LEARNING A DIFFUSION-NATIVE LATENT SPACE FOR NEURAL COMBINATORIAL OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion-based neural combinatorial optimization requires specifying how combinatorial solutions should be corrupted and subsequently denoised. Generic variable-wise noise ignores the structured geometry of feasible solutions, while search-oriented methods rely on task-specific perturbations such as masking or local-search moves, which can also be difficult to define when solutions lack a meaningful notion of perturbation. We instead learn a targeted space in which diffusion should operate. We introduce LCO-Diff, which uses an instance-conditioned variational autoencoder to transform discrete combinatorial solutions into a diffusion-native latent space. The latent representation is trained to remain sufficient for reconstructing the solution while being regularized toward a Gaussian geometry, so that standard continuous transport becomes native to the learned solution representation rather than being imposed on the original discrete space. In this latent space, we train an instance-conditioned rectified-flow model using a canonical Gaussian interpolation, yielding the same generative dynamics across different combinatorial problem classes. Denoised latents are decoded into solution scores and converted into feasible solutions via constraint-aware decoding and repair. The learned latent geometry also turns the training-time diffusion process into a natural test-time search mechanism. A candidate solution can be re-encoded into the latent space, partially renoised along the same Gaussian path used during training, and denoised again to generate improved candidates, without introducing a separate solution-space perturbation operator. Across problem sizes and inference budgets, our latent-space diffusion achieves state-of-the-art solution quality at low sampling cost and improves further with renoise-based test-time scaling.

1 INTRODUCTION

Combinatorial optimization (CO) problems are NP hard and underpin a wide range of applications in logistics [46], planning [31], and infrastructure [33]. Classical methods typically rely on carefully engineered algorithms, problem-specific heuristics, and extensive instance-wise search, whose design often requires substantial domain expertise. Instead of specifying the solution strategy entirely by hand, neural combinatorial optimization (NCO) [32; 5] learns reusable optimization patterns directly from collections of problem instances, amortizing part of the search effort into a neural solver [45; 21; 34; 22]. Once trained, such solvers can rapidly generate high-quality solutions, while automatically exploiting structural regularities present in the training distribution.

Among these approaches, generative models have shown promise. They separate learning a solution distribution during training from sampling and refining candidate solutions during inference. This decoupling provides a natural mechanism for allocating additional computation at test time [25; 27]. Denoising diffusion models [17; 41; 39] are particularly appealing in this setting because generation itself takes the form of iterative refinement from a corrupted state toward a structured solution. Following advances in diffusion over discrete states [2; 18], diffusion-based CO solvers such as DIFUSCO [42], T2T [25], and their successors [26; 38] have shown that this iterative generative process can provide both strong solution quality and a natural handle for test-time computation. More recent methods further turn denoising itself into a search mechanism [27; 7; 47].

Once denoising becomes an optimization mechanism, however, the corruption process becomes part of the search algorithm itself. It determines which intermediate states the model encounters, which neighborhoods of the solution space it learns to navigate, and ultimately which trajectories it must

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

learn to reverse. Existing approaches expose a fundamental tension. Generic discrete corruption, such as independently flipping or masking solution variables [2; 18], can be defined broadly but is largely agnostic to the global structure of a feasible combinatorial solution. Search-oriented methods therefore introduce more structured corruptions: for example, GenSCO [27] perturbs candidate solutions using randomized local-search operations and trains a rectified-flow model to regenerate them toward high-quality targets, while masking-based approaches remove selected solution components and learn to reconstruct them [7; 47]. Such structured perturbations can provide useful optimization trajectories, but they also require the solver designer to specify what a meaningful perturbation should be for the representation and constraints at hand. As solution structure becomes richer, e.g., when routes must simultaneously respect ordering, capacity, or time-window constraints, designing such perturbations becomes increasingly intertwined with problem-specific structure.

The difficulty may arise not only from how a solution is corrupted, but also from the space in which corruption is defined. Combinatorial solutions are represented by discrete and strongly constrained variables, whereas standard continuous diffusion is naturally formulated as smooth transport between distributions. Instead of repeatedly adapting the diffusion process to the geometry of each original solution representation, we ask whether the representation itself can be learned so that simple continuous generative dynamics become appropriate. We introduce Latent CO-Diffusion (LCO-Diff), which uses an instance-conditioned variational autoencoder to map an instance and its solution to a continuous node-level latent representation. The reconstruction objective encourages the latent variables to retain the information needed to recover the combinatorial solution, while variational regularization shapes their distribution toward a smooth and approximately Gaussian geometry. We refer to the resulting representation as diffusion-native because it is learned specifically to serve as the space in which a canonical Gaussian generative process operates.

By design, LCO-Diff is trained in two stages (Sec. 3, Fig. 1). In the first stage, we train an *instance-conditioned embedder*, instantiated as a variational autoencoder that maps an (instance, solution) pair to node-level latent vectors $\boldsymbol{z} \in \mathbb{R}^{n \times d}$ and decodes a latent back into a solution heatmap; it serves as a bidirectional interface between the discrete solution space and a smooth, approximately Gaussian latent space. In the second stage, we train an instance-conditioned *latent rectified-flow model*. The corrupted object is simply the canonical interpolation between standard Gaussian noise and the clean latent, and the network learns to recover the clean latent from the instance encoding. At inference, we start from a standard Gaussian latent, integrate a small number of denoising steps, decode the resulting latent into a heatmap, and repair it into a feasible solution with a lightweight greedy-insertion and local-search routine. This decomposition keeps the form of the corruption and generative dynamics shared across problem classes, while problem-specific feasibility requirements are handled at the decoding interface.

Since the noising process is the canonical Gaussian interpolation of *latent vectors* rather than of solutions, it inherits three desirable properties. First, it requires no problem-specific corruption heuristic, and unlike generic variable-level noise, its corruption acts on representations that have already absorbed the solution structure, so intermediate states can remain potentially adaptive by construction. Second, it applies uniformly to routing and other graph problems, including the Traveling Salesman Problem (TSP) and the Maximum Independent Set Problem (MIS), and extends naturally to structured routing settings such as CVRP and TSPTW, where solution-space noise is hard to define and existing solvers are rarely applied. Third, it makes test-time scaling *problem-agnostic*, as candidate solutions can be repeatedly re-encoded, re-noised, and re-denoised to generate improved solutions, without retraining or any hand-designed search operators (Sec. 3.3). Across these tasks, the same Gaussian corruption and rectified-flow formulation achieves strong solution quality with a small number of denoising steps, while additional renoise cycles consistently translate test-time computation into further improvements.

2 Preliminaries and Background

2.1 Related Work

Diffusion for CO. Denoising diffusion probabilistic models [17; 40; 41; 39] have been adapted to structured combinatorial data, from denoising over discrete states [2; 18] to diffusion-based solvers e.g. DIFUSCO [42], which formulates solutions as binary edge indicators and pairs the denoising model with a dedicated inference schedule. Its discrete formulation also outperforms diffusing a continuous space corrupted by Gaussian noise, and the same formulation underlies the strong re-

sults of the follow-up solvers T2T [25], Fast T2T [26] and COExpander [30]. GenSCO [27] instead returns to the continuous space, replacing the Gaussian source distribution with a disrupted solution and combining it with target disruption during training. [14] introduces a TSP-specific geometric representation where nodes are transported to a circular arrangement and the tour is recovered from their angular order. While it similarly seeks a more favorable space for generative dynamics, its geometry is prescribed specifically for TSP rather than learning from general combinatorial solutions.

Latent Space for CO. A complementary line of work relocates the search to a learned latent space, while solution construction is left to an autoregressive decoder. CVAE-Opt [19] compresses an (instance, solution) pair of a routing problem into one global latent vector with a conditional VAE, and expands that vector back into a solution autoregressively; at test time the latent is explored by differential evolution. COMPASS [6] places the latent inside an autoregressive reinforcement learning pipeline, where it injects stochasticity into construction, with different latent values inducing different specialized policies that are searched at inference time to adapt the solver to a given instance. In both designs, a global low-dimensional latent variable serves as a search or control variable for an autoregressive solution generator. In contrast, our proposed LCO-Diff departs from this design at both ends: its *per-node* latent representation preserves as much of the solution structure as reconstruction allows, and it is decoded directly into the solution heatmap, with no autoregressive stage to unfold it. We further employ a diffusion process to model and sample the latent solution distribution, rather than relying on an external optimizer to search the latent space.

