SPSCO: A SPECULATIVE SAMPLING APPROACH TO NEURAL COMBINATORIAL OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

An open challenge in neural combinatorial optimization (CO), such as using reinforcement learning (RL) and diffusion models (DMs), is the speed–quality trade-off: sequential RL decoders generalize well but tend to settle for suboptimal tours, while DMs generate high-quality full solutions at the cost of long training and slow iterative sampling. We present **SpSCO**, a new framework inspired by speculative sampling (SpS) for large language models (LLMs) inference. Resembling SpS in LLMs, a light-weight draft model (analogous to the sequential RL decoder in SpSCO) collaborates with a high-capacity target model (analogous to a DM in SpSCO) to achieve fast, robust, and high-quality inference – the target model is triggered only when there is a "cognitive divergence" between the draft and target models or internal uncertainty of the draft model. This SpS strategy allows SpSCO to achieve high solution quality while reducing the computational overhead from DMs. Notably, SpSCO is model-agnostic and can be plug-and-play across various RL and DM backbones. It also shows strong robustness: even with undertrained, suboptimal RL and diffusion backbones, SpSCO achieves state-of-the-art performance on diverse CO instances across various scales while attaining faster inference time on large-scale instances.

1 INTRODUCTION

Combinatorial optimization (CO) problems like the Traveling Salesman Problem (TSP) are a cornerstone of computational and operations research, which aim to find an optimal solution from exponentially growing combinations of candidates (Garey & Johnson, 1979; Papadimitriou & Steiglitz, 1982). While traditional exact solvers like Concorde (Applegate et al., 2006) provide optimal solutions, the exponential complexity of CO problems makes them impractical for large-scale instances. This has spurred the design of heuristic methods and, more recently, learning-based approaches.

Neural solvers have emerged as a promising paradigm, learning heuristics directly from data. The state-of-the-art (SOTA) methods broadly fall into two categories, each with a distinct trade-off. Reinforcement learning (RL) methods (Bello et al., 2016; Kool et al., 2019a) construct solutions sequentially in an autoregressive fashion. They are fast and exhibit strong generalization, but their greedy, step-by-step nature often results in suboptimal solutions, as early mistakes cannot be corrected. To mitigate this limitation, some works augment these RL frameworks with search procedures like Monte Carlo Tree Search (MCTS), but this comes at the cost of substantially increased inference cost (Fu et al., 2021b; Wang et al., 2021). Diffusion models (DMs) (Sohl-Dickstein et al., 2015; Graikos et al., 2022) emerge as the new SOTA neural CO solvers, which learn the global structure of the solution space and generate the entire solution from random noises during inference. While they produce exceptionally high-quality solutions, the iterative denoising process results in a steep computational overhead during inference.

This inherent speed-quality dilemma motivates our work. Our underlying research question is: can we synergistically combine the best of both worlds—the speed of RL models and the quality of DMs—and create a hybrid solver that is both fast and highly accurate?

We introduce **SpSCO** (pronounced as "SPESS-koh", see Figure 1), a *fast*, *robust*, *high-quality*, and *model-agnostic* inference framework for neural CO problems that is inspired by the idea of speculative sampling (SpS) (Chen et al., 2023a; Leviathan et al., 2023; Li et al., 2024; Xu et al., 2024) for large language models (LLMs). SpSCO adaptively couples a sequential RL decoder (analogous to a

light-weight draft model in LLMs SpS) with a conditional DM (analogous to a high-capacity target model in LLMs SpS). The core of our approach is an *a priori* trigger mechanism that intelligently decides *when* to invoke the powerful but expensive DM. At each step of the RL decoding process, this trigger monitors for two states, which we quantify using two complementary signals: (1) the RL policy's high internal uncertainty, measured by its entropy, and (2) the "cognitive divergence" between the RL policy and a global prior derived from the DM, measured by KL divergence.

Only when a critical juncture is detected—high RL uncertainty or significant cognitive divergence—is the DM activated. Upon activation, it performs two tasks: it corrects the immediate next step for the RL model and generates a set of complete, high-quality candidate solutions based on the RL partial one. A final selection stage then chooses the best tour from the RL-DM hybrid solution and all DM proposals. This strategic allocation of computational resources allows SpSCO to harness the DM's strengths precisely when they are most needed, avoiding its overhead in decisions.

Our experiments demonstrate the efficacy and robustness of SpSCO. Even when equipped with under-trained, suboptimal RL and diffusion backbones, SpSCO achieves a state-of-the-art optimality gap of 0.02% with an inference time of approximately 1 second per instance on the standard TSP-100 benchmark. On the larger TSP-500 (TSP-1000) problem, it attains a 3.56% (4.58%) gap equipped with under-trained RL and DM components, outperforming most learning-based baselines in quality while being substantially faster than other diffusion models. These results underscore our central thesis: a principled, divergence-driven coordination of heterogeneous models can achieve top-tier performance and efficiency during test time, offering a more computationally effective path forward than simply scaling up models or relying on exhaustive search. Our code is included in the supplementary material for reproducibility and will be released upon paper acceptance.

2 Related Work

RL Solvers. RL solvers frame CO problems as a sequential decision-making process. The seminal work of (Bello et al., 2016) applied a Pointer Network (Vinyals et al., 2015) with policy gradient methods to construct solutions step-by-step. This was advanced by Kool et al. (2019a) with the Transformer architecture that became a foundational blueprint for many subsequent models like POMO (Kwon et al., 2020), which introduced techniques to leverage multiple optima, and Sym-NCO (Kim et al., 2022), which exploited solution symmetries. NAR4TSP (Xiao et al., 2024) proposed a non-autoregressive RL algorithm, but with lower solution quality than the autoregressive versions. Later studies used RL to learn a generalized policy for certain COPs in graphs (Bengio et al., 2020; Chen et al., 2021; Feng et al., 2025) or proposed more advanced learning paradigms to enhance solution quality, such as searching in a continuous latent space (Chalumeau et al., 2023), optimizing policies based on preferences between solutions (Pan et al., 2025), and framing problems within game-theoretic contexts (Li et al., 2025). Some RL algorithms like Invit (Fang et al., 2024) and UDC (Zheng et al., 2024) are designed for large-scale COPs. While these models are computationally efficient and serve as strong baselines, their myopic, autoregressive construction process poses a fundamental limitation, often trapping them in local optima. SpSCO uses an RL model as its fast, "*draft model*", but crucially seeks to mitigate the limitation with a global-aware component.

Heatmap and Diffusion Models for CO. To overcome the sequential limitations of RL solvers, another line of research has focused on non-autoregressive methods that generate a solution in a single shot, often by producing a "heatmap" of edge or node probabilities (Li et al., 2018; Fu et al., 2021a). Joshi et al. (2019) used Graph Convolutional Networks (GCNs) to predict edge inclusion probabilities for TSP. More recently, diffusion models, with their remarkable success in generative tasks, have been adapted for CO. Graikos et al. (2022) mapped TSP instances to images to be solved by a standard image diffusion model. A more direct approach, DIFUSCO (Sun & Yang, 2023), introduced a graph-based diffusion framework that operates on the problem's native graph structure, casting it as a discrete vector generation task. This paradigm has been explored for TSP (Athaide et al., 2023) and extended to other routing problems like VRP (Chen et al., 2023b), while related score-based generative models have been proposed for a broader class of CO problems (Weinberg & Welling, 2021). These models excel at capturing the global structure of optimal solutions, leading to state-of-the-art quality. However, their reliance on a slow, multi-step iterative sampling process makes them computationally intensive. Our work incorporates a diffusion model as a costly, "*target model*", but mitigates its high inference cost through a sparse and adaptive invocation strategy.

