Hephaestus: Mixture Generative Modeling with Energy Guidance for Large-scale QoS Degradation

Nguyen Do[1,†], **Bach Ngo**[2,†], **Youval Kashuv**[1], **Canh V. Pham**[3], **Hanghang Tong**[4], **My T. Thai**[1,*]
[1]University of Florida, FL, USA [2]The Frazer School, FL, USA
[3]ORLab, Phenikaa University, Viet Nam [4]University of Illinois at Urbana-Champaign, IL, USA
[†]Equal contribution [*]Correspondence to: mythai@cise.ufl.edu

Abstract

We study the Quality of Service Degradation (QoSD) problem, in which an adversary perturbs edge weights to degrade network performance. This setting arises in both network infrastructures and distributed ML systems, where communication quality, not just connectivity, determines functionality. While classical methods rely on combinatorial optimization, and recent ML approaches address only restricted linear variants with small-size networks, no prior model directly tackles the QoSD problem under nonlinear edge-weight functions. This work proposes `Hephaestus`, a self-reinforcing generative framework that synthesizes feasible solutions in latent space, to fill this gap. Our method includes three phases: (1) Forge: a Predictive Path-Stressing (PPS) algorithm that uses graph learning and approximation to produce feasible solutions with performance guarantee, (2) Morph: a new theoretically grounded training paradigm for Mixture of Conditional VAEs guided by an energy-based model to capture solution feature distributions, and (3) Refine: a reinforcement learning agent that explores this space to generate progressively near-optimal solutions using our designed differentiable reward function. Experiments on both synthetic and real-world networks show that our approach consistently outperforms classical and ML baselines, particularly in scenarios with nonlinear cost functions where traditional methods fail to generalize.

1 Introduction

We consider the problem of degrading path-based system performance through minimal, stealthy perturbations over a set of weighted connections. Formally, let a directed graph $G = (V, E)$ with $|V| = n$ nodes and $|E| = m$ edges, represent a system of interacting components, where each i-th edge is associated with a non-decreasing weight function $f_i : \mathbb{N}^0 \to \mathbb{R}^+$. A perturbation $\mathbf{x} = (x_1, x_2, \ldots, x_m) \in \mathbb{N}^{|E|}$ is used to increase edge weights where the new weight i-th edge becomes $f_i(x_i) \, \forall i \in [m]$. We also refer to x_e as the budget allocated to increase the weight of edge e to $f_e(x_e)$. Given a set of critical source-target pairs $\mathcal{K} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$, a maximum perturbation budgets (box constraints) $\mathbf{b} = (b_1, b_2, \ldots, b_m) \in \mathbb{N}^m$ and a threshold $T \in \mathbb{R}^+$, the goal is to find the lowest-cost perturbation $\mathbf{x}$ such that every shortest path between $(s_i, t_i) \in \mathcal{K}$ exceeds T. We formulate this problem, also called Quality of Service Degradation (QoSD) [1] as the following constrained optimization:

$$\begin{aligned} &\min_{\mathbf{x} \in \mathbb{N}^m} \|\mathbf{x}\|_1 \\ &\text{subject to: } \mathrm{SP}_G(s_i, t_i; \mathbf{x}) \geq T \quad \forall (s_i, t_i) \in \mathcal{K} \\ &\qquad 0 \leq x_i \leq b_i, x_i \in \mathbb{Z}^+ \cup \{0\} \quad \forall i \in [m] \end{aligned} \tag{1}$$

where $\mathrm{SP}_G(s, t; \mathbf{x}) = \sum_{e \in \ddot{\rho}_{s,t}} f_e(x_e)$ denotes the length of the shortest path $\ddot{\rho}_{s,t}$ from s to t, under perturbation $\mathbf{x}$. QoSD is an NP-complete problem [1]. Note that the total path cost in $\mathrm{SP}_G(s, t; \mathbf{x})$ is

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

determined by the edge functions $f_i(x_i)$, rather than directly by x_i. Although $x_i \in \mathbb{Z}^+ \cup \{0\}$, the problem is not an Integer Program (IP) in general, since the cost functions $f_i(x_i)$ may be nonlinear (e.g., quadratic convex or log-concave). Intuitively, the QoSD problem seeks perturbation $\mathbf{x}$ that "stress" global system paths beyond their functional thresholds, not only connectivity while remaining subtle at the local level.

This problem arises in many real-world systems, from networked infrastructures to machine learning, where performance depends not just on connectivity but on effective communication. In blockchain, for example, transactions must reach all miners promptly to maintain consensus; targeted delays can disrupt synchronization and increase the risk of forks [2, 3, 4, 5]. The same story as with traffic control systems [2, 6, 7, 8, 9], attackers can manipulate signal timing to induce congestion and delay key routes. In Graph Neural Networks (GNN) [10, 11], steathly perturbing edge weights or node features can distort message passing and harm predictive performance. In all these settings, (i) the system functionality is compromised, regardless of its connectivity, making all the state of the art (SOTA) methods focusing on structural failures become ineffective; and (ii) attacks are stealthy distributed across the network, making local defenses ineffective. These characteristics make QoSD a natural and general framework for modeling such vulnerabilities.

Despite its compact formulation, this problem presents several challenges: First, the objective is non-submodular and becomes more complex under nonlinear edge functions such as quadratic or log-concave costs, making greedy or relaxation-based methods ineffective. Second, feasibility checking is expensive: every update to $\mathbf{x}$ requires recomputing global shortest paths for all pairs in $\mathcal{K}$—a bottleneck for large graphs or many constraints. Third, the solution space is exponentially large as the number of perturbation vectors grows exponentially with the number of edges, making the search for quality solutions intractable. These challenges limit the applicability of classical optimization and learning methods; thus, it calls for scalable alternatives that can handle a large number of path-based constraints and complex edge dynamics.

To address these challenges, we propose `Hephaestus`, a three-phase Forge-Morph-Refine generative framework for solving QoSD at scale under both linear and nonlinear edge functions. The framework begins with Forge, generating diverse feasible solutions using our shortest-path attention-based Predictive Path-Stressing (PPS) algorithm, which efficiently approximates global path constraints and *addresses both scalability and feasibility checking*. We next employ Morph, a mixture-of-generative-experts model guided by an Energy-Based Model (EBM), which allows deterministic expert expansion to approximate arbitrarily complex multimodal distributions over combinatorial solution spaces. Finally at Refine, a reinforcement learner refines the generated candidates in the latent space using a differentiable reward function, enabling smooth and efficient optimization. As a result, for any unseen graph that shares structural characteristics with those observed during training, the RL agent can quickly select appropriate latent variables and decode them into high-quality solutions—effectively addressing *the challenge of intractable over large and combinatorial spaces*.

Overall, our key contributions are summarized as follows:

- We propose `Hephaestus`, the first end-to-end, generative self-reinforcing framework for QoSD that unifies feasibility search, solution modeling, and optimization into a scalable pipeline capable of handling non-linear costs and large path constraints.
- Theoretically, we provide (i) an approximation guarantee for PPS, a corner stone of Forge, that bounds the cost of the generated solutions relative to the optimum, (ii) a convergence analysis for Energy-Based Model guided mixture training that avoids intractable normalizing constants for Morph, and (iii) a proof that policy refinement in the latent space can be directly performed via gradient ascent, enabling near-optimal solution tuning in Refine with RL.
- Extensive experiments on both synthetic and real-world networks demonstrate that `Hephaestus` consistently outperforms classical and ML baselines. Ablation studies further validate the complementary role of each phase in improving overall performance, especially under nonlinear cost regimes where traditional methods fail to generalize.

2 Related Work

Network Interdiction. The QoSD problem was first introduced by Nguyen and Thai [1]. It models a novel form of soft network interdiction, where attackers degrade end-to-end performance by increasing edge weights without explicitly disrupting topological connectivity. This contrasts sharply

with classical hard interdiction [12, 13, 14, 2], which assume complete removal of nodes or edges to disrupt flow or connectivity under budget constraints. These classical settings are less realistic in stealthy or infrastructure-constrained environments. Until recently, Stackelberg routing games under logit-based attacker response models [15] adopt a bounded-rational view of adversaries and optimize defender strategies probabilistically over sensitive nodes. While conceptually related to soft disruption, these models differ significantly from QoSD: they focus on node-based protection, rely on attacker stochasticity, and do not generalize to continuous edge-based disruptions.

Possible Solving Methods. Nguyen and Thai [1] proposed three approximation algorithms—Adaptive Trading (AT), Iterative Greedy (IG), and Sampling Algorithm (SA)—for solving the QoSD problem. These methods support both linear and nonlinear f_e but guarantee performance only in the linear case and treat source-target pairs independently, leading to less efficient budget spending. In contrast, `Hephaestus` leverages a PPS algorithm with guarantees for both settings. This foundation enables the training of self-reinforcing generative model that captures structural correlations across source-target pairs, resulting in more globally effective degradation strategies.

On the other hand, QoSD can be reformulated as an ILP in the special case when weight function $f_e(.)$ is linear, making recent ML techniques for integer and mixed-integer linear programs (ILPs/MILPs) [16, 17, 18, 19] applicable to this setting. One direction integrates ML into solver heuristics [20, 21, 22], such as branching [23], separation [24, 25], and cut selection [26, 27]. Another focuses on learning to generate heuristic solutions [28, 29, 30, 31, 32]. Notably, the Predict-and-Search (PS) framework [33, 34] leverages solvers such as Gurobi [35] or SCIP [36] to generate label solutions during training. At inference time, it predicts initial solutions, which are then refined using the same solvers, following the predict-then-optimize paradigm [37, 38, 39]. In contrast, DiffILO [40] relaxes ILPs into differentiable surrogate objectives and optimizing them end-to-end via gradient descent, enabling unsupervised training without solver-generated labels but only works effectively with small to medium ILP. `Hephaestus` fundamentally differs from these approaches by directly handling both linear and non-linear weight functions, including quadratic-convex and log-concave—without relying on solver and can be scalable to the large graph and number of constraints.

3 Hephaestus

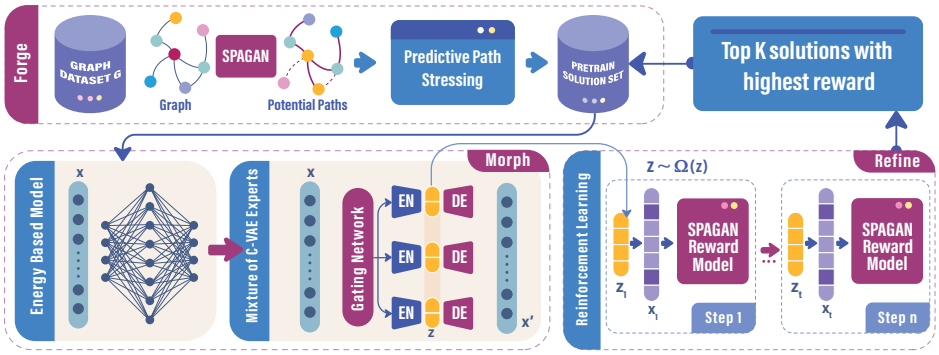


Figure 1: Given a graph dataset, we train SPAGAN to predict shortest path costs and guide PPS in generating feasible solutions across varying thresholds, forming a Pretrained Solution Dataset. Then, EBM approximates the underlying solution distribution, while a Mix-CVAE learns to closely match this distribution. Once the Mix-CVAE converges, an RL agent modifies latent samples, which are decoded into lower-cost solutions. The top-k high-reward samples are added back to the Solution Dataset to retrain both the EBM and Mix-CVAE in future episodes, enabling continual improvement.

We present our `Hephaestus` framework, illustrated in Figure 1, which consists of three phases: (1) **Forge**: Feasible Solution Generation, (2) **Morph**: Energy-Guided Generative Modeling, and (3) **Refine**: Latent Policy Optimization. In Forge, we use SPAGAN [41] to estimate shortest-path costs and guide our Predictive Path Stressing (PPS) algorithm, which generates feasible solutions (or perturbations) to QoSD and forms a pre-trained solution set $\mathfrak{D}^{\text{sol}}$. The Morph phase trains an Energy-Based Model (EBM) [42, 43] on $\mathfrak{D}^{\text{sol}}$ for guiding a Mixture-of-Conditional VAEs (Mix-CVAE) to approximate the underlying solution feature distribution. To improve coverage, new

CVAE [44, 45, 46] experts are dynamically added in regions of EBM where Mix-CVAE shows high modeling error. In the final phase Refine, a reinforcement learning agent operates in the latent space of Mix-CVAE to refine latent variables and synthesize better solutions. These new samples are iteratively added back into $\mathfrak{D}^{\text{sol}}$, allowing both EBM and Mix-CVAE to improve over time.

3.1 Forge: Feasible Solution Generation

Training strong generative model in Morph requires a high-quality dataset of feasible solutions $\mathbf{x}$, as the model can only learn to produce effective solution if the training data itself covers a wide range of valid perturbations with low budget. However, generating such data is challenging: for each pair (s, t), the QoSD objective is often non-submodular and the number of feasible paths having cost shorter than T can be extremely large, making it computationally expensive to enumerate or verify them all. To tackle this challenge, we design a hybrid ML-approximation algorithm that leverages a trained shortest-path attention model to guide the approximation search. We begin with a dataset of graphs $\mathfrak{D}^{\text{graph}} = \{G_i = (V_i, E_i)\}_{i=1}^{N}$ representing diverse sampled network topologies. For each G_i, we define a collection of target node pairs $\mathcal{K}_i = \{(s_j, t_j)\}_j$, from which shortest path estimations will be drawn. To this end, we train a Shortest Path Graph Attention Network (SPAGAN) [41], denoted by $\mathfrak{F}_{\boldsymbol{\theta}} : (G, s, t, \mathbf{x}) \mapsto \widehat{d}_{s,t}$, where $\mathfrak{F}_{\boldsymbol{\theta}}$ takes as input a graph $G = (V, E)$ and a source-target pair (s, t), and outputs a real-valued estimate $\widehat{d}_{s,t} \in \mathbb{R}_+$ of the shortest-path distance between s and t under baseline weight function f_e. The model $\mathfrak{F}_{\boldsymbol{\theta}}$ is trained using a supervised regression loss (i.e Huber Loss [47, 48]) over mini-batches of path distances from subgraphs, where ground truth distances are obtained via exact computation (i.e Dijkstra algorithm [49]). By training over a diverse set of subgraphs extracted from graph, the model learns transferable embeddings that capture structural priors and efficiently generalize to large graphs, enabling efficient solution verification in unseen instances.

Once $\mathfrak{F}_{\boldsymbol{\theta}}$ is trained, we use it to guide the searching of diverse solutions $\mathbf{x}$. Specifically, we propose a PPS algorithm to iteratively search adversarial perturbations that elevate the predicted path length $\widehat{d}_{s,t}(\mathbf{x})$ of each pair (s, t) beyond a given threshold T. Unlike traditional approximation methods [1] that require costly repeat exact path computation, PPS leverages the efficiency of $\mathfrak{F}_{\boldsymbol{\theta}}$ to estimate marginal gains in violation potential and guides the perturbation accordingly. Let P denote the current set of shortest paths under perturbation $\mathbf{x}$, i.e., $P = \{\ddot{\rho}_{s,t} \mid (s, t) \in \mathcal{K},\ \mathfrak{F}_{\boldsymbol{\theta}}(G, s, t; \mathbf{x}) < T\}$ where $\mathfrak{F}_{\boldsymbol{\theta}}(G, s, t, \mathbf{x}) \approx \sum_{e \in \ddot{\rho}_{s,t}} f_e(x_e)$ returns the predicted shortest path length under $\mathbf{x}$, if $\mathfrak{F}_{\boldsymbol{\theta}}$ is well trained. To quantify the progress of $\mathbf{x}$, we define the potential function $\mathcal{C}(P, \mathbf{x}) = \sum_{\ddot{\rho}_{s,t} \in P} \min\left(T, \mathfrak{F}_{\boldsymbol{\theta}}(G, s, t, \mathbf{x})\right)$. Each iteration of PPS selects an edge $e \in \mathcal{E}$ and an increment $\Delta x_e \in \mathbb{N}$ that maximizes the predicted gain-to-cost ratio:

$$(e^*, \Delta^*) = \arg \max_{e \in \mathcal{E},\ \Delta \in [1, b_e - x_e]} \frac{\mathcal{C}(P, \mathbf{x} + \Delta \cdot \mathbf{1}_e) - \mathcal{C}(P, \mathbf{x})}{\Delta} \tag{2}$$

where $\mathbf{1}_e$ is the standard basis vector corresponding to edge e, and b_e is the upper budget bound for that edge. After determining (e^*, Δ^*), perturbation is updated via $\mathbf{x} \leftarrow \mathbf{x} + \Delta^* \cdot \mathbf{1}_{e^*}$, and the path set P is recomputed via $\mathfrak{F}_{\boldsymbol{\theta}}$. This process is repeated until almost all node pairs $(s, t) \in \mathcal{K}$ are predicted to have path lengths exceeding the target threshold, i.e., $\mathcal{C}(P, \mathbf{x}) \geq |P|T - \bar{\epsilon}$, where $\bar{\epsilon}$ is an input parameter. For each graph $G_i \sim \mathfrak{D}^{graph}$, the PPS algorithm is executed independently across a set of QoS thresholds $\mathcal{T} = \{T_1, \ldots, T_M\}$ and pairs $\mathbf{K} = \{\mathcal{K}_1, \ldots, \mathcal{K}_M\}$, producing a corresponding set of feasible perturbations $\{\mathbf{x}^{(i)}_{\mathcal{K},T} | \mathcal{K} \in \mathbf{K}, T \in \mathcal{T}\}$. The resulting pretrained solution dataset is thus formalized as $\mathfrak{D}^{sol} = \{(G_i, \mathcal{K}, T, \mathbf{x}^{(i)}_{\mathcal{K},T}) | G_i \in \mathfrak{D}^{graph}, \mathcal{K} \in \mathbf{K}, T \in \mathcal{T}\}$ which forms the basis of the Morph phase (Section 3.2). For notational simplicity, we refer to each element of $\mathfrak{D}^{sol}$ as a training instance $(G, \mathcal{K}, T, \mathbf{x})$, representing a specific degradation scenario and having a guarantee:

Theorem 1. *(PPS Ratio) Let $h = \lceil T/w_{\min} \rceil$, where $w_{\min} = \min_{e \in E} w_e$. Assume that the set $\mathcal{E}$ is chosen from E such that $\Pr[\mathcal{E}^* \subseteq \mathcal{E}] = \mathsf{a}$, where $\mathcal{E}^*$ is the set of edges of the optimal solution and $\mathsf{a} \in (0, 1)$ is a constant. Given a parameter $\bar{\epsilon} > 0$, then running PPS on $\mathcal{E}$ yields a solution $\boldsymbol{x}$ such that $\mathcal{C}(P, \boldsymbol{x}) \geq |P|T - \bar{\epsilon}$ and $\mathbb{E}[\|\boldsymbol{x}\|_1] \leq (1 + h\ln(n) + \ln T + \ln(1/\bar{\epsilon}))\mathsf{OPT}/\mathsf{a}$. (Proof in Appx **B.1**)*

Theorem 1 ensures that every solution generated by PPS achieves a provably bounded cost relative to the optimal. Such quality assurance is critical: the generative model introduced in Morph will be trained purely on $\mathfrak{D}^{sol}$, thus its quality directly determines the generative capacity and generalization

performance. Without such a theoretical grounding, the learned latent distribution could diverge or collapse around suboptimal patterns.

3.2 Morph: Energy-Guided Generative Modeling

Given the pre-trained solution set $\mathfrak{D}^{\text{sol}}$ from the Forge phase, Morph aims to model the underlying pattern of high-quality solutions through the conditional distribution $p(\mathbf{x} \mid \mathbf{c})$, where the condition $\mathbf{c} = [G, \mathcal{K}, T]$ encodes the graph, the set of critical node pairs, and the threshold context. Since the true distribution p is unknown, we approximate it using a conditional generative model $\Omega(\mathbf{x} \mid \mathbf{c})$ that can generate feasible solutions for arbitrary $[G, \mathcal{K}, T]$ instance. To achieve this, we implement Ω as a Mixture of Conditional VAEs (Mix-CVAEs), denoted $\Omega = [\Omega_0, \ldots, \Omega_N]$, where each expert Ω_i consists of an encoder $\mathcal{P}_\psi$ and a decoder $\mathcal{M}_\phi$. For a given training pair $(\mathbf{x}, \mathbf{c})$, the encoder produces a latent posterior $\tilde{q}_\psi(\mathbf{z} \mid \mathbf{x}, \mathbf{c})$, and the decoder reconstructs the input via $\tilde{p}_\phi(\mathbf{x} \mid \mathbf{z}, \mathbf{c})$, forming the mapping $\Omega_i(\mathbf{x}, \mathbf{c}) = \mathcal{M}_\phi(\mathcal{P}_\psi(\mathbf{x}, \mathbf{c})) = \mathcal{M}_\phi \circ \mathcal{P}_\psi$. We train each CVAE expert by maximizing the evidence lower bound (ELBO) [45] on samples $(G, \mathcal{K}, T, \mathbf{x}) \sim \mathfrak{D}^{\text{sol}}$:

$$\mathcal{L}_{\Omega_i}^{ELBO} = \mathbb{E}_{\tilde{q}_\psi} \log \tilde{p}_\phi(\mathbf{x} \mid \mathbf{z}, \mathbf{c}) - \mathrm{KL}[\tilde{q}_\psi(\mathbf{z} \mid \mathbf{x}, \mathbf{c}) \,\|\, \tilde{p}_\phi(\mathbf{z} \mid \mathbf{c})] \tag{3}$$

where the first term encourages accurate reconstruction of $\mathbf{x}$, and the second regularizes the latent representation $\mathbf{z} \in \mathbb{R}^d$ to align with a context-aware prior. After training, new solutions can be generated by sampling $\mathbf{z} \sim \tilde{p}_\phi(\mathbf{z} \mid \mathbf{c})$ and decoding via $\tilde{\mathbf{x}} = \mathcal{M}_\phi(\mathbf{z}, \mathbf{c})$.

Mode-Seeking Problem. Since optimizing the ELBO is merely an indirect way to minimize the reverse KL divergence, this objective often encourages the learned distribution $\tilde{p}_\phi(\mathbf{x} \mid \mathbf{c})$ to focus on the dominant modes that appear most frequently in the dataset $\{\mathbf{x}_i\}$. However, if $p(\mathbf{x})$ is truly multimodal [50], a single expert Ω_i can exhibit mode-seeking behavior [51]—i.e., it may ignore rare but critical regions of $p(\mathbf{x} \mid \mathbf{c})$. More severely, we do not even know which regions of p are not being captured by the CVAE Ω_i, since the true form of p is unknown.

Energy-Based Guidance for Generative Modeling. To address the above limitation, we introduce an Energy-Based Model (EBM) [43] defined as $q(\mathbf{x}) = \frac{1}{Z} \exp(-E(\mathbf{x})/\tau)$, where $E(\mathbf{x})$ is a learnable energy function, $\tau > 0$ is a temperature parameter controlling the sharpness of the distribution and the normalizing constant function $Z = \int_{\mathcal{X}} \exp(-E(\mathbf{x})/\tau)\, d\mathbf{x}$ ensures $q(\mathbf{x})$ is a valid probability distribution. Unlike CVAE, EBM makes no strong parametric assumptions about the form of the true distribution $p(\mathbf{x})$; it instead defines a flexible energy landscape shaped directly by the energy values. We thus employ EBM $q(\mathbf{x})$ as a *surrogate* for the true—but unknown—distribution $p(\mathbf{x})$, and aim to learn q by minimizing $\min_{q \in \mathcal{Q}} \mathrm{KL}(p(\mathbf{x}) \,\|\, q(\mathbf{x}))$, where $\mathcal{Q}$ denotes the feasible family of EBMs parameterized by $E(\mathbf{x})$. Simultaneously, we train the generative model $\Omega(\mathbf{x})$ (e.g., a Mix-CVAEs) to match $q(\mathbf{x})$ by minimizing the KL divergence $\min_{\Omega \in \mathfrak{E}} \mathrm{KL}(q(\mathbf{x}) \,\|\, \Omega(\mathbf{x}))$, where $\mathfrak{E}$ represents the feasible set of generative distributions realizable by our model class. Ideally, when the two KL divergences become zero, then $p(x) = q(x) = \Omega(x)$ or indirectly $\Omega(x) = p(x)$.

However, a single C-VAE may only capture a dominant mode of the distribution $q(x)$ as we mentioned earlier. To ensure comprehensive coverage, we monitor the density gap between the EBM and the current generative model via the log-ratio function $\chi(\mathbf{x}) = \log q(\mathbf{x})/\Omega(\mathbf{x})$. Whenever $\chi(\mathbf{x}) > \delta$ for some threshold $\delta > 0$, we interpret this region as underrepresented by the current C-VAE model Ω, and dynamically add a new C-VAE expert $\Omega_{N+1}(\mathbf{x})$ specialized to this region forming new Mixture of C-VAE Ω'. This expert is integrated into the mixture through a gating mechanism, ensuring that the overall generative model incrementally improves its coverage of high-density regions in $q(\mathbf{x})$, and thereby indirectly approximates the true data distribution $p(\mathbf{x})$ (See Figure 2) .

Theorem 2. *(Expert Augmentation Efficiency): Suppose there exists a constant $\epsilon > 0$ such that $\mathbb{P}_{p(x)}\{\chi(x) > \delta\} \geq \epsilon$; on the region $\{x : \chi(x) > \delta\}$, a new expert $\Omega_{N+1}(x)$ is added with gating weight $\alpha(x) = \mathbf{1}\{\chi(x) > \delta\} a_0$ ($0 < a_0 \leq \bar{a} < 1$) and trained to satisfy $\Omega_{N+1}(x) \geq c\, q(x)$ for some $c \in (0, 1)$; then the updated mixture $\Omega'(x) = \alpha(x)\, \Omega_{N+1}(x) + (1 - \alpha(x))\, \Omega(x)$ satisfies $\mathrm{KL}(q \| \Omega') \leq \mathrm{KL}(q \| \Omega) - \gamma(\delta, \epsilon)$, where $\gamma(\delta, \epsilon) = a_0(\delta + \log c)\, \epsilon_0 > 0$ and $\epsilon_0 > 0$ is a lower bound on $\int_{\chi(x) > \delta} q(x) dx$. (Proof in Appendix **B.2**)*

From Theorem 2, we observe that each time a new expert is added, $\mathrm{KL}(q \| \Omega')$ is reduced by at least $\gamma(\delta, \epsilon)$ relative to the previous $\mathrm{KL}(q \| \Omega)$. Since the initial KL divergence with only one expert is finite, repeatedly augmenting the mixture with new experts allows us, in the limit of infinitely many experts without sacrificing the computational cost [52, 53], to drive the $\mathrm{KL}(q \| \Omega')$ to zero on $\mathfrak{D}^{sol}$.

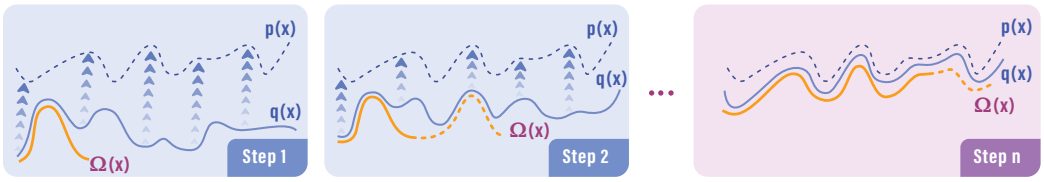


Figure 2: At each iteration, the generative model $\Omega(x)$ adds an expert in regions where the log-ratio $\chi(x) = \log\left(q(x)/\Omega(x)\right)$ exceeds δ, thereby covering more of $q(x)$. Meanwhile, $q(x)$ adapts to align with $p(x)$. As the steps progress, $q(x)$ increasingly approximates $p(x)$, and $\Omega(x)$, augmented by new experts, converges to $q(x)$, ultimately yielding $p(x) \approx q(x) \approx \Omega(x)$.

However, directly computing $\mathrm{KL}(p(\mathbf{x})\|q(\mathbf{x}))$ and $\mathrm{KL}(q(\mathbf{x})\|\Omega(\mathbf{x}))$ is highly challenging in practice, as it requires evaluating the normalization constant Z of the EBM. Estimating Z typically involves Markov Chain Monte Carlo (MCMC) [54], which becomes extremely expansive during the EBM training. Fortunately, as we will show next, there is no need to compute Z.

Theorem 3. *(Normalization Free Function) The objective function* $\min_q \max_\Omega\{\mathrm{KL}(p\|q) - \mathrm{KL}(\Omega\|q)\}$, *is normalizing free which is independent with* Z. *(Proof in Appendix* ***B.3****)*

Per Theorem 3, we completely avoid the need to compute Z by reformulating the problem as a minimax objective, where the normalizing constant cancels out in the difference of KL divergences:

$$\min_{q\in\mathcal{Q}} \max_{\Omega\in\mathfrak{C}} \{\mathrm{KL}(p\|q) - \mathrm{KL}(\Omega\|q)\} \tag{4}$$

Any terms involving $\log Z$ appear in both $\mathrm{KL}(p\|q)$ and $\mathrm{KL}(\Omega\|q)$ and hence cancel each other exactly. This cancellation frees us from having to perform expensive MCMC estimates of Z, yet still allows the EBM q to guide the generative model Ω effectively.

