DAG SCHEDULING BASED ON DIFFUSION MODEL WITH PARTIAL-ORDER FEASIBILITY GUARANTEE

Anonymous authors
Paper under double-blind review

ABSTRACT

A central question for learning-based non-autoregressive directed acyclic graph (DAG) scheduling is how to represent and learn the solutions' global ordering structure, while guaranteeing that the solutions satisfy the partial-order constraints in the DAG. Existing approaches commonly predict the distribution of itemwise priorities, leaving the solution structure and partial-order feasibility to a post-processing feasibility repair procedure. In this paper, we propose a DAG scheduling framework based on a diffusion model in the doubly stochastic matrix (DSM) space with partial-order feasibility guarantee. The DSMs directly provide a continuous representation of scheduling solutions. We promote and ensure the feasibility of the DAG scheduling solutions generated by our framework from both the perspective of training and inference. At inference, we propose a feasible solution decoding procedure for DAG scheduling that can directly produce feasible solutions without additional post-processing repair, which applies our Birkhoff-von Neumann (BvN)-based feasible solution extraction algorithm to the diffusion model's DSM output. The efficiency of the decoding is supported by a successor interpretation of permutation matrices. At training, we further design a learning method guided by feasible components, to promote the DAG scheduling solutions' feasibility and quality. Comparison experiments and ablation studies evaluate the solution optimization performance of our framework across different DAG scheduling tasks.

1 INTRODUCTION

Directed acyclic graph (DAG) scheduling is a class of combinatorial optimization problems (COPs) that seeks a low-cost solution, represented by an ordering of a finite set of items, whose precedence relationships must satisfy the partial-order constraints represented by the DAG. Such problems arise in constrained ranking, scheduling, routing, and related sequencing applications (Van Zuylen & Williamson, 2009).

Existing autoregressive methods, including reinforcement learning (RL)-based sequential methods, construct a solution for DAG scheduling by selecting one item at a time (Song et al., 2023; Gu et al., 2025; Wang et al., 2021). This procedure requires repeated model evaluations and has high computational complexity. More recently, non-autoregressive methods for DAG scheduling directly predict the probability distribution of itemwise priorities of the solution in one forward propagation of the neural network (Gagrani et al., 2022; Jeon et al., 2023; Qi et al., 2025). But the predicted probability must subsequently be converted into a discrete ordering that represents a candidate solution through probability distribution estimation (Bhat & Sen, 2026), and the solutions are usually not guaranteed to be feasible and require a post-processing feasibility repair. This raises a question for non-autoregressive DAG scheduling: how to represent the solution in a continuous space, and how to generate and learn a solution while satisfying the partial-order constraints.

In this paper, we propose a DAG scheduling framework based on a diffusion model in the DSM space with partial-order feasibility guarantee. In this framework, a graph-conditioned denoiser and a few-step reverse process within the DSM space enable non-autoregressive neural generation of feasible and high-quality DAG scheduling solutions. The DSMs perform as a continuous representation of scheduling solutions. At inference, to guarantee feasible scheduling solutions from the DSM output without additional post-processing repair, we propose a feasible solution decoding procedure

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

from DSMs subject to partial-order constraints. This decoding procedure relies on a successor interpretation of permutation matrices and a BvN-based feasible solution extraction algorithm. For the training of this framework, we design a learning method guided by feasible components, in addition to supervision from reference solutions. We construct the feasible component of the diffusion output, and minimizing the difference between the DSM output and this component guides learning toward the feasible polytope. The difference between the denoised DSM output and its feasible component provides the feasibility guidance signal.

Our contributions are as follows:

1. We propose a DAG scheduling framework based on a diffusion model in the DSM space with partial-order feasibility guarantee. The DSMs directly provide a continuous representation for scheduling solutions. A graph-conditioned denoiser and a few-step reverse process enable generating feasible and high-quality DAG scheduling solutions in a non-autoregressive neural network manner.
2. We propose a feasible solution decoding procedure from DSMs for DAG scheduling. The decoding procedure directly produces feasible DAG scheduling solutions from the diffusion model's DSM output, without additional post-processing repair. The decoding procedure is based on our proposed BvN-based feasible solution extraction. To support this extraction, we introduce a successor interpretation of the permutation matrix, rather than the conventional positional interpretation, enabling feasible solutions to be effectively constructed.
3. We design a learning method guided by feasible components at training to promote the feasibility and quality of DAG scheduling solutions. We formalize the challenge of learning a feasible DSM for DAG scheduling, and then guide the learning by reducing the difference between the diffusion output and its feasible component, in addition to the supervision from reference solutions. The feasible component is constructed through a normalized weighted combination of the extracted matrices.

We evaluate the framework on multiple DAG scheduling problem families, measuring solution quality, feasibility, and constraint satisfaction.

2 PRELIMINARY

Problem formulation. We consider DAG scheduling over a set V containing n items. Each item is represented by a node in the constraint DAG and corresponds to a task in the scheduling applications. A scheduling solution is represented by an ordering π of V. The partial-order constraint set C over V is represented by a DAG $G_C = (V, E_C)$, where $(i, j) \in E_C$ requires i to precede j in the ordering. An arbitrary ordering need not satisfy the precedence constraints. An ordering constitutes a feasible solution if it satisfies all precedence constraints in C, equivalently, if it is a topological ordering of G_C. We denote the feasible set of orderings by Π_C. Without loss of generality, we preprocess each constraint DAG to have a unique source 0 and a unique sink $n-1$. If the original DAG has multiple sources and sinks, a dummy source is connected to every source, and every sink is connected to a dummy sink. After this augmentation, n denotes the total number of nodes. Every feasible solution ordering starts at 0 and ends at $n-1$. Given an application-specific cost g, we seek a feasible solution π satisfying Eq. (1). Here, $\mathrm{pos}_\pi(i)$ is the position of item i in the ordering π.

$$\begin{aligned} \min_{\pi} \quad & g(\pi) \\ \text{subject to} \quad & \mathrm{pos}_\pi(i) < \mathrm{pos}_\pi(j), \qquad \forall\, (i,j) \in E_C, \end{aligned} \tag{1}$$

Permutation matrix and positional interpretation. A permutation matrix, denoted generically by H, is a binary square matrix with exactly one 1 in each row and each column. Under the conventional positional interpretation, H is instantiated as the matrix P_π representing an ordering π through $(P_\pi)_{i,a} = \mathbb{I}[\mathrm{pos}_\pi(i) = a]$. That is, $(P_\pi)_{i,a} = 1$ indicates that item i occupies position a in π. An ordering π uniquely determines its positional permutation matrix P_π, and P_π uniquely determines the ordering π. Under the successor interpretation introduced later, an ordering similarly determines its successor permutation matrix Q_π, and Q_π determines the ordering given the designated source and sink.

DSM. A DSM is a nonnegative square matrix whose rows and columns all sum to 1. Such matrices form the Birkhoff polytope $\mathcal{B}_n$ of $n \times n$ square matrices. This Birkhoff polytope is the convex hull of all $n \times n$ permutation matrices. By the BvN theorem (Dufossé & Uçar, 2016), every $S \in \mathcal{B}_n$ admits a decomposition $S = \sum_{k=1}^{m} \lambda_k H_k$, where each H_k is a permutation matrix, $\lambda_k \geq 0$, and $\sum_{k=1}^{m} \lambda_k = 1$.

Sinkhorn normalization. An arbitrary network output may not satisfy doubly stochasticity. Sinkhorn normalization can convert a square matrix to a DSM by scaling each row and column (Cuturi, 2013; Mena et al., 2018). In practice, a finite number of Sinkhorn normalization iterations gives an approximately doubly stochastic output.

3 A DAG Scheduling Framework Based on a Diffusion Model

3.1 Motivation

Some existing non-autoregressive solvers optimize distributions over discrete solutions represented by orderings using policy-gradient objectives, which require stochastic estimation over sampled solutions. Our formulation instead learns a continuous denoising process from reference solutions, providing a different training and generation route. The DSM explicitly models ordering structure in a continuous representation, rather than constructing an ordering from priority distribution with a pairwise or listwise ranking model. However, DSMs still leave two obstacles to partial-order feasibility for DAG scheduling, motivating the decoding and training methods below.

The efficiency of extracting a feasible solution motivates our successor interpretation and the BvN-based feasible solution extraction without post-processing repair in Section 3.3. Under the conventional positional interpretation of permutation matrices, it is not trivial to find such a matching. The raw ordering corresponding to a random extraction need not satisfy the partial-order constraints. When partial-order feasibility is required, extracting a P_{π_k} from the current residual $R^{(k-1)} = S - \sum_{r=1}^{k-1} \lambda_r P_{\pi_r}$ at the k-th iteration of BvN decomposition is a bipartite perfect matching. We need to match items to positions, with each assignment (i, a) satisfying the residual support $R_{i,a}^{(k-1)} > 0$. We must satisfy both residual support and partial-order constraints. As the residual becomes sparse during the decomposition, assigning an item to a supported position becomes harder. A straightforward positional extraction procedure may require backtracking or enumeration over candidate matchings. Therefore, we need an effective extraction mechanism that achieves partial-order feasibility. Under our successor interpretation and the BvN-based feasible solution extraction, the position assignment is omitted, and we can directly extract a feasible solution from the DSM residual without any backtracking.

