
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:

1. DETAILS OF HEPHAESTUS	2
1.1. Hephaestus Main Framework	2
1.2. Forge	2
1.3. SPAGAN Training	3
1.4. Predictive Path Stressing (PPS)	4
1.5. Morph	4
1.6. Refine	6
1.7. Inference Process	7
1.8. Predictive Path Stressing - Inference (PPS-I)	7
2. THEOREMS AND PROOFS	8
2.1. Predictive Path Stressing Algorithm Ratio	8
2.2. Expert Augmentation Efficiency	10
2.3. Normalization Free Function	13
2.4. Differentiable Reward Function	14
2.5. Reward Estimation Consistency	16
3. DETAILS EXPERIMENTS AND ABLATION STUDIES	18
3.1. Dataset Details	18
3.2. Hyperparameter Settings	18
3.3. SPAGAN Generalization	21
3.4. Energy Distribution Convergence during Minimax Training	22
3.5. Latent Space Visualization	23
3.6. Impact of Expert Addition in Mix-CVAE	23
3.7. Performance under Non-Linear Edge Weight Functions	25
3.8. Soundness of the Reward Function	26
3.9. Relative optimal gap convergence	28
3.10. Hephaestus with different feasibility refinement	28
3.10. DISCUSSION	30

39 1 DETAILS OF HEPHAESTUS

40 1.1 Hephaestus Main Framework

Algorithm 1: Hephaestus Main Framework

Input: Initial graph dataset $\mathcal{D}^{\text{graph}}$, set of critical pairs \mathbf{K} , set of thresholds \mathbf{T} , hyperparameters for all components.
Output: Trained Hephaestus model (SPAGAN \mathfrak{F}_θ , 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  $\mathcal{D}^{\text{sol}} \leftarrow \emptyset$ 
  // -- Phase 1: Forge --
2  $(\mathfrak{F}_\theta, \mathcal{D}_{\text{initial}}^{\text{sol}}) \leftarrow \text{Forge}(\mathcal{D}^{\text{graph}}, \mathbf{K}, \mathbf{T})$ 
3  $\mathcal{D}^{\text{sol}} \leftarrow \mathcal{D}^{\text{sol}} \cup \mathcal{D}_{\text{initial}}^{\text{sol}}$ 
41 // -- Phase 2: Morph --
4  $(q, \Omega) \leftarrow \text{Morph}(\mathcal{D}^{\text{sol}})$ 
  // -- Phase 3: Refine (Iterative Self-Reinforcement) --
5 for each reinforcement learning episode  $e = 1, \dots, E_{\text{max\_episodes}}$  do
6   Top-K-Solutions  $\leftarrow \text{Refine}(q, \Omega, \pi, \mathcal{D}^{\text{sol}}, \mathfrak{F}_\theta)$ 
7    $\mathcal{D}^{\text{sol}} \leftarrow \mathcal{D}^{\text{sol}} \cup \text{Top-K-Solutions}$  // Augment solution dataset
8   if  $e \bmod E_{\text{retrain\_freq}} == 0$  then
9     Periodically retrain Morph with augmented data  $(q, \Omega) \leftarrow \text{Morph}(\mathcal{D}^{\text{sol}})$ 
10  $\pi \leftarrow \text{Finalize RL Agent Training from collected experiences}$ 
11 Return  $\mathfrak{F}_\theta, q, \Omega, \pi$ 

```

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

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

59 1.2 Forge

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

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

Algorithm 2: Forge

```

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

```

73 **1.3 SPAGAN Training**

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

Algorithm 3: SPAGAN Training with Subgraph Sampling

```

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

```

87 **1.4 Predictive Path Stressing (PPS)**

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

potential function is defined as: $\mathcal{C}(P, \mathbf{x}) = \sum_{\rho_{s,t} \in P} \min(T, \mathfrak{F}_\theta(G, s, t; \mathbf{x}))$, 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 $\mathfrak{F}_\theta(G, s, t; \mathbf{x}) \geq T$, at which point the final perturbation vector \mathbf{x} is returned (Line 27).

Algorithm 4: Predictive Path Stressing (PPS)

```

1 Procedure  $PPS(G = (V, E), \mathcal{K}, T, \{f_e\}, \mathbf{b}, \mathbf{x}_{\text{initial}}, \mathfrak{F}_\theta)$ 
   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  $\mathfrak{F}_\theta$ 
   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    $\mathcal{K}_{\text{violate}} \leftarrow \{(s, t) \in \mathcal{K} \mid \sum_{e \in \rho_{s,t}} f_e(x_e) < T, \rho_{s,t} = \text{SPAGANPath}(\mathfrak{F}_\theta, G, s, t; \mathbf{x})\};$ 
4   while  $\mathcal{K}_{\text{violate}} \neq \emptyset$  do
5      $P \leftarrow \emptyset;$ 
6     foreach  $(s, t) \in \mathcal{K}_{\text{violate}}$  do
7        $\rho_{s,t} \leftarrow \text{SPAGANPath}(\mathfrak{F}_\theta, G, s, t; \mathbf{x});$ 
8        $P \leftarrow P \cup \{\rho_{s,t}\};$ 
9     // Evaluate soft potential function
10     $\mathcal{C}(P, \mathbf{x}) \leftarrow 0;$ 
11    while  $\mathcal{C}(P, \mathbf{x}) < |P| \cdot T - \bar{\epsilon}$  do
12       $(e^*, \Delta^*, \delta_{\text{max}}) \leftarrow (\text{None}, \text{None}, -\infty);$ 
13       $\mathcal{E}_P \leftarrow \bigcup_{\rho \in P} \rho;$ 
14      foreach  $\rho_{s,t} \in P$  do
15         $\mathcal{C}(P, \mathbf{x}) \leftarrow \mathcal{C}(P, \mathbf{x}) + \min(T, \mathfrak{F}_\theta(G, s, t; \mathbf{x}));$ 
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             $\mathcal{C}(P, \mathbf{x}') \leftarrow \mathcal{C}(P, \mathbf{x}') + \min(T, \mathfrak{F}_\theta(G, s, t; \mathbf{x}'))$ 
22             $\delta \leftarrow \frac{\mathcal{C}(P, \mathbf{x}') - \mathcal{C}(P, \mathbf{x})}{\Delta};$ 
23            if  $\delta > \delta_{\text{max}}$  then
24               $(e^*, \Delta^*, \delta_{\text{max}}) \leftarrow (e, \Delta, \delta);$ 
25          // Apply optimal update
26           $\mathbf{x} \leftarrow \mathbf{x} + \Delta^* \cdot \mathbf{1}_{e^*};$ 
27          // Update violating pairs using SPAGAN
28           $\mathcal{K}_{\text{violate}} \leftarrow \{(s, t) \in \mathcal{K} \mid \sum_{e \in \rho_{s,t}} f_e(x_e) < T, \rho_{s,t} = \text{SPAGANPath}(G, s, t; \mathbf{x})\};$ 
29  Return  $\mathbf{x}$ 

```

1.5 Morph

The Morph phase is designed to model the distribution of high-quality QoS 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

114 $\mathbf{x}_{\text{real}} \sim \mathcal{D}^{\text{sol}}$, effectively pulling its density toward regions with meaningful feasible solutions. At the
 115 same time, it is encouraged to assign *high energy* (i.e., low likelihood) to generated (fake) samples
 116 $\mathbf{x}_{\text{fake}} \sim \Omega$, which come from the current generative model Ω . This adversarial learning setup drives
 117 the EBM away from areas the generator covers poorly, creating a pressure that helps both models
 118 evolve: the EBM becomes more selective, and the generator learns to cover harder regions.

Algorithm 5: Morph

