DIFFUSION-DRIVEN LATENT GUIDED SAMPLING FOR NEURAL COMBINATORIAL OPTIMIZATION

Sobihan Surendran[1,2*] **Adeline Fermanian**[2] **Sylvain Le Corff**[1]
[1]Sorbonne Université and Université Paris Cité, CNRS,
Laboratoire de Probabilités, Statistique et Modélisation, F-75005 Paris, France
[2]LOPF, Califrais' Machine Learning Lab, Paris, France

ABSTRACT

Combinatorial Optimization problems are widespread in domains such as logistics, manufacturing, and drug discovery, yet their NP-hard nature makes them computationally challenging. Recent Neural Combinatorial Optimization (NCO) methods leverage deep learning to learn policies for constructing solutions, trained via Supervised or Reinforcement Learning. While promising, these approaches often rely on task-specific augmentations, perform poorly on out-of-distribution instances, and lack robust inference mechanisms. Moreover, existing latent space models either require labeled data or use an instance-independent latent distribution. In this work, we introduce LGS-Net, an instance-conditioned latent space model that enables sampling-based inference beyond standard decoding methods. Leveraging this representation, we propose two diffusion-driven inference schemes: a diffusion-prior Markov Chain Monte Carlo and a diffusion-guided Sequential Monte Carlo, both coupled with Stochastic Approximation for test-time adaptation toward low-cost solutions. Empirical results on benchmark routing tasks show that our method achieves state-of-the-art performance among NCO baselines.

1 INTRODUCTION

Combinatorial Optimization (CO) consists of finding the best solution from a discrete set of possibilities by optimizing a given objective function subject to constraints. It has widespread applications across various domains, including vehicle routing (Veres & Moussa, 2019), production planning (Dolgui et al., 2019), and drug discovery (Liu et al., 2017). However, its NP-hard nature and the complexity of many problem variants make solving CO problems highly challenging. Traditional heuristic methods (e.g., (Kirkpatrick et al., 1983; Glover, 1989; Mladenović & Hansen, 1997)) rely on hand-crafted rules to guide the search, providing near-optimal solutions with significantly lower computational costs. Inspired by the success of deep learning in computer vision (Krizhevsky et al., 2012; He et al., 2016) and natural language processing (Vaswani et al., 2017; Devlin et al., 2019), recent years have seen a surge in Neural Combinatorial Optimization (NCO) approaches for solving CO problems, including the Travelling Salesman Problem (TSP) and the Capacitated Vehicle Routing Problem (CVRP).

These methods leverage neural networks to learn a policy that generates solutions, trained via either Supervised Learning (SL) (Vinyals et al., 2015; Joshi et al., 2019; Hottung et al., 2021; Fu et al., 2021; Joshi et al., 2022; Kool et al., 2022) or Reinforcement Learning (RL) (Bello et al., 2017; Nazari et al., 2018; Kool et al., 2019; Chen & Tian, 2019; Kwon et al., 2020; Hottung & Tierney, 2020; Grinsztajn et al., 2023; Chalumeau et al., 2023). SL-based methods often struggle to obtain sufficient high-quality labeled data, whereas RL-based approaches can surpass them by exploring solutions autonomously. Despite their success on in-distribution problem instances, these methods often generalize poorly to out-of-distribution cases, limiting their applicability in real-world scenarios. Moreover, once a policy is trained, inference typically relies on relatively simple strategies such as stochastic sampling (Kool et al., 2019; Kwon et al., 2020), beam search (Steinbiss et al., 1994), or Monte Carlo Tree Search (Browne et al., 2012). A popular alternative, based on online fine-tuning

*Corresponding author: sobihan.surendran@sorbonne-universite.fr

(Bello et al., 2017; Hottung et al., 2022), is to actively adapt the policy for each new problem instance. However, this approach introduces significant computational and practical challenges.

Meanwhile, latent space models have proven effective across diverse tasks, including image generation (Rombach et al., 2022), text generation (Bowman et al., 2016), anomaly detection (An & Cho, 2015), and molecule design (Gómez-Bombarelli et al., 2018). More recently, Hottung et al. (2021); Chalumeau et al. (2023) have explored learning a continuous latent space for discrete routing problems. Importantly, Chalumeau et al. (2023) have shown that RL-based latent methods can achieve competitive results without relying on the augmentation trick (Kwon et al., 2020), which is used in several prior inference methods. This augmentation strategy enhances performance by generating variations of the same problem, such as rotating the coordinates, but is task-specific and only applicable to certain routing problems in Euclidean space. However, existing latent space methods still have important limitations: the approach of Hottung et al. (2021) relies on labeled training data for SL, whereas Chalumeau et al. (2023) lacks instance-dependent structure in its latent space and therefore does not adapt to the specific problem instance. In contrast, we introduce a latent space model that conditions the latent representation on problem instances and is trained with RL, addressing these limitations and enabling more effective latent space optimization. We further propose two diffusion-driven inference schemes for latent-based models: a diffusion-prior Markov Chain Monte Carlo (MCMC) sampler and a diffusion-guided Sequential Monte Carlo (SMC) alternative, both coupled with Stochastic Approximation (SA) to adapt the decoder at test time and guide inference toward low-cost solutions. Empirically, this approach achieves state-of-the-art performance on benchmark routing tasks against strong NCO baselines.

More precisely, our contributions are summarized as follows.

- We introduce LGS-Net, a novel latent space model for Neural Combinatorial Optimization that does not require labeled data, learns a structured, instance-conditioned latent representation, and is supported by a rigorous mathematical framework.
- We propose a diffusion-driven inference framework with a diffusion-prior MCMC sampler as the main inference scheme, and a diffusion-guided SMC variant as an exploratory alternative, both coupled with Stochastic Approximation for cost-aware test-time adaptation.
- We empirically show that our inference method is effective in low-budget regimes, where gradient-based approaches tend to slow down. Furthermore, we establish that it achieves state-of-the-art performance among NCO methods, consistently outperforming existing techniques across diverse problem types, both with and without the augmentation trick.

2 Related Work

Neural Combinatorial Optimization. The application of neural networks to CO problems dates back to Hopfield & Tank (1985), who used a Hopfield network for small TSP instances. With advancements in deep learning, a major breakthrough came with Pointer Networks (Vinyals et al., 2015), inspired by sequence-to-sequence models (Sutskever et al., 2014). Pointer Networks enabled variable-length outputs but relied on supervised training and thus could not surpass the quality of the provided solutions. To address this limitation, Bello et al. (2017) adapted the model for RL. For the CVRP, Nazari et al. (2018) extended Pointer Networks with element-wise projections, followed by the Attention Model (AM) of Kool et al. (2019), based on transformers (Vaswani et al., 2017). Since then, numerous AM variants have been proposed (Kwon et al., 2020; Xin et al., 2020; Kim et al., 2021; Xin et al., 2021; Kwon et al., 2021; Hottung et al., 2025). For instance, Kwon et al. (2020) introduced POMO, an attention-based model that incorporates a more robust learning and inference strategy grounded in multiple optimal policies. In addition, graph neural network approaches have been explored, notably graph embeddings (Khalil et al., 2017), attention networks (Deudon et al., 2018), and convolutional networks (Joshi et al., 2019) for the TSP.

Several approaches combine heuristic algorithms, such as local search (Papadimitriou & Steiglitz, 1998; De Moura & Bjørner, 2008), with machine learning techniques to tackle routing problems. For example, Chen & Tian (2019); Lu et al. (2019) propose an RL-based improvement method that iteratively selects a region of the solution and applies a local heuristic determined by a trainable policy. To improve the generalization ability of constructive methods during inference, various strategies have been introduced, such as Efficient Active Search (EAS) (Hottung et al., 2022) and Simulation-guided

Beam Search (SGBS) (Choo et al., 2022). Notably, EAS builds on POMO by fine-tuning a subset of model parameters at inference time using Gradient Descent, in contrast to Active Search (Bello et al., 2017), which updates all model parameters. More recently, generalization-boosting methods have also been explored (Gao et al., 2024; Zhou et al., 2024). Recent state-of-the-art NCO methods include memory-enhanced approaches (Chalumeau et al., 2025) as well as latent space and search-based methods such as Chalumeau et al. (2023); Hottung et al. (2022).

Among the latent space models designed to map discrete routing problems to continuous spaces, Hottung et al. (2021) used a conditional variational autoencoder (CVAE) (Sohn et al., 2015) that maps solutions to a continuous latent space. However, their approach relies on supervised training, which is hindered by the significant cost of acquiring high-quality labeled data. To overcome this limitation, Chalumeau et al. (2023) proposed COMPASS, an RL-based approach that learns a latent space on top of a policy. However, the latent distribution in COMPASS lacks instance-dependent structure: it is defined via a fixed prior in latent space and does not adapt to the specific problem instance, making it conceptually similar to a GAN (Goodfellow et al., 2014) without a discriminator. In contrast, we introduce a latent space model that conditions the latent representation on problem instances and is trained with RL from cost feedback, without requiring labeled data. Our approach can be interpreted as a VAE with a modified encoder that conditions only on problem instances, rather than on both problem instances and solutions. Furthermore, we propose a diffusion-driven inference method in latent space, rather than using Differential Evolution (DE) (Storn & Price, 1997) or Covariance Matrix Adaptation (CMA) (Hansen & Ostermeier, 2001), which were used in previous works on latent space models.

Sampling Methods. The Metropolis–Hastings algorithm (Metropolis et al., 1953; Hastings, 1970) is a classical and widely used MCMC method. Numerous variants have been developed, including the Gibbs sampler (Geman & Geman, 1984) and Hamiltonian Monte Carlo (Duane et al., 1987). A crucial factor influencing the performance of MCMC methods is the choice of the proposal distribution, which significantly affects the convergence rate (Gelman et al., 1997). Traditionally, tuning the proposal distribution relies on heuristics and manual adjustments. To address this limitation, adaptive MCMC methods have been developed, where the proposal distribution is adjusted dynamically based on previous samples (Haario et al., 2001).

Beyond MCMC methods, several recent works consider Boltzmann sampling with non-differentiable reward functions (Fan et al., 2023; Uehara et al., 2024; Venkatraman et al., 2024), proposing procedures to fine-tune diffusion models to maximize potentially non-differentiable reward functions. This test-time adaptation perspective is closely related to Active Search (Bello et al., 2017), which fine-tunes policy parameters via RL, and to Efficient Active Search (Hottung et al., 2022), which updates only a subset of parameters and incorporates Imitation Learning so that a fraction of samples imitates the best solutions found. Finally, GFlowNets (Bengio et al., 2023) have been applied to vehicle routing problems, including approaches that jointly train a GFlowNet generator with an adversarial discriminator to tackle large-scale instances (Zhang et al., 2025).

3 Notation and Background

3.1 Notation

In the following, for all distribution μ (resp. probability density p) we write $\mathbb{E}_\mu$ (resp. $\mathbb{E}_p$) the expectation under μ (resp. under p). We may also write $\mathbb{E}_{x\sim\mu}$ for the expectation under μ. The Hadamard product of two vectors u and v is denoted by $u \odot v$. For a sequence $(a_m)_{m\in\mathbb{N}}$, and all $u \leq v$, we write $a_{u:v} = \{a_u, \dots, a_v\}$. Table 3 provides a summary of the notations used throughout the paper for ease of reference.

3.2 Problem Setting

In a Combinatorial Optimization problem, the objective is to determine the best assignment of discrete variables that satisfies the constraints of the problem. Let x represent a given problem instance and y denote a solution. An instance $x = \{\mathsf{x}_i\}_{i=1}^n \in \mathsf{X} \subset \mathbb{R}^{n\times d_x}$ consists of a set of n nodes, each represented by a feature vector $\mathsf{x}_i \in \mathbb{R}^{d_x}$, which encodes relevant information about the node. For a

given instance x, we aim to find a solution y^* that minimizes the associated cost function C:

$$y^* \in \operatorname*{argmin}_{y \in \mathsf{Y}} C(y, x) \ , \tag{1}$$

where Y denotes the discrete set of all feasible solutions for the given problem x. This setting covers problems such as the TSP, CVRP, Knapsack, and Job Scheduling. For instance, in TSP, x represents the coordinates of all nodes and Y consists of all possible node permutations. In CVRP, x additionally includes demands, and Y comprises all feasible routes satisfying the capacity constraints. In both cases, the cost corresponds to the cumulative distance of the route. Further details on both problems are provided in Appendix A.1.