The geometric inconsistency between DSMs and the feasibility of DAG schedules motivates learning guided by feasible components. Under the positional interpretation, the feasible polytope of the permutation matrices corresponding to DAG scheduling solutions is $\mathcal{P}_C^{\text{pos}} := \text{conv}\{P_\pi : \pi \in \Pi_C\} \subseteq \mathcal{B}_n$. By the BvN theorem, a DSM is only guaranteed to be in $\mathcal{B}_n$, not in $\mathcal{P}_C^{\text{pos}}$. In general, a DSM may lie outside the feasible polytope and thus cannot be feasibly decomposed. For example, consider $V = \{0, 1\}$ with $C = \{0 \prec 1\}$. $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ is the only feasible permutation matrix, whereas there is a DSM $S = \begin{pmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. It is impossible to produce an all-feasible decomposition of S. The same geometric mismatch applies to the successor representation introduced below. This motivates learning to drive the DSMs toward the corresponding feasible polytope, as developed in Section 3.4.

3.2 Framework Overview

We propose a DAG scheduling framework based on a diffusion model, as illustrated in Fig. 1. A few-step Gaussian diffusion process with a graph-conditioned denoiser, which is based on Denoising Diffusion Implicit Models (DDIM), represents denoising states in DSM space. We first propose a feasible solution decoding procedure that produces solutions satisfying partial-order constraints without post-processing repair. We then propose a learning method guided by feasible components,

which reduces the difference between the framework's DSM output and its feasible component, in addition to supervision from reference solution targets. Implementation details are deferred to Appendix B.

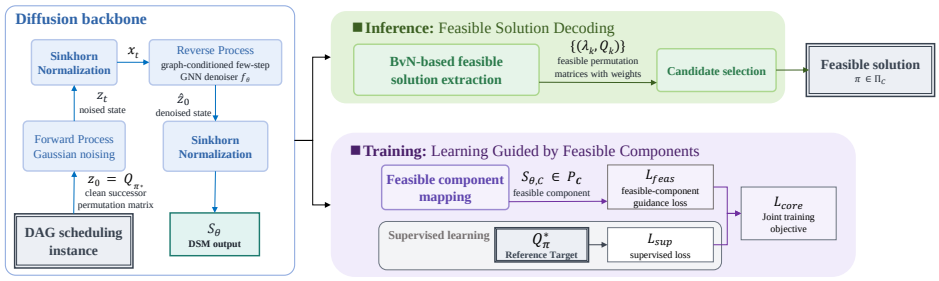


Figure 1: Overview of the proposed DAG scheduling framework. A graph-conditioned diffusion model generates DSMs as the solution representation, followed by the feasible solution decoding at inference and feasible-component-guided learning during training.

We write $Q_\pi \in \{0,1\}^{(n-1)\times(n-1)}$ for the successor permutation matrix, as detailed in Section 3.3.1. The corresponding feasible polytope is $\mathcal{P}_C^{\text{succ}} := \text{conv}\{Q_\pi : \pi \in \Pi_C\} \subseteq \mathcal{B}_{n-1}$. The diffusion process is described in Eq. (2). π^{ref} is a feasible reference solution and its corresponding successor permutation matrix is $Q^{\text{ref}} := Q_{\pi^{\text{ref}}}$. We use Q^{ref} as the supervision target in training. G_C is the constraint graph. The diffusion backbone adds Gaussian noise to the perturbed state, as shown in Eq. (3), where F is the conditional features obtained by a graph encoder based on graph neural network (GNN), $\bar{\alpha}_t$ is determined by the noise schedule, and $\mathcal{S}_\tau$ is the Sinkhorn operator with temperature τ. The conditional denoiser $f_\theta(x_t, t, F, G_C)$, which is based on GNN, predicts $\widehat{z}_0$, and a finite number of Sinkhorn normalization iterations output $S_\theta = \mathcal{S}_\tau(\widehat{z}_0)$, which is approximately DSM and therefore lies close to $\mathcal{B}_{n-1}$.

$$z_t \xrightarrow{\mathcal{S}_\tau} x_t \xrightarrow{f_\theta(\cdot,t,F,G_C)} \widehat{z}_0 \xrightarrow{\mathcal{S}_\tau} S_\theta \tag{2}$$

$$z_t = \sqrt{\bar{\alpha}_t} Q^{\text{ref}} + \sqrt{1-\bar{\alpha}_t}\,\epsilon, \qquad \epsilon \sim \mathcal{N}(0, I), \qquad x_t = \mathcal{S}_\tau(z_t), \tag{3}$$

At inference, the decoder applies the BvN-based feasible solution extraction algorithm to S_θ and selects a feasible solution without applying an additional post-processing repair (Section 3.3.2). The forward process, the reverse process, and the feasible solution decoding procedure are illustrated in Fig. 3 in Appendix B. For training, the same extraction algorithm obtains permutation matrices satisfying constraints in C, each with a positive weight. Feasible component mapping forms the training target $S_{\theta,C} \in \mathcal{P}_C^{\text{succ}}$ as the normalized weighted combination of the extracted matrices when their total weight is positive. Minimizing the difference between S_θ and this component guides the output toward the successor feasible polytope, in addition to supervision from Q^{ref} (Section 3.4).

3.3 Feasible Solution Decoding of DSM Outputs

Here, inference denotes the complete test-time process, including denoising, decoding, and top-K solution selection. Decoding maps the DSM output of one trajectory to a discrete feasible solution. We first describe the successor interpretation and extraction algorithm, then the decoder used at inference.

3.3.1 BvN-Based Feasible Solution Extraction

The extraction algorithm returns feasible permutation matrices with positive weights from a DSM. During decoding, these matrices represent the feasible solutions from which one is selected. During training, the same extracted matrices are used to construct feasible components for learning. We first describe the successor interpretation that supports this extraction, then the extraction procedure shared by decoding and training.

Successor interpretation. To effectively extract a feasible permutation matrix from S_θ, we introduce a successor interpretation of the permutation matrix when representing an ordering. The constraint graph G_C has a unique source 0 and a unique sink $n-1$. Then, every feasible ordering $\pi = (\pi_1, \ldots, \pi_n)$ starts at 0 and ends at $n-1$. For the directed linked list of π, every node except the sink has exactly one successor, and every node except the source has exactly one predecessor. Let A_π be the adjacency matrix of this directed linked list; its upper-right block is $Q_\pi = (A_\pi)_{\{0,\ldots,n-2\},\{1,\ldots,n-1\}}$. Every row and column of Q_π contains one 1: it is exactly a permutation matrix. Under the successor interpretation, the generic permutation matrix H is instantiated as Q_π. We use Q_π as the interpretation of the permutation matrix in our method. For active matrices, row labels are $i \in \{0, \ldots, n-2\}$ and column labels are $j \in \{1, \ldots, n-1\}$; column label j corresponds to storage index $j-1$. We use these node labels in entries such as $(Q_\pi)_{i,j}$, $S_{i,j}$, and $R_{i,j}$. We illustrate an example successor permutation matrix of a DAG instance with $n = 7$ and its distinction from the positional permutation P_π in Fig. 2 in Appendix B.4.

Extracting one feasible permutation matrix. Under the successor interpretation of the permutation matrix, in order to extract a feasible Q_{π_k} from a DSM S, we should construct a complete source-to-sink topological chain in the positive residual support, instead of finding a bipartite matching.

Firstly, consider extracting a feasible Q_π from a DSM S, where every entry of S is nonzero. In this case, extracting a feasible Q_π is equivalent to finding any topological sorting. We precompute a node-relation matrix $M \in \{0,1\}^{n\times n}$ that allows incomparable nodes and direct DAG successors:

$$M_{i,j} = \begin{cases} 1, & (i,j) \in E_C \text{ or } (i \neq j,\ i \not\prec_C j,\ j \not\prec_C i), \\ 0, & \text{otherwise}, \end{cases} \qquad i,j \in \{0,\ldots,n-1\}, \tag{4}$$

Here $i \prec_C j$ means that G_C contains a directed path from i to j. Entries of M are indexed by the same node labels used for active matrices S and R. A feasible Q_π can be constructed by an $O(n^2)$ walkthrough in M: starting at 0, when the current row is i, select an unvisited j with $M_{i,j} = 1$ and $d_U(j) = 0$, then continue from that j-th row, where $d_U(j) = 0$ is the predecessor-availability condition formally defined in Appendix B.4. Appendix B.4 first gives this preliminary construction for the special case with no zero residual entries, then the general extraction procedure and its analysis. Then we consider extracting a feasible Q_π from the extraction residual R or S containing zero entries. In this case, $R_{i,j} = 0$ means that it is impossible to extract a Q_π with $(Q_\pi)_{i,j} = 1$. In other words, successor relationship (i,j) is prohibited during the walkthrough in M. To extract a feasible Q_π, we conduct the same availability-constrained walkthrough in M, additionally requiring $R_{i,j} > 0$ for each selected successor.