Latent Space Diffusion. Variational autoencoders (VAEs) [20] learn a smooth, approximately Gaussian latent space through variational reconstruction, which makes latent-space sampling a natural interface for generative and denoising models. In vision, denoising diffusion has become the de-facto standard for high-quality generative modeling [35; 9], increasingly run *in the latent space* of a pretrained autoencoder, and rectified flow [28] provides a direct, stable objective for the linear interpolation between noise and data. Images, however, are rich in redundancy, so the autoencoder there acts mainly as a compressor, and the low-dimensional representation it returns directly relieves the denoiser of computation. [37] turn this observation into practice, showing that diffusion in the latent space is far more efficient than its pixel-space counterpart. [12] then combine the same recipe with rectified flow and scale it with transformer capacity, establishing the paradigm as both efficient and reliably scalable. Discrete CO offers no such redundancy, since every node of a solution is essential and a slight change in the node features may already describe a substantially different optimal solution. The space we build with a VAE therefore does not aim primarily to reduce the computation of the denoiser, but to reshape the discrete solution representation into a smoother, approximately Gaussian geometry that is better suited to continuous diffusion.

2.2 PRELIMINARIES

Rectified Flow. Rectified flow connects a source distribution to the data distribution through a time-dependent velocity field [28]. We use the convention that $t = 0$ corresponds to noise and $t = 1$ to data. For a clean latent $\boldsymbol{z}$ and Gaussian noise $\boldsymbol{\varepsilon}$, the interpolation is

$$\boldsymbol{z}_t \;=\; t\,\boldsymbol{z} + (1-t)\,\boldsymbol{\varepsilon}, \qquad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), \tag{1}$$

where t is sampled uniformly from $[0, 1]$ during training. The velocity along this path is $\boldsymbol{z} - \boldsymbol{\varepsilon} = (\boldsymbol{z} - \boldsymbol{z}_t)/(1-t)$, so predicting the clean latent vector is an affine reparameterization of the velocity field, and a denoiser $f_{\boldsymbol{\theta}}(c, t, \boldsymbol{z}_t)$ conditioned on instance features c is trained under the resulting rectified-flow objective to do exactly that. Sampling integrates the induced ODE from a Gaussian draw $\boldsymbol{z}_0 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ at $t = 0$ toward $t = 1$. This is the concrete instance of the generative process of Sec. 2.2: $\mathrm{Sample}_{\boldsymbol{\theta}}$ is a high-order multi-step ODE scheme on a linear timestep grid of K steps, a setting that reaches high quality in few steps.

Latent Diffusion. Latent diffusion runs the diffusion process in the representation space of an autoencoder instead of in the raw data space. An encoder maps a datum $\boldsymbol{u}$ to a latent $\boldsymbol{z} = \mathrm{Enc}_{\varphi}(\boldsymbol{u}) \in \mathbb{R}^{n \times d}$ and a decoder maps a latent back to a reconstruction $\hat{\boldsymbol{u}} = \mathrm{Dec}_{\varphi}(\boldsymbol{z})$; the two are trained by reconstruction plus a regularizer that keeps the latent space smooth and close to Gaussian, so that nearby latents decode to similar outputs and a simple prior can cover the space [20]. On top of that space, a denoiser $f_{\boldsymbol{\theta}}(c, t, \boldsymbol{z}_t)$, conditioned on side information c, is trained to invert a Gaussian corruption of a latent, either as a noise predictor [17] or as the drift of a deterministic ODE sampler [39; 41]; Sec. 2.2 states the objective we use. Time is indexed so that $t = 0$ is pure noise and

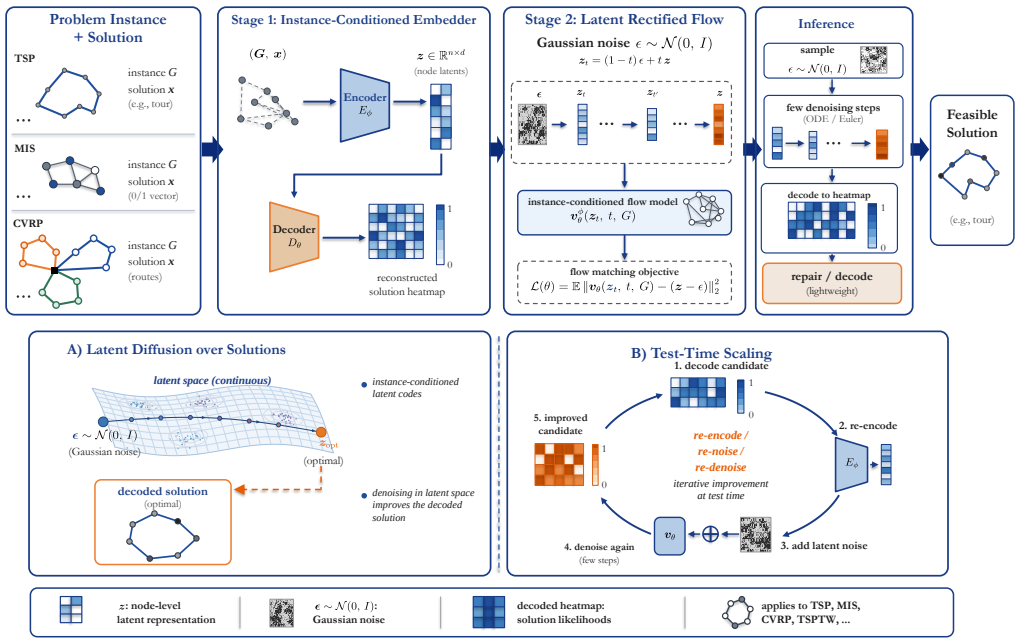


Figure 1: Overview of LCO-Diff. *Stage 1*: the instance-conditioned VAE embedder is trained to map (instance, solution) pairs to node-level latent vectors and to decode latent vectors back to solution heatmaps, with a reconstruction objective plus KL regularization. *Stage 2*: a rectified-flow model is trained directly in that latent space, so generation starts from standard Gaussian noise and no problem-specific corruption process is required. *Inference*: a few solver steps give latent vectors that are decoded into a solution heatmap and turned into a feasible solution by lightweight constraint-aware repair. The lower panels show latent diffusion over solutions (left), a continuous trajectory in latent space from Gaussian noise to the optimal latent code and its decoded tour, and test-time scaling (right), where a candidate is re-encoded, re-noised, and denoised again for improvement.

$t = 1$ is data, and $\mathrm{Sample}_{\boldsymbol{\theta}}(\boldsymbol{\varepsilon}, c)$ denotes the latent returned by running the denoiser for K solver steps from the noise $\boldsymbol{\varepsilon}$ on a grid in $[0, 1]$ [35; 9].

With this operator the whole recipe is two lines, one fitted and one run:

$$\begin{aligned} \text{training:}\quad & \boldsymbol{u} \longrightarrow \boldsymbol{z} = \mathrm{Enc}_{\varphi}(\boldsymbol{u}) \longrightarrow f_{\boldsymbol{\theta}} \text{ trained by diffusion toward the target } \boldsymbol{z}, \\ \text{inference:}\quad & \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \longrightarrow \hat{\boldsymbol{z}} = \mathrm{Sample}_{\boldsymbol{\theta}}(\boldsymbol{\varepsilon}, c) \text{ (denoising)} \longrightarrow \hat{\boldsymbol{u}} = \mathrm{Dec}_{\varphi}(\hat{\boldsymbol{z}}) \text{ (decoding)}. \end{aligned} \tag{2}$$

A datum enters the model only through the first step of training, where the encoder output *is* the diffusion target, while inference never sees the data at all and starts from a fixed Gaussian prior. Inference is the same three stages for every modality: sample in latent space by denoising, decode, then apply whatever task-specific post-processing turns a decoded output into a valid object.

3 Methodology: The Latent CO-Diffusion

Overview. Fig. 1 summarizes LCO-Diff. Our design separates two roles that are entangled when diffusion is performed directly on combinatorial solutions. The first stage learns how a discrete solution should be represented for continuous generative modeling, while the second stage learns the generative dynamics on that representation. The resulting latent variables remain aligned with the problem nodes, preserving the structure needed for direct solution reconstruction, but their distribution is regularized toward the Gaussian source used by the flow model. Consequently, the diffusion model does not need to define a separate notion of solution corruption for each problem class. Task-specific feasibility is instead handled after generation by the decoding interface.

The two stages are trained sequentially. We first optimize the instance-conditioned embedder and then freeze it to define a fixed latent target distribution for the rectified-flow model. At inference time, the encoder is not required for unconditional generation from noise and is used only when an

existing candidate is re-encoded for renoise-based refinement. The following sections describe the learned representation, the latent generative model, and the resulting test-time scaling procedure.

