SchedDiff: Diffusion-Based Priority Refinement for Job Shop Scheduling

Dong-Yoon Oh, Sang-Hyun Cho, Inguk Choi, Hyun-Jung Kim*
Korea Advanced Institute of Science and Technology (KAIST)
{odong,ie02002,inguk0826,hyunjungkim}@kaist.ac.kr

Abstract

Neural combinatorial optimization methods for the Job Shop Scheduling Problem (JSSP) face a fundamental trade-off between solution quality and inference efficiency: constructive methods build high-quality schedules through sequential decisions whose computational cost scales with problem size, whereas one-shot methods offer highly efficient single-pass inference at the expense of solution quality. We propose SchedDiff, the first diffusion-based framework for JSSP, which bridges this gap by iteratively refining operation priorities through a fixed number of denoising steps. Unlike existing diffusion models for combinatorial optimization that operate in a binary space, SchedDiff operates directly in a discrete ordinal space, introducing a forward process and learning objective specifically tailored to ranking structures. To decode the refined priorities into feasible schedules, we augment standard list scheduling with gap insertion, which provably never yields a longer makespan for the same priority ordering. Extensive experiments on TA, DMU, and LA benchmarks demonstrate that SchedDiff achieves state-of-the-art performance among learning-based methods, with efficiency advantages that become more pronounced on large-scale instances. SchedDiff further exhibits strong generalization, scaling to instances significantly larger than those seen during training, remaining robust under unseen problem distributions, and enabling zero-shot application to flexible JSSP.

1 Introduction

The Job Shop Scheduling Problem (JSSP) is one of the most challenging combinatorial optimization problems with wide-ranging applications in manufacturing [1], logistics [2], and cloud computing [3]. A JSSP requires sequencing operations across machines to minimize predefined objectives, most notably the makespan (i.e., the completion time of the last operation), while respecting precedence and resource constraints. Due to the combinatorial explosion of the solution space, exact methods quickly become intractable as problem sizes grow [4]. While classical approaches such as dispatching rules and heuristic methods have proven effective in practice, they are inherently limited by their reliance on manual design and specific domain knowledge, often struggling to generalize to different problem distributions [5].

Neural Combinatorial Optimization (NCO) has emerged as a promising alternative, aiming to automatically learn effective scheduling policies from data. Among existing approaches, constructive methods have become the most widely adopted paradigm because they can generate high-quality solutions [6, 7]. These methods build schedules sequentially, selecting one operation at a time based on the current partial schedule. However, this autoregressive nature requires N sequential forward passes for N operations. This sequential dependency fundamentally limits scalability, making these methods increasingly impractical as problem sizes grow. To address these scalability challenges,

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

one-shot methods have gained attention as an alternative [8, 9]. These methods predict all operation priorities simultaneously in a single forward pass. The predicted priorities are then directly decoded into a schedule via list scheduling. This single-pass inference offers significant speedups over constructive approaches, and the advantage widens as problem size increases.

However, this efficiency comes with significant drawbacks. The reliance on a single forward pass creates an inherent trade-off between inference speed and solution quality. By predicting all priorities at once, these methods sacrifice the opportunity to refine their predictions, leaving no room to correct suboptimal decisions. As a result, one-shot methods typically yield inferior solution quality despite their speed advantage. Moreover, because these methods learn a single deterministic mapping from problem instances to priorities, inference-time sampling perturbs only the final output of that mapping, and the resulting solutions remain local variations around one dominant ordering rather than structurally distinct schedules [10, 11].

These limitations raise a fundamental question: *How can we achieve high solution quality while maintaining the efficiency of one-shot methods?* One promising direction is to iteratively refine predictions rather than committing to a single fixed output, progressively correcting suboptimal priorities toward high-quality solutions. Crucially, the number of refinement steps remains fixed regardless of problem size, preserving the efficiency advantage of one-shot methods. Diffusion models provide a natural framework for this approach, as their iterative denoising process enables gradual refinement from a random initialization to a high-quality solution [12]. Furthermore, since refinement proceeds through a sequence of stochastic updates rather than a single mapping, independent runs traverse different trajectories and can converge to structurally distinct solutions. Together, these properties directly address both limitations of one-shot methods.

Contributions. In this paper, we introduce SchedDiff, the first diffusion-based framework for JSSP that refines operation priorities through iterative denoising. Rather than committing to fixed predictions in a single pass, SchedDiff corrects suboptimal priorities over a fixed number of denoising steps, far fewer than the N steps required by constructive methods. This enables progressive improvement toward high-quality solutions while preserving computational efficiency. Unlike prior diffusion models for combinatorial optimization that operate in Euclidean or binary spaces, SchedDiff operates directly in a discrete ordinal space, as operation priorities are inherently rankings. To this end, we design a forward process and a learning objective specifically tailored to ranking structures. Furthermore, we augment the standard list scheduling used in one-shot methods with gap insertion, which never yields a longer makespan for the same priority ordering while guaranteeing feasibility.

To validate the effectiveness of SchedDiff, we conduct extensive experiments on widely studied JSSP benchmarks, including TA, DMU, and LA. The experimental results show that SchedDiff achieves state-of-the-art performance among learning-based methods, with efficiency advantages that become more pronounced on larger instances. Moreover, SchedDiff exhibits strong generalization: it scales to instances up to $40\times$ larger than those seen during training, remains robust under unseen problem distributions, and even transfers zero-shot to flexible JSSP without any architectural modification.

2 Related Work

Neural Combinatorial Optimization. Neural approaches to combinatorial optimization have gained significant attention, with scheduling being a key application domain. Within this line of research, constructive methods have become the dominant paradigm, pioneered by L2D [6], which introduced a graph neural network (GNN)-based policy to construct schedules in an autoregressive manner. Building on this foundation, subsequent works have extended this framework by developing expressive network architectures such as attention mechanisms and Transformers [13–16], or by exploring alternative learning paradigms [7, 17–20]. Despite their strong solution quality, these methods inherently require N sequential forward passes, limiting scalability to large instances. Rather than building solutions from scratch, improvement-based methods start from complete solutions and seek to enhance their quality. These approaches typically employ neural-guided local search [21–23] or hybrid frameworks that integrate learning with classical heuristics [11, 24, 25]. While capable of producing high-quality solutions, they incur substantial computational overhead because they rely on extensive search procedures. Conversely, one-shot methods prioritize computational efficiency to overcome these scalability challenges. These methods predict all priorities simultaneously in a single forward pass, producing a global priority map that is then decoded into a feasible schedule via list

scheduling. Recent approaches include Jeon et al. [8], which samples priorities via the Gumbel-Top-k trick to enable policy gradient optimization, and Qi et al. [9], which identifies valid comparabilities between operations and leverages dense rewards for training. This single-pass inference enables highly efficient scheduling but typically sacrifices solution quality compared to constructive or improvement-based methods, as all priorities are determined at once without the opportunity for correction.

Diffusion Models for Combinatorial Optimization. Diffusion models have demonstrated state-of-the-art performance in image generation [26, 27], language modeling [28, 29], and molecular design [30, 31]. Recently, they have attracted growing interest in combinatorial optimization. DIFUSCO [32] applied diffusion models to combinatorial optimization, formulating TSP and MIS as binary classification tasks where models predict the probability of edges or nodes being part of the optimal solution. T2T [33] introduced a gradient-based search during inference to refine solutions by leveraging the objective function guidance. Building on this work, FastT2T [34] utilized optimization consistency models to enable consistency-based gradient search. GenSCO [35] used generation as a search operator for large-scale instances, while StruDiCO [36] integrated structural guidance directly into the diffusion process to better learn valid topologies. NEXCO [37] incorporated adaptive solution expansion into masked diffusion, using feasibility projection to maintain valid partial solutions during generation. Targeting inference cost, EFLOCO [38] replaced stochastic denoising with deterministic transition paths and an adaptive time-step schedule.

However, these approaches operate on binary space representations, which are poorly suited for JSSP. Formulating JSSP in this framework requires selecting disjunctive arcs in the disjunctive graph, resulting in $O(m \cdot n^2)$ decision variables for m machines and n jobs, making the representation intractable as the problem size grows. Furthermore, these methods do not inherently guarantee feasibility and rely on post-processing to recover valid solutions, which becomes particularly difficult under the strict precedence and resource constraints of JSSP. Departing from binary representations, SymmetricDiffusers [39] defined diffusion directly over permutations. Its reverse transition, however, parameterizes a distribution over the entire permutation space with $O(N^2)$ scores per denoising step, which is prohibitive for JSSP, where $N = n \times m$ can reach thousands even at moderate problem sizes. These limitations highlight the need for a diffusion framework specifically tailored to the structural properties of JSSP.

3 Preliminaries

Job Shop Scheduling Problem. A JSSP involves scheduling a set of n jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ on a set of m machines $\mathcal{M} = \{M_1, \ldots, M_m\}$. Each job J_i consists of a sequence of operations $O_{i,1} \to O_{i,2} \to \cdots \to O_{i,m}$, which must be processed in the given order (precedence constraints). We denote the complete set of $N = n \times m$ operations as $\mathcal{O} = \{O_k\}_{k=1}^{N}$, where each O_k corresponds to an operation $O_{i,j}$. Each operation $O_{i,j}$ requires a specific machine $\mu_{i,j} \in \mathcal{M}$ and a processing time $p_{i,j} > 0$. Each machine can process at most one operation at a time (resource constraints), and preemption (i.e., interrupting an operation before completion) is not allowed. The objective is to minimize the makespan, defined as the completion time of the last operation $C_{\max} = \max_{i,j}(S_{i,j} + p_{i,j})$, by determining start times $S_{i,j}$ for each operation.

A standard representation for JSSP is the *disjunctive graph* $G = (V, C \cup D)$ [40]. The node set V includes all operations, along with dummy source and sink nodes. The edge set consists of two types: (1) *conjunctive arcs* C, representing the precedence constraints within each job (e.g., $O_{i,j} \to O_{i,j+1}$); and (2) *disjunctive arcs* D, connecting pairs of operations that share the same machine.