Minimax Optimization. From Equation 4, the EBM is optimized by minimizing the difference between the expected energy under the data distribution and the expected energy under the Mix-CVAE model distribution $\min_\theta \mathbb{E}_{p(\mathbf{x})}[E_\theta(\mathbf{x})] - \mathbb{E}_{\Omega(\mathbf{x})}[E_\theta(\mathbf{x})]$ (minimax derivation in Appendix **B.3**). To stabilize training and ensure that energy values remain bounded, we incorporate a commonly-used regularization term [42] based on the squared energies $\gamma\left(\mathbb{E}_{p(\mathbf{x})}[E_\theta(\mathbf{x})^2] + \mathbb{E}_{\Omega(\mathbf{x})}[E_\theta(\mathbf{x})^2]\right)$ where $\gamma > 0$ is a regularization coefficient. For Mix-CVAE approximating EBM, we modify the standard ELBO to incorporate EBM guidance during expert training. Specifically, based on Equation 3, we define a guided objective $\mathcal{L}_{\Omega_i}^{guide} = \mathcal{L}_{\Omega_i}^{ELBO} + \lambda \cdot \mathbb{E}_{\tilde{p}_\phi(\mathbf{z}|c)}\left[E_\theta\left(\tilde{p}_\phi(\mathbf{x} \mid \mathbf{z}, c)\right)\right]$, where the added term penalizes low-quality samples in high-energy (i.e., low-density) regions under the energy-based model $q_\phi(\mathbf{x}) \propto \exp(-E_\theta(\mathbf{x}))$.

3.3 Refine: Latent Policy Optimization

Once the Mix-CVAE Ω has successfully captured the true solution distribution, similar inputs (consisting of $\mathbf{x}$, G, $\mathcal{K}$, and T) that lead to similar-quality solutions are mapped to nearby points in the continuous latent space. This property makes the search and optimization process more efficient. In Refine, we train an RL agent with a policy π to blend and explore that latent space, aiming to generate solutions that are better than those in the original pretrained dataset $\mathfrak{D}^{sol}$. These newly discovered solutions are then added back into $\mathfrak{D}^{sol}$, reinforcing their feature patterns and making it easier to retrain the generative model in future iterations. The optimization of policy π over the latent space is formulated as a Markov Decision Process (MDP) [55], $\mathcal{M} \stackrel{\text{def}}{=} (\mathcal{S}, \mathcal{A}, \Gamma, \mathcal{R})$. This includes (i) a finite sets of states $\mathcal{S}$, (ii) a finite set of actions $\mathcal{A}$, (iii) a transition distribution $\Gamma\left(s' \mid s, a\right)$ where $s, s' \in \mathcal{S}, a \in \mathcal{A}$ and (iv) a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. We specify each component as follows:

State $(s \in \mathcal{S})$: A state is defined by latent representations $s_i = (\boldsymbol{z}_i, \mathbf{c})$ for $\boldsymbol{z}_i = \mathcal{P}_\psi\left(\mathbf{x}_i, \mathbf{c}\right)$ and $\mathbf{x}_i \in \mathfrak{D}_{\text{ep}}^{sol}$, where $\boldsymbol{z}_i \in \mathbb{R}^d$ is the latent vector produced by the encoder $\mathcal{P}_\psi$ from the input data $\mathbf{x}_i$.

Action $(a \in \mathcal{A})$: An action $a_i = (\mu_i, \sigma_i)$ is a vector of two components: $\mu_i \in \mathbb{R}^d$ (predicted mean) and $\sigma_i \in \mathbb{R}^d$ (predicted scale). The modification vector $\delta_i = \sigma_i \cdot \epsilon_{noise} + \mu_i$, where $\epsilon_{noise} \sim \mathcal{N}(0, I)$, and the latent vector is updated as $\hat{\boldsymbol{z}}_i = \boldsymbol{z}_i + \delta_i$.

Transition Dynamics (Γ): When an action $a_i = (\mu_i, \sigma_i)$ is taken, a new state $s_{i+1} = (\boldsymbol{z}_{i+1}, \mathbf{c})$, where $\boldsymbol{z}_{i+1} = \hat{\boldsymbol{z}}_i$, is then formed.

Reward Function ($\mathcal{R}$): Given z_{i+1}, the decoder $\mathcal{M}_\phi$ decodes it into a solution $\hat{\mathbf{x}}_{i+1} = \{x_1, \ldots, x_m\}$. To make optimization smoother, we apply a soft transformation to each element: $\bar{x}_j = \log(1 + e^{x_j})$ where $x_j \in \hat{\mathbf{x}}_{i+1}$ to produce a smooth vector $\bar{\mathbf{x}}_{i+1}$. Moreover, instead of using a binary feasibility label (feasible or not) for each pair $(s,t) \in S$, we define a smooth total feasibility score over all pairs:

$$F(G, \mathcal{K}, \hat{\mathbf{x}}_{i+1}) = \sum_{(s,t)\in\mathcal{K}} \frac{1}{1 + \exp\left(-\zeta \cdot (\mathfrak{F}_{\boldsymbol{\theta}}(G, s, t; round(\hat{\mathbf{x}}_{i+1})) - T)\right)}. \tag{5}$$

Here, for each pair (s,t), ζ controls how sharply the score changes near the threshold T. If the predicted path cost slightly exceeds T, the sigmoid quickly pushes the score closer to 1. This formulation ensures that $F(G, S, \hat{\mathbf{x}}_{i+1})$ provides a continuous and differentiable reward signal, which is more suitable for optimizing the RL policy. Finally, our reward function is defined as:

$$\mathcal{R}(\mathbf{x}_{i+1}) = F(G, \mathcal{K}, \hat{\mathbf{x}}_{i+1}) - \varkappa \cdot \log(1 + \|\bar{\mathbf{x}}_{i+1}\|_1). \tag{6}$$

This reward formulation balances two objectives: the first term encourages the generation of solution vectors by pushing the predicted path cost above T, while the second term, weighted by $\varkappa$, penalizes excessive budget usage, thus promoting stealth and cost-effectiveness. We intentionally keep the cost penalty relatively small so that the RL agent focuses primarily on finding a valid solution (i.e, only increasing weights) before optimizing cost. In what follows, we show several properties of our dense reward function which make the RL training process much easier.

Lemma 1. *(Differentiable Reward Function) Let $\mathfrak{F}_\theta : \mathcal{X} \to \mathbb{R}$ be differentiable on an open set $\mathcal{X} \subset \mathbb{R}^m$. The reward function $\mathcal{R}(\mathbf{x})$ is differentiable on $\mathcal{X}$. (Proof in Appendix **B.4**)*

Theorem 4. *(Reward Estimation Consistency) Assume that Ω has converged properly, for any perturbed latent vector $\hat{z}_i := z_i + \hat{\epsilon} \cdot \nabla_{z_i} \mathcal{R}(\mathcal{M}_\phi(z_i, \boldsymbol{c}))$ with small $\hat{\epsilon} > 0$, we have $\mathcal{R}(\hat{\mathbf{x}}_i) > \mathcal{R}(\mathbf{x}_i)$, where $\hat{\mathbf{x}}_i = \mathcal{M}_\phi(\hat{z}_i, \boldsymbol{c})$. (Proof in Appendix **B.5**)*

Specifically, by Theorem 4, small perturbations in the latent code z_i following the gradient of the reward yield strictly higher returns. Since the reward $R(x)$ is differentiable (Lemma 1), we can perform gradient ascent [56] directly in latent space as in Equation 6. Concretely, starting from z_i, we iterate $\hat{z}_i \leftarrow z_i + \hat{\epsilon} \cdot \nabla_{z_i} \mathcal{R}(\mathcal{M}_\phi(z_i, \mathbf{c}))$ until convergence to z_i^* leading to next state $s_{i+1} = (\boldsymbol{z}_{i+1} = \hat{z}_i^*, \mathbf{c})$. The resulting latent offset $\hat{\delta}_i = \hat{z}_i - z_i$ together with σ_i defines a feasible action $\tilde{a}_i = (\hat{\delta}_i, \sigma_i)$. This warm-start procedure mitigates inefficient random exploration in the early stages and accelerates convergence by guiding policy π towards locally optimal regions. Once the model reaches sufficient reward levels, we allow it to autonomously explore and exploit the latent space using standard reinforcement learning framework, leveraging efficient exploration strategies from the RL literature [57, 58, 59]. See Appendices **A** and **C** for the full algorithm and its analysis.

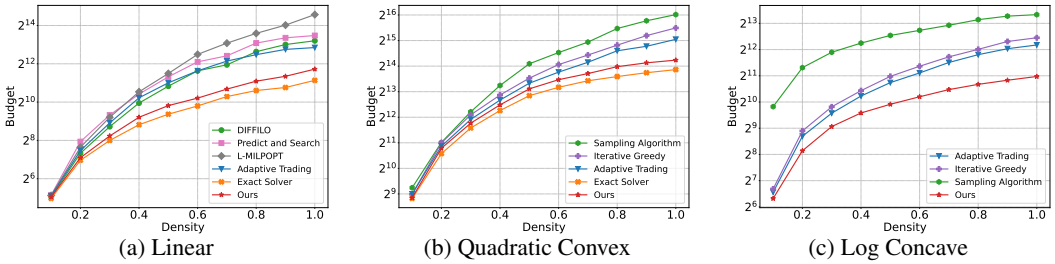
Inference Process. Given a new instance problem, policy π iteratively modifies a random latent vector z in latent space to maximize the reward defined in Equation 6. However, it is not guaranteed that the decoded solution $\hat{\mathbf{x}}$ from $\mathcal{M}_\phi(z)$ is valid. To address this, we introduce a feasibility refinement algorithm called PPS-I, a variant of the PPS algorithm. Unlike PPS, which relies on the learned estimator $\mathfrak{F}_{\boldsymbol{\theta}}(\cdot; \hat{\mathbf{x}})$ for the approximation of the shortest path, PPS-I replaces it with the exact calculation of the shortest path using Dijkstra's algorithm to ensure 100% feasibility. This yields $\mathsf{a} = 1$ in the ratio stated in Theorem 1. Moreover, with strong Mix-CVAE Ω, the initial solution $\hat{\mathbf{x}}$ produced by π is often close to valid, making the refinement overhead of PPS-I significantly minimized. Implementation details and analysis of PPS-I are provided in Appendix **A4**.

4 Experiments

Settings. We evaluate the effectiveness and scalability of Hephaestus on synthetic and real-world networks under varying QoSD thresholds. For synthetic graphs, we generate Erdős–Rényi topologies with $n = 1024$ nodes and vary edge density l, fixed threshold $T = 20$, $|\mathcal{K}| = 10$ critical pairs. Real-world datasets include Email [60], Gnutella [61], RoadCA [62], and Skitter [63], covering diverse scales and domains. For each, we sample 50 critical pairs, compute their maximum baseline shortest-path length, and set T to 140%–260% of this value to normalize degradation difficulty across topologies. Dataset statistics are in Appendix **C.1**. For training the entire `Hephaestus` framework in both synthetic and real networks, we generate a large corpus of graphs with varying architectures (e.g., Barabási-Albert, Erds-Rényi, and Watts-Strogatz), as well as diverse edge densities and thresholds, while keeping the number of nodes consistent with those in the testing graphs.

Baselines. We compare Hephaestus with a range of baselines, including classical approximation methods and learning-based IP solvers. Approximation baselines include Adaptive Trading (AT), Iterative Greedy (IG), and Sampling Algorithm (SA) from [1], which offer varying trade-offs between cost-effectiveness, concavity handling, and scalability. For learning-based IP methods, we evaluate three SOTA methods for ILP: DIFFILO [40], Predict-and-Search [33], and L-MILPOPT [25], which combine neural predictors with ILP solvers like Gurobi. Baseline solutions are refined using Gurobi with a 3000s timeout—tripled from the original setup. For large networks (e.g., RoadCA, Skitter), we extend this to 6000s and enable 'MIPFocus=1' [40, 64, 65] with heuristics to prioritize feasibility. This setup is crucial, as exact solving often leads to long runtime or memory errors in large networks. In constrast, our method employs PPS-I to ensure 100% feasible refinement without relying on exact solvers, while still achieving significantly faster performance due to its approximate nature. Note that we excluded a diffusion-based generative solver (e.g., the Gurobi-guided diffusion model [66]) due to its scalability issues in large-scale constrained combinatorial optimization problems, particularly over graphs with millions of nodes. In particular, this approach requires costly iterative sampling using a series of VAE models, with 100 denoising steps for Denoising Diffusion Implicit Model (DDIM) [67] and 1000 steps for Denoising Diffusion Probabilistic Model (DDPM) [68], making them computationally impractical to evaluate in large-scale setups such as RoadCA or Skitter.

Edge Weight Function Settings. We evaluate methods under three types of $f_e(x)$ representing different degradation dynamics: (i) *Linear*: $f_e(x) = \aleph(x)$, modeling uniform delay per unit cost; (ii) *Quadratic Convex*: $f_e(x) = \aleph(x^2)$, capturing congestion effects with rapidly increasing cost; and (iii) *Log Concave*: $f_e(x) = \aleph(\ln x)$, modeling diminishing returns as in error-rate degradation. ML-based IP solvers operate only under linear functions, so we compare all methods in this setting, including the exact solver. For convex and concave settings, only the approximation methods and Hephaestus are applicable. Notably, Gurobi fails on the Log Concave case due to the non-convex feasible region induced by logarithmic constraints, and is excluded from that evaluation.



(a) Linear (b) Quadratic Convex (c) Log Concave

Figure 3: Our solution quality on the testing synthetic graph with varying density is evaluated under Linear, Quadratic Convex, and Log-Concave functions. ML-based ILPs are compared with the best-performing approximation baseline AT for Linear weights but excluded for the other two due to incompatibility; exact solver also does not support Log-Concave.

	Email				Gnutella				RoadCA				Skitter			
Method	140%	180%	220%	260%	140%	180%	220%	260%	140%	180%	220%	260%	140%	180%	220%	260%
Adaptive Trading	2653	5460	8226	9675	3528	6613	8528	10656	12651	16972	23504	32935	367026	903278	1739299	3018467
Iterative Greedy	2673	5458	8258	9713	3537	6626	8564	10674	13782	17830	24076	33127	389415	915682	1983011	3125008
Sampling Alg.	4363	9249	14665	17290	4200	7775	14920	16427	23742	43822	70823	90166	597239	1789237	2871248	6294921
DIFFILO	**2621**	**5308**	8303	9695	**3490**	6662	8619	10987	11043	17161	24031	32976	350397	902366	1724336	3012839
Predict & Search	2654	5421	8458	9701	3645	6808	8839	11364	11196	18084	24967	33954	358937	929115	1849725	3197237
L-MILPOPT	2651	5462	8486	9811	3615	6922	9185	11772	12069	19302	25310	34866	360994	931678	1901972	3220155
Exact Solver	2570	5268	7968	9318	3383	6402	8211	10073	—	—	—	—	—	—	—	—
Ours	2647	5329	**8206**	**9601**	3497	**6511**	**8376**	**10495**	**9276**	**14415**	**20186**	**27699**	**255789**	**658727**	**1258765**	**2199372**

Table 1: Performance of Hephaestus and baselines on four real datasets at different thresholds T under Linear edge weight function setting. The best is highlighted in bold excluding exact solution. For RoadCA and Skitter dataset, ML-based methods can only use Gurobi with heuristic mode.

Solution Quality Evaluation. On the synthetic dataset, Hephaestus achieves the lowest total cost in all graph densities, outperforming AT and closely matching the exact solver (Figure 3-Linear). This is because Hephaestus uses various solutions from the PPS algorithm to train Ω_Θ which learns where the budget should be spent effectively. As a result, Hephaestus often spends budgets on common edges that affect multiple source-target pairs. In contrast, even the best approximation algorithm in

baselines, AT, cannot provide better solution than `Hephaestus` as it handles each pair independently, which can lead to spending on edges that do not help other pairs.

Learning-based methods also exhibit the same behavior, resulting in solutions that are worse than ours. DiffILO spreads budget too widely to fix constraint violations because it relaxes binary decisions into soft probabilistic variables, while Light-MILPopt splits the graph, which breaks global structure and leads to redundant spending. Predict-and-Search depends on Gurobi-generated labels: if these are poor, it cannot refine them due to a narrow search space. For the Quadratic Convex (QC) and Log Concave scenarios, all IP-based ML methods become inapplicable, and even the exact solver (Gurobi) is only able to handle the QC case. Our method continues to outperform the approximation algorithms in both settings. Appendix C.8 shows that `Hephaestus` with PPS-I consistently achieves higher solution quality and faster runtime compared to ML-based baselines refined with Gurobi.

Across the four real-world networks under thresholds $T \in \{140\%, 180\%, 220\%, 260\%\}$, the advantage of `Hephaestus` becomes more evident, especially on larger graphs. As shown in Table 1, ours consistently achieves the lowest total budget at the highest threshold $T = 260\%$. On large-scale networks where exact solvers are no longer applicable, `Hephaestus` achieves the most significant improvements. Specifically, it reduces total cost by 28.1% on Skitter and 16.8% on RoadCA compared to the second-best method, DIFFILO. On medium-scale graphs like Gnutella, `Hephaestus` outperforms DIFFILO by 4.5% at the highest threshold, and is only marginally behind it (by 7 units) at the lowest threshold. On the smaller Email network, ours achieves a 1.0% improvement over DIFFILO at the highest threshold and is only 1% worse at the lowest. Due to space limit, we encourage readers to Appendix **C.7** for a more detailed comparison of `Hephaestus` against other baselines in the Quadratic Convex and Log Concave cost functions in real dataset.

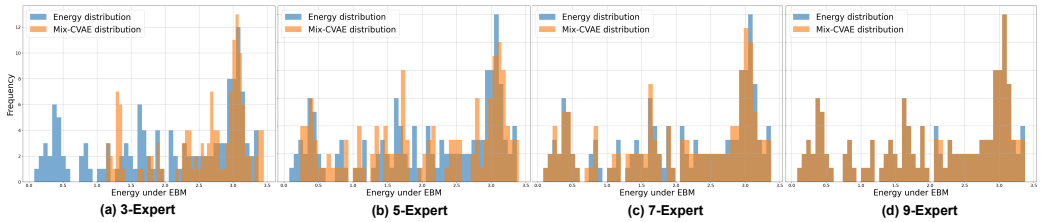


Figure 4: Comparison between the distribution from the EBM and the distribution from the Mix-CVAE with varying numbers of experts on a synthetic dataset with maximum density. Increasing the number of experts improves mode coverage and alignment with the target EBM distribution.

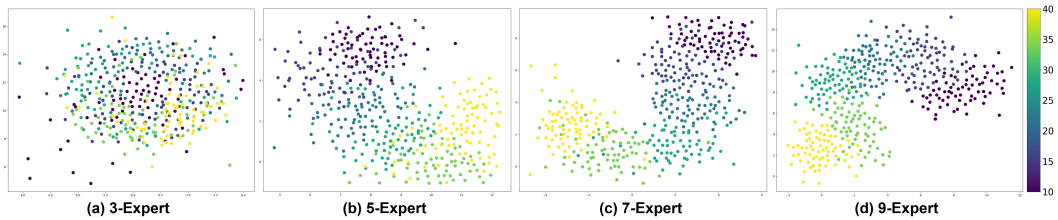


Figure 5: An example of conditional latent space visualization via UMAP [69] for the trained Mix-CVAE on the same synthetic graph and pairs but different thresholds. Points represent latent vectors $\mathbf{z}$, colored by threshold T. Clear clustering shows the latent space captures meaningful patterns.

Impact of Expert Addition in Mix-CVAE guided by EBM. We seek to determine whether Mix-CVAE, when guided by the EBM and augmented with additional experts via our method, can fully capture the distribution induced by the EBM. Figure 4 empirically demonstrates the effectiveness of our expert expansion strategy. As the number of experts in Ω increases from 3 to 9, the distribution of generated samples (orange curves) increasingly aligns with the energy-based target distribution (blue curves). Notably, the generated distribution progressively captures more modes and exhibits a better approximation of both the shape and spread of the underlying the target EBM. This improvement verifies that expert addition enables Mix-CVAE to overcome mode collapse and better reflect the true multimodal structure of the solution space $p(x \mid c)$. Figure 5 further corroborates this benefit via latent space visualization. Using UMAP projections of latent variables z sampled across different thresholds in the same synthetic network, we observe clear cluster formation corresponding to structural and

contextual variations. These clusters become more evident as more experts are added, implying that the augmented latent space encodes richer semantic distinctions. This structured separation is critical for downstream reinforcement learning in Phase 3, where the agent explores the latent space to synthesize stronger attacks. Readers are referred to Appendix **C** for more ablation experiments.

5 Conclusion

In this paper, we introduced `Hephaestus`, a generative mixture model for assessing network QoS under adversarial budget constraints. By combining predictive path stressing with self-improving generative learning, it captures structural vulnerabilities and generalizes across network scales. Experiments on synthetic and real-world graphs show that our framework outperforms both approximation and learning-based baselines in cost, feasibility, and scalability. These results highlight the potential of generative modeling with structured latent search for scalable network topology optimization.

6 Discussion

Limitations. While the proposed learning-based framework performs well on QoSD problems, it still has some key limitations. In theory, the overall architecture can be extended to other combinatorial optimization tasks on graphs, as long as suitable solvers or estimators are available to support gradient-free training and iterative refinement. However, its success relies on the quality of the initial solutions, which are used to train the generative model. Specifically, the framework depends on having a reasonably good approximation algorithm to produce the initial dataset in Forge. In addition, the model's ability to generalize to unseen network instances is influenced by two important factors. First, it depends on how well the GNN-based estimator (such as SPAGAN) can generalize when predicting shortest-path costs on new graphs. Second, it relies on the quality of the graph representation that is used as input to the conditional generative model. If the estimator or the encoder fails to capture meaningful structural features, the overall performance both in terms of feasibility and cost, may degrade. Future research could improve generalization by incorporating graph-invariant features, training on more diverse graph distributions, or using more expressive graph-based foundation models.

Broader Impact. This work proposes a learning-based framework for solving Quality of Service Degradation (QoSD) problems in large-scale networks, with the potential to influence both algorithmic research and practical applications in infrastructure security, communication planning, and critical network vulnerability assessment. By combining generative modeling, energy-based guidance, and reinforcement learning, the framework offers a scalable alternative to traditional combinatorial solvers, enabling approximate yet effective solutions for computationally intractable tasks. This approach can assist network designers and operators in stress test systems, proactively identifying fragile components, and evaluating resilience under adversarial or high-load scenarios. It also provides a reusable architecture for structured optimization over graphs, which could benefit domains such as transportation, power grids, and supply chain logistics.

7 Acknowledgments

This work was partially supported by National Science Foundation grant IIS-2416606. Any opinions, findings, and conclusions or recommendations expressed in this paper, however, are those of the authors and do not necessarily reflect the views of the funding agency.

References

[1] Lan N Nguyen and My T Thai. Network resilience assessment via qos degradation metrics: an algorithmic approach. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(1):1–32, 2019.

[2] Alan Kuhnle, Victoria G. Crawford, and My T. Thai. Network resilience and the length-bounded multicut problem: Reaching the dynamic billion-scale with guarantees. *Proc. ACM Meas. Anal. Comput. Syst.*, 2(1), April 2018.

[3] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 375–392, 2017.

[4] Carlos Pinzón and Camilo Rocha. Double-spend attack models with time advantange for bitcoin. *Electronic Notes in Theoretical Computer Science*, 329:79–103, 2016.

[5] Richard Dennis, Gareth Owenson, and Benjamin Aziz. A temporal blockchain: a formal analysis. In *2016 International Conference on Collaboration Technologies and Systems (CTS)*, pages 430–437. IEEE, 2016.

[6] Qi Alfred Chen, Yucheng Yin, Yiheng Feng, Z Morley Mao, and Henry X Liu. Exposing congestion attack on emerging connected vehicle based traffic signal control. In *NDSS*, 2018.

[7] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, page 6, USA, 2011. USENIX Association.

[8] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage. Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*, pages 447–462, 2010.

[9] Sahar Mazloom, Mohammad Rezaeirad, Aaron Hunter, and Damon McCoy. A security analysis of an {in-vehicle} infotainment and app platform. In *10th USENIX workshop on offensive technologies (WOOT 16)*, 2016.

[10] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

[11] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[12] Eitan Israeli and R. Kevin Wood. Shortest-path network interdiction. *Networks*, 40(2):97–111, 2002.

[13] J. Cole Smith, Mike Prince, and Joseph Geunes. Modern network interdiction problems and algorithms. In *Handbook of Combinatorial Optimization*, pages 1949–1987. Springer, 2013.

[14] J. Cole Smith and Yuyuan Song. A survey of network interdiction models and algorithms. *European Journal of Operational Research*, 283(3):797–811, 2020.

[15] Tien Mai, Avinandan Bose, Arunesh Sinha, Thanh H. Nguyen, and Ayushman K. Singh. Tackling stackelberg network interdiction against a boundedly rational adversary. In *Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2024.

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

[17] Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. From distribution learning in training to gradient search in testing for combinatorial optimization. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[18] Jiayi Zhang, Chang Liu, Xijun Li, Hui-Ling Zhen, Mingxuan Yuan, Yawen Li, and Junchi Yan. A survey for solving mixed integer programming via machine learning. *Neurocomputing*, 519:205–217, 2023.

[19] Xijun Li, Fangzhou Zhu, Hui-Ling Zhen, Weilin Luo, Meng Lu, Yimin Huang, Zhenan Fan, Zirui Zhou, Yufei Kuang, Zhihai Wang, Zijie Geng, Yang Li, Haoyang Liu, Zhiwu An, Muming Yang, Jianshu Li, Jie Wang, Junchi Yan, Defeng Sun, Tao Zhong, Yong Zhang, Jia Zeng, Mingxuan Yuan, Jianye Hao, Jun Yao, and Kun Mao. Machine learning insides optverse ai solver: Design principles and applications, 2024.

[20] He He, Hal Daumé, and Jason Eisner. Learning to search in branch and bound algorithms. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[21] Radu Baltean-Lugojan, Pierre Bonami, Ruth Misener, and Andrea Tramontani. Scoring positive semidefinite cutting planes for quadratic optimization via trained neural networks. *preprint: http://www. optimization-online. org/DB HTML/2018/11/6943. html*, 2019.

[22] Yufei Kuang, Jie Wang, Fangzhou Zhu, Meng Lu, Zhihai Wang, Jia Zeng, Houqiang Li, and Yongdong Zhang. Accelerate presolve in large-scale linear programming via reinforcement learning. *IEEE transactions on pattern analysis and machine intelligence*, PP, 04 2025.

[23] Maxime Gasse, Didier Chetelat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[24] Sirui Li, Wenbin Ouyang, Max B. Paulus, and Cathy Wu. Learning to configure separators in branch-and-cut. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[25] Huigen Ye, Hua Xu, and Hongyan Wang. Light-MILPopt: Solving large-scale mixed integer linear programs with lightweight optimizer and small-scale training dataset. In *The Twelfth International Conference on Learning Representations*, 2024.

[26] Zhihai Wang, Xijun Li, Jie Wang, Yufei Kuang, Mingxuan Yuan, Jia Zeng, Yongdong Zhang, and Feng Wu. Learning cut selection for mixed-integer linear programming via hierarchical sequence model. In *International Conference on Learning Representations*, 2023.

[27] Hongyu Cheng and Amitabh Basu. Learning cut generating functions for integer programming. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 61455–61480. Curran Associates, Inc., 2024.

[28] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, Ravichandra Addanki, Tharindi Hapuarachchi, Thomas Keck, James Keeling, Pushmeet Kohli, Ira Ktena, Yujia Li, Oriol Vinyals, and Yori Zwols. Solving mixed integer programs using neural networks, 2021.

[29] Taehyun Yoon. Confidence threshold neural diving, 2022.

[30] Elias B. Khalil, Christopher Morris, and Andrea Lodi. Mip-gnn: A data-driven framework for guiding combinatorial solvers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10219–10227, 2022.

[31] Huigen Ye, Hua Xu, Hongyan Wang, Chengming Wang, and Yu Jiang. Gnn&gbdt-guided fast optimizing framework for large-scale integer programming. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pages 39864–39878, 2023.

[32] Hao Zeng, Jiaqi Wang, Avirup Das, Junying He, Kunpeng Han, Haoyuan Hu, and Mingfei Sun. Effective generation of feasible solutions for integer programming via guided diffusion. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4107–4118, 2024.

[33] Qingyu Han, Linxin Yang, Qian Chen, Xiang Zhou, Dong Zhang, Akang Wang, Ruoyu Sun, and Xiaodong Luo. A gnn-guided predict-and-search framework for mixed-integer linear programming. *arXiv preprint arXiv:2302.05636*, 2023.