3.1 Instance-Conditioned Embedder

Rather than compress the entire solution into a single global code, we adopt the node-level organization, where each latent vector remains aligned with one combinatorial element of the instance. This preserves the structural resolution required for direct prediction of the solution heatmap and avoids requiring an autoregressive decoder to unfold a global latent representation. The purpose of the VAE is therefore not primarily for dimensionality reduction. Instead, it reparameterizes the discrete solution into a continuous one while retaining the local organization required by the decoding head.

The instance is the conditioning input of both stages, while the solution plays two other roles: it is the datum the encoder embeds and the target the decoder reconstructs. The solution therefore reaches the network through the encoder alone. The denoiser of Sec. 3.2 never receives it, and a plain generative pass never requires it; the encoder is driven by a solution again only in the re-encoding step of the renoise cycle (Sec. 3.3), where that solution is a candidate the solver produced itself rather than a supervision target. The embedder minimizes a β-VAE objective:

$$\mathcal{L}^{\mathrm{E}} = \mathbb{E}_{\boldsymbol{x},\boldsymbol{s},\boldsymbol{\varepsilon}}\left[\underbrace{\mathcal{L}_{\mathrm{rec}}\big(\mathrm{Dec}_{\varphi}(\boldsymbol{z}), A_{\boldsymbol{s}}\big)}_{\text{cross-entropy on solution logits}} + \beta \underbrace{D_{\mathrm{KL}}\big(q_{\varphi}(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{s}) \,\|\, \mathcal{N}(\mathbf{0}, \boldsymbol{I})\big)}_{\text{mean over nodes}}\right], \qquad \boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\varepsilon}, \tag{3}$$

with $\beta = 1$ by default, and $\beta = 0.5$ on MIS. The two terms are complementary.

The reconstruction term is the cross-entropy between the decoded solution logits and the incidence $A_{\boldsymbol{s}}$, under the logit head of the problem at hand; it makes the latent *sufficient*, so that the latent vectors alone carry everything needed to reproduce the given solution and the decoding head can turn them back into an accurate solution heatmap. The KL term shapes the latent *geometry*: it bounds the latents, removes arbitrary per-node scaling, and keeps the encoded solutions close to the standard Gaussian that the generative process of Sec. 3.2 starts from, so the denoiser has only a mild deviation to learn rather than an unconstrained latent distribution. Reconstruction alone would admit latents that vary wildly across instances and are hard to generate, while the KL term alone would discard the solution; β sets the trade-off between solution fidelity and a latent space the flow model can cover. This term is also what makes the asymmetry above safe to exploit: the decoding head is fitted on latents produced by an encoder that has seen the solution, whereas at inference the latents come from a denoiser that has not, and pulling both toward $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ keeps the two sources on one manifold so that the head treats them identically.

Algorithm 1: Training the latent denoiser

Input: minibatch $\{(\boldsymbol{x}, \boldsymbol{s})\}$; frozen embedder Enc_{φ}; instance encoder Enc_{ψ}; denoiser $f_{\boldsymbol{\theta}}$

$\boldsymbol{z} \leftarrow \boldsymbol{\mu}_{\varphi}(\boldsymbol{x}, A_{\boldsymbol{s}})$;
$\boldsymbol{c} \leftarrow \mathrm{Enc}_{\psi}(\boldsymbol{x})$;
draw $t \sim \mathcal{U}(0, 1)$ and $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, per example;
$\boldsymbol{z}_t \leftarrow t\,\boldsymbol{z} + (1 - t)\,\boldsymbol{\varepsilon}$;
$\mathcal{L}^{\mathrm{D}} \leftarrow \| f_{\boldsymbol{\theta}}(\boldsymbol{c},\, t,\, \boldsymbol{z}_t) - \boldsymbol{z} \|_F^2$;
update $\boldsymbol{\theta}$ by gradient descent;

Together, the two terms make the embedder an interface between two representations with different requirements. The discrete solution space is suitable for evaluating feasibility and objective quality, while the latent representation is optimized for reconstruction and continuous generative modeling. The reconstruction loss preserves solution structure, and the KL regularizer encourages a latent distribution compatible with Gaussian interpolation. This is the sense in which we use the term *diffusion-native*. The representation is learned specifically so that the subsequent generative model can operate with a simple, shared corruption rule.

3.2 Latent-Space Diffusion

Once the embedder is fixed, each training solution induces a deterministic node-level target representation through the posterior mean $\boldsymbol{z} = \boldsymbol{\mu}_{\varphi}(\boldsymbol{x}, A_{\boldsymbol{s}})$. The second stage treats these encoded solutions as the data distribution to be modeled conditionally on the problem instance. The task of the denoiser is therefore not to predict combinatorial decisions directly, but to generate a latent configuration that the frozen decoder interprets as a high-quality solution. The latent denoiser f_{θ} is a transformer conditioned on the instance features; the noisy latent vector is injected by adding a learned linear projection of $\boldsymbol{z}_t$ to the projected instance features, together with a sinusoidal embed-

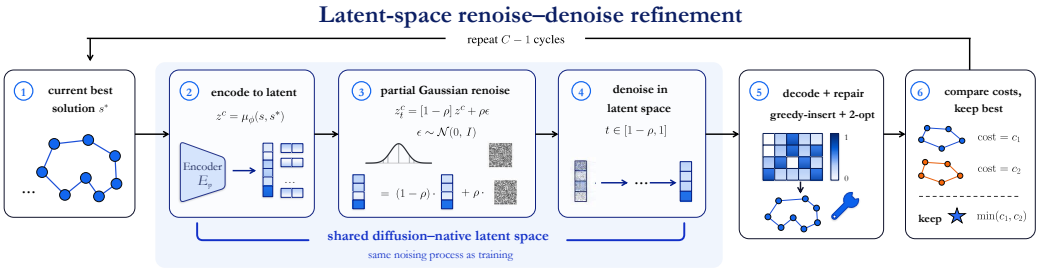


Figure 2: The renoise cycle for test-time scaling. *Step 1*: the best candidate found so far $\boldsymbol{s}^\star$ is the current solution. *Steps 2 to 4*, all inside the shared latent space: the frozen embedder maps $\boldsymbol{s}^\star$ to node-level latents $\boldsymbol{z}^\star$, those latents are partially renoised toward the Gaussian prior with factor $\rho \in (0, 1)$, and the denoiser integrates only the remaining interval $t \in [1-\rho, 1]$. *Step 5*: the resulting latents are decoded to a heatmap and repaired by greedy insertion together with the problem-adapted local search. *Step 6*: the two candidates are compared on cost and the better carries over to the next cycle, repeating $C-1$ times. Because the corruption of step 3 is the same Gaussian noising process used in training, renoising is problem-agnostic and requires no dedicated search operator.

ding of t. Training is the rectified-flow objective of Sec. 2.2, applied to *latent vectors*:

$$\mathcal{L}^{\mathrm{D}}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x},\boldsymbol{z},t,\boldsymbol{\varepsilon}}\left[\left\| f_\theta\big(\mathrm{Enc}_\psi(\boldsymbol{x}),\, t,\, t\,\boldsymbol{z} + (1-t)\,\boldsymbol{\varepsilon}\big) - \boldsymbol{z}\right\|^2\right], \qquad t \sim \mathcal{U}(0,1), \tag{4}$$

i.e. the corruption in Eq. 1 is the standard Gaussian interpolation of Sec. 2.2 carried out on latents, and the regression on the clean latent is the affine reparameterization of the velocity field noted there.

Alg. 1 lists one training step: the solution enters only through the frozen embedder, so the instance encoder never sees the target and every step asks the network to undo the same Gaussian corruption that sampling starts from. Since the VAE makes the latent space amenable to standard diffusion by construction, neither a problem-specific disruption heuristic nor a custom source distribution of the kind adopted by GenSCO is required.

Table 1: Results on TSP. C: renoise/search cycles; AS: active search, G: greedy, S: sampling, BS: beam search, aug.: augmentation.