Iteratively extract feasible solutions. We apply the above walkthrough repeatedly to extract feasible permutation matrices with positive weights from S. Initialize the residual $R^{(0)} = S$. Every walkthrough uses the same fixed priority $r(j)$. At inference, this priority follows the lexicographical order of node indices; during training, it is $r(j) = \mathrm{pos}_{\pi^{\mathrm{ref}}}(j)$, given by the feasible reference label π^{ref}. At iteration k, start from source 0 and conduct a walkthrough in M, restricting each selected successor to $d_U(j) = 0$ and $R^{(k-1)}_{i,j} > 0$.

Write $Q_k = Q_{\pi_k}$ for the matrix extracted in iteration k. During each extraction iteration, we compute its largest admissible coefficient and update the residual, as formulated in Eq. (5).

$$\lambda_k = \min_{(i,j):(Q_k)_{i,j}=1} R^{(k-1)}_{i,j}, \qquad R^{(k)} = R^{(k-1)} - \lambda_k Q_k. \tag{5}$$

Each successful subtraction removes at least one positive entry, so subsequent walkthroughs operate on a shrinking support. Repeat until the residual is zero or the current priority-based walkthrough cannot complete a chain. After m successful rounds, the output obeys $\mathcal{D}_C(S) = \big(\{(\lambda_k, Q_k)\}_{k=1}^m, R^{(m)}\big)$, and $S = \sum_{k=1}^m \lambda_k Q_k + R^{(m)}$. For an exactly doubly stochastic input, $R^{(m)} = 0$ yields a complete feasible decomposition with $\sum_k \lambda_k = 1$. If the walkthrough stops earlier, the extracted matrices still define a feasible part of S. Appendix B.4 provides the complete extraction algorithm, and a quadratic-time analysis for each iteration.

3.3.2 Feasible Decoding at Inference

At inference time, each trajectory starts from $z_{T-1} \sim \mathcal{N}(0, I)$ and follows a short decreasing denoise path $[T-1, t_m, t_{m-1}, \ldots, t_0, 0]$. After the final transition, finite-iteration Sinkhorn normalization produces the approximately doubly stochastic output S_θ. We then apply the BvN-based feasible solution extraction algorithm in Section 3.3.1, without the training-time feasible component mapping, obtaining a decomposition $\mathcal{D}_C(S_\theta) = \big(\{(\lambda_k, Q_k)\}_{k=1}^m, R\big)$. At inference, the supervision target Q^{ref} is unavailable, so we use lexicographical order as the priority when generating feasible solutions. For a strictly positive DSM in exact arithmetic, the extraction algorithm can obtain at least one feasible solution because every edge of a topological ordering is supported. The default decoder selects the extracted permutation matrix with the largest coefficient, as formulated in Eq. (6).

$$k^\star = \arg\max_{1 \le k \le m} \lambda_k, \qquad \widehat{Q} = Q_{k^\star}, \qquad \widehat{\pi} = \pi_{k^\star} \in \Pi_C. \tag{6}$$

Every solution returned in this way is feasible because feasibility is checked during extraction. Thus, the decoder guarantees the feasibility of returned solutions, and this procedure does not require a post-processing repair. We also use top-K inference to select the lowest-cost feasible solution from multiple sampled trajectories.

3.4 Learning Guided by Feasible Components

The decoder produces feasible solutions, but doubly stochasticity alone does not ensure that the entire DSM lies in the feasible polytope, as analyzed in Section 3.1. We therefore train the diffusion solver under guidance from feasible components, in addition to the supervision target Q^{ref}. The supervised loss encourages learning the reference solution, whereas the feasible-component loss supplies additional guidance toward partial-order feasibility. Feasible component mapping constructs the training target for this guidance from the output of the extraction algorithm in Section 3.3.1.

Feasible component mapping and feasible-component guidance. After the iterative extraction, we obtain the total weight $\widehat{w}_C(S) = \sum_{k=1}^m \lambda_k$ and the summation $A_C(S) = \sum_{k=1}^m \lambda_k Q_k$ of the extracted matrices. To construct a feasible training target, the mapping forms the normalized feasible component in Eq. (7). This DSM is a convex combination of precedence-feasible permutation matrices. We therefore call $S_C(S)$ a feasible component of S. This mapping supplies the target for feasibility guidance during training; it is not needed to obtain discrete solutions at inference. More details about how this feasible component is constructed are in Fig. 4, Appendix B.6. We treat the feasible component $S_{\theta,C}$ as a detached training target and minimize the element-wise mean squared error (MSE) in Eq. (8). Since $S_{\theta,C} \in \mathcal{P}_C^{\mathrm{succ}}$, this objective encourages agreement with a feasible convex combination.

$$S_C(S) = \frac{A_C(S)}{\widehat{w}_C(S)} \in \mathcal{P}_C^{\mathrm{succ}}, \qquad \widehat{w}_C(S) > 0. \tag{7}$$

$$\mathcal{L}_{\mathrm{feas}}(S_\theta; C) = \frac{1}{(n-1)^2} \left\| S_\theta - S_{\theta,C} \right\|_F^2, \qquad \widehat{w}_C(S_\theta) > 0, \tag{8}$$

Learning from the supervision target. Similar to the optimization consistency of Fast-T2T (Li et al., 2024), we develop a supervised loss target in the continuous DSM space. We independently sample two diffusion levels s and t and measure the distance between the few-step diffusion output and the reference solution at both levels. Considering $S_\theta^{(u)}$ as the DSM output obtained at level $u \in \{s, t\}$, we formalize the distance function in the continuous DSM space as the element-wise MSE in Eq. (9). The supervised loss is the sum of the two sampled-level distances in Eq. (10).

$$d_{\mathrm{DSM}}\Big(S_\theta^{(u)}, Q^{\mathrm{ref}}\Big) = \frac{1}{(n-1)^2} \left\| S_\theta^{(u)} - Q^{\mathrm{ref}} \right\|_F^2. \tag{9}$$

$$\mathcal{L}_{\mathrm{sup}}(S_\theta, Q^{\mathrm{ref}}, s, t) = d_{\mathrm{DSM}}\Big(S_\theta^{(t)}, Q^{\mathrm{ref}}\Big) + d_{\mathrm{DSM}}\Big(S_\theta^{(s)}, Q^{\mathrm{ref}}\Big). \tag{10}$$

The joint training objective. Finally, we optimize the expected joint training objective in Eq. (11). Empirically, we apply $\lambda_{\text{feas}} = 0.1$ as the coefficient of the feasible-component guidance term $\mathcal{L}_{\text{feas}}$.

$$\mathcal{L}_{\text{core}}(\theta) = \mathbb{E}_{\substack{(G_C, Q^{\text{ref}}) \sim \mathcal{D},\ s,t \sim p \\ \epsilon_s, \epsilon_t \sim \mathcal{N}(0,I)}} \Big[\mathcal{L}_{\text{sup}}(S_\theta, Q^{\text{ref}}, s, t) + \lambda_{\text{feas}} \big(\mathcal{L}_{\text{feas}}(S_\theta^{(t)}; C) + \mathcal{L}_{\text{feas}}(S_\theta^{(s)}; C) \big) \Big]. \tag{11}$$

4 Evaluation

4.1 Reference Solutions and Supervision

In this section, we evaluate solution quality and partial-order feasibility on DAG scheduling problems. We first describe the evaluation datasets and the compared methods, then define the main evaluation metrics. We conduct comparison experiments with baselines and ablation studies on the designs in our proposed framework.

4.2 Evaluation datasets

Sequential ordering problem (SOP). SOP is a variant of the asymmetric traveling salesman problem (ATSP). In addition to finding a minimum-length Hamiltonian path, SOP requires taking given partial-order constraints into account. Each partial-order constraint requires a node i to be visited before another node j. Such ordering constraints can be represented in the form of a directed acyclic graph (DAG). For evaluation, we use the benchmark instances provided by **TSPLIB-SOP**[1].

Workflow scheduling problem. The workflow scheduling problem is a more general form of DAG scheduling, corresponding to real-world task scheduling on multicore computing systems. It involves generating a scheduling solution, represented by an execution ordering, for a set of tasks with partial-order constraints. These partial-order constraints define the workflow structure. We use two workflow scheduling datasets, **TPC-H**[2] and **Pegasus** (Deelman et al., 2015)[3]. In TPC-H, each scheduling instance consists of multiple small, parallelizable workflows, with an average of approximately 9 tasks per workflow, so the partial constraints are relatively sparse. In our experiments, we consider six scales of the number of workflows in TPC-H: 10, 20, 30, 50, 100, and 150. In Pegasus, each scheduling instance corresponds to the workflow of an entire scientific computing application, resulting in relatively dense precedence constraints. We consider six scales of the number of tasks in Pegasus.