[34] Taoan Huang, Aaron M Ferber, Arman Zharmagambetov, Yuandong Tian, and Bistra Dilkina. Contrastive predict-and-search for mixed integer linear programs. In *Forty-first International Conference on Machine Learning*, 2024.

[35] Gurobi Optimization, LLC. Gurobi optimizer, 2021.

[36] Tobias Achterberg. Scip: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[37] Adam N. Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Management Science*, 68(1):9–26, 2022.

[38] Aaron Ferber, Bryan Wilder, Bistra Dilkina, and Milind Tambe. Mipaal: Mixed integer program as a layer. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, volume 34, pages 1504–1511, 2020.

[39] Arman Zharmagambetov, Brandon Amos, Aaron Ferber, Taoan Huang, Bistra Dilkina, and Yuandong Tian. Landscape surrogate: Learning decision losses for mathematical optimization under partial information. *Advances in Neural Information Processing Systems*, 36, 2024.

[40] Zijie Geng, Jie Wang, Xijun Li, Fangzhou Zhu, Jianye HAO, Bin Li, and Feng Wu. Differentiable integer linear programming. In *The Thirteenth International Conference on Learning Representations*, 2025.

[41] Yiding Yang, Xinchao Wang, Mingli Song, Junsong Yuan, and Dacheng Tao. Spagan: Shortest path graph attention network. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 4099–4105. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[42] Yilun Du and Igor Mordatch. Implicit generation and modeling with energy based models. *Advances in neural information processing systems*, 32, 2019.

[43] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, Fujie Huang, et al. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

[44] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015.

[45] Huyen Nguyen, Hieu Dam, Nguyen Hoang Khoi Do, Cong Tran, and Cuong Pham. Rem: A scalable reinforced multi-expert framework for multiplex influence maximization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(25):27099–27107, Apr. 2025.

[46] Nguyen Hoang Khoi Do, Truc Nguyen, Malik Hassanaly, raed alharbi, Jung Taek Seo, and My T. Thai. Swift hydra: Self-reinforcing generative framework for anomaly detection with multiple mamba models. In *The Thirteenth International Conference on Learning Representations*, 2025.

[47] Puning Zhao, Lifeng Lai, Li Shen, Qingming Li, Jiafei Wu, and Zhe Liu. A huber loss minimization approach to mean estimation under user-level differential privacy. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 130018–130056. Curran Associates, Inc., 2024.

[48] Gregory Meyer. An alternative probabilistic interpretation of the huber loss. In *Conference on Computer Vision and Pattern Recognition*, pages 5257–5265, 06 2021.

[49] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[50] Huy Nguyen, Xing Han, Carl Harris, Suchi Saria, and Nhat Ho. On expert estimation in hierarchical mixture of experts: Beyond softmax gating functions. *arXiv preprint arXiv:2410.02935*, 2024.

[51] Hien Dang, Tho Tran, Tan Nguyen, and Nhat Ho. Neural collapse for cross-entropy class-imbalanced learning with unconstrained relu feature model. *arXiv preprint arXiv:2401.02058*, 2024.

[52] Huy Nguyen, Pedram Akbarian, Fanqi Yan, and Nhat Ho. Statistical perspective of top-k sparse softmax gating mixture of experts. In *The Twelfth International Conference on Learning Representations*, 2024.

[53] Zixiang Chen, Yihe Deng, Yue Wu, Quanquan Gu, and Yuanzhi Li. Towards understanding the mixture-of-experts layer in deep learning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 23049–23062. Curran Associates, Inc., 2022.

[54] Walter R Gilks, Sylvia Richardson, and David Spiegelhalter. *Markov chain Monte Carlo in practice*. CRC press, 1995.

[55] Chelsea C. White and Douglas J. White. Markov decision processes. *European Journal of Operational Research*, 39(1):1–16, 1989.

[56] Sebastian Ruder. An overview of gradient descent optimization algorithms. *ArXiv*, 1609.04747, 2016.

[57] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-explore: A new approach for hard-exploration problems. arxiv. 2019, 1901.

[58] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

[59] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[60] Hao Yin, Austin R Benson, Jure Leskovec, and David F Gleich. Local higher-order graph clustering. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 555–564, 2017.

[61] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2–es, 2007.

[62] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[63] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187, 2005.

[64] Dimitris Bertsimas, Angela King, and Rahul Mazumder. Best subset selection via a modern optimization lens. *The Annals of Statistics*, 44(2):813–852, 2016.

[65] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

[66] Hao Zeng, Jiaqi Wang, Avirup Das, Junying He, Kunpeng Han, Haoyuan Hu, and Mingfei Sun. Effective generation of feasible solutions for integer programming via guided diffusion. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 4107–4118, New York, NY, USA, 2024. Association for Computing Machinery.

[67] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021.

[68] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[69] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims made in the abstract and introduction (Lines 1–18, 67–78) accurately reflect the contributions of the paper: (i) proposing a new generative framework Hephaestus for QoSD with both linear and nonlinear edge weights, (ii) introducing a predictive path-stressing algorithm with provable guarantees, (iii) theoritically modeling solution distributions with an energy-guided mixture of CVAEs, and (iv) refining solutions via latent-space RL.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations of our framework are discussed in the Limitation Section (Appendix/Supplementary file).

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The paper provides theoretical guarantees in Theorem 1 (Line 159), Theorem 2 (Line 209), Theorem 3 (Line 223), Lemma 1 (Line 272), and Theorem 4 (Line 274), with explicit assumptions involving constants such as δ, ε, and bounds on energy/log-ratio terms. Complete proofs are deferred to Appendix B. Due to space limitations, additional corollaries, theorems, and lemmas that support and extend the main results are also included in the appendix.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper provides sufficient detail to reproduce the main experimental results. Section 4 outlines datasets, baselines, evaluation protocols, and QoSD settings. Appendix C includes ablations, dataset statistics, and details on thresholds, graph generation, and tuning parameters for all baselines.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: An anonymized code is in the supplemental material with scripts for training SPAGAN, Mix-CVAE, EBM, and the RL refinement phase, along with instructions and preprocessed data for both synthetic and real datasets.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4 and Appendix C detail dataset, thresholds, sampling strategy, SPAGAN training, CVAE settings, and reinforcement learning parameters.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: While results are consistently benchmarked across multiple thresholds and densities, the paper reports the average performance over 20 independent trials per instance, as stated in Appendix C. However, it does not include error bars, standard deviations, or confidence intervals.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 4 provides timeout settings for baselines (e.g., 3000s–6000s for Gurobi), and Appendix C includes compute specifications and other individual experiment run for training and inference phases.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research complies with the NeurIPS Code of Ethics. All experiments are conducted on publicly available or synthetic datasets. No human subjects or personally identifiable data were involved.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.

- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper explicitly discusses social impacts of the model studied, and the advancement of our solutions. The broader impact statement is in Appendix.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The models developed are not of high misuse risk (e.g., they are not language or image generators), and datasets used are standard graph datasets or synthetic ones without sensitive content.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All datasets (e.g., Email, Gnutella, RoadCA) and tools (e.g., Gurobi, SCIP) are properly cited with references [35], [41], [60–63], and their usage complies with open academic licensing terms.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: They are provided with documentation and instructions in the supplemental material.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: The research does not involve any crowdsourcing or human subject experiments.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.

- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve human subjects; therefore, IRB approval is not applicable.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: No large language models (LLMs) are used in our core methodologies, lemmas and theorems.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

Supplementary for Hephaestus: Mixture Generative Modeling with Energy Guidance for Large-scale QoS Degradation

This supplementary material serves as the appendix to the main paper to provide full algorithmic pseudocode for all components of the framework, detailed derivations and proofs of key theoretical results, comprehensive ablation studies and hyperparameter settings, as well as expanded discussions on implementation to reproduce results, scalability, and generalization. It is organized as follows:

8 DETAILS OF HEPHAESTUS

8.1 Hephaestus Main Framework

Algorithm 1: Hephaestus Main Framework

Input: Initial graph dataset $\mathfrak{D}^{\text{graph}}$, set of critical pairs $\mathbf{K}$, set of thresholds $\mathcal{T}$, hyperparameters for all components.

Output: Trained `Hephaestus` model (SPAGAN $\mathfrak{F}_{\boldsymbol{\theta}}$, EBM q, Mix-CVAE Ω, RL agent π), and ability to generate near-optimal solutions $\mathbf{x}^*$ for new QoSD instances.

`// Initialize storage for solutions`

1 $\mathfrak{D}^{\text{sol}} \leftarrow \emptyset$

`// -- Phase 1: Forge --`

2 $(\mathfrak{F}_{\boldsymbol{\theta}}, \mathfrak{D}^{\text{sol}}_{\text{initial}}) \leftarrow \text{Forge}(\mathfrak{D}^{\text{graph}}, \mathbf{K}, \mathcal{T})$

3 $\mathfrak{D}^{\text{sol}} \leftarrow \mathfrak{D}^{\text{sol}} \cup \mathfrak{D}^{\text{sol}}_{\text{initial}}$

`// -- Phase 2: Morph --`

4 $(q, \Omega) \leftarrow \text{Morph}(\mathfrak{D}^{\text{sol}})$

`// -- Phase 3: Refine (Iterative Self-Reinforcement) --`

5 **for** *each episode* $e = 1, \ldots, E_{\textit{max\_episodes}}$ **do**

6 Top-K-Solutions $\leftarrow \text{Refine}(q, \Omega, \pi, \mathfrak{D}^{\text{sol}}, \mathfrak{F}_{\boldsymbol{\theta}})$

7 $\mathfrak{D}^{\text{sol}} \leftarrow \mathfrak{D}^{\text{sol}} \cup$ Top-K-Solutions `// Augment solution dataset`

8 **if** $e \mod E_{\textit{retrain\_freq}} == 0$ **then**

9 Periodically retrain Morph with augmented data $(q, \Omega) \leftarrow \text{Morph}(\mathfrak{D}^{\text{sol}})$

10 $\pi \leftarrow$ Finalize RL Agent Training from collected experiences

11 **Return** $\mathfrak{F}_{\boldsymbol{\theta}}, q, \Omega, \pi$

The pseudocode for the main framework of `Hephaestus` encapsulates a full pipeline for solving the QoS Degradation (QoSD) problem through a self-reinforcing generative approach. The algorithm begins by initializing an empty solution set (line 1) and entering the Forge phase (line 2), where it trains a Shortest Path Graph Attention Network (SPAGAN) to approximate shortest-path costs, and then runs the Predictive Path Stressing (PPS) algorithm to generate diverse, feasible perturbation solutions across multiple graphs, thresholds, and critical source-target pairs. These solutions are collected into a pretrained dataset $\mathfrak{D}^{\text{sol}}$ (line 3), which is then assigned as the foundation training set for the next phase, Morph (line 4).

In Morph (line 4), an Energy-Based Model (EBM) is trained to estimate the underlying solution density, and a Mixture of Conditional VAEs (Mix-CVAE) is optimized to match this density, with new experts dynamically added to cover poorly modeled regions. Finally, the Refine phase (starting from line 6) trains an RL agent to explore and optimize in the latent space of the Mix-CVAE, improving solution quality while maintaining feasibility. Furthermore, after each episode, the best performing (highest reward) solutions are added back to $\mathfrak{D}^{\text{sol}}$, and for all $E_{\text{retrain\_freq}}$ episodes, the generative model (q, Ω) is periodically re-trained on this augmented dataset (lines 10-11), allowing continual improvement and adaptation. The framework outputs all trained components: SPAGAN, EBM, Mix-CVAE, and RL policy—to synthesize near-optimal QoSD solutions on new graphs (line 12).

8.2 Forge

The Forge phase is responsible for constructing an initial dataset of feasible QoSD perturbation solutions using a graph-learned approximation method. It begins by training an SPAGAN, denoted $\mathfrak{F}_{\boldsymbol{\theta}}$, on the provided set of input graphs $\mathfrak{D}^{\text{graph}}$ (Line 2). This model learns to efficiently approximate the shortest-path costs under varying edge perturbations. An empty set $\mathfrak{D}^{\text{sol}}_{\text{new}}$ is initialized to store solutions (Line 3). Then, for every graph G_i in the dataset (Line 4), and for every configuration of critical source-target pairs $\mathcal{K} \in \mathbf{K}$ and thresholds $T \in \mathcal{T}$ (Lines 5–6), the PPS is executed to produce a perturbation vector $\mathbf{x}^{(i)}_{\mathcal{K},T}$ (Line 7). This vector is expected to increase all relevant shortest path costs exceed the threshold T, using SPAGAN as an efficient surrogate for shortest path cost estimation. Each resulting solution instance is then added to the solution set $\mathfrak{D}^{\text{sol}}_{\text{new}}$ (Line 8), preserving its associated graph, set of critical pairs, and threshold. Once all iterations are complete, the function

returns both the trained SPAGAN model and the newly constructed dataset of feasible perturbations (Line 10), which will serve as the training base for the Morph phase.

Algorithm 2: Forge

1 **Procedure** *Forge*($\mathfrak{D}^{graph}, \mathbf{K}, \mathcal{T}$)
2 $\quad \mathfrak{F}_{\boldsymbol{\theta}} \leftarrow$ Train_SPAGAN($\mathfrak{D}^{\text{graph}}$) `// Train SPAGAN for path cost estimation`
3 $\quad \mathfrak{D}^{\text{sol}}_{\text{new}} \leftarrow \emptyset$;
4 $\quad$ **for** *each graph* $G_i \in \mathfrak{D}^{graph}$ **do**
5 $\quad\quad$ **for** *each critical pair configuration* $\mathcal{K} \in \mathbf{K}$ **do**
6 $\quad\quad\quad$ **for** *each threshold* $T \in \mathcal{T}$ **do**
7 $\quad\quad\quad\quad \mathbf{x}^{(i)}_{\mathcal{K},T} \leftarrow \text{PPS}(G_i, \mathcal{K}, T, \mathfrak{F}_{\boldsymbol{\theta}}, \{f_e\}, \mathbf{b})$;
8 $\quad\quad\quad\quad \mathfrak{D}^{\text{sol}}_{\text{new}} \leftarrow \mathfrak{D}^{\text{sol}}_{\text{new}} \cup \{(G_i, \mathcal{K}, T, \mathbf{x}^{(i)}_{\mathcal{K},T})\}$;
9 $\quad$ **Return** $(\mathfrak{F}_{\boldsymbol{\theta}}, \mathfrak{D}^{\text{sol}}_{\text{new}})$

8.3 SPAGAN Training

This procedure describes the supervised training for SPAGAN, used to approximate shortest-path distances between node pairs in a graph. The function begins by initializing the SPAGAN model $\mathfrak{F}_{\boldsymbol{\theta}}$ (Line 2), which is parameterized to learn over graph-structured input. For a fixed number of training epochs, the model is iteratively updated (starting from line 3). During each epoch, the algorithm samples mini-batches of data—each consisting of a graph G, a source-target pair (s, t), and the corresponding ground-truth shortest-path distance true_dist, typically computed via Dijkstra's algorithm (Line 5). For each instance, the model predicts the baseline shortest-path cost $\widehat{d}_{s,t}$ using the sub-graph (Line 6). The prediction error is measured using the Huber loss function, which provides robustness to outliers and smooth gradients (Line 7). The model parameters are then updated using backpropagation (Line 8), allowing the network to gradually learn a transferable representation of graph structure and path dynamics. Once training converges, the fully trained SPAGAN model $\mathfrak{F}_{\boldsymbol{\theta}}$ is returned (Line 9) to be used in Forge for shortest path estimation.

Algorithm 3: SPAGAN Training with Subgraph Sampling

1 **Procedure** *Train_SPAGAN*($\mathfrak{D}^{graph}$)
2 $\quad$ Initialize SPAGAN model $\mathfrak{F}_{\boldsymbol{\theta}}$;
3 $\quad \mathfrak{D}^{\text{subgraph}} \leftarrow$ ExtractSubgraphs($\mathfrak{D}^{\text{graph}}$) ; `// Generate training subgraphs from full graphs`
4 $\quad$ **for** *each training epoch* **do**
5 $\quad\quad$ **for** *each batch* $(G_{sub}, s, t, true\_dist)$ *from* $\mathfrak{D}^{subgraph}$ **do**
6 $\quad\quad\quad \widehat{d}_{s,t} \leftarrow \mathfrak{F}_{\boldsymbol{\theta}}(G_{\text{sub}}, s, t, \mathbf{0})$ `// Predict baseline shortest path on subgraph`
7 $\quad\quad\quad$ loss $\leftarrow$ HuberLoss($\widehat{d}_{s,t}$, true_dist) ;
8 $\quad\quad\quad$ Update $\mathfrak{F}_{\boldsymbol{\theta}}$ parameters using backpropagation ;
9 $\quad$ **Return** Trained $\mathfrak{F}_{\boldsymbol{\theta}}$

8.4 Predictive Path Stressing (PPS)

Solving the QoS Degradation (QoSD) problem is computationally challenging due to the exponential number of feasible paths and the non-submodular nature of the objective function under nonlinear edge costs. To mitigate these challenges, PPS avoids enumerating all feasible paths and instead incrementally constructs a feasible perturbation vector $\mathbf{x} \in \mathbb{N}^{|E|}$, using shortest-path predictions from a pretrained SPAGAN model $\mathfrak{F}_{\boldsymbol{\theta}}$. Unlike exact methods, PPS relies entirely on SPAGAN to both determine shortest paths and estimate their costs under perturbation. The algorithm starts from an initial perturbation $\mathbf{x}_{\text{initial}}$ (Line 2), and constructs a set $\mathcal{K}_{\text{violate}}$ of source-target pairs whose predicted path costs given by $\mathfrak{F}_{\boldsymbol{\theta}}(G, s, t; \mathbf{x})$—fall below threshold T (Line 3). For each violating pair, it obtains the corresponding shortest path $\rho_{s,t}$ using the SPAGAN shortest-path predictor (Lines 6–8). A soft

potential function is defined as: $\mathcal{C}(P, \mathbf{x}) = \sum_{\rho_{s,t} \in P} \min(T, \mathfrak{F}_{\boldsymbol{\theta}}(G, s, t; \mathbf{x}))$, which serves as a proxy to measure gap to feasibility (Lines 13–14). While $\mathcal{C}(P, \mathbf{x})$ remains less than the relaxed threshold $|P| \cdot T - \bar{\epsilon}$, the algorithm performs updates by selecting the edge e^* and increment Δ^* that lead to the greatest increase in potential per budget unit (Lines 10–24). This is computed by evaluating each candidate update $\mathbf{x}'$ and recomputing the potential function $\mathcal{C}(P, \mathbf{x}')$ using SPAGAN predictions (Lines 17–20), without requiring explicit path enumeration or cost function evaluations. Once the optimal update is applied (Line 24), the algorithm checks whether the potential function $\mathcal{C}(P, \mathbf{x})$ has exceeded the soft feasibility threshold $|P| \cdot T - \bar{\epsilon}$. If so, the set of violating pairs is refreshed by recomputing their shortest paths and corresponding predicted costs using SPAGAN (Line 25). The process repeats until all pairs satisfy the constraint $\mathfrak{F}_{\boldsymbol{\theta}}(G, s, t; \mathbf{x}) \geq T$, at which point the final perturbation vector $\mathbf{x}$ is returned (Line 27).

Algorithm 4: Predictive Path Stressing (PPS)

1 **Procedure** *PPS*$(G = (V, E), \mathcal{K}, T, \{f_e\}, \mathbf{b}, \mathbf{x}_{initial}, \mathfrak{F}_{\boldsymbol{\theta}})$
  Input: Graph $G = (V, E)$, target pairs $\mathcal{K}$, threshold T, edge cost functions $\{f_e\}$, budget box $\mathbf{b}$, initial vector $\mathbf{x}_{\text{initial}}$, trained SPAGAN $\mathfrak{F}_{\boldsymbol{\theta}}$
  Output: Feasible adversarial budget vector $\mathbf{x}$ such that estimated path cost $\sum_{e \in \rho_{s,t}} f_e(x_e) \geq T$ for all $(s, t) \in \mathcal{K}$
2 $\mathbf{x} \leftarrow \mathbf{x}_{\text{initial}}$;
3 $\mathcal{K}_{\text{violate}} \leftarrow \{(s, t) \in \mathcal{K} \mid \sum_{e \in \rho_{s,t}} f_e(x_e) < T,\ \rho_{s,t} = \texttt{SPAGANPath}(\mathfrak{F}_{\boldsymbol{\theta}}, G, s, t; \mathbf{x})\}$;
4 **while** $\mathcal{K}_{violate} \neq \emptyset$ **do**
5 $P \leftarrow \emptyset$;
6 **foreach** $(s, t) \in \mathcal{K}_{violate}$ **do**
7 $\rho_{s,t} \leftarrow \texttt{SPAGANPath}(\mathfrak{F}_{\boldsymbol{\theta}}, G, s, t; \mathbf{x})$;
8 $P \leftarrow P \cup \{\rho_{s,t}\}$;
  `// Evaluate soft potential function`
9 $\mathcal{C}(P, \mathbf{x}) \leftarrow 0$;
10 **while** $\mathcal{C}(P, \mathbf{x}) < |P| \cdot T - \bar{\epsilon}$ **do**
11 $(e^*, \Delta^*, \delta_{\max}) \leftarrow (\text{None}, \text{None}, -\infty)$;
12 $\mathcal{E}_P \leftarrow \bigcup_{\rho \in P} \rho$;
13 **foreach** $\rho_{s,t} \in P$ **do**
14 $\mathcal{C}(P, \mathbf{x}) \leftarrow \mathcal{C}(P, \mathbf{x}) + \min(T, \mathfrak{F}_{\boldsymbol{\theta}}(G, s, t, \mathbf{x}))$;
15 **foreach** $e \in \mathcal{E}_P$ **do**
16 **for** $\Delta = 1$ **to** $b_e - x_e$ **do**
17 $\mathbf{x}' \leftarrow \mathbf{x} + \Delta \cdot \mathbf{1}_e$;
18 $\mathcal{C}(P, \mathbf{x}') \leftarrow 0$;
19 **foreach** $\rho_{s,t} \in P$ **do**
20 $\mathcal{C}(P, \mathbf{x}') \leftarrow \mathcal{C}(P, \mathbf{x}') + \min(T, \mathfrak{F}_{\boldsymbol{\theta}}(G, s, t, \mathbf{x}'))$
21 $\delta \leftarrow \frac{\mathcal{C}(P, \mathbf{x}') - \mathcal{C}(P, \mathbf{x})}{\Delta}$;
22 **if** $\delta > \delta_{\max}$ **then**
23 $(e^*, \Delta^*, \delta_{\max}) \leftarrow (e, \Delta, \delta)$;
  `// Apply optimal update`
24 $\mathbf{x} \leftarrow \mathbf{x} + \Delta^* \cdot \mathbf{1}_{e^*}$;
  `// Update violating pairs using SPAGAN`
25 $\mathcal{K}_{\text{violate}} \leftarrow \{(s, t) \in \mathcal{K} \mid \sum_{e \in \rho_{s,t}} f_e(x_e) < T,\ \rho_{s,t} = \texttt{SPAGANPath}(G, s, t; \mathbf{x})\}$;
26 **Return** $\mathbf{x}$

8.5 Morph

The Morph phase is designed to model the distribution of high-quality QoSD solutions using a generative framework guided by energy-based learning. Intuitively, the goal is to make the Energy-Based Model (EBM) q_θ approximate the true—but unknown—solution distribution as closely as possible. To do this, the EBM is trained to assign *low energy* (i.e., high likelihood) to real solutions

$\mathbf{x}_{\text{real}} \sim \mathfrak{D}^{\text{sol}}$, effectively pulling its density toward regions with meaningful feasible solutions. At the same time, it is encouraged to assign *high energy* (i.e., low likelihood) to generated (fake) samples $\mathbf{x}_{\text{fake}} \sim \Omega$, which come from the current generative model Ω. This adversarial learning setup drives the EBM away from areas the generator covers poorly, creating a pressure that helps both models evolve: the EBM becomes more selective, and the generator learns to cover harder regions.

Algorithm 5: Morph

1 **Procedure** *Morph($\mathfrak{D}^{sol}$)*
2 Initialize EBM q_θ and Mix-CVAE $\Omega = [\Omega_0, \ldots, \Omega_N]$ (initially $N = 0$ for first expert) ;
3 **for** *each minimax training iteration* $k = 1, \ldots, K_{max\_morph}$ **do**
`// Update EBM` q_θ
4 Sample $(\mathbf{x}_{\text{real}}, \mathbf{c})$ from $\mathfrak{D}^{\text{sol}}$;
5 Sample $\mathbf{z}_{\text{fake}} \sim \tilde{p}_\phi(\mathbf{z}|\mathbf{c})$ (from any expert Ω_i);
6 $\mathbf{x}_{\text{fake}} \leftarrow \mathcal{M}_\phi(\mathbf{z}_{\text{fake}}, \mathbf{c})$ (from current Ω's decoder $\mathcal{M}_\phi$);
7 $L_q \leftarrow \mathbb{E}_{(\mathbf{x}_{\text{real}},\mathbf{c})\sim\mathfrak{D}^{\text{sol}}}[E_\theta(\mathbf{x}_{\text{real}})] - \mathbb{E}_{\mathbf{x}_{\text{fake}}\sim\Omega}[E_\theta(\mathbf{x}_{\text{fake}})] + \gamma(\mathbb{E}[E_\theta(\mathbf{x}_{\text{real}})^2] + \mathbb{E}[E_\theta(\mathbf{x}_{\text{fake}})^2])$;
Update parameters θ of EBM to minimize L_q;
`// Update Mix-CVAE` Ω
8 Sample $(\mathbf{x}_{\text{real}}, \mathbf{c})$ from $\mathfrak{D}^{\text{sol}}$;
9 $L_\Omega \leftarrow 0$;
10 **for** *each expert* $\Omega_i = (\mathcal{P}_\psi^{(i)}, \mathcal{M}_\phi^{(i)}) \in \Omega$ **do**
11 $\mathbf{z}_{\text{encoded}} \leftarrow \mathcal{P}_\psi^{(i)}(\mathbf{x}_{\text{real}}, \mathbf{c})$;
12 $L_{\Omega_i}^{ELBO} \leftarrow \mathbb{E}_{\mathbf{z}\sim\tilde{q}_{\psi^{(i)}}(\mathbf{z}|\mathbf{x}_{\text{real}},\mathbf{c})}[\log \tilde{p}_{\phi^{(i)}}(\mathbf{x}_{\text{real}} \mid \mathbf{z}, \mathbf{c})] - \mathrm{KL}[\tilde{q}_{\psi^{(i)}}(\mathbf{z} \mid \mathbf{x}_{\text{real}}, \mathbf{c}) \,\|\, \tilde{p}_{\phi^{(i)}}(\mathbf{z} \mid \mathbf{c})]$
13 Sample $\mathbf{z}_{\text{prior}} \sim \tilde{p}_{\phi^{(i)}}(\mathbf{z} \mid \mathbf{c})$;
14 $\mathbf{x}_{\text{generated}} \leftarrow \mathcal{M}_\phi^{(i)}(\mathbf{z}_{\text{prior}}, \mathbf{c})$;
15 $L_{\Omega_i}^{guide} \leftarrow L_{\Omega_i}^{ELBO} + \lambda \cdot E_\theta(\mathbf{x}_{\text{generated}})$ `// Penalize high energy generations`
16 $L_\Omega \leftarrow L_\Omega + L_{\Omega_i}^{guide}$ `// Potentially use gating weights here`
17 Update Mix-CVAE parameters ψ, ϕ for all experts to minimize L_Ω ;
`// Expert Addition Strategy`
18 **if** $k \pmod{K_{check\_expert}} == 0$ **then**
19 Sample $\mathbf{x}_{\text{check}}$ from $\mathfrak{D}^{\text{sol}}$ or generate from current Ω;
20 $\chi(\mathbf{x}_{\text{check}}) \leftarrow \log(q_\theta(\mathbf{x}_{\text{check}})/\Omega(\mathbf{x}_{\text{check}}|\mathbf{c}_{\text{check}}))$ `// Density ratio`
21 **if** $\chi(\mathbf{x}_{check}) > \delta_{expert\_add}$ *and current number of experts* $< N_{max}$ **then**
22 Add a new CVAE expert Ω_{N+1} to Ω ;
23 Initialize/Train Ω_{N+1} (e.g., focused on data from regions where $\chi > \delta_{\text{expert\_add}}$ or re-train mixture) ;
24 $N \leftarrow N + 1$;
25 **Return** (Trained q_θ, Trained Ω)

Formally, the algorithm starts by initializing both the EBM q_θ and a mixture of Conditional VAEs $\Omega = [\Omega_0, \ldots, \Omega_N]$, starting with one single expert (Line 2). Each minimax training iteration proceeds in two stages. First, the EBM is updated by contrasting energy scores between real samples and generated ones. Fake samples are produced by sampling a latent vector $\mathbf{z}_{\text{fake}} \sim \tilde{p}_\phi(\mathbf{z} \mid \mathbf{c})$ from the prior of any expert and decoding it via decoder $\mathcal{M}_\phi$ of Ω (Lines 4–6). The EBM loss pushes energy lower on real samples and higher on generated ones, with variance-based regularization to stabilize training (Line 7), and the parameters θ are updated accordingly (Line 8).