Method	TSP-500 (128 inst.) Obj.	Gap	Time	TSP-1000 (128 inst.) Obj.	Gap	Time
Heuristics						
Concorde [1]	16.55	0.000%	39.8m	23.12	0.000%	3.0h
LKH-3 [16] (trials=64)	16.55	0.015%	50s	23.12	0.025%	2.5m
LKH-3 [16] (trials=256)	16.55	0.006%	1.6m	23.12	0.009%	4.1m
LKH-3 [16] (trials=4096)	16.55	0.000%	15.0m	23.12	0.001%	36.3m
Neural Solvers						
DIMES [36] (AS+S+2-opt)	17.29	4.48%	2.1h	24.32	5.17%	4.5h
LEHD [29] (PRC 100)	16.61	0.34%	8.0m	23.44	1.22%	43.0m
LEHD [29] (PRC 1000)	16.58	0.17%	1.2h	23.32	0.72%	7.0h
BQ-NCO [10] (BS)	16.64	0.55%	15m	23.47	1.38%	38m
GLOP [48]	16.91	1.99%	1.5m	23.84	3.11%	3.0m
UDC [50]	16.78	1.58%	4.0m	23.53	1.78%	8.0m
COExpander [30]	16.59	0.25%	1.41m	23.27	0.64%	5.2m
Discrete Diffusion Solvers						
DIFUSCO (T_s=50) [42] (G+2-opt)	16.80	1.50%	4.7m	23.55	1.89%	14.4m
DIFUSCO (T_s=100) [42] (S+2-opt)	16.69	0.87%	19.1m	23.42	1.31%	51.9m
T2T (T_s=50,T_g=30) [25] (G+2-opt)	16.68	0.82%	6.5m	23.44	1.40%	19.7m
T2T (T_s=50,T_g=30) [25] (S+2-opt)	16.63	0.48%	19.7m	23.37	1.07%	51.1m
Fast T2T (T_s=5,T_g=5) [26] (G+2-opt)	16.61	0.39%	2.2m	23.25	0.58%	8.6m
Fast T2T (T_s=5,T_g=5) [26] (S+2-opt)	16.58	0.21%	6.9m	23.22	0.42%	18.3m
Continuous Diffusion Solvers						
GenSCO [27] (C=10, 2-opt)	16.55	0.019%	5s	23.13	0.063%	16s
GenSCO [27] (C=40, 2-opt)	16.55	0.012%	18s	23.13	0.046%	58s
GenSCO [27] (C=160, 2-opt)	16.55	0.010%	1.2m	23.13	0.036%	3.9m
Latent-Space Diffusion Solvers						
LCO-Diff (C=8)	16.547	0.0065%	8s	23.126	0.0348%	15s
LCO-Diff (C=32)	16.546	0.0018%	21s	23.123	0.0199%	46s
LCO-Diff (C=128)	16.546	0.0007%	1.2m	23.121	0.0130%	2.8m
LCO-Diff (C=512)	16.546	0.0004%	4.7m	23.120	0.0080%	11.0m

3.3 Native Test-Time Scaling via Gaussian Renoising

The latent model gives test-time scaling a simple and uniform way across problems. The first decoding pass yields an initial solution, then each cycle (i) encodes the best candidate so far, $\boldsymbol{s}^\star$, to $\boldsymbol{z}^\star = \mu_\varphi(\boldsymbol{x}, A_{\boldsymbol{s}^\star})$, (ii) *partially renoises* it as $\boldsymbol{z}^\star_\rho = (1-\rho)\,\boldsymbol{z}^\star + \rho\,\boldsymbol{\varepsilon}$ with factor $\rho \in (0, 1)$, and (iii) resumes the denoiser on the remaining interval $t \in [1-\rho, 1]$, then decodes and repairs the result. Since $\boldsymbol{z}^\star_\rho$ lies on the same Gaussian interpolation that the denoiser is trained on, it should remain in distribution across a wide range of renoise factors, and the cycle carries over to every problem unchanged. It is sketched in Fig. 2 and Alg. 3

(appendix), and composes directly with best-of-R sampling, which keeps the best of R independent runs under different noise seeds. The contrast with GenSCO [27] is sharpest precisely here: its generation-as-search requires a dedicated search operator, where a hand-crafted disruption plays the role that Gaussian noise plays in our formulation, whereas our cycle reuses the training-time noising process verbatim within the learned latent space and introduces no new mechanism. As Sec. 4 shows, this native noising alone matches or surpasses such purpose-built test-time scaling schemes on every problem, with no per-problem operator or schedule tuning.

4 EXPERIMENTS

We evaluate the proposed LCO-Diff on TSP, MIS, as well as CVRP and TSPTW, two problem classes that prior solution-space diffusion solvers struggle to handle.

Task and Data. The TSP and MIS data configurations are listed in Table 7. *TSP:* uniform instances in the unit square with $n \in \{500, 1000\}$, 128K training / 128 test instances for TSP-500 and TSP-1000, referred to Concorde [1]. *MIS:* RB graphs [49] with $n \in [200, 300]$ (90K training, 500 test graphs) and Erdős–Rényi (ER) graphs [11] with $n \in [700, 800]$ and edge probability 0.15 (163,840 training graphs, test set of [36]), referenced to KaMIS [23]. *Routing:* CVRP-100 with uniform unit-square customers, demands uniform in $\{1, \ldots, 9\}$ and vehicle capacity 50 [21], referenced to HGS [44]; TSPTW-50 in the medium and hard time-window tightness, referred to LKH-3 [16], with both configurations listed in Table 8.

Settings. The embedder is an instance conditioned VAE of Sec. 3.1 (β=1, and β=0.5 on MIS), and the denoiser is trained with the rectified-flow objective of Sec. 3.2.

For the VAE, we uniformly adopt a 6-layer encoder and a 6-layer decoder, both with a hidden dimension of 256, and set the latent dimension to 128. For the denoiser configuration, we follow GenSCO [27] on TSP and MIS, and for CVRP and TSPTW we use the same network configuration as for TSP, with only the input dimension differing. The configurations and hyperparameters are tabulated in Appendices D and E. The test-time budget is set by the number of renoise cycles C (Sec. 3.3). Experiments are conducted on a NVIDIA RTX 4090 GPU unless otherwise specified.

Table 2: Results on MIS. G: greedy, S: sampling, TS: tree search, SL/RL/UL: supervised, reinforcement and unsupervised learning; T_s/T_g: sampling / gradient-search steps; C: search cycles.

Method	Type	RB-[200-300] Obj.	RB-[200-300] Gap	RB-[200-300] Time	ER-[700-800] Obj.	ER-[700-800] Gap	ER-[700-800] Time
Heuristics & Exact Solvers							
KaMIS [23]	Heuristics	20.10	–	1.4h	44.87	–	52.1m
Gurobi [15]	Exact	19.98	0.01%	47.6m	41.28	7.78%	50.0m
Discrete Diffusion Solvers							
DIFUSCO (T_s=100) [42]	SL+G	18.52	7.81%	16.1m	37.03	18.53%	5.5m
DIFUSCO (T_s=100) [42]	SL+S	19.13	4.79%	20.5m	39.12	12.81%	21.7m
T2T (T_s=50,T_g=30) [25]	SL+G	18.98	5.49%	21.0m	39.81	11.28%	7.1m
T2T (T_s=50,T_g=30) [25]	SL+S	19.38	3.53%	30.3m	41.41	7.72%	27.8m
Fast T2T (T_s=5,T_g=5) [26]	SL+G	19.49	2.89%	4.7m	40.68	9.34%	1.5m
Fast T2T (T_s=5,T_g=5) [26]	SL+S	19.70	1.90%	7.0m	41.73	6.99%	5.9m
Continuous Diffusion Solvers							
GenSCO (C=500)	SL+G	19.85	1.24%	1.1m	42.46	5.37%	40s
GenSCO (C=1k)	SL+G	19.91	0.96%	2.2m	42.91	4.38%	1.3m
GenSCO (C=2k)	SL+G	19.97	0.64%	4.4m	43.33	3.44%	2.7m
GenSCO (C=4k)	SL+G	20.01	0.43%	8.8m	43.77	2.46%	5.3m
GenSCO (C=8k)	SL+G	20.04	0.29%	17.6m	44.13	1.66%	10.7m
GenSCO (C=16k)	SL+G	20.06	0.22%	35.2m	44.41	1.03%	21.4m
Latent-Space Diffusion Solvers							
LCO-Diff (C=4)	SL+G	19.87	1.12%	14s	42.79	4.64%	22s
LCO-Diff (C=16)	SL+G	20.01	0.43%	51s	43.82	2.35%	1.4m
LCO-Diff (C=64)	SL+G	20.08	0.12%	3.3m	44.51	0.81%	5.4m
LCO-Diff (C=256)	SL+G	**20.09**	**0.06%**	13.1m	**44.95**	**−0.17%**	21.3m

Baselines. We compare four groups of methods, the first three being the closest to our formulation: *(1) discrete diffusion solvers*, DIFUSCO [42], T2T [25] and Fast T2T [26]; *(2) continuous diffusion solvers*, GenSCO [27]; *(3) latent-space solvers*, CVAE-Opt [19] and COMPASS [6]; and *(4) other representative neural solvers*. Some baselines may appear only for a subset of the problems, for two main reasons: i) the problem scope is limited; for example, GenSCO and Fast T2T cannot handle CVRP or TSPTW; ii) the problem is supported but the problem scale is limited; for example, CVAE-Opt and COMPASS can handle TSP but struggle to scale to sizes such as 500 to 1000.