Detailed descriptions of these datasets and the procedures for obtaining the reference solutions are provided in Appendix C.

4.3 Compared Methods

Our approach is compared against the following baselines: (1) PPO-BiHyb (Wang et al., 2021), a reinforcement-learning-based autoregressive framework for graph-structured optimization problems, including DAG scheduling; (2) Qi et al. (2025), a non-autoregressive framework for DAG scheduling based on reinforcement learning and listwise ordering model, which generates item priorities using a GNN and then performs a feasibility repair based on a priority topological sorting on these priorities, while it can get consistent probability estimation with the solutions obtained after this repair; (3) *Positional Heatmap Diffusion* and *Successor Heatmap Diffusion*, two implementations for DAG scheduling based on the discrete diffusion backbone of Fast-T2T (Li et al., 2024), in which the solutions are denoised into heatmaps of binary variables instead of continuous DSMs, following the conventional practice of diffusion-based COP solvers; (4) Ebrahimi et al. (2026), a diffusion-based learning-to-rank model with a pairwise ordering representation; Details about some of these baselines are in Appendix D.

[1] https://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/index.html
[2] https://www.tpc.org/tpch/
[3] https://pegasus.isi.edu/workflow_gallery/

4.4 Evaluation Metrics

Our main metrics are *mean cost* and the *relative gap* between the mean cost and the mean reference cost. For both metrics, the lower the better. Some of the compared methods cannot achieve full feasibility for partial-order constraints and require post-processing repair, so we evaluate their *feasibility rate* before post-processing repair. Over the N evaluated instances, we report the mean cost and its relative gap to the mean reference cost in Eq. 12, and the feasibility rate in Eq. (13). Here, $\widehat{\pi}_\ell$ is the candidate solution represented by the ordering decoded from the diffusion output. For methods requiring repair, $\widehat{\pi}_\ell$ is the raw ordering before repair, whereas $\widehat{\pi}_\ell^{\mathrm{eval}}$ is the repaired solution used for cost evaluation.

$$\begin{aligned} \mathrm{MeanCost} &= \frac{1}{N}\sum_{\ell=1}^{N} g_\ell\big(\widehat{\pi}_\ell^{\mathrm{eval}}\big), \quad \mathrm{MeanRefCost} \;= \frac{1}{N}\sum_{\ell=1}^{N} g_\ell\big(\pi_\ell^{\mathrm{ref}}\big), \\ \mathrm{MeanGap} &= \frac{\mathrm{MeanCost} - \mathrm{MeanRefCost}}{\mathrm{MeanRefCost}}. \end{aligned} \tag{12}$$

$$\mathrm{Feasibility} = \frac{1}{N}\sum_{\ell=1}^{N} \mathbb{I}[\widehat{\pi}_\ell \in \mathrm{LE}(C_\ell)]\,. \tag{13}$$

4.5 Comparison results

Table 1, 2 and 3 show the comparison results. We observe that our method achieves particularly better performance in large-scale instances (especially TPC-H 150 in Table 1), compared with methods that rely on post-processing repair. The methods relying on post-processing repair have substantial difficulty generating feasible solutions for large-scale scheduling instances, leaving the feasibility of many of their outputs to the subsequent repair procedure. We hypothesize such models tend to converge to infeasible solutions that can be repaired into good solutions during training, which may lead to generalization issues as the problem scale increases. In contrast, our method uses feasibility guidance to enable the diffusion model to learn towards the feasible region in the DSM space, and applies feasible solution decoding to obtained solutions obeying the learned feasible solution set. This provides a more direct mechanism for generating feasible solutions and consequently leads to better performance, particularly on large-scale instances.

Table 1: Comparison and ablation results on the TPC-H dataset on 6 different scheduling problem scales of workflows. $\bar{n}$ the corresponding average number of items (task nodes). Cost is the mean cost, Gap is the relative gap between the mean cost and mean reference cost (%), and Feas the feasibility rate before repair (%). N/A terms in the feasibility rate column indicate that the method does not require a post-processing repair.

Method	TPC-H 10 $\bar{n}$=97.1			TPC-H 20 $\bar{n}$=190.2			TPC-H 30 $\bar{n}$=294.3			TPC-H 50 $\bar{n}$=500			TPC-H 100 $\bar{n}$=912			TPC-H 150 $\bar{n}$=1385		
	Cost	*Gap(%)*	*Feas*	*Cost*	*Gap(%)*	*Feas*	*Cost*	*Gap(%)*	*Feas*	*Cost*	*Gap(%)*	*Feas*	*Cost*	*Gap(%)*	*Feas*	*Cost*	*Gap(%)*	*Feas*
Reference cost	3.13	N/A	N/A	6.65	N/A	N/A	9.84	N/A	N/A	15.98	N/A	N/A	32.45	N/A	N/A	47.29	N/A	N/A
PPO-BiHyb	3.30	5.43	N/A	8.48	27.52	N/A	12.18	23.78	N/A	19.25	20.46	N/A	37.78	16.43	N/A	54.47	15.18	N/A
Qi et al. (2025)	3.30	5.43	95.3	**6.98**	**4.96**	98.2	11.26	14.43	85.6	17.49	9.45	81.2	37.22	14.70	58.2	73.47	55.36	51.4
Positional Heatmap	6.10	94.89	12.4	14.95	124.81	6.5	13.18	33.91	0.0	24.75	54.88	0.0	52.47	61.69	0.0	88.32	86.76	0.0
Successor Heatmap	4.36	39.30	N/A	9.30	39.80	N/A	10.65	8.23	N/A	20.55	28.60	N/A	38.36	18.21	N/A	62.54	32.25	N/A
Ebrahimi et al. (2026)	3.53	12.68	96.4	7.32	10.10	84.1	11.93	21.21	89.7	17.69	10.69	64.5	37.94	16.92	60.8	54.64	15.54	0.0
Ours	**3.29**	**5.11**	N/A	7.02	5.56	N/A	**10.29**	**4.57**	N/A	**17.44**	**9.14**	N/A	**36.25**	**11.71**	N/A	**52.01**	**9.98**	N/A
Ours w/o FCG & FSD	3.30	5.43	96.6	7.79	17.11	61.0	12.86	30.69	74.4	22.84	42.93	56.1	44.86	38.24	50.5	79.07	67.20	36.8
Ours w/o FCG	3.59	14.70	N/A	7.82	17.59	N/A	12.22	24.19	N/A	18.22	14.02	N/A	47.15	45.30	N/A	62.37	31.89	N/A
Ours w/o FSD	3.30	5.43	96.6	7.92	19.10	65.5	12.78	29.88	78.6	22.40	40.18	60.0	46.87	44.44	44.0	74.59	57.73	20.0

4.6 Ablation Study

In this subsection, we examine feasible decoding at inference, and feasible component guidance during training. The ablations maintain the framework of continuous diffusion in the DSM space.

No feasible decoding at inference. For inference, we compare our feasible decoder, with an alternative topological decoder applied to the same DSM output S_θ. The alternative decoder directly samples the highest-weighted successor permutation matrix through the Hungarian algorithm (Wright,

Table 2: Comparison and ablation results on the Pegasus dataset on 6 different scheduling problem scales of task nodes. Cost is the mean cost, Gap is the relative gap between the mean cost and mean reference cost (%), and Feas the feasibility rate before repair (%). N/A terms in the feasibility rate column indicate that the method does not require a post-processing repair.