Next, the generative model Ω is updated (Lines 10–17). For each expert Ω_i, the encoder maps real inputs to latent space, and the decoder reconstructs the solution. The training objective combines the standard ELBO which promotes good reconstruction and posterior–prior alignment—with an energy penalty term (Line 15). This penalty uses the EBM to discourage high-energy generations, i.e., samples that lie in unrealistic or undersampled regions. The total loss L_Ω aggregates across all experts and is minimized to improve the generative model's coverage of low-energy regions (Line 16). To adaptively expand model capacity, the algorithm includes a periodic *expert addition*

strategy (Lines 21–28). Every $K_{\text{check\_expert}}$ steps, it evaluates whether the current mixture Ω underfits any region of the solution space using a density ratio test: $\chi(\mathbf{x}) = \log(q_\theta(\mathbf{x})/\Omega(\mathbf{x} \mid \mathbf{c}))$. If this score exceeds a threshold $\delta_{\text{expert\_add}}$, indicating insufficient generative density, a new CVAE expert is added and trained specifically on that difficult region. This enables the generator to incrementally cover diverse and potentially multimodal distributions. The process continues until the maximum number of experts is reached or training ends. Finally, Morph returns both the trained EBM and the mixture-based generative model.

8.6 Refine

Algorithm 6: Refine

1 **Procedure** *Refine*$(q, \Omega, \pi, \mathfrak{D}^{sol}, \mathfrak{F}_{\boldsymbol{\theta}})$
  Input: Energy model q, generative model Ω, RL policy π, solution dataset $\mathfrak{D}^{\text{sol}}$, SPAGAN estimator $\mathfrak{F}_{\boldsymbol{\theta}}$
  Output: Top-K refined feasible solutions with low cost
2 Initialize $\bar{\mathcal{S}}_{\text{new}} \leftarrow \emptyset$ `// Storage for solutions generated in this episode`
3 **for** *each RL training step* $s = 1, \ldots, S_{max}$ **do**
4 Sample instance $(G, \mathcal{K}, T, \mathbf{x})$ from $\mathfrak{D}^{\text{sol}}$;
5 $\mathbf{c} \leftarrow [G, \mathcal{K}, T]$ `// Context input`
6 $\mathbf{z}_{\text{current}} \leftarrow \mathcal{P}_\psi(\mathbf{x}, \mathbf{c})$ `// Encode current solution into latent space`
7 **for** *each step* $t = 1, \ldots, \mathfrak{T}_{max}$ **do**
8 $(\mu_t, \sigma_t) \leftarrow \pi(\text{state}(\mathbf{z}_{\text{current}}, \mathbf{c}))$ `// Sample action`
9 $\epsilon \sim \mathcal{N}(0, I)$;
10 $\delta_t \leftarrow \mu_t + \sigma_t \cdot \epsilon$ `// Perturb latent`
11 $\mathbf{z}_{\text{next}} \leftarrow \mathbf{z}_{\text{current}} + \delta_t$;
12 $\hat{\mathbf{x}} \leftarrow \mathcal{M}_\phi(\mathbf{z}_{\text{next}}, \mathbf{c})$ `// Decode to solution`
13 $\bar{\mathbf{x}} \leftarrow \log(1 + e^{\hat{\mathbf{x}}})$ `// Soft transform for reward`
  `// Evaluate feasibility score via SPAGAN approximation`
14 $f\text{score} \leftarrow \sum_{(s_p,t_p)\in\mathcal{K}} \frac{1}{1+\exp(-\zeta(\mathfrak{F}_{\boldsymbol{\theta}}(G,s_p,t_p;\text{round}(\hat{\mathbf{x}}))-T))}$
  `// Compute reward (Eq. 6)`
15 $R_t \leftarrow f\text{score} - \varkappa \cdot \log(1 + \|\bar{\mathbf{x}}\|_1)$
16 Store transition $((\mathbf{z}_{\text{current}}, \mathbf{c}), (\mu_t, \sigma_t), R_t, (\mathbf{z}_{\text{next}}, \mathbf{c}))$ in replay buffer ;
17 Update policy π using replay buffer (e.g., PPO, DDPG, gradient-ascent, etc.) ;
18 $\mathbf{z}_{\text{current}} \leftarrow \mathbf{z}_{\text{next}}$;
19 **if** $R_t \geq R_{thresh}$ *or* $t = \mathfrak{T}_{max}$ **then**
20 $\mathbf{x}_{\text{refined}} \leftarrow \text{PPS-I}(G, \mathcal{K}, T, \hat{\mathbf{x}})$ `// Ensure 100% feasibility`
21 $\bar{\mathcal{S}}_{\text{new}} \leftarrow \bar{\mathcal{S}}_{\text{new}} \cup \{(G, \mathcal{K}, T, \mathbf{x}_{\text{refined}})\}$;
22 **Break**
  `// Select best K solutions based on true cost`
23 Sort $\bar{\mathcal{S}}_{\text{new}}$ in ascending order of $\|\mathbf{x}_{\text{refined}}\|_1$;
24 Select top K entries as $\bar{\mathcal{S}}_{\text{topK}}$;
25 **Return** $\bar{\mathcal{S}}_{\text{topK}}$

This algorithm implements a single episode of the Refine phase, which performs latent-space optimization via reinforcement learning to improve solution quality for the QoSD problem. The goal is to generate low-cost, feasible perturbation vectors by guiding a latent policy network using a differentiable reward structure. The algorithm begins by initializing an empty buffer $\bar{\mathcal{S}}_{\text{new}}$ to store high-quality solutions generated during the episode (Line 2). At each RL training step (Line 3), an instance $(G, \mathcal{K}, T, \mathbf{x})$ is sampled from the solution dataset $\mathfrak{D}^{\text{sol}}$, and the corresponding context vector $\mathbf{c} = [G, \mathcal{K}, T]$ is constructed (Line 5). The current solution $\mathbf{x}$ is encoded into latent space via the encoder $\mathcal{P}_\psi$ of any CVAE expert $\Omega_i \in \Omega$ to yield $\mathbf{z}_{\text{current}}$ (Line 6). An RL trajectory is then simulated over $\mathfrak{T}_{\text{max}}$ steps (Line 7). At each step t, the RL agent samples an action $a_t = (\mu_t, \sigma_t)$ from the policy network given the current state (Line 8), and applies a stochastic perturbation to the latent vector using Gaussian noise $\epsilon \sim \mathcal{N}(0, I)$ (Lines 9–10). The next latent state $\mathbf{z}_{\text{next}}$ is computed and decoded into a candidate perturbation vector $\hat{\mathbf{x}}$ using the decoder $\mathcal{M}_\phi$ (Lines 11–12). A soft

transformation $\bar{\mathbf{x}} = \log(1 + e^{\hat{\mathbf{x}}})$ is then applied to allow stable reward evaluation (Line 13). To assess the quality of the decoded solution, a feasibility score is computed based on SPAGAN's predictions of shortest path costs for each critical pair (Line 14), followed by a differentiable reward R_t that penalizes excessive cost via a logarithmic budget term (Line 16). The resulting transition is stored in a replay buffer, and the RL policy is updated using any standard algorithm such as PPO, DDPG, or curiosity-driven exploration (Lines 17). The latent vector is then updated for the next step (Line 18). If the reward exceeds a predefined threshold R_{thresh} or the trajectory reaches the time limit (Line 19), the decoded solution is passed through PPS-I (Line 20) to continue refine and ensure exact feasibility. The resulting solution is stored in $\bar{\mathcal{S}}_{\text{new}}$ (Line 21), and the trajectory is terminated (Line 22). After all episodes are completed, the algorithm ranks the refined solutions by their true cost $\|\mathbf{x}_{\text{refined}}\|_1$ (Line 23), selects the top-K best ones (Line 24), and returns them as the output of the refinement episode (Line 25). These solutions are later fed back into the self-reinforcement loop, improving both the generative model and the energy function in subsequent iterations.

8.7 Inference Process

Algorithm 7: Inference Process

1 **Procedure** *Inference*($G_{new}, \mathcal{K}_{new}, T_{new}, \Omega, \pi, \mathfrak{F}_{\boldsymbol{\theta}}$)
  Input: New instance $(G_{\text{new}}, \mathcal{K}_{\text{new}}, T_{\text{new}})$, trained $\Omega, \pi, \mathfrak{F}_{\boldsymbol{\theta}}$
  Output: Near-optimal feasible solution $\mathbf{x}^*_{\text{final}}$
2 $\mathbf{c}_{\text{new}} \leftarrow [G_{\text{new}}, \mathcal{K}_{\text{new}}, T_{\text{new}}]$;
3 Sample initial latent vector $\mathbf{z}_{\text{init}}$ (e.g., from Ω's prior $\tilde{p}_\phi(\mathbf{z}|\mathbf{c}_{\text{new}})$ or encode a heuristic solution) ;
4 $\mathbf{z}^* \leftarrow \mathbf{z}_{\text{init}}$;
  `// RL agent refines latent vector for the new instance`
5 **for** $k = 1, \ldots, K_{\textit{inference\_steps}}$ **do**
6 Action $a = (\mu, \sigma) \leftarrow \pi(\text{state}(\mathbf{z}^*, \mathbf{c}_{\text{new}}))$ `// RL policy acts on current latent state`
7 $\epsilon_{\text{noise}} \sim \mathcal{N}(0, I)$ `// Exploration or deterministic if` σ `is small`
8 $\delta \leftarrow \sigma \cdot \epsilon_{\text{noise}} + \mu$;
9 $\mathbf{z}^* \leftarrow \mathbf{z}^* + \delta$;
10 $\mathbf{x}_{\text{raw}} \leftarrow \mathcal{M}_\phi(\mathbf{z}^*, \mathbf{c}_{\text{new}})$ (Decoder from Ω) ;
11 $\mathbf{x}^*_{\text{final}} \leftarrow$ PPS-I$(G_{\text{new}}, \mathcal{K}_{\text{new}}, T_{\text{new}}, \mathbf{x}_{\text{raw}})$ `// Final refinement for feasibility`
12 **Return** $\mathbf{x}^*_{\text{final}}$

In Algorithm 7, the inference process of the Hephaestus framework aims to generate near-optimal, feasible solutions on a new unseen QoSD instance $(G_{\text{new}}, \mathcal{K}_{\text{new}}, T_{\text{new}})$ using the trained models Ω, π, and $\mathfrak{F}_{\boldsymbol{\theta}}$. The goal is to leverage the learned latent-space generator and RL policy to efficiently synthesize a high-quality perturbation vector without needing to re-run the full training pipeline. The process begins by constructing the context vector $\mathbf{c}_{\text{new}} = [G_{\text{new}}, \mathcal{K}_{\text{new}}, T_{\text{new}}]$ (Line 2). An initial latent vector $\mathbf{z}_{\text{init}}$ is then obtained either by sampling from the prior of the trained Mix-CVAE $\tilde{p}_\phi(\mathbf{z} \mid \mathbf{c}_{\text{new}})$ (Line 3). This serves as the starting point for iterative improvement. The current latent solution is set to $\mathbf{z}^* = \mathbf{z}_{\text{init}}$ (Line 4). To refine this latent vector, the trained RL policy π is applied iteratively (Lines 5–9). At each step k, the policy takes the current latent state and produces an action $a = (\mu, \sigma)$, which defines a mean and uncertainty over latent perturbations (Line 6). A Gaussian perturbation δ is sampled and applied to the latent vector $\mathbf{z}^*$, gradually steering the solution toward more feasible and lower-cost regions (Lines 9–10). This process continues either for a fixed number of inference steps $K_{\text{inference\_steps}}$ or until the reward (implicitly computed within the policy) stabilizes. After refinement, the latent code $\mathbf{z}^*$ is decoded into a raw perturbation vector $\mathbf{x}_{\text{raw}}$ using the decoder $\mathcal{M}_\phi$ of the generative model Ω (Line 10). To ensure full feasibility, this vector is passed through the PPS-I post-processing module (Line 11), which guarantees that the final solution $\mathbf{x}^*_{\text{final}}$ satisfies the QoSD constraints. The final output is then returned (Line 12).

8.8 Predictive Path Stressing - Inference (PPS-I)

This procedure describes PPS-I, a variant of the Predictive Path Stressing (PPS) algorithm that retains the same iterative update mechanism over shortest-path constraints. However, PPS-I differs from PPS in two key aspects. First, instead of relying on SPAGAN predictions, it uses Dijkstra's algorithm

to compute exact shortest paths $\rho_{s,t}$ and evaluates their true costs using the nonlinear edge functions $f_e(x_e)$, as shown in Lines 7, 8 and 27. This guarantees correctness under arbitrary cost functions but incurs higher computational overhead. Second, PPS-I accepts a non-zero initial perturbation vector $\mathbf{x}_{\text{initial}}$ as input (Line 3), allowing it to continually refine approximate solutions produced by learning-based modules rather than starting from scratch. The rest of the logic—computing soft potential (Lines 13–15), evaluating marginal gain for candidate updates (Lines 16–23), and selecting optimal edge increments (Line 26)—remains structurally similar to PPS. The set of violating pairs is refreshed using recomputed exact shortest paths (Line 27), and the process continues until full feasibility is achieved.

Algorithm 8: Predictive Path Stressing - Inference (PPS-I)

1 **Procedure** *PPS-I*($G = (V, E), \mathcal{K}, T, \{f_e\}, \mathbf{b}, \mathbf{x}_{initial}$)
  Input: Graph $G = (V, E)$, critical pairs $\mathcal{K}$, threshold T, edge cost functions $\{f_e\}$, budget bounds $\mathbf{b}$, initial solution $\mathbf{x}_{\text{initial}}$
  Output: Feasible budget vector $\mathbf{x}$ such that $\sum_{e\in\rho_{s,t}} f_e(x_e) \geq T$ for all $(s,t) \in \mathcal{K}$
2 $\mathbf{x} \leftarrow \mathbf{x}_{\text{initial}}$;
3 $\mathcal{K}_{\text{violate}} \leftarrow \{(s,t) \in \mathcal{K} \mid \sum_{e\in\rho_{s,t}} f_e(x_e) < T,\ \rho_{s,t} = \texttt{DijkstraPath}(G, s, t; \mathbf{x})\}$;
4 **while** $\mathcal{K}_{violate} \neq \emptyset$ **do**
5 $P \leftarrow \emptyset$ `// Shortest paths for current violations`
6 **foreach** $(s,t) \in \mathcal{K}_{violate}$ **do**
7 $\rho_{s,t} \leftarrow \texttt{DijkstraPath}(G, s, t; \mathbf{x})$;
8 $P \leftarrow P \cup \{\rho_{s,t}\}$;
  `// Evaluate potential function`
9 $\mathcal{C}(P, \mathbf{x}) \leftarrow 0$;
10 **while** $\mathcal{C}(P, \mathbf{x}) < |P| \cdot T - \bar{\epsilon}$ **do**
11 $(e^*, \Delta^*, \delta_{\max}) \leftarrow (\text{None}, \text{None}, -\infty)$;
12 $\mathcal{E}_P \leftarrow \bigcup_{\rho\in P} \rho$;
13 **foreach** $\rho_{s,t} \in P$ **do**
14 $c_\rho \leftarrow \sum_{e\in\rho_{s,t}} f_e(x_e)$;
15 $\mathcal{C}(P, \mathbf{x}) \leftarrow \mathcal{C}(P, \mathbf{x}) + \min(T, c_\rho)$;
16 **foreach** $e \in \mathcal{E}_P$ **do**
17 **for** $\Delta = 1$ **to** $b_e - x_e$ **do**
18 $\mathbf{x}' \leftarrow \mathbf{x} + \Delta \cdot \mathbf{1}_e$;
19 $\mathcal{C}(P, \mathbf{x}') \leftarrow 0$;
20 **foreach** $\rho_{s,t} \in P$ **do**
21 $c'_\rho \leftarrow \sum_{e'\in\rho_{s,t}} f_{e'}(x'_{e'})$;
22 $\mathcal{C}(P, \mathbf{x}') \leftarrow \mathcal{C}(P, \mathbf{x}') + \min(T, c'_\rho)$;
23 $\delta \leftarrow \frac{\mathcal{C}(P,\mathbf{x}') - \mathcal{C}(P,\mathbf{x})}{\Delta}$;
24 **if** $\delta > \delta_{\max}$ **then**
25 $(e^*, \Delta^*, \delta_{\max}) \leftarrow (e, \Delta, \delta)$;
  `// Apply optimal update`
26 $\mathbf{x} \leftarrow \mathbf{x} + \Delta^* \cdot \mathbf{1}_{e^*}$;
  `// Update violating pairs`
27 $\mathcal{K}_{\text{violate}} \leftarrow \{(s,t) \in \mathcal{K} \mid \sum_{e\in\rho_{s,t}} f_e(x_e) < T,\ \rho_{s,t} = \texttt{DijkstraPath}(G, s, t; \mathbf{x})\}$;
28 **Return** $\mathbf{x}$

9 THEOREMS AND PROOFS

9.1 Predictive Path Stressing Algorithm Ratio

Theorem 1 (PPS Ratio) Let $h = \lceil T/w_{\min} \rceil$, where $w_{\min} = \min_{e\in E} w_e$. Assume that the set $\mathcal{E}$ is chosen from E such that $\Pr[\mathcal{E}^* \subseteq \mathcal{E}] = \mathsf{a}$, where $\mathcal{E}^*$ is the set of edges of the optimal solution and

$\mathsf{a} \in (0,1)$ is a constant. Given a parameter $\bar{\epsilon} > 0$, then running the Predictive Path Stressing algorithm on $\mathcal{E}$ yields a solution $\mathbf{x}$ such that $\mathcal{C}(P,\mathbf{x}) \geq |P|T - \bar{\epsilon}$ and $\mathbb{E}[\|\mathbf{x}\|_1] \leq \frac{(1+h\ln(n)+\ln T+\ln(1/\bar{\epsilon}))}{\mathsf{a}}\mathsf{OPT}$

Proof. Recall that $\mathcal{C}(P,\mathbf{x}) = \sum_{p\in P}\min\Big(T,\ \sum_{e\in p} f_e(x_e)\Big)$ and thus $\mathcal{C}(P,\cdot)$ is monotone. Denote $\mathbf{x}^i$ is the partial solution after iteration i of the algorithm, e_i is the edge added to the solution in iteration i with $x_{e_i} = j_i$ and $\mathbf{u}(e_i, j_i)$ be a vector we add to the solution $\mathbf{x}^i$ in the i-th iteration.

Since the Predictive Path Stressing algorithm selects, in each iteration i, the edge e that maximizes the marginal gain per unit cost, given by $\frac{\mathcal{C}(P,\mathbf{x}^i+\mathbf{u}(e,j_i))-\mathcal{C}(P,\mathbf{x}^i)}{j_i}$. If $\mathcal{E}^* \subseteq \mathcal{E}$, this value is at least as large as the average marginal gain per unit cost in the optimal solution given by $\frac{|P|T-\mathcal{C}\left(P,\mathbf{x}^i\right)}{\mathsf{OPT}}$. Therefore, given the solution $\mathbf{x}^{i-1}$, we have

$$\frac{\mathcal{C}(P,\mathbf{x}^i) - \mathcal{C}(P,\mathbf{x}^{i-1})}{j_i} = \frac{\mathcal{C}(P,\mathbf{x}^{i-1}+\mathbf{u}(e_i,j_i)) - \mathcal{C}(P,\mathbf{x}^{i-1})}{j_i} \tag{7}$$

$$\geq \frac{|P|T - \mathcal{C}(P,\mathbf{x}^{i-1})}{\mathsf{OPT}} \tag{8}$$

$$\implies \quad \mathcal{C}(P,\mathbf{x}^i) - \mathcal{C}(P,\mathbf{x}^{i-1}) \geq \frac{j_i}{\mathsf{OPT}}(|P|T - \mathcal{C}(P,\mathbf{x}^{i-1}). \tag{9}$$

Therefore,

$$\mathbb{E}\Big[\mathcal{C}(P,\mathbf{x}^i) - \mathcal{C}(P,\mathbf{x}^{i-1})|\mathbf{x}^{i-1}\Big] \tag{10}$$

$$\geq \mathbb{E}\Big[\Pr[\mathcal{E}^* \subseteq \mathcal{E}] \cdot \frac{j_i}{\mathsf{OPT}}(|P|T - c(P,\mathbf{x}^{i-1})) + (1 - \Pr[\mathcal{E}^* \subseteq \mathcal{E}]) \cdot 0|\mathbf{x}^{i-1}\Big] \tag{11}$$

$$\geq \mathbb{E}\Big[\frac{\mathsf{a}j_i}{\mathsf{OPT}}(|P|T - c(P,\mathbf{x}^{i-1}))\mid \mathbf{x}^{i-1}\Big]. \tag{12}$$

By taking expectation over $\mathbf{x}^{i-1}$, we obtain

$$\mathbb{E}\Big[\mathcal{C}(P,\mathbf{x}^i) - \mathcal{C}(P,\mathbf{x}^{i-1})\Big] \geq \mathbb{E}\Big[\frac{\mathsf{a}j_i}{\mathsf{OPT}}(|P|T - \mathcal{C}(P,\mathbf{x}^{i-1}))\Big]. \tag{13}$$

Re-arranging the above inequality gives:

$$\mathbb{E}\Big[|P|T - \mathcal{C}(P,\mathbf{x}^i)\Big] \leq \mathbb{E}\Big[|P|T - (\mathcal{C}(P,\mathbf{x}^{i-1}) + \frac{\mathsf{a}j_i}{\mathsf{OPT}}\left(|P|T - \mathcal{C}(P,\mathbf{x}^{i-1})\right))\Big] \tag{14}$$

$$\leq \mathbb{E}\Big[\left(|P|T - \mathcal{C}(P,\mathbf{x}^{i-1})\right) - \frac{\mathsf{a}j_i}{\mathsf{OPT}}\left(|P|T - \mathcal{C}(P,\mathbf{x}^{i-1})\right)\Big] \tag{15}$$

$$= \mathbb{E}\Big[(1 - \frac{\mathsf{a}j_i}{\mathsf{OPT}})\left(|P|T - \mathcal{C}(P,\mathbf{x}^{i-1})\right)\Big]. \tag{16}$$

Let t is the number iteration. By applying the inequality (16) iteratively over these t iterations, we obtain the following:

$$\begin{aligned}
\mathbb{E}\Big[|P|T - \mathcal{C}(P,\mathbf{x}^t)\Big] &\leq \mathbb{E}\Big[|P|T\prod_{i=1}^{t}\left(1 - \frac{\mathsf{a}j_i}{\mathsf{OPT}}\right)\Big] \\
&\leq \mathbb{E}\Big[|P|T\prod_{i=1}^{t}\exp\left(-\frac{\mathsf{a}j_i}{\mathsf{OPT}}\right)\Big] \quad \text{(using } 1 - z \leq e^{-z}\text{)} \\
&\leq \mathbb{E}\Big[|P|T\exp\left(-\frac{\mathsf{a}\sum_{i=1}^{t}j_i}{\mathsf{OPT}}\right)\Big] \\
&\leq \mathbb{E}\Big[|P|T\exp\left(-\frac{\mathsf{a}\|\mathbf{x}^t\|_1}{\mathsf{OPT}}\right)\Big].
\end{aligned}$$

By the terminal condition of the algorithm, $\mathcal{C}(P, \mathbf{x}^t) \geq |P|T - \bar{\epsilon}$ and $\mathcal{C}(P, \mathbf{x}^{t-1}) < |P|T - \bar{\epsilon}$ so we have

$$\bar{\epsilon} \leq |P|T - \mathcal{C}\left(P, \mathbf{x}^{t-1}\right) \leq |P|T \cdot \exp\left(-\frac{\mathsf{a}\|\mathbf{x}^{t-1}\|}{\mathsf{OPT}}\right) \tag{17}$$

$$\iff |P|T \cdot \exp\left(-\frac{\mathsf{a}\|\mathbf{x}^{t-1}\|_1}{\mathsf{OPT}}\right) \geq \bar{\epsilon} \tag{18}$$

$$\iff \exp\left(-\frac{\mathsf{a}\|\mathbf{x}^{t-1}\|_1}{\mathsf{OPT}}\right) \geq \frac{\bar{\epsilon}}{|P|T} \tag{19}$$

$$\iff \frac{\mathsf{a}\|\mathbf{x}^{t-1}\|_1}{\mathsf{OPT}} \leq \ln\left(\frac{|P|T}{\bar{\epsilon}}\right) \tag{20}$$

$$\iff \|\mathbf{x}^{t-1}\|_1 \leq \frac{\mathsf{OPT}}{\mathsf{a}} \ln\left(\frac{|P|T}{\bar{\epsilon}}\right). \tag{21}$$

Besides, from the inequality (13), we also have (in expectation)

$$\mathcal{C}\left(P, \mathbf{x}^{t-1}\right) + \frac{\mathsf{a}j_t}{\mathsf{OPT}}\left(|P|T - \mathcal{C}\left(P, \mathbf{x}^{t-1}\right)\right) \leq \mathcal{C}\left(P, \mathbf{x}^{t}\right) \tag{22}$$

$$\iff \frac{\mathsf{a}j_t}{\mathsf{OPT}}\left(|P|T - \mathcal{C}\left(P, \mathbf{x}^{t-1}\right)\right) \leq \mathcal{C}\left(P, \mathbf{x}^{t}\right) - \mathcal{C}\left(P, \mathbf{x}^{t-1}\right) \tag{23}$$

$$\iff \mathsf{a}j_t\left(|P|T - \mathcal{C}\left(P, \mathbf{x}^{t-1}\right)\right) \leq \mathsf{OPT}\left(\mathcal{C}\left(P, \mathbf{x}^{t}\right) - \mathcal{C}\left(P, \mathbf{x}^{t-1}\right)\right) \tag{24}$$

$$\implies \mathsf{a}j_t \leq \mathsf{OPT} \;\; (\text{Since } |P|T - \mathcal{C}\left(P, \mathbf{x}^{t-1}\right) > \mathcal{C}\left(P, \mathbf{x}^{t}\right) - \mathcal{C}\left(P, \mathbf{x}^{t-1}\right)) \tag{25}$$

$$\implies j_t \leq \frac{\mathsf{OPT}}{\mathsf{a}}. \tag{26}$$

The set P of feasible paths can be upper bounded in terms of the maximum path length and the number of nodes. In particular, the number of edges of a feasible path is upper-bounded by $h = \left\lceil \frac{T}{w_{\min}} \right\rceil$ (since each edge has weight at least $w_{\min}$), the number of feasible paths of the is upper-bounded by n^h. We therefore have:

$$\mathbb{E}[\|\mathbf{x}\|_1] = \mathbb{E}[\|\mathbf{x}^{t-1}\|_1] + \mathbb{E}[j_t] \leq \frac{\mathsf{OPT}}{\mathsf{a}} + \frac{\mathsf{OPT}\ln\left(\frac{|P|T}{\bar{\epsilon}}\right)}{\mathsf{a}} \tag{27}$$

$$\leq \frac{\mathsf{OPT}}{\mathsf{a}}\left(1 + \ln\left(\frac{|P|T}{\bar{\epsilon}}\right)\right) \tag{28}$$

$$\leq \frac{\mathsf{OPT}}{\mathsf{a}}\left(1 + \ln\left(\frac{n^h T}{\bar{\epsilon}}\right)\right) \tag{29}$$

$$\leq \frac{\mathsf{OPT}}{\mathsf{a}}\left(1 + h\ln(n) + \ln T + \ln(\frac{1}{\bar{\epsilon}})\right) \tag{30}$$

which completes the proof. See Remarks 1 and 2 for how our ratio generalizes to both linear and non-linear cases, and how the performance behaves when model $\mathfrak{J}_{\boldsymbol{\theta}}$ accurately estimates the exact value.