Metrics. 1) *Objective* (Obj.): mean tour length on TSP, mean routing cost on CVRP/TSPTW, and mean independent-set size on MIS; 2) *Gap*: relative gap (%) to the reference solver named above, a negative gap meaning that we beat the reference; 3) *Time*: wall-clock time to solve the whole test set on a single GPU with batched inference. On TSPTW we additionally report the *infeasible rate*, the fraction of instances whose decoded solution violates a time-window constraint, with Obj. and Gap

computed over feasible instances, following [3]. Results of our LCO-Diff are averaged over the test set, and over three seeds for TSP, MIS, CVRP and TSPTW.

4.1 Main Results

TSP Results. Table 1 evaluates TSP, where the solution-space diffusion solver is strongest. On TSP-500, LCO-Diff shows a clear advantage over the solution-space baseline [27]: even the smallest budget $C{=}8$ already improves on its strongest setting (0.0065% against 0.010%) at about a ninth of its wall clock, and the gap to the Concorde reference keeps shrinking to 0.0004% as C grows, with $C{=}128$ already ahead of the 256-trial LKH-3 result in less time (0.0007% in 1.2m versus 0.006% in 1.6m). On TSP-1000, the same cycle starts below that GenSCO setting at $C{=}8$ and reaches 0.0080% at $C{=}512$, and $C{=}128$ already improves on LKH-3 with 64 trials at a comparable cost (0.0130% in 2.8m versus 0.025% in 2.5m); extra budget keeps improving on both sizes.

Table 3: Results on CVRP-100. A: attempts; I: iterations.

Method	CVRP-100 (1,280 inst.) Obj.	Gap	Time
Heuristics			
HGS [44]	15.550	0.000%	13h
Latent-Space Solvers			
CVAE-Opt [19] (I=20)	16.278	4.69%	1.2h
CVAE-Opt [19] (I=40)	16.147	3.85%	2.4h
CVAE-Opt [19] (I=80)	16.030	3.09%	4.7h
COMPASS [6] (A=400)	15.651	0.65%	5.0m
COMPASS [6] (A=800)	15.639	0.58%	9.7m
COMPASS [6] (A=1600)	15.632	0.53%	18.9m
Latent-Space Diffusion Solvers			
LCO-Diff (C=8)	15.625	0.49%	12s
LCO-Diff (C=32)	15.595	0.29%	35s
LCO-Diff (C=128)	15.577	0.18%	2.1m
LCO-Diff (C=512)	**15.567**	**0.11%**	8.2m

MIS Results. Table 2 evaluates MIS, a pure graph problem without coordinates, where a solution-space disruption has no geometric handle and feasibility is recovered entirely by decoding. On RB, LCO-Diff shows a clear advantage over the discrete diffusion baselines: even the smallest budget $C{=}4$ improves on the strongest published setting under sampling, Fast T2T [26], with a $1.7\times$ smaller gap at a $30\times$ lower cost, and the gap to the KaMIS [23] reference keeps shrinking to 0.06% as C grows, a quality GenSCO reaches only with $250\times$ more cycles (51s versus 8.8m at the 0.43% level). On ER, LCO-Diff reaches -0.17% at $C{=}256$, a mean set size above the published reference mean, whereas the discrete diffusion baselines stay above a 6.9% gap and GenSCO remains 1.03% short of the reference at an essentially identical wall clock (21.4m versus 21.3m).

CVRP & TSPTW Results. Tables 3 and 4 evaluate routing, the problem class that the solution-space paradigm cannot natively express. On CVRP-100, LCO-Diff shows a clear advantage over the latent-space baselines: even the smallest budget $C{=}8$ already surpasses the best COMPASS and CVAE-Opt settings at a fraction of their wall clock, and the gap to the HGS reference keeps shrinking to 0.11% as C grows. On TSPTW-50, LCO-Diff stays below the LKH-3 reference in both regimes, whereas the constructive neural baselines remain above the reference with substantial infeasible rates; extra budget further drives the infeasible rate down.

Table 4: Results on TSPTW-50 under medium and hard time-window tightness. *: quotations from [3; 4]. Rate: Infeasible rate.

Method	TSPTW-50 (Medium time tightness) Obj.	Gap	Rate	Time	TSPTW-50 (Hard time tightness) Obj.	Gap	Rate	Time
Heuristics								
LKH-3* [16]	13.020	0.00%	0.00%	7h	25.611	0.00%	0.12%	7h
Neural Solvers								
AM+PIP* [21; 3]	13.68	5.06%	0.35%	11m	25.71	0.38%	1.98%	11m
AM+PIP-D* [21; 3]	13.65	4.87%	0.33%	11m	25.80	0.67%	4.40%	11m
POMO+PIP* [22; 3]	13.40	2.91%	0.90%	15s	25.66	0.18%	2.67%	15s
POMO+PIP-D* [22; 3]	13.45	3.32%	0.65%	15s	25.69	0.28%	3.07%	15s
Latent-Space Diffusion Solvers								
LCO-Diff (C=4)	12.966	−0.41%	0.16%	25s	25.513	−0.38%	1.91%	25s
LCO-Diff (C=16)	12.920	−0.76%	0.05%	1.4m	25.513	−0.38%	1.78%	1.4m
LCO-Diff (C=64)	**12.885**	**−1.03%**	**0.01%**	5.5m	**25.512**	**−0.38%**	**1.68%**	5.5m

4.2 Effectiveness of Learned Latent Space

We isolate the contribution of the learned latent space by fixing the denoising backbone, training data, and inference budget, and comparing only the space in which diffusion is performed. To avoid attributing the gains to a stronger architecture or objective, we first upgrade the raw-space diffusion baseline to the same setup as Sec. 2.2.

Table 5: Diffusion in the raw solution space versus diffusion in the learned latent space.

Method	C=8	C=64	C=256
TSP-500			
Raw-space diffusion	0.0665%, 10s[1]	0.0294%, 32s	0.0140%, 2.0m
LCO-Diff	**0.0065%**, 8s	**0.0012%**, 38s	**0.0006%**, 2.4m
MIS-RB-[200-300]			
Raw-space diffusion	7.03%, 23s	4.36%, 3.1m	3.42%, 12.3m
LCO-Diff	**0.75%**, 26s	**0.12%**, 3.3m	**0.06%**, 13.1m

[1] In this setting raw-space diffusion is slower than latent-space one because its diffusion step yields lower-quality solutions, so the subsequent 2-opt needs more iterations to converge, which increases CPU overhead.

Improved Solution-Space Baseline. We take the solution-space continuous diffusion described in DIFUSCO [42] as the starting point and bring it up to the setup of Sec. 2.2, so that the space in which the diffusion runs is the only remaining difference. Particularly, Its GNN denoiser by the transformer backbone of LCO-Diff, and its L_2 loss by a cross-entropy loss over the categorical solution variables. The upgraded baseline is a strong solver (comparable with recent solvers like GenSCO and Fast-T2T on TSP-500).

Analysis. Table 5 shows that LCO-Diff achieves significantly lower optimality gaps across all C. For each problem, the gap is reduced by more than a factor of **8** in all settings, and the advantage becomes more pronounced as C increases. These gains require comparable or slightly higher inference time, confirming the learned latent space is more effective without sacrificing efficiency.

4.3 β Sensitivity Analysis

To evaluate the influence of the parameter β, For each task we retrain the embedder at half and at twice the value used in Sec. 4.1 ($\beta \in \{0.5, 2\}$ around the default β=1 on CVRP, and $\beta \in \{0.25, 1\}$ around the default β=0.5 on MIS), and we retrain the denoiser from scratch on the latents of each embedder, so that every column of Table 6 is a complete model rather than a decoding head transplanted onto a foreign latent space.

Table 6: Sensitivity to the β for VAE training at matched training and inference budgets.

CVRP-100, gap (%)↓						
β	C=8	16	32	64	128	256
0.5	**0.44**	**0.32**	**0.24**	**0.18**	**0.14**	**0.11**
1 (default)	0.49	0.38	0.29	0.23	0.18	0.14
2	0.49	0.37	0.30	0.23	0.18	0.14
MIS-RB-[200-300], gap (%)↓						
β	C=8	16	32	64	128	256
0.25	0.70	0.36	**0.22**	**0.11**	**0.05**	**0.05**
0.5 (default)	0.75	0.43	0.24	0.12	0.10	0.06
1	**0.59**	**0.34**	0.24	0.13	0.07	0.06

Quality is Flat over a $4\times$ Range of β. Taking the three rows of each block together, the best and the worst setting never differ by more than 0.06 percentage points on CVRP, and on MIS by 0.16 points at the smallest budget and by at most 0.05 points from C=32 on. This robustness to β can be attributed to the use of per-node latents rather than a global latent, which provides sufficient expressive capacity. Note that the default serves as an untuned operating point rather than a best-performing choice.