3.3 Constructive NCO Methods

Constructive NCO methods (Vinyals et al., 2015; Bello et al., 2017; Nazari et al., 2018; Kool et al., 2019; Kwon et al., 2020) generate solutions sequentially using a stochastic policy $p_\theta(y|x)$, which defines the probability of selecting a solution y given a problem instance x. This policy is parameterized by $\theta \in \Theta$, where Θ is a parameter space, and factorized as:

$$p_\theta(y|x) = \prod_{t=1}^{T} p_\theta(\mathsf{y}_t|\mathsf{y}_{1:t-1}, x) \ ,$$

with the convention $p_\theta(\mathsf{y}_1|\mathsf{y}_{1:0}, x) = p_\theta(\mathsf{y}_1|x)$, where $\mathsf{y}_t \in \{0, \dots, n\}$ is the selected node at step t, $\mathsf{y}_{1:t-1}$ denotes the sequence of nodes selected up to step $t-1$, and T is the total number of decoding steps. Following Bello et al. (2017), writing $\mathbb{P}_x$ the distribution of the problem instances, the policy is trained via RL by minimizing an empirical estimate of the expected cost:

$$J(\theta) = \mathbb{E}_{x \sim \mathbb{P}_x, y \sim p_\theta(.|x)} \left[C(y, x) \right] \ .$$

where $C(y, x)$ denotes the tour cost. This objective is optimized using RL techniques, such as REINFORCE (Williams, 1992) or Actor-Critic methods (Konda & Tsitsiklis, 1999).

3.4 Score-based Generative Models

Score-based Generative Models (SGM) construct a tractable generative procedure by coupling (i) a *forward process* that progressively perturbs the training distribution π_{data} by adding noise to the data until its distribution approaches a simple reference distribution π_∞, and (ii) a *backward process* that learns to reverse this noising dynamics by sequentially removing the noise. The forward noising process (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2021) is the solution to the following stochastic differential equation (SDE): $\overrightarrow{Z}_0 \sim \pi_{\text{data}}$ and, for $\tau \in [0, \mathcal{T}]$,

$$\mathrm{d}\overrightarrow{Z}_\tau = -\beta_{\mathrm{sch}}(\tau)\overrightarrow{Z}_\tau \mathrm{d}\tau + \sqrt{2\beta_{\mathrm{sch}}(\tau)}\mathrm{d}B_\tau \ ,$$

where $(B_\tau)_{\tau \in [0,\mathcal{T}]}$ is a Brownian motion, $\beta_{\mathrm{sch}}(\tau) : [0, \mathcal{T}] \to \mathbb{R}_+$ is a noise schedule, and $\mathcal{T}$ is a fixed time horizon. Under mild regularity conditions, this transformation can be reversed (Haussmann & Pardoux, 1986). The corresponding reverse-time dynamics are also governed by an SDE, known as the backward process: $\overleftarrow{Z}_0 \sim p_{\mathcal{T}}$ and, for $\tau \in [0, \mathcal{T}]$,

$$\mathrm{d}\overleftarrow{Z}_\tau = \left(\beta_{\mathrm{sch}}(\mathcal{T}-\tau)\overleftarrow{Z}_\tau + 2\beta_{\mathrm{sch}}(\mathcal{T}-\tau)\nabla \log p_{\mathcal{T}-\tau}(\overleftarrow{Z}_\tau) \right) \mathrm{d}\tau + \sqrt{2\beta_{\mathrm{sch}}(\mathcal{T}-\tau)}\mathrm{d}B_\tau \ ,$$

where p_τ denotes the probability density function of $\overrightarrow{Z}_\tau$ for $0 \leq \tau \leq \mathcal{T}$. The score function $\nabla \log p_\tau$ depends on the unknown data distribution and therefore cannot be evaluated in closed form. It is approximated using a neural network $s_\psi : [0, \mathcal{T}] \times \mathbb{R}^d \to \mathbb{R}^d$, parameterized by $\psi \in \Psi$, and trained via denoising score matching (Vincent, 2011). Finally, exact i.i.d. sampling from the resulting continuous-time reverse process is generally intractable, so discretization is required, for instance using Euler–Maruyama or higher-order schemes such as exponential integrators.

4 Latent Guided Sampling

4.1 Model

The proposed model introduces a continuous latent search space for routing problems similar to Hottung et al. (2021); Chalumeau et al. (2023), which can be efficiently explored by any continuous

optimization method at inference time. To achieve this, we model the target distribution that generates a solution y given a problem instance x as a latent-variable model:

$$p_{\phi,\theta}(y|x) = \int p_\phi(z|x)p_\theta(y|x,z)\,\mathrm{d}z\ .$$

The encoder $p_\phi(z|x)$ maps the problem instance x to a continuous d_z-dimensional latent representation z. The decoder $p_\theta(y|x,z)$ then generates a solution y conditioned on both z and x. Both the encoder and decoder are parameterized by neural networks with learnable parameters $\phi \in \Phi$ and $\theta \in \Theta$, respectively. The probability of the decoder generating a solution y is then factorized as:

$$p_\theta(y|x,z) = \prod_{t=1}^{T} p_\theta(\mathsf{y}_t|\mathsf{y}_{1:t-1},x,z)\ ,$$

where $\mathsf{y}_t \in \{0,\dots,n\}$ is the selected node at step t, and $\mathsf{y}_{1:t-1}$ denotes the sequence of nodes selected up to step $t-1$. It is important to note that the encoder differs from the variational distribution $q_\phi(z|x,y)$ (Kingma & Welling, 2014); it corresponds to a CVAE-Opt (Hottung et al., 2021), which requires labeled data for training.

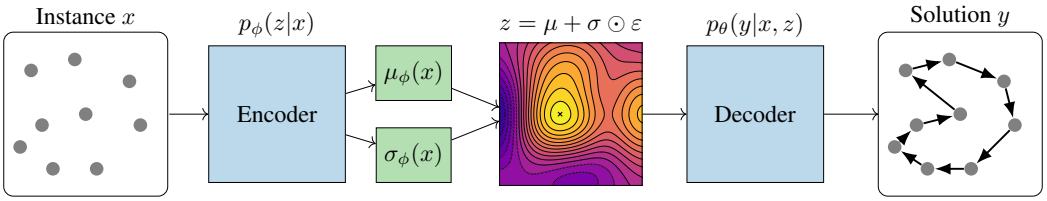

Figure 1: Our Latent Model Architecture with the Gaussian Reparameterization Trick

Our encoder architecture follows the general structure of Kool et al. (2019) but includes additional layers to compute the parameters of the encoder distribution. To compute output probabilities, we use a single decoder layer with multi-head attention to enable efficient inference. At step $0 \le t \le T$, for all $i \in \{0,\dots,n\}$, this layer computes the probability $p_\theta(\mathsf{y}_t = i|\mathsf{y}_{1:t-1},x,z)$ while masking nodes that lead to infeasible solutions. Details on the encoder and decoder are provided in Appendix A.2.

4.2 Training

During training, our objective is to minimize the cost C while encouraging diversity in the generated solutions to improve inference efficiency. To achieve this, we introduce an entropic regularization term (Ziebart et al., 2008; Haarnoja et al., 2018) controlled by a parameter β. The training loss is given by:

$$\mathcal{L}(\phi,\theta;x) = \sum_{k=1}^{K} \mathbb{E}_{z^k\sim p_\phi(\cdot|x)}\left[\mathbb{E}_{y^k\sim p_\theta(\cdot|x,z^k)}\left[w^k C(y^k,x)\right] + \beta\mathcal{H}(p_\theta(\cdot\mid x,z^k))\right]\ . \tag{2}$$

where $\mathcal{H}(p_\theta(\cdot|x,z))$ denotes the entropy of the conditional decoder distribution $p_\theta(y|x,z)$. The loss in (2) can be interpreted as a cost-weighted extension of Maximum Entropy RL (Ziebart et al., 2008), with weights $w^k = \exp(-C(y^k,x)/\upsilon)$ acting as importance factors that emphasize low-cost solutions. This formulation is inspired by IWAE (Burda et al., 2016) and the "best-of-many" objective (Bhattacharyya et al., 2018; Grinsztajn et al., 2023), interpolating between uniform weighting (large υ) that encourages exploration and near-greedy weighting (small υ) that prioritizes exploitation. In practice, υ is gradually decreased to encourage broad exploration early on, followed by concentration on promising regions of the latent space. A more detailed derivation and the complete training procedure are provided in Appendix A.3.

4.3 Inference

Any search strategy can be used at inference time to find the best solution while keeping the computational cost manageable. Possible approaches include evolutionary algorithms such as DE and

Algorithm 1 Diffusion-Prior Latent MCMC Sampling

Input: Problem instance x, pretrained encoder p_ϕ, pretrained decoder p_{θ_0}, score network s_ψ, proposal distribution q, number of particles K, number of iterations M, cost function C, and temperature λ.
Initialize:
Sample initial particles: $z_0^k \sim s_\psi(\cdot|x)$ for all $k = 1, \dots, K$.
for $m = 0, 1, \dots, M-1$ **do**
  Propagate new particles: $\tilde{z}_{m+1}^k \sim q(\cdot|z_m^{1:K})$ for all $k = 1, \dots, K$.
  Generate candidate solutions: $y_{m+1}^k \sim p_{\theta_m}(\cdot|x, \tilde{z}_{m+1}^k)$ for all $k = 1, \dots, K$.
  Compute acceptance probabilities:

$$\alpha_{m+1}^k = \min\left(1, \frac{p_\phi(\tilde{z}_{m+1}^k|x)}{p_\phi(z_m^k|x)} \times e^{-\lambda(C(y_{m+1}^k, x) - C(y_m^k, x))}\right) .$$

  Accept $z_{m+1}^k = \tilde{z}_{m+1}^k$ with probability α_{m+1}^k, otherwise $z_{m+1}^k = z_m^k$ for all $k = 1, \dots, K$.
  Compute the gradient estimate $H_{\theta_m}\left(x, \{(z_{m+1}^k, y_{m+1}^k)\}_{k=1}^K\right)$ using (5).
  Update parameters: $\theta_{m+1} = \theta_m - \gamma_{m+1} H_{\theta_m}\left(x, \{(z_{m+1}^k, y_{m+1}^k)\}_{k=1}^K\right)$.
end for

CMA-ES, as well as learnable methods like Active Search (Bello et al., 2017) and Efficient Active Search (Hottung et al., 2022). We formulate inference as a sampling problem: given an instance x, and the learned encoder and decoder parameters θ and ϕ, our goal is to sample from the distribution:

$$\pi_\theta(y|x) \propto \int \pi_\theta(z, y|x) \mathrm{d}z \ , \quad \text{where} \quad \pi_\theta(z, y|x) \propto p_\phi(z|x) p_\theta(y|x, z) e^{-\lambda C(y,x)} \ . \tag{3}$$

We omit the explicit dependence on ϕ since p_ϕ serves as a prior over latent variables. To favor lower-cost solutions, we introduce the reweighting factor $\exp(-\lambda C(y, x))$, where λ controls the trade-off between likelihood and cost. However, incorporating this reweighting renders the distribution in (3) intractable to sample from directly. While methods such as MCMC (Hastings, 1970) can be used to approximate it, they are often inefficient in practice. To address this challenge, we propose two diffusion-driven inference schemes. The first is a diffusion-prior MCMC method: we initialize a set of latent particles using a score-based model, and then, starting from these particles, we generate sequences of latent samples and corresponding solutions by running multiple interacting Markov Chains to encourage better exploration. The second is a diffusion-guided SMC variant that propagates a particle population through score-based latent diffusion dynamics, followed by cost-based importance weighting and resampling. Both schemes are coupled with Stochastic Approximation to adapt the decoder parameters θ at test time to minimize the following objective:

$$\mathcal{L}_{test}(\theta; x) = \mathbb{E}_{\pi_\theta(\cdot|x)}\left[C(y, x)\right] \ . \tag{4}$$

Since the solution quality depends on the trained parameters, it is natural to iteratively update θ. In contrast, the encoder parameters ϕ are kept fixed to avoid the high computational cost associated with backpropagating through them. The gradient is estimated using previously sampled latent variables:

$$H_\theta\left(x, \{(z^k, y^k)\}_{k=1}^K\right) = \frac{1}{K} \sum_{k=1}^K \left(C(y^k, x) - b(x)\right) \nabla_\theta \log p_\theta(y^k|x, z^k) \ , \tag{5}$$

where $b(x)$, defined in (10), serves as a baseline to reduce the variance. Both procedures are detailed in Algorithm 1 (diffusion-prior MCMC) and Algorithm 2 (diffusion-guided SMC). In both settings, the value of K should be chosen to balance stability (reducing the variance of gradient estimates), effective exploration of the latent space, and computational efficiency. Since the particles are generated by MCMC, the resulting gradient estimates are generally biased; convergence of SA methods with biased updates has been studied in Karimi et al. (2019); Surendran et al. (2024). Although we use standard gradient updates in both algorithms, alternative optimizers such as Adam (Kingma & Ba, 2015) can also be used.