Remark 1. *(Approximation ratio when f_i is integer-valued). In Theorem 1, if f_i is integer-valued for $i \in [m]$ and we set $\bar{\epsilon} \in (0, 1)$, then running the Predictive Path Stressing algorithm on $\mathcal{E}$ yields a feasible solution $\boldsymbol{x}$, i.e, $\mathcal{C}(P, \boldsymbol{x}) = |P|T$ such that $\mathbb{E}[\|\boldsymbol{x}\|_1] \leq \frac{(1+h\ln(n)+\ln T)}{\mathsf{a}}\mathsf{OPT}$*

Remark 2. *If PPS selectes all edges in the optimal solution, i.e, $\Pr[\mathcal{E}^* \subseteq \mathcal{E}] = 1$, then running the Predictive Path Stressing algorithm on $\mathcal{E}$ yields a solution $\boldsymbol{x}$ such that $\mathcal{C}(P, \boldsymbol{x}) \geq |P|T - \bar{\epsilon}$ and $\mathbb{E}[\|\boldsymbol{x}\|_1] \leq (1 + h\ln(n) + \ln T + \ln(1/\bar{\epsilon}))\mathsf{OPT}$*

9.2 Expert Augmentation Efficiency

Theorem 2 (Expert Augmentation Efficiency): Suppose there exists a constant $\epsilon > 0$ such that $\mathbb{P}_{p(x)}\{\chi(x) > \delta\} \geq \epsilon$; on the region $\{x : \chi(x) > \delta\}$, a new expert $\Omega_{N+1}(x)$ is added with gating weight $\alpha(x) = \mathbf{1}\{\chi(x) > \delta\}a_0$ $(0 < a_0 < 1)$ and trained to satisfy $\Omega_{N+1}(x) \geq c\, q(x)$ for some $c \in (0, 1)$; then the updated mixture $\Omega'(x) = \alpha(x)\,\Omega_{N+1}(x) + (1 - \alpha(x))\,\Omega(x)$ satisfies

$\mathrm{KL}(q\|\Omega') \leq \mathrm{KL}(q\|\Omega) - \gamma(\delta, \epsilon)$, where $\gamma(\delta, \epsilon) = a_0(\delta + \log c)\,\epsilon_0 > 0$ and $\epsilon_0 > 0$ is a lower bound on $\int_{\chi(x)>\delta} q(x)dx$.

Proof. In the regions where $\chi(x) = \log \frac{q(x)}{\Omega(x)}$ and $\chi(x) > \delta$, we add a new expert $\Omega_{N+1}(x)$ and extend the gating network so that its output becomes:

$$w_i'(x), \quad i = 0, 1, \ldots, N+1, \quad \text{with } \sum_{i=0}^{N+1} w_i'(x) = 1. \tag{31}$$

Without loss of generality and for analysis purposes, we form a new mixture by taking a convex combination of the old mixture and the new expert:

$$\Omega'(x) = \alpha(x)\,\Omega_{N+1}(x) + \big(1 - \alpha(x)\big)\,\Omega(x). \tag{32}$$

We choose the gating function $\alpha(x)$ to focus on regions where $\chi(x) > \delta$. In particular, we set:

$$\alpha(x) = \mathbf{1}\{\chi(x) > \delta\}\,a(x), \tag{33}$$

and for simplicity we take $a(x) = a_0$ for those x with $\chi(x) > \delta$ where $0 < a_0 < 1$. Next, we consider the KL divergence between $q(x)$ and the original mixture $\Omega(x)$,

$$\mathrm{KL}(q\|\Omega) = \int q(x) \log \frac{q(x)}{\Omega(x)}\,dx, \tag{34}$$

and the KL divergence between $q(x)$ and the updated mixture $\Omega'(x)$,

$$\begin{aligned}
\mathrm{KL}(q\|\Omega') &= \int q(x) \log \frac{q(x)}{\Omega'(x)}\,dx \\
&= \int q(x) \log \frac{q(x)}{\alpha(x)\,\Omega_{N+1}(x) + (1 - \alpha(x))\,\Omega(x)}\,dx \\
&= \int q(x) \log q(x) - q(x) \log\Big[\alpha(x)\,\Omega_{N+1}(x) + (1 - \alpha(x))\,\Omega(x)\Big]\,dx \\
&\leq \int q(x)\Big[\log q(x) - \alpha(x)\,\log \Omega_{N+1}(x) - (1 - \alpha(x))\,\log \Omega(x)\Big]dx \\
&= \int q(x) \log q(x)\,dx - \int q(x)\,\alpha(x) \log \Omega_{N+1}(x)\,dx - \int q(x)(1 - \alpha(x)) \log \Omega(x)\,dx.
\end{aligned} \tag{35}$$

We can achieve the above because the logarithm is concave (and thus $-\log$ is convex), Jensen's inequality implies that:

$$\begin{aligned}
\log \Omega'(x) &= \log\Big(\alpha(x)\,\Omega_{N+1}(x) + \big(1 - \alpha(x)\big)\,\Omega(x)\Big) \\
&\geq \alpha(x) \log \Omega_{N+1}(x) + \big(1 - \alpha(x)\big) \log \Omega(x)
\end{aligned} \tag{36}$$

$$\begin{aligned}
&\implies -\log \Omega'(x) \leq -\alpha(x) \log \Omega_{N+1}(x) - (1 - \alpha(x)) \log \Omega(x). \\
&\implies -\int q(x) \log \Omega'(x)\,dx \leq -\int q(x)\,\alpha(x) \log \Omega_{N+1}(x)\,dx - \int q(x)(1 - \alpha(x)) \log \Omega(x)\,dx.
\end{aligned} \tag{37}$$

Recall that the KL divergence between q and Ω is defined as:

$$\begin{aligned}\mathrm{KL}(q\|\Omega) &= \int q(x)\log\frac{q(x)}{\Omega(x)}\,dx \\ &= \int q(x)\log q(x)\,dx - \int q(x)\log\Omega(x)\,dx.\end{aligned} \tag{38}$$

Similarly, for the updated mixture $\Omega'(x)$, we have the following.

$$\begin{aligned}\mathrm{KL}(q\|\Omega') &= \int q(x)\log\frac{q(x)}{\Omega'(x)}\,dx \\ &= \int q(x)\log q(x)\,dx - \int q(x)\log\Omega'(x)\,dx.\end{aligned} \tag{39}$$

If we subtract the original divergence from the new one, the terms involving $\int q(x)\log q(x)\,dx$ cancel, leaving:

$$\begin{aligned}\mathrm{KL}(q\|\Omega') - \mathrm{KL}(q\|\Omega) &= \left[\int q(x)\log q(x)\,dx - \int q(x)\log\Omega'(x)\,dx\right] \\ &\quad - \left[\int q(x)\log q(x)\,dx - \int q(x)\log\Omega(x)\,dx\right] \\ &= -\int q(x)\log\Omega'(x)\,dx + \int q(x)\log\Omega(x)\,dx.\end{aligned} \tag{40}$$

$$\begin{aligned}\mathrm{KL}(q\|\Omega') - \mathrm{KL}(q\|\Omega) &\le \left[-\int q(x)\,\alpha(x)\log\Omega_{N+1}(x)\,dx - \int q(x)(1-\alpha(x))\log\Omega(x)\,dx\right] \\ &\quad + \int q(x)\log\Omega(x)\,dx \\ &= -\int q(x)\,\alpha(x)\log\Omega_{N+1}(x)\,dx - \int q(x)(1-\alpha(x))\log\Omega(x)\,dx \\ &\quad + \int q(x)\log\Omega(x)\,dx \\ &= -\int q(x)\,\alpha(x)\Big[\log\Omega_{N+1}(x) - \log\Omega(x)\Big]dx.\end{aligned} \tag{41}$$

This completes the derivation. We have shown that the difference in the KL divergence between q and the updated mixture Ω' and that between q and the original mixture Ω is bounded above by:

$$\mathrm{KL}(q\|\Omega') - \mathrm{KL}(q\|\Omega) \le -\int q(x)\,\alpha(x)\left[\log\Omega_{N+1}(x) - \log\Omega(x)\right]dx. \tag{42}$$

Notice that on the region where $\chi(x) \le \delta$, the indicator in $\alpha(x)$ is zero, so we only integrate over the region where $\chi(x) > \delta$. Setting $\alpha(x) = a_0$ on that region, we obtain:

$$\mathrm{KL}(q\|\Omega') - \mathrm{KL}(q\|\Omega) \le -a_0\int_{\chi(x)>\delta} q(x)\log\frac{\Omega_{N+1}(x)}{\Omega(x)}\,dx. \tag{43}$$

Now, for any x with $\chi(x) > \delta$, we have $\log\frac{q(x)}{\Omega(x)} > \delta$ so that $\frac{q(x)}{\Omega(x)} > \exp(\delta)$. Suppose further that the new expert is designed such that, on this region, $\Omega_{N+1}(x) \ge c\,q(x)$, where $c \in (0,1)$ is a constant. Then it holds that:

$$\frac{\Omega_{N+1}(x)}{\Omega(x)} \ge c\,\frac{q(x)}{\Omega(x)} > c\,\exp(\delta). \tag{44}$$

Taking logarithms gives:

$$\log \frac{\Omega_{N+1}(x)}{\Omega(x)} \geq \delta + \log c. \tag{45}$$

Thus, if we let:

$$\eta = \int_{\chi(x)>\delta} q(x)\, dx, \tag{46}$$

and assume (through technical equivalence between $q(x)$ and $p(x)$) that $\eta \geq \epsilon_0 > 0$, then we obtain:

$$\int_{\chi(x)>\delta} q(x) \log \frac{\Omega_{N+1}(x)}{\Omega(x)}\, dx \geq (\delta + \log c)\, \epsilon_0.$$

Therefore, the reduction in KL divergence satisfies:

$$\mathrm{KL}(q\|\Omega') - \mathrm{KL}(q\|\Omega) \leq -a_0\, (\delta + \log c)\, \epsilon_0.$$

Defining:

$$\gamma(\delta, \epsilon) = a_0\, (\delta + \log c)\, \epsilon_0,$$

We conclude that:

$$\mathrm{KL}(q\|\Omega') \leq \mathrm{KL}(q\|\Omega) - \gamma(\delta, \epsilon),$$

which shows that the addition of the new expert decreases the KL divergence by at least $\gamma(\delta, \epsilon) > 0$. Thus, this completes our proof.

Corollary 3 (Effectiveness of Expert Addition and Gating Retraining). Suppose that at each iteration, a new expert is added in regions where the discrepancy $\chi(x) > \delta$, and the gating network is retrained using a softmax function that produces a convex combination of expert outputs. The updated mixture then takes the form $\Omega^{(t+1)}(x) = \sum_{i=0}^{N+1} w_i'(x)\Omega_i(x)$, where the new gating weights $w_i'(x)$ are derived from a softmax layer.

We have that the loss function for the gating network, defined as the KL divergence

$$\mathcal{L}_g\left(\varsigma_g\right) = \int q(x) \log \frac{q(x)}{\sum_{i=0}^{N+1} w_i'\left(x; \varsigma_g\right) \Omega_i(x)} dx$$

is L_g-smooth and μ_g-strongly convex in a neighborhood of the optimal parameter ς_g^*. Then, if we update the gating parameters via gradient descent with a learning rate small enough to ensure convergence, the KL divergence between the target distribution $q(x)$ and the current mixture $\Omega^{(t)}(x)$ decreases by at least a fixed amount $\gamma > 0$ at each iteration. That is,

$$\mathrm{KL}\left(q\|\Omega^{(t+1)}\right) \leq \mathrm{KL}\left(q\|\Omega^{(t)}\right) - \gamma$$

As a result, the sequence of mixtures $\left\{\Omega^{(t)}(x)\right\}$ converges to $q(x)$, in the sense that

$$\lim_{t\to\infty} \mathrm{KL}\left(q\|\Omega^{(t)}\right) = 0, \quad \text{so} \quad \Omega^{(t)}(x) \to q(x) \text{ almost everywhere.}$$

Finally, if $q(x)$ is a good approximation of the true target distribution $p(x)$-for example, because it comes from a properly trained energy-based model-then the mixture also converges to $p(x)$. In the limit, we obtain the desired equilibrium where

$$\Omega(x) = q(x) = p(x),$$

showing that the mixture model has successfully learned the target distribution through iterative refinement.

Proof: After the new expert is added, the gating network is retrained using a softmax layer that produces new weights

$$w_i'(x) = \frac{\exp(f_i'(x))}{\sum_{j=0}^{N+1} \exp(f_j'(x))}, \quad i = 0, \ldots, N+1,$$

so that the updated mixture becomes

$$\Omega'(x) = \sum_{i=0}^{N+1} w_i'(x)\, \Omega_i(x).$$

We define the gating network's loss as the KL divergence

$$\mathcal{L}_g(\varsigma_g) = \int q(x) \log \frac{q(x)}{\sum_{i=0}^{N+1} w_i'(x;\varsigma_g)\, \Omega_i(x)}\, dx.$$

Because softmax produces a convex combination, the mapping $x \mapsto \sum_{i=0}^{N+1} w_i'(x;\varsigma_g)\, \Omega_i(x)$ is convex in the gating weights. Under the assumptions that $\mathcal{L}_g$ is L_g-smooth and μ_g-strongly convex near the optimum ς_g^*, we update the parameters by gradient descent:

$$\varsigma_g^{(t+1)} = \varsigma_g^{(t)} - \eta_g\, \nabla_{\varsigma_g} \mathcal{L}_g(\varsigma_g^{(t)}).$$

By standard results, the error in the gating network decreases as

$$\|\varsigma_g^{(t+1)} - \varsigma_g^*\|^2 \leq \left(1 - \eta_g(2\mu_g - \eta_g L_g^2)\right) \|\varsigma_g^{(t)} - \varsigma_g^*\|^2,$$

ensuring convergence provided the learning rate η_g is chosen sufficiently small. As the gating network converges, the new mixture $\Omega'(x) = \sum_{i=0}^{N+1} w_i'(x;\varsigma_g)\, \Omega_i(x)$ becomes an even better approximation of $q(x)$, reinforcing the reduction in KL divergence we established earlier. If this process of detecting regions where $\chi(x) > \delta$, adding a new expert, and retraining the gating network is repeated iteratively, we obtain a sequence of mixtures $\{\Omega^{(t)}(x)\}$ for which

$$\mathrm{KL}(q\|\Omega^{(t+1)}) \leq \mathrm{KL}(q\|\Omega^{(t)}) - \gamma(\delta, \epsilon).$$

Thus, by repeated application, we ensure

$$\lim_{t\to\infty} \mathrm{KL}(q\|\Omega^{(t)}) = 0.$$

Assuming that $q(x)$ is a good proxy for $p(x)$ (due to proper training of the EBM), it follows that

$$\Omega^{(t)}(x) \to q(x) \quad \text{and ultimately } \Omega^{(t)}(x) \to p(x),$$

establishing the desired Nash equilibrium $p(x) = q(x) = \Omega(x)$.

9.3 Normalization Free Function

Theorem 3 (Normalization Free Function): The objective function $\min_q \max_\Omega \{\mathrm{KL}(p\|q) - \mathrm{KL}(\Omega\|q)\}$, is normalizing free which is independent with Z.

Proof. By definition, the KL divergence between $p(x)$ and $q(x)$ is $\mathrm{KL}(p\|q) = \int p(x) \log \frac{p(x)}{q(x)}\, dx$. Since $q(x) = \frac{\exp\left(-\frac{E(\mathbf{x})}{\tau}\right)}{Z}$, we can rewrite it as:

$$\begin{aligned}
\mathrm{KL}(p\|q) &= \int p(x) \log \frac{p(x)}{q(x)}\, dx \\
&= \int p(x) \Big(\log p(x) - \log q(x) \Big)\, dx \\
&= \int p(x) \log p(x)\, dx - \int p(x) \log q(x)\, dx \\
&= \int p(x) \log p(x)\, dx - \int p(x) \left[\log \left(\frac{\exp\left(-\frac{E(\mathbf{x})}{\tau}\right)}{Z} \right) \right] dx \\
&= \int p(x) \log p(x)\, dx - \int p(x) \left[-\frac{E(x)}{\tau} - \log Z \right] dx \\
&= \int p(x) \log p(x)\, dx + \frac{1}{\tau} \int p(x) E(x)\, dx + \log Z \int p(x)\, dx \\
&= \int p(x) \log p(x)\, dx + \frac{1}{\tau} \int p(x) E(x)\, dx + \log Z,
\end{aligned}$$

Similarly, the KL divergence between $\Omega(x)$ and $q(x)$ is:

$$\begin{aligned}
\mathrm{KL}(\Omega\|q) &= \int \Omega(x) \log \frac{\Omega(x)}{q(x)}\, dx \\
&= \int \Omega(x) \log \Omega(x)\, dx - \int \Omega(x) \log q(x)\, dx \\
&= \int \Omega(x) \log \Omega(x)\, dx - \int \Omega(x) \left[-\frac{E(x)}{\tau} - \log Z \right] dx \\
&= \int \Omega(x) \log \Omega(x)\, dx + \frac{1}{\tau} \int \Omega(x) E(x)\, dx + \log Z \int \Omega(x)\, dx \\
&, = \int \Omega(x) \log \Omega(x)\, dx + \frac{1}{\tau} \int \Omega(x) E(x)\, dx + \log Z
\end{aligned}$$

The difference between the two KL divergences is then:

$$\begin{aligned}
\mathrm{KL}(p\|q) - \mathrm{KL}(\Omega\|q) &= \left[\int p(x) \log p(x)\, dx + \frac{1}{\tau} \int p(x) E(x)\, dx + \log Z \right] \\
&\quad - \left[\int \Omega(x) \log \Omega(x)\, dx + \frac{1}{\tau} \int \Omega(x) E(x)\, dx + \log Z \right] \\
&= \int p(x) \log p(x)\, dx - \int \Omega(x) \log \Omega(x)\, dx \\
&\quad + \frac{1}{\tau} \int p(x) E(x) - \frac{1}{\tau} \int \Omega(x) E(x)\, dx. \\
&= \underbrace{\int p(x) \log p(x)\, dx - \int \Omega(x) \log \Omega(x)\, dx}_{\textit{Entropy Different}} \\
&\quad + \frac{1}{\tau} \Big\{ \mathbb{E}_p[E_\theta(x)] - \mathbb{E}_\Omega[E_\theta(x)] \Big\}
\end{aligned}$$

Notice that the $\log Z$ terms cancel, so the difference is independent of Z.

Corollary 4 (Reduction to Energy Expectation Difference) Under the setting of Theorem 3, the minimax objective

$$\min_\theta \max_q \Big\{ \mathrm{KL}\big(p\|q\big) \;-\; \mathrm{KL}\big(\Omega\|q\big) \Big\}$$

is equivalent (up to additive and multiplicative constants independent of θ) to

$$\min_{\theta} \Big[\mathbb{E}_{x\sim p}\big[E_\theta(x)\big] \;-\; \mathbb{E}_{x\sim \Omega}\big[E_\theta(x)\big]\Big].$$

Proof. By Theorem 3 we have:

$$\begin{aligned}
\mathrm{KL}(p\|q) - \mathrm{KL}(\Omega\|q) &= \int p(x)\log p(x)\,dx - \int p(x)\log q(x)\,dx - \Big(\int \Omega(x)\log\Omega(x)\,dx && (47)\\
&\quad - \int \Omega(x)\log q(x)\,dx\Big) && (48)\\
&= \int p(x)\log p(x)\,dx - \int \Omega(x)\log\Omega(x)\,dx + \int \big[\Omega(x)-p(x)\big]\log q(x)\,dx && (49)\\
&= \underbrace{\int p(x)\log p(x)\,dx - \int \Omega(x)\log\Omega(x)\,dx}_{C} && (50)\\
&\quad + \int \big[p(x)-\Omega(x)\big]\Big(-\frac{1}{\tau}E_\theta(x) - \log Z\Big)\,dx && (51)\\
&= C + \frac{1}{\tau}\int \big[p(x)-\Omega(x)\big]\,E_\theta(x)\,dx \quad \Big(\text{since } \int \big[p(x)-\Omega(x)\big]\,dx = 0\Big) && (52)\\
&= C + \frac{1}{\tau}\Big(\mathbb{E}_{x\sim p}\big[E_\theta(x)\big] - \mathbb{E}_{x\sim\Omega}\big[E_\theta(x)\big]\Big). && (53)
\end{aligned}$$

Since adding the constant C and scaling by $1/\tau$ do not affect the location of the minimizer in θ, it follows that

$$\arg\min_{\theta}\Big\{\mathrm{KL}(p\|q) - \mathrm{KL}(\Omega\|q)\Big\} \;=\; \arg\min_{\theta}\Big\{\mathbb{E}_p[E_\theta(x)] - \mathbb{E}_\Omega[E_\theta(x)]\Big\},$$

which establishes the claimed reduction.

9.4 Differentiable Reward Function

Lemma 1 Let $\mathfrak{F}_\theta : \mathcal{X} \to \mathbb{R}$ be differentiable on an open set $\mathcal{X} \subset \mathbb{R}^m$. The reward function $\mathcal{R}(\mathbf{x})$ is differentiable on $\mathcal{X}$.

Proof. Let $\mathfrak{F}_{\boldsymbol{\theta}}(G, s, t; \mathbf{x})$ be a neural estimator of shortest-path cost from source node s to target node t, parameterized by $\boldsymbol{\theta}$. Define the reward function as:

$$\mathcal{R}(\mathbf{x}) = \underbrace{\sum_{(s,t)\in\mathcal{K}} \pounds\left(\mathfrak{F}_{\boldsymbol{\theta}}(G,s,t;\mathbf{x}) - T\right)}_{\text{Smooth feasibility Term}} - \varkappa \cdot \underbrace{\log\left(1 + \|\Upsilon(\mathbf{x})\|_1\right)}_{\text{Soft Cost Penalty Term}}$$

where $\pounds(z) = \frac{1}{1+e^{-\zeta z}}$ is a sigmoid function with slope parameter $\zeta > 0$, $\Upsilon(\mathbf{x}) = \log(1 + e^{\mathbf{x}})$, applied coordinate-wise, and $T > 0$ is the feasibility threshold, and $\kappa > 0$ is a regularization hyperparameter. We aim to show that $\mathcal{R}(\mathbf{x})$ is continuously differentiable on any open domain $\mathcal{X} \subset \mathbb{R}^m$. First, the estimator $\mathfrak{F}_{\boldsymbol{\theta}}$ is a deep graph neural network constructed as a composition of L differentiable layers:

$$\mathfrak{F}_{\boldsymbol{\theta}} = f_L \circ f_{L-1} \circ \cdots \circ f_1$$

where each f_i may involve affine maps, ELU activations, attention mechanisms, and message-passing steps, all of which are differentiable. Hence, by the chain rule, $\mathfrak{F}_{\boldsymbol{\theta}}(G, s, t; \cdot)$ is differentiable on $\mathcal{X}$. For each $(s, t) \in \mathcal{K}$, define:

$$\pounds_{s,t}(\mathbf{x}) := \pounds\left(\mathfrak{F}_\theta(G, s, t; \mathbf{x}) - T\right).$$

Since $\mathfrak{F}_{\boldsymbol{\theta}}$ is differentiable and $\pounds(z)$ is smooth (C^∞) on $\mathbb{R}$ with $\pounds'(z) = \zeta \cdot \pounds(z) \cdot (1 - \pounds(z))$, it follows from the chain rule that $\pounds_{s,t}(\mathbf{x})$ is differentiable:

$$\nabla \pounds_{s,t}(\mathbf{x}) = \pounds'\left(\mathfrak{F}_{\boldsymbol{\theta}}(G, s, t; \mathbf{x}) - T\right) \cdot \nabla \mathfrak{F}_{\boldsymbol{\theta}}(G, s, t; \mathbf{x}).$$

Summing over all (s, t) in $\mathcal{K}$, we obtain:

$$F(\mathbf{x}) := \sum_{(s,t)\in\mathcal{K}} \pounds_{s,t}(\mathbf{x})$$

which is a finite sum of differentiable functions and therefore differentiable on X. Let $\Upsilon : \mathbb{R}^m \to \mathbb{R}^m$ be the vector-valued softplus function such as:

$$\Upsilon(\mathbf{x}) := \left[\log\left(1 + e^{x_1}\right), \ldots, \log\left(1 + e^{x_m}\right)\right]$$

where each component $\Upsilon_i\left(x_i\right)$ is differentiable, with:

$$\Upsilon_i'\left(x_i\right) = \frac{e^{x_i}}{1 + e^{x_i}} = \pounds\left(x_i\right)$$

Hence $\Upsilon \in C^\infty\left(\mathbb{R}^m\right)$, and so the map $\mathbf{x} \mapsto \|\Upsilon(\mathbf{x})\|_1 = \sum_{i=1}^m \Upsilon_i\left(x_i\right)$ is also differentiable as a finite sum of differentiable functions. Now we consider the scalar penalty $\Lambda(\mathbf{x}) := \log\left(1 + \|\Upsilon(\mathbf{x})\|_1\right)$. Because $\Upsilon_i\left(x_i\right) > 0$ for all x_i, we have $\|\Upsilon(\mathbf{x})\|_1 > 0$, and $\log(1 + u)$ is smooth on $(0, \infty)$. Hence, by the chain rule:

$$\nabla\Lambda(\mathbf{x}) = \frac{1}{1 + \|\Upsilon(\mathbf{x})\|_1} \cdot \nabla\left(\sum_{i=1}^m \Upsilon_i\left(x_i\right)\right)$$

which is well-defined and continuous on $\mathbb{R}^m$. Putting both together that $F(\mathbf{x}) = \sum_{(s,t)\in\mathcal{K}} \pounds_{s,t}(\mathbf{x})$ and $\Lambda(\mathbf{x}) = \log\left(1 + \|\Upsilon(\mathbf{x})\|_1\right)$ is both differentiable. Therefore, the reward function $\mathcal{R}(\mathbf{x}) = F(\mathbf{x}) - \varkappa \cdot \Lambda(\mathbf{x})$ is differentiable on any open set $\mathcal{X} \subset \mathbb{R}^m$, i.e., $\mathcal{R} \in C^1(X)$.

This concludes the proof.

9.5 Reward Estimation Consistency

Theorem 4 (Reward Estimation Consistency) Assume that Ω has converged properly, for any perturbed latent vector $\hat{z}_i := z_i + \hat{\epsilon} \cdot \nabla_{z_i}\mathcal{R}(\mathcal{M}_\phi(z_i, \mathbf{c}))$ with small $\hat{\epsilon} > 0$, we have $\mathcal{R}\left(\hat{\mathbf{x}}_i\right) > \mathcal{R}\left(\mathbf{x}_i\right)$, where $\hat{\mathbf{x}}_i = \mathcal{M}_\phi(\hat{z}_i, \mathbf{c})$.

Proof. To prove this theorem, we first prove that for a well-converged Ω_Θ, $\mathcal{M}_\phi$ is Lipschitz-continuous.

Consider the decoder $\mathcal{M}_\phi : \mathbb{R}^d \to \mathbb{R}^P$ composed of N layers. For $j = 1$ to $N - 1$, each layer computes:

$$h_j = \mathfrak{q}_j(h_{j-1}) = \text{ReLU}(W_j h_{j-1} + b_j),$$

where $h_0 = z_i \in \mathbb{R}^d$, $W_j \in \mathbb{R}^{d_j \times d_{j-1}}$, and $b_j \in \mathbb{R}^{d_j}$. The output layer computes:

$$\mathbf{x}_i = \mathcal{M}_\phi(z_i) = \mathfrak{q}_N(h_{N-1}) = W_N h_{N-1} + b_N,$$

with $W_N \in \mathbb{R}^{P \times d_{N-1}}$ and $b_N \in \mathbb{R}^P$.

To prove that $\mathcal{M}_\phi$ is Lipschitz continuous, consider two inputs $z_i, \hat{z}_i \in \mathbb{R}^d$. We aim to show:

$$\|\mathcal{M}_\phi(z_i) - \mathcal{M}_\phi(\hat{z}_i)\| \leq K\|z_i - \hat{z}_i\|,$$

where K is a finite constant.

Starting from the output layer:

$$\begin{aligned}\|\mathcal{M}_\phi(z_i) - \mathcal{M}_\phi(\hat{z}_i)\| &= \|\mathfrak{q}_N(h_{N-1}^{(z_i)}) - \mathfrak{q}_N(h_{N-1}^{(\hat{z}_i)})\| \\ &= \|W_N h_{N-1}^{(z_i)} + b_N - W_N h_{N-1}^{(\hat{z}_i)} - b_N\| \\ &= \|W_N(h_{N-1}^{(z_i)} - h_{N-1}^{(\hat{z}_i)})\| \\ &\leq \|W_N\|_2 \|h_{N-1}^{(z_i)} - h_{N-1}^{(\hat{z}_i)}\|,\end{aligned}$$

where $\|W_N\|_2$ denotes the spectral norm of W_N.