4.4 Latent Space Unlocks Efficient Native Test-Time Scaling

To achieve effective test-time scaling, GenSCO [27] customizes a source distribution and hand-crafts a disruption for each problem. In contrast, our renoise cycle of Sec. 3.3 reuses the same Gaussian noising as during training and introduces no problem-specific operator. As shown in Fig. 3, in latent space, this native renoising scales effectively across both problems: the optimality gap continues to shrink as the runtime budget grows, achieving lower optimality gaps than both baselines while requiring only a fraction of their runtime.

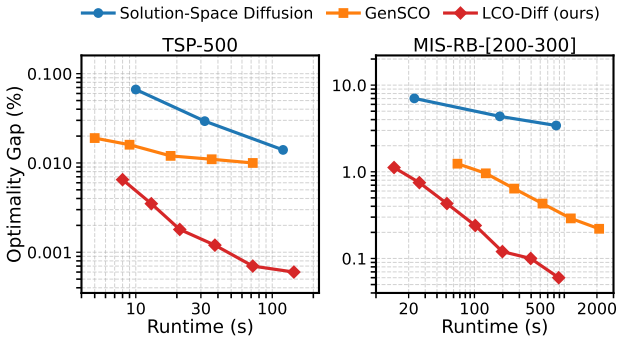


Figure 3: Scaling curves on TSP-500 and MIS-RB.

5 Conclusion

We have proposed to learn a latent-space for diffusion-based CO in which the noising process is the standard Gaussian interpolation of node-level solution latents rather than a hand-crafted solution-space disruption heuristic. An instance-conditioned VAE embedder provides the latent space and the decoding head; a rectified-flow denoiser generates solutions in that space from pure Gaussian noise in a small number of steps; and encode–renoise–denoise cycles give a native test-time scaling procedure. Our LCO-Diff uses a shared corruption and generative dynamics across problem classes. Our experiments show that latent-space diffusion attains competitive quality at a fraction of the sampling budget of the solution-space baseline, and that the renoise cycle converts additional test-time compute into further quality gains without any new operator or training.

AI USE STATEMENT

AI tools were used to support or refine the writing. In particular, they assisted with language editing, grammar and spelling checks, improving clarity and readability, and polishing the academic style. The authors reviewed, verified, and revised all AI-generated suggestions and assume full responsibility for the final content.

AI tools were also employed to draft portions of the paper. More specifically, AI assistance helped prepare preliminary outlines for the Methodology section; these outlines were then reviewed and rewritten by the authors.

ETHICS STATEMENT

This study was carried out in line with the ICLR Code of Ethics and with a firm commitment to responsible research conduct. It does not involve human participants, sensitive personal data, or applications that could cause harm. Throughout the project, we have aimed to uphold transparency, reproducibility, and scientific integrity, including by accurately reporting our methods and findings. We have no conflicts of interest related to funding sources or affiliations that might affect this work. Our goal is for this research to benefit the machine learning community and society at large by advancing fairness, accessibility, and open scientific inquiry. Should any unforeseen broader impacts emerge, we remain committed to addressing them in a responsible manner.

REPRODUCIBILITY STATEMENT

We are committed to upholding high standards of scientific excellence and transparency in accordance with the ICLR Code of Ethics. To ensure the reproducibility of our results, the main text gives a complete description of the methodology, namely the instance-conditioned embedder (Sec. 3.1), the latent-space diffusion formulation (Sec. 3.2), and the renoising-based test-time scaling procedure (Sec. 3.3). Model architectures are fully specified in Appendix D, with the embedder and denoiser configurations tabulated in Tables 9 and 10, and all training and inference hyperparameters are listed in Appendix E, in Tables 11, 12 and 13; the per-problem formal definitions are recalled in Appendix A. The two-stage training procedure and the inference procedure with renoising cycles are stated as Algorithms 2 and 3, and the decode-then-repair operators that map generated latents back to feasible solutions are specified problem by problem in Appendix F, with the same operators used for our method and for all neural baselines. The dataset configuration is described in Sec. 4 and Appendix B, with the instance generators, instance counts and reference solvers tabulated in Tables 7 and 8. The protocol of Sec. 4 additionally fixes the hardware (a single NVIDIA RTX 4090 GPU), the evaluation metrics and the timing measurement. Code will be made publicly available upon acceptance.

REFERENCES

[1] David Applegate, Ribert Bixby, Vasek Chvatal, and William Cook. Concorde tsp solver, 2006.

[2] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993, 2021.

[3] Jieyi Bi, Yining Ma, Jianan Zhou, Wen Song, Zhiguang Cao, Yaoxin Wu, and Jie Zhang. Learning to handle complex constraints for vehicle routing problems, 2024. URL `https://arxiv.org/abs/2410.21066`.

[4] Jieyi Bi, Zhiguang Cao, Jianan Zhou, Wen Song, Yaoxin Wu, Jie Zhang, Yining Ma, and Cathy Wu. Towards efficient constraint handling in neural solvers for routing problems, 2026. URL `https://arxiv.org/abs/2602.16012`.

[5] Quentin Cappart, Didier Chételat, Elias Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *arXiv preprint arXiv:2102.09544*, 2021.

[6] Felix Chalumeau, Shikha Surana, Clement Bonnet, Nathan Grinsztajn, Arnu Pretorius, Alexandre Laterre, and Thomas D. Barrett. Combinatorial optimization with policy adaptation using latent space search, 2024. URL https://arxiv.org/abs/2311.13569.

[7] Lvda Chen, Yang Li, and Junchi Yan. Maskco: Masked generation drives effective representation learning and exploiting for combinatorial optimization. In *The Fourteenth International Conference on Learning Representations*, 2026.

[8] George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

[9] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021.

[10] Darko Drakulic, Sofia Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. Bq-nco: Bisimulation quotienting for efficient neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:77416–77429, 2023.

[11] Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[12] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Lacey, Alex Goodwin, Yannik Marek, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis, 2024. URL https://arxiv.org/abs/2403.03206.

[13] Merrill M Flood. The traveling-salesman problem. *Operations research*, 4(1):61–75, 1956.

[14] Benjy Friedmann and Nadav Dym. Transport, don't generate: Deterministic geometric flows for combinatorial optimization, 2026. URL https://arxiv.org/abs/2602.10794.

[15] Gurobi Optimization. Gurobi optimizer reference manual. http://www.gurobi.com, 2020.

[16] Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, pp. 24–50, 2017.

[17] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[18] Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in Neural Information Processing Systems*, 34:12454–12465, 2021.

[19] 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.

[20] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

[21] Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

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

[23] Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F Werneck. Finding near-optimal independent sets at scale. In *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pp. 138–150. SIAM, 2016.

[24] Gilbert Laporte. Fifty years of vehicle routing. *Transportation science*, 43(4):408–416, 2009.

[25] Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. T2t: From distribution learning in training to gradient search in testing for combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2023.

[26] Yang Li, Jinpei Guo, Runzhong Wang, Hongyuan Zha, and Junchi Yan. Fast t2t: Optimization consistency speeds up diffusion-based training-to-testing solving for combinatorial optimization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[27] Yang Li, Lvda Chen, haonan Wang, Runzhong Wang, and Junchi Yan. Generation as search operator for test-time scaling of diffusion-based combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2025.

[28] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow, 2022. URL https://arxiv.org/abs/2209.03003.

[29] Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. *Advances in Neural Information Processing Systems*, 36, 2024.

[30] Jiale Ma, Wenzheng Pan, Yang Li, and Junchi Yan. COExpander: Adaptive solution expansion for combinatorial optimization. In *Forty-second International Conference on Machine Learning*, 2025.

[31] T. L. Magnanti. Combinatorial optimization and vehicle fleet planning: Perspectives and prospects. *Networks*, 11(2):179–213, 1981. doi: https://doi.org/10.1002/net.3230110209. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230110209.

[32] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

[33] Michel Minoux. Networks synthesis and optimum network design problems: Models, solution methods and applications. *Networks*, 19:313–360, 1989. URL https://api.semanticscholar.org/CorpusID:35836076.

[34] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, 31, 2018.

[35] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pp. 8162–8171, 2021.

[36] Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. Dimes: A differentiable meta solver for combinatorial optimization problems. *arXiv preprint arXiv:2210.04123*, 2022.

[37] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022. URL https://arxiv.org/abs/2112.10752.

[38] Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization, 2024. URL https://arxiv.org/abs/2406.01661.