Method	Pegasus 100			Pegasus 200			Pegasus 300			Pegasus 400			Pegasus 500			Pegasus 1000		
	Cost	*Gap(%)*	*Feas*	*Cost*	*Gap(%)*	*Feas*	*Cost*	*Gap(%)*	*Feas*	*Cost*	*Gap(%)*	*Feas*	*Cost*	*Gap(%)*	*Feas*	*Cost*	*Gap(%)*	*Feas*
Reference cost	9468.3	N/A	N/A	17184.5	N/A	N/A	24257.9	N/A	N/A	28439.5	N/A	N/A	46329.3	N/A	N/A	88101.6	N/A	N/A
PPO-BiHyb	10068.0	6.33	N/A	17938.0	4.38	N/A	25033.0	3.20	N/A	29288.0	2.98	N/A	47559.0	2.65	N/A	90244.0	2.43	N/A
Qi et al. (2025)	10005.0	5.67	87.3	**17457.0**	**1.59**	87.5	25704.0	5.96	85.6	29807.0	4.81	82.0	47213.0	1.91	65.2	90997.0	3.29	48.1
Positional Heatmap	11018.0	16.37	0.0	19604.5	14.08	0.0	27128.0	11.83	0.0	31483.0	10.70	0.0	50771.0	9.59	0.0	91541.0	3.90	0.0
Successor Heatmap	10901.0	15.13	N/A	19549.0	13.76	N/A	26942.0	11.06	N/A	31283.0	10.00	N/A	50998.0	10.08	N/A	93099.0	5.67	N/A
Ebrahimi et al. (2026)	10133.0	7.02	96.3	18553.0	7.96	83.1	25916.0	6.84	64.3	29899.0	5.13	36.4	48577.0	4.85	0.0	91239.0	3.56	0.0
Ours	**9844.8**	**3.98**	N/A	18024.0	4.89	N/A	**25030.0**	**3.18**	N/A	**28948.0**	**1.79**	N/A	**47136.0**	**1.74**	N/A	**89477.0**	**1.56**	N/A
Ours w/o FCG & FSD	10412.7	9.97	100.0	18987.0	10.49	100.0	26922.8	10.99	87.0	32225.3	13.31	21.0	50613.3	9.25	14.0	94553.2	7.32	0.0
Ours w/o FCG	10408.0	9.92	N/A	18678.0	8.69	N/A	25893.0	6.74	N/A	30254.0	6.38	N/A	48744.0	5.21	N/A	91401.0	3.74	N/A
Ours w/o FSD	10287.0	8.65	100.0	18776.0	9.26	100.0	26083.0	7.52	96.0	30693.0	7.92	93.0	49551.0	6.95	82.0	91640.0	4.02	68.0

Table 3: Results on TSPLIB-SOP. Cost is the mean cost, Gap is the relative gap between the mean cost and mean reference cost (%), and Feas is the feasibility rate before repair (%). N/A entries in the feasibility rate column indicate that the method does not require post-processing repair.

Method	Mean Cost	Mean Gap (%)	Feasibility (%)
Reference cost	17583.1	N/A	N/A
PPO-BiHyb	20758.1	18.06	N/A
Qi et al. (2025)	22053.5	25.42	86.7
Positional Heatmap	38213.9	117.33	8.3
Successor Heatmap	24782.9	40.95	N/A
Ebrahimi et al. (2026)	26632.9	51.47	45.1
Ours	**20589.0**	**17.10**	N/A
Ours w/o FCG & FSD	31065.0	76.68	53.4
Ours w/o FCG	30743.0	74.84	N/A
Ours w/o FSD	20827.0	18.45	97.9

1990) and converts the permutation matrix to a raw ordering. This candidate solution is not guaranteed to be feasible, so a post-processing repair based on prioritized topological sorting is required, which repeatedly selects the item with the earliest position in the raw ordering. Details are described in Appendix D.

No feasible component guidance. We compare training with and without $\mathcal{L}_{\text{feas}}$, i.e., setting $\lambda_{\text{feas}} = 0$. The successor interpretation and feasible decoder are held fixed in this configuration.

The ablation results are presented in the rows near the bottom in the Tables 1, 2 and 3, where FSD represents the feasible solution decoding, and FCG represents the feasible component guidance. Our complete framework with feasibility guidance learning and feasible solution decoding achieves substantial improvements across different DAG scheduling tasks. In contrast,Either component alone is generally insufficient to match the complete framework. These results suggest that the effectiveness of our framework relies on the joint interaction between feasibility guidance and feasible solution decoding. Specifically, feasibility guidance first enables the diffusion model to learn toward the feasible region in the DSM space, while feasible solution decoding subsequently generates feasible solutions in this region.

5 Conclusion

This paper presents a DAG scheduling framework based on a diffusion model. The DSM representation and diffusion model support non-autoregressive generation without estimating the joint probability of a discrete solution. The proposed feasible solution decoding procedure enables producing feasible schedules without an additional post-processing repair. We construct feasible components that guide learning alongside supervision from reference solutions. For future work, we plan to investigate how other diffusion-based learning-to-rank methods can generate solutions satisfying partial-order constraints. We also plan to extend the framework to optimization and scheduling problems subject to other combinatorial constraints.

6 LLM USAGE OF THIS PAPER

In the paper writing, generative AI tools were used to draft the sections, polish the writing (including the mathematical formulations and proofs), assist with translation, and check typos. We also used LLM-assisted coding to implement the method and to support the data analysis scripts and qualitative analysis. We did not use generative AI tools to help develop theoretical models or conceptual frameworks, provide critical ingredients for proving mathematical claims, propose or refine hypotheses, design or provide feedback on research methodology or experiments, or clean and reformat datasets. Generating synthetic datasets is not applicable to this work. Additionally, we used generative AI tools for editing to improve readability. We have reviewed all AI-assisted results of writing and coding. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Riyaz Ahmad Bhat and Jaydeep Sen. Logit inflation in listmle: Theoretical analysis and mitigation strategies. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '26, pp. 76–87, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400725999. doi: 10.1145/3805712.3809676. URL https://doi.org/10.1145/3805712.3809676.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

Suhong Chen and Xudong Wang. Ptrtasking: Pointer network based task scheduling for multi-connectivity enabled mec services. *IEEE Transactions on Mobile Computing*, 23(10):9398–9409, 2024. doi: 10.1109/TMC.2024.3363662.

Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper_files/paper/2013/file/af21d0c97db2e27e13572cbf59eb343d-Paper.pdf.

Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J. Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, and Kent Wenger. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 2015. ISSN 0167-739X. doi: 10.1016/j.future.2014.10.008. URL https://www.sciencedirect.com/science/article/pii/S0167739X14002015.

Fanny Dufossé and Bora Uçar. Notes on birkhoff-von neumann decomposition of doubly stochastic matrices. *Linear Algebra and its Applications*, 497:108–115, 2016. ISSN 0024-3795. doi: 10.1016/j.laa.2016.02.023. URL https://www.sciencedirect.com/science/article/pii/S0024379516001257.

Sajad Ebrahimi, Bhaskar Mitra, Negar Arabzadeh, Ye Yuan, Haolun Wu, Fattane Zarrinkalam, and Ebrahim Bagheri. From noise to order: Learning to rank via denoising diffusion. In *Proceedings of the 2026 International ACM SIGIR Conference on Innovative Concepts and Theories in Information Retrieval (ICTIR)*, ICTIR '26, pp. 62–72, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400726002. doi: 10.1145/3805713.3820437. URL https://doi.org/10.1145/3805713.3820437.

Artyom Gadetsky, Kirill Struminsky, Christopher Robinson, Novi Quadrianto, and Dmitry Vetrov. Low-variance black-box gradient estimates for the plackett-luce distribution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 10126–10135, 2020.

Mukul Gagrani, Corrado Rainone, Yang Yang, Harris Teague, Wonseok Jeon, Roberto Bondesan, Herke van Hoof, Christopher Lott, Weiliang Zeng, and Piero Zappi. Neural topological ordering for computation graphs. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 17327–17339. Curran Associates, Inc., 2022. doi: 10.52202/

068431-1260. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/6ef586bdf0af0b609b1d0386a3ce0e4b-Paper-Conference.pdf.

Yan Gu, Zhaoze Liu, Shuhong Dai, Cong Liu, Ying Wang, Shen Wang, Georgios Theodoropoulos, and Long Cheng. Deep reinforcement learning for job scheduling and resource management in cloud computing: An algorithm-level review. *arXiv preprint arXiv:2501.01007*, 2025.

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.

Yang Li, Jinpei Guo, Runzhong Wang, Hongyuan Zha, and Junchi Yan. Fast t2t: Optimization consistency speeds up diffusion-based training-to-testing solving for combinatorial optimization. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 30179–30206. Curran Associates, Inc., 2024. doi: 10.52202/079017-0950. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/352b13f01566ae34affacc60e98c16af-Paper-Conference.pdf.

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 of *AAAI'26/IAAI'26/EAAI'26*, pp. 37063–37071. AAAI Press, 2026. ISBN 978-1-57735-906-7. doi: 10.1609/aaai.v40i43.41035. URL https://doi.org/10.1609/aaai.v40i43.41035.

Christian Löwens, Inaam Ashraf, Alexander Gembus, Genesis Cuizon, Jonas K Falkner, and Lars Schmidt-Thieme. Solving the traveling salesperson problem with precedence constraints by deep reinforcement learning. In *German Conference on Artificial Intelligence (Künstliche Intelligenz)*, pp. 160–172. Springer, 2022.

Hengyuan Ma, Wenlian Lu, and Jianfeng Feng. Efficient combinatorial optimization via heat diffusion. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 112104–112133. Curran Associates, Inc., 2024. doi: 10.52202/079017-3559. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/cb2b9198daacf5f4343721226714 50ef-Paper-Conference.pdf.

Gonzalo E. Mena, David Belanger, Scott W. Linderman, and Jasper Snoek. Learning latent permutations with gumbel-sinkhorn networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=Byt3oJ-0W.

