbin Zheng, Junhong Lou, Yuxuan Liang, Liuqing Yang, Roger Zimmermann, Youfang Lin, and Huaiyu Wan. LaDe: The first comprehensive last-mile express dataset from industry. In *SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5991–6002. ACM, 2024.

Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems. *IEEE Transactions on Neural Networks and Learning Systems*, 33 (9):5057–5069, 2022.

Yubin Xiao, Di Wang, Xuan Wu, Yuesong Wu, Boyang Li, Wei Du, Liupu Wang, and You Zhou. Improving generalization of neural vehicle routing problem solvers through the lens of model architecture. *Neural Networks*, 187:107380, 2025.

Shengxiang Yang. Evolutionary computation for dynamic optimization problems. In *Annual Conference on Genetic and Evolutionary Computation*, pp. 629–649. Association for Computing Machinery, 2015.

Zikang Yu, Jinbiao Chen, and Jiahai Wang. Combination-of-experts with knowledge sharing for cross-task vehicle routing problems. In *ICLR*, 2026.

Shuyi Zhang, Peter Tino, and Xin Yao. Hierarchical reduced-space drift detection framework for multivariate supervised data streams. *IEEE Transactions on Knowledge and Data Engineering*, 35 (3):2628–2640, 2023.

Yuepeng Zheng, Fu Luo, Zhenkun Wang, Yaoxin Wu, and Yu Zhou. MTL-KD: Multi-task learning via knowledge distillation for generalizable neural vehicle routing solver. In *NeurIPS*, 2025.

Changliang Zhou, Xi Lin, Zhenkun Wang, and Qingfu Zhang. Learning to reduce search space for generalizable neural routing solver. *arXiv preprint arXiv: 2503.03137*, 2025a.

Changliang Zhou, Canhong Yu, Shunyu Yao, Xi Lin, Zhenkun Wang, Yu Zhou, and Qingfu Zhang. URS: A unified neural routing solver for cross-problem zero-shot generalization. *arXiv preprint arXiv: 2509.23413*, 2025b.

Jianan Zhou, Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Towards omni-generalizable neural methods for vehicle routing problems. In *ICML*, volume 202, pp. 42769–42789. PMLR, 2023.

Jianan Zhou, Zhiguang Cao, Yaoxin Wu, Wen Song, Yining Ma, Jie Zhang, and Xu Chi. MVMoE: Multi-task vehicle routing solver with mixture-of-experts. In *ICML*, volume 235, pp. 61804–61824. PMLR, 2024.

A Details of DREE

A.1 General Process

Algorithm 1 demonstrates the process of DREE. For each learning time step (epoch), following common practice (Kwon et al., 2020; Zhou et al., 2023; Pei et al., 2026), DREE iteratively learns in units of batches. In each iteration, a batch of new instances of the current task is sampled to obtain $\mathcal{L}_{\mathrm{DRL}}$ to learn the current task. One experience batch is sampled, on which $\mathcal{L}_{\mathrm{KR}}$ is obtained to learn from buffered behavior. PIR is conditionally conducted based on N, to better balance between plasticity and stability. To calculate each loss term (i.e., $\mathcal{L}_{\mathrm{DRL}}$, $\mathcal{L}_{\mathrm{KR}}$, or $\mathcal{L}_{\mathrm{PIR}}$) based on an instance/experience batch, DREE takes the average value over the batch.

Algorithm 1 DREE Lifelong Learning

Input: $\{P_t\}_{t=1}^{T}, \theta_0$
Parameters: $\alpha, \beta, \mathrm{UB}, \mathrm{LB}, |\mathcal{B}|$
Output: θ
Initialize: $\mathcal{B} \leftarrow \emptyset$, $\theta \leftarrow \theta_0$
for $t \in \{1, \ldots, T\}$ **do**
  for $i \in \{1, \ldots, I\}$ **do**
    $\{p\} \leftarrow$ a batch of M instances from P_t
    $\{[s_\theta, \pi_\theta], f(\theta, p)\} \leftarrow$ Solve $\{p\}$ with θ
    $\mathcal{L}_{\mathrm{DRL}} \leftarrow$ base solver loss
    if $\mathcal{B}$ is not empty **then**
      $\{e\} \leftarrow$ uniformly sample
      $\mathcal{L}_{\mathrm{KR}} \leftarrow$ KR
      if $i - i' = N$ **then**
        $\mathcal{L}_{\mathrm{PIR}}, M^+ \leftarrow$ PIR
        $i' \leftarrow i$
        $N \leftarrow \mathrm{UB} - \frac{M^+}{M}(\mathrm{UB} - \mathrm{LB})$
      else
        $\mathcal{L}_{\mathrm{PIR}} \leftarrow 0$
      end if
    else
      $\mathcal{L}_{\mathrm{PIR}} \leftarrow 0$, $\mathcal{L}_{\mathrm{KR}} \leftarrow 0$
    end if
    $\mathcal{L} \leftarrow \mathcal{L}_{\mathrm{DRL}} + \alpha\mathcal{L}_{\mathrm{KR}} + \beta\mathcal{L}_{\mathrm{PIR}}$
    $\theta \leftarrow$ Optimize(θ, $\mathcal{L}$)
    $\mathcal{B} \leftarrow$ reservoir sampling $(\{\{[s_\theta, \pi_\theta]\}\}, \mathcal{B})$
    update e with $\{\langle p, f(\theta, p), [(s_\theta, \pi_\theta)]\rangle\}$ for any $f(\theta, p) < f_b$
  end for
end for
return θ

A.2 Hyperparameters

DREE introduces five new hyperparameters, i.e., the coefficients of KR and PIR loss terms α and β, the buffer size $|\mathcal{B}|$, and the upper and lower bounds of the PIR interval UB and LB. We set $\alpha = 100$ for CVRP and $\alpha = 10$ for TSP, which have the best performance in (Pei et al., 2026). Intuitively, we set $\beta = 1$ as PIR is analogous to training on new-task instances (obtaining $\mathcal{L}_{\mathrm{DRL}}$).

The UB and LB determine the admissible range of N, thereby affecting the overall computational overhead of PIR throughout lifelong learning as well as the resulting plasticity–stability trade-off. In our implementation, we set this range heuristically to 1–4. As demonstrated in our experiments (cf. Section 5.4), DREE can automatically adapt its behavior, making its performance relatively insensitive to this choice.

$|\mathcal{B}|$ controls the buffer's coverage of instances from previous tasks as well as the diversity of stored instances. Intuitively, a larger $|\mathcal{B}|$ can improve performance, but it also incurs higher storage cost. We set $|\mathcal{B}| = 256$ because it provides broad coverage while maintaining acceptable storage overhead.

A.3 Computation, Training Time, and Memory

We analyze the resources used by DREE at three levels: neural computation, elapsed training time, and persistent storage. The analysis uses the batch-level implementation in Appendix B. A buffer entry in DREE contains a complete instance batch and its behavior trajectories, whereas a buffer entry in LLR-BC contains a batch of state–behavior pairs at one decision step. This distinction determines both encoder reuse and storage requirements.

A.3.1 Per-batch computation

Let b denote the batch size, n the problem scale, and H the average number of decision steps. Let c_E and c_D denote the forward-computation costs of encoding one instance batch and executing one decoder step on that batch, measured in a common unit such as FLOPs. For an encoder–decoder solver that encodes an instance once and reuses its embeddings during construction, the leading neural workload of a complete batch is

$$G(n, H, b) = c_E(n, b) + Hc_D(n, b). \tag{1}$$

The number of parallel trajectories and the augmentation setting are included in these batch-level costs. Comparisons using this expression hold them fixed.

Current-task learning and PIR. Both operations construct solutions with the current policy and evaluate the base solver's reinforcement-learning loss. Each therefore has a leading forward workload G, evaluated at the size and trajectory length of the corresponding batch. The backward pass has the same asymptotic order for a fixed architecture, with an implementation-dependent constant. Re-solving buffered instances supplies the candidate trajectories and objective values required by EE.

KR. For a complete buffered trajectory batch, DREE encodes the instance batch once and evaluates the decoder at its buffered states. Its leading forward workload is also G, with additional probability-matching and confidence-weighting arithmetic. With dense action distributions, this additional arithmetic is $O(bHn)$ for a fixed number of parallel trajectories. In the step-level implementation of LLR-BC, replaying q independently sampled state–behavior batches entails q encoder calls and q decoder calls, giving $q(c_E + c_D)$ before loss evaluation and backpropagation.

EE and buffer operations. EE uses objective values already obtained during PIR. Comparing them with the buffered values takes $O(b)$ scalar comparisons per replayed batch. When a comparison succeeds, the corresponding objective and trajectory are written to the buffer. Writing complete trajectories requires time proportional to their representation size, up to $O(bHn)$ for dense state–behavior records at a fixed trajectory multiplicity. EE adds no extra network forward or backward pass. Reservoir insertion similarly involves a constant-time retention decision for each arriving batch and a write proportional to the retained record size.

A.3.2 Per-epoch neural workload and training budget

Let B be the number of incoming batches per DREE epoch, R the number of PIR batches in that epoch, and B_L the number of incoming batches used by LLR-BC. DREE executes one KR operation for each incoming batch once the buffer is available. Under a representative common (n, H, b), the forward-workload approximations are

$$W_{\text{DREE}} \simeq (2B + R)(c_E + Hc_D), \tag{2}$$

$$W_{\text{LLR-BC}} \simeq B_L(c_E + Hc_D) + qB_L(c_E + c_D). \tag{3}$$

For heterogeneous batch sizes and trajectory lengths, each term is replaced by the sum of the corresponding batch costs. These expressions count the dominant network evaluations; full training also includes backward passes, environment updates, loss evaluation, buffer transfers, and optimizer operations.

For the synthetic POMO setting, this encoding-reuse accounting uses $B = 128$, $B_L = 164$, and $q = 16$. The observed average PIR count is approximately $R = 36$ per epoch. Taking $H = 50$ as a representative trajectory length gives

$$W_{\mathrm{DREE}} \simeq 292c_E + 14{,}600c_D, \tag{4}$$

$$W_{\mathrm{LLR\text{-}BC}} \simeq 2{,}788c_E + 10{,}824c_D. \tag{5}$$

DREE thus uses approximately 10.5% as many encoder evaluations and 134.9% as many decoder evaluations in this accounting. Its forward workload is no greater when $2{,}496c_E \geq 3{,}776c_D$, or $c_E/c_D \geq 1.513$. The realized advantage depends on the backbone, the sizes of replayed instances, and the cost of the remaining training operations.

The episode-budget comparison counts complete solution constructions: DREE performs approximately $128 + 36 = 164$ such batches per epoch, while the compared methods are assigned 164 batches. KR evaluates historical states without generating a new solution-construction episode. Matching solution-construction counts therefore does not impose equality of FLOPs, wall-clock time, or memory. The adaptive interval changes R and hence PIR computation; the buffer capacity changes persistent storage while the number of sampled replay batches per update remains fixed.

A.3.3 Reported FLOPs and wall-clock measurements

Table 5 reports the KR computation counts for the investigated backbones and problem sizes. In this table, N denotes problem scale, m denotes the number of depot returns, and B and M denote billions and millions of floating-point operations. For example, the reported POMO CVRP count at scale 20, batch size 32, and three depot returns is approximately 2.85 billion FLOPs. These KR-level counts describe a component of the training workload, not the total cost of an epoch.

Table 5: Computational cost of KR under different cases and base solvers, in terms of FLOPs. Notably, Omni and POMO share the same model architecture and thus the same cost.

Base solver	Problem	$N = 20$	$N = 50$	$N = 100$
POMO	CVRP	2.84B + 4.26M $\times\, m$	7.20B + 5.46M $\times\, m$	15.28B + 7.45M $\times\, m$
POMO	TSP	2.73B	7.14B	15.28B
INViT	CVRP	78.53B+4.13B$\times\, m$	300.79B+6.14B$\times\, m$	607.71B+6.14B$\times\, m$
INViT	TSP	78.53B	300.79B	607.71B

Table 6 reports the average elapsed time per epoch across the five synthetic task orders, in seconds. In these measurements, DREE uses less time than LLR-BC for both problems, while fine-tuning has the lowest time cost. The comparison reflects the full measured implementation, including the different replay and distillation operations.

Table 6: Average wall-clock seconds per epoch across the five synthetic task orders.