```

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

```

120 Formally, the algorithm starts by initializing both the EBM q_θ and a mixture of Conditional VAEs
 121 $\Omega = [\Omega_0, \dots, \Omega_N]$, starting with one single expert (Line 2). Each minimax training iteration proceeds
 122 in two stages. First, the EBM is updated by contrasting energy scores between real samples and
 123 generated ones. Fake samples are produced by sampling a latent vector $\mathbf{z}_{\text{fake}} \sim \tilde{p}_\phi(\mathbf{z} | \mathbf{c})$ from the
 124 prior of any expert and decoding it via decoder \mathcal{M}_ϕ of Ω (Lines 4–6). The EBM loss pushes energy
 125 lower on real samples and higher on generated ones, with variance-based regularization to stabilize
 126 training (Line 7), and the parameters θ are updated accordingly (Line 8).

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

134 *strategy* (Lines 21–28). Every $K_{\text{check_expert}}$ steps, it evaluates whether the current mixture Ω underfits
 135 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
 136 score exceeds a threshold $\delta_{\text{expert_add}}$, indicating insufficient generative density, a new CVAE expert is
 137 added and trained specifically on that difficult region. This enables the generator to incrementally
 138 cover diverse and potentially multimodal distributions. The process continues until the maximum
 139 number of experts is reached or training ends. Finally, Morph returns both the trained EBM and the
 140 mixture-based generative model.

141 1.6 Refine

Algorithm 6: Refine

```

1 Procedure Refine( $q, \Omega, \pi, \mathcal{D}^{\text{sol}}, \mathfrak{F}_\theta$ )
  Input: Energy model  $q$ , generative model  $\Omega$ , RL policy  $\pi$ , solution dataset  $\mathcal{D}^{\text{sol}}$ , SPAGAN
    estimator  $\mathfrak{F}_\theta$ 
  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  $s = 1, \dots, S_{\text{max}}$  do
4   Sample instance  $(G, \mathcal{K}, T, \mathbf{x})$  from  $\mathcal{D}^{\text{sol}}$ ;
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  $t = 1, \dots, \mathfrak{T}_{\text{max}}$  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
142 // Evaluate feasibility score via SPAGAN approximation
14    
$$f\text{score} \leftarrow \sum_{(s_p, t_p) \in \mathcal{K}} \frac{1}{1 + \exp(-\zeta(\mathfrak{F}_\theta(G, s_p, t_p; \text{round}(\bar{\mathbf{x}})) - T))}$$

    // Compute reward (Eq. 6)
15     $R_t \leftarrow f\text{score} - \kappa \cdot \log(1 + \|\bar{\mathbf{x}}\|_1)$ 
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  $\pi$  using replay buffer (e.g., PPO, DDPG, gradient-ascent, etc.);
18     $\mathbf{z}_{\text{current}} \leftarrow \mathbf{z}_{\text{next}}$ ;
19    if  $R_t \geq R_{\text{thresh}}$  or  $t = \mathfrak{T}_{\text{max}}$  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  $\bar{\mathcal{S}}_{\text{topK}}$ 

```

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

1.7 Inference Process

Algorithm 7: Inference Process

```

1 Procedure Inference( $G_{\text{new}}, \mathcal{K}_{\text{new}}, T_{\text{new}}, \Omega, \pi, \mathfrak{F}_{\theta}$ )
   Input: New instance ( $G_{\text{new}}, \mathcal{K}_{\text{new}}, T_{\text{new}}$ ), trained  $\Omega, \pi, \mathfrak{F}_{\theta}$ 
   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  $\Omega$ ’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
169 5   for  $k = 1, \dots, K_{\text{inference\_steps}}$  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  $\sigma$  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  $\Omega$ );
11   $\mathbf{x}_{\text{final}}^* \leftarrow \text{PPS-I}(G_{\text{new}}, \mathcal{K}_{\text{new}}, T_{\text{new}}, \mathbf{x}_{\text{raw}})$  // 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}_{θ} . 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}_{init} is then obtained either by sampling from the prior of the trained Mix-CVAE $\tilde{p}_{\phi}(\mathbf{z} | \mathbf{c}_{\text{new}})$ (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}_{raw} using the decoder \mathcal{M}_{ϕ} 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 QoS constraints. The final output is then returned (Line 12).

1.8 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

191 to compute exact shortest paths $\rho_{s,t}$ and evaluates their true costs using the nonlinear edge functions
 192 $f_e(x_e)$, as shown in Lines 7, 8 and 27. This guarantees correctness under arbitrary cost functions
 193 but incurs higher computational overhead. Second, PPS-I accepts a non-zero initial perturbation
 194 vector $\mathbf{x}_{\text{initial}}$ as input (Line 3), allowing it to continually refine approximate solutions produced by
 195 learning-based modules rather than starting from scratch. The rest of the logic—computing soft
 196 potential (Lines 13–15), evaluating marginal gain for candidate updates (Lines 16–23), and selecting
 197 optimal edge increments (Line 26)—remains structurally similar to PPS. The set of violating pairs
 198 is refreshed using recomputed exact shortest paths (Line 27), and the process continues until full
 199 feasibility is achieved.

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

```

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

```

201 2 THEOREMS AND PROOFS

202 2.1 Predictive Path Stressing Algorithm Ratio

203 **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
 204 chosen from E such that $\Pr[\mathcal{E}^* \subseteq \mathcal{E}] = a$, where \mathcal{E}^* is the set of edges of the optimal solution and

205 $a \in (0, 1)$ is a constant. Given a parameter $\bar{\epsilon} > 0$, then running the Predictive Path Stressing algorithm
 206 on \mathcal{E} yields a solution \mathbf{x} such that $\mathcal{C}(P, \mathbf{x}) \geq |P|T - \bar{\epsilon}$ and $\mathbb{E}[\|\mathbf{x}\|_1] \leq \frac{(1+h \ln(n) + \ln T + \ln(1/\bar{\epsilon}))}{a} \text{OPT}$

207 *Proof.* Recall that $\mathcal{C}(P, \mathbf{x}) = \sum_{p \in P} \min\left(T, \sum_{e \in p} f_e(x_e)\right)$ and thus $\mathcal{C}(P, \cdot)$ is monotone. Denote \mathbf{x}^i
 208 is the partial solution after iteration i of the algorithm, e_i is the edge added to the solution in iteration
 209 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.