3 PRELIMINARY

SpSCO is built upon two distinct neural solver paradigms: a sequential RL solver and a generative Diffusion Model. We briefly outline the standard inference process for each. The training procedure and details on the two solvers are added in the appendix A.

3.1 INFERENCE WITH RL SOLVERS

An RL solver decomposes the node (or edge, etc. in the CO problem) selection into a sequence and sequentially chooses one based on the trained policy network π_θ. Given the current state in the state space, i.e. $s_k \in \mathcal{S}$, the policy π_θ outputs a probability distribution $\pi_\theta(\cdot|s_k)$ over the action space $\mathcal{A}$. During the inference process, the policy network is queried to determine the next action at each step k until the solution is completed, e.g., when a Hamilton loop is found in a TSP. In a standard greedy decoding setting, the action a with the highest probability in $\pi_\theta(\cdot|s_k)$ is chosen:

$$a_k = \arg\max_{a \in \mathcal{A}_k} \pi_\theta(a|s_k). \tag{1}$$

This greedy decoding generation of solutions is computationally fast, as it requires only N forward passes of the policy network and $\arg\max$ computation, in which N is the length of a solution. To improve the quality or diversity of solutions, different decoding types like sampling and beam search are proposed (Kool et al., 2019b), which introduce more computation burden and increase the solving time. Therefore, SpSCO adopts the greedy action selection in its RL decoding strategy.

3.2 INFERENCE WITH CONDITIONAL DIFFUSION MODELS

In contrast, a diffusion model is a non-autoregressive, generative model that learns to produce a complete solution holistically. For discrete problems like TSP, where a solution can be represented by a binary adjacency matrix $x \in \{0,1\}^{N \times N}$, the model uses a discrete diffusion process. It consists of a denoising network, D_ϕ, trained to reverse a noising process that gradually corrupts the solution by flipping its binary entries. Notably, this process can be conditioned on prior information, such as a given prefix of a tour in a TSP, to steer the generation.

Inference (sampling) starts with a matrix of pure random noise (e.g., a random binary matrix), x_T, and iteratively refines it over T total steps to produce a clean solution, x_0. At each denoising step t (different from the time step k in the episode for RL), the denoising network D_ϕ predicts the original clean solution $\hat{x}_0$ based on the current noisy matrix x_t and a conditioning prefix c. This prediction is then used to sample a slightly less noisy matrix x_{t-1}. The reverse step is generally formulated as:

$$x_{t-1} \sim p_\phi(x_{t-1}|x_t, c), t \in \{1, ..., T\}. \tag{2}$$

This iterative process allows the model to capture complex global dependencies, leading to high-quality results. However, it is computationally expensive, requiring many forward passes through the large denoising network D_ϕ. To accelerate this, we adopt fast sampling strategies like DDIM (Song et al., 2021) to reduce the number of denoising steps while maintaining solution quality.

4 OUR APPROACH: SPSCO

In this section, we detail SpSCO, a framework that adapts the principles of **speculative sampling** (Chen et al., 2023a; Leviathan et al., 2023) to combinatorial optimization. In Section 4.1, we first introduce the motivation from sps and the core "draft-then-verify" concept of SpSCO. The inference loop and designs of the adaptive trigger mechanism are described in Section 4.2 and 4.3. In Section 4.4, we describe the dual-track solution generation strategy initiated by this trigger, explaining how it produces both a corrected hybrid solution and a set of entirely new proposals for final selection.

4.1 MOTIVATION FROM SPS

Foundational work in large language models (LLMs) demonstrated that the high latency of powerful autoregressive models can be amortized by using a smaller, faster "draft model" to generate speculative candidates, which are verified in a single parallel pass by the large "target model" (Leviathan

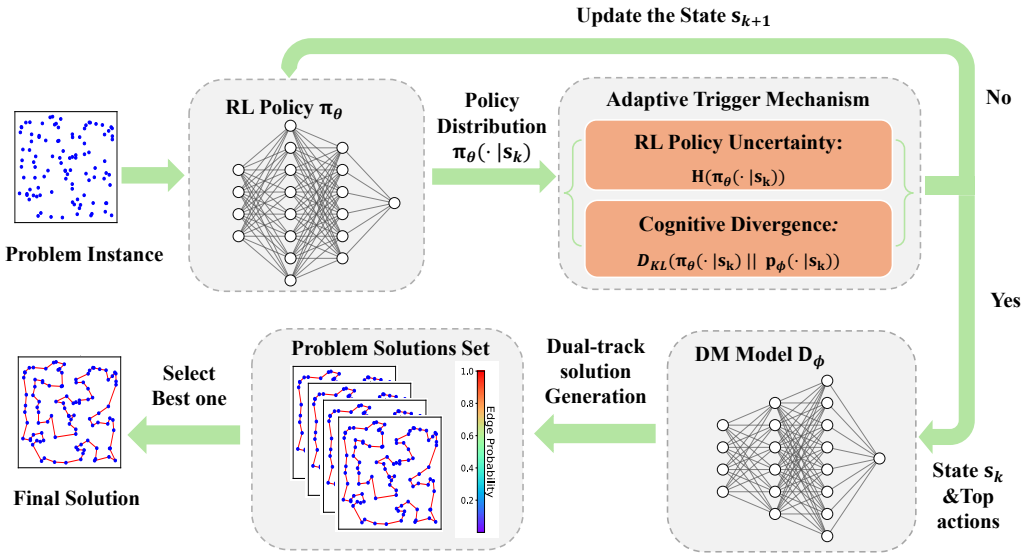

Figure 1: The inference pipeline of SpSCO. At each step, the RL policy (draft model) outputs a probability distribution over the available actions, $\pi_\theta(\cdot|s_k)$. This distribution is then evaluated against two trigger conditions: (1) RL policy's entropy, $H(\pi_\theta(\cdot|s_k))$, exceeds a threshold H_{thresh}, signaling high internal uncertainty; or (2) Cognitive divergence, D_{KL}, between the RL policy and a global prior derived from the DM, denoted as $p_\phi(\cdot|s_k)$, is above its threshold $D_{KL,\text{thresh}}$. Once triggered, DM is invoked. SpSCO begins to operate the dual-track solution generation to collect candidate solutions generated by RL and DM, and returns the best one as the final solution.

et al., 2023). SpSCO is the first framework to successfully translate this "draft-then-verify" paradigm to the structured, sequential decision-making domain of combinatorial optimization. We instantiate this by assigning distinct roles to two complementary solver families. An RL policy serves as the fast draft model, which excels at rapidly constructing a solution by proposing the next action at each step. A DM acts as the powerful target model, possessing a holistic understanding of the global solution space and effectively verifying or correcting the RL model's myopic, step-by-step decisions.

However, the true challenge is not merely to avoid the powerful target model, but to leverage its global, simulative perspective in a computationally feasible manner. To this end, SpSCO repositions the speculative framework as a **high-efficiency simulation engine**. The RL draft model proposes plausible future actions, and the DM target model evaluates them in parallel, providing the deep lookahead needed for high-quality decision-making at a fraction of the cost of traditional methods like Monte Carlo Tree Search. The overall architecture of this process is depicted in Figure 1.