For each hidden layer $j = N-1$ down to 1:

$$\begin{aligned}\|h_j^{(z_i)} - h_j^{(\hat{z}_i)}\| &= \|\mathfrak{q}_j(h_{j-1}^{(z_i)}) - \mathfrak{q}_j(h_{j-1}^{(\hat{z}_i)})\| \\ &= \|\text{ReLU}(W_j h_{j-1}^{(z_i)} + b_j) - \text{ReLU}(W_j h_{j-1}^{(\hat{z}_i)} + b_j)\| \\ &\leq \|W_j h_{j-1}^{(z_i)} - W_j h_{j-1}^{(\hat{z}_i)}\| \quad \text{(since ReLU is 1-Lipschitz)} \\ &\leq \|W_j\|_2 \|h_{j-1}^{(z_i)} - h_{j-1}^{(\hat{z}_i)}\|.\end{aligned}$$

By recursively applying these inequalities, we obtain:

$$\|h_j^{(z_i)} - h_j^{(\hat{z}_i)}\| \leq \left(\prod_{k=1}^{j} \|W_{N-k+1}\|_2\right) \|h_0^{(z_i)} - h_0^{(\hat{z}_i)}\| = \left(\prod_{k=1}^{j} \|W_{N-k+1}\|_2\right) \|z_i - \hat{z}_i\|.$$

At the output layer:

$$\|\mathcal{M}_\phi(z_i) - \mathcal{M}_\phi(\hat{z}_i)\| \leq \|W_N\|_2 \|h_{N-1}^{(z_i)} - h_{N-1}^{(\hat{z}_i)}\|.$$

Substituting the recursive bound:

$$\|\mathcal{M}_\phi(z_i) - \mathcal{M}_\phi(\hat{z}_i)\| \leq \left(\prod_{j=1}^{N} \|W_j\|_2\right) \|z_i - \hat{z}_i\|.$$

Define $K = \prod_{j=1}^{N} \|W_j\|_2$. To ensure K is finite, we enforce bounds (Layer Normalization) on the spectral norms: $\|W_j\|_2 \leq s_j$, where s_j are finite constants. Then:

$$K \leq \prod_{j=1}^{N} s_j.$$

If we choose $s_j = s \leq 1$ for all j, then $K \leq s^N \leq 1$, which is finite. Therefore, $\mathcal{M}_\phi$ is Lipschitz continuous with Lipschitz constant K, satisfying:

$$\|\mathcal{M}_\phi(z_i) - \mathcal{M}_\phi(\hat{z}_i)\| \leq K\|z_i - \hat{z}_i\|.$$

Thus, we finished proving that $\mathcal{M}_\phi$ is Lipschitz-continuous. Given Ω_Θ is well converged, with $\mathcal{M}_\phi$ being Lipschitz continuous and differentiable, a small learning rate $\hat{\epsilon}$ induces a small change in latent vector z_i which results in a small change in the data point $\mathbf{x}_i$ reconstructed by C-VAE. We can use the first-order Taylor expansion for small $\Delta z_i = \hat{z}_i - z_i$:

$$\hat{\mathbf{x}}_i = \mathcal{M}_\phi(\hat{z}_i) \approx \mathcal{M}_\phi(z_i) + J_{\mathcal{M}_\phi}(z_i) \cdot \Delta z_i$$

where $J_{\mathcal{M}_\phi}(z_i)$ is the Jacobian matrix of $\mathcal{M}_\phi$ at z_i.

From the update rule:

$$\Delta z_i = \hat{z}_i - z_i = \hat{\epsilon} \cdot \nabla_{z_i} \mathcal{R}(\mathbf{x}_i)$$

Thus, the change in $\mathbf{x}_i$ is:

$$\hat{\mathbf{x}}_i - \mathbf{x}_i \approx J_{\mathcal{M}_\phi}(z_i) \cdot \Delta z_i = \hat{\epsilon} \cdot J_{\mathcal{M}_\phi}(z_i) \cdot \nabla_{z_i} \mathcal{R}(\mathbf{x}_i)$$

Since $\mathbf{x}_i = \mathcal{M}_\phi(z_i)$, by the chain rule, we have:

$$\nabla_{z_i} \mathcal{R}(\mathbf{x}_i) = J_{\mathcal{M}_\phi}^\top(z_i) \cdot \nabla_{\mathbf{x}_i} \mathcal{R}(\mathbf{x}_i)$$

Therefore:

$$\hat{\mathbf{x}}_i - \mathbf{x}_i \approx \hat{\epsilon} \cdot J_{\mathcal{M}_\phi}(z_i) \cdot J_{\mathcal{M}_\phi}^\top(z_i) \cdot \nabla_{\mathbf{x}_i} \mathcal{R}(\mathbf{x}_i)$$

Let $\mathfrak{Z} = J_{\mathcal{M}_\phi}(z_i) \cdot J_{\mathcal{M}_\phi}^\top(z_i)$, which is a positive semi-definite matrix. Thus:

$$\hat{\mathbf{x}}_i - \mathbf{x}_i \approx \hat{\epsilon} \cdot \mathfrak{Z} \cdot \nabla_{\mathbf{x}_i} \mathcal{R}(\mathbf{x}_i)$$

Using a first-order Taylor expansion of r around $\mathbf{x}_i$:

$$\Delta \mathcal{R} = \mathcal{R}(\hat{\mathbf{x}}_i) - \mathcal{R}(\mathbf{x}_i) \approx \nabla_{\mathbf{x}_i} \mathcal{R}(\mathbf{x}_i)^\top (\hat{\mathbf{x}}_i - \mathbf{x}_i)$$

Substituting $\hat{\mathbf{x}}_i - \mathbf{x}_i$, we obtain:

$$\Delta \mathcal{R} \approx \hat{\epsilon} \cdot \nabla_{\mathbf{x}_i} \mathcal{R}(\mathbf{x}_i)^\top \mathfrak{Z} \cdot \nabla_{\mathbf{x}_i} \mathcal{R}(\mathbf{x}_i)$$

Since $\mathfrak{Z}$ is positive semi-definite and $\hat{\epsilon} > 0$:

$$\Delta \mathcal{R} \geq 0$$

More specifically, $\Delta \mathcal{R} = 0$ if and only if $\nabla_{\mathbf{x}_i} \mathcal{R}(\mathbf{x}_i) = 0$. Otherwise, $\mathcal{R} > 0$. Therefore, under the given conditions and for a sufficiently small $\hat{\epsilon}$:

$$\mathcal{R}(\hat{\mathbf{x}}_i) > \mathcal{R}(\mathbf{x}_i)$$

This completes the proof.

Remark 3 (Trade-off between Lipschitz control and expressiveness)**.** *Imposing strict Lipschitz constraints on the decoder $\mathcal{M}_\phi$, for instance by enforcing $\|W_j\|_2 \leq s_j$ for all layers so that $\prod_{j=1}^N s_j \leq K$, guarantees global smoothness but inevitably limits the network's representational capacity. Such strong spectral normalization can suppress high-frequency components essential for accurate reconstruction, leading to degradation in performance. Hence, while a bounded $\mathcal{M}_\phi$ ensures stability, it often sacrifices fine-grained generative fidelity.*

Remark 4 (Local Lipschitz behavior on latent support)**.** *In practice, it is not necessary to enforce a global Lipschitz constraint across all decoder layers. The decoder $\mathcal{M}_\phi$ only needs to be* locally Lipschitz *over the high-density region of the latent prior $p(z) = \mathcal{N}(0, I)$. Formally, there exists a constant $L_{local} > 0$ such that for any z_1, z_2 within $\Omega = \{z \mid p(z) > \epsilon\}$,*

$$\|\mathcal{M}_\phi(z_1) - \mathcal{M}_\phi(z_2)\| \leq L_{local} \|z_1 - z_2\|.$$

This localized smoothness naturally emerges from the VAE objective, which regularizes $q_\psi(z|x)$ toward $p(z)$ and thereby aligns nearby z's with semantically close reconstructions.

9.6 Safe Ball In Latent Space

Assumption 1 (Edge–cost Lipschitz). Each edge function f_e is L_e–Lipschitz on $[0, b_e]$ and

$$L_f := \max_{e \in E} L_e. \tag{A1}$$

Assumption 2 (Estimator Uniform Error). The SPAGAN surrogate satisfies

$$0 \le \left| \mathfrak{J}_{\boldsymbol{\theta}}(G, s, t; \mathbf{x}) - \mathrm{SP}_G(s, t; \mathbf{x}) \right| \le \varepsilon_{\text{spa}}, \qquad \forall\, (s, t), \mathbf{x}. \tag{A2}$$

Proposition 2 (Safe Ball In Latent Space). Let $\mathbf{x}_\star = \mathcal{M}_\phi(z_\star, c)$ be feasible and denote the true safety margin by $\bar{\Delta} = \min_{(s,t)\in\mathcal{K}} \left[\mathrm{SP}_G(s, t; \mathbf{x}_\star) - T \right] > 0$. Assume a well trained SPAGAN's error is uniformly bounded by $\varepsilon_{\text{spa}} (0 \le \varepsilon_{\text{spa}} < \bar{\Delta})$. With $h = \lceil T / w_{\min} \rceil,\ L_f = \max_e L_e,\ m = |E|$, and decoder-Lipschitz constant $L_{\mathcal{M}}$, every latent vector z satisfying

$$\left\| z - z_\star \right\|_2 \le \frac{\bar{\Delta} - \varepsilon_{\text{spa}}}{h\, L_f\, \sqrt{m}\, L_{\mathcal{M}}}$$

is decoded to $\mathbf{x} = \mathcal{M}_\phi(z, c)$ for which $\mathfrak{J}_\theta(G, s, t; \mathbf{x}) \ge T - 2\epsilon_{spa}$ for all $(s, t) \in \mathcal{K}$; Thus, exploration within this latent ball guarantees provable safety and restores near full feasibility.

Proof. We begin by recalling Decoder $\mathcal{M}_\phi : \mathbb{R}^d \to \mathbb{R}^m$ (with $m = |E|$) is globally $L_{\mathcal{M}}$-Lipschitz

$$\|\mathcal{M}_\phi(z_1, c) - \mathcal{M}_\phi(z_2, c)\|_2 \le L_{\mathcal{M}} \|z_1 - z_2\|_2, \qquad \forall z_1, z_2 \in \mathbb{R}^d. \tag{54}$$

With $w_{\min} := \min_e w_e$, let $h := \lceil T / w_{\min} \rceil$ so every shortest path $\ddot{\rho}$ under threshold T satisfies $|\ddot{\rho}| \le h$. For any two perturbations $\mathbf{x}, \mathbf{x}'$ and any $(s, t) \in \mathcal{K}$, we have:

$$\begin{aligned} \left| \mathrm{SP}_G(s, t; \mathbf{x}) - \mathrm{SP}_G(s, t; \mathbf{x}') \right| &= \Big| \sum_{e \in \ddot{\rho}} f_e(\mathbf{x}_e) - \sum_{e \in \ddot{\rho}} f_e(\mathbf{x}'_e) \Big| \\ &\le \sum_{e \in \ddot{\rho}} L_e |\mathbf{x}_e - \mathbf{x}'_e| \\ &\le h\, L_f\, \|\mathbf{x} - \mathbf{x}'\|_1 \\ &\le h\, L_f\, \sqrt{m}\, \|\mathbf{x} - \mathbf{x}'\|_2. \end{aligned} \tag{55}$$

Switch to the estimator using assumption A2 and by applying the triangle inequality, which states $|a + b + c| \le |a| + |b| + |c|$, we have:

$$
\begin{aligned}
\left|\mathfrak{J}_{\boldsymbol{\theta}}(s,t;\mathbf{x}) - \mathfrak{J}_{\boldsymbol{\theta}}(s,t;\mathbf{x}_\star)\right| &= \Big|\underbrace{\mathfrak{J}_{\boldsymbol{\theta}}(s,t;\mathbf{x}) - \mathrm{SP}_G(s,t;\mathbf{x})}_{a} + \underbrace{\mathrm{SP}_G(s,t;\mathbf{x}) - \mathrm{SP}_G(s,t;\mathbf{x}_\star)}_{b} \\
&\qquad + \underbrace{\mathrm{SP}_G(s,t;\mathbf{x}_\star) - \mathfrak{J}_{\boldsymbol{\theta}}(s,t;\mathbf{x}_\star)}_{c}\Big| \\
&\leq \underbrace{\left|\mathfrak{J}_{\boldsymbol{\theta}}(s,t;\mathbf{x}) - \mathrm{SP}_G(s,t;\mathbf{x})\right|}_{\leq \varepsilon_{\text{spa}}} + \underbrace{\left|\mathrm{SP}_G(s,t;\mathbf{x}) - \mathrm{SP}_G(s,t;\mathbf{x}_\star)\right|}_{\text{true–cost gap}} + \underbrace{\left|\mathrm{SP}_G(s,t;\mathbf{x}_\star) - \mathfrak{J}_{\boldsymbol{\theta}}(s,t;\mathbf{x}_\star)\right|}_{\leq \varepsilon_{\text{spa}}} \\
&\leq \varepsilon_{\text{spa}} + \left|\mathrm{SP}_G(s,t;\mathbf{x}) - \mathrm{SP}_G(s,t;\mathbf{x}_\star)\right| + \varepsilon_{\text{spa}} \\
&= 2\varepsilon_{\text{spa}} + \left|\mathrm{SP}_G(s,t;\mathbf{x}) - \mathrm{SP}_G(s,t;\mathbf{x}_\star)\right| \\
&\leq 2\varepsilon_{\text{spa}} + h\,L_f\,\sqrt{m}\,\left\|\mathcal{M}_\phi(z,c) - \mathcal{M}_\phi(z_\star,c)\right\|_2 \\
&\leq 2\varepsilon_{\text{spa}} + h\,L_f\,\sqrt{m}\,L_{\mathcal{M}}\,\|z - z_\star\|_2 \\
\implies -\left|\mathfrak{J}_{\boldsymbol{\theta}}(s,t;\mathbf{x}) - \mathfrak{J}_{\boldsymbol{\theta}}(s,t;\mathbf{x}_\star)\right| &\geq -(2\varepsilon_{\text{spa}} + h\,L_f\,\sqrt{m}\,L_{\mathcal{M}}\,\|z - z_\star\|_2)
\end{aligned}
\tag{56}
$$

The initial solution $\mathbf{x}_*$ is feasible with a true safety margin $\bar{\Delta}$ that $SP_G\left(s,t;\mathbf{x}_*\right) \geq T + \bar{\Delta}$. Moreoever, from Assumption 2, we have

$$
\begin{aligned}
&\left|\mathfrak{F}_\theta\left(G,s,t;\mathbf{x}_*\right) - SP_G\left(s,t;\mathbf{x}_*\right)\right| \leq \epsilon_{spa} \\
&\iff -\epsilon_{spa} \leq \mathfrak{J}_\theta\left(G,s,t;\mathbf{x}_*\right) - SP_G\left(s,t;\mathbf{x}_*\right) \leq \epsilon_{spa}
\end{aligned}
\tag{57}
$$

From this compound inequality, we are interested in the left-hand side to find a lower bound for $\mathfrak{F}_\theta\left(G,s,t;\mathbf{x}_*\right)$:

$$
\begin{aligned}
&-\epsilon_{spa} \leq \mathfrak{J}_\theta\left(G,s,t;\mathbf{x}_*\right) - SP_G\left(s,t;\mathbf{x}_*\right) \\
&\iff \mathfrak{F}_\theta\left(G,s,t;\mathbf{x}_*\right) \geq SP_G\left(s,t;\mathbf{x}_*\right) - \epsilon_{spa} \\
&\iff \mathfrak{F}_\theta\left(G,s,t;\mathbf{x}_*\right) \geq T + \bar{\Delta} - \epsilon_{spa}
\end{aligned}
\tag{58}
$$

Since the proposition states the condition $\|z - z_\star\|_2 \leq \frac{\bar{\Delta} - \epsilon_{spa}}{hL_f\sqrt{m}L_{\mathcal{M}}}$ and we know that $h, L_- f, \sqrt{m}, L_{\mathcal{M}}$ are all positive, this implies $hL_f\sqrt{m}L_{\mathcal{M}}\,\|z - z_\star\|_2 \leq \bar{\Delta} - \epsilon_{spa}$, hence:

$$
\begin{aligned}
\mathfrak{J}_\theta(G,s,t;\mathbf{x}) &\geq \mathfrak{J}_\theta\left(G,s,t;\mathbf{x}_*\right) - |\mathfrak{J}_\theta(G,s,t;\mathbf{x}) - \mathfrak{J}_\theta\left(G,s,t;\mathbf{x}_*\right)| \\
&\quad \text{(Note: since } A \geq B - |A - B|) \\
&\geq (T + \bar{\Delta} - \epsilon_{spa}) - |\mathfrak{J}_\theta(G,s,t;\mathbf{x}) - \mathfrak{J}_\theta\left(G,s,t;\mathbf{x}_*\right)| \\
&\quad \text{(Note: Substituting the lower bound for } \mathfrak{J}_\theta(G,s,t;\mathbf{x}_*)) \\
&\geq (T + \bar{\Delta} - \epsilon_{spa}) - \left(2\varepsilon_{\text{spa}} + h\,L_f\,\sqrt{m}\,L_{\mathcal{M}}\,\|z - z_\star\|_2\right) \\
&\quad \text{(Note: Substituting the upper bound for } |\mathfrak{J}_\theta(\mathbf{x}) - \mathfrak{J}_\theta(\mathbf{x}_*)|) \\
&= T + \bar{\Delta} - \epsilon_{spa} - 2\varepsilon_{\text{spa}} - h\,L_f\,\sqrt{m}\,L_{\mathcal{M}}\,\|z - z_\star\|_2 \\
&= T + \bar{\Delta} - 3\epsilon_{spa} - hL_f\sqrt{m}L_{\mathcal{M}}\,\|z - z_*\|_2 \\
&\geq T + \bar{\Delta} - 3\epsilon_{spa} - (\overline{\Delta} - \epsilon_{spa}) \\
&\quad \text{(Note: Since } -hL_f\sqrt{m}L_{\mathcal{M}}\,\|z - z_*\|_2 \geq -(\overline{\Delta} - \epsilon_{spa})) \\
&= T + \bar{\Delta} - 3\epsilon_{spa} - \overline{\Delta} + \epsilon_{spa} \\
&= T - 2\epsilon_{spa}
\end{aligned}
\tag{59}
$$

This completes the proof.

10 DETAILS EXPERIMENTS AND ABLATION STUDIES

This section provides comprehensive details of our experimental setup, evaluation metrics, and ablation studies. We aim to assess both the effectiveness and generalizability of the proposed Hephaestus framework across a range of real-world network topologies and baselines. The experiments are designed to evaluate performance from multiple perspectives: solution feasibility, budget efficiency, scalability, and training dynamics. In addition, ablation studies are conducted to isolate the contributions of key architectural components—including the SPAGAN-based path estimator, mixture of generative experts, and reinforcement-based refinement—and to examine how each contributes to the overall performance. We also detail the hyperparameter choices, compute infrastructure, and exact solver setups used for benchmarking.

10.1 Dataset Details

Table2 provides detailed statistics of the real-world network datasets used in our experiments, each chosen to represent different structural and functional properties. The Email network is a directed communication graph among 1,005 individuals with 25,571 edges and a diameter of 7, capturing organizational email interactions characterized by strong community structures. The Gnutella dataset represents a decentralized peer-to-peer file-sharing system from August 2002. It contains 6,301 nodes and 20,777 directed connections, with a small diameter of 9 and low clustering, reflecting its unstructured topology. The RoadCA network is a large-scale undirected graph of California's road infrastructure with approximately 1.96 million nodes and 2.77 million edges, and an unusually large diameter of 849—typical of sparse, planar transportation networks. Finally, the Skitter dataset models the Internet at the autonomous system (AS) level using traceroute data, comprising 1.7 million nodes and 11.1 million directed edges with a diameter of 25. This dataset captures the hierarchical structure of inter-AS connectivity. Collectively, these datasets span a wide spectrum of scales, densities, and network types, forming a comprehensive benchmark for evaluating the scalability and generalizability of Hephaestus and its baselines.

Table 2: Statistics of real network datasets used in experiments.

Data	Type	Nodes	Edges	Diameter
Email	Directed	1,005	25,571	11
Gnutella	Directed	6,301	20,777	9
RoadCA	Undirected	1.96 M	2.77 M	849
Skitter	Directed	1.7 M	11.1 M	25

10.2 Hyperparameter Settings

In Table 3, we provide a detailed summary of the hyperparameters used across the three core phases of the `Hephaestus` framework: Forge, Morph, and Refine. These settings were chosen through extensive trial runs and empirical tuning to identify the best-performing configurations. The selected values aim to ensure training stability, strong generalization across diverse graph instances, and consistent performance throughout the pipeline.

In Forge, we train the SPAGAN model to estimate shortest-path costs efficiently. After trying several network configurations, we found that using 5 layers of Graph Attention Networks (GAT), each with 512 hidden units and 8 attention heads, gave reliable results on a wide range of graph structures. The ELU activation function was chosen because it helps avoid the dead neuron problem that can happen with ReLU, especially during early stages of training. For the learning rate, we tested multiple values and observed that 5×10^{-4} consistently led to stable convergence within 3000 epochs. Larger learning rates often made training unstable, while smaller ones slowed down progress too much. When choosing a loss function, we initially experimented with mean squared error (MSE), but observed that it was highly sensitive to a few long-path outliers, which harms the overall learning process. To address this, we adopted the Huber loss, which combines the benefits of MSE and MAE (Mean Absolute Error): it behaves quadratically for small errors to ensure smooth optimization and transitions to a only linear penalty for large errors. This property allowed us to reduce the influence of extreme outliers while still penalizing them, resulting in more stable convergence and improved

Table 3: Hyperparameter Settings.

Component/Phase	Parameter	Value
Phase 1: Forge		
SPAGAN ($\mathfrak{F}_{\boldsymbol{\theta}}$)	Network Architecture	5 Shortest Path GAT layers, 512 units, 8 heads
	Activation Function	ELU
	Learning Rate (α)	5×10^{-4}
	Optimizer	Adam
	Adam β_1, β_2	(0.9, 0.999)
	Batch Size	256
	Training Epochs	3000
	Loss Function	Huber Loss
	Max per-edge budget (b_e)	T
Phase 2: Morph		
EBM ($q(\mathbf{x})$)	Network Architecture	6 MLP layers, 512 units
	Activation Function	Swish
	Learning Rate (α_{EBM})	2×10^{-4}
	Optimizer	Adam
	Regularization ($\hat{\gamma}$)	1.0
	Batch Size (EBM Update)	256
	Training Iterations (Minimax)	50000
Mix-CVAE (Ω)	Latent Dimension of each CVAE (d)	128
	Encoder Architecture	GAT Encoder + MLP
	Decoder Architecture	MLP Decoder
	Activation Function	LeakyReLU
	Learning Rate (α_Ω)	8×10^{-4}
	Optimizer	Adam
	Batch Size (CVAE Update)	256
	Initial Experts (N_{init})	1
	Max Experts (N_{max})	9
	Expert Add Threshold (δ)	0.425
	KL Weight (β_{KL})	0.1
	Prior	$\mathcal{N}(0, I)$
Phase 3: Refine		
RL Agent (π)	Policy/Value Net Arch.	4 MLP layers, 256 units
	Activation Function	LeakyReLU
	Learning Rate (α_{RL})	1×10^{-4}
	Optimizer	Adam
	Discount Factor (γ_{RL})	0.99
	Reward Smoothness (ζ)	5.0
	Reward Cost Weight ($\varkappa$)	0.05
	Gradient Ascent Step ($\hat{\epsilon}_{GA}$)	2×10^{-3}
	RL Training Episodes	50000
	Top-K Solutions	10

predictive accuracy across diverse graph instances. The maximum per-edge budget $b_e = T$ defines the allowable range of perturbations and ensures that a feasible solution always exists for any input graph. Specifically, by assigning $x_e = T$ to every edge $e \in E$, the cost of any path becomes at least T, thus trivially satisfying the QoSD constraint. This choice guarantees feasibility without loss of generality, while still leaving many rooms for the model to explore more efficient and sparse perturbations to reduce total perturbation cost. Finally, we used a batch size of 256, which provided reliable gradients while fitting comfortably within GPU memory during training.

In Morph, the EBM is a 6-layer MLP with 512 units and Swish activation, which improves smoothness of the learned energy surface. The EBM is trained with a learning rate of 2×10^{-4}, using a minimax schedule for 50,000 iterations, along with regularization term $\hat{\gamma} = 1.0$ to prevent gradient explosion. For the Mix-CVAE, a latent dimension of 128 was sufficient to model the diverse structure of perturbations, while the encoder incorporates graph context via GAT layers. The KL weight $\beta_{KL} = 0.15$ balances reconstruction and latent regularization. We initialize with a single expert and allow expansion up to 9 experts, adding new CVAEs when the density ratio $\chi > 0.425$, indicating insufficient coverage by existing experts.

In Refine, we implement policy π as a 4-layer MLP with 256 units per layer and LeakyReLU activations which helps mitigate vanishing gradients. It commonly occur during late-stage training when the model begins to converge and reward differences become minimal. A conservative learning rate of 1×10^{-4} was selected to ensure policy stability, while a discount factor of 0.99 promotes long-term planning over greedy improvements. The reward shaping parameters $\zeta = 5.0$ and $\varkappa = 0.05$ control the smoothness of feasibility and cost feedback, respectively. We perturb the latent vector with a small gradient ascent step $\hat{\epsilon}_{GA} = 2 \times 10^{-3}$ to explore reward-improving directions, and retain the top-10 feasible solutions per episode to enrich the training buffer.

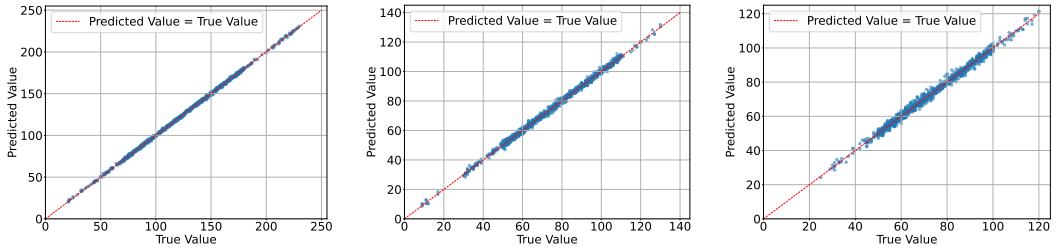
Hardware Specification. All experiments were conducted on a workstation equipped with an Intel Core i9-14900K CPU, 192 GB RAM, and 2× NVIDIA RTX 4090 GPUs (total 48 GB VRAM). While the GPUs played a critical role in training the SPAGAN, Mix-CVAE, and RL components efficiently, the large RAM and CPU core count were especially important for evaluating exact solvers like Gurobi and approximation algorithms. In particular, we relied on Gurobi to refine and benchmark outputs from baseline methods such as DIFFILO, Predict-and-Search, and L-MILPOPT, etc. These refinement steps often required solving large-scale ILPs with huge number of constraints and integer variables, where runtime and memory bottlenecks were significant. Thus, the high-performance CPU and 192 GB RAM were essential for verifying solution quality and feasibility in our exact evaluation pipeline.

Gurobi Heuristic Setup (Important). To assess the quality of learned solutions and evaluate optimality under exact conditions, we use Gurobi as a ground-truth solver. However, solving the full QoSD problem exactly is highly challenging at scale due to the exponential number of potential constraints. Recall $h = \left\lceil \frac{T}{w_{\min}} \right\rceil$ denotes the maximum number of edges in any feasible path, where T is the budget threshold and $w_{\min}$ is the minimum edge weight. Since each path can consist of at most h edges, and the graph contains n nodes, the total number of feasible paths is upper-bounded by n^h. This results in an exponentially large constraint space in the corresponding MILP formulation. This issue is especially severe when the threshold T is high, since a larger number of short paths remain feasible. In our experiments, we observed that Gurobi consistently fails to solve problem instances with more than 10,000 nodes at maximum density when all feasible paths are explicitly enumerated under medium T.

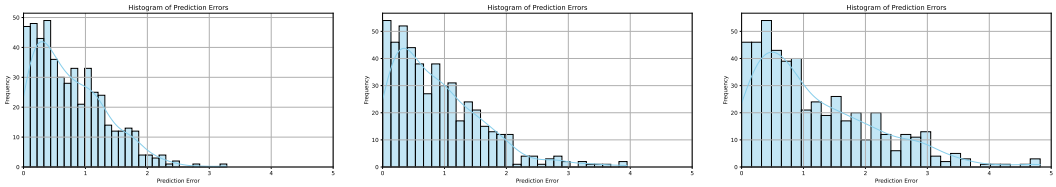
To address this scalability bottleneck, we adopt a heuristic strategy based on sampling for Gurobi refinement, which we refer to as Gurobi-Heuristic. Instead of enumerating all paths, we iteratively sample the current shortest paths between critical source-target pairs—using Dijkstra—and enforce path constraints only on these sampled paths. Gurobi then solves the reduced problem and updates the edge weights accordingly. This process is repeated: new shortest paths are sampled under the updated perturbation $\mathbf{x}$, and the solver is rerun. The iteration stops once no newly sampled path violates the threshold constraint T, thereby ensuring feasibility without needing to enumerate the full exponential set of feasible paths shorter than T. This strategy is crucial for making it feasible to refine the solutions produced by any ML-based method—including Hephaestus or baselines such as DIFFILO and Predict-and-Search—on large-scale graphs. Without it, exact refinement via Gurobi would be computationally infeasible due to the overwhelming number of constraints. To further

improve performance under such settings, Gurobi is configured with heuristic-oriented parameters including ‘Heuristics=0.5‘, ‘MIPFocus=1‘, and full multi-threading support.