210 Since the Predictive Path Stressing algorithm selects, in each iteration i , the edge e that maximizes the
 211 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
 212 as the average marginal gain per unit cost in the optimal solution given by $\frac{|P|T - \mathcal{C}(P, \mathbf{x}^i)}{\text{OPT}}$. Therefore,
 213 given the solution \mathbf{x}^{i-1} , we have

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

$$\geq \frac{|P|T - \mathcal{C}(P, \mathbf{x}^{i-1})}{\text{OPT}} \quad (2)$$

$$\implies \mathcal{C}(P, \mathbf{x}^i) - \mathcal{C}(P, \mathbf{x}^{i-1}) \geq \frac{j_i}{\text{OPT}} (|P|T - \mathcal{C}(P, \mathbf{x}^{i-1})). \quad (3)$$

214 Therefore,

$$\mathbb{E}[\mathcal{C}(P, \mathbf{x}^i) - \mathcal{C}(P, \mathbf{x}^{i-1}) | \mathbf{x}^{i-1}] \quad (4)$$

$$\geq \mathbb{E}\left[\Pr[\mathcal{E}^* \subseteq \mathcal{E}] \cdot \frac{j_i}{\text{OPT}} (|P|T - \mathcal{C}(P, \mathbf{x}^{i-1})) + (1 - \Pr[\mathcal{E}^* \subseteq \mathcal{E}]) \cdot 0 | \mathbf{x}^{i-1}\right] \quad (5)$$

$$\geq \mathbb{E}\left[\frac{aj_i}{\text{OPT}} (|P|T - \mathcal{C}(P, \mathbf{x}^{i-1})) | \mathbf{x}^{i-1}\right]. \quad (6)$$

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

$$\mathbb{E}[\mathcal{C}(P, \mathbf{x}^i) - \mathcal{C}(P, \mathbf{x}^{i-1})] \geq \mathbb{E}\left[\frac{aj_i}{\text{OPT}} (|P|T - \mathcal{C}(P, \mathbf{x}^{i-1}))\right]. \quad (7)$$

216 Re-arranging the above inequality gives:

$$\mathbb{E}[|P|T - \mathcal{C}(P, \mathbf{x}^i)] \leq \mathbb{E}\left[|P|T - (\mathcal{C}(P, \mathbf{x}^{i-1}) + \frac{aj_i}{\text{OPT}} (|P|T - \mathcal{C}(P, \mathbf{x}^{i-1})))\right] \quad (8)$$

$$\leq \mathbb{E}\left[(|P|T - \mathcal{C}(P, \mathbf{x}^{i-1})) - \frac{aj_i}{\text{OPT}} (|P|T - \mathcal{C}(P, \mathbf{x}^{i-1}))\right] \quad (9)$$

$$= \mathbb{E}\left[\left(1 - \frac{aj_i}{\text{OPT}}\right) (|P|T - \mathcal{C}(P, \mathbf{x}^{i-1}))\right]. \quad (10)$$

217 Let t is the number iteration. By applying the inequality (10) iteratively over these t iterations, we
 218 obtain the following:

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

219 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
 220 have

$$\bar{\epsilon} \leq |P|T - \mathcal{C}(P, \mathbf{x}^{t-1}) \leq |P|T \cdot \exp\left(-\frac{a\|\mathbf{x}^{t-1}\|_1}{\text{OPT}}\right) \quad (11)$$

$$\iff |P|T \cdot \exp\left(-\frac{a\|\mathbf{x}^{t-1}\|_1}{\text{OPT}}\right) \geq \bar{\epsilon} \quad (12)$$

$$\iff \exp\left(-\frac{a\|\mathbf{x}^{t-1}\|_1}{\text{OPT}}\right) \geq \frac{\bar{\epsilon}}{|P|T} \quad (13)$$

$$\iff \frac{a\|\mathbf{x}^{t-1}\|_1}{\text{OPT}} \leq \ln\left(\frac{|P|T}{\bar{\epsilon}}\right) \quad (14)$$

$$\iff \|\mathbf{x}^{t-1}\|_1 \leq \frac{\text{OPT}}{a} \ln\left(\frac{|P|T}{\bar{\epsilon}}\right). \quad (15)$$

221 Besides, from the inequality (7), we also have (in expectation)

$$\mathcal{C}(P, \mathbf{x}^{t-1}) + \frac{aj_t}{\text{OPT}} (|P|T - \mathcal{C}(P, \mathbf{x}^{t-1})) \leq \mathcal{C}(P, \mathbf{x}^t) \quad (16)$$

$$\iff \frac{aj_t}{\text{OPT}} (|P|T - \mathcal{C}(P, \mathbf{x}^{t-1})) \leq \mathcal{C}(P, \mathbf{x}^t) - \mathcal{C}(P, \mathbf{x}^{t-1}) \quad (17)$$

$$\iff aj_t (|P|T - \mathcal{C}(P, \mathbf{x}^{t-1})) \leq \text{OPT} (\mathcal{C}(P, \mathbf{x}^t) - \mathcal{C}(P, \mathbf{x}^{t-1})) \quad (18)$$

$$\implies aj_t \leq \text{OPT} \quad (\text{Since } |P|T - \mathcal{C}(P, \mathbf{x}^{t-1}) > \mathcal{C}(P, \mathbf{x}^t) - \mathcal{C}(P, \mathbf{x}^{t-1})) \quad (19)$$

$$\implies j_t \leq \frac{\text{OPT}}{a}. \quad (20)$$

222 The set P of feasible paths can be upper bounded in terms of the maximum path length and the number
 223 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
 224 each edge has weight at least w_{\min}), the number of feasible paths of the is upper-bounded by n^h . We
 225 therefore have:

$$\mathbb{E}[\|\mathbf{x}\|_1] = \mathbb{E}[\|\mathbf{x}^{t-1}\|_1] + \mathbb{E}[j_t] \leq \frac{\text{OPT}}{a} + \frac{\text{OPT} \ln\left(\frac{|P|T}{\bar{\epsilon}}\right)}{a} \quad (21)$$

$$\leq \frac{\text{OPT}}{a} \left(1 + \ln\left(\frac{|P|T}{\bar{\epsilon}}\right)\right) \quad (22)$$

$$\leq \frac{\text{OPT}}{a} \left(1 + \ln\left(\frac{n^h T}{\bar{\epsilon}}\right)\right) \quad (23)$$

$$\leq \frac{\text{OPT}}{a} \left(1 + h \ln(n) + \ln T + \ln\left(\frac{1}{\bar{\epsilon}}\right)\right) \quad (24)$$

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

229 **Remark 2.1.** (Approximation ratio when f_i is integer-valued). In Theorem 1, if f_i is integer-valued
 230 for $i \in [m]$ and we set $\bar{\epsilon} \in (0, 1)$, then running the Predictive Path Stressing algorithm on \mathcal{E} yields a
 231 feasible solution \mathbf{x} , i.e., $\mathcal{C}(P, \mathbf{x}) = |P|T$ such that $\mathbb{E}[\|\mathbf{x}\|_1] \leq \frac{(1+h \ln(n)+\ln T)}{a} \text{OPT}$