Algorithm 2 Latent Diffusion-Guided SMC Sampling

Input: Problem instance x, pretrained encoder p_ϕ, pretrained decoder p_{θ_0}, score network s_ψ, schedule $\beta_{\text{sch}}(\cdot)$, number of particles K, diffusion horizon $\mathcal{T}$, number of steps M, cost function C, and temperature λ.
Initialize:
Sample initial particles: $z_0^k \sim p_\phi(\cdot|x)$ for all $k = 1, \dots, K$ and set step size $h = \mathcal{T}/M$
for $m = 0, 1, \dots, M-1$ **do**
    Set $\tau_m = \mathcal{T} - mh$.
    Sample $\varepsilon_m^k \sim \mathcal{N}(0, I)$ for all $k = 1, \dots, K$.
    Euler–Maruyama diffusion step:

$$\tilde{z}_{m+1}^k = z_m^k + \Big(\beta_{\text{sch}}(\tau_m)\, z_m^k + 2\,\beta_{\text{sch}}(\tau_m)\, s_\psi(\tau_m, z_m^k|x)\Big)\, h + \sqrt{2\,\beta_{\text{sch}}(\tau_m)\, h}\;\varepsilon_m^k\ .$$

    Generate candidate solutions: $y_{m+1}^k \sim p_{\theta_m}(\cdot|x, \tilde{z}_{m+1}^k)$ for all $k = 1, \dots, K$.
    Compute the weights: $\omega_{m+1}^k \propto \exp\big(-\lambda\, C(y_{m+1}^k, x)\big)$ for all $k = 1, \dots, K$.
    Resample $z_{m+1}^{1:K}$ from $\tilde{z}_{m+1}^{1:K}$ with probability $\omega_{m+1}^{1:K}$.
    Compute the gradient estimate $H_{\theta_m}\left(x, \{(z_{m+1}^k, y_{m+1}^k)\}_{k=1}^K\right)$ using (5).
    Update parameters: $\theta_{m+1} = \theta_m - \gamma_{m+1} H_{\theta_m}\left(x, \{(z_{m+1}^k, y_{m+1}^k)\}_{k=1}^K\right)$.
end for

5 EXPERIMENTS

In this section, we illustrate our method using two classic CO problems: the TSP and the CVRP. We evaluate performance using benchmark datasets from the literature (Hottung et al., 2021), consisting of 1,000 instances drawn from a training distribution—100 nodes uniformly sampled within the unit square. To evaluate generalization, we also test on two out-of-distribution datasets with larger sizes of 125 and 150 nodes. All experiments were conducted on a GPU cluster using a single NVIDIA RTX 6000 GPU.

LGS-Net Setup. The encoder uses multi-head attention with 8 heads and an embedding dimension of $d_h = 128$, and consists of 6 layers. The decoder includes a single multi-head attention layer with 8 heads and a key dimension of $d_k = 16$. Training is conducted only for instances with $n = 100$ nodes, using $K = 100$ latent samples. The latent space is defined as a compact space with diameter $R = 40$ and dimension $d_z = 100$. We use the Adam optimizer with a learning rate of 5×10^{-4}, a batch size of 128, and train for 8000 epochs. The entropic regularization parameter β is set to 0.01, and the weights υ in the loss (2) are chosen according to an exponential decay schedule.

Score-based Generative Model Setup. We construct a dataset of 100K instances from the training distribution and compute a target latent vector for each instance using Differential Evolution (500 steps). We then train a conditional variance-preserving score-based model (Section 3.4) with a linear noise schedule $\beta_{\text{sch}}(\cdot)$ on these latent vectors, conditioned on the instance through the frozen encoder representation of LGS-Net. The score network is implemented as a residual MLP with attention pooling and cross-attention between the latent variable and the encoder node embeddings. We optimize using Adam with learning rate 2×10^{-4} for 200 epochs.

Inference Setup. To avoid the overhead of gradient computation and keep inference costs moderate, we design the implementation as follows: (i) parameter updates are applied only to the last layer of the decoder (backpropagation through this layer is inexpensive), and (ii) instead of updating parameters at every iteration, updates are performed at fixed intervals. This avoids backpropagation through the full network at each step and keeps the SA component lightweight. The choice of fixed update intervals is motivated not only by computational considerations but also by the need to let the chains explore the latent space under fixed parameters, since changing the parameters at every iteration can hinder exploration. The effect of the parameter update interval is illustrated in Figure 8.

Baselines. We compare our model to a range of state-of-the-art learning-based NCO methods and industrial solvers. These include Concorde (Applegate et al., 2006), an exact solver specialized for the TSP, LKH3 (Helsgaun, 2017), a leading solver for CO problems, and Google OR-Tools (Perron & Furnon, 2019), a widely used suite of optimization tools. Among the NCO methods, we evaluate our

approach against POMO (Kwon et al., 2020), CVAE-Opt (Hottung et al., 2021), EAS (Hottung et al., 2022), COMPASS (Chalumeau et al., 2023), ELG (Gao et al., 2024), and CNF (Zhou et al., 2024).

Table 1: Experimental results on TSP and CVRP without the augmentation trick. "Obj." denotes the average total travel distance, and "Time" indicates the total runtime for solving 1000 instances.

		Training distribution			**Generalization**					
		n = 100			**n = 125**			**n = 150**		
	Method	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
TSP	Concorde	7.752	0.00%	8M	8.583	0.00%	12M	9.346	0.00%	17M
	LKH3	7.752	0.00%	47M	8.583	0.00%	73M	9.346	0.00%	99M
	POMO (greedy)	7.785	0.429%	<1M	8.640	0.664%	<1M	9.442	1.022%	<1M
	POMO (sampling)	7.768	0.206%	20M	8.614	0.361%	30M	9.406	0.642%	40M
	CVAE-Opt	7.779	0.348%	15H	8.646	0.736%	21H	9.482	1.454%	30H
	EAS	7.767	0.197%	20M	8.607	0.280%	30M	9.387	0.434%	40M
	COMPASS	7.753	0.014%	20M	8.586	0.035%	30M	9.358	0.128%	40M
	ELG	7.783	0.399%	20M	8.634	0.594%	30M	9.427	0.867%	40M
	CNF	7.766	0.181%	20M	8.607	0.279%	30M	9.394	0.514%	40M
	LGS-Net (Alg. 1-a)	**7.752**	**0.002%**	20M	**8.584**	**0.012%**	30M	9.354	0.081%	40M
	LGS-Net (Alg. 1-b)	**7.752**	**0.002%**	20M	**8.584**	**0.012%**	30M	**9.353**	**0.074%**	40M
	LGS-Net (Alg. 2)	7.754	0.025%	20M	8.591	0.089%	30M	9.368	0.235%	40M
CVRP	LKH3	15.54	0.00%	17H	17.50	0.00%	19H	19.22	0.00%	20H
	OR Tools	17.084	9.936%	38M	18.036	3.063%	64M	21.209	10.349%	73M
	POMO (greedy)	15.740	1.287%	<1M	17.905	2.314%	<1M	19.882	3.444%	<1M
	POMO (sampling)	15.607	0.431%	40M	17.652	0.869%	1H	19.539	1.659%	1H30
	CVAE-Opt	15.752	1.364%	32H	17.864	2.080%	36H	19.843	3.240%	46H
	EAS	15.563	0.148%	40M	17.541	0.234%	1H	19.319	0.515%	1H30
	COMPASS	15.561	0.135%	40M	17.546	0.263%	1H	19.358	0.718%	1H30
	ELG	15.736	1.261%	40M	17.729	1.308%	1H	19.516	1.540%	1H30
	CNF	15.591	0.328%	40M	17.682	1.040%	1H	19.998	4.047%	1H30
	LGS-Net (Alg. 1-a)	15.524	-0.102%	40M	17.496	-0.022%	1H	19.286	0.343%	1H30
	LGS-Net (Alg. 1-b)	**15.522**	**-0.113%**	40M	**17.489**	**-0.061%**	1H	**19.278**	**0.302%**	1H30
	LGS-Net (Alg. 2)	15.577	0.236%	40M	17.554	0.309%	1H	19.356	0.710%	1H30

The average performance of each method on TSP and CVRP, evaluated in terms of objective value (cost), optimality gap, and computation time, is reported in Table 1. The corresponding results obtained with the augmentation trick of Kwon et al. (2020) are provided in Tables 4 and 5. The gap to the reference solution is defined as

$$\text{Gap}(y, y^*) = \left(\frac{C(y, x)}{C(y^*, x)} - 1 \right) \times 100\% \,, \tag{6}$$

where y^* denotes the optimal solution for TSP and the near-optimal solution provided by LKH3 for CVRP. For LGS-Net, we report three inference variants: Alg. 1-a corresponds to Algorithm 1 without the diffusion prior, using the encoder-based prior; Alg. 1-b corresponds to Algorithm 1 with the diffusion prior; and Alg. 2 corresponds to the diffusion-guided SMC procedure in Algorithm 2.

Overall, our approach achieves state-of-the-art performance in most settings. For TSP, all three variants, especially both versions of Algorithm 1, produce near-optimal solutions and outperform competing methods on both in-distribution and out-of-distribution instances. Latent space models trained via RL (ours and COMPASS) outperform other baselines, highlighting the effectiveness of learned latent representations for capturing solution diversity. Importantly, our method, in particular Algorithm 1, surpasses COMPASS in all TSP settings. Although EAS achieves reasonable performance, it is considerably more expensive due to gradient computations at each iteration. For the CVRP, our model again outperforms all baselines, including both COMPASS and EAS. Moreover, Algorithm 1 also surpasses LKH3 on instances with $n = 100$ and $n = 125$, and remains the only learning-based model that outperforms LKH3. Comparing the two versions of Algorithm 1, incorporating a diffusion prior yields performance comparable to the encoder-prior variant on TSP, while providing clear improvements on CVRP, particularly in out-of-distribution scenarios. This suggests that diffusion mainly acts as a learned proposal mechanism in latent space, improving initialization and exploration when generalization is more challenging. Algorithm 2 performs moderately on both TSP and CVRP, but still outperforms several baselines.

Finally, all three of our variants consistently outperform both generalization-boosting methods (ELG, CNF) on TSP and CVRP. While each improves upon POMO, both still fall short of our approach. Their benefits are more pronounced under the stricter inference budgets of their original works; with

our slightly larger budget, their advantage diminishes. In summary, our method achieves the best results across all TSP and CVRP settings and remains the top performer. While COMPASS is the closest competitor for TSP, EAS performs more strongly than COMPASS on CVRP, but still falls short of our method. The diffusion prior in Algorithm 1 is most beneficial in out-of-distribution settings, especially for CVRP, while the diffusion-guided SMC variant in Algorithm 2 is less competitive than the diffusion-prior MCMC sampler but still outperforms several baselines.