SpSCO constructs a solution sequentially, but unlike a standard RL solver, each step involves a speculative "propose-verify-correct" cycle. Figure 2 provides a conceptual illustration of this process. At each step, the RL draft model proposes its greedy next action (e.g., node 37 in Step 3). Instead of immediately accepting this proposal, SpSCO's trigger mechanism assesses whether the decision is critical enough to warrant verification by the DM target model. If triggered, the DM evaluates the RL's proposal. As shown in Figure 2, the DM may reject the draft (node 37) and provide a superior correction (node 25), thereby steering the solution path away from a locally optimal but globally poor trajectory. This dynamic interplay allows SpSCO to retain the speed of RL for "obvious" decisions while harnessing the wisdom of the DM for critical ones.

4.2 The SpSCO Inference Loop

The SpSCO inference process is an iterative solution construction loop. At each step k, the RL policy generates a probability distribution over available actions ($\pi_\theta(\cdot|s_k)$). Before committing to the greedy choice, the adaptive trigger mechanism (introduced in Section 4.3) is evaluated.

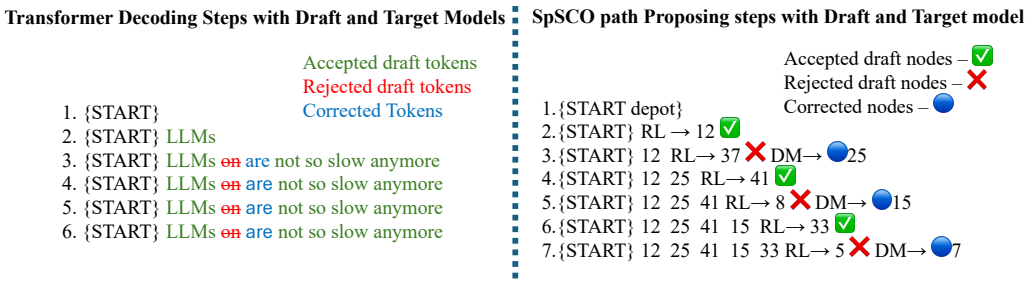

Figure 2: A conceptual illustration of the SpSCO path construction process compared to standard speculative decoding in LLMs. The RL draft model proposes the next node in the tour (e.g., RL→37). The DM target model may either accept this proposal or reject it and provide a globally superior correction (e.g., DM→25), thus correcting the trajectory at critical junctures.

If the trigger conditions are not met, the RL model's locally optimal greedy action is accepted, and the partial solution is extended. However, if the trigger fires, indicating a critical juncture, SpSCO initiates its dual-track path generation strategy (in Section 4.4). This process is governed by a single-trigger design; once the DM is invoked at step k^*, a flag is set, and the RL policy completes the remainder of the hybrid path without further checks. This design ensures a predictable computational budget and maximizes the impact of the single, high-leverage intervention.

4.3 The Adaptive Trigger Mechanism

The core of SpSCO is the adaptive trigger that determines *when* to speculatively invoke the DM. It is designed to activate only at critical junctures where the RL policy's decision is likely to be unreliable. We identify two such states: RL policy is internally uncertain, and its confident decision externally disagrees with the DM's global knowledge. As illustrated in Figure 1, the trigger is based on two complementary signals, and the DM is invoked if either signal surpasses a predefined threshold:

$$H(\pi_\theta(\cdot|s_k)) > H_{\text{thresh}} \quad \vee \quad D_{KL}(\pi_\theta \| p_\phi) > D_{KL,\text{thresh}} \tag{3}$$

The following subsections state the mathematical computation of each signal.

4.3.1 Signal 1: RL Policy Uncertainty

The first signal measures the RL model's internal uncertainty. High uncertainty suggests the RL policy does not have a single, confident choice, making it a prime moment for guidance. We quantify this uncertainty using the standard policy entropy:

$$H(\pi_\theta(\cdot|s_k)) = -\sum_{a \in \mathcal{A}_k} \pi_\theta(a|s_k) \log \pi_\theta(a|s_k) \tag{4}$$

We use policy entropy because it directly quantifies the "flatness" of the action probability distribution: a high entropy value signifies a more uniform distribution where probabilities are spread across many actions—precisely reflecting the model's indecisiveness as no single action is a clear winner.

4.3.2 Signal 2: Cognitive Divergence between RL and DM

The second signal measures the external disagreement, or "cognitive divergence," between the RL policy's choice and the DM's global perspective. This is designed to catch the most dangerous failure mode of a greedy policy: confidently making a locally optimal move that leads to a globally suboptimal solution. Quantifying this requires two steps: efficiently probing the DM to form a prior distribution, and then measuring the divergence between this prior and the RL's own distribution.

Diffusion-Derived Prior via Energy Probing. To obtain the DM's opinion without the cost of a full sampling run, we introduce a lightweight single-step denoising **energy probe**. For a set of Top-M candidate actions $\{a^{(i)}\}_{i=1}^{M}$ proposed by the RL policy, we calculate the "energy" $E_\phi(c_i)$ of each

corresponding prefix. This energy is the masked binary cross-entropy (BCE) loss from a single-step denoising task at a fixed intermediate timestep t_{probe}:

$$E_\phi(c_i) = \mathcal{L}_{\text{BCE}}(\sigma(D_\phi(X_{t_{\text{probe}}}, c_i, t_{\text{probe}})), X_0(c_i))|_{\text{prefix mask}} \tag{5}$$

A lower energy signifies that the prefix is more "plausible" to the DM. We then convert these energy values into a DM-derived prior distribution, $p_\phi(\cdot|s_k)$, using a Boltzmann distribution with an inverse temperature β:

$$p_\phi(a^{(i)}|s_k) = \frac{\exp(-\beta \cdot E_\phi(s_k \oplus a^{(i)}))}{\sum_{j=1}^{M} \exp(-\beta \cdot E_\phi(s_k \oplus a^{(j)}))} \tag{6}$$

Quantifying Divergence with KL. With the RL policy π_θ and the DM prior p_ϕ defined, we quantify their disagreement using the Kullback-Leibler (KL) Divergence. To ensure a valid comparison, $\pi_\theta(\cdot|s_k)$ is first renormalized over the Top-M candidates. The cognitive divergence is then:

$$D_{KL}(\pi_\theta(\cdot|s_k) \| p_\phi(\cdot|s_k)) = \sum_{i=1}^{M} \pi_\theta(a^{(i)}|s_k) \log \frac{\pi_\theta(a^{(i)}|s_k)}{p_\phi(a^{(i)}|s_k)} \tag{7}$$

A high KL divergence value signals that the RL model's confident local choice is strongly opposed by the DM's global perspective.

4.4 Dual-Track Solution Generation and Final Selection

Our SpSCO framework operates on a "dual-track solution generation" strategy when the adaptive trigger fires at a critical step k^*. The strategy initiates two parallel path generations: the hybrid path correction and full DM proposal generation. For the hybrid path correction track, DM is used to *correct the immediate next action* for the hybrid path π_H. Instead of the RL model's greedy choice, we select the action from the candidate pool that is most preferred by the DM's prior. This corrected action, a_{next}, is appended to the tour, and the fast RL policy then autoregressively completes the remaining steps. Simultaneously, in the full DM proposal generation track, DM is used to generate a set of complete, high-quality candidate solutions, $\{\pi_{DM}\}$. This is done by taking the high-probability candidate actions from the RL policy at step k^* and using each one to form a prefix. The conditional DM then generates a full tour for each prefix via its iterative denoising process.