Problem	Fine-tuning	Li	DREE	LLR-BC
CVRP	83.34	125.12	142.61	147.23
TSP	57.52	108.65	103.67	144.04

Table 7 separately reports PIR component timings and their ratios to the listed fine-tuning measurements. In the synthetic CVRP order-1 profiling record, DREE takes 151.26 seconds per epoch and LLR-BC takes 170.53 seconds; PIR accounts for 26.77% of DREE's time. The percentages in Table 7 instead use that table's fine-tuning time as the denominator. Component ratios and complete-method times describe different quantities and are reported separately.

A.3.4 Persistent storage

Let C_D and C_L be the numbers of retained batch entries in DREE and LLR-BC, respectively. Let U_I be the storage of an instance batch, U_B the storage of a one-step behavior batch excluding the instance already counted in U_I, and U_f the storage of the batch's solution objectives. These units

Table 7: Wall-clock time (seconds) of PIR and its corresponding percentage of the wall-clock time of fine-tuning, under different task orders. GD refers to the gradually drifting scenario, and PS refers to the periodically stationary scenario.

Base Solver	Task Order	CVRP			TSP		
		PIR	Fine-tuning	Percentage	PIR	Fine-tuning	Percentage
POMO	GD1	40.49	88.81	45.59%	28.99	58.96	49.17%
	GD2	30.91	73.63	41.98%	31.27	61.00	51.26%
	GD3	45.65	89.01	51.29%	18.33	58.27	31.46%
	GD4	40.80	76.55	53.30%	25.34	51.33	49.37%
	GD5	47.84	88.69	53.94%	27.77	58.06	47.83%
	PS1	20.02	73.65	27.18%	19.56	59.90	32.65%
	Avg.	37.61	81.72	46.02%	25.21	57.92	43.53%
INViT	GD1	53.26	139.35	38.22%	41.87	95.04	44.06%

include the data types and the number of recorded trajectories. With nonuniform trajectory lengths, DREE's buffer storage is the sum over its records:

$$S_{\mathrm{DREE}} = \sum_{j=1}^{C_D} \left(U_{I,j} + U_{f,j} + \sum_{h=1}^{H_j} U_{B,j,h} \right) + O(C_D). \tag{6}$$

The final term represents indexing and bookkeeping. With representative per-batch sizes, the comparison becomes

$$S_{\mathrm{DREE}} \simeq C_D(U_I + U_f + HU_B), \tag{7}$$

$$S_{\text{LLR-BC}} \simeq C_L(U_I + U_B). \tag{8}$$

The LLR-BC expression accounts for the instance information accompanying each independently stored state–behavior batch. Any explicit sharing of instance arrays would need to be reflected in the corresponding storage units.

With fixed-width node features and dense action distributions, $U_I = O(bn)$, $U_B = O(bn)$, and $U_f = O(b)$ at a fixed number of recorded trajectories. Thus DREE's buffer has size $O(C_D bn(1+H))$, whereas the LLR-BC step-level buffer has size $O(C_L bn)$. For fixed C_D, bounded instance size, and bounded trajectory length, DREE's retained memory does not grow with the number of tasks encountered.

For the representative $H = 50$ and $C_L = 1000$, the default DREE capacity $C_D = 256$ gives, omitting objective scalars and bookkeeping,

$$S_{\mathrm{DREE}} \simeq 256U_I + 12{,}800U_B, \qquad S_{\text{LLR-BC}} \simeq 1000U_I + 1000U_B. \tag{9}$$

The smaller-buffer setting $C_D = 64$, evaluated in Table 20, gives $S_{\mathrm{DREE}} \simeq 64U_I + 3200U_B$. Under this accounting, the relative storage depends on the representation ratio U_I/U_B: DREE uses no more storage when this ratio is at least $11{,}800/744 \simeq 15.86$ for $C_D = 256$, or $2200/936 \simeq 2.35$ for $C_D = 64$. Equal numbers of buffer entries therefore do not imply equal byte budgets. The available buffer-size experiments describe a performance–capacity trade-off; an equal-byte performance comparison is not reported.

Other stored state. All methods retain the current solver and its training state. For P model parameters, the parameter, gradient, and fixed-size optimizer-state storage is $O(P)$. Fine-tuning does not retain historical experience. Li additionally retains a reference model, with $O(P)$ parameter storage. Its real-world adaptation described in Appendix B also retains an instance buffer of 256 batches for each previously observed task; for K_t such task buffers, this contributes approximately $256K_tU_I$. MT-cu retains the incoming instances accumulated over time. At 1000 epochs with 128 incoming batches per epoch, it retains 128,000 instance batches, which is 500 times the default DREE entry count of 256. A byte-level comparison additionally accounts for DREE's stored trajectories and objective values.

A.3.5 WORKING MEMORY AND INFERENCE

Persistent replay storage and peak training-device memory measure different resources. The latter also includes the current batches, intermediate activations, gradients, and temporary tensors for the operations being differentiated. It depends on the architecture, batch size, trajectory length, augmentation, and whether computation graphs coexist before an optimizer update. The buffer-storage expressions do not quantify this peak; no measured peak-device-memory values are reported here.

For a fixed batch workload, DREE samples a fixed number of entries for KR and PIR rather than scanning all retained experience. Increasing buffer capacity therefore increases persistent storage without changing the leading neural workload of an individual update. During inference, DREE uses the base solver without its replay buffer, PIR, KR, or EE. With the same backbone and decoding configuration, its inference architecture and asymptotic inference cost are unchanged.

A.4 NOVELTY AND DIFFERENCE TO EXISTING LIFELONG LEARNING SOLVERS

The default DREE implementation buffers instances, objectives, and behaviors for knowledge consolidation. It reuses encountered instances and preserves knowledge through behavior targets. DREE-Li instead uses the reference-model consolidation of Li (Li et al., 2025a), as detailed in Appendix F.3.

DREE adopts the behavior consolidation mechanism of LLR-BC (Pei et al., 2026), a state-of-the-art lifelong learning method for neural VRP solvers. Its confidence-aware weighting and reverse-KL matching preserve historical decision patterns. DREE addresses the low quality of behaviors collected under insufficient per-task training through a replay–evaluate–refine process. PIR constructs new solutions for buffered instances, objective values assess their quality, and EE replaces the historical behavior targets when better solutions are found. Subsequent KR then consolidates these refined targets. The following differences describe this coupling.

- DREE defines experience at the instance level, represented by an instance, its objective value, and the corresponding solution trajectory, thereby retaining richer information. In contrast, LLR-BC defines experience at the step level, using only state–behavior pairs.
- DREE replays problem instances to enable further exploration for better solutions, whereas LLR-BC does not support this type of replay.
- By comparing objective values, DREE can assess and compare the quality of stored experiences, which facilitates continuous improvement of experience quality. LLR-BC does not provide such a mechanism.
- In DREE, buffered experiences can be updated when better solutions are found. In LLR-BC, however, experiences remain fixed once stored, regardless of their quality.

Relation to DER and DER++. Dark Experience Replay (DER) (Buzzega et al., 2020) stores inputs and historical logits, and preserves past predictions through logit matching. DER++ additionally replays ground-truth labels through a supervised loss. Both use reservoir sampling to retain experience from the stream. These mechanisms establish replay and prediction matching as general tools for continual learning.

In neural routing, a buffered trajectory can be evaluated directly through its solution objective. DREE uses this signal to connect instance replay to behavior consolidation. PIR generates a candidate trajectory, and EE updates its stored objective and behavior target when the solution improves. KR subsequently learns from that target. Table 8 summarizes the conceptual relationship. This comparison concerns the replay mechanisms and their training signals.

A.5 EARLY EXPERIENCE REFINEMENT

Insufficient per-task training can produce low-quality initial behavior targets. To examine how actively EE refines these targets early in learning, we record its objective comparisons in epoch 2 of TSP task order 1. Across 96 replayed batches of 32 instances, EE makes 3,072 comparisons and performs 2,715 behavior-target updates, corresponding to an update rate of 88.4%. These comparison-level events

Table 8: Conceptual comparison of replay mechanisms.

Method	Retained information	Historical learning signal	Target refinement
DER	Inputs and logits	Historical logit matching	Logits retained from collection
DER++	Inputs, logits, and labels	Logit matching and supervised replay	Logits retained from collection
DREE	Instances, objectives, and trajectories	RL instance replay and behavior matching	Better routing objectives gate behavior-target replacement

show that refinement is active early in training. An entry may be sampled and updated repeatedly, so the event count measures refinement activity. Sensitivity to different initial experience qualities requires a separate controlled comparison.

B COMPARED METHODS AND IMPLEMENTATION DETAILS.

All compared methods are implemented on top of the same base neural solver to ensure a fair comparison. All methods are trained from the same randomly initialized model. We use instance augmentation (Kwon et al., 2020) with a factor of 8 during learning. All experiments are conducted on a cluster with 512 GB of memory, utilizing a single Nvidia A100. A full synthetic run took approximately 2 days, and the real-world run took approximately 12 hours.

LLR-BC. LLR-BC (Pei et al., 2026) is a behavior-level lifelong learning method for neural VRP solvers, designed to mitigate catastrophic forgetting while preserving the solver's ability to adapt to newly arriving tasks. Instead of constraining model parameters directly, LLR-BC stores past solving experiences in a fixed-size buffer maintained by reservoir sampling and consolidates previous knowledge by aligning the current policy with buffered historical behaviours. Its main components are confidence-aware experience weighting, which assigns larger weights to low-confidence decisions, and decision-seeking behavior consolidation, which uses reverse KL divergence to encourage the current solver to preserve the high-probability actions recorded in previous behaviors. This makes LLR-BC particularly suitable for lifelong neural routing, as it consolidates decision patterns rather than entire models or regenerated instances. In our continual drifting scenarios, we adopt LLR-BC using the official implementation and the recommended hyperparameter settings. Since the original LLR-BC design caches experiences in the final epoch of each task, and each epoch in our continual drifting setting corresponds to a new task, we buffer experiences at every epoch following the same design principle. Appendix G.2 further compares LLR-BC variants with different buffering frequencies, and the results show that this setting does not introduce negative performance effects. All other configurations are kept the same as in the original LLR-BC paper.

Li. *Li* (intra version) (Li et al., 2025a) combines regularization with the experience replay strategy in the original framework. It periodically updates the exemplar model with a fixed epoch interval and regularizes the current policy toward the most recent exemplar model. The training objective combines the original reinforcement learning loss with a KL-divergence-based regularization term, weighted by a coefficient α. This design encourages the model to retain desirable recent behaviours while continuing to adapt to the current problem distribution. In addition, the original framework revisits instances of previous tasks while learning new tasks by randomly selecting an old task and generating new instances of it on the fly. We adapt Li to the continual drifting scenario with the updating interval as 25 epochs, identical to the original paper. Following the original formulation, the current policy is regularized toward the most recently saved reference model during training. Since the original paper does not specify a recommended value or tuning procedure for α, we set $\alpha = 0.5$, following (Pei et al., 2026). Since new problem instances cannot be generated arbitrarily in our real-world continual drifting scenarios, we maintain an instance buffer with a size of 256 batches (comparable to the buffer size used by DREE) for each previously observed task using the instances encountered during training, maintained with reservoir sampling. All other settings are

kept consistent with the original method whenever applicable. As no official implementation was available, we implemented it ourselves based on the methodological description in the paper.

MT-Ref. MT-Ref is an intuitive multitask learning setting we included as a performance reference baseline. This method has access to all past and future tasks and can generate instances from each task on the fly. Specifically, for each batch in an epoch, it randomly selects one task and trains on newly generated instances from that task. Joint access to all tasks removes sequential task unavailability, but finite training and interference among tasks can still limit performance. Notably, *MT-Ref* only serves as an idealized reference, and it is not practically applicable in lifelong learning scenarios. Thus, we only run MT-Ref once on task order 1 on the synthetic dataset, also because different task orders have the same principal tasks. In the real-world dataset, MT-Ref is run on the tasks of each region, as different regions have significantly different task characteristics. In addition, we design a more realistic cumulated multitask variant, which cannot access future tasks and instead incrementally records tasks as they arrive, denoted *MT-cu*. Every 200 epochs, it trains from scratch on all collected tasks using the same training procedure as the multitask reference for 200 epochs. The performance of this cumulated multitask variant is reported in Appendix G.3. The results show that DREE significantly outperforms this periodically retrained cumulated multitask method.