We also compare training our model with the proposed cost-weighted loss to training with the POPPY loss of Grinsztajn et al. (2023). As shown in Table 6, our loss yields slightly better results, though the gap remains small because our sufficiently large inference-time budget tends to mitigate differences induced by the training objective. We further compare deterministic and stochastic latent variable models in Table 8, and we compare our instance-conditioned model with its instance-independent variant in Table 7. These results highlight the importance of using a stochastic, instance-conditioned latent model.

Table 2: Comparison of different inference methods with our model on CVRP with $n = 100$

Method	Obj.	Gap
Sampling	15.652	0.721%
DE	15.561	0.135%
CMA-ES	15.582	0.271%
EAS	15.685	0.933%
Adam	15.632	0.592%
SGLD	15.607	0.431%
Single MCMC	15.649	0.701%
Parallel MCMC (ours)	15.557	0.109%
Interacting MCMC (ours)	15.535	-0.032%
Alg. 1-a (ours)	15.524	-0.102%
Alg. 1-b (ours)	**15.522**	**-0.113%**
Alg. 2 (ours)	15.577	0.236%

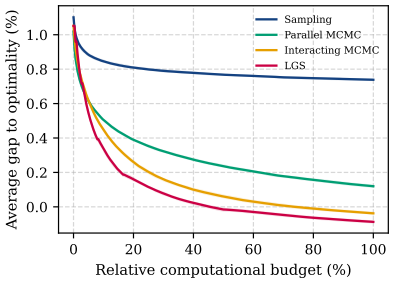

Figure 2: Performance of sampling-based methods on CVRP with $n = 100$, with bold lines indicating the mean over 10 inference runs

Table 2 and Figure 2 present the results of ablation studies, focusing on the comparison of different inference methods with our model on CVRP instances with $n = 100$ for a fixed time. Among the methods evaluated, Parallel MCMC, Interacting MCMC, and LGS (Algorithm 1) consistently outperform other techniques, particularly DE and CMA-ES, which are commonly used inference methods in continuous spaces. In contrast, Single MCMC struggles due to limited exploration, leading to poor performance. Surprisingly, even gradient-based approaches such as Stochastic Gradient Langevin Dynamics (SGLD) and Adam perform poorly, as the high cost of gradient computations limits their effectiveness. EAS also underperforms, as the initial particles are insufficiently effective, and adjusting the parameters does not significantly improve the solution. This highlights the critical importance of particle propagation and parameter learning in improving solution quality. Notably, our method is the only one with negative gaps, yielding lower-cost solutions than LKH3 on CVRP.

Additionally, the "Sampling" method corresponds to direct sampling from the distribution defined in (3), without the reweighting factor $\exp(-\lambda C(y, x))$. Figure 2 highlights the advantage of incorporating this reweighting factor: the blue curve shows sampling without this factor, whereas the green curve shows sampling with this factor using parallel MCMC but with no learning or interaction between chains. Adding interaction and learning further improves performance. Without SA, interacting MCMC may sample from a mismatched distribution and produce suboptimal solutions, even though the reweighting factor still provides some bias. Figure 3 illustrates how our method explores the continuous latent space to discover high-quality solutions, concentrating on lower-cost regions.

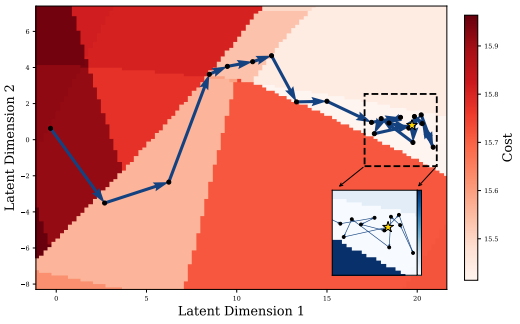

Figure 3: Visualization of the 2-dimensional latent space ($d_z = 2$) learned by our model on a problem instance. The plotted path illustrates the search trajectory leading to the best-found solution.

6 CONCLUSION

This paper introduces LGS-Net, a novel latent space model for Neural Combinatorial Optimization that conditions directly on problem instances, thereby removing the need for labeled data and pretrained policies. We also propose two diffusion-driven guided inference methods for LGS-Net: a diffusion-prior MCMC sampler and a diffusion-guided SMC alternative, both coupled with Stochastic Approximation for test-time adaptation toward low-cost solutions. We evaluate our approach on TSP and CVRP and achieve state-of-the-art performance among learning-based CO methods. In particular, the diffusion-prior MCMC method achieves the most competitive performance, although it introduces additional offline cost due to the construction of target latent vectors used to train the diffusion model. A promising direction for future work is to strengthen diffusion-based guidance during inference, building on our diffusion-guided SMC variant, and to explore training diffusion samplers without incurring substantial computational cost.

ACKNOWLEDGEMENTS

The PhD of Sobihan Surendran was funded by the Paris Region PhD Fellowship Program of Région Ile-de-France. We would like to thank SCAI (Sorbonne Center for Artificial Intelligence) for providing the computing clusters.

REFERENCES

Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special lecture on IE*, 2(1):1–18, 2015.

David Applegate, Ribert Bixby, Vasek Chvatal, and William Cook. Concorde TSP solver, 2006.

Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *International Conference on Learning Representations, Workshop Track*, 2017.

Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J Hu, Mo Tiwari, and Emmanuel Bengio. Gflownet foundations. *Journal of Machine Learning Research*, 24(210):1–55, 2023.

Apratim Bhattacharyya, Bernt Schiele, and Mario Fritz. Accurate and diverse sampling of sequences based on a "best of many" sample objective. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8485–8493, 2018.

Samuel Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. In *Proceedings of the 20th SIGNLL conference on computational natural language learning*, pp. 10–21, 2016.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. In *International Conference on Learning Representations*, 2016.

Felix Chalumeau, Shikha Surana, Clément Bonnet, Nathan Grinsztajn, Arnu Pretorius, Alexandre Laterre, and Tom Barrett. Combinatorial optimization with policy adaptation using latent space search. In *Advances in Neural Information Processing Systems*, volume 36, pp. 7947–7959, 2023.

Felix Chalumeau, Refiloe Shabe, Noah De Nicola, Arnu Pretorius, Thomas D Barrett, and Nathan Grinsztajn. Memory-enhanced neural solvers for routing problems. In *Advances in Neural Information Processing Systems*, 2025.

Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Jinho Choo, Yeong-Dae Kwon, Jihoon Kim, Jeongwoo Jae, André Hottung, Kevin Tierney, and Youngjune Gwon. Simulation-guided beam search for neural combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 35, pp. 8760–8772, 2022.

Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340. Springer, 2008.

Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the tsp by policy gradient. In *International conference on the integration of constraint programming, artificial intelligence, and operations research*, pp. 170–181. Springer, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

Alexandre Dolgui, Dmitry Ivanov, Suresh P Sethi, and Boris Sokolov. Scheduling in production, supply chain and industry 4.0 systems by optimal control: fundamentals, state-of-the-art and applications. *International journal of production research*, 57(2):411–432, 2019.

Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics letters B*, 195(2):216–222, 1987.

Ying Fan, Olivia Watkins, Yuqing Du, Hao Liu, Moonkyung Ryu, Craig Boutilier, Pieter Abbeel, Mohammad Ghavamzadeh, Kangwook Lee, and Kimin Lee. Dpok: Reinforcement learning for fine-tuning text-to-image diffusion models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 79858–79885, 2023.

Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large tsp instances. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 7474–7482, 2021.

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.

Andrew Gelman, Walter R Gilks, and Gareth O Roberts. Weak convergence and optimal scaling of random walk metropolis algorithms. *The Annals of Applied Probability*, 7(1):110–120, 1997.

Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6): 721–741, 1984.

Fred Glover. Tabu search—part I. *ORSA Journal on computing*, 1(3):190–206, 1989.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, volume 27, 2014.

Nathan Grinsztajn, Daniel Furelos-Blanco, Shikha Surana, Clément Bonnet, and Tom Barrett. Winner takes it all: Training performant rl populations for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 36, pp. 48485–48509, 2023.

Heikki Haario, Eero Saksman, and Johanna Tamminen. An adaptive metropolis algorithm. *Bernoulli*, 7(2):223–242, 2001.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pp. 1861–1870. PMLR, 2018.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 1970.

Ulrich G Haussmann and Etienne Pardoux. Time reversal of diffusions. *The Annals of Probability*, pp. 1188–1205, 1986.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12:966–980, 2017.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in neural information processing systems*, volume 33, pp. 6840–6851, 2020.

John J Hopfield and David W Tank. "neural" computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.

André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. In *ECAI 2020*, pp. 443–450. IOS Press, 2020.

André Hottung, Bhanu Bhandari, and Kevin Tierney. Learning a latent search space for routing problems using variational autoencoders. In *International Conference on Learning Representations*, 2021.

André Hottung, Yeong-Dae Kwon, and Kevin Tierney. Efficient active search for combinatorial optimization problems. In *International Conference on Learning Representations*, 2022.

André Hottung, Mridul Mahajan, and Kevin Tierney. Polynet: Learning diverse solution strategies for neural combinatorial optimization. In *International Conference on Learning Representations*, 2025.

Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1501–1510, 2017.

Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

Chaitanya K Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning the travelling salesperson problem requires rethinking generalization. *Constraints*, 27(1):70–98, 2022.

Belhal Karimi, Blazej Miasojedow, Eric Moulines, and Hoi-To Wai. Non-asymptotic analysis of biased stochastic approximation scheme. In *Conference on Learning Theory*, pp. 1944–1974. PMLR, 2019.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Minsu Kim, Jinkyoo Park, et al. Learning collaborative policies to solve np-hard routing problems. In *Advances in Neural Information Processing Systems*, volume 34, pp. 10418–10430, 2021.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, volume 12, 1999.

Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

Wouter Kool, Herke van Hoof, Joaquim Gromicho, and Max Welling. Deep policy dynamic programming for vehicle routing problems. In *International conference on integration of constraint programming, artificial intelligence, and operations research*, pp. 190–213. Springer, 2022.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

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

Yeong-Dae Kwon, Jinho Choo, Iljoo Yoon, Minah Park, Duwon Park, and Youngjune Gwon. Matrix encoding networks for neural combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 34, pp. 5138–5149, 2021.

Ruiwu Liu, Xiaocen Li, and Kit S Lam. Combinatorial chemistry in drug discovery. *Current opinion in chemical biology*, 38:117–126, 2017.

Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle routing problems. In *International Conference on Learning Representations*, 2019.

Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

Laurent Perron and Vincent Furnon. OR-Tools. `https://developers.google.com/optimization/`, 2019.

David Pisinger and Paolo Toth. Knapsack problems. In *Handbook of Combinatorial Optimization: Volume1–3*, pp. 299–428. Springer, 1998.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pp. 2256–2265. PMLR, 2015.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

Volker Steinbiss, Bach-Hiep Tran, and Hermann Ney. Improvements in beam search. In *ICSLP*, volume 94, pp. 2143–2146, 1994.

Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11:341–359, 1997.

Sobihan Surendran, Adeline Fermanian, Antoine Godichon-Baggioni, and Sylvain Le Corff. Non-asymptotic analysis of biased adaptive stochastic approximation. In *Advances in Neural Information Processing Systems*, volume 37, pp. 12897–12943, 2024.

Sobihan Surendran, Antoine Godichon-Baggioni, and Sylvain Le Corff. Theoretical convergence guarantees for variational autoencoders. In *International Conference on Artificial Intelligence and Statistics*, 2025.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, volume 27, 2014.