Finally, after both tracks are complete, SpSCO compares the total cost of the hybrid path against the costs of all tours in the proposal set and returns the one with the minimum cost. Algorithm 1 formalizes this entire procedure, summarizing the interplay between the adaptive trigger and the dual-track strategy. It details the step-by-step logic where the trigger is evaluated (Lines 5-11), and depending on the outcome, either a standard RL step is taken (Line 21) or the dual-track generation is initiated (Lines 13-19), culminating in a final selection of the best overall solution (Line 25).

5 Experiments

We evaluate SpSCO on the 2D Euclidean Traveling Salesman Problem (TSP), a standard benchmark for neural combinatorial optimization. Our evaluation is designed to assess SpSCO's solution quality, inference speed, and the effectiveness of its core components across various problem scales.

5.1 Experimental Settings

Datasets. We use standard public datasets for TSP with 50, 100, and 500, 1000 nodes. Problem instances are generated by sampling node coordinates uniformly from the unit square $[0, 1]^2$. We use the Concorde solver (Applegate et al., 2006) to obtain optimal solutions for training our diffusion model and for calculating optimality gaps during evaluation.

Baselines. We compare SpSCO with a comprehensive set of methods: exact solver Concorde, heuristic solver LKH-3 (Helsgaun, 2017), and state-of-the-art learning-based approaches, including RL models (AM (Kool et al., 2019a), POMO (Kwon et al., 2020)), supervised models (GCN (Joshi et al., 2019)), and diffusion-based solvers (DIFUSCO (Sun & Yang, 2023), T2T (Li et al., 2023)).

Algorithm 1 SpSCO: Speculative Sampling for Combinatorial Optimization

Input: TSP instance I, RL policy π_θ, DM denoiser D_ϕ, thresholds $H_{thresh}, D_{KL,thresh}$
Initialize: Hybrid path $\pi_H \leftarrow ()$, DM proposals $\mathcal{P}_{DM} \leftarrow \emptyset$, state s_0, $dm\_triggered \leftarrow$ false
for $k = 0, \ldots, N-1$ **do**
  if tour not complete AND not $dm\_triggered$ **then**
    {*Adaptive Trigger Evaluation*}
    Get policy distribution $\pi_\theta(\cdot|s_k)$ from RL draft model.
    Calculate policy entropy $H_k \leftarrow H(\pi_\theta(\cdot|s_k))$.
    Select Top-M candidate actions $\{a^{(i)}\}_{i=1}^{M}$ from $\pi_\theta(\cdot|s_k)$.
    Calculate denoising energy $E_\phi(s_k \oplus a^{(i)})$ for each candidate via the energy probe.
    Derive DM prior distribution $p_\phi(\cdot|s_k)$ from energies.
    Calculate KL divergence $D_{KL,k} \leftarrow D_{KL}(\pi_\theta||p_\phi)$.
    if $H_k > H_{thresh}$ or $D_{KL,k} > D_{KL,thresh}$ **then**
      {*Trigger Activated: Dual-Track Generation*}
      Select candidate actions $\{a^*\}$ from $\pi_\theta(\cdot|s_k)$ based on a cumulative probability threshold.
      for each selected action a^* **do**
        Form prefix $c^* = s_k \oplus a^*$.
        Generate a full tour proposal $\pi_{DM} \sim$ DM_Sampler(D_ϕ, c^*) and add to $\mathcal{P}_{DM}$.
      end for
      Set $dm\_triggered \leftarrow$ true.
      Let a_{next} be the next action from the best DM proposal found so far.
    else
      {*No Trigger: Standard RL Step*}
      Let $a_{next} \leftarrow \arg\max_a \pi_\theta(a|s_k)$.
    end if
  end if
  Append a_{next} to hybrid path π_H and update state to s_{k+1}.
end for
{*Final Selection*}
Calculate cost $C(\pi_H)$ and all costs for proposals in $\mathcal{P}_{DM}$.
return $\arg\min_{\pi \in \{\pi_H\} \cup \mathcal{P}_{DM}} C(\pi)$

Implementation Details. Our SpSCO framework orchestrates an RL policy and a conditional DM. For RL backbones, we use official pre-trained checkpoints. Our conditional DM, termed **Prefix-Difusco**, adapts the GNN architecture from DIFUSCO and is specifically trained with a masked loss and a curriculum learning strategy to complete tours from given prefixes. To convert the DM's probabilistic heatmap output into a valid tour, we employ a deterministic greedy decoding strategy. Full architectural details, training procedures, and decoding algorithms are provided in Appendix A.

Evaluation Metrics. We report three key metrics: the average tour length, the percentage optimality gap to the exact solver's solutions, and the average inference time per problem instance.

5.2 Main Results

To validate the performance and plug-and-play nature of our model-agnostic framework, we integrated SpSCO with two different RL backbones, Attention Model (AM) and POMO. Besides, the generalizability of SpSCO to other CO problems is supplemented in Appendix B.1.

5.2.1 Performance on TSP-50 and TSP-100

Table 1 presents the performance of SpSCO and baseline methods on TSP-50 and TSP-100 instances. The results are categorized by whether a **2-opt** local search post-processing step is applied. The 2-opt algorithm (Croes, 1958) is a classic and effective local search heuristic for the TSP, which iteratively improves a tour by removing two edges and reconnecting the two resulting paths in the only other possible way to see if the new tour is shorter. It is important to first clarify that the Prefix_Difusco model listed in the table is our custom version of Difusco, which we specifically adapted to handle partial tours (prefixes) as conditions, making it compatible with the SpSCO framework.

Table 1: Results with **Greedy Decoding** on TSP-50 and TSP-100. RL: Reinforcement Learning, SL: Supervised Learning, G: Greedy Decoding. * denotes results that are quoted from previous works.The "Prefix_Difusco (16 sample)" represents the best result obtained by running our Prefix_Difusco model sampling 16 times in parallel.

ALGORITHM	TYPE	TSP-50		TSP-100	
		LENGTH ↓	DROP ↓	LENGTH ↓	DROP ↓
Concorde (Applegate et al., 2006)	Exact	5.69	0.00%	7.76	0.00%
2Opt (Croes, 1958)	Heuristics	5.86	2.95%	8.03	3.54%
AM* Kool et al. (2019a)	RL+G	5.80	1.76%	8.12	4.53%
GCN* Joshi et al. (2019)	SL+G	5.87	3.10%	8.41	8.38%
Transformer* Bresson & Laurent (2021)	RL+G	5.71	0.31%	7.88	1.42%
POMO* Kwon et al. (2020)	RL+G	5.73	0.64%	7.84	1.07%
Sym-NCO* Kim et al. (2022)	RL+G	5.73	0.64%	7.84	0.94%
Image Diffusion* Graikos et al. (2022)	SL+G	5.76	1.23%	7.92	2.11%
DIFUSCO (T_s=50) Sun & Yang (2023)	SL+G	5.71	0.45%	7.85	1.21%
DIFUSCO (T_s=100)	SL+G	5.71	0.41%	7.84	1.16%
T2T (T_s=50,T_t=15) Li et al. (2023)	SL+G	5.69	0.07%	7.77	0.20%
Prefix_Difusco (16 sample)	SL+G	5.69	0.04%	7.76	0.04%
Prefix_Difusco (Used in SpSCO)	SL+G	5.70	0.27%	7.81	0.62%
SpSCO (AM + DF)	**RL+SL+G**	5.69	**0.03%**	7.76	**0.02%**
SpSCO (POMO + DF)	**RL+SL+G**	5.69	**0.03%**	7.76	**0.02%**
AM	RL+G+2OPT	5.77	1.41%	8.02	3.32%
GCN	SL+G+2OPT	5.77	1.40%	8.01	3.21%
Transformer	RL+G+2OPT	5.70	1.06%	7.96	1.89%
POMO	SL+G+2OPT	5.73	0.63%	7.91	1.62%
Sym-NCO	SL+G+2OPT	5.73	0.64%	7.90	0.76%
DIFUSCO	SL+G+2OPT	5.69	0.09%	7.78	0.22%
T2T	SL+G+2OPT	5.69	**0.02%**	7.76	0.06%
SpSCO	**RL+SL+G+2OPT**	5.69	0.03%	7.76	**0.01%**