4 Methodology

4.1 SchedDiff Framework

We adopt a priority-based approach to JSSP, assigning each operation a priority score. These scores induce a ranking, i.e., a discrete assignment of ranks to operations. Given any ranking, list scheduling deterministically constructs a schedule that satisfies the constraints. Consequently, the scheduling problem reduces to generating high-quality priorities. Since only the relative ordering matters, the solution space is defined by rankings over N operations. We formulate this as a conditional generative

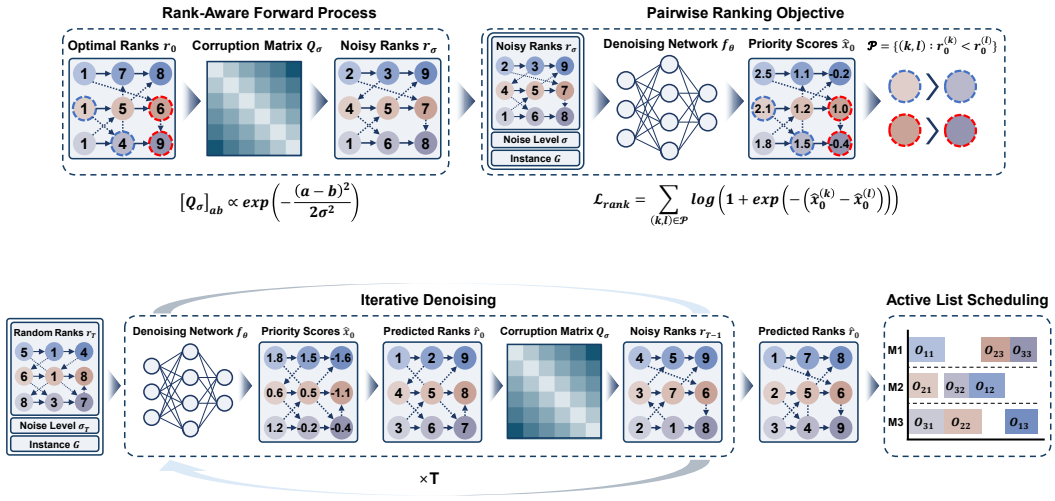


Figure 1: Overview of the SchedDiff framework. (Top) Training: optimal ranks are corrupted, and the denoising network is trained to recover the original ordering. (Bottom) Inference: random ranks are iteratively refined and decoded into a feasible schedule.

task, in which a diffusion model learns to generate optimal ranks conditioned on the problem instance. As in diffusion models for combinatorial optimization [32–37], each training instance is paired with a single optimal solution, so each instance is associated with a single target ranking (see Appendix C). Figure 1 illustrates the overall pipeline.

During training, the pipeline consists of two stages: (1) **Rank-Aware Forward Process**, which corrupts optimal ranks by perturbing each operation's rank according to a corruption matrix, and (2) **Pairwise Ranking Objective**, where a denoising network predicts priority scores from the corrupted ranks and is trained to preserve pairwise ordering with respect to the optimal ranks. At inference time, the pipeline also consists of two stages: (1) **Iterative Denoising**, where the model progressively refines random ranks into high-quality ranks through multiple denoising steps, and (2) **Active List Scheduling**, where the generated ranks are decoded into a feasible schedule by inserting operations into idle gaps to produce more compact schedules.

4.2 Diffusion on Discrete Ordinal Space

Forward Process. Diffusion models for combinatorial optimization typically operate over Euclidean spaces (adding Gaussian noise) or binary spaces (flipping bits with some probability). Neither is well suited for rankings, whose structure lies in relative ordering: Gaussian noise on continuous priority values perturbs absolute values that are meaningful only up to their order, while bit flipping treats states as unordered and ignores the proximity between nearby ranks. To address this, our forward process operates directly on the discrete ordinal space induced by priority scores. Let $r_0 \in \{1, 2, \ldots, N\}^N$ denote the clean ranks of N operations, where ties are allowed (see Appendix C). Unlike discrete diffusion models that define a Markov chain over discrete time steps [41], we define the forward process as a family of corruption marginals indexed by a continuous noise level σ [42]. Since the reverse process only needs to corrupt clean ranks to a given noise level (see below), the corruption at each noise level can be specified directly, without requiring corruptions at different noise levels to compose into a Markov chain. Specifically, for each operation k, the noisy rank is sampled from a categorical distribution defined by a corruption matrix $\mathbf{Q}_\sigma \in \mathbb{R}^{N\times N}$, inspired by the discretized Gaussian kernel [41]:

$$[\mathbf{Q}_\sigma]_{ab} = \frac{\exp\left(-(a-b)^2/2\sigma^2\right)}{\sum_{b'=1}^{N} \exp\left(-(a-b')^2/2\sigma^2\right)}, \tag{1}$$

where $[\mathbf{Q}_\sigma]_{ab}$ denotes the probability that an operation with clean rank a is assigned rank b at noise level σ. This kernel assigns higher probabilities to nearby ranks, while the ordering information

gradually degrades as σ increases. The marginal distribution for each operation k is then given by:

$$q(r_\sigma^{(k)} \mid r_0^{(k)}) = \mathrm{Cat}\big(r_\sigma^{(k)};\, \mathbf{e}_{r_0^{(k)}}^\top \mathbf{Q}_\sigma\big), \tag{2}$$

where $\mathbf{e}_{r_0^{(k)}} \in \{0,1\}^N$ denotes the one-hot vector corresponding to the rank index $r_0^{(k)}$. During training, the noise level is sampled from a log-normal distribution as in [42] (see Appendix D.2). As $\sigma \to \infty$, the distribution converges to the uniform distribution over ranks.

Reverse Process. The reverse process iteratively denoises rankings in the discrete ordinal space over a decreasing schedule of noise levels $\{\sigma_t\}_{t=1}^T$, where t indexes the denoising steps and r_t denotes the ranking at noise level σ_t. It starts from r_T drawn uniformly at random. At each step t, a neural network f_θ predicts clean priority scores $\hat{x}_0 = f_\theta(r_t, \sigma_t, G)$ from noisy ranks r_t, noise level σ_t, and problem instance G. The network f_θ outputs a scalar score per operation rather than a distribution over the N rank indices, which keeps the output dimension independent of problem size. The predicted scores are then mapped into the ordinal space to form clean ranks, $\hat{r}_0 = \mathrm{Rank}(\hat{x}_0)$, where rank indices are assigned by sorting the scores in descending order. To construct r_{t-1}, we adopt a predict-and-corrupt strategy [34, 43, 44], which re-corrupts the predicted clean ranks $\hat{r}_0$ using only the forward corruption at the next noise level:

$$p_\theta(r_{t-1} \mid r_t, G) = q(r_{t-1} \mid \hat{r}_0; \sigma_{t-1}). \tag{3}$$

Since f_θ is trained on rankings corrupted by the forward process, this provides it at each subsequent step with inputs of the same form as in training. In particular, whenever $\hat{r}_0$ recovers the ordering of the target ranking, the re-corrupted ranks follow the forward marginal at σ_{t-1} up to tie-breaking. The sequence of noise levels $\{\sigma_t\}_{t=1}^T$ follows the polynomial schedule [42] (see Appendix D.2).

Learning Objective. Diffusion models are typically trained by maximizing a variational lower bound on the data likelihood $p(x_0)$, which is commonly simplified to a mean squared error under a Gaussian parameterization [12], while x_0-parameterized discrete diffusion models combine it with a per-coordinate cross-entropy over the clean state [41]. In our setting, however, each instance has a single target ranking, so the role of the objective is to recover this target rather than to fit a distribution over solutions, and a likelihood bound on $p(r_0 \mid G)$ is not required. Moreover, these standard forms are ill-suited to rankings. The Gaussian form supervises absolute priority values, yet any two priority vectors inducing the same ranking produce the same schedule, so these values are only meaningful up to their relative order. The discrete form avoids this by supervising rank indices directly, but it requires an N-way output per operation, tying the output dimension to problem size and preventing inference on instances larger than those seen during training. It also treats ranks as unordered labels, discarding the ordinal proximity our forward process is designed to preserve. Since list scheduling depends only on the relative order of priorities, we take the pairwise orderings induced by r_0 as the prediction target. Given the predicted scores $\hat{x}_0$, the Bradley-Terry model [45] assigns to each pair the probability $P(k \succ l) = \frac{1}{1+\exp\left(-(\hat{x}_0^{(k)} - \hat{x}_0^{(l)})\right)}$, which depends only on score differences. We then maximize the composite likelihood of the pairwise comparisons over the ordered pairs $\mathcal{P} = \{(k,l) : r_0^{(k)} < r_0^{(l)}\}$, i.e., the product $\prod_{(k,l)\in\mathcal{P}} P(k \succ l)$. This is equivalent to minimizing the following pairwise ranking loss (see Appendix A):

Algorithm 1: SchedDiff Training

Input : Dataset $\mathcal{D} = \{(G, r_0)\}$, model f_θ, noise $p(\sigma)$
for *each epoch* **do**
  for *each* $(G, r_0) \in \mathcal{D}$ **do**
    Sample noise level $\sigma \sim p(\sigma)$;
    Sample $r_\sigma \sim q(r_\sigma \mid r_0)$ for each operation;
    Predict $\hat{x}_0 = f_\theta(r_\sigma, \sigma, G)$;
    Compute $\mathcal{L}_{\text{rank}}(\hat{x}_0, r_0)$ and update θ;
  end
end

$$\mathcal{L}_{\text{rank}} = \sum_{(k,l)\in\mathcal{P}} \log\left(1 + \exp\left(-(\hat{x}_0^{(k)} - \hat{x}_0^{(l)})\right)\right). \tag{4}$$

Minimizing $\mathcal{L}_{\text{rank}}$ encourages $\hat{x}_0$ to order every pair in $\mathcal{P}$ as in r_0, which is precisely the condition stated above under which the reverse process follows the forward marginals. Supervising pairwise orderings also requires only the scalar scores defined above, preserving both size-agnostic inference and the ordinal structure of the ranking. The complete training procedure is summarized in Algorithm 1.

4.3 Denoiser Architecture

The denoising network f_θ takes noisy ranks, the noise level, and the problem instance as input, and predicts clean priority scores. We design a denoiser that integrates graph attention with noise-adaptive modulation to perform effective denoising by leveraging the disjunctive graph structure. The architecture consists of an input projection layer, L stacked Adaptive Graph Attention (AGA) blocks, and a final output layer. The full architecture is illustrated in Figure 6.

Input. The input to the denoiser consists of the disjunctive graph G, noisy ranks r_σ, and noise level σ. Each node k has static features z_k^{static} (detailed in Appendix D.1) concatenated with the noisy rank $r_\sigma^{(k)}$. These are then projected to an initial representation:

$$h_k^0 = W_{\text{proj}}[z_k^{\text{static}} \,\|\, r_\sigma^{(k)}]. \tag{5}$$

The noise level σ is incorporated via adaptive conditioning, described below.

Adaptive Graph Attention Block. The backbone of our architecture comprises L stacked AGA blocks, each combining graph attention with σ-conditioned modulation. To capture the distinct characteristics of conjunctive and disjunctive arcs, we adapt Graph Attention Networks (GAT) by incorporating learnable edge-type embeddings directly into the attention computation. Furthermore, to dynamically condition node features across varying noise levels, we employ Adaptive Layer Normalization with zero-initialized gating (AdaLN-Zero) [27] to modulate each block as a function of σ (see Appendix E.3 for an ablation study). Specifically, we obtain a noise-level embedding $\sigma_{\text{emb}} = \text{SinEmb}(\sigma)$ via a sinusoidal embedding, and for each block l, an MLP predicts layer-specific modulation parameters (γ, β) and gating parameters α from σ_{emb}:

$$(\gamma_1, \beta_1, \alpha_1, \gamma_2, \beta_2, \alpha_2)^{(l)} = \text{MLP}^{(l)}(\sigma_{\text{emb}}). \tag{6}$$

These parameters modulate the GAT and Feed-Forward Network via gated residual connections:

$$h^{l'} = h^l + \alpha_1^{(l)} \odot \text{GAT}\left((1+\gamma_1^{(l)}) \odot \text{LN}(h^l) + \beta_1^{(l)},\, E\right), \tag{7}$$

$$h^{l+1} = h^{l'} + \alpha_2^{(l)} \odot \text{FFN}\left((1+\gamma_2^{(l)}) \odot \text{LN}(h^{l'}) + \beta_2^{(l)}\right), \tag{8}$$

where E denotes the edge set of the disjunctive graph and LN denotes layer normalization.