Abinay Reddy Naini, Fernando Diaz, and Carlos Busso. Ranklist–a listwise preference learning framework for predicting subjective preferences. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 24486–24494, 2026.

Xumai Qi, Dongdong Zhang, Taotao Liu, and Hongcheng Wang. Reinforcement learning for one-shot dag scheduling with comparability identification and dense reward. 38:149539–149563, 2025. doi: 10.52202/085713-5002. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/dc16950e510bcd573783f5c2d7d8cc31-Paper-Conference.pdf.

Yutao Song, Chen Li, Lihua Tian, and Hui Song. A reinforcement learning based job scheduling algorithm for heterogeneous computing environment. *Computers and Electrical Engineering*, 107:108653, 2023. ISSN 0045-7906. doi: 10.1016/j.compeleceng.2023.108653. URL https://www.sciencedirect.com/science/article/pii/S0045790623000782.

Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 3706–3731. Curran Associates, Inc., 2023. doi: 10.52202/075280-0164. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/0ba520d93c3df592c83a611961314c98-Paper-Conference.pdf.

Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, 2002. doi: 10.1109/71.993206.

Anke Van Zuylen and David P Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. *Mathematics of Operations Research*, 34(3):594–620, 2009.

Mingzhao Wang, You Zhou, Zhiguang Cao, Yubin Xiao, Xuan Wu, Wei Pang, Yuan Jiang, Hui Yang, Peng Zhao, and Yuanshu Li. An efficient diffusion-based non-autoregressive solver for traveling salesman problem. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1*, KDD '25, pp. 1469–1480, New York, NY, USA, 2025a. Association for Computing Machinery. ISBN 9798400712456. doi: 10.1145/3690624.3709343. URL https://doi.org/10.1145/3690624.3709343.

Runzhong Wang, Zhigang Hua, Gan Liu, Jiayi Zhang, Junchi Yan, Feng Qi, Shuang Yang, Jun Zhou, and Xiaokang Yang. A bi-level framework for learning to solve combinatorial optimization on graphs. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 21453–21466. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/b2f627fff19fda463cb386442eac2b3d-Paper.pdf.

Ying Wang, Preslav Nakov, and Shangsong Liang. Denoiserank: Learning to rank by diffusion models. *arXiv preprint arXiv:2604.20852*, 2026.

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 D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 71201–71228. Curran Associates, Inc., 2025b. doi: 10.52202/085713-2394. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/6728fcf94660c59c938319a6833a6073-Paper-Conference.pdf.

Mike B Wright. Speeding up the hungarian algorithm. *Computers & Operations Research*, 17(1): 95–96, 1990.

Hai-Tao Yu, Degen Huang, Fuji Ren, and Lishuang Li. Diagnostic evaluation of policy-gradient-based ranking. *Electronics*, 11(1), 2022. ISSN 2079-9292. doi: 10.3390/electronics11010037.

Hang Zhao, Kexiong Yu, Yuhang Huang, Renjiao Yi, Chenyang Zhu, and Kai Xu. Disco: Efficient diffusion solver for large-scale combinatorial optimization problems. *Graphical Models*, 141: 101284, 2025. ISSN 1524-0703. doi: 10.1016/j.gmod.2025.101284. URL https://www.sciencedirect.com/science/article/pii/S1524070325000311.

Zhaomeng Zhu, Gongxuan Zhang, Miqing Li, and Xiaohui Liu. Evolutionary multi-objective workflow scheduling in cloud. *IEEE Transactions on Parallel and Distributed Systems*, 27(5):1344–1357, 2016. doi: 10.1109/TPDS.2015.2446459.

A Related Works

A.1 Learning-Based Methods for DAG Scheduling

Existing autoregressive methods, including reinforcement learning (RL)-based sequential methods, construct a solution for DAG scheduling by selecting one item at a time (Song et al., 2023; Gu et al., 2025; Wang et al., 2021; Chen & Wang, 2024). This procedure requires repeated model evaluations and has high computational complexity. More recently, non-autoregressive methods for DAG scheduling directly predict the probability distribution of itemwise priorities of the solution in one forward propagation of the neural network (Gagrani et al., 2022; Jeon et al., 2023). However, the generated priority orderings represent candidate solutions that do not naturally satisfy the DAG's partial-order constraints and may require an additional feasibility repair. The repaired solution after post-processing is inconsistent with the estimation of the joint probability of the raw ordering. The model is likely to confidently converge to an infeasible candidate solution that can be repaired to a good solution after post-processing. This affects the generalization performance of the model. In this situation, the model does not truly learn the distribution of good solutions. Qi et al. (2025) partially resolves this issue of inconsistent probability estimation. However, in their pairwise ranking representations, the sampled solutions can still be numerically sensitive to logits (Naini et al., 2026; Bhat & Sen, 2026; Yu et al., 2022). Estimating the joint probability of discrete solutions can also introduce high variance and affect training (Gadetsky et al., 2020). These issues lead to fluctuations in training and solution sampling.

A.2 Diffusion-based Methods for Combinatorial Optimization

Diffusion-based methods provide a non-autoregressive and generative route for solving COPs (Sun & Yang, 2023; Ma et al., 2024; Li et al., 2024; Wang et al., 2025a; Li et al., 2026). Among them, a common practice is to represent a solution by binary decision variables, denoise them into a heatmap, and then sample a discrete solution. However, the heatmap space does not naturally model the dependencies between items in a permutation, and a separate decoder is required to restore permutation structure. Partial-order constraints bring a further difficulty. A decoder that restores a permutation does not ensure feasibility under the partial-order constraints. Some diffusion solvers handle simple constraints by defining problem-specific forward transition kernels (Zhao et al., 2025; Wang et al., 2025b), but such mechanisms are tied to particular problem structures and do not directly provide a general representation of partial-order relations over permutations. Several studies have explored the use of diffusion models for generating orderings or rankings (Ebrahimi et al., 2026; Wang et al., 2026), but these studies are not specifically designed for partial-order constraints or other constraints in the generated orderings.

A.3 DSM for ordering and permutation

The output of a neural network can be differentially projected to a DSM through Sinkhorn normalization (Cuturi, 2013; Mena et al., 2018), and then permutation matrices can be extracted from the DSM through BvN decomposition (Wright, 1990). However, the DSM alone does not address the partial-order constraint issue. Under the conventional positional interpretation of permutation matrices, the BvN decomposition is not guaranteed to effectively find a feasible permutation matrix.

B Implementation Details

This appendix describes the default implementation of the framework in Section 3, following the order of the main text: the diffusion backbone, feasible decoding at inference, and learning guided by feasible components. The extraction algorithm is shared by decoding and training; feasible component mapping is used only to construct the training target for feasible component guidance.

B.1 Input Representation and Constraint Preprocessing

Each instance contains a DAG G_C, its node attributes (including its problem-specific attributes, in-degrees, out-degrees, the distance to the sink and source nodes), a reference permutation matrix Q^{ref},

and the relation matrix M for feasible solution extraction. Following the preprocessing convention in Section 2, every input DAG has a unique source 0 and a unique sink $n-1$.

The Sinkhorn operator alternates row and column normalization. For logits Z, one iteration can be written schematically as:

$$Y_{i,j} \leftarrow \exp(Z_{i,j}/\tau), \qquad Y \leftarrow \mathrm{RowNorm}(Y), \qquad Y \leftarrow \mathrm{ColNorm}(Y). \tag{14}$$

The normalizations are repeated 20 times. The default temperature is $\tau = 0.2$. A small constant of 10^{-5} is added to denominators and logarithms in the implementation to avoid numerical overflow or undefined values.

B.2 Conditional Denoising Network

The framework consists of a feature encoder and a matrix denoiser. The feature network is the graph encoder that produces the conditional features F in Section 3.2. It applies two bidirectional graph-attention layers with hidden state dimension of 32. One branch propagates messages along the original precedence edges and the other along the reversed edges. Their outputs are concatenated and passed through a linear map, residual connection, and layer normalization, producing $F \in \mathbb{R}^{n\times d}$. This allows each item representation to summarize both predecessor and successor information.

The noise network is the conditional denoiser $f_\theta(x_t, t, F, G_C)$ in Section 3.2. It is based on a graph attention network (Brody et al., 2021) with hidden state dimension of 32. It receives the current Sinkhorn state, the node features, a sinusoidal time embedding, positional encodings, and the original graph structure. For each item, the node, time, and position features are concatenated and processed by bidirectional graph attention. A multi-head attention module then constructs the score matrix while also incorporating the corresponding entries of the current matrix. The output on the active successor block is an $(n-1)\times(n-1)$ real matrix interpreted as the direct prediction $\widehat{z}_0$, rather than an estimate of ϵ.