10.3 SPAGAN Generalization

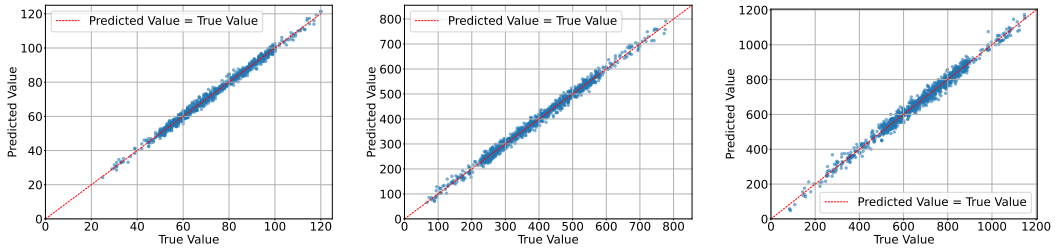


(a) Graph 1000 (b) Graph 2000 (c) Graph 3000

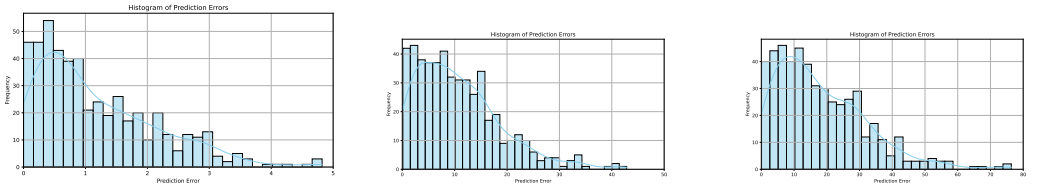


(d) Graph 1000 (e) Graph 2000 (f) Graph 3000

Figure 6: Predictive Accuracy Across Different Graph Sizes.



(a) Erdős-Rényi Model (b) Barabási-Albert Model (c) Watts-Strogatz Model



(d) Erdős-Rényi Model (e) Barabási-Albert Model (f) Watts-Strogatz Model

Figure 7: Predictive Accuracy Across Different Graph Topologies.

In our framework, SPAGAN model ($\mathfrak{F}_\theta$) plays a vital role in Forge, where it serves as a fast and accurate estimator of shortest-path costs. These estimates directly support the Predictive Path Stressing (PPS) algorithm in generating initial feasible perturbation vectors without requiring full shortest-path solver calls. Therefore, we want to evaluate the generalization capability of SPAGAN, by designing two sets of experiments using synthetic graphs. The first examines how well the model scales across different graph sizes, while the second investigates its performance across distinct graph structures.

To test scalability, we trained SPAGAN on Erdős–Rényi (ER) graphs with 1,000 nodes at maximum density, and then evaluated its predictions on ER graphs with larger sizes—2,000 and 3,000 nodes—also at maximum density. Figures 6 and 7 show the results. The scatter plots in Figure 6 (a–c) compare SPAGAN’s predictions with ground-truth shortest-path distances obtained via Dijkstra’s

algorithm. In all cases, the predicted values closely follow the diagonal (Predicted = True), indicating high prediction accuracy. The corresponding error histograms in Figure 7 (a–c) further support this, with prediction errors concentrated around zero and minimal high-magnitude deviations, suggesting both low bias and low variance.

In the second evaluation, to evaluate SPAGAN's ability to generalize across different graph topologies, we fixed the training on ER graphs with 1,000 nodes and tested the model on two structurally distinct types: Barabási–Albert (scale-free) and Watts–Strogatz (small-world), each with the same node count. As shown in Figures 6 and 7, SPAGAN's predictions remain highly accurate. The scatter plots in Figure 7 (a–c) continue to show strong alignment between predicted and true path costs, and the histograms in Figure 7 (a–c) maintain the same error concentration pattern seen in the size generalization study. Taken two evaluations together, these results demonstrate that SPAGAN is capable of generalizing effectively across both varying graph sizes and structural types. Its predictive accuracy and stability under diverse conditions make it a reliable and efficient surrogate for shortest-path estimation within the broader `Hephaestus` framework.

10.4 Robustness to SPAGAN Errors and Exact-Path Safeguard

10.4.1 Exact-Path Feasibility Check and Safeguard (PPS-I)

Let $G = (V, E)$ be a weighted directed graph, where each edge $e \in E$ has a perturbed weight $f_e(x_e)$ determined by the allocated budget x_e from the perturbation vector $\mathbf{x} \in \mathbb{R}^{|E|}_{\geq 0}$. Given a set of source–target pairs $\mathcal{K} = \{(s_i, t_i)\}_{i=1}^m$ and a threshold $T > 0$, we define the overall *feasibility rate* of a perturbation $\mathbf{x}$ as

$$\mathrm{Feas}(\mathbf{x}) \;:=\; \frac{1}{m}\sum_{i=1}^{m} \mathbf{1}\{\mathrm{SP}_G(s_i, t_i; \mathbf{x}) \;\geq\; T\}\,,$$

where the shortest-path length under perturbation $\mathbf{x}$ is given by $\mathrm{SP}_G(s, t; \mathbf{x}) = \min_{\rho \in \mathcal{P}(s,t)} \sum_{e \in \rho} f_e(x_e)$. **PPS-I** takes any candidate x (from PPS or RL) and performs: (i) compute $d_x(s_i, t_i)$ for all pairs via Dijkstra; (ii) while $d_x(s_i, t_i) < T$ for any i, identify a shortest path $\rho^\star$ and increment $\{x_e\}_{e \in \rho^\star}$ by the minimum amount to push $d_x(s_i, t_i)$ to T. This yields a final $\hat{x}$ with $\mathrm{Feas}(\hat{x}) = 1$.

Proposition (Guarantee of PPS-I). Let $\mathcal{K}$ be the set of QoSD pairs with threshold $T > 0$, and let the per-pair slack under a perturbation $\mathbf{x}$ be

$$\mathrm{slack}_{\mathbf{x}}(s, t) = \max\{0,\, T - \mathrm{SP}_G(s, t; \mathbf{x})\},$$

with total slack $S(\mathbf{x}) = \sum_{(s,t) \in \mathcal{K}} \mathrm{slack}_{\mathbf{x}}(s, t)$. Assume every edge-weight function $f_e(\cdot)$ is non-decreasing and locally Lipschitz, with a positive lower bound on its incremental gain:

$$f_e(x_e + 1) - f_e(x_e) \geq c_{\min} > 0.$$

Then, for any initial perturbation $\mathbf{x}$, PPS-I returns an updated $\hat{\mathbf{x}}$ such that $\mathrm{SP}_G(s, t; \hat{\mathbf{x}}) \geq T$ for all $(s, t) \in \mathcal{K}$, thereby ensuring **100% feasibility**. Moreover, the total additional cost is bounded by

$$\|\hat{\mathbf{x}}\|_1 - \|\mathbf{x}\|_1 \leq \tfrac{1}{c_{\min}}\, S(\mathbf{x}).$$

Proof. At each iteration, PPS-I selects a violating pair (s, t) with $\mathrm{SP}_G(s, t; \mathbf{x}) < T$, extracts its exact shortest path $\rho^\star$ under current $\mathbf{x}$, and increases budgets on edges $e \in \rho^\star$ by the smallest nonnegative increments Δx_e such that

$$\sum_{e \in \rho^\star} \big[f_e(x_e + \Delta x_e) - f_e(x_e)\big] \;\geq\; T - \mathrm{SP}_G(s, t; \mathbf{x}).$$

Since each f_e is non-decreasing, all shortest-path lengths are non-decreasing under the update. Hence the total slack $S(\mathbf{x}) = \sum_{(u,v) \in \mathcal{K}} \max\{0,\, T - \mathrm{SP}_G(u, v; \mathbf{x})\}$ strictly decreases whenever there is a violation. Because $\mathcal{K}$ is finite and slack is bounded below by 0, PPS-I must terminate in finitely many steps at some $\hat{\mathbf{x}}$ with $S(\hat{\mathbf{x}}) = 0$, i.e., $\mathrm{SP}_G(s, t; \hat{\mathbf{x}}) \geq T$ for all $(s, t) \in \mathcal{K}$.

We now relate the total increase in edge weights to the increase in budgets. From the discrete lower bound assumption

$$f_e(x_e + 1) - f_e(x_e) \geq c_{\min},$$

each unit increase of x_e increases the corresponding edge cost by at least $c_{\min}$. For multiple steps, we can sum this inequality Δx_e times:

$$f_e(\hat{x}_e) - f_e(x_e) = \sum_{j=0}^{\Delta x_e - 1} \big[f_e(x_e + j + 1) - f_e(x_e + j)\big] \geq \sum_{j=0}^{\Delta x_e - 1} c_{\min} = c_{\min}(\hat{x}_e - x_e).$$

This simply means that f_e grows at least linearly with slope $c_{\min}$. In other words, the function lies above a line of slope $c_{\min}$ passing through $(x_e, f_e(x_e))$. Summing this inequality over all edges yields

$$\sum_{e \in E} \big[f_e(\hat{x}_e) - f_e(x_e)\big] \geq c_{\min} \sum_{e \in E} (\hat{x}_e - x_e) = c_{\min} \|\hat{\mathbf{x}} - \mathbf{x}\|_1.$$

The left-hand side represents the total increase in edge weights caused by all PPS-I updates. By construction, this cumulative increase cannot exceed the initial total slack $S(\mathbf{x})$, since $S(\mathbf{x})$ quantifies exactly the total amount of shortest-path length that must be compensated to reach feasibility. Hence

$$c_{\min} \|\hat{\mathbf{x}} - \mathbf{x}\|_1 \leq S(\mathbf{x}) \quad \Rightarrow \quad \|\hat{\mathbf{x}}\|_1 - \|\mathbf{x}\|_1 \leq \tfrac{S(\mathbf{x})}{c_{\min}}.$$

Dually, the incremental upper bound (by Lipschitz) implies

$$f_e(\hat{x}_e) - f_e(x_e) \;\leq\; C_{\max}\,(\hat{x}_e - x_e) \quad \Rightarrow \quad \sum_{e \in E} \big[f_e(\hat{x}_e) - f_e(x_e)\big] \;\leq\; C_{\max}\,\|\Delta \mathbf{x}\|_1.$$

To eliminate the initial violations, the cumulative increase of shortest-path lengths must be at least the total deficit one needs to fill, which is lower-bounded by the initial total slack:

$$\sum_{e \in E} \big[f_e(\hat{x}_e) - f_e(x_e)\big] \;\geq\; S(\mathbf{x}).$$

Therefore,

$$S(\mathbf{x}) \;\leq\; C_{\max}\,\|\Delta \mathbf{x}\|_1 \quad \Rightarrow \quad \|\hat{\mathbf{x}}\|_1 - \|\mathbf{x}\|_1 \;\geq\; \frac{S(\mathbf{x})}{C_{\max}}.$$

It gives us the desired sandwich bound $\frac{S(\mathbf{x})}{C_{\max}} \leq \|\hat{\mathbf{x}}\|_1 - \|\mathbf{x}\|_1 \leq \frac{S(\mathbf{x})}{c_{\min}}$, together with guaranteed feasibility at termination.

10.4.2 Robustness to SPAGAN Prediction Noise

The purpose of this experiment is to evaluate the robustness of HEPHAESTUS under imperfect shortest-path predictions produced by SPAGAN. In real-world deployment, SPAGAN may encounter topologies or edge-weight distributions not seen during training, leading to degraded path estimates that could cascade through subsequent stages of the pipeline. Hence, it is crucial to quantify how such prediction noise affects overall feasibility, cost efficiency, and the effectiveness of our safeguard module PPS-I.

To model degraded SPAGAN predictions, we inject independent zero-mean noise with rate $\eta \in \{30\%, 10\%, 5\%\}$ into SPAGAN-estimated shortest-path scores during PPS and RL decision-making. Let $\tilde{d}(s,t)$ denote the SPAGAN-predicted distance, we perturb it as $\tilde{d}_\eta(s,t) = \tilde{d}(s,t) + \epsilon_{s,t}$, where $\epsilon_{s,t}$ is sampled to achieve the target noise level (calibrated on the validation split). We then evaluate: (i) PPS (greedy optimization on noisy estimates); (ii) RL refinement in latent space (also using noisy estimates for reward shaping); and (iii) PPS-I, which applies exact post-check correction via Dijkstra to guarantee feasibility. We report feasibility rate (%) over $m = 50$ source–target pairs and the total perturbation budget cost.

From Table 4 (with m=50 source–target pairs), we observe four key findings:

1. **PPS degrades under noisy predictions:** Feasibility drops by up to 22% (e.g., *RoadCA* with 30% noise), and cost increases by about 20% as perturbation budgets are misallocated to irrelevant edges.

Table 4: Sensitivity to SPAGAN noise and exact-path safeguard (PPS-I). We report feasibility (%) and total cost after each stage; PPS-I also shows wall-clock fix time (s) for the exact Dijkstra-based correction.

Dataset	Noise	PPS Feas.	PPS Cost	RL Feas.	RL Cost	PPS-I Feas.	PPS-I Cost	PPS-I Time
Email	30%	90.0	3284	94.0	2911	**100.0**	3005	20.2
	10%	94.0	3185	96.0	2759	**100.0**	2802	15.7
	5%	96.0	3157	98.0	2640	**100.0**	2712	6.1
	0%	**100.0**	3091	**100.0**	2691	**100.0**	2691	0.0
RoadCA	30%	78.0	13730	86.0	10985	**100.0**	11061	214.8
	10%	88.0	12556	92.0	10148	**100.0**	10795	137.2
	5%	92.0	12123	94.0	9892	**100.0**	9907	48.6
	0%	**100.0**	11283	**100.0**	9184	**100.0**	9277	0.0

2. **RL refinement recovers performance:** RL improves feasibility and reduces cost over PPS by leveraging gradient signals in latent space. For instance, on the *RoadCA* dataset with 30% noise, RL improves feasibility from 78% to 86% and reduces cost from 13,730 to 10,985.

3. **PPS-I ensures 100% feasibility in all cases:** Regardless of SPAGAN noise level, PPS-I consistently restores feasibility via exact Dijkstra-based correction. This confirms the pipeline's correctness guarantee.

4. **Cost penalty of PPS-I is modest:** Even in the worst case (*RoadCA*, 30% noise), the PPS-I cost increases by only $\sim$19% relative to the ideal (0% noise) case i.e., 11,061 vs. 9,277, while feasibility remains perfect.

10.5 Resilience to Weak Initial Data

This experiment aims to assess the system's ability to recover from weak or suboptimal initial data and to validate whether the latent-space refinement mechanism truly corrects, rather than amplifies, early-stage imperfections. In other words, we examine the self-reinforcing capability of the Morph–Refine loop under noisy or incomplete initialization. Our framework is explicitly designed to mitigate the influence of suboptimal or noisy solutions generated in the early phase. Unlike heuristic post-processing, our refinement operates entirely in the latent space, where a reinforcement-learning (RL) agent optimizes a differentiable reward that reflects both feasibility and cost efficiency. Consequently, even if the initial perturbation samples from the PPS stage are imperfect, the refinement progressively corrects them rather than propagating the errors.

To validate this behavior, we conduct a cycle-level robustness experiment. After each refinement cycle, the top-k highest-reward latent samples are added back to the pre-trained solution set $\mathcal{D}_{\text{sol}}$, forming a self-reinforcing feedback loop between the Morph (Mix-CVAE) and Refine (RL) phases. This feedback mechanism provides two benefits:

(i) Mix-CVAE retraining incorporates improved latent samples, enriching the coverage of feasible solution modes.

(ii) The RL policy learns on an increasingly structured latent manifold, enabling it to discover even better perturbations in subsequent cycles.

On *Email* with T=260%, the improvement saturates after roughly five refinement cycles, yielding a total cost reduction of $\sim$ 18.5%. On the larger-scale *RoadCA* dataset, cost reductions continue to increase even after eight cycles, peaking at over $\sim$ 26% under high thresholds. These results confirm that **HEPHAESTUS remains robust to weak or noisy initial data**, and that the latent-space feedback loop effectively improves data quality over successive cycles.

Table 5 reports the relative cost reduction (%) across refinement cycles under varying feasibility thresholds T. Both *Email* and *RoadCA* datasets exhibit clear monotonic improvements, showing that later cycles consistently achieve higher-quality perturbations.

Table 5: Cycle-level improvement in cost reduction (%) across thresholds T. The values represent budget savings relative to the initial solution (Cycle 0).

Dataset	Cycle	T=140%	T=180%	T=220%	T=260%
Email	0	0.00	0.00	0.00	0.00
	1	4.01	5.25	6.82	7.95
	2	8.67	10.15	12.44	14.21
	3	10.86	12.50	15.03	17.88
	4	11.07	12.97	15.58	18.04
	5	11.23	13.11	15.98	18.52
RoadCA	0	0.00	0.00	0.00	0.00
	1	7.01	8.55	10.11	11.58
	2	10.91	12.98	15.03	17.34
	3	12.83	15.21	17.85	20.19
	4	15.67	17.89	20.45	22.87
	5	17.42	19.53	22.10	24.53
	6	18.27	20.33	22.98	25.66
	7	18.51	20.76	23.42	26.09
	8	18.61	20.89	23.54	26.21

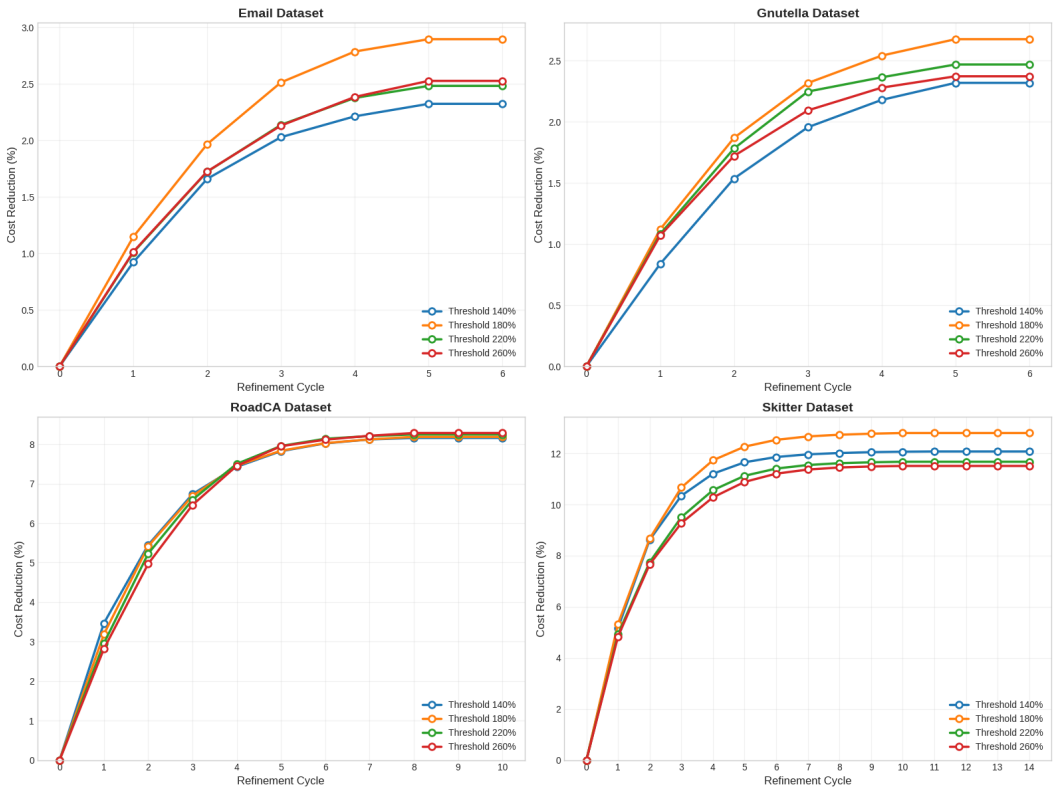


Figure 8: Evolution of Total Budget during the Refine phase (RL agent training)

10.5.1 RL Convergence Behavior under Different Problem Scales

To analyze how the RL policy behaves across different problem scales, we provide detailed training statistics on two real-world datasets: *Email* (1005 nodes, 25,571 edges) and *RoadCA* (1.96M nodes, 2.77M edges). Both experiments are conducted using clean SPAGAN predictions to isolate convergence dynamics.

Table 6: Training-phase convergence statistics on datasets of different scales. We report feasibility, expected total budget, and reward over three distinct training stages.

	Email			RoadCA		
Training Phase	**Feas. (%)**	**Expected Total Budget**	**Reward**	**Feas. (%)**	**Expected Total Budget**	**Reward**
Stage 1: Feasibility Search Behavior						
Episodes 0 - 5K	$80.0 \rightarrow 100$	$1921.65 \rightarrow 15147.21$	$201.44 \rightarrow 551.95$			
Episodes 0 - 10K				$40.0 \rightarrow 100$	$2641.58 \rightarrow 64453.49$	$545.37 \rightarrow 2348.62$
Stage 2: Cost Optimization Behavior						
Episodes 5K - 15K	98.0 ± 2	$15147.21 \rightarrow 2697.84$	$551.95 \rightarrow 874.72$			
Episodes 10K - 35K				98.0 ± 2	$64453.49 \rightarrow 9265.72$	$2348.62 \rightarrow 3975.03$
Stage 3: Convergence						
Episodes 15K - 50K	98.0 ± 2	2697.84 ± 17	874.72 ± 14			
Episodes 35K - 50K				98.0 ± 2	9265.72 ± 49	3975.03 ± 37

As shown in Table 6, training proceeds in three stages based on episode count. In Stage 1 (Episodes 0–5K / 0–10K), the policy rapidly improves feasibility (maximizing term 1 of reward function) from 80% to 100% (Email) and from 40% to 100% (RoadCA), demonstrating its ability to learn constraint satisfaction from scratch. In Stage 2, the focus shifts to cost optimization (term 2 of reward function): total budget drops significantly from 15147.21 to 2697.84 on Email, and from 64453.49 to 9265.72 on RoadCA, while feasibility remains approximately 98%. Finally, in Stage 3, the reward and cost metrics converge with low variance, e.g., on RoadCA, reward stabilizes at 3975.03 ± 37 and budget at 9265.72 ± 49, confirming stable convergence even in large-scale settings.

10.6 Comparison with Alternative Latent Optimization Strategies

A central motivation for the HEPHAESTUS framework is that reinforcement learning (RL) can effectively optimize latent representations beyond what static or population-based methods can achieve. To verify this, we conduct controlled comparisons against two alternative latent optimization strategies: Bayesian Optimization (BO) and Evolutionary Strategies (ES). This experiment aims to evaluate whether the RL-based refinement phase indeed provides superior sample efficiency, scalability to higher latent dimensions, and robustness to noisy SPAGAN predictions.

Each method is evaluated on the *Gnutella* dataset under varying latent dimensionalities d, reporting the final perturbation cost (lower is better) and feasibility rate (%). All evaluations are conducted using clean SPAGAN predictions (i.e., without additional noise) for a fair comparison.

Table 7: Comparison between RL-based refinement and alternative latent-space optimization strategies. Lower cost indicates better efficiency.

Method	**Latent Dim.** d	**Final Cost** $\downarrow$	**Feas. Rate** $\uparrow$
PPS (no refinement)	–	4118	100%
RL (Ours)	16	3435	100%
	64	**3419**	**100%**
Bayesian Opt. (BO)	16	3590	98%
	28	4055	52%
Evolutionary Strat. (ES)	16	3612	94%
	64	3598	98%

Table 8: Comparison of RL and ES under different levels of SPAGAN noise. All experiments are conducted on the *Gnutella* dataset with latent dimensions $d \in \{16, 64\}$. Lower cost indicates better performance, and higher feasibility denotes stronger robustness.

Method	**Latent Dim.** d	**SPAGAN Noise (%)**	**Final Cost** $\downarrow$	**Feas. Rate** $\uparrow$
PPS	–	0	4118	100%
		5	4221	96%
		10	4318	92%
		30	4392	86%
RL (Ours)	16	0	3435	100%
		5	3596	98%
		10	3714	92%
		30	3832	86%
	64	0	**3419**	**100%**
		5	3550	98%
		10	3668	96%
		30	3775	92%
Evo. Strat. (ES)	16	0	3612	94%
		5	3798	91%
		10	3945	87%
		30	4201	81%
	64	0	3598	98%
		5	3756	95%
		10	3912	91%
		30	4187	85%

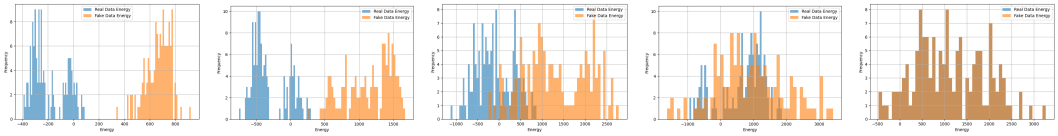
We observe that BO performs well only in low-dimensional latent spaces ($d \leq 30$) but suffers from severe sample inefficiency and surrogate modeling errors as dimensionality increases. ES scales better but converges slowly, often resulting in suboptimal costs. In contrast, our RL-based refinement achieves both the lowest final cost and full constraint satisfaction (100% feasibility) across all latent dimensions. This confirms that the policy-gradient–based refinement is more robust and sample-efficient than model-based (BO) or population-based (ES) strategies, especially in high-dimensional latent manifolds.

Robustness under Noisy SPAGAN Predictions. To further evaluate the resilience of these methods, we extend the comparison to noisy settings by injecting controlled levels of perturbation into SPAGAN's path predictions (5%, 10%, and 30%). This additional test closes the loop with the robustness discussion in Section 10.4.2 and examines how optimization strategies behave when their latent objectives are corrupted by upstream model uncertainty.

From these results, we observe that Evolutionary Strategies are more sensitive to SPAGAN prediction noise, likely due to their strong reliance on the quality of the pre-trained solution set $\mathcal{D}^{\text{sol}}$ generated during the Forge phase. In contrast, our RL-based refinement maintains both higher feasibility and lower cost under noise, demonstrating stronger robustness across latent dimensions and noise levels.

10.7 Energy Distribution Convergence during Minimax Training

The objective of this experiment is to evaluate how well the generative model Mix-CVAE (Ω) can progressively align its generated samples with the energy distribution implicitly defined by the Energy-Based Model (EBM) during adversarial co-training. Specifically, we aim to assess whether, as training proceeds, Ω learns to generate samples that lie in low-energy regions—those that the EBM assigns to real data. Since the true data distribution is unknown and cannot be visualized directly, we rely on the EBM's energy outputs as a surrogate for this alignment. The hypothesis is that if Ω successfully learns the real data distribution, the energy distributions of fake (generated) and real samples will gradually converge.



(a) Step 8 (b) Step 32 (c) Step 64 (d) Step 128 (e) Step 256

Figure 9: Energy histograms comparing real versus fake data across minimax training. The x-axis denotes the energy score assigned by the Energy-Based Model (EBM), while the y-axis indicates the frequency of samples observed at each energy level. Real data samples (blue), drawn from the ground-truth dataset $\mathfrak{D}^{\text{sol}}$, typically have low energy values and cluster near zero on the left of the x-axis, reflecting the EBM's preference for them. Fake data samples (orange), generated by the Mix-CVAE Ω, initially have higher energies but gradually shift leftward as training progresses. This convergence of the two distributions indicates that Ω is learning to generate samples that align more closely with the EBM's learned energy landscape.

Figure 9 illustrates this convergence process by plotting energy histograms over five key training steps. Real data from the dataset $\mathfrak{D}^{\text{sol}}$ are shown in blue (low energy region on left-side in x-axis), while fake data sampled from Ω are shown in orange on the right side. At early stages such as Step 8 and Step 32, there is a significant contrast: the EBM assigns low energy to real samples, which are densely concentrated near the left of the histogram, while fake samples occupy much higher energy regions, reflecting their low realism. As training progresses (Steps 64 and 128), the two distributions begin to overlap, indicating that Ω is learning to produce more realistic outputs that better match the EBM's learned energy profile. By Step 256, the energy distributions of real and fake samples nearly coincide, suggesting that Ω has successfully learned to generate samples that the EBM considers indistinguishable from real data. This result highlights a key insight: although the EBM is explicitly trained to minimize energy for real data and maximize it for generated data, the Mix-CVAE generator gradually catches up. Initially, it produces unrealistic, high-energy samples, but through adversarial feedback, it learns to synthesize low-energy (high-quality) solutions. Thus, we demonstrates that the generator effectively "chases" the moving energy boundary set by the EBM and ultimately converges to regions of high data likelihood.