232 **Remark 2.2.** If PPS selects all edges in the optimal solution, i.e., $\Pr[\mathcal{E}^* \subseteq \mathcal{E}] = 1$, then running
 233 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
 234 $\mathbb{E}[\|\mathbf{x}\|_1] \leq (1 + h \ln(n) + \ln T + \ln(1/\bar{\epsilon})) \text{OPT}$

235 2.2 Expert Augmentation Efficiency

236 **Theorem 2 (Expert Augmentation Efficiency):** Suppose there exists a constant $\epsilon > 0$ such that
 237 $\mathbb{P}_{p(x)}\{\chi(x) > \delta\} \geq \epsilon$; on the region $\{x : \chi(x) > \delta\}$, a new expert $\Omega_{N+1}(x)$ is added with
 238 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
 239 some $c \in (0, 1)$; then the updated mixture $\Omega'(x) = \alpha(x) \Omega_{N+1}(x) + (1 - \alpha(x)) \Omega(x)$ satisfies

240 $\text{KL}(q||\Omega') \leq \text{KL}(q||\Omega) - \gamma(\delta, \epsilon)$, where $\gamma(\delta, \epsilon) = a_0(\delta + \log c) \epsilon_0 > 0$ and $\epsilon_0 > 0$ is a lower bound
 241 on $\int_{\chi(x) > \delta} q(x) dx$.

242 *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
 243 extend the gating network so that its output becomes:

$$w'_i(x), \quad i = 0, 1, \dots, N+1, \quad \text{with} \quad \sum_{i=0}^{N+1} w'_i(x) = 1. \quad (25)$$

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

$$\Omega'(x) = \alpha(x) \Omega_{N+1}(x) + (1 - \alpha(x)) \Omega(x). \quad (26)$$

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

$$\alpha(x) = \mathbf{1}\{\chi(x) > \delta\} a(x), \quad (27)$$

247 and for simplicity we take $a(x) = a_0$ for those x with $\chi(x) > \delta$ where $0 < a_0 < 1$. Note that, \bar{a}
 248 being an upper bound to ensure that the new expert does not completely override the previous mixture.
 249 Next, we consider the KL divergence between $q(x)$ and the original mixture $\Omega(x)$,

$$\text{KL}(q||\Omega) = \int q(x) \log \frac{q(x)}{\Omega(x)} dx, \quad (28)$$

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

$$\begin{aligned} \text{KL}(q||\Omega') &= \int q(x) \log \frac{q(x)}{\Omega'(x)} dx \\ &= \int q(x) \log \frac{q(x)}{\alpha(x) \Omega_{N+1}(x) + (1 - \alpha(x)) \Omega(x)} dx \\ &= \int q(x) \log q(x) - q(x) \log [\alpha(x) \Omega_{N+1}(x) + (1 - \alpha(x)) \Omega(x)] dx \\ &\leq \int q(x) [\log q(x) - \alpha(x) \log \Omega_{N+1}(x) - (1 - \alpha(x)) \log \Omega(x)] dx \\ &= \int q(x) \log q(x) dx - \int q(x) \alpha(x) \log \Omega_{N+1}(x) dx - \int q(x) (1 - \alpha(x)) \log \Omega(x) dx. \end{aligned} \quad (29)$$

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

$$\begin{aligned} \log \Omega'(x) &= \log (\alpha(x) \Omega_{N+1}(x) + (1 - \alpha(x)) \Omega(x)) \\ &\geq \alpha(x) \log \Omega_{N+1}(x) + (1 - \alpha(x)) \log \Omega(x) \end{aligned} \quad (30)$$

$$\begin{aligned} \implies -\log \Omega'(x) &\leq -\alpha(x) \log \Omega_{N+1}(x) - (1 - \alpha(x)) \log \Omega(x). \\ \implies -\int q(x) \log \Omega'(x) dx &\leq -\int q(x) \alpha(x) \log \Omega_{N+1}(x) dx - \int q(x) (1 - \alpha(x)) \log \Omega(x) dx. \end{aligned} \quad (31)$$

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

$$\begin{aligned}
\text{KL}(q\|\Omega) &= \int q(x) \log \frac{q(x)}{\Omega(x)} dx \\
&= \int q(x) \log q(x) dx - \int q(x) \log \Omega(x) dx.
\end{aligned} \tag{32}$$

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

$$\begin{aligned}
\text{KL}(q\|\Omega') &= \int q(x) \log \frac{q(x)}{\Omega'(x)} dx \\
&= \int q(x) \log q(x) dx - \int q(x) \log \Omega'(x) dx.
\end{aligned} \tag{33}$$

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

$$\begin{aligned}
\text{KL}(q\|\Omega') - \text{KL}(q\|\Omega) &= \left[\int q(x) \log q(x) dx - \int q(x) \log \Omega'(x) dx \right] \\
&\quad - \left[\int q(x) \log q(x) dx - \int q(x) \log \Omega(x) dx \right] \\
&= - \int q(x) \log \Omega'(x) dx + \int q(x) \log \Omega(x) dx.
\end{aligned} \tag{34}$$

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

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

$$\text{KL}(q\|\Omega') - \text{KL}(q\|\Omega) \leq - \int q(x) \alpha(x) [\log \Omega_{N+1}(x) - \log \Omega(x)] dx. \tag{36}$$

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

$$\text{KL}(q\|\Omega') - \text{KL}(q\|\Omega) \leq -a_0 \int_{\chi(x) > \delta} q(x) \log \frac{\Omega_{N+1}(x)}{\Omega(x)} dx. \tag{37}$$

261 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
262 new expert is designed such that, on this region, $\Omega_{N+1}(x) \geq c q(x)$, where $c \in (0, 1)$ is a constant.
263 Then it holds that:

$$\frac{\Omega_{N+1}(x)}{\Omega(x)} \geq c \frac{q(x)}{\Omega(x)} > c \exp(\delta). \tag{38}$$

264 Taking logarithms gives:

$$\log \frac{\Omega_{N+1}(x)}{\Omega(x)} \geq \delta + \log c. \quad (39)$$

265 Thus, if we let:

$$\eta = \int_{\chi(x) > \delta} q(x) dx, \quad (40)$$

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

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

267 Therefore, the reduction in KL divergence satisfies:

$$\text{KL}(q \parallel \Omega') - \text{KL}(q \parallel \Omega) \leq -a_0 (\delta + \log c) \epsilon_0.$$

268 Defining:

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

269 We conclude that:

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

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

272 **Remark 2.3** (Expert Augmentation under Reverse KL). *Theorem 2 remains applicable under the*
273 *reverse KL objective $\text{KL}(\Omega \parallel q)$. While reverse KL encourages mode-seeking behavior, driving each*
274 *individual expert $\Omega_k(x)$ to focus on distinct high-density regions and overlook other regions—the*
275 *overall mixture $\Omega(x) = \sum_k w_k(x) \Omega_k(x)$ can still approximate a multi-modal target $q(x)$ effectively,*
276 *as long as each expert is trained to specialize well on different mode regions of q . In this case, the*
277 *gating network $w_k(x)$ only needs to route a given input to the most suitable expert, allowing the*
278 *mixture to cover the full distribution even under a mode-seeking training objective.*

279 2.3 Normalization Free Function

280 **Theorem 3 (Normalization Free Function):** The objective function $\min_q \max_{\Omega} \{\text{KL}(p \parallel q) -$
281 $\text{KL}(\Omega \parallel q)\}$, is normalizing free which is independent with Z .