5.2.2 Performance on Large-Scale TSP

SpSCO's superior speed-quality trade-off becomes even more pronounced on large-scale problems, as shown in Table 2. On TSP-500, using a DM-undertrained that was only briefly fine-tuned for 20 epochs, SpSCO achieves a **3.56%** optimality gap in just **1.20 mins**. This result is not only significantly better in quality than other RL-based methods and standalone DM solvers like T2T (5.09%), but also faster than high-performance competitors like DIFUSCO (5.70m) and T2T (4.90m). This good performance scales to even larger instances. On TSP-1000, even when pairing an RL model directly from the TSP-500 checkpoint with another similarly undertrained DM, SpSCO delivers a state-of-the-art optimality gap of **4.58%** among learning-based methods. It achieves this result in just **2.43 mins**, over 6 times faster than T2T. This efficiency stems directly from our core design: the fast RL model handles the majority of decisions, while the computationally intensive DM is invoked only for a strategic, high-impact correction. This allows SpSCO to scale effectively, delivering top-tier solutions without the prohibitive runtime of exhaustive search or full iterative generation.

5.3 Ablation Study

We conduct ablation studies on TSP-100 to validate SpSCO's core components and design choices.

Importance of the Trigger Mechanism. As shown in Table 3, our dual-signal trigger is essential. Relying solely on either the entropy or the KL divergence trigger leads to significantly worse optimality gaps (0.05% and 0.09%, respectively). This confirms that both internal policy uncertainty (entropy) and external disagreement with the DM (KL divergence) are complementary and necessary signals for making effective intervention decisions.

Table 2: Results on large-scale TSP problems. RL, SL, AS, and G denote Reinforcement Learning, Supervised Learning, Active Search, and Greedy decoding, respectively. Len means the average tour length. * indicates the baseline for computing the performance gap.The TIME column reports the average inference time per instance.

ALGORITHM	TYPE	TSP-500			TSP-1000		
		LENGTH↓	DROP↓	TIME	LENGTH↓	DROP↓	TIME
Concorde	Exact	16.55*	0.00%	37.66m	23.53*	0.00%	6.65h
Gurobi	Exact	16.55	0.00%	45.63h	23.53	0.00%	48h
LKH-3 (default)	Heuristics	16.55	0.00%	46.28m	23.53	0.00%	2.57h
AM	RL+G	20.02	20.99%	1.51m	28.52	21.21%	3.18m
GCN	SL+G	29.72	79.61%	6.67m	43.15	83.38%	28.52m
POMO+AS-Emb	RL+AS+G	19.24	16.25%	12.80h	-	-	-
POMO+AS-Tab	RL+AS+G	24.54	48.22%	11.61h	-	-	-
DIMES	RL+G	18.93	14.38%	0.97m	27.23	15.73%	2.08m
DIMES	RL+AS+G	17.81	7.61%	2.10h	25.11	6.72%	4.49h
DIFUSCO	SL+G	18.11	9.41%	5.70m	25.68	9.14%	11.5m
T2T	SL+G	17.39	5.09%	4.90m	25.17	8.87%	15.66m
Prefix_DIFUSCO	SL+G	17.92	8.23%	**0.50m**	25.86	9.91%	**0.38m**
SpSCO	**RL+SL+G**	**17.24**	**3.56%**	1.20m	**24.61**	**4.58%**	2.43m

Table 3: Ablation study on the core components of SpSCO, Trigger Mechanism, evaluated on the TSP-100 dataset. Gap (%) is the optimality gap compared to the Concorde solver. Time (s) is the average inference time per instance.

Model / Method	Description	Gap (%) ↓
Entropy-Only	Policy entropy trigger	0.05%
KL-Only	KL divergence trigger	0.09%
Full Model	**Full Model**	**0.02%**

Trigger Behavior and Sensitivity. Our analysis reveals a clear trade-off landscape for the trigger thresholds. The entropy threshold directly balances solution quality and inference time (Table 12 in Appendix B), while a stricter (lower) KL divergence threshold proves superior for performance (Table 13 in Appendix B). Notably, with our default thresholds, the trigger fires in 100% of instances at a very early average step of 0.52. This supports our rationale of "strategic early intervention": SpSCO identifies the initial steps as the most critical, correcting the RL model's trajectory before errors can propagate. The full analysis, including sensitivity to candidate exploration strategies and the robustness of the energy probe, is detailed in Appendix B.

6 Conclusion

We introduced SpSCO, a novel speculative sampling framework that synergistically combines a fast, sequential RL model with a high-quality, nonautoregressive diffusion model for solving combinatorial optimization problems. The core of our contribution is a lightweight trigger mechanism that uses policy entropy and KL divergence to adaptively invoke the diffusion model at critical decision points. This "cognitive divergence" metric, calculated via an efficient single-step energy probe, effectively identifies when the local, greedy decisions of the RL model begin to deviate from the global optimum manifold learned by the diffusion model. Our results on standard TSP benchmarks demonstrate that SpSCO achieves state-of-the-art solution quality while being more computationally efficient than other high-performance methods. This work opens up a promising new direction in neural CO solvers: better neural CO solvers may lie not just in developing larger or more complex models, but in the principled and intelligent hybridization of diverse, complementary approaches.

REFERENCES

David Applegate, Robert Bixby, Vasek Chvatal, and William Cook. Concorde tsp solver, 2006.

Roch G. Athaide, K. S. Sesh Kumar, and S. Anil Kumar. A denoising diffusion probabilistic model for the travelling salesman problem. *arXiv preprint arXiv:2302.06219*, 2023.

Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 2020.

Federico Berto, Chuanbo Hua, Junyoung Park, Laurin Luttmann, Yining Ma, Fanchen Bu, Jiarui Wang, Haoran Ye, Minsu Kim, Sanghyeok Choi, et al. Rl4co: an extensive reinforcement learning for combinatorial optimization benchmark. *arXiv preprint arXiv:2306.17100*, 2023.

Xavier Bresson and Thomas Laurent. The transformer network for the traveling salesman problem, 2021.

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

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023a.

Haipeng Chen, Wei Qiu, Han-Ching Ou, Bo An, and Milind Tambe. Contingency-aware influence maximization: A reinforcement learning approach. In *UAI*, pp. 1535–1545, 2021.

Zihan Chen, Tianyu Wang, Yutong Wang, Ming-Xuan Wu, Wentao Wang, and Gaoang Wang. GPS-VRP: A graph-based partition-and-search framework for vehicle routing problems with diffusion model. *arXiv preprint arXiv:2309.16016*, 2023b.

G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958.

Han Fang, Zhihao Song, Paul Weng, and Yutong Ban. Invit: a generalizable routing problem solver with invariant nested view transformer. In *ICML*, pp. 12973–12992, 2024.