[39] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[40] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[41] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[42] Zhiqing Sun and Yiming Yang. DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=JV8Ff0lgVV.

[43] Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

[44] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, and Walter Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624, 2012.

[45] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

[46] Qi Wang and Chunlei Tang. Deep reinforcement learning for transportation network combinatorial optimization: A survey. *Know.-Based Syst.*, 233(C), December 2021. ISSN 0950-7051. doi: 10.1016/j.knosys.2021.107526. URL https://doi.org/10.1016/j.knosys.2021.107526.

[47] Yu Wang, Yang Li, Jiale Ma, Junchi Yan, and Yi Chang. NExCO: Native solution expansion for diffusion-based combinatorial optimization. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=084SvT55yk.

[48] Haoran Ye, Jiarui Wang, Helan Liang, Zhiguang Cao, Yong Li, and Fanzhang Li. Glop: Learning global partition and local construction for solving large-scale routing problems in real-time. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 20284–20292, 2024.

[49] Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron Courville, Yoshua Bengio, and Ling Pan. Let the flows tell: Solving graph combinatorial optimization problems with gflownets. *arXiv preprint arXiv:2305.17010*, 2023.

[50] Zhi Zheng, Changliang Zhou, Tong Xialiang, Mingxuan Yuan, and Zhenkun Wang. Udc: A unified neural divide-and-conquer framework for large-scale combinatorial optimization problems, 2025. URL https://arxiv.org/abs/2407.00312.

A Problem Definitions

TSP and MIS. The Traveling Salesman Problem (TSP) [13] takes n points $\boldsymbol{x} \in [0,1]^{n\times 2}$ and seeks a Hamiltonian cycle $\boldsymbol{s} \in \{1,\dots,n\}^n$ of minimum length $\ell(\boldsymbol{s}) = \sum_i \|\boldsymbol{x}_{\boldsymbol{s}_i} - \boldsymbol{x}_{\boldsymbol{s}_{i+1}}\|_2$ (indices modulo n). The Maximum Independent Set (MIS) problem takes a graph $\mathcal{G} = (\boldsymbol{V}, \boldsymbol{E})$ and seeks a largest subset of mutually non-adjacent nodes.

CVRP and TSPTW. The Capacitated Vehicle Routing Problem (CVRP) [8; 24] takes a depot and n customers with coordinates $\boldsymbol{x}$ and demands $d_i > 0$, and seeks a set of routes that start and end at the depot, cover every customer exactly once and carry at most Q per route, minimizing the total route length. The Traveling Salesman Problem with Time Windows (TSPTW) [43] seeks a single tour through a depot and n customers each holding a time window $[e_i, \ell_i]$ that must be met (early arrival allowed only by waiting), again minimizing tour length.

A tour and an independent set are simple edge- and node-selection structures, whereas a routing solution is ordered, variable-length and constraint-coupled: a corrupted routing solution is hard even to define with a solution-space disruption heuristic, which motivates the latent-space treatment of routing (Sec. 3).

B Datasets

The TSP and MIS benchmarks follow GenSCO [27], with the configuration listed in Table 7. TSP instances draw n i.i.d. uniform coordinates in the unit square, their references are labeled by Concorde, and the TSP-500 and TSP-1000 test sets are the public 128-instance sets carried over from that protocol. MIS instances follow the two generators of the graph-learning literature, namely RB graphs [49] and Erdős–Rényi (ER) graphs at edge probability 0.15 [11]; the references are labeled by KaMIS, and the ER test set is the public set of [36]. The routing benchmarks use the standard generators of the learning-to-route literature, with the configuration listed in Table 8. CVRP-100 draws customer coordinates i.i.d. uniformly in the unit square, with demands uniform in $\{1,\dots,9\}$ and vehicle capacity 50 [21], and its references are labeled by HGS. TSPTW-50 draws its coordinates from the same generator and takes its time windows at the medium and hard tightness levels of [3], with references labeled by LKH-3.

Table 7: Dataset configurations for TSP and MIS, following the data protocol of [27].

	TSP		MIS	
Configuration	TSP-500	TSP-1000	RB-$[200, 300]$	ER-$[700, 800]$
Instance generator	uniform in $[0,1]^2$		RB graphs [49]	ER graphs, p=0.15 [11]
Training instances	128K	128K	90K	163,840
Test instances	128	128	500	128
Reference solver	Concorde [1]		KaMIS [23]	

Table 8: Dataset configurations for CVRP and TSPTW, following the data protocols of [21] and [3].

Configuration	CVRP-100	TSPTW-50 (medium / hard)
Instance generator	uniform in $[0,1]^2$	
Constraint data	demands $d \sim \mathrm{U}\{1,\dots,9\}$, capacity Q=50 [21]	time windows at medium and hard tightness [3]
Training instances	1,536K	1,024K
Test instances	1,280	10,000
Reference solver	HGS [44]	LKH-3 [16]

C ALGORITHMS

Alg. 2 states the full two-stage training procedure, and Alg. 3 states inference with the renoise cycles of Sec. 3.3. The two training stages are decoupled: the embedder is fitted on (instance, solution) pairs and then frozen, so the denoiser regresses toward a fixed target rather than a moving one, and the two checkpoints can be trained, reused and swapped independently, as the warm starts of Appendix E illustrate. Solution information enters the second stage only through the latent that the interpolation corrupts, so the instance encoder Enc_ψ never sees the target solution and the network must recover it from the instance and the noisy latents alone. In Stage 1 the embedder keeps the full-solution visibility flag $t{=}1$, the only setting it is ever used in, and the latent $\boldsymbol{\mu}_\varphi(\boldsymbol{x}, A_{\boldsymbol{s}})$ it returns passes to the denoiser unchanged.

Algorithm 2: Training LCO-Diff

Input: dataset $\{(\boldsymbol{x}^b, \boldsymbol{s}^b)\}_{b=1}^{B}$; embedder $\mathrm{Enc}_\varphi, \mathrm{Dec}_\varphi$; denoiser $f_{\boldsymbol{\theta}}$; β

Stage 1: instance-conditioned embedder;
for *each embedder step* **do**
  sample a batch $\{(\boldsymbol{x}, \boldsymbol{s})\}$ and form $A_{\boldsymbol{s}}$;
  $t \leftarrow 1$ (the embedder sees the full solution);
  $\boldsymbol{\mu}, \log \boldsymbol{\sigma} \leftarrow \mathrm{Enc}_\varphi(\boldsymbol{x}, A_{\boldsymbol{s}}, t)$;
  $\boldsymbol{z} \leftarrow \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$;
  $\mathcal{L}^{\mathrm{E}} \leftarrow \mathcal{L}_{\mathrm{rec}} + \beta\, D_{\mathrm{KL}}$ by Eq. 3;
  update φ;
freeze φ;
Stage 2: latent denoiser, as in Alg. 1;
for *each denoiser step* **do**
  sample a batch; $\boldsymbol{z} \leftarrow \boldsymbol{\mu}_\varphi(\boldsymbol{x}, A_{\boldsymbol{s}})$;
  $\boldsymbol{c} \leftarrow \mathrm{Enc}_\psi(\boldsymbol{x})$;
  draw $t \sim \mathcal{U}(0,1), \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$;
  $\boldsymbol{z}_t \leftarrow t\, \boldsymbol{z} + (1-t)\, \boldsymbol{\varepsilon}$;
  $\mathcal{L}^{\mathrm{D}} \leftarrow \| f_{\boldsymbol{\theta}}(\boldsymbol{c}, t, \boldsymbol{z}_t) - \boldsymbol{z} \|_F^2$;
  update $\boldsymbol{\theta}$;

Algorithm 3: Latent inference with renoise test-time scaling

Input: instance $\boldsymbol{x}$; trained models; steps K; runs R; cycles C; factor ρ

$\boldsymbol{c} \leftarrow \mathrm{Enc}_\psi(\boldsymbol{x})$; $\boldsymbol{s}^\star \leftarrow \emptyset$;
for $r = 1, \ldots, R$ **do**
  $\boldsymbol{z}_0 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$;
  $\hat{\boldsymbol{z}} \leftarrow \mathrm{Sample}_{\boldsymbol{\theta}}(\boldsymbol{z}_0, \boldsymbol{c})$;
  $\hat{H} \leftarrow \mathrm{Dec}_\varphi(\hat{\boldsymbol{z}})$;
  $\hat{\boldsymbol{s}} \leftarrow \mathrm{Repair}(\hat{H})$;
  $\boldsymbol{s}^\star \leftarrow \arg\mathrm{best}(\boldsymbol{s}^\star, \hat{\boldsymbol{s}})$;
  for $c = 1, \ldots, C-1$ **do**
    $\boldsymbol{z}^\star \leftarrow \mu_\varphi(\boldsymbol{x}, A_{\boldsymbol{s}^\star})$;
    $\boldsymbol{z}^\star_\rho \leftarrow (1-\rho)\, \boldsymbol{z}^\star + \rho\, \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$;
    $\hat{\boldsymbol{z}}' \leftarrow \mathrm{Sample}_{\boldsymbol{\theta}}(\boldsymbol{z}^\star_\rho, \boldsymbol{c})$ on $t \in [1{-}\rho, 1]$;
    $\hat{H}' \leftarrow \mathrm{Dec}_\varphi(\hat{\boldsymbol{z}}')$;
    $\hat{\boldsymbol{s}}' \leftarrow \mathrm{Repair}(\hat{H}')$;
    $\boldsymbol{s}^\star \leftarrow \arg\mathrm{best}(\boldsymbol{s}^\star, \hat{\boldsymbol{s}}')$;