Masatoshi Uehara, Yulai Zhao, Tommaso Biancalani, and Sergey Levine. Understanding reinforcement learning-based fine-tuning of diffusion models: A tutorial and review. *arXiv preprint arXiv:2407.13734*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Siddarth Venkatraman, Moksh Jain, Luca Scimeca, Minsu Kim, Marcin Sendera, Mohsin Hasan, Luke Rowe, Sarthak Mittal, Pablo Lemos, Emmanuel Bengio, et al. Amortizing intractable inference in diffusion models for vision, language, and control. In *Advances in Neural Information Processing Systems*, volume 37, pp. 76080–76114, 2024.

Matthew Veres and Medhat Moussa. Deep learning for intelligent transportation systems: A survey of emerging trends. *IEEE Transactions on Intelligent transportation systems*, 21(8):3152–3168, 2019.

Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674, 2011. doi: 10.1162/NECO_a_00142.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Step-wise deep learning models for solving routing problems. *IEEE Transactions on Industrial Informatics*, 17(7):4861–4871, 2020.

Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 12042–12049, 2021.

Ni Zhang, Jingfeng Yang, Zhiguang Cao, and Xu Chi. Adversarial generative flow network for solving vehicle routing problems. In *International Conference on Learning Representations*, 2025.

Jianan Zhou, Yaoxin Wu, Zhiguang Cao, Wen Song, Jie Zhang, and Zhiqi Shen. Collaboration! towards robust neural methods for routing problems. In *Advances in Neural Information Processing Systems*, volume 37, pp. 121731–121764, 2024.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pp. 1433–1438. Chicago, IL, USA, 2008.

NOTATION

Table 3: Summary of notation used throughout the paper.

Object	Description
$x = \{\mathsf{x}_i\}_{i=1}^n \in \mathsf{X} \subset \mathbb{R}^{n \times d_x}$	Problem instance
$y = (\mathsf{y}_1, \cdots, \mathsf{y}_T) \in \mathsf{Y} \subset \{0, \cdots, n\}^T$	Solution
$z \in \mathsf{Z} \subset \mathbb{R}^{d_z}$	Latent variable
$\mathbb{P}_x$	Distribution over problem instances
$p_\phi(z\|x)$	Encoder distribution
$p_\theta(y\|x, z)$	Decoder distribution
C	Cost function
n	Number of nodes in the instance
t, T	Decoding step index and horizon
m, M	Inference step index and total iterations
k, K	Particle index and total number of particles
$\tau, \mathcal{T}$	Diffusion time and diffusion horizon
B	Batch size used during training

For a given batch of problem instances, we denote by $x_{(i)}$ the i-th input in a training batch. The corresponding solution and latent variable samples are denoted by $y_{(i)}^k$ and $z_{(i)}^k$ respectively, where k indexes multiple samples drawn for the same input $x_{(i)}$. During inference, we denote by y_m^k and z_m^k the solution and latent variable of the k-th particle at the m-th inference iteration.

A PROBLEM AND MODEL DESCRIPTION

A.1 PROBLEM SETTING

Traveling Salesman Problem (TSP). A TSP instance $x = \{\mathsf{x}_i\}_{i=1}^n$ consists of a set of n nodes, where the feature x_i corresponds to its coordinates $c_i \in \mathbb{R}^2$. The objective is to find a permutation $y = (\mathsf{y}_1, \ldots, \mathsf{y}_n)$ of the nodes, where $\mathsf{y}_t \in \{1, \ldots, n\}$ and $\mathsf{y}_t \neq \mathsf{y}_{t'}$ for all $t \neq t'$, that minimizes the total tour length:

$$C(y, x) = \sum_{i=1}^{n-1} \|x_{\mathsf{y}_{i+1}} - x_{\mathsf{y}_i}\| + \|x_{\mathsf{y}_n} - x_{\mathsf{y}_1}\| , \tag{7}$$

where $\|\cdot\|$ denotes the Euclidean norm. Note that the number of decoder steps T equals the number of nodes n, i.e., $T = n$.

Capacitated Vehicle Routing Problem (CVRP). CVRP generalizes TSP by introducing a depot (indexed as 0) and multiple routes, each starting and ending at the depot. Each customer $i \in \{1, \ldots, n\}$ has a demand $d_i > 0$ and a location $c_i \in \mathbb{R}^2$, while the depot has $d_0 = 0$. A fleet of vehicles, each with a capacity $D > 0$, serves the customers. The goal is to determine the minimum number of vehicles and the corresponding routes, ensuring that each customer is visited exactly once and that the total demand in each route does not exceed D: for any route j,

$$\sum_{i \in R_j} d_i \leq D ,$$

where R_j denotes the set of customers assigned to route j.

A.2 MODEL ARCHITECTURE DETAILS

A.2.1 ENCODER

Given d_x-dimensional input features x_i, the encoder initially computes d_h-dimensional node embeddings $h_i^{(0)}$ through a learned linear projection using parameters W_0 and b_0:

$$h_i^{(0)} = W_0 \mathsf{x}_i + b_0 .$$

The embeddings are updated using L attention layers, each consisting of two sublayers: a multi-head attention (MHA) layer followed by a node-wise fully connected feed-forward (FF) layer. Each sublayer adds a skip connection (He et al., 2016) and instance normalization (InstanceNorm) (Huang & Belongie, 2017). Denoting $h_i^{(l)}$ as the node embeddings produced by layer $l \in \{1, \ldots, L\}$, the updates are defined as follows:

$$\hat{h}_i^{(l+1)} = \mathsf{InstanceNorm}\left(h_i^{(l)} + \mathsf{MHA}\left(h_1^{(l)}, \ldots, h_n^{(l)}\right)\right),$$
$$h_i^{(l+1)} = \mathsf{InstanceNorm}\left(\hat{h}_i^{(l+1)} + \mathsf{FF}(\hat{h}_i^{(l+1)})\right) .$$

Then, it computes an aggregated embedding $\bar{h}^{(L)}$ of the input graph as the mean of the final node embeddings $h_i^{(L)}$. Finally, the encoder generates a latent space vector using a reparameterization trick:

$$z = \mu_\phi(x) + \sigma_\phi(x) \odot \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, I_{d_z}) ,$$

where the mean $\mu_\phi(x)$ and log-variance $\log \sigma_\phi(x)^2$ of the conditional distribution $p_\phi(z|x)$ are given by:

$$\mu_\phi(x) = \mathsf{FF}\left(\bar{h}^{(L)}\right) \qquad \text{and} \qquad \log \sigma_\phi(x)^2 = \mathsf{FF}\left(\bar{h}^{(L)}\right) ,$$

where the feedforward network FF uses a soft-clipping activation function (Surendran et al., 2025).

A.2.2 TSP Decoder

The context c_t for the TSP decoder at time t is derived by combining the latent vector z and the output up to time $t-1$. Specifically, the context is defined as:

$$c_t = \left[z, h_{\mathsf{y}_{t-1}}^{(L)}, h_{\mathsf{y}_0}^{(L)}\right] ,$$

where $h_{\mathsf{y}_0}^{(L)}$ and $h_{\mathsf{y}_{t-1}}^{(L)}$ represent the embeddings of the starting node and the previously selected node, respectively.

Computation of Probabilities. The output probabilities for the TSP decoder are computed using a single decoder layer with multi-head attention. This layer computes probabilities while incorporating a masking mechanism:

$$p_\theta(\mathsf{y}_t = i|\mathsf{y}_{0:t-1}, x, z) = \begin{cases} \text{softmax}\left(\omega \tanh\left(\frac{q_{(c_t)}^T k_i}{\sqrt{d_k}}\right)\right) & \text{if } i \neq \mathsf{y}_s \quad \forall s < t , \\ 0 & \text{otherwise} , \end{cases}$$

where the query and key are given by $q_{(c_t)} = \mathsf{MHA}\left(c_t, \{h_i^{(L)}\}_{i=1}^n, \{h_i^{(L)}\}_{i=1}^n\right)$ and $k_i = W^K h_i^{(L)}$ respectively.

A.2.3 CVRP Decoder

Similar to the TSP decoder, the context c_t for the CVRP decoder at time t is derived by combining the latent vector z and the output up to time $t-1$. Specifically, the context is defined as:

$$c_t = \left[z, h_{\mathsf{y}_{t-1}}^{(L)}, \widehat{D}_t\right] ,$$

where we keep track of the remaining vehicle capacity $\widehat{D}_t$ at time t. At $t = 1$, this is initialized as $\widehat{D}_t = D$, after which it is updated as follows:

$$\widehat{D}_{t+1} = \begin{cases} \max(\widehat{D}_t - d_{\mathsf{y}_t, t}, 0) & \text{if } \mathsf{y}_t \neq 0 , \\ D & \text{if } \mathsf{y}_t = 0 . \end{cases}$$

Computation of Probabilities. The output probabilities for the CVRP decoder are computed using a single decoder layer with multi-head attention. This layer computes probabilities while incorporating a masking mechanism:

$$p_\theta(\mathsf{y}_t = i|\mathsf{y}_{0:t-1}, x, z) = \begin{cases} \text{softmax}\left(\omega \tanh\left(\frac{q_{(c_t)}^T k_i}{\sqrt{d_k}}\right)\right) & \text{if } i \neq \mathsf{y}_s \quad \forall s < t \quad \text{and} \quad d_{i,t} \leq \widehat{D}_t , \\ 0 & \text{otherwise} , \end{cases}$$

where the query and key are given by $q_{(c_t)} = \mathsf{MHA}\left(c_t, \{h_i^{(L)}\}_{i=1}^n, \{h_i^{(L)}\}_{i=1}^n\right)$ and $k_i = W^K h_i^{(L)}$ respectively.

A.3 TRAINING

When setting $w^k = 1$ and $\beta = 0$ in the training objective defined in (2), the loss reduces to minimizing the expected cost over K latent samples, without weighting or entropy regularization. We introduce the entropy term (controlled by β) to encourage diversity in the decoder's outputs, inspired by maximum entropy reinforcement learning (Ziebart et al., 2008; Haarnoja et al., 2018).

One could, in principle, also regularize the encoder p_ϕ via an entropy term. However, we found empirically that this had negligible impact on performance. Our intuition is that the decoder already induces sufficient stochasticity, and that exploration of the latent space is effectively handled during inference. Moreover, including encoder regularization would require tuning additional hyperparameters, which we deliberately avoided for the sake of simplicity.

Justification for the weights w^k.
In Bhattacharyya et al. (2018), the authors motivate the "Best-of-Many" sample objective by observing that the standard multi-sample VAE (Kingma & Welling, 2014) loss averages the reconstruction error across all latent samples, allowing even low-quality samples to influence learning. To address this, they propose training on only the best sample. However, relying solely on the best sample may overlook valuable learning signals from other informative samples.

The idea behind our introduction of weights $w^k = \exp(-C(y^k, x)/\upsilon)$ serves to softly prioritize lower-cost solutions among multiple latent samples drawn from $p_\phi(z|x)$. This design is inspired by importance weighting techniques used in IWAE (Burda et al., 2016), where more promising samples are assigned greater influence during training. In our setting, all samples contribute to the gradient estimator, but poor-quality samples have reduced impact, while more promising ones are emphasized.

As discussed in Section 4.2, when $w^k = 1$ (υ is large), the model treats all samples equally, encouraging exploration. In contrast, when υ is small, the weighting scheme closely resembles training on only the best sample, as in Bhattacharyya et al. (2018). In our experiments, we employ a decreasing schedule for υ, which enables more stable learning in the early stages while progressively concentrating on high-quality regions of the latent space.