Output. The final layer projects the node representations h_k^L to scalar priority scores $\hat{x}_0^{(k)}$ via an AdaLN-conditioned linear layer. These scores are then converted to ranks using the Rank function and decoded into a feasible schedule via active list scheduling.

4.4 Active List Scheduling

Given the predicted ranks, we decode them into a feasible schedule using list scheduling. List scheduling iteratively selects the highest-priority eligible operation and assigns it to its designated machine, ensuring that both precedence and resource constraints are satisfied. An eligible operation is an unscheduled operation whose predecessors within the same job have all been scheduled. This deterministic decoding guarantees feasibility for any predicted ranks. Standard list scheduling, commonly used in one-shot methods, appends each operation at the end of its machine's timeline [46]. However, this can leave idle gaps when operations with different predecessors are interleaved.

Algorithm 2: SchedDiff Inference

Input: Instance G, model f_θ, steps T, noise schedule $\{\sigma_t\}_{t=1}^T$
Initialize r_T uniformly at random; $\mathcal{R} \leftarrow \emptyset$;
for $t = T, T-1, \ldots, 1$ **do**
  Predict $\hat{x}_0 = f_\theta(r_t, \sigma_t, G)$;
  Convert to ranks: $\hat{r}_0 \leftarrow \text{Rank}(\hat{x}_0)$;
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{\hat{r}_0\}$;
  if $t > 1$ **then**
    Re-corrupt: $r_{t-1} \sim q(r_{t-1} \mid \hat{r}_0; \sigma_{t-1})$ for each operation;
  end
end
Decode all $\hat{r}_0 \in \mathcal{R}$ via ActiveListScheduling in parallel;
Output: Schedule with the minimum makespan

To address this inefficiency, we extend standard list scheduling with gap insertion, which we refer to as *active list scheduling*, as illustrated in Figure 2. When scheduling an operation, we search for idle gaps in the existing schedule where the operation can fit while respecting precedence and resource

constraints. This produces active schedules in which no operation can finish earlier by changing the processing order without delaying another. The class of active schedules is guaranteed to contain an optimal solution for regular objectives such as makespan [47] (see Appendix B.1 for details). We prove that this approach never yields a longer makespan than standard list scheduling:

Proposition 4.1. *Let $C_{\max}^{\text{std}}$ and $C_{\max}^{\text{active}}$ denote the makespans produced by standard list scheduling and active list scheduling, respectively, for the same priority ordering. Then $C_{\max}^{\text{active}} \leq C_{\max}^{\text{std}}$.*

The proof of Proposition 4.1 and the detailed algorithm for active list scheduling are provided in Appendix B.2. The complete inference procedure is summarized in Algorithm 2.

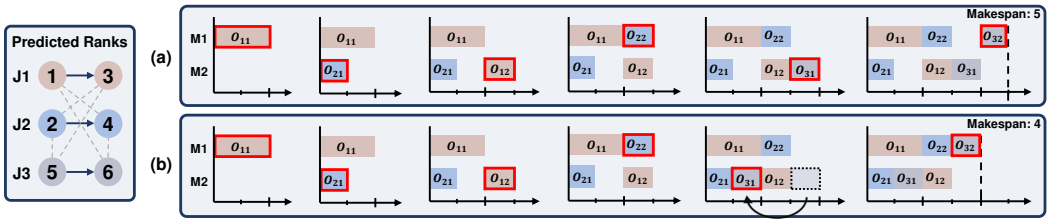


Figure 2: (a) Standard list scheduling and (b) active list scheduling for the same predicted ranks.

5 Experiments

To demonstrate the effectiveness and efficiency of SchedDiff, we conduct extensive experiments on widely studied JSSP benchmarks covering various problem sizes and difficulties. We first outline the experimental setup, including datasets and baselines, in Section 5.1, followed by a discussion of the main results and analyses in Section 5.2. All experiments are conducted on a 24-core Intel(R) i9-14900KS CPU and a single NVIDIA GeForce RTX 4090 GPU with 24 GB memory.

5.1 Experimental Settings

Datasets and Evaluation. We generate synthetic JSSP instances for training [6], with processing times sampled uniformly from $[1, 99]$, creating a dataset of 2,000 instances for each problem size $n \times m \in \{15 \times 15, 20 \times 15, 20 \times 20\}$. For each instance, we obtain optimal or near-optimal ranks using OR-Tools [48] with a time limit (detailed in Appendix C; see Appendix E.6 for results under a reduced labeling budget). We evaluate on standard public benchmarks: TA [49], DMU [50], and LA [51]. At inference time, we initialize the denoising process from K independent random ranks, refine each over T denoising steps, and report the best solution encountered throughout the denoising process. For the main results, we use $K \in \{1, 5\}$ and $T = 200$.

Baseline Methods. We compare SchedDiff against a wide range of baselines, including current state-of-the-art methods across four categories: (1) *Exact Solvers*: Gurobi [52] and OR-Tools [48]; (2) *Constructive Methods*: L2D [6], SN [13], RASCL [18], SI GD [19], SLIM [7], MP-ASIL [53], and BOPO [20]; (3) *Improvement-based Methods*: L2S [22] and TBGAT [23]; (4) *One-shot Methods*: Jeon et al. [8] and Qi et al. [9]. To ensure a fair comparison, we re-implement the one-shot baselines using our model architecture, as they share the same priority-based formulation.

Performance Metrics. We report the average performance gap (Gap) and total inference time (Time). The gap is calculated as $\frac{1}{|\mathcal{D}|} \sum_{s \in \mathcal{D}} \frac{C_{\max}(s) - C_{\max}^*(s)}{C_{\max}^*(s)} \times 100\ (\%)$, where $C_{\max}(s)$ denotes the makespan achieved by the model for instance s, and $C_{\max}^*(s)$ is the best-known makespan. Since hardware configurations and experimental settings vary across methods, the reported times may not be strictly comparable. Results sourced from the original papers are marked with an asterisk (*).

5.2 Experimental Results

Benchmark Results. Table 1 summarizes the performance gap and inference time on the TA benchmark across problem sizes from 15×15 to 100×20. SchedDiff achieves the lowest average gap among all learning-based methods, reaching **4.9%** with $K = 1$ and **4.2%** with $K = 5$. This

represents a substantial improvement over one-shot methods (16.9%) and constructive methods such as MP-ASIL (7.4%), while narrowing the gap to OR-Tools (2.0%). Furthermore, SchedDiff's speed advantage over constructive baselines becomes increasingly pronounced as instances grow larger. SchedDiff also maintains the best performance on most unseen problem sizes up to $5\times$ larger than those seen during training (marked with •), demonstrating strong size generalization. This robustness further extends to the DMU benchmark (Table 2), which poses a more challenging out-of-distribution setting: processing times are drawn from $U(1, 200)$ rather than the training distribution $U(1, 99)$. Despite this distributional shift, SchedDiff consistently outperforms all learning-based baselines, achieving an average gap of **9.1%** with $K = 1$ and **8.5%** with $K = 5$, compared to BOPO (12.9%). Results on the LA benchmark are provided in Appendix E.1.

Table 1: Experimental results on the TA benchmark. 's', 'm', and 'h' denote seconds, minutes, and hours. • indicates problem sizes unseen during training. G denotes greedy decoding, while K represents the number of sampled solutions. **Bold** highlights the best among learning-based methods.

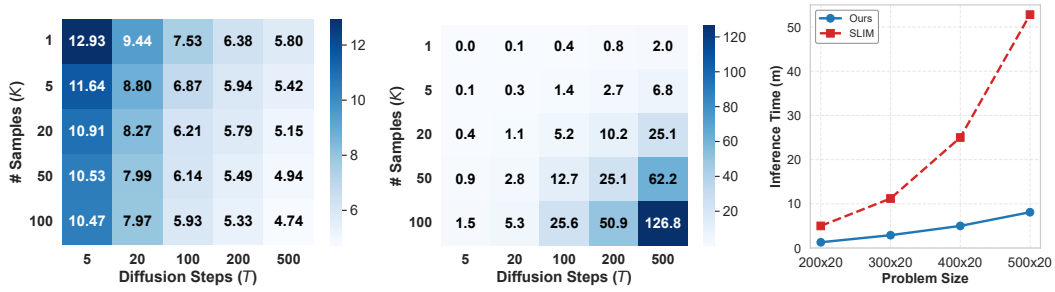
Method	15 × 15		20 × 15		20 × 20		30 × 15 •		30 × 20 •		50 × 15 •		50 × 20 •		100 × 20 •		Avg.
	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap
Gurobi* (3600s) [52]	0.1%	(10h)	3.2%	(10h)	2.9%	(10h)	10.7%	(10h)	13.2%	(10h)	12.2%	(10h)	13.6%	(10h)	11.0%	(10h)	8.4%
OR-Tools* (3600s) [48]	0.1%	(77m)	0.2%	(8h)	0.7%	(10h)	2.1%	(10h)	2.8%	(10h)	3.0%	(10h)	2.8%	(10h)	3.9%	(10h)	2.0%
L2D* (G) [6]	26.0%	(4s)	30.0%	(6s)	31.6%	(7s)	33.0%	(10s)	33.6%	(14s)	22.4%	(18s)	26.5%	(30s)	13.6%	(1.6m)	27.1%
L2D* (K=128) [6]	17.1%	(6.8m)	23.7%	(9.0m)	22.6%	(13.1m)	24.4%	(15.6m)	28.4%	(26.3m)	17.1%	(35.9m)	20.4%	(1.0h)	10.3%	(3.4h)	20.5%
SN* (G) [13]	15.3%	(35s)	19.4%	(1.1m)	17.2%	(1.8m)	19.1%	(2.9m)	23.7%	(4.7m)	13.9%	(8.8m)	13.5%	(16.0m)	6.7%	(1.2h)	16.1%
RASCL* (G) [18]	14.3%	(8s)	16.5%	(11s)	17.3%	(14s)	18.5%	(15s)	21.5%	(17s)	12.2%	(28s)	13.2%	(39s)	5.9%	(1.6m)	14.9%
SI GD* (G) [19]	9.6%	(10s)	9.9%	(11s)	11.1%	(11s)	9.5%	(12s)	13.8%	(15s)	2.7%	(24s)	6.7%	(39s)	1.7%	(5.0m)	8.1%
SLIM* (K=512) [7]	6.5%	(3s)	8.8%	(5s)	9.0%	(7s)	10.6%	(9s)	12.7%	(12s)	4.9%	(22s)	7.6%	(29s)	2.1%	(1.5m)	7.8%
MP-ASIL* (K=512) [53]	6.8%	(3s)	8.5%	(5s)	8.7%	(7s)	10.4%	(9s)	12.8%	(12s)	4.2%	(22s)	7.0%	(29s)	**1.0%**	(1.5m)	7.4%
BOPO* (K=512) [20]	6.3%	(8s)	8.3%	(10s)	9.1%	(12s)	10.3%	(14s)	12.2%	(19s)	4.9%	(26s)	7.4%	(34s)	1.4%	(1.3m)	7.5%
L2S*-500 [22]	9.3%	(1.6m)	11.6%	(1.7m)	12.4%	(1.8m)	14.7%	(2.1m)	17.5%	(2.3m)	11.0%	(2.7m)	13.0%	(3.8m)	7.9%	(8.4m)	12.2%
TBGAT*-500 [23]	8.0%	(2.1m)	9.9%	(2.4m)	10.0%	(2.9m)	13.3%	(2.9m)	16.4%	(3.2m)	9.6%	(4.0m)	11.9%	(4.1m)	6.4%	(7.0m)	10.7%
Jeon et al. (K=512) [8]	14.9%	(<1s)	15.7%	(<1s)	16.7%	(<1s)	19.0%	(<1s)	23.4%	(<1s)	14.1%	(<1s)	18.6%	(<1s)	13.0%	(1s)	16.9%
Qi et al. (K=512) [9]	15.8%	(<1s)	16.5%	(<1s)	17.6%	(<1s)	18.8%	(<1s)	24.0%	(<1s)	13.9%	(<1s)	17.8%	(<1s)	11.9%	(2s)	17.0%
SchedDiff (K=1, T=200)	**5.5%**	(9s)	**5.2%**	(9s)	**5.4%**	(9s)	**4.4%**	(9s)	**9.0%**	(9s)	**1.3%**	(9s)	**6.4%**	(9s)	2.0%	(24s)	**4.9%**
SchedDiff (K=5, T=200)	**4.4%**	(9s)	**3.7%**	(9s)	**5.1%**	(9s)	**4.3%**	(9s)	**8.0%**	(13s)	**0.8%**	(22s)	**6.0%**	(29s)	1.5%	(1.4m)	**4.2%**