282 *Proof.* By definition, the KL divergence between $p(x)$ and $q(x)$ is $\text{KL}(p \parallel q) = \int p(x) \log \frac{p(x)}{q(x)} dx$.

283 Since $q(x) = \frac{\exp(-\frac{E(\mathbf{x})}{\tau})}{Z}$, we can rewrite it as:

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

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

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

285 The difference between the two KL divergences is then:

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

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

287 2.4 Differentiable Reward Function

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

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

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

292 where $\mathcal{L}(z) = \frac{1}{1+e^{-\zeta z}}$ is a sigmoid function with slope parameter $\zeta > 0$, $\Upsilon(\mathbf{x}) = \log(1 + e^{\mathbf{x}})$,
 293 applied coordinate-wise, and $T > 0$ is the feasibility threshold, and $\kappa > 0$ is a regularization
 294 hyperparameter. We aim to show that $\mathcal{R}(\mathbf{x})$ is continuously differentiable on any open domain
 295 $\mathcal{X} \subset \mathbb{R}^m$. First, the estimator \mathfrak{F}_θ is a deep graph neural network constructed as a composition of L
 296 differentiable layers:

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

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

$$\mathcal{L}_{s,t}(\mathbf{x}) := \mathcal{L}(\mathfrak{F}_\theta(G, s, t; \mathbf{x}) - T).$$

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

$$\nabla \mathcal{L}_{s,t}(\mathbf{x}) = \mathcal{L}'(\mathfrak{F}_\theta(G, s, t; \mathbf{x}) - T) \cdot \nabla \mathfrak{F}_\theta(G, s, t; \mathbf{x}).$$

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

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

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

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

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

$$\Upsilon'_i(x_i) = \frac{e^{x_i}}{1 + e^{x_i}} = \mathcal{L}(x_i)$$

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

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

310 which is well-defined and continuous on \mathbb{R}^m . Putting both together that $F(\mathbf{x}) = \sum_{(s,t) \in \mathcal{K}} \mathcal{L}_{s,t}(\mathbf{x})$
 311 and $\Lambda(\mathbf{x}) = \log(1 + \|\Upsilon(\mathbf{x})\|_1)$ is both differentiable. Therefore, the reward function $\mathcal{R}(\mathbf{x}) =$
 312 $F(\mathbf{x}) - \kappa \cdot \Lambda(\mathbf{x})$ is differentiable on any open set $\mathcal{X} \subset \mathbb{R}^m$, i.e., $\mathcal{R} \in C^1(X)$.

313 This concludes the proof.

2.5 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 $\mathcal{R}(\hat{\mathbf{x}}_i) > \mathcal{R}(\mathbf{x}_i)$, 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}_ϕ is Lipschitz-continuous.

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

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

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:

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

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

To prove that \mathcal{M}_ϕ 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:

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

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

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

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

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:

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

Therefore, \mathcal{M}_ϕ 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}_ϕ is Lipschitz-continuous. Given Ω_Θ is well converged, with \mathcal{M}_ϕ 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}_ϕ 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 \mathcal{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.

3 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.

3.1 Dataset Details

Table 1 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 1: 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

3.2 Hyperparameter Settings

In Table 2, 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 2: Hyperparameter Settings.

Component/Phase	Parameter	Value
Phase 1: Forge		
SPAGAN (\mathfrak{F}_θ)	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{e}_{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 $\hat{e}_{GA} = 2 \times 10^{-3}$ 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 2x 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.

3.3 SPAGAN Generalization

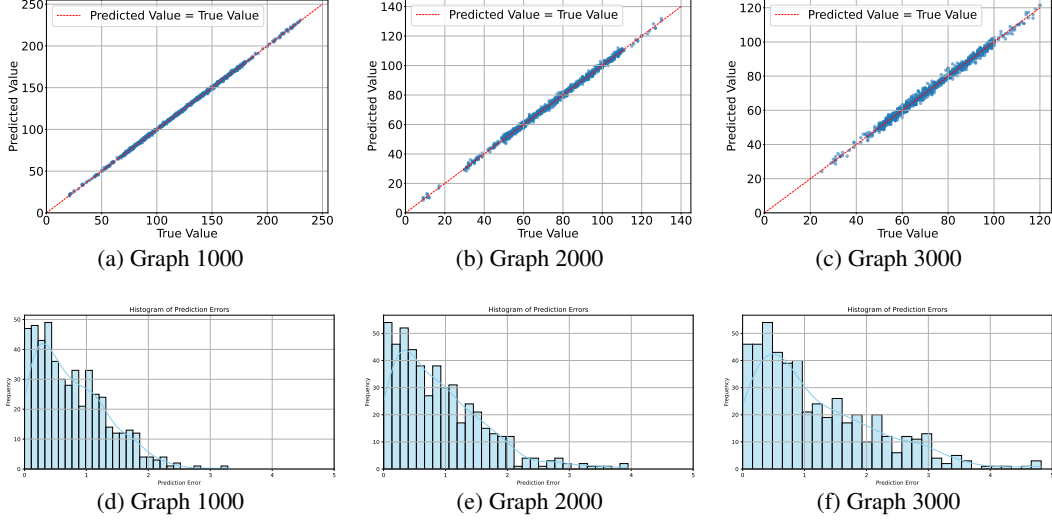


Figure 1: Predictive Accuracy Across Different Graph Sizes.

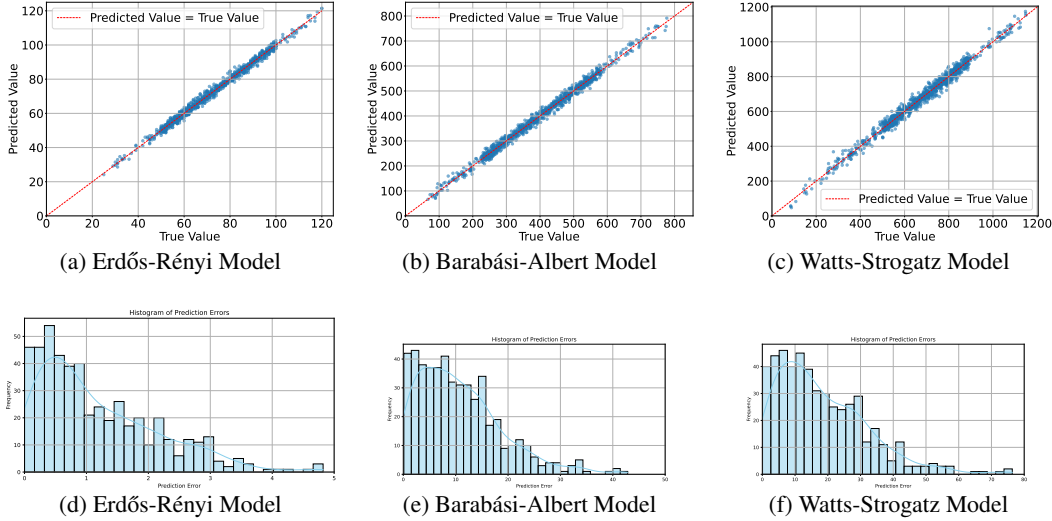


Figure 2: Predictive Accuracy Across Different Graph Topologies.