Importance sampling view of the weighted objective.
Let $q_{\theta,\phi}(z, y \mid x) = p_\phi(z \mid x) p_\theta(y \mid x, z)$ denote the joint distribution and

$$\pi_{\theta,\phi}(z, y \mid x) \propto q_{\theta,\phi}(z, y \mid x) e^{-C(y,x)/\upsilon}$$

the cost-weighted joint distribution, with normalizing constant $Z(x)$. The importance ratio $w(z, y) = \pi_{\theta,\phi}/q_{\theta,\phi} \propto e^{-C(y,x)/\upsilon}$ satisfies

$$\mathbb{E}_{q_{\theta,\phi}}\big[w(z, y)C(y, x)\big] = \mathbb{E}_{\pi_{\theta,\phi}}[C(y, x)].$$

Indeed, we have:

$$\begin{aligned}
\mathbb{E}_{q_{\theta,\phi}}\big[w(z, y)C(y, x)\big] &= \int C(y, x) \frac{\pi_{\theta,\phi}(z, y \mid x)}{q_{\theta,\phi}(z, y \mid x)} q_{\theta,\phi}(z, y \mid x) dz dy \\
&= \frac{1}{Z(x)} \int C(y, x) e^{-C(y,x)/\upsilon} q_{\theta,\phi}(z, y \mid x) dz dy \\
&= \mathbb{E}_{\pi_{\theta,\phi}(\cdot|x)}\big[C(y, x)\big].
\end{aligned}$$

Drawing K i.i.d. pairs $(z^k, y^k) \sim q_{\theta,\phi}(\cdot \mid x)$, we obtain

$$\mathbb{E}_{q_{\theta,\phi}^{\otimes K}}\left[\sum_{k=1}^K w^k C\big(y^k, x\big)\right] = \mathbb{E}_{\pi_{\theta,\phi}}[C(y, x)].$$

Hence, the weighted multi-sample loss can be interpreted as an importance sampling estimator of the expected cost under the cost-weighted target distribution $\pi_{\theta,\phi}$.

Gradient computation.

The following proposition provides the gradient of the training objective defined in (2).

Proposition A.1. *For all $x \in \mathsf{X}$, $\theta \in \Theta$ and $\phi \in \Phi$, we have:*

$$\begin{aligned}
\nabla_\theta \mathcal{L}(\phi, \theta; x) = &\sum_{k=1}^{K} \mathbb{E}_{z^k \sim p_\phi(\cdot|x)} \left[\mathbb{E}_{y^k \sim p_\theta(\cdot|x,z^k)} \left[w^k C(y^k, x) \nabla_\theta \log p_\theta(y^k|x, z^k) \right] \right] \\
&- \beta \sum_{k=1}^{K} \mathbb{E}_{z^k \sim p_\phi(\cdot|x)} \left[\mathbb{E}_{y^k \sim p_\theta(\cdot|x,z^k)} \left[\log p_\theta(y^k|x, z^k) \nabla_\theta \log p_\theta(y^k|x, z^k) \right] \right] , \\
\nabla_\phi \mathcal{L}(\phi, \theta; x) = &\sum_{k=1}^{K} \mathbb{E}_{z^k \sim p_\phi(\cdot|x)} \left[\mathbb{E}_{y^k \sim p_\theta(\cdot|x,z^k)} \left[w^k C(y^k, x) \right] \nabla_\phi \log p_\phi(z^k|x) \right] \\
&+ \beta \sum_{k=1}^{K} \mathbb{E}_{z^k \sim p_\phi(\cdot|x)} \left[\mathcal{H}(p_\theta(\cdot \mid x, z^k)) \nabla_\phi \log p_\phi(z^k|x) \right] .
\end{aligned}$$

Proof. For the gradient with respect to θ, we have:

$$\nabla_\theta \mathcal{L}(\phi, \theta; x) = \sum_{k=1}^{K} \mathbb{E}_{z^k \sim p_\phi(\cdot|x)} \left[\nabla_\theta \mathbb{E}_{y^k \sim p_\theta(\cdot|x,z^k)} \left[w^k C(y^k, x) \right] + \beta \nabla_\theta \mathcal{H}(p_\theta(\cdot \mid x, z^k)) \right] .$$

For all $x \in \mathsf{X}$ and $z \in \mathsf{Z}$, applying the score-function (log-derivative trick) identity gives

$$\nabla_\theta \, \mathbb{E}_{y \sim p_\theta(\cdot|x,z)} \left[w C(y, x) \right] = \mathbb{E}_{y \sim p_\theta(\cdot|x,z)} \left[w C(y, x) \nabla_\theta \log p_\theta(y|x, z) \right]$$

Moreover, since $\mathcal{H}(p_\theta(\cdot \mid x, z)) = -\mathbb{E}_{y \sim p_\theta(\cdot|x,z)} \left[\log p_\theta(y|x, z) \right]$, its gradient is

$$\nabla_\theta \mathcal{H}(p_\theta(\cdot \mid x, z)) = -\mathbb{E}_{y \sim p_\theta(\cdot|x,z)} \left[\log p_\theta(y|x, z) \nabla_\theta \log p_\theta(y|x, z) \right] .$$

Combining these two identities yields the desired expression for the gradient with respect to θ. For the gradient with respect to ϕ, we apply the score-function identity to the function $z \mapsto \mathbb{E}_{y \sim p_\theta(\cdot|x,z)} \left[w C(y, x) \right] + \beta \mathcal{H}(p_\theta(\cdot \mid x, z))$ which does not depend on ϕ. □

The estimator of the gradient of the objective defined in (2) is computed using the Monte Carlo method:

$$\widehat{\nabla}_\theta \mathcal{L} = \frac{1}{B} \sum_{i=1}^{B} \sum_{k=1}^{K} \left(w_{(i)}^k C(y_{(i)}^k, x_{(i)}) - \beta \log p_\theta(y_{(i)}^k|x_{(i)}, z_{(i)}^k) - b(x_{(i)}) \right) \nabla_\theta \log p_\theta(y_{(i)}^k|x_{(i)}, z_{(i)}^k) , \tag{8}$$

$$\widehat{\nabla}_\phi \mathcal{L} = \frac{1}{B} \sum_{i=1}^{B} \sum_{k=1}^{K} \left(w_{(i)}^k C(y_{(i)}^k, x_{(i)}) - \beta \log p_\theta(y_{(i)}^k|x_{(i)}, z_{(i)}^k) - b(x_{(i)}) \right) \nabla_\phi \log p_\phi(z_{(i)}^k|x_{(i)}) , \tag{9}$$

where $b(x)$ denotes the baseline function, which is given by

$$b(x) = \frac{1}{K} \sum_{k=1}^{K} \left(w^k C(y^k, x) - \beta \log p_\theta(y^k \mid x, z^k) \right) .$$

This update rule has an intuitive interpretation: it adjusts the parameters θ and ϕ in directions that favor solutions yielding the highest reward, while simultaneously constraining the latent space to remain bounded and encouraging diversity in the sampled trajectories. The training procedure is outlined in Algorithm 3. The update step ADAM for the parameters (ϕ, θ) corresponds to a single Adam update step (Kingma & Ba, 2015).

Algorithm 3 REINFORCE training

Input: Distribution over problem instances $\mathbb{P}_x$, number of training steps N, batch size B, and number of latent samples K.

1: Initialize the model parameters θ and ϕ.
2: **for** epoch $= 1$ to N **do**
3: Sample problem instances $x_{(i)} \sim \mathbb{P}_x$ for $i \in \{1, \dots, B\}$.
4: Generate latent samples $z^1_{(i)}, \dots, z^K_{(i)} \sim p_\phi^{\otimes K}(\cdot|x_{(i)})$ using the reparameterization trick.
5: Sample solutions $y^k_{(i)} \sim p_\theta\left(\cdot \mid x_{(i)}, z^k_{(i)}\right)$ for all $k = 1, \dots, K$.
6: Compute the gradient estimates $\widehat{\nabla}_{\phi,\theta}\mathcal{L}(\phi, \theta)$ using (8) and (9).
7: Update parameters: $(\phi, \theta) \leftarrow \mathrm{ADAM}\left((\phi, \theta), \widehat{\nabla}_{\phi,\theta}\mathcal{L}(\phi, \theta)\right)$.
8: **end for**

Output: Optimized parameters ϕ and θ.

A.4 INFERENCE

The gradient of the inference objective is obtained using the log-derivative trick, in the same way as in Proposition A.1:

$$\nabla_\theta \mathcal{L}_{test}(\theta; x) = \mathbb{E}_{\pi_\theta(\cdot|x)}\left[C(y, x)\nabla_\theta \log \pi_\theta(y \mid x)\right] \ .$$

This leads to the estimator in (5), with the baseline $b(x)$ defined as the average cost over the K latent samples for a given x:

$$b(x) = \frac{1}{K}\sum_{k=1}^{K} C(y^k, x) \ . \tag{10}$$

B Additional Experiments

B.1 Training Details

In this section, we provide the details of our model and training procedure. The encoder uses multi-head attention with 8 heads and an embedding dimension of $d_h = 128$, and consists of 6 layers. The decoder includes a single multi-head attention layer with 8 heads and a key dimension of $d_k = 16$.

For both the TSP and CVRP, node coordinates c_i are sampled uniformly within the unit square. In CVRP instances, customer demands d_i are drawn from a uniform distribution $\mathcal{U}([1, 10])$. The vehicle capacity D is set based on the number of nodes: $D = 50$ for $n = 100$, $D = 55$ for $n = 125$, and $D = 60$ for $n = 150$, following the setup used in the literature (Hottung et al., 2021).

Training is conducted only for instances with $n = 100$ nodes, using $K = 100$ latent samples. The latent space is defined as a compact space with diameter $R = 40$ and dimension $d_z = 100$. We use the Adam optimizer with a learning rate of 5×10^{-4}, a batch size of 128, and train for 8000 epochs. The momentum parameters are fixed at $\beta_1 = 0.9$ and $\beta_2 = 0.999$, with a weight decay of 1×10^{-6}. The entropic regularization parameter β is set to 0.01, and the weights υ in the loss (2) are chosen according to an exponential decay schedule. The training loss and corresponding cost are shown in Figure 4.

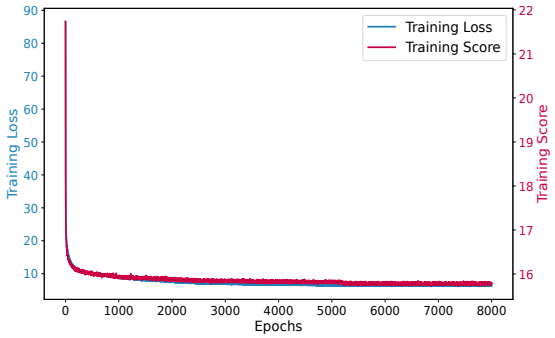

Figure 4: Training loss and score for our model trained on CVRP instances with nodes $n = 100$

B.2 Inference Details and Additional Experiments

B.2.1 Inference Details

The inference results typically include the objective value (cost), the optimality gap, and the computation time. For example, in Tables 4 and 5, Obj. denotes the value of the cost function defined in (7) for the TSP and CVRP.

In our experiments, we use a batch size of 200 for TSP and 100 for CVRP with $K = 600$ latent samples when the augmentation trick is not applied. When using the augmentation trick, we reduce the batch size to 100 for TSP and 50 for CVRP, and $K = 300$ latent samples.

The latent dimension is set to $d_z = 100$, and the latent space is constrained to a ball of radius $R = 40$. For MCMC, we use a Gaussian proposal distribution with density, for all $m \in \mathbb{N}$ and $1 \leq k \leq K$,

$$q(z_{m+1}^k \mid z_m^{1:K}) = \mathcal{N}\left(z_{m+1}^k; z_m^k + \gamma\left(z_m^{I_1} - z_m^{I_2}\right), \sigma^2 I_{d_z}\right) \ ,$$

where $I_1, I_2 \sim \mathcal{U}(\{1, \ldots, K\})$. The variance parameter is set to $\sigma^2 = 0.01$ and the scaling factor is $\gamma = 0.319$ for TSP and $\gamma = 0.379$ for CVRP. Instead of updating the parameters at every iteration, updates are performed at fixed intervals. The update schedule is described in the next section and illustrated in Figure 8.