Table 2: Experimental results on the DMU benchmark.

Method	20 × 15	20 × 20	30 × 15 •	30 × 20 •	40 × 15 •	40 × 20 •	50 × 15 •	50 × 20 •	Avg.
Gurobi* (3600s) [52]	5.3%	4.7%	14.2%	16.7%	16.3%	22.5%	14.9%	22.5%	14.6%
OR-Tools* (3600s) [48]	1.8%	1.9%	2.5%	4.4%	4.1%	4.6%	3.8%	4.8%	3.5%
L2D* (G) [6]	39.0%	37.7%	42.0%	39.7%	35.6%	39.6%	36.5%	39.5%	38.7%
L2D* (K=128) [6]	29.3%	27.1%	34.0%	33.6%	31.5%	35.8%	32.7%	36.1%	32.5%
RASCL* (K=512) [18]	19.4%	16.0%	16.5%	20.2%	17.6%	25.6%	21.7%	15.2%	19.0%
SI GD* (G) [19]	15.4%	15.8%	16.6%	16.1%	11.6%	16.6%	10.1%	17.4%	14.9%
SLIM* (K=512) [7]	11.3%	12.3%	14.0%	15.8%	10.9%	14.8%	10.6%	15.0%	13.1%
BOPO* (K=512) [20]	10.4%	11.8%	12.9%	15.5%	10.9%	15.9%	10.4%	15.5%	12.9%
L2S*-500 [22]	21.1%	18.2%	26.4%	27.5%	25.8%	29.2%	26.4%	32.4%	25.8%
Jeon et al. (K=512) [8]	25.5%	23.2%	30.8%	31.7%	30.5%	35.3%	32.6%	40.1%	31.2%
Qi et al. (K=512) [9]	24.8%	23.1%	31.0%	30.0%	30.4%	32.7%	30.1%	38.1%	30.0%
SchedDiff (K=1, T=200)	**7.1%**	**7.8%**	**8.6%**	**10.8%**	**7.1%**	**11.4%**	**7.1%**	**13.0%**	**9.1%**
SchedDiff (K=5, T=200)	**6.4%**	**7.2%**	**7.5%**	**9.7%**	**6.7%**	**11.2%**	**6.9%**	**12.6%**	**8.5%**

Impact of Inference Parameters (K and T). To understand the trade-off between solution quality and inference cost, we analyze the performance on the TA 50×20 instances as a representative benchmark. As illustrated in Figures 3a and 3b, increasing both T and K consistently improves solution quality, albeit at the cost of increased time. Specifically, larger T allows for more refined denoising, while larger K further reduces the performance gap by exploring structurally distinct solutions (see Appendix E.4 for denoising trajectory analysis). This flexibility enables SchedDiff to balance computational cost and solution quality according to the available computational budget.

Scalability and Efficiency. To evaluate scalability beyond standard benchmarks, we generate large-scale synthetic instances up to 500×20, which are $25\times$ larger than the training sizes. Figure 3c compares the inference time of SchedDiff against SLIM [7], one of the strongest constructive baselines in both speed and solution quality. SchedDiff is consistently 3–7$\times$ faster than SLIM, with the speedup widening as problem size increases, while maintaining comparable or better solution quality. This is because SLIM requires N sequential forward passes, resulting in superlinear scaling, whereas SchedDiff performs a fixed number (T) of forward passes regardless of problem size. Beyond this scale, SLIM encounters out-of-memory errors, whereas SchedDiff continues to scale robustly (see Appendix E.2 for results up to 800×20).



(a) Performance Gap (%) (b) Inference Time (s) (c) Scalability Comparison

Figure 3: (a) Performance gap for varying K and T. (b) Inference time for varying K and T. (c) Scalability comparison: SchedDiff scales efficiently to large instances, whereas SLIM suffers from rapid time growth and out-of-memory errors.

Diversity Analysis. To assess the diversity of generated schedules, we employ two widely adopted metrics: Graph Edit Distance (GED) and start-time L_2 distance (d_{inv}) [54]. GED measures structural dissimilarity between schedules by comparing their machine-wise precedence graphs and is computed as the normalized difference between the corresponding adjacency matrices. d_{inv} captures temporal divergence between schedules and is computed as the L_2 distance between operation start-time vectors. To ensure that high diversity does not stem from poor-quality solutions, we evaluate these metrics exclusively on the top 10% of solutions by makespan. We focus on the TA 15×15 instances, as smaller problem sizes constrain the solution space and make high diversity challenging to achieve. As shown in Table 3, where each method draws 100 solutions per instance, SchedDiff generates more diverse solutions than one-shot baselines [8, 9] while achieving the best average makespan. One-shot methods perturb only the final output of a single deterministic mapping, so sampled solutions remain local variations of a single schedule. In contrast, SchedDiff's independent trajectories start from random initializations and diverge through stochastic refinement steps, converging to structurally distinct schedules.

Method	Ms.↓	d_{inv} ↑	GED↑
Jeon et al. [8]	1433	414	0.23
Qi et al. [9]	1444	244	0.11
SchedDiff	**1296**	**1506**	**1.27**

Table 3: Diversity analysis.

Ablation Study. To validate our key design choices, we ablate two components: the *ranking formulation* (rank-aware forward process + pairwise ranking loss) and *active list scheduling*, as shown in Table 4. Replacing the ranking formulation with standard continuous diffusion (Gaussian noise + MSE loss) causes the most severe degradation among individual components (up to +8.3%). This stems from a fundamental misalignment with ranking structure: Gaussian noise operates on absolute priority values rather than ordinal structure, providing no coherent notion of proximity in the solution space. MSE compounds this issue. Any two priority vectors inducing the same ranking produce identical schedules, yet MSE treats them as distinct targets, failing to reflect the fact that schedule quality depends solely on their relative ordering. Replacing active list scheduling with standard list scheduling also degrades performance, with the gap widening as problem size increases (from +0.3% to +4.2%). This trend suggests that larger instances generate more idle gaps due to increased contention among jobs across machines, providing more opportunities for active list scheduling to insert operations and produce more compact schedules.

Ranking Formulation	Active List Scheduling	15×15	30×15	50×20	100×20
–	–	15.4% (+11.0%)	16.8% (+12.5%)	20.9% (+14.9%)	13.7% (+12.2%)
✓	–	4.7% (+0.3%)	4.7% (+0.4%)	7.6% (+1.6%)	5.7% (+4.2%)
–	✓	10.9% (+6.5%)	12.6% (+8.3%)	13.2% (+7.2%)	7.3% (+5.8%)
✓	✓	**4.4%**	**4.3%**	**6.0%**	**1.5%**

Table 4: Ablation study on TA. Red values indicate degradation relative to the full model.

Experiments with Flexible JSSP. To evaluate the transferability of the learned priority model, we apply SchedDiff, trained exclusively on standard JSSP, to the Flexible Job Shop Scheduling Problem (FJSP) without retraining or architectural changes. Following the standard priority dispatching rule paradigm for FJSP, SchedDiff determines operation priorities, while the Earliest Completion Time (ECT) rule assigns operations to machines. We benchmark performance on widely used FJSP datasets, including Brandimarte [56] and Hurink (edata, rdata, vdata) [57], and compare against DANIEL [15] and RESCHED [55], state-of-the-art neural solvers specifically designed for FJSP. We report the average gap over all 130 instances: 10 from Brandimarte and 40 from each Hurink group. As shown in Table 5, SchedDiff achieves the lowest average gap among neural solvers (**5.1%**), compared with DANIEL (5.3%) and RESCHED (5.5%). SchedDiff performs better on Brandimarte and edata, while both baselines achieve lower gaps on the more flexible rdata and vdata instances. Overall, these results support the generalization of the learned priority representations from standard JSSP to FJSP when paired with ECT machine assignment.

Method	Brand.	edata	rdata	vdata	**Avg.**
OR-Tools* (1800s) [48]	1.5%	0.0%	0.1%	0.0%	0.1%
DANIEL* (K=100) [15]	9.5%	9.1%	5.0%	**0.7%**	5.3%
RESCHED* (K=100) [55]	8.1%	10.4%	**4.9%**	**0.7%**	5.5%
SchedDiff (K=5, T=200)	**7.1%**	**6.2%**	7.5%	1.2%	**5.1%**

Table 5: Experimental results on FJSP benchmarks.