Output: best solution $\boldsymbol{s}^\star$

Each run of Alg. 3 draws R independent latents and refines each of them by the encode–renoise–denoise cycles, so a run evaluates $R \times C$ candidates and returns the best one under the objective of the problem at hand. A full pass costs K denoiser evaluations and one cycle about $(1-\rho)K$, which is the budget counted in Sec. 4.4, so R and C trade off against each other at equal compute. Every candidate is decoded to a heatmap and passed through the constraint-aware repair routine of Appendix F before it is compared, so the incumbent $\boldsymbol{s}^\star$ is feasible throughout and a cycle is kept only when it improves the objective. The loop itself is problem-agnostic: only Repair and the objective are instantiated per problem.

D MODEL ARCHITECTURE DETAILS

All transformers are pre-norm with RMSNorm, 8-head attention (4 heads for MIS at width 256), SwiGLU feed-forward blocks, and SiLU activations. The attention bias is added to the pre-softmax attention scores, so an adjacency-structured input, the instance graph or the solution adjacency, enters the transformer at every layer; for MIS, padded nodes are masked from attention. Logits from the TSP decoder are the inner product of RMSNorm-normalized node features with scale $16/256$ (embedder) or 1.0 (denoiser latent head). Decoder construction: the embedder decoder is a linear up-projection of the latents followed by a 6-layer transformer and a final MLP; the denoiser mixes the noisy latent $\boldsymbol{z}_t$ into the instance features by a learned linear projection applied *after* the feature projection.

TSP Instantiation. The token features are the coordinates $\boldsymbol{x} \in \mathbb{R}^{n\times 2}$ (centered and scaled), and $A_{\boldsymbol{s}} \in \{0,1\}^{n\times n}$ is the tour adjacency matrix (symmetric, zero diagonal), used directly as the attention bias. The solution logit head is the pairwise head with the diagonal masked and logit scale

$16/256$, and the reconstruction loss is the row-softmax cross-entropy on $A_{\boldsymbol{s}}$ normalized to the average per-edge value.

MIS Instantiation. Node token features are drawn i.i.d. from $\mathcal{N}(0, \boldsymbol{I})$ and act as identity tokens; the graph adjacency matrix is the attention bias, and nodes of the (partially observed) solution receive an additional learnable *selected-node bias* added to their token features. $A_{\boldsymbol{s}}$ is the node-membership vector and the solution logit head projects each node embedding to a scalar inclusion logit, so the reconstruction loss is a per-node binary cross-entropy restricted to real (non-padded) nodes, with the timestep set to the fraction of selected nodes still present after dropout. The latent denoiser is a decoder-only transformer that consumes the identity features, the adjacency bias, and the noisy latents.

Routing (CVRP/TSPTW) Instantiation. For CVRP the token features are $(x, y, d_i/Q)$ with a learned depot bias on the depot token, and the route adjacency plays the role of $A_{\boldsymbol{s}}$ in the attention bias. For TSPTW the node token features are the four-dimensional concatenation (x_i, y_i, e_i, ℓ_i) of coordinates and time-window bounds, projected linearly as elsewhere; the embedder, the latent space and the renoise cycle are otherwise identical to the TSP instantiation, and time-window compliance is handled by the decoding stage (Appendix F).

Denoiser configuration. The TSP and MIS denoisers adopt the same architecture configuration as GenSCO [27], detailed in Table 10; the CVRP and TSPTW denoisers reuse the TSP configuration, with only the input dimension differing. Table 9 lists the embedder configuration.

Table 9: Embedder (VAE) configuration.

Item	**Configuration**
Architecture	encoder–decoder transformer
Encoder depth	6 layers
Decoder depth	6 layers
Embedding dimension	256
Attention heads	8 (4 for MIS)
Latent dimension	128

Table 10: Denoiser configurations, following GenSCO [27].

TSP / CVRP / TSPTW denoiser	
Architecture	encoder–decoder transformer
Encoder depth	16 layers
Decoder depth	6 layers
Embedding dimension	256
Attention heads	8
MIS denoiser	
Architecture	decoder-only transformer
Decoder depth	12 layers (RB), 8 layers (ER)
Embedding dimension	256
Attention heads	4

E HYPERPARAMETERS

For hyperparameters, we largely followed the configuration used in GenSCO [27]. Specifically, we adopted the same denoiser architecture and training hyperparameters to ensure a fair comparison and to highlight the benefits of the learned latent space.

Table 11: TSP training hyperparameters.

	TSP-500	**TSP-1000**
embedder steps / batch	100,000 / 128	100,000 / 64
embedder warm start	from TSP-100	from TSP-500
embedder β	1	1
denoiser warm start	from TSP-100	from TSP-500
denoiser steps / batch	600,000 / 128	600,000 / 64

Table 12: CVRP, TSPTW and MIS training hyperparameters, with the TSPTW columns referring to the medium and hard splits.

	CVRP & TSPTW		**MIS**	
	CVRP	TSPTW	RB-[200-300]	ER-[700-800]
embedder steps / batch	100,000 / 1024	100,000 / 2048	100,000 / 256	100,000 / 256
embedder β	1	1	0.5	0.5
denoiser steps / batch	100,000 / 2048	100,000 / 2048	211,000 / 256	384,000 / 256

Table 13: Inference hyperparameters.

	TSP		**MIS**		**CVRP & TSPTW**	
	TSP-500	TSP-1000	RB-[200-300]	ER-[700-800]	CVRP-100	TSPTW-50
sampling steps K	3	3	7	13	3	4
runs R	8	8	8	8	8	8
renoising ratio ρ	0.4	0.1	0.75	0.75	0.5	0.5

TSP: the completed TSP-500 and TSP-1000 evaluations (Table 1) average seeds 0/1/2 at K=3 denoising steps per pass, R=8 best-of runs with coordinate augmentation at level 1, and C swept over 8–512. TSP-500 uses ρ=0.4 and a 10-step ($= N/50$) 2-opt repair. TSP-1000 uses ρ=0.1 and a 20-step ($= N/50$) 2-opt repair with the top-20,000 candidate edges in the greedy insertion.

MIS: embedder 100,000 steps, batch 256, $\beta = 0.5$; denoiser 12 layers (RB) and 8 layers (ER), width 256; $K = 7$ (RB) / $K = 13$ (ER) denoising steps per renoise cycle at $\rho = 0.75$; no local search (greedy insertion only).

CVRP / TSPTW: $K = 3$ / $K = 4$ denoising steps per pass at $\rho = 0.5$, $R = 8$ runs with coordinate augmentation, and a 20-step 2-opt repair.

F Repair Operators

The concrete instantiations of the decode-then-repair routine, which follows [27], are as follows. On TSP, the heatmap is a row-wise softmax scaled by 2 so that each node's two tour edges receive unit mass, and the pipeline uses: (i) a batched *greedy insertion* that processes the top-k candidate edges in descending order, maintaining a partial tour and inserting an edge only if it keeps the tour feasible (no node degree > 2, no premature cycles); (ii) a 2-opt local search limited to $N/50$ steps, as in [27]. On MIS, the per-node logits are argsorted and a batched greedy insertion adds nodes in logit order, rejecting a node whenever it conflicts with an already-added node; no local search is applied. On CVRP, the insertion is route-aware: a candidate edge is admitted only if the partial solution remains a set of depot-rooted routes that visit each customer once and respect the capacity on every route, and the subsequent 2-opt search is capacity-aware. On TSPTW, candidate edges are inserted with a tour-level insertion, and the 2-opt search carries a time-window violation penalty, with the residual infeasible rate reported in Table 4. On the geometric problems (TSP, CVRP, TSPTW), the R runs additionally apply coordinate augmentations (90° rotations and reflections) to the instance before decoding. The operators are shared by our method and all neural baselines.