B.2.2 ADDITIONAL EXPERIMENTS

Here, we present the experimental results for TSP and CVRP with the augmentation trick. All additional experiments are conducted with Algorithm 1 using the encoder-based prior (Alg. 1-a). "Obj." denotes the average total cost (travel distance) over all instances, while "Time" indicates the total runtime for all 1,000 instances. "Gap" defined in (6), measures the difference from the best-known solution (Concorde for TSP and LKH3 for CVRP). Concorde is an exact solver (optimal solutions), whereas LKH3 is a heuristic (near-optimal solutions). Consequently, negative gaps indicate that the corresponding method achieves lower-cost solutions than LKH3 on CVRP. For clarity and ease of comparison, we also report the results without augmentation, as originally shown in the main paper.

Table 4: Experimental results on TSP without and with the augmentation trick

		Training distribution			**Generalization**					
		n = 100			**n = 125**			**n = 150**		
	Method	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
	Concorde	7.752	0.00%	8M	8.583	0.00%	12M	9.346	0.00%	17M
	LKH3	7.752	0.00%	47M	8.583	0.00%	73M	9.346	0.00%	99M
no aug.	POMO (greedy)	7.785	0.429%	<1M	8.640	0.664%	<1M	9.442	1.022%	<1M
	POMO (sampling)	7.768	0.206%	20M	8.614	0.361%	30M	9.406	0.642%	40M
	CVAE-Opt	7.779	0.348%	15H	8.646	0.736%	21H	9.482	1.454%	30H
	EAS	7.767	0.197%	20M	8.607	0.280%	30M	9.387	0.434%	40M
	COMPASS	7.753	0.014%	20M	8.586	0.035%	30M	9.358	0.128%	40M
	ELG	7.783	0.399%	20M	8.634	0.594%	30M	9.427	0.867%	40M
	CNF	7.766	0.181%	20M	8.607	0.279%	30M	9.394	0.514%	40M
	LGS-Net (ours)	**7.752**	**0.002%**	20M	**8.584**	**0.012%**	30M	**9.354**	**0.081%**	40M
aug.	POMO (greedy)	7.762	0.132%	<1M	8.607	0.280%	<1M	9.397	0.541%	<1M
	POMO (sampling)	7.756	0.051%	40M	8.596	0.156%	60M	9.378	0.342%	90M
	EAS	7.755	0.042%	60M	8.591	0.093%	100M	9.363	0.177%	160M
	COMPASS	7.752	0.002%	40M	8.585	0.024%	60M	9.352	0.059%	90M
	ELG	7.761	0.116%	40M	8.606	0.268%	75M	9.391	0.481%	140M
	CNF	7.756	0.052%	40M	8.595	0.139%	75M	9.377	0.332%	140M
	LGS-Net (ours)	**7.752**	**0.000%**	40M	**8.583**	**0.001%**	60M	**9.349**	**0.027%**	90M

Table 5: Experimental results on CVRP without and with the augmentation trick

		Training distribution			**Generalization**					
		n = 100			**n = 125**			**n = 150**		
Method		Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
	LKH3	15.54	0.00%	17H	17.50	0.00%	19H	19.22	0.00%	20H
	OR Tools	17.084	9.936%	38M	18.036	3.063%	64M	21.209	10.349%	73M
no aug.	POMO (greedy)	15.740	1.287%	<1M	17.905	2.314%	<1M	19.882	3.444%	<1M
	POMO (sampling)	15.607	0.431%	40M	17.652	0.869%	1H	19.539	1.659%	1H30
	CVAE-Opt	15.752	1.364%	32H	17.864	2.080%	36H	19.843	3.240%	46H
	EAS	15.563	0.148%	40M	17.541	0.234%	1H	19.319	0.515%	1H30
	COMPASS	15.561	0.135%	40M	17.546	0.263%	1H	19.358	0.718%	1H30
	ELG	15.736	1.261%	40M	17.729	1.308%	1H	19.516	1.540%	1H30
	CNF	15.591	0.328%	40M	17.682	1.040%	1H	19.998	4.047%	1H30
	LGS-Net (ours)	**15.524**	**-0.102%**	40M	**17.496**	**-0.022%**	1H	**19.286**	**0.343%**	1H30
aug.	POMO (greedy)	15.652	0.721%	1M	17.756	1.463%	1M	19.701	2.503%	1M
	POMO (sampling)	15.561	0.137%	80M	17.583	0.475%	2H10	19.462	1.261%	3H20
	EAS	15.508	-0.205%	80M	17.466	-0.194%	2H10	**19.212**	**-0.041%**	3H20
	COMPASS	15.531	-0.057%	80M	17.512	0.068%	2H10	19.318	0.509%	3H20
	ELG	15.635	0.611%	90M	17.623	0.703%	2H30	19.421	1.046%	4H15
	CNF	15.553	0.084%	90M	17.607	0.611%	2H30	19.878	3.423%	4H15
	LGS-Net (ours)	**15.501**	**-0.251%**	80M	**17.461**	**-0.223%**	2H10	19.229	0.046%	3H20

The average performance of each method is reported in Table 4 (TSP) and Table 5 (CVRP), both with and without the augmentation trick of Kwon et al. (2020). Overall, our approach achieves state-of-the-art performance across most settings. For the TSP, our method produces near-optimal solutions even without augmentation, and consistently reaches optimality when the augmentation trick is applied.

For the CVRP without augmentation, our model again outperforms all baselines, including both COMPASS and EAS. It also surpasses the performance of LKH3 on the instances with $n = 100$

and $n = 125$. When augmentation is applied, performance improves further. However, for $n = 150$, EAS outperforms our method, likely due to the reduced number of latent samples imposed by the computational budget. In this case, exploring the latent space effectively may require a higher sample count. We note that, when solving one instance at a time (thus relaxing the budget constraint), our model can achieve even stronger performance under augmentation.

Illustration of the cost-weighted loss in training. We compare training our model with the proposed cost-weighted loss to using the POPPY loss from Grinsztajn et al. (2023); Chalumeau et al. (2023), which focuses on the best sample. Further details on the cost-weighted loss and its mathematical justification are provided in Appendix A.3. The empirical comparison is reported in Table 6.

Table 6: Comparison of cost-weighted loss and POPPY loss when training on CVRP with $n = 100$

Method	Obj.	Gap
POPPY loss	15.527	-0.083%
Cost-weighted loss	**15.524**	**-0.102%**

Training with the cost-weighted loss (used in our main experiments) yields slightly better results than the POPPY loss, although the performance gap remains small. We attribute this to the strength of the inference procedure. In particular, we allocate a sufficiently large inference-time budget, which tends to reduce performance differences arising from the choice of training objective. Moreover, since our inference method includes parameter updates, the model can partially "forget" suboptimal training parameters and adapt at test time.

Choice of the inference budget. In our experiments, all comparisons are carried out under a fixed wall-clock budget, rather than a fixed number of rollouts or attempts, so that methods with different per-iteration costs are evaluated under comparable computational effort. This wall-clock budget is fixed for the entire test set at a given problem size. To choose this budget, we proceeded as follows. Since our architecture builds on the CVAE-Opt setting, we first ran Differential Evolution (the inference method used in Hottung et al. (2021)) on our model with the same number of iterations as in their original setup and recorded the corresponding runtime. We then used this runtime as a reference and applied the same wall-clock budget to all methods at that problem size. The resulting budgets of 20M for TSP100 and 40M for CVRP100 correspond to roughly 1 second and 2 seconds per instance, respectively, which we found to provide a reasonable trade-off between solution quality and computational cost. All methods are evaluated in the same PyTorch codebase and on the same hardware (a single NVIDIA RTX 6000 GPU).

Comparison of our method with an instance-independent latent model. To isolate the effect of conditioning the latent space on the instance, we additionally study a variant of our model in which the latent distribution is instance-independent. Concretely, we replace the encoder $p_\phi(z \mid x)$ with an unconditional latent distribution $p_\phi(z)$ (a standard Gaussian), while keeping the decoder architecture, training objective, and inference procedure unchanged. In this setting, the same latent space must be shared across all instances, and the search in latent space can no longer adapt its prior structure to the specific geometry of each problem instance.

We report a comparison between the instance-conditioned and instance-independent variants, as well as our inference method applied to the COMPASS model, in Table 7. First, the combined approach (COMPASS + LGS) does not improve over the original COMPASS model and remains clearly below the performance of our full method. This is reasonable, since COMPASS differs from our model not only in the lack of instance conditioning and in its loss (POPPY), but also in how the latent space is used: COMPASS draws a small number of latent samples and, for each latent sample, generates a solution with different starting points. As a result, only a limited region of the latent space is actually explored, which we believe restricts the effectiveness of our latent-space search when applied on top of a COMPASS-trained model.

The instance-independent model underperforms both the instance-conditioned variant and COMPASS. We suspect that COMPASS nevertheless works well, despite using an instance-independent latent distribution, because its latent space is learned on top of a strong pre-trained policy. Overall, these

results confirm that conditioning on the instance x provides a more informative latent prior for the search and yields measurable gains, in line with the intuition that an instance-dependent latent space can better capture problem-specific structure.

Table 7: Comparison of instance-conditioned and instance-independent model on CVRP with $n = 100$

Method	Obj.	Gap
COMPASS + LGS	15.579	0.249%
Instance-independent model	15.641	0.648%
Instance-conditioned model	**15.524**	**-0.102%**

Comparison of deterministic and stochastic latent variables. In the deterministic case, the encoder maps each instance to a single latent vector, $z = f_\phi(x)$, where ϕ denotes the encoder parameters. This collapses the latent space to a single representation per instance, so the Markov chain no longer has a meaningful space to explore. In contrast, a stochastic encoder $p_\phi(z \mid x)$ defines a distribution over the latent space that captures model uncertainty and multiple promising regions. This distribution is exactly what our method exploits: the MCMC proposals and accept/reject steps are defined with respect to a target $\pi_\theta(z \mid x)$, and exploration in latent space translates into diverse and high-quality solutions in the original solution space.

We evaluate our inference method with a deterministic latent representation (deterministic embedding from POMO). Specifically, we apply (i) a deterministic latent model with our LGS inference, and (ii) a deterministic latent model with a gradient-based method similar to EAS. The results are reported in Table 8.

Table 8: Comparison of deterministic and stochastic latent variables on CVRP with $n = 100$

Method	Obj.	Gap
Deterministic latent (+LGS)	15.572	0.206%
Deterministic latent (+SGD)	15.564	0.154%
LGS (ours)	**15.524**	**-0.102%**

We observe a clear degradation for both deterministic latent variants compared to the stochastic latent model, highlighting the benefit of stochasticity. In this setting, the gradient-based method performs slightly better than applying LGS directly on a deterministic latent representation, which is consistent with the fact that, without a proper target distribution in latent space, MCMC-based exploration becomes less natural. In summary, the deterministic latent model is effective for producing greedy solutions under a very small inference budget, whereas for a fixed, larger budget, a stochastic latent variable substantially improves exploration and solution quality.

Comparison with other sampling methods. We now compare our method to the approaches discussed in Section 2: Active Search (Bello et al., 2017), which appears to be the most effective existing approach related to sampling non-differentiable reward functions (Fan et al., 2023; Uehara et al., 2024; Venkatraman et al., 2024), and the Adversarial Generative Flow Network (AGFN) (Zhang et al., 2025), which builds on GFlowNet. We evaluate these methods on CVRP instances with $n = 100$, under the same setup as in Table 2, and report the results in Table 9. The results show that Active Search outperforms POMO (both greedy and sampling) but remains inferior to our method. In contrast, AGFN performs surprisingly poorly compared to POMO-greedy, despite using the same inference budget. Notably, the original AGFN paper also reports sub-par performance on instances with $n = 200$. While the method shows improvements on extremely large-scale problems, such settings are beyond the scope of our current study but represent a valuable direction for future work.

Table 9: Additional comparison of different inference methods on CVRP with $n = 100$

Method	Obj.	Gap
Active Search + IL	15.618	0.502%
AGFN (greedy)	15.873	2.142%
LGS (ours)	**15.524**	**-0.102%**

Comparison with gradient-based methods. Our initial motivation for using a continuous latent space was to gain the flexibility of applying continuous inference methods, particularly gradient-based techniques such as SGLD. The primary challenge in our setting is to obtain high-quality solutions within a limited computational budget. However, we found that computing gradients—especially backpropagating through the decoder in the latent space—is computationally expensive. This observation motivated us to adopt a sampling-and-learning approach instead.