6 Conclusion

In this paper, we propose SchedDiff, a diffusion-based framework for JSSP that iteratively refines operation priorities in a discrete ordinal space. By replacing one-shot prediction with a fixed number of denoising steps, SchedDiff resolves the trade-off between inference efficiency and solution quality, while its refinement from random initializations through stochastic updates yields diverse high-quality schedules. We further design a rank-aware forward process and a pairwise ranking objective tailored to the ranking structure, together with active list scheduling that never yields a longer makespan for the same priority ordering. Extensive experiments across diverse benchmarks demonstrate that SchedDiff consistently outperforms existing neural solvers, achieving state-of-the-art performance in both solution quality and scalability. Overall, this work establishes diffusion over a discrete ordinal space as a powerful paradigm for scheduling, opening new directions for generative approaches in this domain.

Limitations and Future Directions. SchedDiff formulates scheduling as priority generation, where the model determines only the processing order of operations. In our FJSP experiments, machine assignment is delegated to the ECT rule. Extending SchedDiff to generate operation priorities and machine assignments jointly could enable fully end-to-end optimization and serve as a stepping stone toward a general-purpose neural scheduling model for diverse scheduling problems. In addition, although multiple optimal solutions may exist, SchedDiff is trained on a single solution per instance, as in existing diffusion models for combinatorial optimization, because obtaining optimal or near-optimal solutions is computationally expensive. Accordingly, our framework is designed to recover a single target ranking per instance rather than to capture the full set of high-quality schedules. Although stochastic refinement already yields structurally distinct schedules at inference, exploring self-supervised objectives that do not rely on solver-generated labels could relax this limitation, allowing the model to explicitly learn a distribution over high-quality schedules while further improving practicality.

Acknowledgments and Disclosure of Funding

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP)-Information Technology Research Center (ITRC) grant funded by the Korea government (Ministry of Science and ICT, MSIT) (IITP-2026-RS-2024-00437268); by the MSIT, Republic of Korea, through the National IT Industry Promotion Agency (NIPA) (Grant No. RS-2026-25621649); and by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2025-02216640).

References

[1] Yilin Fang, Chao Peng, Ping Lou, Zude Zhou, Jianmin Hu, and Junwei Yan. Digital-twin-based job shop scheduling toward smart manufacturing. *IEEE Transactions on Industrial Informatics*, 15(12):6425–6435, 2019.

[2] Lei Cai, Wenfeng Li, Yun Luo, and Lijun He. Real-time scheduling simulation optimisation of job shop in a production-logistics collaborative environment. *International Journal of Production Research*, 61(5):1373–1393, 2023.

[3] Mohit Kumar, Subhash Chander Sharma, Anubhav Goel, and Santar Pal Singh. A comprehensive survey for scheduling techniques in cloud computing. *Journal of Network and Computer Applications*, 143:1–33, 2019.

[4] Wen-Yang Ku and J Christopher Beck. Mixed integer programming models for job shop scheduling: A computational analysis. *Computers & Operations Research*, 73:165–173, 2016.

[5] Teofilo F Gonzalez. *Handbook of approximation algorithms and metaheuristics*. Chapman and Hall/CRC, 2007.

[6] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1621–1632, 2020.

[7] Andrea Corsini, Angelo Porrello, Simone Calderara, and Mauro Dell'Amico. Self-labeling the job shop scheduling problem. *Advances in Neural Information Processing Systems*, 37: 105528–105551, 2024.

[8] Wonseok Jeon, Mukul Gagrani, Burak Bartan, Weiliang Will Zeng, Harris Teague, Piero Zappi, and Christopher Lott. Neural DAG scheduling via one-shot priority sampling. In *The Eleventh International Conference on Learning Representations*, 2023.

[9] Xumai Qi, Dongdong Zhang, Taotao Liu, and Hongcheng Wang. Reinforcement learning for one-shot DAG scheduling with comparability identification and dense reward. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[10] Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*, 2017.

[11] Minsu Kim, Sanghyeok Choi, Hyeonah Kim, Jiwoo Son, Jinkyoo Park, and Yoshua Bengio. Ant colony sampling with GFlowNets for combinatorial optimization. *arXiv preprint arXiv:2403.07041*, 2024.

[12] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[13] Junyoung Park, Sanjar Bakhtiyar, and Jinkyoo Park. ScheduleNet: Learn to solve multi-agent scheduling problems with reinforcement learning. *arXiv preprint arXiv:2106.03051*, 2021.

[14] Jun-Ho Lee and Hyun-Jung Kim. Reinforcement learning for robotic flow shop scheduling with processing time variations. *International Journal of Production Research*, 60(7):2346–2368, 2022.

[15] Runqing Wang, Gang Wang, Jian Sun, Fang Deng, and Jie Chen. Flexible job shop scheduling via dual attention network-based reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 35(3):3091–3102, 2023.

[16] Ruiqi Chen, Wenxin Li, and Hongbing Yang. A deep reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for the job-shop scheduling problem. *IEEE Transactions on Industrial Informatics*, 19(2):1322–1331, 2022.

[17] Junyoung Park, Jaehyeong Chun, Sang Hun Kim, Youngkook Kim, and Jinkyoo Park. Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning. *International Journal of Production Research*, 59(11):3360–3377, 2021.

[18] Zangir Iklassov, Dmitrii Medvedev, Ruben Solozabal Ochoa De Retana, and Martin Takac. On the study of curriculum learning for inferring dispatching policies on the job shop scheduling. In *International Joint Conference on Artificial Intelligence*, pages 5350–5358, 2023.

[19] Jonathan Pirnay and Dominik G Grimm. Self-improvement for neural combinatorial optimization: Sample without replacement, but improvement. *arXiv preprint arXiv:2403.15180*, 2024.

[20] Zijun Liao, Jinbiao Chen, Debing Wang, Zizhen Zhang, and Jiahai Wang. BOPO: Neural combinatorial optimization via best-anchored and objective-guided preference optimization. *arXiv preprint arXiv:2503.07580*, 2025.

[21] Jonas K Falkner, Daniela Thyssens, Ahmad Bdeir, and Lars Schmidt-Thieme. Learning to control local search for combinatorial optimization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 361–376. Springer, 2022.

[22] Cong Zhang, Zhiguang Cao, Wen Song, Yaoxin Wu, and Jie Zhang. Deep reinforcement learning guided improvement heuristic for job shop scheduling. *arXiv preprint arXiv:2211.10936*, 2022.

[23] Cong Zhang, Zhiguang Cao, Yaoxin Wu, Wen Song, and Jing Sun. Learning topological representations with bidirectional graph attention network for solving job shop scheduling problem. *arXiv preprint arXiv:2402.17606*, 2024.

[24] Hyeonah Kim, Sanghyeok Choi, Jiwoo Son, Jinkyoo Park, and Changhyun Kwon. Neural genetic search in discrete spaces. *arXiv preprint arXiv:2502.10433*, 2025.

[25] Haoran Ye, Jiarui Wang, Zhiguang Cao, Helan Liang, and Yong Li. DeepACO: Neural-enhanced ant systems for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:43706–43728, 2023.

[26] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[27] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023.

[28] Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. Diffusion-LM improves controllable text generation. *Advances in Neural Information Processing Systems*, 35:4328–4343, 2022.

[29] Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

[30] Minkai Xu, Lantao Yu, Yang Song, Chence Shi, Stefano Ermon, and Jian Tang. GeoDiff: A geometric diffusion model for molecular conformation generation. *arXiv preprint arXiv:2203.02923*, 2022.

[31] Joseph L Watson, David Juergens, Nathaniel R Bennett, Brian L Trippe, Jason Yim, Helen E Eisenach, Woody Ahern, Andrew J Borst, Robert J Ragotte, Lukas F Milles, et al. De novo design of protein structure and function with RFdiffusion. *Nature*, 620(7976):1089–1100, 2023.

[32] Zhiqing Sun and Yiming Yang. DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:3706–3731, 2023.

[33] 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.

[34] Yang Li, Jinpei Guo, Runzhong Wang, Hongyuan Zha, and Junchi Yan. Fast T2T: Optimization consistency speeds up diffusion-based training-to-testing solving for combinatorial optimization. *Advances in Neural Information Processing Systems*, 37:30179–30206, 2024.

[35] Yang Li, Lvda Chen, Haonan Wang, Runzhong Wang, and Junchi Yan. Generation as search operator for test-time scaling of diffusion-based combinatorial optimization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[36] Yu Wang, Yang Li, Junchi Yan, and Yi Chang. StruDiCO: Structured denoising diffusion with gradient-free inference-stage boosting for memory and time efficient combinatorial optimization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[37] Yu Wang, Yang Li, Jiale Ma, Junchi Yan, and Yi Chang. NEXCO: Native solution expansion for diffusion-based combinatorial optimization. In *The Fourteenth International Conference on Learning Representations*, 2026.

[38] Yuanshu Li, Di Wang, Wei Du, Xuan Wu, Peng Zhao, Yubin Xiao, and You Zhou. Efficient few-step solution generation via discrete flow matching for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 37063–37071, 2026.

[39] Yongxing Zhang, Donglin Yang, and Renjie Liao. SymmetricDiffusers: Learning discrete diffusion models over finite symmetric groups. In *The Thirteenth International Conference on Learning Representations*, 2025.

[40] Jacek Błażewicz, Erwin Pesch, and Małgorzata Sterna. The disjunctive graph machine representation of the job shop scheduling problem. *European Journal of Operational Research*, 127 (2):317–331, 2000.

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

[42] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in Neural Information Processing Systems*, 35: 26565–26577, 2022.

[43] Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. In *Advances in Neural Information Processing Systems*, volume 26, 2013.

[44] Arpit Bansal, Eitan Borgnia, Hong-Min Chu, Jie Li, Hamid Kazemi, Furong Huang, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Cold diffusion: Inverting arbitrary image transforms without noise. *Advances in Neural Information Processing Systems*, 36:41259–41282, 2023.

[45] Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. The method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

[46] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[47] Bernard Giffler and Gerald Luther Thompson. Algorithms for solving production-scheduling problems. *Operations Research*, 8(4):487–503, 1960.

[48] Laurent Perron and Frédéric Didier. OR-Tools. URL https://developers.google.com/optimization.

[49] Eric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

[50] Ebru Demirkol, Sanjay Mehta, and Reha Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141, 1998.

[51] S Lawrence. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. *GSIA, Carnegie Mellon University*, 1984.

[52] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2026. URL https://www.gurobi.com.

[53] Inguk Choi, Woo-Jin Shin, Sanghyun Cho, and Hyun-Jung Kim. Towards generalizable multi-policy optimization with self-evolution for job scheduling. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[54] David W Zhang, Corrado Rainone, Markus Peschl, and Roberto Bondesan. Robust scheduling with GFlowNets. In *The Eleventh International Conference on Learning Representations*, 2023.

[55] Xiangjie Xiao, Zhiguang Cao, Cong Zhang, and Wen Song. RESCHED: Rethinking flexible job shop scheduling from a transformer-based architecture with simplified states. In *The Fourteenth International Conference on Learning Representations*, 2026.

[56] Paolo Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41(3):157–183, 1993.

[57] Johann Hurink, Bernd Jurisch, and Monika Thole. Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations Research Spektrum*, 15(4):205–215, 1994.

[58] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 89–96, 2005.

[59] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 5 edition, 2016. ISBN 978-3-319-26580-3. doi: 10.1007/978-3-319-26580-3.