Xinsong Feng, Zihan Yu, Yanhai Xiong, and Haipeng Chen. Sequential stochastic combinatorial optimization using hierarchal reinforcement learning. In *International Conference on Learning Representations*, 2025.

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

Zhiqing Fu, Zhaoxing Zhang, Zhaokun Wang, and Han Sun. Neural mcts: A learning-based solution for combinatorial optimization on graphs. In *International Conference on Learning Representations*, 2021b.

Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, San Francisco, CA, 1979.

Alexandros Graikos, Nikolay Malkin, Nebojsa Jojic, and Dimitris Samaras. Diffusion models as plug-and-play priors. In *Advances in Neural Information Processing Systems*, 2022.

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

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

Minsu Kim, Jinkyoo Park, et al. Sym-NCO: Leveraging symmetricity for neural combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

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

Wouter Kool, Herke Van Hoof, and Max Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *ICML*, pp. 3499–3508, 2019b.

Yeong-Dae Kwon, Juhan Choo, Bumm Kim, Iljoo Yoon, Young-Joon Gwon, and Seung-won Min. POMO: Policy optimization with multiple optima for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 21188–21198, 2020.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286, 2023.

Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. T2t: From distribution learning in training to gradient search in testing for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:50020–50040, 2023.

Yuheng Li, Panpan Wang, and Haipeng Chen. Can reinforcement learning solve asymmetric combinatorial-continuous zero-sum games? *International Conference on Learning Representations*, 2025.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. In *International Conference on Machine Learning*, pp. 28935–28948, 2024.

Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, pp. 562–572, 2018.

Mingjun Pan, Guanquan Lin, You-Wei Luo, Bin Zhu, Zhien Dai, Lijun Sun, and Chun Yuan. Preference optimization for combinatorial optimization problems. *International Conference on Machine Learning*, 2025.

Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ, 1982.

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

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

Zhiqing Sun and Yiming Yang. DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

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

Qi Wang, Yongsheng Hao, and Jie Cao. Learning to traverse over graphs with a monte carlo tree search-based self-play framework. *Engineering Applications of Artificial Intelligence*, 105: 104422, 2021.

Ben Weinberg and J. M. K. Welling. Score-based generative models for np-hard combinatorial optimization. *arXiv preprint arXiv:2106.05831*, 2021.

Yubin Xiao, Di Wang, Boyang Li, Huanhuan Chen, Wei Pang, Xuan Wu, Hao Li, Dong Xu, Yanchun Liang, and You Zhou. Reinforcement learning-based nonautoregressive solver for traveling salesman problems. *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

Han Xu, Jingyang Ye, Yutong Li, and Haipeng Chen. Can speculative sampling accelerate react without compromising reasoning quality? In *The Second Tiny Papers Track at International Conference on Learning Representations 2024*, 2024.

Zhi Zheng, Changliang Zhou, Tong Xialiang, Mingxuan Yuan, and Zhenkun Wang. Udc: A unified neural divide-and-conquer framework for large-scale combinatorial optimization problems. *Neurips*, 37:6081–6125, 2024.

A IMPLEMENTATION DETAILS OF PREFIX_DIFUSCO AND RL MODELS

Here we provide detailed specifications for our backbone conditional diffusion model, Prefix_Difusco, used in the experiments. About the RL model, we uniformly use the code and environment provided by the RL4CO Berto et al. (2023) package for training [1]. The detailed hyperparameters are listed in A.4.

A.1 MODEL ARCHITECTURE

The core of Prefix_Difusco modifies the GNN architecture from DIFUSCOSun & Yang (2023) for the conditional generation task. The key components are:

- **Node Feature Embedding**: Node coordinates are first converted into high-dimensional features using a sinusoidal positional embedding. A binary feature indicating whether a node is part of the prefix is concatenated to this embedding. A final linear layer projects this combined feature vector to the GNN's expected input dimension ($d_{node} = 128$).
- **PrefixEncoder**: An LSTM-based encoder takes the sequence of node features corresponding to the prefix tour and outputs a single global conditioning vector ($d_{cond} = 256$). This vector summarizes the properties of the given partial tour.
- **DifuscoGNNEncoder**: This is the main denoising network. It is a 12-layer GNN that processes a graph where nodes have the features described above. At each layer, the GNN's message passing is conditioned by both the global prefix vector from the `Prefix_Encoder` and a sinusoidal embedding of the current timestep t.
- **Output Head**: The GNN outputs a logit for each potential edge in the graph, representing the probability of that edge being part of the optimal tour.

A.2 TRAINING PROCEDURE

The model is trained to predict the ground-truth adjacency matrix x_0 from a noised version x_t and a conditional prefix.

- **Dataset and Conditioning**: The training dataset consists of TSP instances and their optimal tours. For each sample, we derive a training instance by randomly selecting a prefix of length k from the optimal tour.
- **Diffusion Process**: We use a discrete diffusion process over $T = 1000$ steps with a cosine noise schedule to corrupt the ground-truth adjacency matrix x_0 into a noisy matrix x_t.
- **Masked Loss Function**: The model's objective is to minimize the Binary Cross-Entropy (BCE) loss between the predicted adjacency matrix and the ground truth x_0. Crucially, the loss is only computed on the "suffix" edges—that is, all edges except those whose both endpoints are within the given prefix. This forces the model to learn how to best complete the tour.
- **Curriculum Learning**: To improve convergence and performance, we employ a multi-stage curriculum. The training starts with a distribution of long prefixes (e.g., $k \in [60, 90]$), making the completion task easier. As training progresses, the distribution of k shifts towards shorter, more difficult prefixes (e.g., $k \in [1, 30]$), allowing the model to gradually master the full conditional generation task.

A.2.1 TRAINING FOR TSP-50

The `Prefix_Difusco` model for TSP-50 was trained from scratch. We employed a 5-stage curriculum learning strategy designed to gradually increase the task difficulty. Each stage was trained for 10 epochs, initializing from the best checkpoint of the previous stage.

- **Stage 1 (Easy):** Trained on long prefixes with lengths $k \in [30, 49]$.
- **Stage 2 (Medium):** Trained on prefix lengths $k \in [10, 30]$.

[1] https://github.com/ai4co/rl4co

- **Stage 3 (Hard):** Trained on the full range of prefix lengths $k \in [1, 49]$.
- **Stage 4 (Short Focus):** Focused on short prefixes with lengths $k \in [1, 20]$.
- **Stage 5 (Very Short Focus):** Further focused on very short prefixes with $k \in [1, 10]$ to enhance performance on early-step decisions.

A.2.2 TRAINING FOR TSP-100

The model for TSP-100 was also trained from scratch following a similar multi-stage curriculum learning approach. The prefix length ranges for each stage were adjusted proportionally for the larger problem size to ensure a smooth learning progression from easy to hard completion tasks. The core hyperparameters, such as hidden dimensions and learning rate, were kept consistent with the TSP-50 model.

- **Stage 1 (Easy):** Trained on long prefixes with lengths $k \in [61, 99]$.
- **Stage 2 (Medium):** Trained on prefix lengths $k \in [30, 60]$.
- **Stage 3 (Hard):** Trained on the full range of prefix lengths $k \in [1, 99]$.
- **Stage 4 (Short Focus):** Focused on short prefixes with lengths $k \in [1, 20]$.
- **Stage 5 (Very Short Focus):** Further focused on very short prefixes with $k \in [1, 10]$ to enhance performance on early-step decisions.

A.2.3 TRAINING FOR TSP-500

Due to the significantly larger scale of TSP-500, we adopted a more advanced training strategy combining **transfer learning** and a tailored curriculum.