In our framework, SPAGAN model (\mathcal{F}_θ) 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 1 and 2 show the results. The scatter plots in Figure 1 (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 2 (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 1 and 2, SPAGAN’s predictions remain highly accurate. The scatter plots in Figure 2 (a–c) continue to show strong alignment between predicted and true path costs, and the histograms in Figure 2 (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.

3.4 Energy Distribution Convergence during Minimax Training

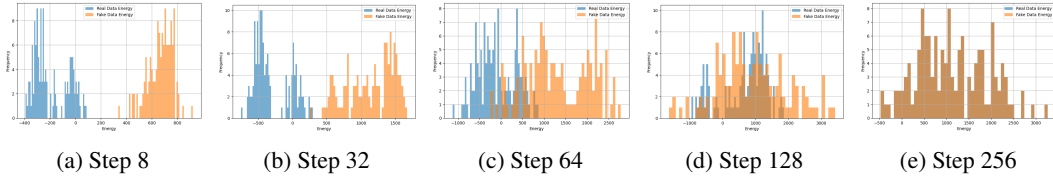


Figure 3: Energy histograms comparing real versus fake data across minimax training. The x-axis denotes the energy score assigned by the Energy-Based Model (EBM), while the y-axis indicates the frequency of samples observed at each energy level. Real data samples (blue), drawn from the ground-truth dataset \mathcal{D}^{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.

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.

Figure 3 illustrates this convergence process by plotting energy histograms over five key training steps. Real data from the dataset \mathcal{D}^{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 demonstrate that the generator effectively “chases” the moving energy boundary set by the EBM and ultimately converges to regions of high data likelihood.

3.5 Latent Space Visualization

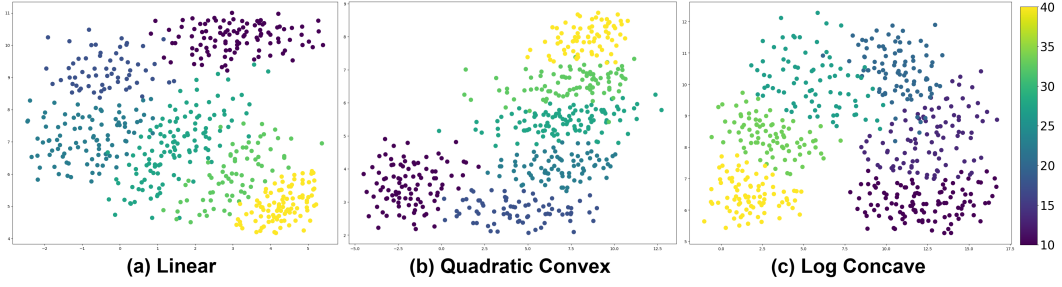


Figure 4: Conditional Latent Space Visualization via UMAP for the trained Mix-CVAE, conducted on the same synthetic graph but under varying threshold values T across different edge weight functions: Linear, Quadratic (Convex), and Log-Concave. Each point represents a latent vector \mathbf{z} corresponding 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.

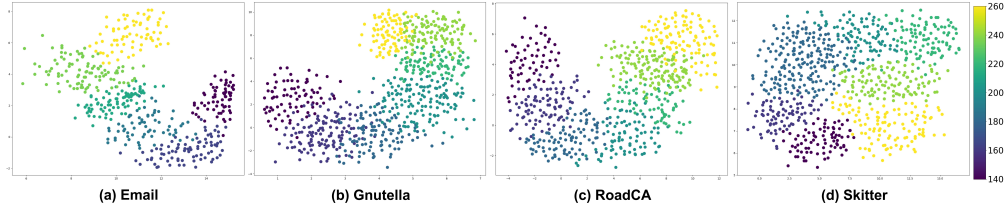


Figure 5: Conditional Latent Space Visualization for the trained Mix-CVAE under the Linear weight function setting. Each point represents a latent vector \mathbf{z} corresponding to a generated solution, colored by either the input graph type or the threshold T . The presence of distinct clusters or 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.

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 4 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 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 5 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.

3.6 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 6, 8, and 9 together provide

526 a comprehensive view of how the number of experts (3, 5, 7, 9) influences the generative model’s
 527 ability to approximate the true solution distribution and optimize budget outcomes.

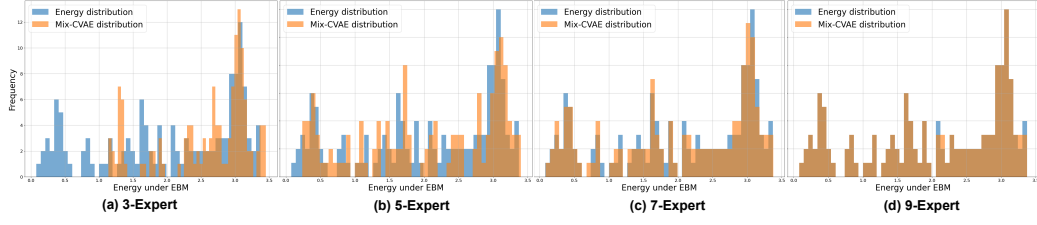


Figure 6: Comparison between the distribution from the EBM and the distribution from the Mix-CVAE with varying numbers of experts on a *synthetic network* with maximum density. Increasing the number of experts improves mode coverage and alignment with the target EBM distribution.