A Ranking Loss Derivation

We derive the pairwise ranking loss used to train SchedDiff. In our setting, the model predicts continuous priority scores $\hat{x}_0 \in \mathbb{R}^N$, which are used to induce a ranking over operations. Since only the relative ordering of these scores matters for determining the schedule, we require a training objective that supervises the pairwise ordering structure rather than the absolute values of the predictions.

Bradley-Terry Model. To this end, we adopt the Bradley-Terry model [45], a classical probabilistic framework for modeling pairwise comparisons. Given predicted priority scores $\hat{x}_0$, the Bradley-Terry model defines the probability that operation k is ranked higher than operation l as:

$$P(k \succ l) = \frac{1}{1 + \exp\left(-(\hat{x}_0^{(k)} - \hat{x}_0^{(l)})\right)}. \tag{9}$$

This formulation depends only on the difference between scores, making it invariant to any constant shift applied to all scores, a desirable property since such shifts do not affect the induced ranking.

Maximum Composite Likelihood. Given the clean ranking r_0, we define the set of ordered pairs $\mathcal{P} = \{(k, l) : r_0^{(k)} < r_0^{(l)}\}$, which contains all pairs where operation k should be ranked higher than operation l. Pairs of operations sharing the same rank are excluded, as their relative order does not affect the resulting schedule. The composite likelihood of the pairwise comparisons over these pairs is:

$$L_{\text{CL}} = \prod_{(k,l)\in\mathcal{P}} P(k \succ l). \tag{10}$$

Taking the logarithm and substituting the definition of $P(k \succ l)$ yields:

$$\log L_{\text{CL}} = \sum_{(k,l)\in\mathcal{P}} \log P(k \succ l) = -\sum_{(k,l)\in\mathcal{P}} \log\left(1 + \exp\left(-(\hat{x}_0^{(k)} - \hat{x}_0^{(l)})\right)\right). \tag{11}$$

Negating the log-likelihood gives the final training loss:

$$\mathcal{L}_{\text{rank}} = -\log L_{\text{CL}} = \sum_{(k,l)\in\mathcal{P}} \log\left(1 + \exp\left(-(\hat{x}_0^{(k)} - \hat{x}_0^{(l)})\right)\right). \tag{12}$$

Connection to Pairwise Learning-to-Rank. We note that $\mathcal{L}_{\text{rank}}$ is equivalent to summing the binary cross-entropy loss over all ordered pairs, where each pair is treated as a binary classification problem (predicting which operation should be ranked higher). This connection places our objective within the well-studied family of pairwise learning-to-rank methods [58], inheriting favorable properties such as convexity with respect to score differences and smooth gradients that facilitate stable optimization. Unlike pointwise losses (e.g., MSE), which penalize deviations in absolute score values, this pairwise formulation focuses exclusively on the ordering structure, making it naturally suited for our rank-based diffusion framework.

B Active List Scheduling

In this section, we provide background on the hierarchy of schedule classes in JSSP and describe our active list scheduling algorithm in detail.

B.1 Schedule Classification

The set of all feasible schedules for a JSSP instance can be organized into a well-known hierarchy of increasingly restrictive classes [47, 59], as illustrated in Figure 4.

Semi-active Schedules. A schedule is *semi-active* if no operation can be started earlier without changing the processing order on any machine. In other words, every operation is left-shifted as much as possible given the fixed sequence of operations on each machine. Semi-active schedules eliminate trivially wasteful idle time but may still contain unnecessary gaps caused by suboptimal sequencing decisions, as illustrated in Figure 5a.

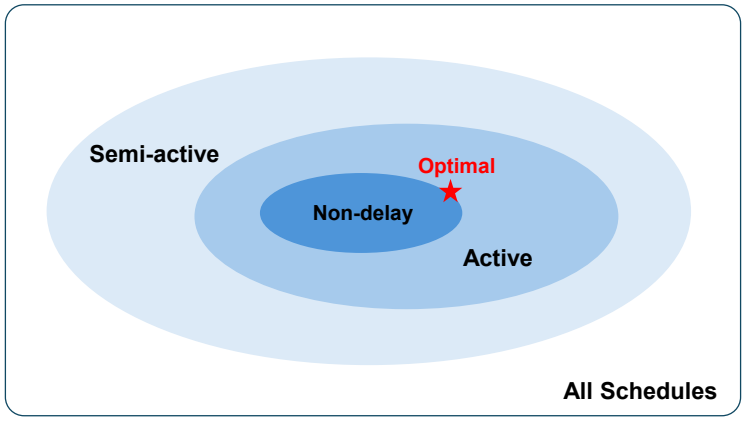
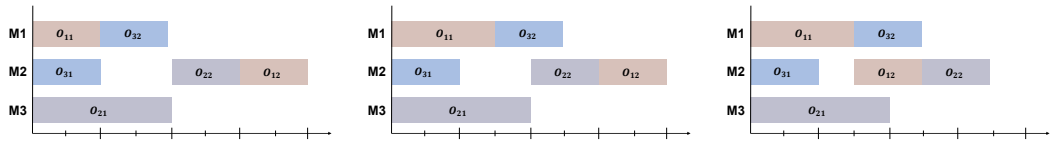


Figure 4: Hierarchy of schedule classes in JSSP. Each inner class is a strict subset of the outer class. The set of active schedules is the smallest class guaranteed to contain an optimal solution.

Active Schedules. A schedule is *active* if no operation can be completed earlier by changing the processing order without delaying any other operation. Active schedules form a strict subset of semi-active schedules, as they additionally require that no operation could have been started earlier by reordering operations on a machine. Crucially, it is well established that the set of active schedules is guaranteed to contain at least one optimal solution [47]. This makes the active schedule class the smallest standard class that is still guaranteed to include the optimum, providing a favorable trade-off between search space size and solution quality, as illustrated in Figure 5b.

Non-delay Schedules. A schedule is *non-delay* if no machine is kept idle when an operation is available for processing, as illustrated in Figure 5c. Non-delay schedules form the most restrictive class, as they never allow intentional machine idling. While this leads to the smallest search space, the optimal solution may not be a non-delay schedule, as it is sometimes beneficial to leave a machine temporarily idle to process a higher-priority operation that becomes available shortly after.



(a) Semi-active schedule. (b) Active schedule. (c) Non-delay schedule.

Figure 5: Examples of the three schedule classes.

B.2 Algorithm Details

Standard list scheduling, commonly used in one-shot methods, processes operations in priority order and appends each operation at the end of its designated machine's timeline, starting no earlier than its job predecessor's completion time.

Our active list scheduling extends standard list scheduling with a gap insertion mechanism, as summarized in Algorithm 3. The key difference from standard list scheduling lies in the gap search step: instead of directly appending each operation at the end of the machine timeline, we first scan for existing idle gaps that can accommodate the operation while respecting precedence constraints. This gap insertion mechanism produces active schedules, ensuring that the search space contains at least one optimal solution. As shown in Proposition 4.1, active list scheduling never yields a longer makespan than standard list scheduling for the same priority ordering, while the additional computational overhead of gap scanning is negligible in practice.

Algorithm 3: Active List Scheduling

Input: Processing times p, machine assignments μ, priority scores π
Output: Start times S for all operations
while *there exists an unscheduled operation* **do**
  $o \leftarrow$ highest-priority eligible operation;
  $t_{\text{ready}} \leftarrow$ completion time of o's job predecessor (or 0 if o is the first operation of its job);
  $\mu_o \leftarrow$ designated machine of o;
  for *each idle gap* $[g_s, g_e)$ *in machine* μ_o*'s timeline* **do**
    if $\max(g_s, t_{ready}) + p_o \leq g_e$ **then**
      Schedule o at time $\max(g_s, t_{\text{ready}})$;
      break;
    end
  end
  if o *is not yet scheduled* **then**
    Append o at the end of machine μ_o's timeline, starting no earlier than t_{ready};
  end
end

Proof of Proposition 4.1. Let operations be scheduled in priority order $o_1, o_2, \ldots, o_N$, and let s_k^{std} and s_k^{active} denote the start times of operation o_k under standard and active list scheduling, respectively. We show by induction that $s_k^{\text{active}} \leq s_k^{\text{std}}$ for all k. **Base case.** For $k = 1$, both methods schedule o_1 at the same time since the machine is empty. **Inductive step.** For $k > 1$, assume $s_l^{\text{active}} \leq s_l^{\text{std}}$ for all $l < k$. Since processing times are fixed, this implies $C_l^{\text{active}} \leq C_l^{\text{std}}$ for all $l < k$. The earliest feasible start time of o_k is determined by two constraints: the completion time of its job predecessor and the availability of its designated machine. Since both depend on the completion times of previously scheduled operations, they are not later under active list scheduling. Thus, even appending o_k at the end of the machine timeline under active list scheduling would yield a start time no later than s_k^{std}. Since active list scheduling additionally searches for idle gaps that may allow an even earlier start, we have $s_k^{\text{active}} \leq s_k^{\text{std}}$. Since $C_{\max} = \max_k(s_k + p_k)$, it follows that $C_{\max}^{\text{active}} \leq C_{\max}^{\text{std}}$. □

C Training Label Construction

Rank Extraction. For each training instance, we solve the JSSP using the OR-Tools CP-SAT solver [48] with a time limit of 1200 seconds, obtaining an optimal or near-optimal solution with start times for all operations. We convert these start times into ranks $r_0 \in \{1, 2, \ldots, N\}^N$ by ranking the operations in ascending order of their start times. Operations that start simultaneously receive the same rank, as they necessarily occupy different machines and their relative order does not affect the resulting schedule. These ranks serve as the training targets for the diffusion model.

Schedule Quality Guarantee. A natural question is whether decoding the extracted ranks via list scheduling recovers the quality of the original solution. We first show that this is guaranteed when the solution is optimal.

Proposition C.1. *Let $\{S_{i,j}^*\}$ denote the start times of an optimal solution, and let the priority ordering be defined by sorting operations in ascending order of $S_{i,j}^*$, with ties broken arbitrarily. Then the makespan produced by standard list scheduling under this ordering satisfies $C_{\max} = C_{\max}^*$.*

Proof. Tied operations start simultaneously in the optimal solution and therefore occupy different machines, so list scheduling produces the same schedule regardless of the order chosen among them. We first show that $C_{\max} \leq C_{\max}^*$ by induction on the scheduling order. **Base case.** The first operation in priority order has the smallest start time in the optimal solution. Since list scheduling assigns it to the earliest feasible time, it starts no later than in the optimal solution. **Inductive step.** Assume all previously scheduled operations satisfy $S_k \leq S_k^*$. Consider the next operation O_k in priority order. Its start time is constrained by: (1) its job predecessor, which, by the inductive hypothesis, finishes no later than in the optimal solution; and (2) the preceding operations on the same machine,

which, by the inductive hypothesis, also finish no later. Since neither constraint is tighter than in the optimal solution, list scheduling places O_k at the earliest feasible time, yielding $S_k \leq S_k^*$. Since every operation starts no later than in the optimal solution, $C_{\max} \leq C_{\max}^*$. Since $C_{\max}^*$ is optimal, no feasible schedule can achieve a strictly smaller makespan, so $C_{\max} = C_{\max}^*$. □

More generally, when the solver returns a suboptimal feasible solution, the same inductive argument shows that list scheduling produces a makespan no worse than the original solution. Combined with Proposition 4.1, active list scheduling further guarantees:

$$C_{\max}^{\text{active}} \leq C_{\max}^{\text{std}} \leq C_{\max}^{\text{sol}}, \tag{13}$$

where $C_{\max}^{\text{sol}}$ denotes the makespan of the solver output. This ensures that the training labels, when decoded through active list scheduling, always yield schedules at least as compact as the original solver solutions.