Excluded baselines. The LLR-BC study (Pei et al., 2026) involves more lifelong learning baselines, which are all model-buffering based, for comparison in the periodically stationary scenario, including Li (inter version), LiBOG (Pei et al., 2025a), Feng (Feng et al., 2025), and EWC (Kirkpatrick et al., 2017). However, we do not compare against them in our continual drifting scenarios because these methods require storing a separate model for each stationary task and comparing the current model with every previously stored model during learning. In our setting, however, each epoch corresponds to a new task induced by the continuously changing real-world distribution. Therefore, the number of required historical models would grow with the number of epochs, making their computational and memory costs prohibitive. We also exclude *Restart*, which trains an independent model for each task. Since each real-world task contains only one epoch of data, restarting from scratch for every task is unable to sufficiently learn the task-specific distribution and is expected to perform poorly. Furthermore, LLR-BC, which we have included, is verified to be able to outperform these methods (Pei et al., 2026).

C Benchmark from Real-World Dataset

C.1 LaDe: Real-world Urban Delivery Dataset

LaDe We obtain real-world daily delivery data from the **LaDe** dataset (Wu et al., 2024), which is a publicly available large-scale urban logistics dataset collected from real industrial operations. LaDe covers five Chinese cities, namely *Shanghai*, *Hangzhou*, *Chongqing*, *Jilin*, and *Yantai*, and spans six months from *2022-05-01* to *2022-10-31* (Wu et al., 2024). The dataset contains both pickup and delivery data. In this work, we focus on the package delivery data in LaDe, i.e., LaDe-Delivery. As the number of regions varies significantly between cities, we select one region per city, namely `sh-2`, `hz-120`, `jl-31`, `yt-167`, and `cq-144`, for equal analysis and experiments among cities.

LaDe-Delivery is organized as package-level delivery records rather than pre-defined combinatorial optimization instances. Each record contains package-, courier-, spatial-, and temporal-related information, including package identifier, courier identifier, delivery date, geographic coordinates, city, region identifier, AOI information, and timestamps associated with the delivery process (Wu et al., 2024). Notably, LaDe-Delivery is not originally released as a benchmark for lifelong learning for VRP. Instead, the original paper positions LaDe as a comprehensive real-world dataset for prediction problems like *estimated time of arrival (ETA) prediction* and *spatio-temporal graph forecasting* (Wu et al., 2024).

Table 9 summarizes the key characteristics of the regions. These regions differ substantially in scale and temporal evolution, providing diverse real-world lifelong learning environments.

From region-day package sets to a stream of real-world TSP instances. To transform LaDe-Delivery into a benchmark for lifelong combinatorial optimization, we group all package records by *city*, *region*, and *delivery date*. For each region-day tuple, all packages delivered in that region on

Table 9: Characteristics of the selected regions. PpD stands for the number of packages per day.

Region	City	PpD Range	PpD Mean (std.)
sh-2	Shanghai	4–575	254.84 (146.69)
hz-120	Hangzhou	88–608	327.8 (97.9)
jl-31	Jilin	2–175	56.9 (28.7)
yt-167	Yantai	9–108	42.2(21.8)
cq-144	Chongqing	3–153	26.3 (32.0)

that day are treated as nodes, and together form one raw TSP instance with real-world customers. Demand and capacity information are not provided in LaDe (e.g., the size/weight of a package), so we formulate TSP instances instead of CVRP instances. We do linear minmax normalization on each instance so that all nodes are located in $[0, 1]^2$, following the common practice. Formally, for a region r, and date t, we define

$$I_{r,t} = \{x_1, x_2, \ldots, x_{n_{r,t}}\},$$

where each node x_i corresponds to one package delivery location (coordinate) in the corresponding region-day package set, and $n_{r,t}$ is the number of packages on that day.

For a fixed region r, sorting these daily instances by date yields a chronological stream

$$\mathcal{S}_r = (I_{r,1}, I_{r,2}, \ldots, I_{r,T}),$$

which we regard as a **stream of raw real-world TSP instances**. At this stage, each instance is directly induced by a real operational day in a real delivery region.

Figure 5 demonstrates the location of nodes on each day and each region, with different days marked with different colors. Intuitively, some general drift trend is observed: in the early days customers are more likely to concentrate in some clusters, and over the days, the customers gradually spread to the whole space.

C.2 Real-World Continual Drift

The real-world delivery problem instance stream in each region exhibits **continual drift** over time. We characterize such drift from two complementary perspectives: **problem scale** and **customer location distribution**.

Scale drift. For each region stream $\mathcal{S}_r$, we first track the daily instance size $n_{r,t}$, i.e., the number of package nodes in the corresponding region-day instance. By visualizing $n_{r,t}$ over time (cf. Figure 6), we observe that although local fluctuations exist, the overall trend shows a pattern of continual drift. This indicates that the real-world stream exhibits gradual temporal variation in instance size.

Distribution drift. The problem scale alone does not fully characterize the drifting of the delivery distribution. We therefore further quantify the structural discrepancy between instances using the **Gromov–Wasserstein (GW) distance** (Mémoli, 2011), which is suitable for comparing relational structures defined on different node sets.

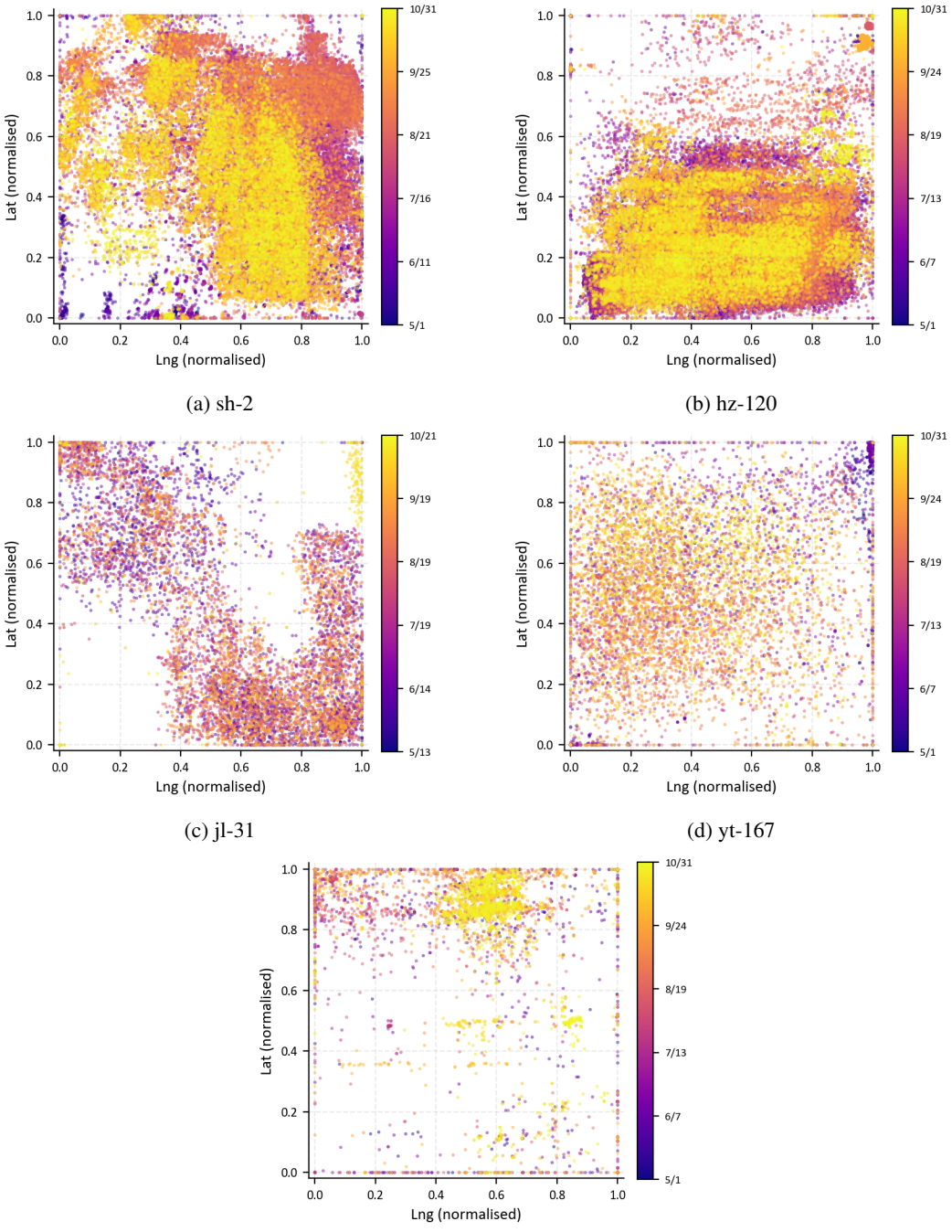
Given two TSP instances I_t and I_{t+h} with empirical node distributions μ_t and μ_{t+h} and intra-instance node matrices $D_t \in \mathbb{R}^{n_t \times n_t}$ and $D_{t+h} \in \mathbb{R}^{n_{t+h} \times n_{t+h}}$, respectively, we compute their GW distance as

$$\mathrm{GW}(I_t, I_{t+h}) = \min_{\pi \in \Pi(\mu_t, \mu_{t+h})} \left(\sum_{i,i',j,j'} \left| D_t(i,i') - D_{t+h}(j,j') \right|^2 \pi_{ij} \pi_{i'j'} \right)^{1/2},$$

where $\Pi(\mu_t, \mu_{t+h})$ denotes the set of couplings between the two empirical measures. In our implementation, the ground structure (i.e., D) of each instance is defined by its pairwise Euclidean distance matrix over node coordinates.

To measure temporal drift at different scales, we compute the **average lag-h GW distance**:

$$\overline{\mathrm{GW}}(h) = \frac{1}{T-h} \sum_{t=1}^{T-h} \mathrm{GW}(I_t, I_{t+h}), \qquad h = 1, 2, \ldots, H.$$



(a) sh-2

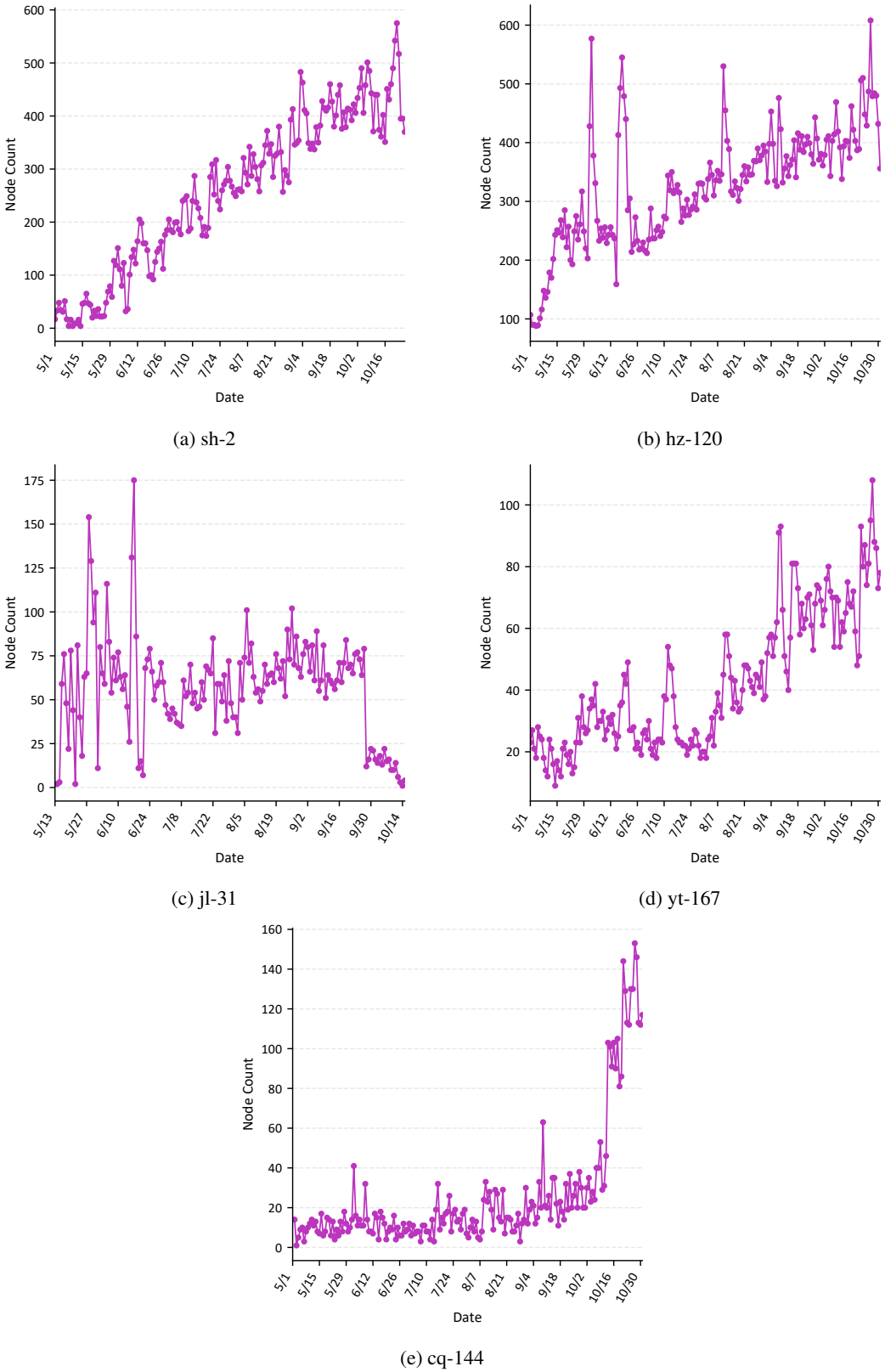
(b) hz-120

(c) jl-31

(d) yt-167

(e) cq-144

Figure 5: TSP instance stream in each region. Nodes of different days are represented by different colors.