B.3 Forward Process

Following the common practice of Gaussian diffusion, we use a cosine schedule with the zero-based time indices of Eq. (3). The upper bound on β_t avoids a zero signal scale at the final time level. We set $\alpha_t = 1-\beta_t$ and $\bar{\alpha}_t = \prod_{r=0}^{t}\alpha_r$. For every selected time level, Gaussian noise is added according to Eq. (3); the noisy matrix is passed through Sinkhorn before entering the noise network. The prediction is again normalized to obtain S_θ.

$$\begin{aligned} a_{\cos}(u) &= \cos^2\!\left(\frac{u+\delta}{1+\delta}\frac{\pi}{2}\right), \qquad \delta = 0.008, \\ \beta_t &= \min\!\left(1 - \frac{a_{\cos}((t+1)/T)}{a_{\cos}(t/T)},\, 0.999\right), \qquad t \in \{0,\dots,T-1\}. \end{aligned} \tag{15}$$

B.4 Successor Walkthrough for Feasible Solution Extraction and Complexity

The preliminary walkthrough in Section 3.3.1 and the residual extraction below use the same successor construction. The former can be viewed as a special case in which the residual has no zero entries, and the residual support does not restrict the choice of successors. We first describe this case, then the general residual extraction procedure.

Preliminary walkthrough with no zero residual entries. We construct the relation matrix

$$M_{i,j} = \mathbb{I}\big[j \in \mathcal{I}_C(i) \text{ or } (i,j) \in E_C\big], \qquad i,j \in \{0,\dots,n-1\}. \tag{16}$$

, where $\mathcal{I}_C(i) = \{j \neq i : i \not\prec_C j,\ j \not\prec_C i\}$ is the incomparability set. In M, incomparable nodes and direct successors in G_C are 1, and all other entries are 0. To construct one valid topological ordering, we maintain a visited set U and, for each unvisited node j, a counter $d_U(j)$ of its unvisited immediate predecessors.

1. Initialize the list to (0), set $U = \{0\}$ and the current node to $i = 0$, and initialize an all-zero $n \times n$ matrix A. Initialize the counters from the DAG in-degrees, then decrement the counter of each outgoing neighbor of 0.
2. Scan row i and choose any node j satisfying $M_{i,j} = 1$, $j \notin U$, and $d_U(j) = 0$. Set $A_{i,j} = 1$, append j, add it to U, and decrement the counters of its outgoing neighbors. Set $i = j$ and continue from row j.
3. Repeat until all n nodes have been visited. The last node is $n - 1$; return the list and the upper-right block of A as Q_π.

Walkthrough in a residual with zero entries. The procedure for general cases, where the residual has 0 entries, applies the same successor walkthrough to the shrinking residual support, additionally checking the residual value of each selected edge. It reuses the source/sink convention and reduced adjacency representation in Section 3.3.1. Array indices below use full node labels: an active-matrix column with zero-based index b represents node $b + 1$.

Successor interpretation and preprocessing procedure. For each graph, we build the full $n \times n$ representation with the row and column labels of A_π, then its active $(n-1) \times (n-1)$ block. Fig. 2 illustrates a DAG instance with $n = 7$ and an ordering π, showing how its successor permutation matrix Q_π is constructed, and its distinction from the positional permutation P_π. By the preprocessing convention in Section 2, 0 and $n - 1$ are the unique source and sink. From the directed paths in G_C, cache lists of all predecessors and successors of each node, excluding the node itself, and counts $d_0(j)$ of all its predecessors. In the residual algorithm below, $d_U(j)$ counts all predecessors of j that have not been visited, rather than only its immediate predecessors as in the preliminary walkthrough. The two counts give the same availability test for orderings constructed by these walkthroughs when updated consistently. Cache the fixed priority table $r(j)$ as well: use lexicographical node order at inference and $r(j) = \mathrm{pos}_{\pi^{\mathrm{ref}}}(j)$ during training. Reuse these data across extraction rounds. Mutable counters and visited flags are reset for each round.

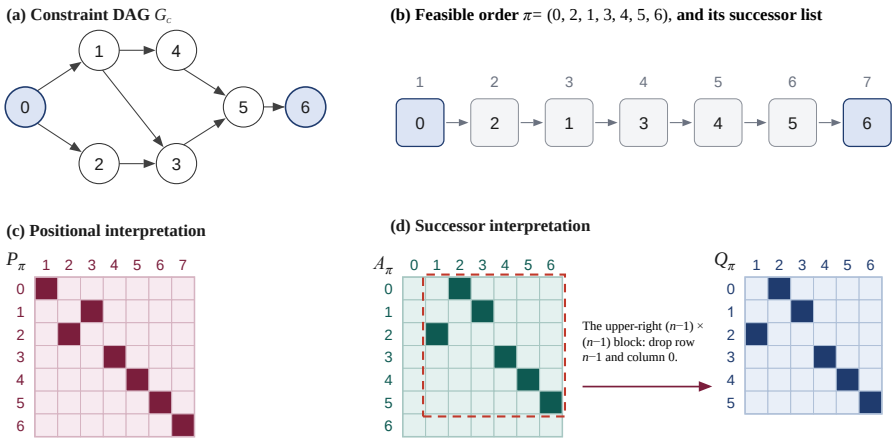


Figure 2: Positional vs. successor interpretation of a permutation matrix for one instance with $n = 7$. (a) The constraint DAG instance has a unique source 0 and a unique sink $n - 1 = 6$, so every feasible solution is represented by an ordering that starts at 0 and ends at 6. (b) A feasible solution represented by the topological ordering π and the corresponding successor list. (c) Under the positional interpretation, P_π marks the position of each item, so precedence feasibility couples the two index sets jointly. (d) The successor list's adjacency matrix A_π has a zero first column and a zero last row; its upper-right $(n-1) \times (n-1)$ block is a permutation matrix Q_π, whose entry (i, j) marks the successor j of node i in π.

Algorithm: feasible solution extraction. Inputs are a generic active matrix S (set to S_θ in the framework), the relation matrix M, cached predecessor/successor lists and counts, a fixed priority table $r(j)$, and the feasible reference label π^{ref} only during training. Outputs are the extracted feasible permutation matrices with their coefficients, their weighted sum $A_C(S)$ as in Section 3.4,

the residual R, and the matrix $Q_{\max}$ with the largest coefficient. For a nonnegative network output, the initialization below equals $R^{(0)} = S$. The priority table is fixed throughout all extraction rounds; the label-free inference choice is discussed under reference availability below.

1. Initialize $R = \max(S, 0)$, $A_C(S) = 0$, an empty list of coefficient–matrix pairs, and $\lambda_{\max} = 0$.
2. If $R = 0$, return the accumulated extraction results. Otherwise reset the visited set and predecessor counts to d_0. Place source 0, and decrement the counts of all its successors along directed paths in G_C.
3. From the last placed node i, construct the candidate set:

$$\mathcal{A}(i, U) = \{j \notin U : d_U(j) = 0,\ M_{i,j} = 1,\ R_{i,j} > 0, \quad j \neq n-1 \text{ or } |U| = n-1\}. \tag{17}$$

   If this set is empty before completion, return a failed attempt and terminate the extraction procedure. Otherwise select $j = \arg\min_{v \in \mathcal{A}(i,U)} r(v)$, append j, mark it visited, and decrement the counts of every successor of j along a directed path in G_C. Repeat until all nodes are placed.
4. For the completed chain π, construct its reduced one-hot adjacency matrix $Q = Q_\pi$. Read the residual values on its $n-1$ edges and set λ to their minimum. Apply $R \leftarrow R - \lambda Q$ and $A_C(S) \leftarrow A_C(S) + \lambda Q$, recording (λ, Q). Clamp numerical zeros after subtraction. If $\lambda > \lambda_{\max}$, update $(\lambda_{\max}, Q_{\max}) \leftarrow (\lambda, Q)$. Return to Step 2.

In a batch, Step 4 reads the residual values on each graph's selected chain, takes their minimum, subtracts that coefficient from the selected entries, and adds the weighted permutation matrix to $A_C(S)$. Graphs whose attempts fail stop independently; successful graphs continue with their updated residuals. Ignoring numerical clipping, the invariant is $R^{(0)} = A_C(S) + R$. Thus termination with $R = 0$ gives a complete feasible decomposition of the initialized input, whereas failed completion of a chain returns the extracted components and the remaining residual. At inference, the decoder uses the extracted permutation matrices directly. During training, the weighted sum $A_C(S)$ is normalized by the total extracted weight to construct the feasible-component target in Section 3.4, provided that weight is positive.

Empirically, during decoding, to avoid allocating computational effort to successor permutation matrices with negligible weights λ_k while also limiting the impact of errors introduced by Sinkhorn normalization, we set entries in the residual matrix R that are smaller than $1/(n \times \mathrm{Thres})$ to 0 at each iteration. This excludes these entries from the allowable walkthrough. We set $\mathrm{Thres} = 100$ in our experiments, and the values of these entries are typically on the order of 10^{-5}, which is close to the numerical error introduced by 20 iterations of Sinkhorn normalization.