D Denoiser & Implementation Details

D.1 Denoiser Architecture

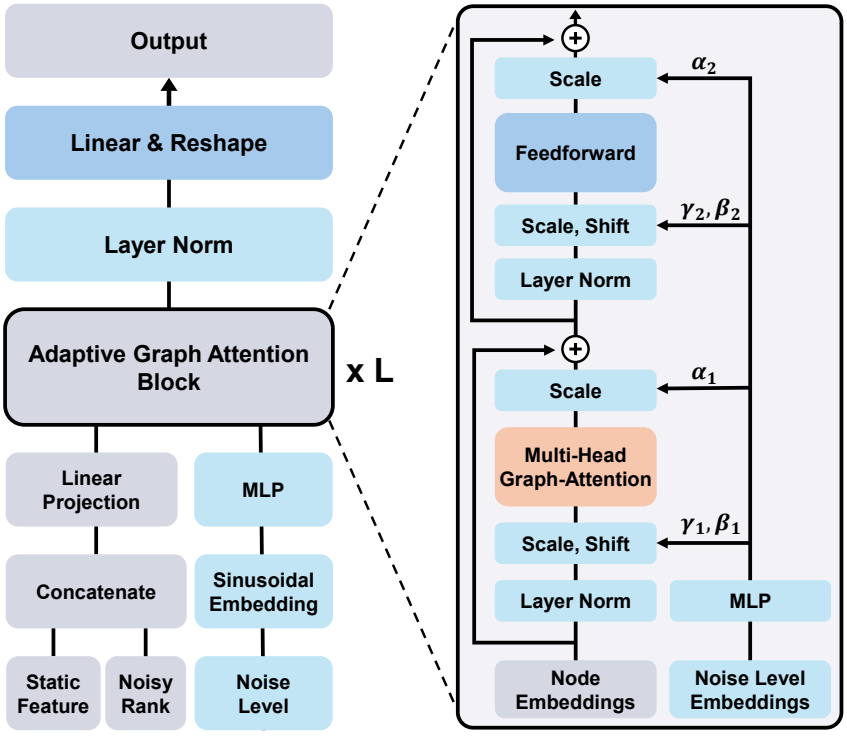


Figure 6: Detailed architecture of the SchedDiff denoising network.

The denoising network takes as input the disjunctive graph G, noisy ranks r_σ, and noise level σ, and predicts continuous priority scores $\hat{x}_0 \in \mathbb{R}^N$. The architecture consists of three components: an input projection layer, L stacked AGA blocks conditioned on a sinusoidal noise-level embedding, and a final output layer. The network uses a hidden dimension of 128, 6 AGA blocks with 4 attention heads, and a dropout rate of 0.1.

Input Features. Each operation node $O_{i,j}$ is represented by 6 static features concatenated with the current noisy rank, yielding a 7-dimensional input vector. The static features are:

1. $p_{i,j}$: processing time of the operation,
2. $\tau_{i,j}$: earliest start time, computed as the cumulative processing time of all preceding operations in the same job ($\sum_{j'<j} p_{i,j'}$),

3. $p_{i,j}^{\text{tail}}$: total remaining processing time in the same job ($\sum_{j'>j} p_{i,j'}$),
4. $N_{i,j}^{\text{tail}}$: number of remaining operations in the same job ($m - j$),
5. $\eta_{i,j}$: processing time ratio, defined as $p_{i,j}/(p_{i,j} + p_{i,j}^{\text{tail}})$,
6. $\lambda_{i,j}$: machine load ratio, defined as the total processing time on the assigned machine divided by the total processing time across all machines.

Time-related features ($p_{i,j}$, $\tau_{i,j}$, $p_{i,j}^{\text{tail}}$) are normalized by the maximum processing time in the instance, and $N_{i,j}^{\text{tail}}$ is normalized by the number of machines. The noisy rank $r_\sigma^{(k)}$ is standardized to zero mean and unit variance within each instance, ensuring a consistent input scale across problem sizes. It is then concatenated with the static features and projected to the hidden dimension via a linear layer. For the flexible JSSP experiments, where each operation can be assigned to multiple machines, we replace processing times with their average over all eligible machines and compute all derived features accordingly. We also connect operations with disjunctive edges whenever their sets of eligible machines overlap.

Edge Types. The disjunctive graph contains three types of edges, each represented by a learnable embedding of dimension 32:

- *Predecessor* ($O_{i,j} \to O_{i,j+1}$): directed edges from each operation to its immediate successor within the same job,
- *Successor* ($O_{i,j+1} \to O_{i,j}$): reverse edges of predecessor edges,
- *Disjunctive* ($O_{i,j} \leftrightarrow O_{i',j'}$), $\mu_{i,j} = \mu_{i',j'}$: bidirectional edges between all pairs of operations that share the same machine.

These edge embeddings are incorporated into the graph attention computation, allowing the model to distinguish between different types of structural relationships.

D.2 Implementation Details

Following [42], the noise level during training is sampled from a log-normal distribution $\ln \sigma \sim \mathcal{N}(P_{\text{mean}}, P_{\text{std}}^2)$, clipped to $[\sigma_{\min}, \sigma_{\max}]$. At inference time, we use the polynomial noise schedule:

$$\sigma_t = \left(\sigma_{\min}^{1/\rho} + \frac{t-1}{T-1}\left(\sigma_{\max}^{1/\rho} - \sigma_{\min}^{1/\rho}\right)\right)^{\rho}, \tag{14}$$

where $t = 1, \ldots, T$ indexes the denoising steps, with $\sigma_T = \sigma_{\max}$ and $\sigma_1 = \sigma_{\min}$. We adjust the diffusion hyperparameters to better suit the priority-based scheduling domain. Table 6 summarizes all hyperparameters used for training and diffusion, with training taking approximately 10 hours.

Hyperparameter	**Value**
Optimizer	Adam
Learning rate	10^{-4}
Batch size	50
Training epochs	4,000
P_{mean}	2.3
P_{std}	1.5
$\sigma_{\min}$	0.5
$\sigma_{\max}$	200
ρ	7

Table 6: Training and diffusion hyperparameters.

E Additional Experimental Results

E.1 LA Benchmark Results

Table 7 presents the performance gap on the LA benchmark, which includes diverse problem sizes from 10×5 to 30×10 with processing times drawn from $U(5, 99)$, slightly different from the training distribution. All sizes except 15×15 are unseen during training. SchedDiff achieves the best average gap (**1.9%**) among all learning-based methods, outperforming MP-ASIL (2.3%), SLIM (2.5%), and TBGAT (2.3%). The advantage is particularly evident on harder instances such as 15×15 and 20×10, where SchedDiff improves upon the best baselines by a notable margin. Furthermore, SchedDiff substantially outperforms one-shot baselines, reducing the average gap by more than fourfold compared to Jeon et al. (7.7%) and Qi et al. (8.7%), further demonstrating the benefit of iterative refinement over single-pass inference.

Table 7: Experimental results on the LA benchmark. • indicates problem sizes unseen during training. **Bold** highlights the best result among learning-based methods.

Method	10×5 •	10×10 •	15×5 •	15×10 •	15×15	20×5 •	20×10 •	30×10 •	Avg.
Gurobi* (3600s) [52]	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
OR-Tools* (3600s) [48]	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
L2D* (G) [6]	14.3%	23.7%	7.8%	27.2%	27.1%	6.3%	24.6%	8.4%	17.4%
L2D* (K=128) [6]	8.8%	10.4%	2.8%	16.2%	17.4%	3.1%	18.3%	6.8%	10.6%
SN* (G) [13]	12.1%	11.9%	2.7%	14.6%	16.1%	3.6%	15.7%	3.1%	10.0%
SLIM* (K=512) [7]	**1.1%**	2.5%	**0.0%**	5.0%	5.6%	**0.0%**	5.6%	**0.0%**	2.5%
MP-ASIL* (K=512) [53]	1.3%	2.1%	**0.0%**	4.3%	5.5%	**0.0%**	4.9%	**0.0%**	2.3%
BOPO* (K=512) [20]	2.1%	2.1%	**0.0%**	4.9%	4.9%	**0.0%**	4.6%	**0.0%**	2.3%
L2S*-500 [22]	2.1%	4.4%	**0.0%**	6.4%	7.3%	**0.0%**	7.0%	0.2%	3.4%
TBGAT*-500 [23]	2.1%	**1.8%**	**0.0%**	**3.6%**	5.5%	**0.0%**	5.0%	**0.0%**	2.3%
Jeon et al. (K=512) [8]	9.1%	8.4%	4.0%	10.3%	12.9%	3.0%	12.2%	2.0%	7.7%
Qi et al. (K=512) [9]	11.9%	9.7%	5.1%	9.9%	14.9%	4.9%	11.9%	1.6%	8.7%
SchedDiff (K=5, T=200)	3.4%	3.0%	**0.0%**	**3.6%**	**3.3%**	**0.0%**	**2.1%**	**0.0%**	**1.9%**

E.2 Large-Scale Synthetic Instances

To further evaluate the scalability of SchedDiff, we generate synthetic JSSP instances of sizes 200×20, 300×20, 400×20, 500×20, and 800×20, with processing times sampled uniformly from $[1, 99]$. These instances are 10–40$\times$ larger than the training sizes (15×15, 20×15, 20×20), providing a stringent test of size generalization. We compare SchedDiff against OR-Tools (with a 600-second time limit) and SLIM. For SchedDiff, we use $K = 1$ and $T = 200$ to evaluate single-trajectory performance. We report the average gap and total inference time over 10 generated instances for each size.

As shown in Table 8, SchedDiff maintains strong performance on instances up to $40\times$ larger than the training set, demonstrating robust size generalization without any retraining. While SLIM achieves comparable gaps, SchedDiff is consistently 3–7$\times$ faster across all sizes. Moreover, SLIM encounters out-of-memory errors on 800×20, whereas SchedDiff scales to all tested sizes.

Table 8: Experimental results on large-scale synthetic JSSP instances. Gaps are computed against OR-Tools (600s). 'm' and 'h' denote minutes and hours. – indicates out-of-memory.