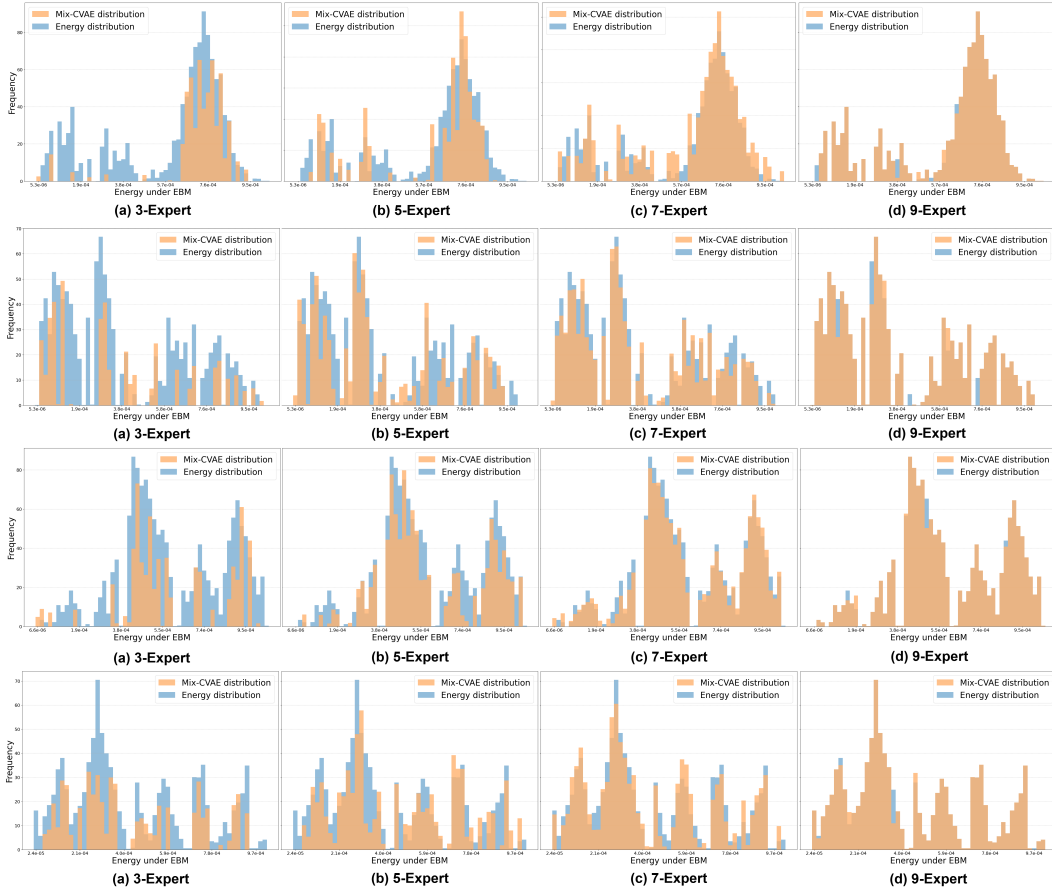


Figure 7: Impact of adding experts on *real-world networks*. From top to bottom: results on Email, Gnutella, RoadCA, and Skitter networks.

528 **Impact of Expert Addition on Generative Modeling.** Figures 6 and 7 compare the energy distribu-
 529 tions of real solutions (as evaluated under the EBM) and fake samples generated by the Mix-CVAE
 530 across both synthetic and real network settings. When only 3 experts are used, the generated samples
 531 show poor alignment with the real data distribution, evidenced by noticeable discrepancies between
 532 the orange (Mix-CVAE) and blue (EBM) bars. As the number of experts increases, this alignment
 533 improves significantly—indicating that the model is better able to capture the underlying modes
 534 and structure of the target distribution. This validates the core motivation behind expert addition:
 535 increasing the number of experts enhances the model’s capacity to represent multimodal solution
 536 spaces. Furthermore, Figure 8 provides additional insight by visualizing the latent space of the
 537 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 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.

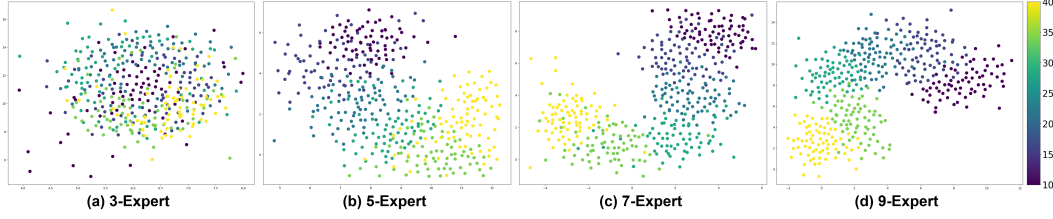


Figure 8: An example of conditional latent space visualization for the trained Mix-CVAE on the same synthetic graph and pairs but different thresholds. Points represent latent vectors \mathbf{z} , colored by threshold T . Clear clustering shows the latent space captures meaningful patterns.

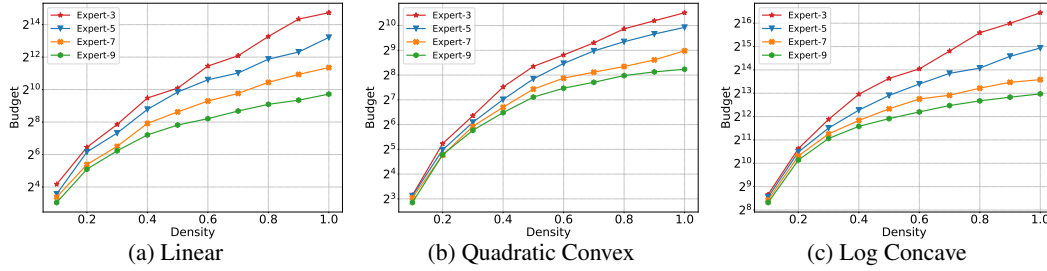


Figure 9: Impact of adding of experts (3, 5, 7, and 9) on the resulting total budget across varying graph densities, testing on the synthetic graph under (a) *Linear*, (b) *Quadratic Convex*, and (c) *Log Concave* edge weight functions.

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 9 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 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.

3.7 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 3 evaluates the Quadratic Convex setting, where each edge weight follows the form $f_e(x_e) = \aleph(x_e^2)$. 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 4 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 4.

Method	Email				Gnutella				RoadCA				Skitter			
	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 3: Performance of HEPHAESTUS and baselines on four real datasets at different thresholds T . The best is highlighted in bold excluding exact solution. (Convex)

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 4: 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 Quadratic Convex f_e : In Table 3 (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.

Performance under Log Concave f_e : Table 4 (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.

3.8 Soundness of the Reward Function

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

$$\mathcal{R}(\mathbf{x}_{i+1}) = \underbrace{F(G, \mathcal{K}, \hat{\mathbf{x}}_{i+1})}_{\text{(i) Smooth feasibility term}} - \underbrace{\kappa \cdot \log(1 + \|\bar{\mathbf{x}}_{i+1}\|_1)}_{\text{(ii) Soft cost penalty term}} \quad (41)$$

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

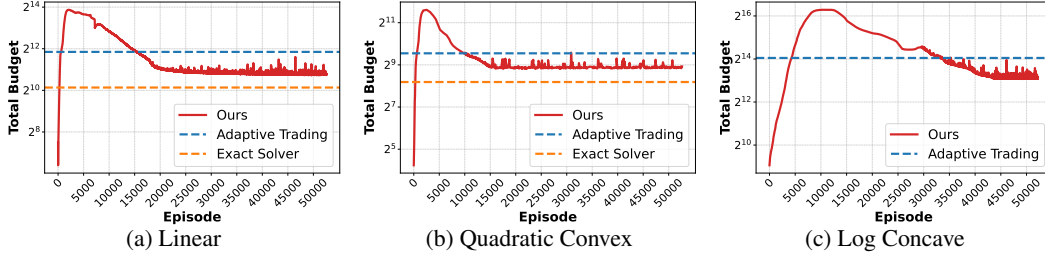


Figure 10: Evolution of Total Budget during the Refine phase (RL agent training)

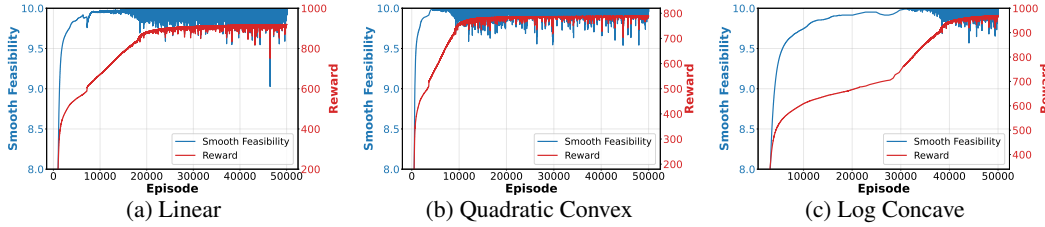


Figure 11: Evolution of Feasible Solution quality (blue, left axis) and Agent Reward (red, right axis) over training episodes.