(a) sh-2

(b) hz-120

(c) jl-31

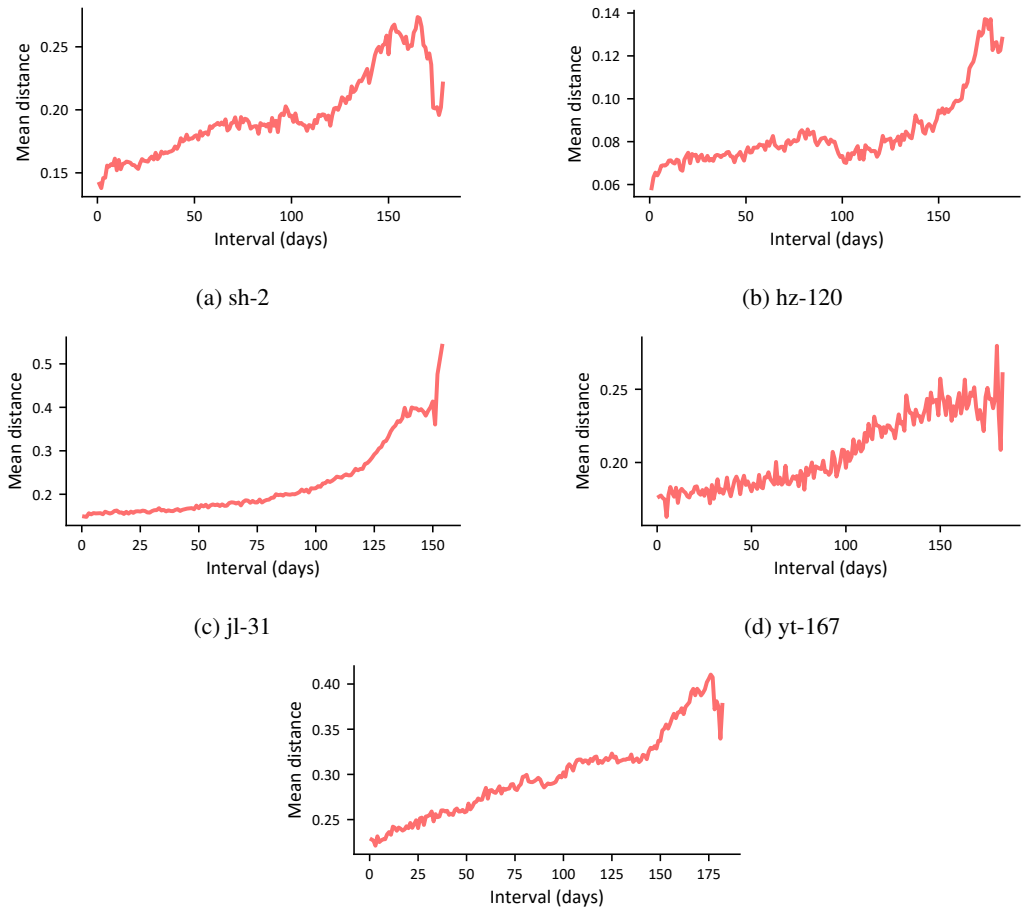
(d) yt-167

(e) cq-144

Figure 6: Daily instance sizes over time for the selected region streams. Although local fluctuations exist, the overall scale changes progressively in most of the cases, indicating real-world continual drift in problem size.

If the stream is gradually drifting, then instances that are farther apart in time should, on average, become more dissimilar. Accordingly, we expect $\overline{\mathrm{GW}}(h)$ to show an increasing trend as the lag h grows, possibly with local irregularities due to short-term operational noise.

Figure 7 reports the average lag-h GW distance curves. The downturn at the end of the curves suggests that previous patterns, or patterns similar to earlier ones, may reappear over time. This highlights the importance of retaining knowledge from previous tasks. We observe that the average structural discrepancy generally increases with temporal lag, again with local fluctuations. Together, these two analyses provide direct evidence that the real-world delivery streams exhibit the target form of **continual drift**: the underlying problem distribution changes over time in a globally progressive but locally noisy manner.



(a) sh-2

(b) hz-120

(c) jl-31

(d) yt-167

(e) cq-144

Figure 7: Average lag-h GW distance for the selected region streams. The average structural discrepancy generally increases with temporal lag, supporting the existence of continual distribution drift in real-world delivery problems.

C.3 From Raw TSP Instance Streams to a Real-World Lifelong Learning Dataset

We next convert the region-day-level real-world TSP streams into a lifelong learning scenario for benchmarking continual combinatorial optimization methods. For each selected region, we retain only the days whose original region-day instance contains at least 10 package nodes, in order to avoid extremely small and uninformative cases.

Instance resampling. The original real-world data stream contains large-scale instances with nearly one thousand customers, which exceed the capability of the base solvers used in this work. However, addressing such large-scale optimization challenges is beyond the primary scope of this study. To obtain solver-compatible instances while preserving the original spatial distribution, we generate benchmark instances by **node resampling** from each original region-day instance. Specifically, for an original instance with n_t nodes, we sample a subset whose size is determined by a region-specific ratio q and then clip it into the interval $[10, 100]$:

$$\tilde{n}_t = \mathrm{clip}\left(\lfloor q \cdot n_t \rfloor,\ 10,\ 100\right).$$

The value of q is chosen separately for different regions according to their original scale characteristics. q is 0.3, 0.2, 0.6, 0.9 and 0.7 for sh-2, hz-120, jl-31, yt-167 and cq-144, respectively. The same resampling protocol is applied to all benchmark instances used in training and testing.

Another benefit of resampling is that, for each time step, namely each day, we obtain a distribution of problem instances rather than a single problem instance. This is consistent with the design principle of existing neural solvers, which typically learn from multiple problem instances sampled from an underlying distribution rather than from one isolated example.

Training and test splits. For each region stream, we sort the valid daily instances chronologically. The first 95% of days are used to form the lifelong training stream, while the last 5% of days are held out as a future generalization test set, for obtaining the Gen value. This split evaluates whether a lifelong method can generalize to future real-world data beyond the training horizon.

In addition, to measure adaptation and forgetting on previously encountered patterns, we construct a process test set, similar to the principal-task test set in the synthetic dataset, for obtaining AP, AFB, AMFB and ABPl. Concretely, we generate one test instance set every 30 days along the training horizon. These periodic test sets are built from the corresponding historical real-world instances and are used to evaluate the performance of each lifelong learning method on previously observed tasks/patterns over time.

Reference solutions and gap computation. Following the setting on the synthetic dataset, we use Gurobi to solve each test instance in this real-world dataset to obtain an reference solution whenever tractable. These reference solutions are used to compute the gap of each compared lifelong learning method. For a method producing solution length L_i on instance i, with Gurobi optimum $L_i^\star$, we define the relative gap as

$$\mathrm{Gap}_i = \frac{L_i - L_i^\star}{L_i^\star} \times 100\%.$$

We report the average gap over each test set. The Gurobi-derived optima also serve as the reference for evaluating the absolute solution quality of different lifelong learning methods.

C.4 Amazon CVRP Streams

We construct four lifelong CVRP streams from the Amazon Last Mile Routing Research Challenge dataset (Merchán et al., 2022), with each stream corresponding to one delivery station. Customer demand is derived from package volume. For each station, the largest total demand of an observed single route defines a capacity proxy. Instance resampling and chronological training–test splitting follow the LaDe protocol in Appendix C.3, using the first 95% of valid days for learning and the final 5% for future-task evaluation.

Station characteristics. Table 10 summarizes the characteristics of the four station streams. The stations are `DBO2`, `DBO3`, `DSE4`, and `DSE5`.

The `DBO2` and `DSE5` streams cover 2018-06-02 to 2018-10-02 (123 calendar days), with 26 and 20 missing days, respectively. The `DBO3` and `DSE4` streams cover 2018-06-01 to 2018-10-02 (124 calendar days), with 16 and 12 missing days, respectively.

The capacity proxies for `DBO2`, `DBO3`, `DSE4`, and `DSE5` are 4,221,904.004, 4,302,357.095, 3,698,959.590, and 3,762,682.669 cm^3, respectively.

Table 10: Characteristics of the selected Amazon stations. Size is the number of customer nodes per raw CVRP instance. Size means and population standard deviations are computed over valid days.

Station	City	Valid days	Routes	Size range	Size mean (std.)
DBO2	Boston	97	448	59–2716	628.43 (646.45)
DBO3	Boston	108	775	108–5381	1128.37 (1368.54)
DSE4	Seattle	112	612	79–3054	765.91 (771.98)
DSE5	Seattle	103	651	109–3709	959.36 (1024.78)

Spatial distribution. Figure 8 shows customer locations in each station stream. Within each window, coordinates are min–max normalized to $[0, 1]^2$ using the depot and all customers. Color indicates the date.