Method	200×20		300×20		400×20		500×20		800×20	
	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time
OR-Tools (600s) [48]	0.0%	(1.7h)	0.0%	(1.7h)	0.0%	(1.7h)	0.0%	(1.7h)	0.0%	(1.7h)
SLIM (K=512) [7]	**-0.5%**	(5.0m)	-1.1%	(11.2m)	**-0.1%**	(25.0m)	0.1%	(52.8m)	–	–
SchedDiff (K=1, T=200)	**-0.5%**	(1.3m)	**-1.2%**	(2.9m)	**-0.1%**	(5.0m)	**0.0%**	(8.1m)	**0.0%**	(17.9m)

We also evaluate SLIM with $K = 200$ to match the number of candidate schedules generated by SchedDiff ($K = 1$, $T = 200$). As shown in Table 9, reducing K for SLIM decreases its inference time but increases its performance gap. SchedDiff still achieves lower gaps across all four evaluated sizes while remaining approximately 2.1–3.2$\times$ faster.

Table 9: Experimental results on large-scale synthetic JSSP instances with matched numbers of candidate schedules. Gaps are computed against OR-Tools (600s). 'm' and 'h' denote minutes and hours. – indicates out-of-memory.

Method	200×20		300×20		400×20		500×20		800×20	
	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time
OR-Tools (600s) [48]	0.0%	(1.7h)	0.0%	(1.7h)	0.0%	(1.7h)	0.0%	(1.7h)	0.0%	(1.7h)
SLIM (K=200) [7]	-0.1%	(2.8m)	-0.8%	(6.2m)	0.2%	(13.2m)	0.3%	(25.6m)	–	–
SchedDiff (K=1, T=200)	**-0.5%**	(1.3m)	**-1.2%**	(2.9m)	**-0.1%**	(5.0m)	**0.0%**	(8.1m)	**0.0%**	(17.9m)

E.3 Ablation Study on AdaLN-Zero Conditioning

To validate the effectiveness of AdaLN-Zero conditioning, we ablate this component by replacing it with a simpler baseline in which the noise level embedding σ_{emb} is concatenated directly to the node features at each block, leaving the layer normalization and residual structure unchanged. All experiments are conducted with K=5 and T=200. Table 10 reports the performance gap on the TA benchmark across four problem sizes. Applying AdaLN-Zero consistently improves performance across all problem sizes, validating our architectural choice of using adaptive modulation for noise-level conditioning in SchedDiff. We attribute this to the ability of AdaLN-Zero to dynamically scale and shift intermediate representations in a noise-level-dependent manner, enabling the model to more effectively adapt its computation across different denoising steps. In contrast, simple concatenation provides only an additive input signal, without directly modulating the normalized intermediate representations.

Table 10: Experimental results for different noise conditioning strategies on the TA benchmark.

Conditioning	15 × 15	30 × 15	50 × 20	100 × 20
Concatenation	5.7% (+1.3%)	5.6% (+1.3%)	7.0% (+1.0%)	3.8% (+2.3%)
AdaLN-Zero (ours)	**4.4%**	**4.3%**	**6.0%**	**1.5%**

E.4 Denoising Trajectory Analysis

Figure 7 visualizes the makespan at each denoising step for representative instances of different sizes. Each gray line represents an independent trajectory starting from random priorities, the dashed line shows the average trend, and the red line highlights the trajectory that achieves the global minimum. Several observations are noteworthy. First, the iterative denoising process improves solution quality across all trajectories, with makespans showing an overall decreasing trend despite local fluctuations along individual trajectories. Second, the trajectories exhibit significant diversity in their paths, reflecting the stochastic nature of the refinement process. Third, while the average trend converges smoothly, individual trajectories vary in their final quality, highlighting the benefit of running multiple samples to increase the chance of finding high-quality solutions.

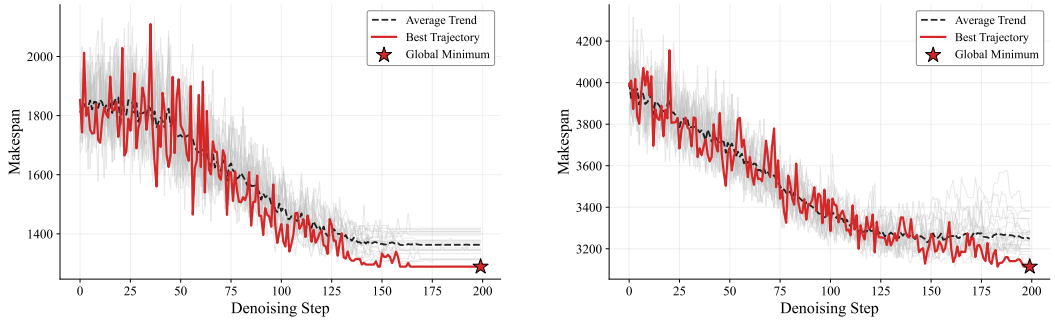


Figure 7: Denoising trajectories on representative TA instances of size 15 × 15 (left) and 50 × 20 (right), with $K = 20$ and $T = 200$.

E.5 Qualitative Analysis of Predicted Priority Maps

Figure 8 compares the priority maps of the optimal solution and SchedDiff on a representative 15×15 instance. Each entry (i, j) represents the normalized priority rank assigned to the j-th operation of job i, with higher values (brighter color) indicating higher priority. As shown, the two maps exhibit similar patterns, demonstrating that SchedDiff successfully learns the priority structure of optimal solutions through iterative denoising. This confirms that the rank-aware forward process and pairwise ranking objective effectively guide the model toward the priority structure of optimal solutions.

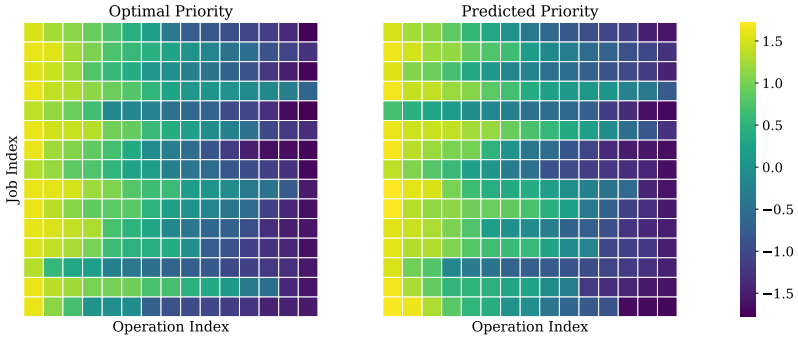


Figure 8: Priority map comparison between the optimal solution and SchedDiff.

E.6 Performance under Reduced Labeling Budget

SchedDiff relies on solver-generated labels for training, which incurs a labeling cost. To verify that SchedDiff maintains its performance under a reduced labeling budget, we train a variant using 1,000 instances per size (instead of 2,000) with a 60-second solver time limit (instead of 1,200 seconds), reducing the total labeling cost by approximately $40\times$. As shown in Table 11, even under this substantially cheaper labeling setup, SchedDiff still outperforms all existing learning-based baselines on average.

Table 11: Experimental results on the TA benchmark with a reduced labeling budget. † indicates training with a reduced labeling budget.

Method	15×15	20×15	20×20	30×15 •	30×20 •	50×15 •	50×20 •	100×20 •	Avg.
Gurobi* (3600s) [52]	0.1%	3.2%	2.9%	10.7%	13.2%	12.2%	13.6%	11.0%	8.4%
OR-Tools* (3600s) [48]	0.1%	0.2%	0.7%	2.1%	2.8%	3.0%	2.8%	3.9%	2.0%
L2D* (G) [6]	26.0%	30.0%	31.6%	33.0%	33.6%	22.4%	26.5%	13.6%	27.1%
L2D* (K=128) [6]	17.1%	23.7%	22.6%	24.4%	28.4%	17.1%	20.4%	10.3%	20.5%
SN* (G) [13]	15.3%	19.4%	17.2%	19.1%	23.7%	13.9%	13.5%	6.7%	16.1%
RASCL* (G) [18]	14.3%	16.5%	17.3%	18.5%	21.5%	12.2%	13.2%	5.9%	14.9%
SI GD* (G) [19]	9.6%	9.9%	11.1%	9.5%	13.8%	**2.7%**	**6.7%**	1.7%	8.1%
SLIM* (K=512) [7]	6.5%	8.8%	9.0%	10.6%	12.7%	4.9%	7.6%	2.1%	7.8%
MP-ASIL* (K=512) [53]	6.8%	8.5%	8.7%	10.4%	12.8%	4.2%	7.0%	**1.0%**	7.4%
BOPO* (K=512) [20]	6.3%	8.3%	9.1%	10.3%	12.2%	4.9%	7.4%	1.4%	7.5%
L2S*-500 [22]	9.3%	11.6%	12.4%	14.7%	17.5%	11.0%	13.0%	7.9%	12.2%
TBGAT*-500 [23]	8.0%	9.9%	10.0%	13.3%	16.4%	9.6%	11.9%	6.4%	10.7%
Jeon et al. (K=512) [8]	14.9%	15.7%	16.7%	19.0%	23.4%	14.1%	18.6%	13.0%	16.9%
Qi et al. (K=512) [9]	15.8%	16.5%	17.6%	18.8%	24.0%	13.9%	17.8%	11.9%	17.0%
SchedDiff (K=1, T=200)†	**5.3%**	**5.8%**	**6.4%**	**6.9%**	**9.7%**	3.4%	7.5%	3.4%	**6.0%**
SchedDiff (K=5, T=200)†	**4.4%**	**4.9%**	**5.9%**	**6.5%**	**9.0%**	2.9%	7.0%	3.0%	**5.5%**

F Broader Impact

This paper introduces the first diffusion-based framework for JSSP. By enabling efficient and high-quality scheduling through learned priority refinement, SchedDiff has the potential to enhance productivity across manufacturing, logistics, and cloud computing. On the other hand, our method introduces additional complexity compared to classical dispatching rules, making it more difficult to trace how specific scheduling decisions are made. Ensuring reliability and transparency of such

generative scheduling systems in real-world deployment remains an important consideration for future work.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and introduction clearly state our main contributions and scope.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: We discuss limitations of our work in Section 6.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

   Answer: [Yes]

Justification: The proof of Proposition 4.1 is provided in Appendix B.2, and the proof of Proposition C.1 is provided in Appendix C.

Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: We provide all implementation details in Appendix D.

   Guidelines:

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

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

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

Answer: [Yes]

Justification: The source code is provided in the supplemental material, and we use publicly available benchmarks.

Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: We provide all training and evaluation details in Section 5.1 and Appendix D.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [No]

   Justification: We report average performance gaps on fixed public benchmarks without error bars, following standard practice in the JSSP literature.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: We specify computational resources in Section 5.1.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: We have reviewed and conform with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: We discuss the broader impacts of our work in Appendix F.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: We properly cite all benchmarks and models used, and respect their licenses and terms of use.

    Guidelines:

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

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

    Answer: [N/A]

    Justification: We use publicly available benchmarks.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

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

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

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing nor research with human subjects.

    Guidelines:

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

16. **Declaration of LLM usage**

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

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.