604 In Figure 10, we show how the total budget evolves over training episodes across three different edge
605 weight settings: Linear, Quadratic Convex, and Log-Concave. A noteworthy observation across all
606 plots is that the budget initially increases sharply during the early training phase, especially visible
607 in episodes 0–3000. This behavior can be explained as: at this stage, the RL agent has not yet
608 learned effective policies and instead prioritizes feasibility, attempting to push more critical paths
609 above the required threshold T , often by over-allocating budget. This is aligned with the feasibility
610 curve in Figure 11 (a–c), where the smooth feasibility skyrockets during the same initial phase,
611 confirming that the agent is successfully raising path costs. As training continues and the feasibility
612 nears saturation (close to a value of 10, meaning that the total cost of the shortest paths for all 10
613 target pairs exceeds the threshold T), we begin to observe a transition: the agent shifts focus toward
614 reducing the overall budget while preserving feasibility. This can be seen in Figure 10(a), where from
615 episode 3000 onwards, the total budget starts to steadily decline. Correspondingly, Figure 11(a) shows
616 that feasibility remains stable, indicating that the agent is successfully optimizing the second term
617 of the reward without sacrificing the first. However, a particularly interesting phenomenon occurs
618 around episode 7000. As the agent becomes increasingly focused on minimizing the cost penalty, it
619 temporarily neglects feasibility. This is reflected in Figure 11(a) as a sharp dip in smooth feasibility
620 around that episode, suggesting some paths have dropped below the threshold. Simultaneously, Figure
621 10(a) shows a sudden plunge in total budget, confirming that the policy has aggressively reduced edge
622 weights to cut cost. Eventually, the agent corrects this behavior. Realizing that dropping feasibility
623 lowers the total reward, the policy re-adjusts — it selectively increases the budget on critical edges to
624 recover feasibility, while continuing to reduce less impactful edges. This adaptive dynamic is exactly
625 what the reward function is designed to elicit: the policy explores aggressively, corrects when needed,
626 and eventually converges to a high-feasibility, low-cost solution.

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

634 3.9 Relative optimal gap convergence

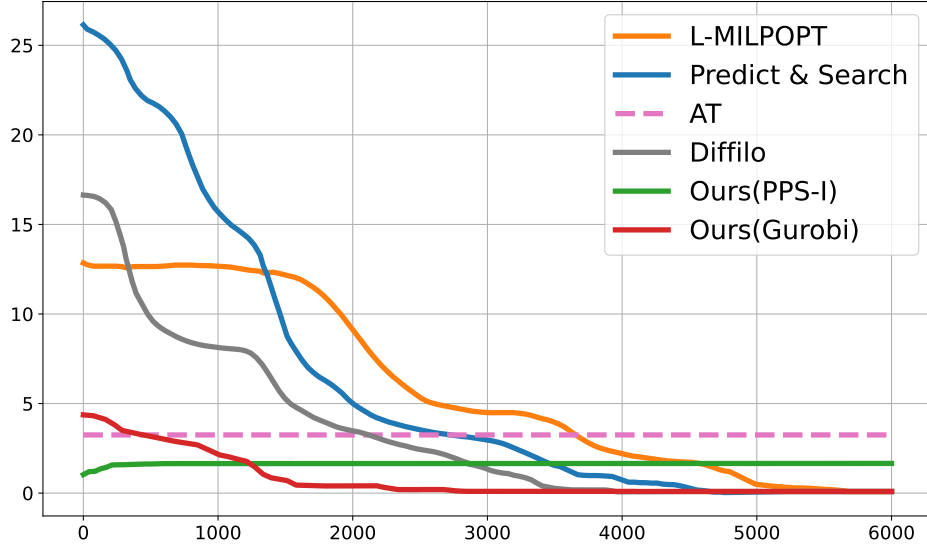


Figure 12: Convergence comparison of the relative optimal gap across Hephaestus and baseline methods under the Linear cost setting. The relative optimal gap measures the percentage deviation from the best-known (optimal) solution, with lower values indicating better performance. Existing ML-based methods (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.

635 The goal of this experiment is to evaluate how effectively and efficiently each method approaches
 636 the optimal solution in terms of total budget, using relative optimal gap as the key metric. This gap
 637 quantifies the percentage deviation from the best-known solution, where lower values indicate better
 638 performance and zero denotes exact optimality. As observed in the Figure 12, existing learning-based
 639 baselines, such as L-MILPOPT, Predict-and-Search, and DiffILO—require post-processing using an
 640 exact solver like Gurobi to reduce their optimality gap. While this refinement phase eventually helps
 641 those methods converge, it often takes several thousand seconds and incurs substantial computational
 642 cost. In contrast, Hephaestus with Gurobi refinement (red curve) starts with high-quality initial
 643 solutions, thanks to the expressiveness of Mix-CVAE, and rapidly converges to zero gap much
 644 faster. This demonstrates the synergy between generative modeling and reinforcement learning
 645 in guiding the solution space effectively. Meanwhile, Hephaestus with PPS-I (green curve) does
 646 not reach exact optimality, as PPS-I is an approximation algorithm that does not enforce all path
 647 constraints exhaustively. However, it consistently achieves low optimality gaps in a fraction of
 648 the time compared to exact methods, offering an effective trade-off between solution quality and
 649 efficiency. An interesting observation in the curve is the initial rise in total cost during the first
 650 few seconds, after which the value quickly plateaus. This behavior comes from the fact that the
 651 initial solution generated by the RL policy π —may not fully satisfy all feasibility constraints. PPS-I
 652 then incrementally increases edge budgets based on marginal benefit-to-cost analysis, just enough
 653 to achieve feasibility, and terminates once all critical pairs are satisfied. Thus, this experiment
 654 underscores that Hephaestus is capable of adapting to different computational needs. It delivers
 655 fast approximate results with PPS-I, or provably optimal solutions with Gurobi refinement—while
 656 consistently outperforming existing baselines in both convergence quality and speed.

657 3.10 Hephaestus with different feasibility refinement

658 In this section, we want to show performance of Hephaestus under three refinement modes: initial
 659 prediction only (no refinement), refinement via PPS-Inference (PPS-I), and full Gurobi-based refine-
 660 ment. These configurations are compared against classical approximation baselines (AT, IG, SA),

Table 5: 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

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 5, 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 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 DIFFILO, 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.

DISCUSSION

Limitations. While the proposed learning-based framework performs well on QoS 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 (QoS) 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.