- **Transfer Learning**: The TSP-500 model was not trained from scratch. Instead, it was initialized using the weights from our best-trained TSP-100 checkpoint. Weights were transferred for all layers with matching names and shapes (e.g., GNN layers, prefix encoder), providing a strong starting point and accelerating convergence.
- **Curriculum Learning**: After initialization, the model was fine-tuned on TSP-500 data using a 5-stage curriculum similar to the one for TSP-50, but with ranges adjusted for $N = 500$ (e.g., Stage 1: $k \in [50, 100]$, Stage 2: $k \in [20, 50]$, etc.). For the results reported in this paper, we ran an accelerated training schedule of approximately 20 epochs in total, focusing on the most critical curriculum stages to balance performance and computational cost.

A.2.4 TRAINING FOR TSP-1000

For the largest scale, TSP-1000, we continued the strategy of combining transfer learning with a specialized curriculum to manage the increased complexity and computational demands.

- **Transfer Learning**: The TSP-1000 model was initialized with the weights from our best-performing TSP-100 checkpoint. This transfer learning approach provided a robust feature foundation, significantly accelerating the training convergence on the larger graph size.
- **Curriculum Learning**: Following initialization, the model was fine-tuned on the TSP-1000 dataset using a 4-stage curriculum over a total of 25 epochs. The training began with easier tasks (completing tours from long prefixes, with k up to 500) and progressively moved to more difficult scenarios, focusing on shorter prefixes (k down to 1) in later stages to refine the model's ability to make critical early decisions.

A.3 DECODING ALGORITHMS

To convert the probabilistic heatmap output from our **`Prefix_Difusco`** model into a valid TSP tour, we employ a deterministic greedy decoding strategy inspired by DIFUSCO (Sun & Yang, 2023). This approach ensures that given a heatmap, the resulting tour is always the same, which

is crucial for the stability of the SpSCO framework. The core decoding process follows a principled, multi-stage procedure designed to construct high-quality tours while respecting the prefix constraints.

The decoding algorithm proceeds as follows:

- **Edge Score Calculation:** The raw adjacency probability matrix P from the diffusion model is first symmetrized to ensure consistency ($P' = (P + P^T)/2$). To favor shorter edges, which are fundamental to good TSP solutions, we compute an edge score for each potential edge (i, j) by dividing its symmetrized probability by its Euclidean distance: $S_{ij} = P'_{ij}/\text{dist}(i, j)$. All possible edges are then sorted in descending order based on these scores.
- **Enforce Prefix Constraint:** Before any greedy selection, the decoder first enforces the given conditional prefix. All edges that form the given partial tour are mandatorily included in the solution set. A Union-Find data structure is initialized, and the degrees of the prefix nodes are updated accordingly to ensure these edges are fixed.
- **Greedy Spanning Path Construction:** The algorithm iterates through the globally sorted list of edges. For each candidate edge, it performs three checks:
  1. It is not an existing prefix edge.
  2. Adding the edge will not result in any node having a degree greater than two.
  3. Adding the edge will not form a premature cycle (verified using the Union-Find data structure).

  If all conditions are met, the edge is added to the solution set, and the node degrees and Union-Find structure are updated. This process continues until a total of $N - 1$ edges have been selected, forming a spanning path of all nodes.
- **Tour Finalization:** Once a spanning path of $N - 1$ edges is formed, there will be exactly two nodes with a degree of one (the endpoints of the path). The final edge connecting these two endpoints is deterministically added to close the path and form a valid Hamiltonian cycle. The final list of N edges is then converted into a sequential tour starting from the first node of the original prefix (or node 0 if no prefix was given).

A.4 HYPERPARAMETERS

This subsection provides a comprehensive summary of the hyperparameters for the core models employed in the SpSCO framework: The configurations for our conditional diffusion model (`Prefix_Difusco`), the Attention Model (AM), and POMO Kool et al. (2019a); Kwon et al. (2020) are detailed in Table 4, Table 5, and Table 6, respectively. These tables are intended to ensure full reproducibility of our experimental results.

Table 4: Hyperparameters for Prefix_Difusco across different problem sizes.

Parameter	TSP-50	TSP-100	TSP-500	TSP-1000
Model Architecture				
Node Count (N)	50	100	500	1000
sparse factor(K)	N.A	N.A	N.A	100
GNN Layers (L)	12	12	12	12
Hidden Dimension	256	256	256	256
Node Embedding Dim	128	128	128	128
Prefix Condition Dim	256	256	256	256
Diffusion Process				
Timesteps (T)	1000	1000	1000	1000
Beta Schedule	cosine	cosine	cosine	cosine
Inference Steps	10	10	50	50
Inference Sampler	DDIM	DDIM	DDIM	DDIM
Training				
Batch Size (per GPU)	128	96	4	8
Epoch	50	50	20	25
Training data (per epoch)	1500000	1500000	128000	65000
Learning Rate	2e-4	2e-4	2e-5	1e-4
Optimizer	Adam	Adam	Adam	Adam
Training Method	Curriculum	Curriculum	Transfer&Curriculum	Transfer&Curriculum
Environment	2x NVIDIA A40 GPUs			

Table 5: Hyperparameters for Attention Model across different problem sizes. Need to notice, AM and POMO models for TSP-500 are trained on the tsp-200 instances and then generalized into the TSP 500.

Parameter	TSP-50	TSP-100	TSP-500
Model Architecture			
GNN Layers (L)	3	3	3
Hidden Dimension	512	512	512
Node Embedding Dim	128	128	128
Attention heads	8	8	8
Baseline	rollout	rollout	critic
Training			
Batch Size (per GPU)	512	512	1024
Training data (each epoch)	1280000	1280000	1280000
Epoch	100	100	120
Learning Rate	1e-4	1e-4	1e-4
LR Scheduler	multistep LR, gamma=0.1, milestone = [80, 95]		
Normalization	batch		
Environment	1x NVIDIA A40 GPUs		

Table 6: Hyperparameters for POMO across different problem sizes.

Parameter	TSP-50	TSP-100	TSP-500
Model Architecture			
GNN Layers (L)	6	6	6
Hidden Dimension	512	512	512
Node Embedding Dim	128	128	128
Attention heads	8	8	8
Augment	8	8	8
Training			
Batch Size (per GPU)	512	512	256
Training data (each epoch)	100000	100000	100000
Epoch	400	800	400
Learning Rate	1e-4	1e-4	1e-4
LR Scheduler	multistep LR, gamma=0.1, milestone = [80, 95]		
Normalization	instance		
Environment	1x NVIDIA A40 GPUs		2x NVIDIA A40 GPUs

B Additional experiment and Ablation Study Results

B.1 Generalizability to Other Combinatorial Optimization Problems

While the main body of this paper focuses on the Traveling Salesman Problem (TSP), the SpSCO framework is designed to be general. To demonstrate its potential on other tasks, we conducted a preliminary evaluation on the Orienteering Problem (OP).

Table 7 summarizes the performance on the OP-100 benchmark. The results highlight the synergy of our hybrid approach. While the standalone RL (AM) and Diffusion (prefix_difusco) backbones produce solutions with a significant 8-9% optimality gap, SpSCO successfully combines their strengths to cut this gap nearly in half, achieving a much improved **4.657%** gap.

Crucially, this substantial gain in solution quality is achieved with a minimal increase in computational cost, keeping the inference time under one second. This successful application to a different, complex routing problem validates the plug-and-play nature of our framework and its ability to create a superior solver from weaker components, showcasing its potential for broader use across various combinatorial optimization problems.