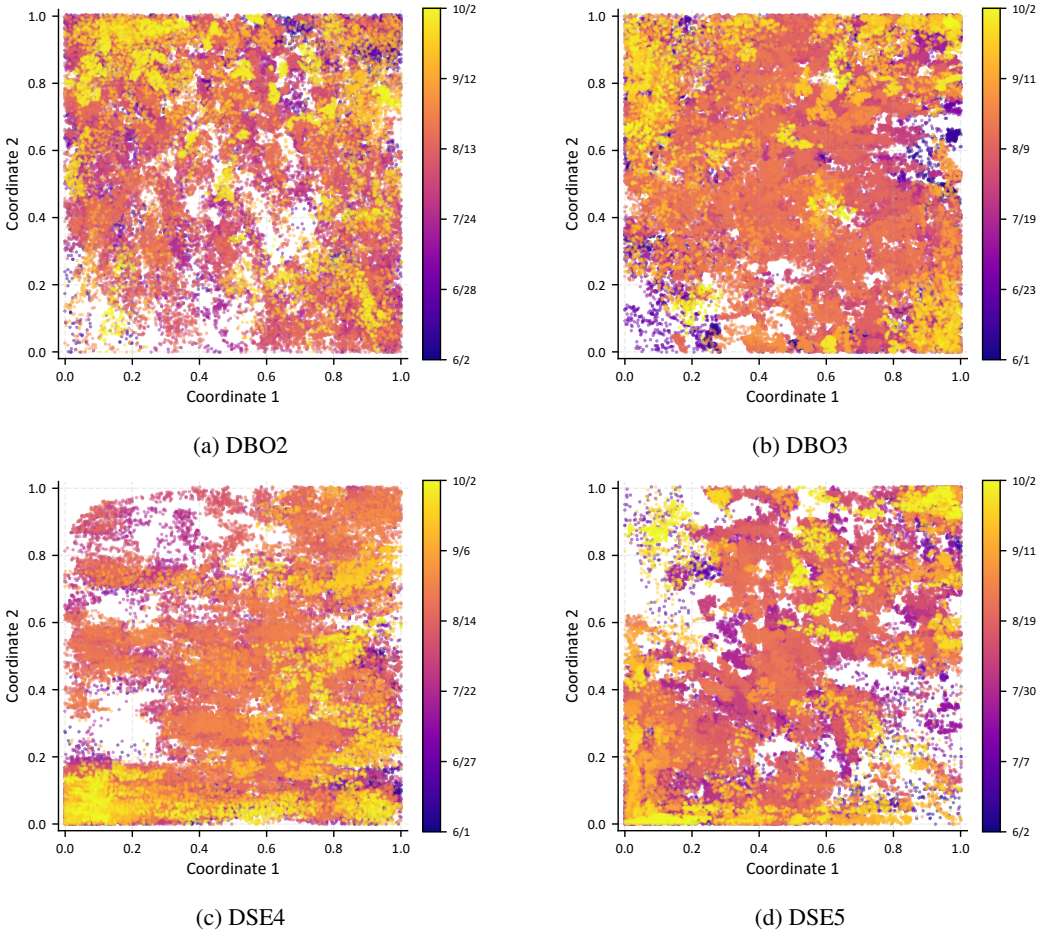
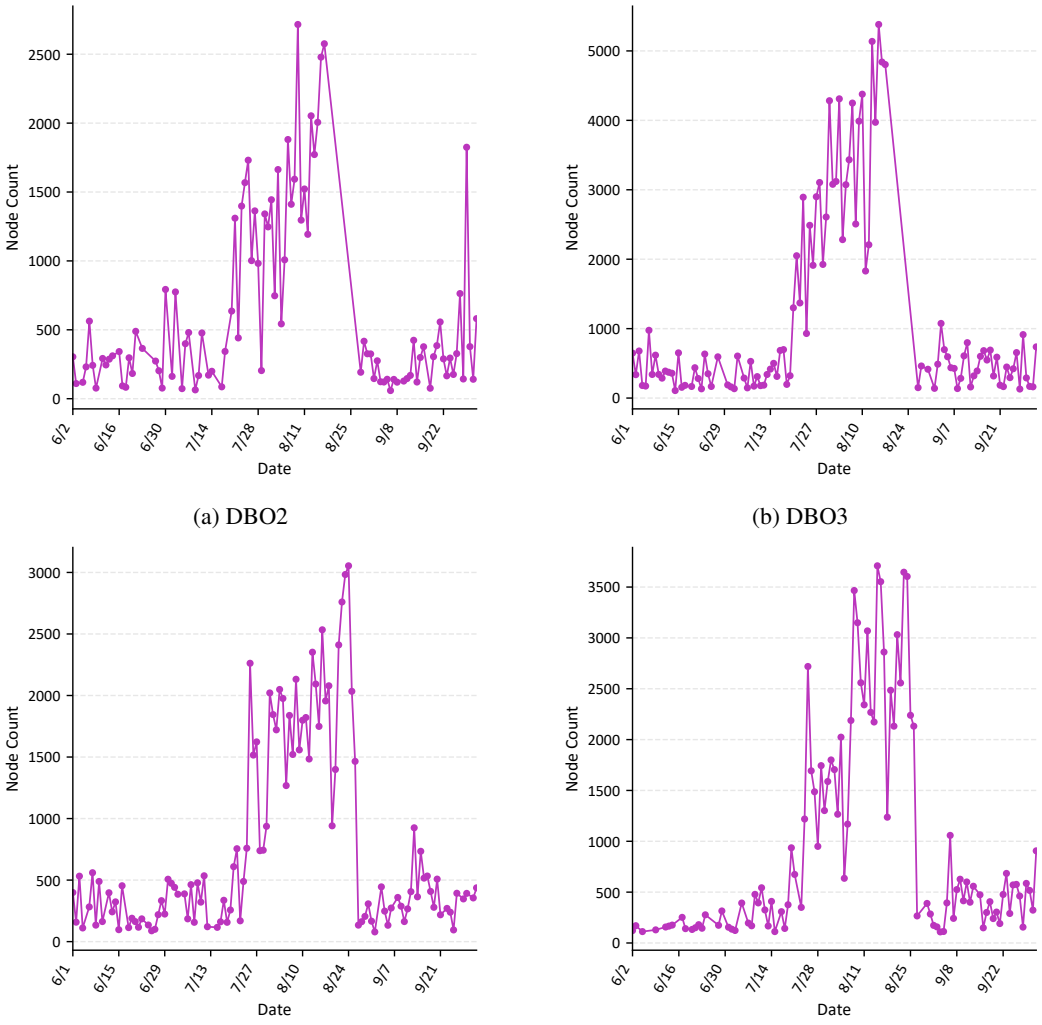


Figure 8: Customer locations in each Amazon station stream, with coordinates displayed in $[0, 1]^2$. Color indicates the date (month/day).

Scale drift. Figure 9 shows the daily instance sizes in each station stream, measured by the number of customer nodes before resampling. Node counts fluctuate substantially, with higher counts in late July and August followed by a sharp decline.

Distribution drift. Figure 10 reports the temporal GW analysis separately for each station, with lag measured in days. Mean GW distances fluctuate with lag, with clearer upward trends for `DBO3` and `DSE4`.



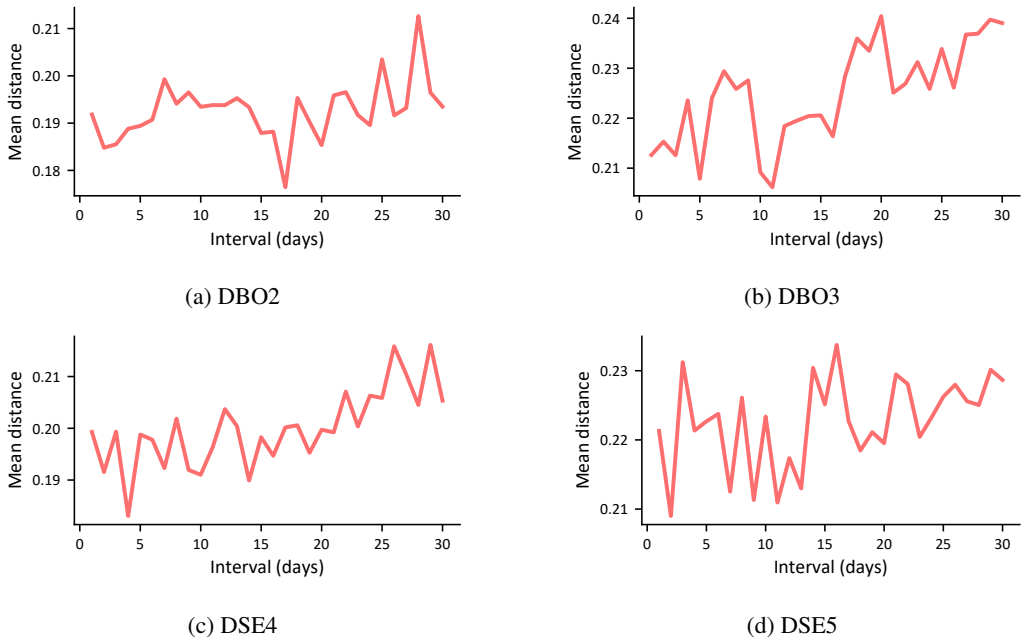
(a) DBO2

(b) DBO3

(c) DSE4

(d) DSE5

Figure 9: Daily instance sizes over time for the selected Amazon station streams.



(a) DBO2

(b) DBO3

(c) DSE4

(d) DSE5

Figure 10: Mean GW distance as a function of temporal lag (days) for each Amazon station stream.

Lifelong learning results. Table 4 reports results aggregated over the four stations. DREE has the lowest mean AP, AFB, and AMFB, while fine-tuning has the lowest ABPl.

D Details of Synthetic Dataset

D.1 Lifelong Learning Tasks

We adopt the six tasks ($P^U, P^R, P^{GM}, P^E, P^C, P^G$) from (Pei et al., 2026) as our principal tasks for the synthetic dataset and restate the instance-generation procedure here for completeness.

The six tasks are named with the node location distribution: Uniform (U), Gaussian Mixture (GM), Explosion (E), Compression (C), Grid (G), and Ring (R). Four of them follow the coordinate-generation protocols used in (Bossek et al., 2019), while the Grid and Ring tasks are additionally included in (Pei et al., 2026) to increase the diversity of spatial structures. Tasks U and R correspond to scale 20, tasks G and E to scale 50, and tasks C and GM to scale 100, following (Pei et al., 2026). For CVRP, each task is also associated with a task-specific demand distribution. Unless otherwise specified, customer/city coordinates are normalized to the unit square $[0, 1]^2$. For CVRP, the depot location is sampled independently from $\mathcal{U}([0, 1]^2)$. Following (Kwon et al., 2020), the vehicle capacity is set to 30, 40, and 50 for problem sizes 20, 50, and 100, respectively.

Uniform (U). For each customer/city node i, its coordinate is sampled independently as

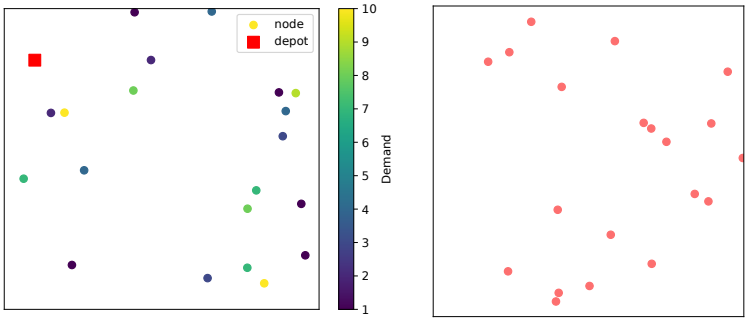
$$(x_i, y_i) \sim \mathcal{U}([0, 1]^2).$$

For CVRP, the customer demand is sampled from a discrete uniform distribution over $\{1, \ldots, 10\}$.

Gaussian Mixture (GM). We first sample five cluster centers $\{c_z\}_{z=1}^5$ independently from $[0, 50]^2$. Around each center, customer/city nodes are generated from an isotropic Gaussian distribution,

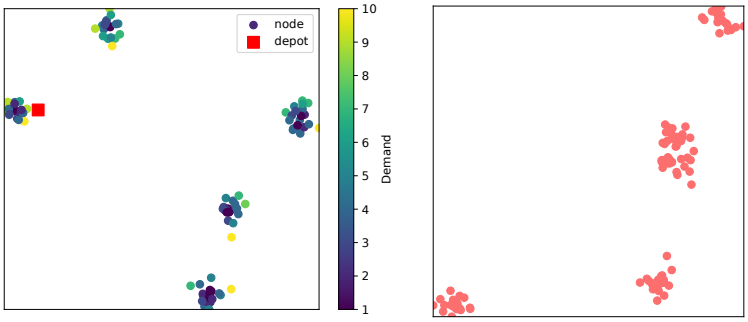
$$(x_i, y_i) \sim \mathcal{N}(c_z, I),$$

where I denotes the identity covariance matrix. The resulting coordinates are then mapped to $[0, 1]^2$ using min–max normalization. For CVRP, the demand is made dependent on the local cluster structure. Specifically, for each node i, we compute its Euclidean distance to the corresponding cluster center, normalize these distances to $[0, 1]$, and set the demand to the rounded value of $10 \cdot \text{dist}_i$.



(a) CVRP (b) TSP

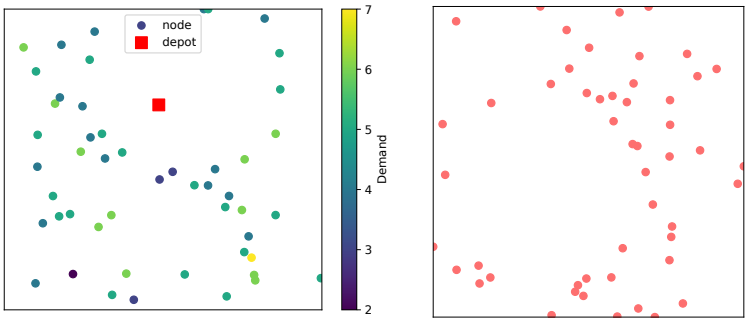
Figure 11: Example problem instance of Task P^U



(a) CVRP (b) TSP

Figure 12: Example problem instance of Task P^{GM}

Explosion (E). The initial node coordinates are sampled uniformly from $[0, 1]^2$. A reference point $p \sim \mathcal{U}([0, 1]^2)$ is then sampled independently. For every node whose distance to p is smaller than 0.3, the node is displaced away from p by a distance $0.3 + s$, where $s \sim \text{Exp}(40)$. After the displacement, all coordinates are clipped to $[0, 1]^2$. For CVRP, each demand is sampled from $\mathcal{N}(5, 1)$, rounded to the nearest integer, and clipped to the range $[0, 10]$.