In our experiments, all methods (including ours) were evaluated on batches of 1,000 problem instances. Gradient-based approaches, however, cannot process such large batches within GPU memory limits, requiring much smaller batch sizes and thereby reducing their practical efficiency. In contrast, our sampling-and-learning method updates only the decoder's final layer, and does so at fixed intervals rather than at every iteration, thereby avoiding costly end-to-end backpropagation. This design reduces computation time while preserving solution quality, underscoring the practical advantages of sampling-and-learning inference over fully gradient-based alternatives.

Extension to larger instances ($n = 200$). While our main experiments follow the CVAE-Opt setup with instance sizes up to $n = 150$, we additionally evaluate scalability on larger CVRP instances with $n = 200$. Using the same inference setup as for $n = 100$ (without the augmentation trick), we obtain the results reported in Table 10. LGS-Net remains competitive with strong baselines such as EAS and COMPASS on CVRP200.

Table 10: Experimental results on CVRP with $n = 200$ without the augmentation trick

Method	Obj.	Gap	Time
LKH3	22.00	0.00%	25H
POMO (greedy)	23.296	5.891%	1M
POMO (sampling)	23.371	6.232%	2H10
EAS	22.361	1.643%	2H10
COMPASS	22.455	2.069%	2H10
ELG	22.460	2.092%	2H10
CNF	23.389	6.314%	2H10
LGS-Net (ours)	**22.284**	**1.289%**	2H10

Scalability of our method. While our experiments are limited to $n \leq 200$, our method is in principle applicable to larger instances. The main computational cost arises from (i) propagating multiple latent particles during inference, and (ii) updating the final decoder layer via Stochastic Approximation. Both steps parallelize efficiently on GPUs, with the number of particles K and the update frequency remaining constant across scales. However, the decoding horizon T inevitably grows with instance size, a limitation shared by all constructive NCO methods. An interesting direction for future work is to design inference strategies that exploit problem structure by adapting the latent space at each decoding step, though this would incur significant computational cost.

B.2.3 Illustration of hyperparameters

Figure 5 illustrates solutions generated by our method, following a similar visualization style as in Perron & Furnon (2019); Kool et al. (2019). Visually, the solutions appear to be optimal.

To highlight the impact of important hyperparameters, we evaluate our inference method while varying the number of latent samples K, the latent space radius R, the latent dimension d_z, and the parameter update frequency.

Impact of the number of latent samples K. We first focus on the effect of the number of latent samples K. Figure 6 illustrates how the average gap evolves as K increases, and we observe that increasing K improves the performance of our method. This can be explained by the fact that the variance of the gradient estimator in (5) is of order $\mathcal{O}(1/K)$. However, beyond a certain threshold, the improvement becomes marginal. Choosing an appropriate value of K is therefore crucial to balance faster mixing against computational cost.

Sensitivity to latent dimension d_z and radius R. Since our latent space is, in practice, restricted to a ball of radius R, we study the impact of this parameter. Figure 7 illustrates the average optimality

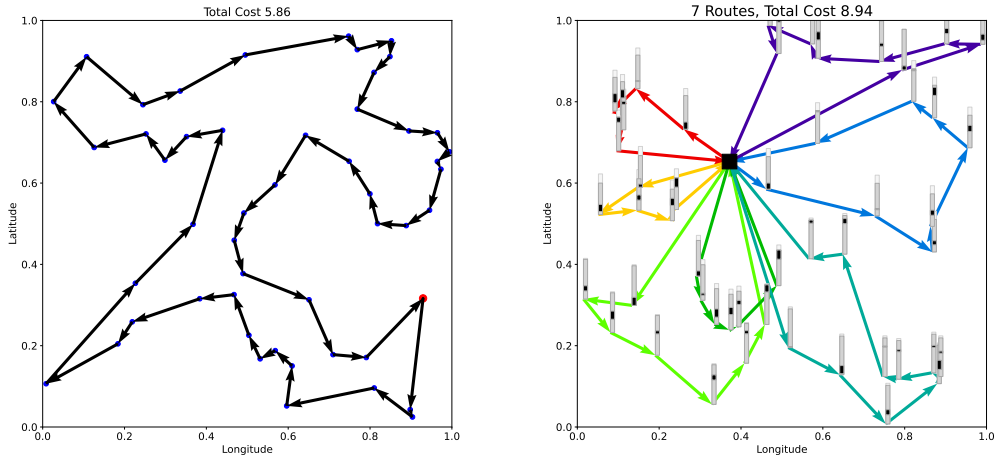

Figure 5: Solution representation produced by our model on the TSP (left) and CVRP (right) with $n = 50$. In the TSP plot, the red point denotes the starting node, and arrows indicate the visiting order. In the CVRP plot, the large square represents the depot, and each color corresponds to a distinct vehicle route. The bar illustrates vehicle capacity usage: black segments show the portion used by each customer, while white segments indicate unused capacity. The overall height of each bar reflects the total load on the corresponding route.

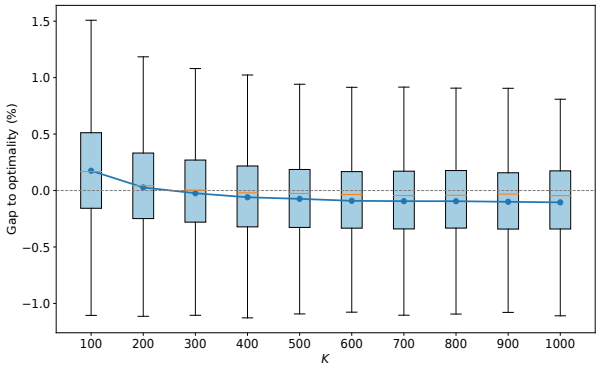

Figure 6: Average gap to optimality for different values of K in the CVRP with $n = 100$

gap for different values of R. For small values of R, the model performance degrades slightly because the latent space is too constrained. For large values of R, the performance also deteriorates slightly, as the latent space becomes too large and harder to exploit effectively. In contrast, moderate values of R strike a good balance and yield the best empirical performance. Nevertheless, across all tested values of R, the performance remains competitive.

Regarding the latent dimension, we train our model with the same configuration while varying the latent dimension $d_z \in \{2, 50, 100, 150\}$, and we use the same inference procedure and budget as in our other experiments on CVRP with $n = 100$. The results are reported in Table 11. As expected, the very low-dimensional model with $d_z = 2$ performs noticeably worse, since such a small latent space cannot capture the structure of routing problems. For $d_z = 50$, the model performs competitively but slightly worse than $d_z = 100$. Increasing the dimension further to $d_z = 150$ does not bring additional gains: performance degrades marginally, likely because the higher-dimensional latent space is harder to explore efficiently under a fixed inference budget.

Impact of the SA step frequency. We now discuss the SA step, focusing on how frequently the parameters should be updated. Our initial motivation for not updating the parameters at every

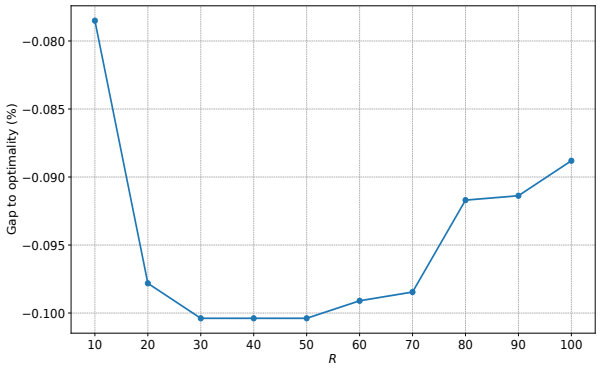

Figure 7: Average gap to optimality for different values of R in the CVRP with $n = 100$

Table 11: Effect of latent dimension d_z in our method on CVRP with $n = 100$

Method	Obj.	Gap
$d_z = 2$	16.189	4.176%
$d_z = 50$	15.529	-0.071%
$d_z = 100$	**15.524**	**-0.102%**
$d_z = 150$	15.543	0.019%

iteration is twofold: to reduce computational cost and to better explore the latent space given the current parameter distribution. Consequently, parameter updates can be performed at regular intervals rather than every iteration.

Since the initial parameters are typically far from optimal, allowing long exploration intervals early in training is often unnecessary. Instead, we begin with short exploration intervals and gradually increase them to enable more thorough exploration as learning progresses. Selecting these intervals is non-trivial; we chose them manually without extensive hyperparameter tuning. The update schedule used in our experiments is $[1, 1, 5, 15, 25, 100, 150]$. Optimizing this schedule remains an open question and presents an interesting direction for future work.

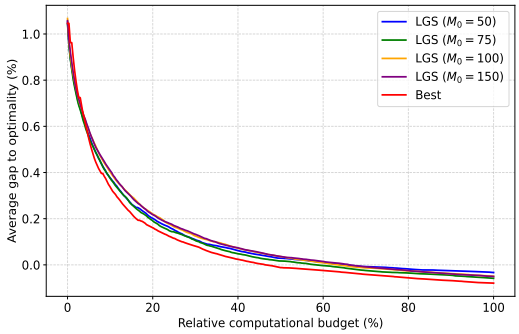

Figure 8: Performance comparison of various SA step update frequencies on CVRP with $n = 100$

Figure 8 illustrates the average gap to optimality for different choices of parameter update frequency, including both regular intervals and the increasing schedule. Here, M_0 denotes the update frequency, with $M_0 = 50$ indicating that parameters are updated every 50 iterations. We observe that while regular intervals produce similar results overall, $M_0 = 75$ yields slightly better performance. Notably, the increasing update schedule achieves a clear improvement over the fixed schedules, highlighting the potential benefits of adaptive strategies.

B.2.4 GENERALIZATION TO OTHER PROBLEMS

We now illustrate how our method can be extended beyond routing problems, using the classical 0/1 knapsack problem (Pisinger & Toth, 1998). Given a set of n items indexed by $i \in \{1, \dots, n\}$, each item is associated with a weight w_i and a value v_i, and the knapsack has a capacity D (we denote the capacity by D for consistency with our routing notation). The 0/1 knapsack problem consists in maximizing the total value of the selected items subject to the capacity constraint:

$$\min_{a \in \{0,1\}^n} \sum_{i=1}^{n} a_i v_i \ , \quad \text{subject to } \sum_{i=1}^{n} a_i w_i \leq D \ ,$$

where $a_i = 1$ indicates that item i is included in the knapsack and $a_i = 0$ otherwise.

We apply our LGS-net model with the encoder described in A.2.1, using input features $x_i = (w_i, v_i) \in \mathbb{R}^2$. The decoder follows the same architecture as the CVRP decoder in A.2.3 and keeps track of the remaining load capacity. The only difference lies in the termination condition: in the knapsack setting, an episode ends as soon as the remaining capacity reaches zero, whereas in the CVRP setting the decoder runs until all nodes have been visited.

We follow the experimental protocol of Bello et al. (2017). We generate two datasets, Knapsack50 and Knapsack100, each containing 1000 instances, with weights and values drawn independently and uniformly at random in $[0, 1]$. The knapsack capacity D is set to 12.5 for Knapsack50 and 25 for Knapsack100. We compare our inference method (LGS) with a sampling-based decoding from the same trained model (*Sampling*), OR-Tools (Perron & Furnon, 2019), and the optimal dynamic-programming solutions; the results are reported in Table 12. We observe that LGS matches the optimal solutions and OR-Tools on average, whereas the sampling-based method struggles to reach this optimal performance.

Table 12: Experimental results on knapsack instances (average objective value; higher is better).

Method	Sampling	LGS (ours)	OR-Tools	Optimal
Knapsack50	20.012	**20.018**	**20.018**	**20.018**
Knapsack100	40.501	**40.512**	**40.512**	**40.512**