Table 7: Performance comparison on the OP-100 benchmark. AVG Score denotes the average collected prize (higher is better). Gap indicates the percentage deviation from the optimal solution (lower is better). Time is the average inference time in seconds.

Method	AVG Score ↑	Gap ↓	Time (s) ↓
Gurobi-30	31.13	0%	30
Gurobi-300	32.10	-3.115%	300
AM (RL)	28.37	8.866%	**0.3**
prefix_difusco	28.28	9.317%	0.662
SpSCO (Ours)	**29.68**	**4.657%**	0.946

B.2 Ablation Study

Therefore, SpSCO provides a flexible blueprint for creating next-generation hybrid neural solvers. By simply swapping the problem-specific RL and DM backbones, our framework can be readily extended to tackle a diverse set of complex optimization challenges, effectively navigating the

speed-quality trade-off across various domains. This section provides detailed results and analysis for the ablation studies presented in the main paper, all conducted on the TSP-100 benchmark. These studies are designed to be self-contained, offering a deeper understanding of SpSCO's internal mechanisms and validating our design choices. Echoing the central thesis from our introduction, the following experiments empirically demonstrate how the principled, divergence-driven coordination of the RL and DM models allows SpSCO to effectively navigate the speed-quality trade-off. We specifically analyze the necessity of our dual-criteria trigger (the heart of our "cognitive divergence" measure), the critical balance between exploration breadth and computational efficiency, and the overall robustness of the framework's components. To maintain consistency with the main text, table numbering in this appendix begins at 6.

Table 8 examines the sensitivity of our framework to the DM energy probe's timestep (`dm_probe_timestep`). The results demonstrate remarkable robustness: across a wide range of timesteps from 100 to 900, the optimality gap remains stable at 0.02%. This is a significant practical advantage, as it indicates that the single-step energy probe is a reliable signal that does not require meticulous hyperparameter tuning.

Table 8: Sensitivity analysis on the DM energy probe timestep, `dm_probe_timestep`.

Energy Probe Ts	Gap (%) ↓	Time (s)	Ave. trigger step (Trig. Rates)
100	0.02	2255.95s	0.66 (100.0%)
300	0.02	2262.54s	0.66 (100.0%)
500	0.02	**2238.88s**	0.52 (100.0%)
700	0.02	2246.30s	0.51 (100.0%)
900	0.02	2243.36s	0.51 (100.0%)

Table 9 investigates the impact of the RL candidate pool size (`probe_rl_top_m`) used for the KL divergence calculation. This experiment highlights the importance of providing a sufficiently large set of candidate actions for the probe. With too few candidates (e.g., 5), the KL divergence is not a reliable indicator, resulting in a poor optimality gap (1.30%) and a low trigger rate (43.0%). Our default setting of 15 ensures that the divergence metric is calculated over a meaningful distribution, allowing for effective detection of critical junctures.

Table 9: Sensitivity analysis on the RL candidate pool size, `probe_rl_top_m`, used for KL divergence calculation.

`Probe_RL_Top_M`	Gap (%) ↓	Time (s)	Ave. trigger step (Trig. Rates)
5	1.30	2894.19s	4.05 (43.0%)
10	0.15	1635.85s	5.20 (97.1%)
15	0.02	2238.88s	0.52 (100.0%)
20	0.01	3025.94s	0.00 (100.0%)

Finally, Table 10 analyzes the effect of the candidate selection threshold for DM exploration (`TopN_cum_Th`). This parameter controls the breadth of the DM's search once it is triggered. The results show a clear trade-off: a small threshold (e.g., 0.2) is faster but often fails to find a high-quality solution, yielding a suboptimal 0.12% gap. Increasing the exploration breadth is crucial for capitalizing on the DM's generative power. Our default value of 0.8 allows the DM to explore a diverse set of high-probability candidates, which is vital for discovering the near-optimal paths that lead to our state-of-the-art 0.02% gap.

Table 11 provides a comprehensive analysis of our core trigger mechanism. The results clearly show that the dual-criteria trigger, which combines both policy entropy and KL divergence, is essential for top performance. Relying solely on the KL-divergence or entropy trigger leads to significantly worse optimality gaps (0.09% and 0.05%, respectively). This confirms our hypothesis that policy uncertainty (entropy) and cognitive divergence (KL) are complementary signals. The former identifies when the RL agent is indecisive, while the latter detects when it is confidently wrong. Together, they form a robust and effective condition for invoking the diffusion model.

Table 10: Ablation study on the candidate selection strategy for DM exploration, `n_cumulative_threshold`.

`TopN_cum_Th`	Gap (%) ↓	Time (s)	Ave. trigger step (Trig. Rates)
0.2	0.12	724.87s	0.52 (100.0%)
0.4	0.12	729.66s	0.52 (100.0%)
0.5	0.08	925.61s	0.52 (100.0%)
0.6	0.05	1205.19s	0.52 (100.0%)
0.8	0.02	2238.88s	0.52 (100.0%)

Table 11: Complete Ablation study on the core components of SpSCO, evaluated on the TSP-100 dataset. Gap (%) is the optimality gap compared to the Concorde solver. Time (s) is the average inference time per instance. Avg. Trigger Step Index indicates the average step number in the tour construction at which the Diffusion Model was first invoked. Trigger Rate denotes the percentage of instances where the DM was triggered at least once. The best-performing model is highlighted in bold.

Model / Method	Description: Triggers on	Gap (%) ↓	Time (s)	Ave. trigger step (Trig. Rates)
Trigger Mechanism Ablation				
SpSCO (Entropy)	policy entropy only	0.05	2435.33s	2.36 (99.8%)
SpSCO (KL)	KL divergence only	0.09	1670.84s	1.35 (97.0%)
SpSCO (Full Model)	**Full Model (Entropy + KL)**	**0.02**	2238.88s	0.52 (100.0%)

Table 12: Sensitivity to the Policy Entropy Threshold

Threshold	Gap (%) ↓	Time (s)	Trigger step (Rates)
1.4	**0.01%**	3031.26s	0 (100%)
1.6	0.02%	2238.88s	0.52 (100.0%)
1.8	0.05%	**1367.75s**	1.47 (99.4%)
2.0	0.09%	1508.01s	1.44 (97.7%)

Table 13: Sensitivity to the KL-Divergence Threshold

Threshold	Gap (%) ↓	Time (s)	Trigger step (Rates)
8	**0.02%**	**2238.88s**	0.52 (100.0%)
10	0.02%	2284.97s	0.83 (100.0%)
12	0.03%	2304.25s	1.11 (100.0%)
14	0.03%	2334.44s	1.32 (100.0%)

C STATEMENT ON THE USE OF LARGE LANGUAGE MODELS (LLMS)

During the preparation of this manuscript, we utilized a Large Language Model (LLM) as a general-purpose writing assistance tool. The use of the LLM was strictly limited to improving the quality and clarity of the English prose.

Specifically, the LLM was employed for the following tasks:

- Proofreading to identify and correct typographical errors.
- Correcting grammatical mistakes and ensuring syntactical correctness.
- Rephrasing sentences to improve readability, flow, and conciseness.

The core scientific ideas, theoretical derivations, experimental design, results, and conclusions presented in this paper were conceived and articulated entirely by the human authors. The LLM did not contribute to any aspect of the research ideation or the generation of the scientific content. Its role was exclusively that of a language editing and refinement tool.