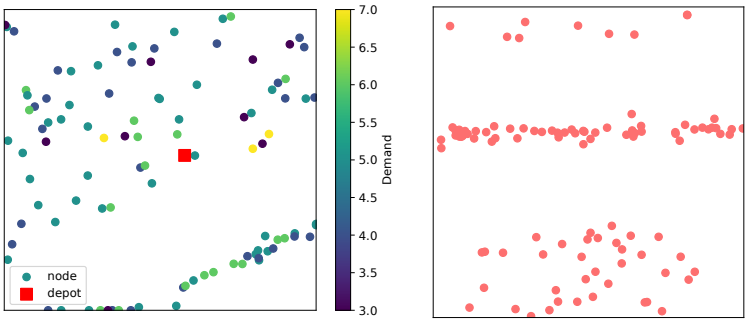


(a) CVRP (b) TSP

Figure 13: Example problem instance of Task P^E

Compression (C). Nodes are first sampled uniformly from $[0, 1]^2$. We then sample two points $p_1, p_2 \sim \mathcal{U}([0, 1]^2)$ and use them to define a line l. For each node whose distance to l is below 0.3, a new signed distance is sampled from $\mathcal{N}(0, 0.1^2)$, and the node is moved along the direction

perpendicular to l so that its new distance to the line matches the sampled value. The final coordinates are clipped to $[0, 1]^2$. For CVRP, the demand is generated as $10 - x$, where $x \sim \mathcal{N}(5, 1)$, followed by rounding and clipping to $[0, 10]$.



(a) CVRP (b) TSP

Figure 14: Example problem instance of Task P^C

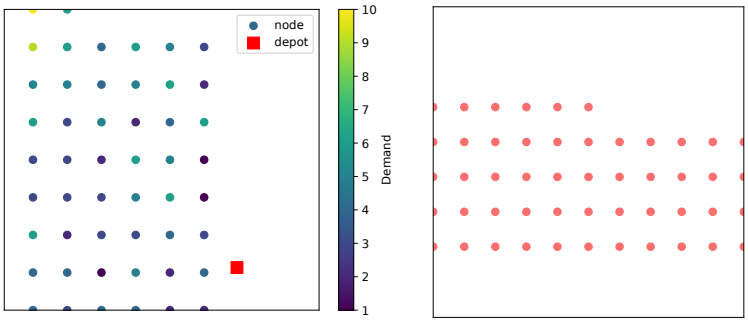
Grid (G). We first sample an aspect ratio parameter $\rho \sim \mathcal{U}(0.2, 0.8)$. With probability 0.5, we set the rectangle width and height to $(w, h) = (1, \rho)$; otherwise, we set $(w, h) = (\rho, 1)$. The center of the rectangle is sampled as

$$c_x \sim \mathcal{U}\left(\frac{w}{2}, 1 - \frac{w}{2}\right), \qquad c_y \sim \mathcal{U}\left(\frac{h}{2}, 1 - \frac{h}{2}\right).$$

A grid is then constructed inside this rectangle. Following the original benchmark protocol (Pei et al., 2026), the number of grid cells along the x- and y-axes is set to

$$a = \left\lceil \sqrt{n \frac{w}{h}} \right\rceil, \qquad b = \left\lceil \sqrt{\frac{n}{a}} \right\rceil.$$

Nodes are placed into the grid cells until the required number of customers is reached. If the grid cannot be evenly filled, the remaining empty cells are chosen from the largest-x cells in the row with the largest y value. For CVRP, the demand of each customer is computed from its depot distance with additive noise sampled from $\mathcal{U}(0, 1)$. The noisy distances are linearly mapped to $[1, 10]$ and then rounded.



(a) CVRP (b) TSP

Figure 15: Example problem instance of Task P^G

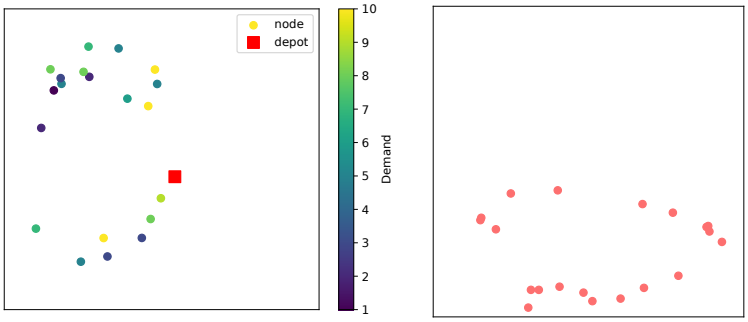
Ring (R). For each node, we sample an angle $\theta \sim \mathcal{U}(0, 2\pi)$ and a radius

$$r = r_1 + r_2, \qquad r_1 \sim \mathcal{U}(0.3, 0.4), \quad r_2 \sim \mathcal{N}(0, 0.05^2).$$

The preliminary coordinate is then given by

$$(x, y) = (0.5 + r\cos\theta,\ 0.5 + r\sin\theta).$$

Following (Pei et al., 2026), an additional anisotropic scaling is applied to one coordinate axis using a random scaling factor, producing horizontally or vertically compressed ring-like structures. For CVRP, customer demands are derived from the depot distance with additive noise sampled from $\mathcal{U}(0, 2)$. The noisy distances are mapped to $[1, 10]$ and rounded.



(a) CVRP (b) TSP

Figure 16: Example problem instance of Task P^R

Task Orders. For both TSP and CVRP, we use the five random task orders from (Pei et al., 2026), as follows.

- Order 1: E → C → G → U → R → GM.
- Order 2: U → GM → E → R → G → C.
- Order 3: E → G → R → C → U → GM.
- Order 4: G → GM → E → U → R → C.
- Order 5: G → C → R → U → GM → E.

Task Sequences with Continual Drift In the continually drifting scenario, we use an epoch as the time step unit. Drift occurs only between epochs, and the task is assumed to be stationary within each epoch. To construct such a stream of tasks, we linearly mix up principal tasks. We assign $K = 6$ predefined principal tasks $\{P^i\}_{i=1}^K$ evenly to K epochs with an interval of $m = \frac{T}{K-1} = \frac{1000}{6-1} = 200$ epochs, i.e., to epochs 0, m, $2m$, ..., T. The task of an epoch between the epochs of two principal tasks is a linear mixture of them. Specifically, at an epoch $t \in \{im + 1, \ldots, (i + 1)m - 1\}$, we generate an intermediate task P_t with scale $S_t = \left\lceil \frac{(i+1)m-t}{m} S^i + \frac{t-im}{m} S^{i+1} \right\rfloor$, with S^i and S^{i+1} as the problem scales of P^i and P^{i+1}, respectively. For an instance of P_t, $\left\lceil \frac{t-im}{m} S_t \right\rfloor$ nodes are generated with the distribution of P^{i+1} while the rest are generated with the distribution of P^i.

Notably, problem scale and node coordinate distribution change between tasks for both CVRP and TSP. For CVRP, the node demand will also change. The coordinate of the depot for CVRP remains unchanged for all tasks.

Since VRP instances are discrete and combinatorial and their natural unit of change is a node, epoch-level granularity is sufficient for our continual drift design. Specifically, with principal problem scales of 20, 50, and 100 and 200 epochs between consecutive principal tasks, a change of one node in scale (or, equivalently, a one-node deviation from a principal distribution) corresponds to roughly two epochs. Hence, treating each epoch as a time step provides adequate resolution, and using finer-grained units would not significantly affect the outcomes.

D.2 Training and Test Settings

Following (Pei et al., 2026), we assume the problem instance generation is uncontrollable. For all methods, during training, we use data augmentation with an augmentation factor of 8, following common settings (Kwon et al., 2020; Fang et al., 2024). During testing, augmentation is not used.

With 1000 epochs and 128 incoming batches per epoch, DREE with $|\mathcal{B}| = 256$ retains about 0.2% of the incoming instance batches. Each retained entry also contains objective values and behavior trajectories. Appendix A.3 accounts for their storage.

For comparisons across task orders, we test on the six principal tasks. This choice is necessary because the intermediate tasks induced by continual drift can vary across different task orders, whereas the six principal tasks always appear regardless of the order. Testing these shared tasks, therefore, enables a fair and comprehensive comparison across orders. Since the principal tasks are uniformly distributed over the learning horizon, performance on them is also representative of overall lifelong-learning behavior. We also conduct an additional comparison on task order 1 with finer-grained test set generation, and the results lead to the same conclusion on the six-principal-task test setting (cf. Appendix G.1), suggesting the effectiveness and efficiency of the setting.

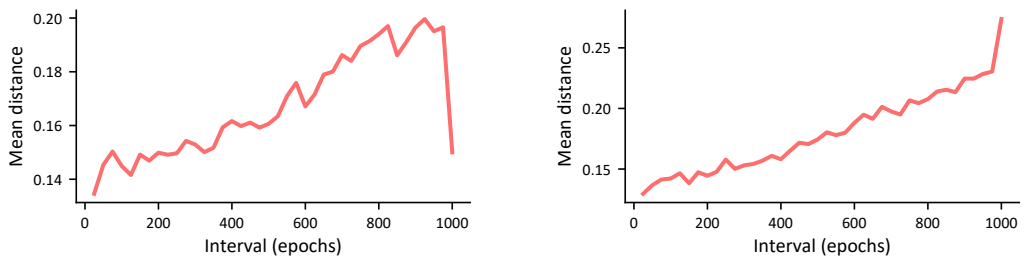
For testing, we report the optimality gap. For each test instance, we obtain an reference solution using HGS for CVRP and Gurobi for TSP, compute the gap per instance, and then average gaps over the test set. All task orders share the same test-instance sets. As for each task order there are 1000 possible models from the 1000 epochs to test, and each of them needs to be tested on all six principal tasks, it would be very expensive to test all of them. Thus, we select the model for every 20 epochs to test, which still gives a comprehensive description of solvers' performance changing during the lifelong learning process.

DREE trains on 128 batches of new-task instances per epoch. We record the number of PIR steps and find that, across different task orders, it is approximately 36 (avg. $N = \frac{128}{36} \simeq 3.56$). Therefore, for other methods (i.e., *fine-tuning*, LLR-BC, and *Li*), we run $128 + 36 = 164$ batches per epoch so that they execute the same number of solving episodes, while processing even more training instances. This matches the average number of complete solution-construction episodes. The forward workload and measured training time are reported separately in Appendix A.3.

Following (Pei et al., 2026), we use representative subsets of TSPLIB and CVRPLIB (Set-X), as listed in Table 15 and Table 16.

D.3 GW Analysis of Synthetic Drift

We apply the GW analysis used for LaDe to the synthetic TSP and CVRP streams, sampling the instance stream every 25 epochs. Figure 17 shows an overall increasing trend in mean GW distance with temporal lag, measured in epochs, for both problems. The curves exhibit local fluctuations, and the CVRP curve drops at the largest lag. This trend is consistent with the temporal spatial drift observed in LaDe. The synthetic streams provide controlled changes in scale and node distribution, while the logistics-derived streams complement them with naturally occurring temporal variation.



(a) CVRP (b) TSP

Figure 17: Mean GW distance versus temporal lag (epochs) for synthetic streams sampled every 25 epochs.

E More Experiment Results on Real-world Dataset

Table 11 reports the detailed performance of all methods on each region. Here, *Gen* denotes the zero-shot generalization performance on the held-out instances from the last 5% of valid days after training on the instances from the first 95% of days. DREE achieves the smallest AP and ABPl among the compared lifelong learning methods in three regions, namely `hz-120`, `jl-31`, and `yt-167`. Its AP is also lower than that of MT-Ref on `hz-120` and `jl-31`. DREE obtains the lowest Gen among the lifelong learning methods on `jl-31` and `yt-167`, and the lowest mean Gen across all five regions (Table 4). These results support its effectiveness on the evaluated logistics-derived task streams, with advantages that vary by region.

Table 11: Detailed metric values on the real-world problem instance stream of LaDe. *Gen* represents the average test performance on test instances from the held-out final 5% of days.