10.8 Latent Space Visualization

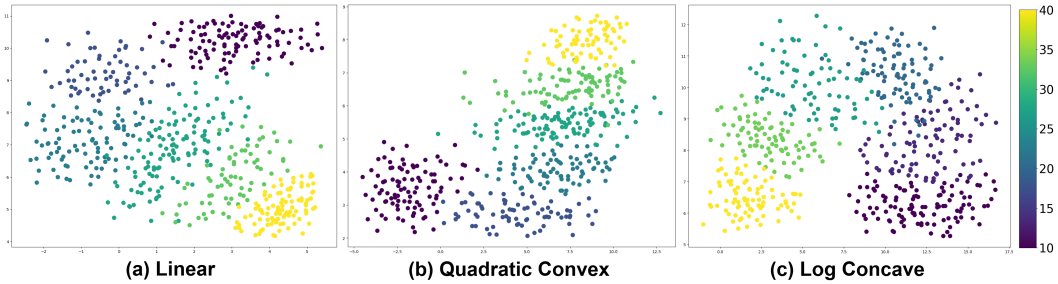


Figure 10: Conditional Latent Space Visualization via UMAP for the trained Mix-CVAE, conducted on the same synthetic graph but under varying threshold values T across different edge weight functions: Linear, Quadratic (Convex), and Log-Concave. Each point represents a latent vector $\mathbf{z}$ corresponding to a solution, colored by the associated threshold T. The structure of the latent space reveals how the model differentiates solution representations under changing constraints and cost dynamics, with clustering patterns indicating sensitivity to the underlying threshold conditions.

In this experiment, the goal is to assess whether the latent space learned by the conditional generative model (Mix-CVAE) meaningfully captures different constraints T on the same graph. Specifically, we visualize the latent vectors $\mathbf{z}$ using UMAP projections to examine whether different threshold values T result in separable or clustered embeddings—an indicator of successful conditional representation learning. Figure 10 shows the latent space organization when Mix-CVAE is trained and evaluated on the same synthetic graph but under three different edge weight functions: Linear, Quadratic (Convex), and Log-Concave. Across all three subfigures, we observe clear gradient-based separation and distinct

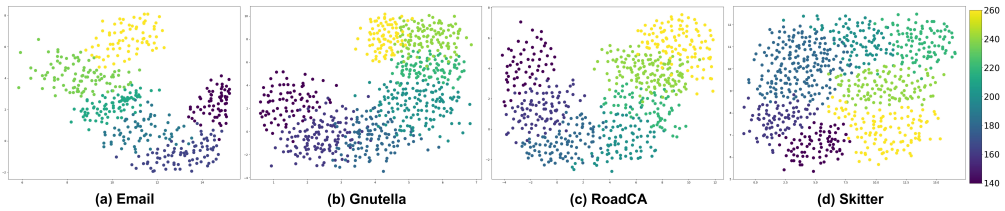


Figure 11: Conditional Latent Space Visualization for the trained Mix-CVAE under the Linear weight function setting. Each point represents a latent vector $\mathbf{z}$ corresponding to a generated solution, colored by either the input graph type or the threshold T. The presence of distinct clusters or separable regions in the latent space suggests that the model captures meaningful variations related to graph structure and problem-specific constraints, indicating successful conditional encoding.

clusters aligned with increasing values of T. This pattern suggests that the generative model is sensitive to threshold constraints and learns to organize the latent space accordingly. Interestingly, the separation structure varies with the cost function type: Linear produces smooth band-like transitions, while Log-Concave shows more localized clustering, potentially reflecting its sharper cost escalation characteristics.

Figure 11 complements this analysis by testing Mix-CVAE on real-world graphs (Email, Gnutella, RoadCA, Skitter) under the same linear edge cost setting. Again, latent vectors are visualized and colored by threshold. Across different graph topologies, we still consistently observe separable clusters and gradual transitions in $\mathbf{z}$-space with respect to T, indicating that the generative model generalizes across input graphs and captures structural information relevant to feasibility under constraints. In particular, datasets with greater structural diversity (e.g., Skitter and RoadCA) exhibit more complex spatial patterns, highlighting the model's ability to encode both graph topology and constraint semantics.

10.9 Impact of Expert Addition in Mix-CVAE

The goal of this experiment is to evaluate the effect of adding more experts in the Mix-CVAE architecture on modeling capability and final solution quality. Figures 12, 14, and 15 together provide a comprehensive view of how the number of experts (3, 5, 7, 9) influences the generative model's ability to approximate the true solution distribution and optimize budget outcomes.

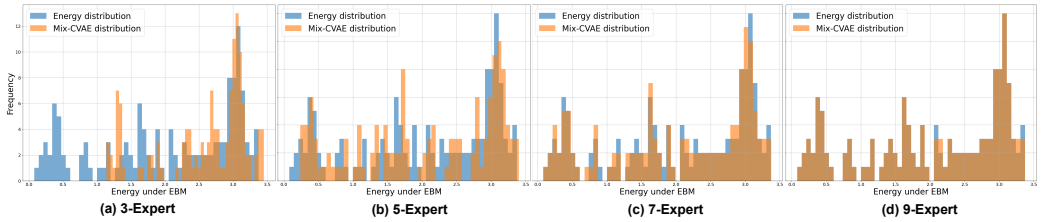


Figure 12: Comparison between the distribution from the EBM and the distribution from the Mix-CVAE with varying numbers of experts on a *synthetic network* with maximum density. Increasing the number of experts improves mode coverage and alignment with the target EBM distribution.

Impact of Expert Addition on Generative Modeling. Figures 12 and 13 compare the energy distributions of real solutions (as evaluated under the EBM) and fake samples generated by the Mix-CVAE across both synthetic and real network settings. When only 3 experts are used, the generated samples show poor alignment with the real data distribution, evidenced by noticeable discrepancies between the orange (Mix-CVAE) and blue (EBM) bars. As the number of experts increases, this alignment improves significantly—indicating that the model is better able to capture the underlying modes and structure of the target distribution. This validates the core motivation behind expert addition: increasing the number of experts enhances the model's capacity to represent multimodal solution spaces. Furthermore, Figure 14 provides additional insight by visualizing the latent space of the trained Mix-CVAE via UMAP on synthetic graphs. Each point represents a latent vector $\mathbf{z}$ colored by its corresponding threshold T. With more experts, the latent space becomes more

Figure 13: Impact of adding experts on *real-world networks*. From top to bottom: results on Email, Gnutella, RoadCA, and Skitter networks.

organized, exhibiting distinct clusters that correspond to different threshold levels. This structured separation suggests that the model is learning to conditionally represent diverse solution modes, improving both interpretability and the quality of generated samples.

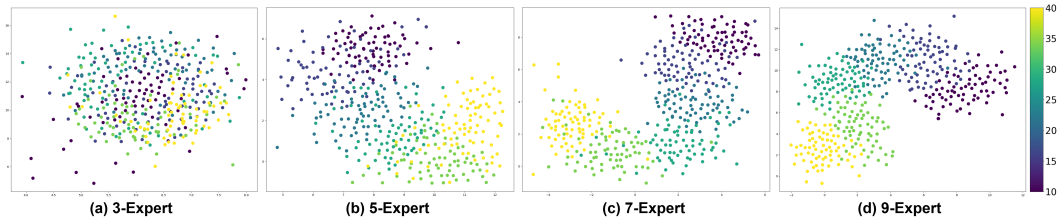
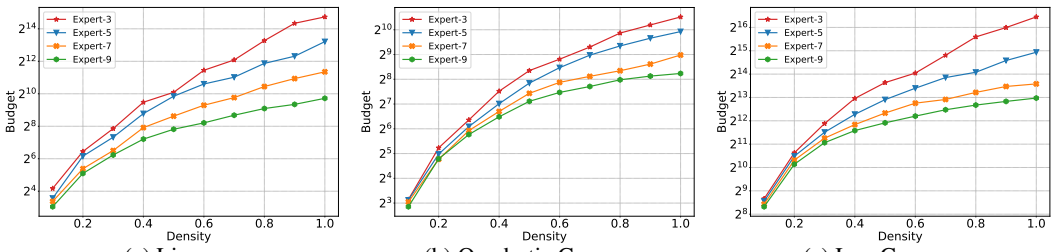


Figure 14: An example of conditional latent space visualization for the trained Mix-CVAE on the same synthetic graph and pairs but different thresholds. Points represent latent vectors $\mathbf{z}$, colored by threshold T. Clear clustering shows the latent space captures meaningful patterns.

Impact of Expert Addition on Total Cost. To evaluate whether the improved modeling capacity from adding more experts in Mix-CVAE leads to better optimization outcomes, we analyze its effect on the total budget required to solve QoSD instances. Figure 15 directly illustrates this downstream impact by plotting the average total budget (in log scale) achieved across a range of graph densities (from sparse to dense) under three distinct edge weight functions: Linear, Quadratic Convex, and Log Concave. We observe a consistent trend: increasing the number of experts from 3 to 9 leads to a progressive reduction in the final budget required to satisfy the QoSD constraint across all edge weight settings. The gap between configurations becomes more pronounced as



(a) Linear (b) Quadratic Convex (c) Log Concave

Figure 15: Impact of adding of experts (3, 5, 7, and 9) on the resulting total budget across varying graph densities, testing on the synthetic graph under (a) *Linear*, (b) *Quadratic Convex*, and (c) *Log Concave* edge weight functions.

graph density increases, where solution spaces become more complex and multimodal. Notably, the 9-expert model consistently outperforms the others, achieving the lowest budgets—especially under nonlinear edge weight functions like Log Concave and Quadratic Convex. This confirms that enhanced expressiveness in the generative model enables better exploration and exploitation of feasible regions in the solution space.

These results demonstrate a clear benefit of the expert addition strategy: as the number of experts increases, Mix-CVAE becomes more capable of capturing diverse solution modes, which directly improves solution quality. The budget savings are especially significant under higher density graphs, where the optimization problem is inherently harder and requires stronger modeling capacity to discover high-quality solutions.

10.10 Performance under Non-Linear Edge Weight Functions

The main paper presents only performance comparisons on real-world networks under the linear weight function, where the QoSD problem can be reformulated as an ILP and solved exactly. This section will show performance of `Hephaestus` in real networks under non-linear edge weight settings where such reformulations are no longer tractable for existing ML-based or optimization baselines. Specifically, Table 9 evaluates the Quadratic Convex setting, where each edge weight follows the form $f_e(x_e) = \aleph(x^2)$. This convex formulation rapidly increases edge cost as budget increases, meaning that even small increases in budget can suffice to surpass the threshold T. Fortunately, the latest versions of Gurobi support solving quadratic objectives, hence, an exact solver is still included in this setting for benchmarking. In contrast, Table 10 examines the Log-Concave case, where the edge weight is defined as $f_e(x_e) = \aleph(\ln x)$. This class of function is much slower to grow, so achieving a total cost above T often requires significantly more budget per edge. Importantly, Gurobi and other solvers do not support log-concave objectives natively in mixed-integer formulations, making exact optimization infeasible. Therefore, only approximation methods—Adaptive Trading (AT), Iterative Greedy (IG), Sampling Algorithm (SA)—and our method Hephaestus are reported in Table 10.

	Email				Gnutella				RoadCA				Skitter			
Method	140%	180%	220%	260%	140%	180%	220%	260%	140%	180%	220%	260%	140%	180%	220%	260%
Adaptive Trading	**1933**	4261	6647	7771	1558	3267	5842	7518	7747	12080	18761	29821	290772	714634	1637765	2743321
Iterative Greedy	1978	4301	6697	7825	1942	3635	7561	9774	7888	13893	19369	32049	347454	889621	1967255	3173690
Sampling Alg.	3687	8835	12532	16384	2803	4609	12670	15126	16271	28856	55014	60716	521783	1580349	2478424	5741999
Exact Solver	1907	4198	6491	7646	1497	3077	5499	7051	—	—	—	—	—	—	—	—
`Hephaestus`	1944	**4256**	**6602**	**7711**	**1529**	**3126**	**5631**	**7327**	**4955**	**9253**	**14107**	**20493**	**171157**	**493507**	**979447**	**1716357**

Table 9: Performance of HEPHAESTUS and baselines on four real datasets at different thresholds T. The best is highlighted in bold excluding exact solution. (Convex)

Performance under Quadratic Convex f_e: In Table 9 (Quadratic Convex), Hephaestus consistently achieves the lowest total cost among all approximation methods. On smaller graphs like Email and Gnutella, its performance is often close to the exact solver when available, while on larger graphs like RoadCA and Skitter, where the exact solver fails to run—Hephaestus still maintains strongest performance compared with remaining available baselines, highlighting its scalability and robustness to nonlinear cost functions.

Method	Email				Gnutella				RoadCA				Skitter			
	140%	180%	220%	260%	140%	180%	220%	260%	140%	180%	220%	260%	140%	180%	220%	260%
Adaptive Trading	3309	7595	12872	14759	6589	10897	15268	21147	20218	29266	46248	69457	1130533	4191546	6931595	9697272
Iterative Greedy	3473	7706	13095	16042	6639	11571	16092	21491	21301	30389	48197	71405	1339904	5174196	8753831	10111558
Sampling Alg.	4404	9589	15218	18336	7980	13957	19632	23286	38427	68553	95579	128118	5621568	7373689	10220892	15464201
`Hephaestus`	**3188**	**7320**	**10942**	**12318**	**6403**	**10754**	**13949**	**19110**	**17299**	**27242**	**42311**	**64796**	**424188**	**1115779**	**2608813**	**7064231**

Table 10: Performance (Total Attack Budget) of HEPHAESTUS and approximation baselines on four real datasets under Log Concave edge weights. Lower is better. Best approximation/HEPHAESTUS result is bolded. Exact Solver fails to run.

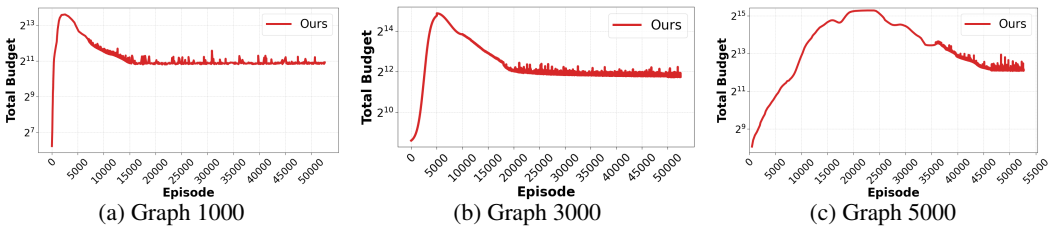
Performance under Log Concave f_e: Table 10 (Log Concave) further highlights Hephaestus' scalability and generality. Despite there is no longer a ground-truth solver, Hephaestus consistently outperforms AT, IG, and SA across all thresholds and network sizes. The gap in total budget becomes particularly significant for larger graphs and higher thresholds, as approximation methods struggle to effectively navigate the slow growth dynamics of the log-concave function. By contrast, Hephaestus's generative-refinement framework allows it to adapt to the more complex shape of the cost landscape.

These findings validate that Hephaestus remains highly effective in settings where other solvers or learning-based methods are inapplicable. Moreover, the results underscore the flexibility of Hephaestus across diverse graph structures, ranging from small, sparse email networks to massive Internet-scale topologies like Skitter. Moreover, unlike many existing learning-based methods—such as DIFFILO or Predict-and-Search—which are designed specifically for linear cost functions, `Hephaestus` is built to work across a wide range of cost models. This is possible because it separates the learning of solution structures (e.g., which paths to attack) from the specific form of f_e (e.g., linear, quadratic, log-concave). In fact, owing to its generative modeling approach in latent space, Hephaestus does not rely on any particular formula for how edge costs increase, hence, can handle different types of cost functions without needing to change its architecture. This flexibility makes Hephaestus more practical for real-world use, where both the structure of the network and how costs behave can vary significantly from one case to another.

10.11 Soundness of the Reward Function

Recall that the reward function used in the Refine phase of `Hephaestus` is defined as follows:

$$\mathcal{R}(\mathbf{x}_{i+1}) = \underbrace{F(G, \mathcal{K}, \hat{\mathbf{x}}_{i+1})}_{\text{(i) Smooth feasibility term}} \underbrace{- \varkappa \cdot \log\left(1 + \|\overline{\mathbf{x}}_{i+1}\|_1\right)}_{\text{(ii) Soft cost penalty term}} \tag{60}$$

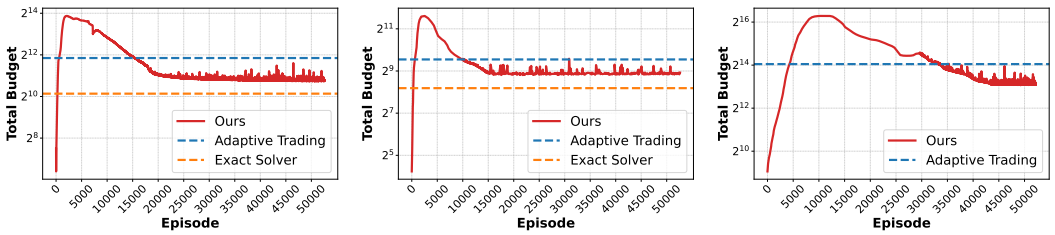


(a) Graph 1000 (b) Graph 3000 (c) Graph 5000

Figure 16: Evolution of Total Budget during the Refine phase (RL agent training)

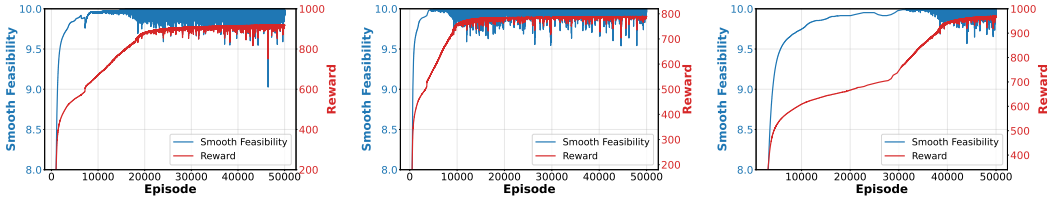
This reward formulation encourages policy π to generate solutions that are both feasible (increasing the sigmoid-based term for all critical pairs (s, t)) and efficient (penalizing a large total budget using the second term). The key idea is to guide the policy to find a balance between maximizing feasibility and minimizing the total cost.

In Figure 17, we show how the total budget evolves over training episodes across three different edge weight settings: Linear, Quadratic Convex, and Log-Concave. A noteworthy observation across all plots is that the budget initially increases sharply during the early training phase, especially visible in episodes 0–3000. This behavior can be explained as: at this stage, the RL agent has not yet learned effective policies and instead prioritizes feasibility, attempting to push more critical paths above the required threshold T, often by over-allocating budget. This is aligned with the feasibility



(a) Linear (b) Quadratic Convex (c) Log Concave

Figure 17: Evolution of Total Budget during the Refine phase (RL agent training)



(a) Linear (b) Quadratic Convex (c) Log Concave

Figure 18: Evolution of Feasible Solution quality (blue, left axis) and Agent Reward (red, right axis) over training episodes.

curve in Figure 18 (a–c), where the smooth feasibility skyrockets during the same initial phase, confirming that the agent is successfully raising path costs. As training continues and the feasibility nears saturation (close to a value of 10, meaning that the total cost of the shortest paths for all 10 target pairs exceeds the threshold T), we begin to observe a transition: the agent shifts focus toward reducing the overall budget while preserving feasibility. This can be seen in Figure 17(a), where from episode 3000 onwards, the total budget starts to steadily decline. Correspondingly, Figure 18(a) shows that feasibility remains stable, indicating that the agent is successfully optimizing the second term of the reward without sacrificing the first. However, a particularly interesting phenomenon occurs around episode 7000. As the agent becomes increasingly focused on minimizing the cost penalty, it temporarily neglects feasibility. This is reflected in Figure 18(a) as a sharp dip in smooth feasibility around that episode, suggesting some paths have dropped below the threshold. Simultaneously, Figure 17(a) shows a sudden plunge in total budget, confirming that the policy has aggressively reduced edge weights to cut cost. Eventually, the agent corrects this behavior. Realizing that dropping feasibility lowers the total reward, the policy re-adjusts — it selectively increases the budget on critical edges to recover feasibility, while continuing to reduce less impactful edges. This adaptive dynamic is exactly what the reward function is designed to elicit: the policy explores aggressively, corrects when needed, and eventually converges to a high-feasibility, low-cost solution.

This self-regulating behavior is consistent across Quadratic Convex (Figure 10b/11b) and Log-Concave (Figure 10c/11c) settings, although the trajectory is slightly more gradual in the latter due to the slower cost escalation of log-concave functions. The reward curves (in red, Figure 11) grow monotonically across all cases, confirming that the policy is indeed optimizing the reward function effectively. Overall, these experiments confirm the soundness of the reward design. It provides a useful training signal that balances feasibility and efficiency, and encourages intelligent exploration–exploitation dynamics during learning.

10.12 Relative optimal gap convergence

The goal of this experiment is to evaluate how effectively and efficiently each method approaches the optimal solution in terms of total budget, using relative optimal gap as the key metric. This gap quantifies the percentage deviation from the best-known solution, where lower values indicate better performance and zero denotes exact optimality. As observed in the Figure 19, existing learning-based baselines, such as L-MILPOPT, Predict-and-Search, and DiffILO—require post-processing using an exact solver like Gurobi to reduce their optimality gap. While this refinement phase eventually helps those methods converge, it often takes several thousand seconds and incurs substantial computational cost. In contrast, `Hephaestus` with Gurobi refinement (red curve) starts with high-quality initial solutions, thanks to the expressiveness of Mix-CVAE, and rapidly converges to zero gap much

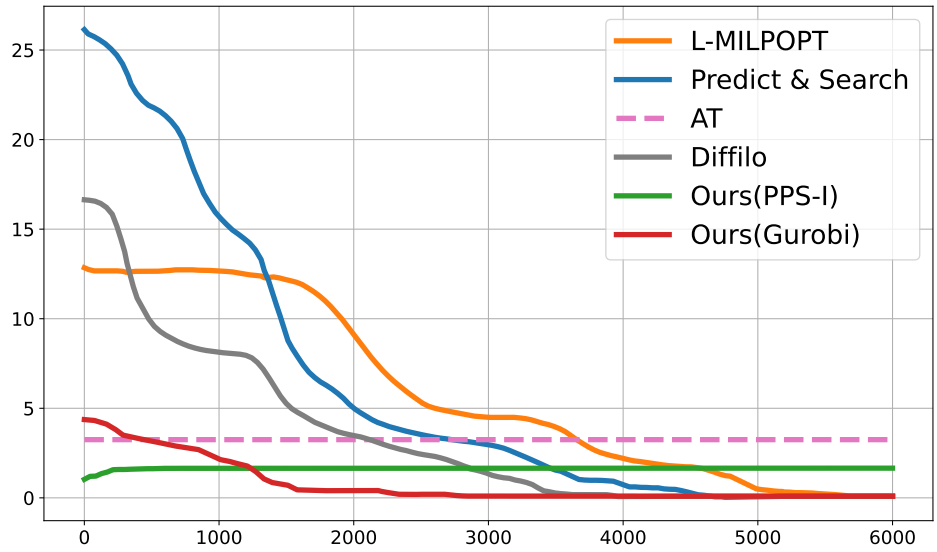


Figure 19: Convergence comparison of the relative optimal gap across `Hephaestus` and baseline methods under the Linear cost setting. The relative optimal gap measures the percentage deviation from the best-known (optimal) solution, with lower values indicating better performance. Existing ML-based methods (L-MILPOPT, Predict-and-Search, DiffILO) gradually reduce their gaps over time through external solver refinement (e.g., Gurobi), but typically require thousands of seconds. In contrast, `Hephaestus` with Gurobi (red curve) rapidly converges to zero gap due to its high-quality initial solutions and efficient generative modeling. `Hephaestus` with PPS-I (green curve), although not exact, achieves low gaps in significantly less time, offering a fast and practical alternative.

faster. This demonstrates the synergy between generative modeling and reinforcement learning in guiding the solution space effectively. Meanwhile, Hephaestus with PPS-I (green curve) does not reach exact optimality, as PPS-I is an approximation algorithm that does not enforce all path constraints exhaustively. However, it consistently achieves low optimality gaps in a fraction of the time compared to exact methods, offering an effective trade-off between solution quality and efficiency. An interesting observation in the curve is the initial rise in total cost during the first few seconds, after which the value quickly plateaus. This behavior comes from the fact that the initial solution generated by the RL policy π—may not fully satisfy all feasibility constraints. PPS-I then incrementally increases edge budgets based on marginal benefit-to-cost analysis, just enough to achieve feasibility, and terminates once all critical pairs are satisfied. Thus, this experiment underscores that `Hephaestus` is capable of adapting to different computational needs. It delivers fast approximate results with PPS-I, or provably optimal solutions with Gurobi refinement—while consistently outperforming existing baselines in both convergence quality and speed.

10.13 Hephaestus with different feasibility refinement

In this section, we want to show performance of `Hephaestus` under three refinement modes: initial prediction only (no refinement), refinement via PPS-Inference (PPS-I), and full Gurobi-based refinement. These configurations are compared against classical approximation baselines (AT, IG, SA), several learning-based approaches (L-MILPOPT, Predict-and-Search, DIFFILO), and the exact solver, across synthetic Erdős–Rényi graphs with increasing edge density ($p = 0.2, 0.5, 1.0$). As illustrated in Table 11, across all densities, `Hephaestus` 's initial predictions already yield strong performance, achieving near-optimal total cost and high feasibility—often outperforming other ML-based methods even before refinement. When PPS-I is applied, `Hephaestus` consistently reaches 100% feasibility with only minor increases in total cost. For instance, at $p = 1.0$, the PPS-I solution attains 2236 total budget, just 1.45% above the exact optimum (2204), while completing in only 37 seconds—over 200× faster than the full exact solver. This efficiency comes from the fact that the latent generative model already places solutions very close to the feasible region. PPS-I only needs to make slight local adjustments (i.e., increasing budgets on a few critical edges) to push all violated shortest-path constraints

Table 11: Performance comparison of `Hephaestus` (Hephaestus) under different refinement strategies (Initial Prediction, PPS-I, and Gurobi) across varying graph densities and benchmarked against baselines. The table reports total budget (lower is better), feasibility rate (higher is better), and running time (in seconds). Results demonstrate that `Hephaestus` consistently achieves optimal or near-optimal solutions with significantly lower computation time using PPS-I, while Gurobi refinement reaches exact feasibility at higher computational cost.

Graph Density (p)	Method	Total Budget ↓	Feasibility Rate ↑	Running Time (s) ↓
0.2	AT	163	100.00	44.10
	IG	189	100.00	43.23
	SA	240	100.00	508.49
	L-MILPOPT (Initial Pred.)	462	95.11	15.22
	L-MILPOPT (Gurobi)	119	100.00	240.96
	P&S (Initial Pred.)	411	95.52	34.57
	P&S (Gurobi)	119	100.00	716.31
	DIFFILO (Initial Pred.)	386	92.78	28.31
	DIFFILO (Gurobi)	119	100.00	412.64
	`Hephaestus` (Initial Pred.)	119	100.00	12.98
	`Hephaestus` **(PPS-I)**	119	100.00	13.72
	`Hephaestus` (Gurobi)	119	100.00	198.22
	Exact Solver	119	100.00	921.18
0.5	AT	856	100.00	147.38
	IG	992	100.00	144.46
	SA	1260	100.00	2409.83
	L-MILPOPT (Initial Pred.)	2735	84.17	37.56
	L-MILPOPT (Gurobi)	639	100.00	1503.01
	P&S (Initial Pred.)	2158	82.91	86.43
	P&S (Gurobi)	630	100.00	4476.94
	DIFFILO (Initial Pred.)	2027	88.36	70.78
	DIFFILO (Gurobi)	630	100.00	2579.00
	`Hephaestus` (Initial Pred.)	631	99.12	21.95
	`Hephaestus` **(PPS-I)**	637	100.00	24.00
	`Hephaestus` (Gurobi)	630	100.00	1238.88
	Exact Solver	630	100.00	5757.38
1.0	AT	2288	100.00	298.00
	IG	2471	100.00	350.33
	SA	3238	100.00	8113.90
	L-MILPOPT (Initial Pred.)	8573	78.12	175.08
	L-MILPOPT (Gurobi)	2237	100.00	5070.23
	P&S (Initial Pred.)	8549	78.36	172.85
	P&S (Gurobi)	2204	100.00	4836.18
	DIFFILO (Initial Pred.)	7095	83.01	141.55
	DIFFILO (Gurobi)	2204	100.00	3919.23
	`Hephaestus` (Initial Pred.)	2215	98.18	28.77
	`Hephaestus` **(PPS-I)**	2236	100.00	37.12
	`Hephaestus` (Gurobi)	2204	100.00	2218.16
	Exact Solver	2204	100.00	8127.64

above threshold T. Thus, PPS-I acts as a lightweight yet effective refinement method, ideal for scenarios where runtime is critical but small optimality gaps are acceptable. When exact optimality is required, combining `Hephaestus` with Gurobi refinement yields the best possible solution (zero optimality gap) in all cases. Importantly, due to the strong initialization from Mix-CVAE and RL, Gurobi converges significantly faster than from scratch—demonstrating the value of learning-based warm starts even for traditional solvers. Compared to L-MILPOPT, Predict-and-Search, and DIF-FILO, which depend heavily on post-hoc Gurobi refinement and suffer from lower feasibility in their initial outputs, `Hephaestus` stands out by producing feasible or near-feasible solutions immediately. This results in better cost-feasibility tradeoffs and reduced reliance on expensive solver calls.