Region	Method	AP	AFB	AMFB	ABPl	Gen
sh-2	MT-Ref	0.59	-	-	-	0.85
	FT	1.83	1.56	4.38	0.27	0.82
	Li	9.11	0.23	1.75	8.88	11.18
	LLR-BC	0.59	0.48	0.75	0.11	0.43
	DREE	0.64	0.41	0.68	0.23	0.79
hz-120	MT-Ref	1.64	-	-	-	1.68
	FT	0.32	0.2	0.78	0.12	0.28
	Li	3.75	0.37	0.49	3.38	5.42
	LLR-BC	0.46	0.34	0.63	0.11	0.37
	DREE	0.27	0.21	0.34	0.06	0.4
jl-31	MT-Ref	0.54	-	-	-	2.11
	FT	2.71	2.15	5.03	0.55	2.45
	Li	12.4	7.46	15.86	4.95	6.07
	LLR-BC	0.58	0.42	1.78	0.16	2.82
	DREE	0.49	0.39	0.58	0.09	1.7
yt-167	MT-Ref	1.36	-	-	-	1.23
	FT	8.56	6.75	15.42	1.81	4.6
	Li	6.1	0.82	2.89	5.28	10.5
	LLR-BC	3.86	3.07	5.53	0.79	2.97
	DREE	2.48	1.82	3.94	0.66	1.58
cq-144	MT-Ref	1.16	-	-	-	1.32
	FT	9.74	8.41	38.88	1.32	1.32
	Li	18.71	13.45	20.58	5.26	30.01
	LLR-BC	1.20	0.83	4.27	0.37	1.05
	DREE	1.26	0.80	4.75	0.46	2.15

Amazon. Table 12 presents the station-level results for the four Amazon CVRP streams, using the same metric columns as Table 11. *Gen* denotes performance on the held-out final 5% of valid days.

F More Experiment Results on Generated Dataset

F.1 Performance on Seen Tasks

Table 13 lists the detailed metric values of each method on each task order on the synthetic dataset. DREE is less sensitive to the task order, leading to the most stable metric values. Furthermore, DREE outperforms other methods in terms of each metric (excluding AFB and AMFB compared with Li) in most of the cases, especially the AP value.

Table 12: Detailed metric values for each Amazon station stream. A dash denotes an inapplicable metric. *Gen* represents the average test performance on test instances from the held-out final 5% of days.

Station	Method	AP	AFB	AMFB	ABPl	Gen
DBO2	MT-Ref	2.33	–	–	–	2.61
	Fine-tuning	2.57	0.90	2.78	1.67	3.11
	Li	3.40	0.43	1.58	2.96	4.02
	LLR-BC	2.37	0.42	0.81	1.95	2.43
	DREE	2.38	0.46	1.22	1.92	2.40
DBO3	MT-Ref	2.92	–	–	–	3.03
	Fine-tuning	3.04	1.70	3.11	1.34	3.98
	Li	5.25	0.50	1.03	4.76	7.12
	LLR-BC	2.28	0.31	0.82	1.97	3.01
	DREE	2.26	0.23	0.31	2.03	2.99
DSE4	MT-Ref	1.77	–	–	–	1.79
	Fine-tuning	1.92	1.21	5.76	0.71	2.21
	Li	6.03	1.07	1.56	4.95	5.98
	LLR-BC	1.37	0.32	0.69	1.06	2.04
	DREE	1.04	0.11	0.11	0.93	1.75
DSE5	MT-Ref	1.29	–	–	–	1.53
	Fine-tuning	1.49	0.49	2.02	1.01	1.66
	Li	1.96	0.58	1.40	1.38	2.23
	LLR-BC	1.69	0.64	0.97	1.05	1.64
	DREE	1.40	0.34	1.29	1.07	1.59

F.2 Generalization on Unseen Benchmark Instances

Table 14 summarizes generalization across the five task orders. DREE achieves lower mean gaps than the compared lifelong learning methods on both benchmarks. The detailed instance-level results in Tables 15 and 16 correspond to task order 1. For that order, DREE obtains a CVRPLIB gap of 13.78%, compared with 12.92% for MT-Ref, and a TSPLIB gap of 18.06%, compared with 19.86% for MT-Ref.

We use the same representative instance sets as those selected in (Pei et al., 2026; Zhou et al., 2023) from CVRPLIB and TSPLIB. We test the solver obtained by learning from task order 1 of the synthetic CVRP and TSP dataset on CVRPLIB and TSPLIB instances, respectively. Tables 15 and 16 report detailed results, respectively. Gaps are computed based on the reported best-known solutions.

F.3 More Details of Applicability Experiment

We implement *fine-tuning*, the most widely used strategy, LLR-BC, the state-of-the-art lifelong learning solver, and DREE on other existing cross-distribution cross-scale neural solvers, INViT (Fang et al., 2024) and Omni (Zhou et al., 2023) based on their official code. Omni is designed to produce a strong initial model that can quickly adapt to new tasks, through meta-learning. Accordingly, we adopt the meta-learned initial model provided by (Zhou et al., 2023) as the starting model for lifelong learning with 16 batches per epoch, and a total of 1000 epochs as in the setting of experiments based on POMO. For INViT, we use the same setting as the experiments with POMO as the base neural solver, i.e., 1000 epochs with 128 batches per epoch for DREE and 164 batches per epoch for other lifelong learning methods. Hyperparameters of the compared methods are set identically to those used on POMO. Hyperparameters of base neural solvers are set identically to those in the original papers. Notably, we focus on comparing different lifelong learning solvers built upon the same base solver, rather than comparing across different base solvers. Results obtained with different base solvers are not directly comparable, since the training setups are different.

Table 13: Detailed metric values on each task order of the synthetic dataset.

Order	Method	CVRP				TSP			
		AP	AFB	AMFB	ABPl	AP	AFB	AMFB	ABPl
1	Fine-tuning	14.05	10.35	21.85	3.7	1.86	0.75	1.59	1.11
	Li	6.12	0.07	0.07	6.06	3.64	0.01	0.01	3.63
	LLR-BC	4.5	1.03	1.99	3.48	1.19	0.22	0.64	0.97
	DREE	3.16	0.22	0.42	2.94	1.12	0.1	0.29	1.02
2	Fine-tuning	5.4	2.6	8.37	2.79	1.59	0.83	1.31	0.76
	Li	5.62	0.07	0.08	5.56	4.18	0.01	0.01	4.17
	LLR-BC	3.75	0.91	1.55	2.84	1.18	0.4	0.51	0.83
	DREE	3.21	0.5	1.03	2.71	1.03	0.21	0.22	0.82
3	Fine-tuning	14.17	10.93	18.93	3.25	1.8	0.92	1.29	0.88
	Li	6.21	0.02	0.03	6.19	3.34	0.01	0.02	3.33
	LLR-BC	4.32	1.35	1.74	2.97	1.15	0.31	0.40	0.84
	DREE	3.24	0.24	0.47	3.00	1.02	0.17	0.19	0.85
4	Fine-tuning	5.78	2.56	8.30	3.23	2.16	0.96	1.96	1.2
	Li	6.04	0.19	0.20	5.85	5.09	0.02	0.12	5.07
	LLR-BC	4.27	1.20	1.68	3.07	1.45	0.34	0.69	1.11
	DREE	2.87	0.10	0.10	2.77	1.08	0.12	0.16	0.96
5	Fine-tuning	4.34	0.91	21.28	3.43	2.44	0.86	2.06	1.58
	Li	6.8	0.12	0.12	6.69	5.95	0.03	0.11	5.92
	LLR-BC	4.13	0.71	1.45	3.42	1.94	0.41	0.99	1.52
	DREE	3.09	0.10	0.10	2.99	1.72	0.22	0.28	1.50

Table 14: Mean (std.) gap (%) on unseen benchmarks across five task orders. MT-Ref is trained once and its parentheses summarize variation across benchmark instances.

Method	CVRPLIB	TSPLIB
MT-Ref	12.92 (4.90)	19.86 (10.69)
Fine-tuning	13.16 (5.36)	21.85 (9.65)
Li	19.14 (6.62)	27.87 (12.21)
LLR-BC	15.76 (10.65)	20.90 (10.98)
DREE	**12.69** (6.39)	**18.08** (9.81)

Table 15: Test results on each included CVRPLIB instance.

Instance	MT-Ref		Fine-tuning		Li		LLR-BC		**DREE**	
	Distance	Gap (%)	Distance	Gap (%)	Distance	Gap (%)	Distance	Gap (%)	Distance	Gap (%)
X-n1001-k43	86266.88	19.23	90196.17	24.66	91417.62	26.35	94755.37	30.96	89635.44	23.88
X-n101-k25	29988.91	8.69	30540.37	10.69	30926.36	12.09	29478.07	6.84	29539.54	7.06
X-n153-k22	24036.21	13.27	24083.69	13.5	25059.6	18.09	24751.04	16.64	23686.61	11.62
X-n200-k36	62479.91	6.66	62848.84	7.29	65328.33	11.52	62325.69	6.4	62200.68	6.18
X-n251-k28	41563.26	7.44	42712.81	10.41	43884.06	13.44	44103.78	14.01	41718.06	7.84
X-n303-k21	24117.38	10.96	26153.8	20.32	27413.56	26.12	25996.89	19.6	24596.6	13.16
X-n351-k40	29544.25	14.09	30682.58	18.48	32419.11	25.19	33240.53	28.36	29324.47	13.24
X-n401-k29	70920.75	7.21	71970.64	8.79	77196.08	16.69	81340.6	22.96	70874.54	7.14
X-n459-k26	28831.45	19.44	30027.02	24.39	32952.42	36.51	30024.71	24.38	28680.1	18.81
X-n502-k39	75043.78	8.4	74583.1	7.74	77735.05	12.29	84633.9	22.26	73331.75	5.93
X-n548-k50	95298.19	9.92	96827.34	11.68	101873.6	17.5	112970.4	30.3	95291.28	9.91
X-n599-k92	121070.5	11.64	125330.2	15.56	138531.8	27.74	132365.1	22.05	121676.1	12.19
X-n655-k131	118827	11.28	115713.6	8.37	124589	16.68	145577.2	36.33	115984.8	8.62
X-n701-k44	93125.35	13.67	96309.91	17.56	95707.08	16.83	96237.26	17.47	92692.63	13.15
X-n749-k98	87428.02	13.15	91824.92	18.84	100277.4	29.78	100074	29.51	89567.84	15.92
X-n801-k40	85760.6	16.98	87867.68	19.86	89375.21	21.91	179235.1	144.49	84788.5	15.66
X-n856-k95	99516.53	11.86	107203	20.5	110038.4	23.69	151211.4	69.97	105347.8	18.41
X-n895-k37	68330.28	26.87	69874.6	29.73	70757.8	31.37	97394.1	80.83	67637.25	25.58
X-n957-k87	97976.01	14.64	104127.2	21.84	109240.4	27.82	130347.2	52.52	108971.7	27.5

Table 16: Test results on each included TSPLIB instance.

Instance	MT-Ref		Fine-tuning		Li		LLR-BC		**DREE**	
	Distance	Gap (%)	Distance	Gap (%)	Distance	Gap (%)	Distance	Gap (%)	Distance	Gap (%)
a280	3042.01	17.95	3067.16	18.93	3360.59	30.31	3127.75	21.28	2944.83	14.18
d493	51872.83	48.2	48369.09	38.19	55392.14	58.25	48362.88	38.17	48186.50	37.67
d657	63991.38	30.83	64949.99	32.79	69454.31	42	64624.70	32.12	64341.84	31.55
fl417	13726.64	15.73	13643.5	15.03	13696.16	15.47	13384.69	12.85	13487.66	13.71
kroA100	22350.05	5.02	22719.97	6.76	24029.82	12.91	22484.86	5.65	22667.00	6.51
kroA150	28621.91	7.91	28467.85	7.33	30349.84	14.42	28174.27	6.22	28293.38	6.67
kroA200	32378.11	10.25	32931.33	12.13	34288.3	16.75	32707.89	11.37	32305.86	10.00
kroB200	33546.88	13.96	33121.93	12.52	35114.09	19.29	33433.37	13.58	32428.63	10.16
lin318	49056.6	16.72	49268.01	17.22	52593.37	25.14	49528.87	17.84	48400.65	15.16
p654	42974.79	24.05	41371.89	19.42	45467.48	31.25	41292.63	19.19	40765.16	17.67
pcb442	61088.63	20.31	61423.84	20.97	69073.19	36.03	62043.08	22.18	62127.62	22.35
pr1002	353243.6	36.36	347019.3	33.96	390824.7	50.87	353517.23	36.47	345387.59	33.33
pr226	83418.01	3.79	83694.54	4.14	86377.18	7.48	83733.49	4.19	84997.44	5.76
pr264	54771.85	11.47	53165.7	8.2	62884.41	27.98	53237.98	8.35	52755.43	7.37
pr299	57926.24	20.2	58615.6	21.63	62212.08	29.09	57044.84	18.37	56933.31	18.14
pr439	134547.6	25.49	133522.1	24.53	147491.2	37.56	133519.80	24.53	131273.62	22.44
rat575	8550.87	26.25	8479.65	25.2	9702.2	43.25	8347.80	23.25	8518.76	25.78
rat783	11577.88	31.48	11608.1	31.82	13347.75	51.58	11389.70	29.34	11447.09	29.99
rd400	18396.2	20.39	18363.85	20.17	19687.65	28.84	18250.00	19.43	18235.69	19.34
ts225	134234.4	5.99	137818.5	8.82	146137.8	15.39	136657.53	7.91	132803.86	4.86
tsp225	4360.97	11.36	4492.83	14.73	4593.41	17.3	4343.73	10.92	4422.60	12.94
u574	46249.28	25.32	48140.74	30.45	51238.19	38.84	46493.46	25.98	45916.29	24.42
u724	53539.75	27.75	54104.91	29.1	60594.15	44.58	53172.20	26.87	52508.60	25.29

CoEKS. We evaluate fine-tuning, LLR-BC, and DREE on CoEKS (Yu et al., 2026), a multitask CVRP solver, using the same initialization and lifelong stream within this comparison. Table 17 reports the results for synthetic task order 1. DREE has the lowest final AP and AFB. LLR-BC has lower AMFB, and both compared methods attain lower ABPl, showing that the advantage in final performance coexists with different learning trajectories.

Table 17: Lifelong learning on CoEKS for CVRP, synthetic task order 1.

Method	AP	AFB	AMFB	ABPl
Fine-tuning	5.32	3.29	7.54	**2.03**
LLR-BC	3.51	0.89	**0.94**	2.62
DREE	**3.39**	**0.73**	1.02	2.66

DREE-Li. DREE-Li replaces DREE's behavior consolidation with Li's model-buffering and distillation mechanism (Li et al., 2025a). A retained reference model supplies policy targets for consolidation,

while problem instance replay provides learning on buffered instances. This implementation evaluates whether the framework can work with model-based knowledge preservation. Table 18 reports its results on synthetic task order 1. DREE-Li improves AP and ABPl relative to Li on both problems, with higher AFB and AMFB. The default DREE implementation achieves lower AP and ABPl than DREE-Li.

Table 18: DREE with different knowledge-consolidation mechanisms, synthetic task order 1.

Method	CVRP				TSP			
	AP	AFB	AMFB	ABPl	AP	AFB	AMFB	ABPl
Li	6.12	**0.07**	**0.07**	6.06	3.64	**0.01**	**0.01**	3.63
DREE-Li	3.57	0.16	0.22	3.42	1.40	0.10	0.13	1.30
DREE	**3.16**	0.22	0.42	**2.94**	**1.12**	0.10	0.29	**1.02**

F.4 More Details of Ablation Study

The ablation variant *w.o.PIR* sets $\mathcal{L}_{\mathrm{PIR}}$ to 0 for all epochs throughout lifelong learning. We still perform PIR so that EE can be executed. The ablation variant *w.o.KR* removes the KR module from DREE. Consequently, EE no longer influences learning, since the enhanced behaviors are not used. The ablation variant *w.o.EE* disables behavior replacement after PIR. This modification affects the KR module (as the quality of buffered behaviors is no longer improved) but does not impact the PIR module. For all ablation variants, we fix $N \leftarrow 3.56$. Because different ablations have different learning capacities, they may converge to different expected values of M^+. Fixing N ensures that all ablation variants perform the same number of PIR executions, enabling a fairer comparison.

Table 19 reports the results on task order 1.

Table 19: Results of ablation study.

Method	CVRP				TSP			
	AP	AFB	AMFB	ABPl	AP	AFB	AMFB	ABPl
w.o.PIR	4.07	0.81	2.23	3.25	1.38	0.28	0.42	1.10
w.o.KR	3.87	0.53	1.70	3.33	1.35	0.29	0.48	1.06
w.o.EE	3.47	0.32	0.49	3.16	1.20	0.17	0.42	1.03
DREE	**3.16**	**0.22**	**0.42**	**2.94**	**1.12**	**0.10**	**0.29**	**1.02**

F.5 More Details of Hyperparameter Analysis

Table 20: Metrics of DREE in different settings, on task order 1.

Method	CVRP				Method	TSP			
	AP	AFB	AMFB	ABPl		AP	AFB	AMFB	ABPl
$\alpha(50)$	3.66	0.32	1.48	3.34	$\alpha(5)$	1.16	0.15	0.34	1.00
$\alpha(200)$	3.25	0.28	0.36	2.97	$\alpha(20)$	1.09	0.08	0.22	1.00
$\beta(0.5)$	3.27	0.23	0.79	3.04	$\beta(0.5)$	1.13	0.14	0.34	0.99
$\beta(2)$	3.13	0.16	0.30	2.97	$\beta(2)$	1.21	0.08	0.21	1.13
$\lvert\mathcal{B}\rvert(128)$	3.19	0.27	0.43	2.92	$\lvert\mathcal{B}\rvert(128)$	1.14	0.12	0.31	1.02
$\lvert\mathcal{B}\rvert(64)$	3.24	0.29	0.35	2.96	$\lvert\mathcal{B}\rvert(64)$	1.16	0.14	0.30	1.02
$\mathrm{UB}(8)$	3.30	0.24	0.63	3.06	$\mathrm{UB}(8)$	1.18	0.13	0.34	1.05
$\mathrm{LB}(2)$	3.14	0.24	0.38	2.90	$\mathrm{LB}(2)$	1.11	0.10	0.26	1.01
DREE	3.16	0.22	0.42	2.94	**DREE**	1.12	0.10	0.29	1.02

A larger $|\mathcal{B}|$ increases persistent storage, while a smaller N (with its range defined by UB and LB) increases PIR computation. The number of entries sampled per replay operation remains fixed. We

evaluate variants with smaller $|\mathcal{B}|$, larger UB, or larger LB to examine these resource–performance trade-offs (Appendix A.3).

As the results listed in Table 20 show, different hyperparameter settings lead to different performance. In terms of AP, $\beta(2)$ and $\alpha(20)$ achieve the best results among the tested settings for CVRP and TSP, respectively. However, compared with the performance of existing methods (cf. Table 1), the performance changes due to different hyperparameter settings are slight, indicating the hyperparameter robustness of DREE.

G Further Experiments on More Settings

G.1 Finer-Grained Test

Instead of testing only on the six principal tasks at epochs 0, 200, 400, 600, 800, and 1000, we additionally conduct tests on 41 tasks by sampling one task every 25 epochs along the 1000-task stream, with task order 1. It provides a finer-grained test and more comprehensively reflects the performance changes during the lifelong learning process. Table 21 reports the results. Forgetting values vary with evaluation granularity, while the relative AP rankings remain consistent on both problems. The relative performance patterns across methods are consistent with those observed under the original test protocol, and the comparative conclusions remain the same.

It supports the validity of the original principal-task evaluation. The 6 principal tasks provide an efficient evaluation protocol that faithfully reflects both absolute method performance trends and the relative comparisons among methods.

Table 21: Performance on task order 1 with a finer-grained test set.

Method	CVRP				TSP			
	AP	AFB	AMFB	ABPl	AP	AFB	AMFB	ABPl
Fine-tuning	21.97	18.12	30.46	3.85	2.29	0.85	1.54	1.44
Li	7.67	**0.06**	**0.07**	7.60	4.69	**0.01**	**0.01**	4.68
LLR-BC	3.96	0.80	1.12	3.16	1.68	0.28	0.65	**1.40**
DREE	**3.10**	0.22	0.33	**2.88**	**1.55**	0.15	0.26	**1.40**

G.2 LLR-BC with Different Buffering Frequencies

Our adaptation of LLR-BC is consistent with its original design and provides a fair comparison. LLR-BC stores buffered experiences at the final epoch of each stationary task, regardless of the number of epochs of the task. In the gradually drifting scenario, each epoch corresponds to a unique stationary task. Therefore, LLR-BC buffers at every epoch.

Nevertheless, we conduct additional experiments to analyze the effect of buffer frequency on the performance of LLR-BC. We further evaluate two LLR-BC variants that buffer every 10 epochs (denoted as LLR-BC(10)) and every 100 epochs (denoted as LLR-BC(100)), respectively, on task order 1 of the synthetic dataset. Table 22 lists the results. Compared with the performance of DREE, the impact of buffering frequency in LLR-BC is negligible.

Table 22: Performance of LLR-BC with different buffering frequencies in units of epochs.

Method	CVRP				TSP			
	AP	AFB	AMFB	ABPl	AP	AFB	AMFB	ABPl
LLR-BC(10)	4.62	1.19	2.01	3.43	1.17	0.10	0.22	1.07
LLR-BC(100)	4.69	1.16	1.78	3.53	1.19	0.08	0.16	1.10
LLR-BC	4.50	1.03	1.99	3.48	1.19	0.22	0.64	0.97

G.3 Cumulative Multi-Tasking Training

MT-cu retrains from scratch every 200 epochs with access to all seen instances, with 200 epochs per retraining for a comparable budget to one lifelong learning run. At 1000 epochs, MT-cu retains 128,000 incoming instance batches, which is 500 times DREE's default buffer-entry count of 256. The byte-level storage comparison also accounts for DREE's behavior trajectories and objective values (Appendix A.3). We run MT-cu on task order 1 with results listed in Table 23. MT-cu performs substantially worse than DREE, highlighting both the efficiency of DREE and the necessity of dedicated lifelong learning methods.

Table 23: Comparison with MT-cu.

Method	CVRP				TSP			
	AP	AFB	AMFB	ABPI	AP	AFB	AMFB	ABPI
MT-cu	4.49	0.93	5.25	3.56	1.25	0.04	4.43	1.21
DREE	**3.16**	**0.22**	**0.42**	**2.94**	**1.12**	**0.10**	**0.29**	**1.02**

G.4 Continual Drift with Sudden Task changes

The synthetic dataset for the continually drifting scenario only captures smooth task transitions. Although we have already evaluated DREE under non-smooth task changes, including the Periodically Stationary scenario with abrupt shifts and real-world data exhibiting overall continual drift with substantial fluctuations, we further construct an additional variant of the continually drifting scenario that explicitly incorporates abrupt changes. This allows us to more directly assess the robustness of DREE under continual drift with sudden distributional shifts. Specifically, with task order 1, we swapped the tasks of three randomly selected pairs of epoch intervals ((23, 43)–(731, 751), (112, 162)–(517, 567), and (278, 378)–(810, 910)). For instance, the task at epoch 23 is the task originally corresponding to epoch 731. Thus, the new scenario contains abrupt task changes at the boundaries of the swapped intervals, and also exhibits recurrence of old principal-task patterns. We then run all lifelong learning methods on it, and Table 3 lists the result. DREE outperforms all competing lifelong learning methods, with performance trends consistent with the smooth setting, further showing that DREE remains robust under abrupt changes and task recurrences.

H Potential Limitations

DREE treats all tasks and stored experiences uniformly, and thus may underperform fine-tuning when task difficulty varies substantially, especially when later tasks are consistently more difficult and require greater adaptation. DREE relies on storing problem instances as experiences, making it unsuitable for settings where such instances cannot be retained due to data security, privacy, confidentiality, or regulatory constraints. DREE assumes a shared representation dimensionality across tasks, and is therefore not directly applicable when different tasks require different state feature dimensions, such as those induced by additional constraints. This limitation may be addressed by combining DREE with a base solver that supports variable-dimensional representations, which we leave for future work. Nevertheless, DREE remains broadly applicable and can achieve strong performance when the scenario assumptions are satisfied.

The PIR scheduler allocates computation using the observed refinement yield. A low yield can occur when historical instances are already well learned or when substantial forgetting makes their buffered solutions difficult to improve. In the latter case, reducing PIR frequency may delay recovery. KR continues to consolidate historical behaviors, and the bounded interval maintains periodic instance replay. Incorporating a retention-sensitive signal into replay allocation is a direction for future work.