Feasibility. Every placed node has all its predecessors in the prefix. Consequently a completed chain is a topological ordering, and its reduced adjacency matrix is a feasible permutation matrix. No post-processing repair is needed for a returned permutation matrix. When the residual has no zero entries, the already placed nodes form the beginning of a topological ordering ending at i. If nodes remain, the remaining DAG has a node j with no unvisited immediate predecessors. This node cannot precede i, since all predecessors of a placed node have already been placed. If $i \prec_C j$ but there is no direct edge $i \to j$, a path from i to j contains an intermediate node that must appear after i and before j. That node cannot already be in an ordering ending at i, contradicting the availability of j. Hence j is either incomparable with i or a direct successor, so $M_{i,j} = 1$. The walkthrough with the predecessor check therefore always completes a topological ordering in this special case. Unique source and sink fix its endpoints, so its reduced adjacency matrix has the claimed matrix structure. In a BvN residual, requiring the additional condition $R_{i,j} > 0$ can instead remove all available successors of the already constructed ordering.

Complexity. The walkthroughs use at most $n-1$ candidate scans of size n. In the preliminary case, immediate-predecessor counter initialization and updates cost $O(n + |E_C|)$, with each edge processed once during updates. In the general procedure, there are at most $O(n^2)$ predecessor-counter decrements. Residual support tests and dense matrix initialization also cost $O(n^2)$; finding the minimum residual on a selected chain and subtracting it from the corresponding entries cost $O(n)$. Thus, during one attempt, including construction of a feasible permutation matrix when

successful, costs $O(n^2)$. Reference support testing requires only $O(n)$ edge accesses. These bounds assume that M and the predecessor/successor lists are already available. Finding directed paths by a graph traversal from each node separately costs $O(n(n+|E_C|))$ and is graph-level preprocessing, not part of the per-attempt bound. For m extracted permutation matrices and one final failed attempt, the total time is $O((m+1)n^2)$ after preprocessing. In exact arithmetic, every positive extraction removes at least one supported edge and no edge is reintroduced, so m is bounded by the initial number of supported entries, at most $(n-1)^2$.

Reference availability. At inference, the algorithm uses a fixed priority table based on the lexicographical order of node indices, without access to the supervision target. Evaluation does not use test-label positions. The training-time priority table is based on the reference ordering and is applied only during training.

B.5 Few-Step Inference and Feasible Decoding

Fig. 3 illustrates the diffusion and the feasible solution decoding procedure on a DAG scheduling instance with $n = 7$.

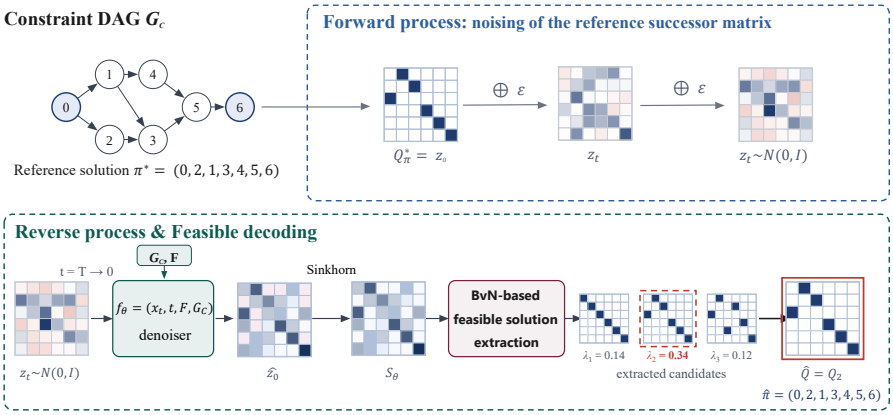


Figure 3: The diffusion and the feasible solution decoding procedure on a DAG scheduling instance with $n = 7$.

In the inplementation, the default diffusion horizon is $T = 200$, with the final indexed state $T-1 = 199$. Sampling initializes z_{T-1} as an i.i.d. Gaussian matrix. Rather than executing all reverse steps, the implementation selects a short decreasing time sequence. With `infer_mid_step=2`, this gives the initial level, two intermediate levels, and zero, hence three large transitions.

For a transition from t to $s < t$, the deterministic update combines the clean prediction and the implied noise estimate with the signal and noise scales of the next time level. No additional transition noise is injected, so the update is deterministic conditional on the initial Gaussian state. After the final transition, Sinkhorn yields S_θ. We apply BvN-based feasible solution extraction from Section 3.3.1, and the default decoder selects the returned matrix with the largest extracted coefficient.

Evaluation uses top-K inference with default $K = 4$. Each attempted trajectory has an independent initial Gaussian matrix. We sample trajectories until K solutions have been generated. We evaluates every generated feasible solution using the original problem-specific cost objective, such as makespan or another problem-specific cost, and retains the solution with minimum cost in Eq. (18). Thus top-K sampling exploits the diversity of the initial Gaussian states while preserving feasibility for every accepted solution.

$$k_{\text{best}} = \arg\min_{1 \le k \le K'} g\big(\widehat{\pi}^{(k)}\big), \qquad \widehat{\pi}^\star = \widehat{\pi}^{(k_{\text{best}})}, \tag{18}$$

B.6 Details about the Feasible Component Guidance

Fig. 4 illustrates the construction of the feasible component during the feasibility guidance. It decomposes S_θ into feasible permutation matrices Q_k with positive weights λ_k, rescales their weighted

sum by $\widehat{w}_C$ into the feasible component $S_{\theta,C}$, and minimizes the MSE between S_θ and $S_{\theta,C}$ (the loss term $\mathcal{L}_{\text{feas}}$), which pulls the output into the feasible polytope.

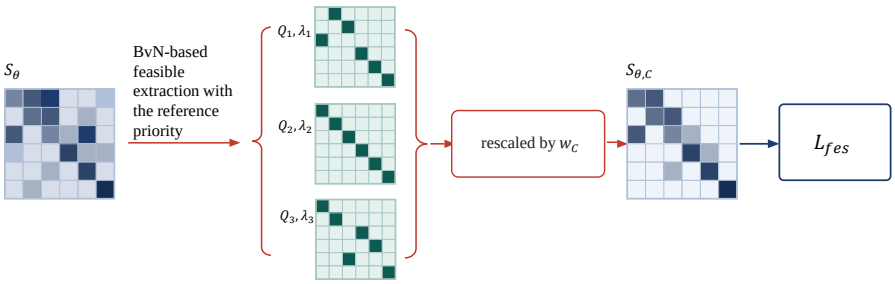


Figure 4: The construction of the feasible component during the feasibility guidance.

During training, the feasible component $S_{\theta,C}$ produced by the mapping in Section 3.4 is detached from the computational graph and used as the target in Eq. (8), and no gradient is propagated through matching selection, extraction coefficients, or the residual. The MSE instead provides the direct gradient $2(S_\theta - S_{\theta,C})/(n-1)^2$ with respect to S_θ, which is backpropagated through Sinkhorn normalization and the denoiser. The supervised loss in Eq. (10) separately propagates gradients from the fixed supervision target Q^{ref}.

B.7 Training Objective and Parameter Settings

The supervision target Q^{ref} supplies the clean diffusion target. Following the optimization consistency objective of Fast-T2T (Li et al., 2024), the implementation independently samples two time levels s and t and sums their DSM space MSE terms as $\mathcal{L}_{\text{sup}}(S_\theta, Q^{\text{ref}}, s, t)$ in Eq. (10). For the additional training guidance, feasible component mapping in Section 3.4 uses the extracted feasible permutation matrices from $S_\theta^{(u)}$ to construct $S_{\theta,C}^{(u)}$, used as a detached target for $\mathcal{L}_{\text{feas}}$ in Eq. (8).

The implementation uses the expected objective:

$$\begin{aligned}\mathcal{L}_{\text{impl}}(\theta) = \mathbb{E}_{\substack{(G_C,Q^{\text{ref}})\sim\mathcal{D},\ s,t\sim p \\ \epsilon_s,\epsilon_t\sim\mathcal{N}(0,I)}}\Big[& d_{\text{DSM}}\left(S_\theta^{(t)}, Q^{\text{ref}}\right) + d_{\text{DSM}}\left(S_\theta^{(s)}, Q^{\text{ref}}\right) \\ & + \lambda_{\text{feas}}\left(\mathcal{L}_{\text{feas}}(S_\theta^{(t)}; C) + \mathcal{L}_{\text{feas}}(S_\theta^{(s)}; C)\right)\Big].\end{aligned} \tag{19}$$

The time sampler uses $p(t) = (t+1)^\kappa / \sum_{r=0}^{T-1}(r+1)^\kappa$; setting $\kappa = 0$ recovers uniform sampling. The default training call independently draws two time levels t and s according to the expectation in Eq. (19). We set $\kappa = 1$ to strengthen learning in high-noise situations. Training uses a logical batch of 16 graphs, processed as micro-batches of 8 with gradient accumulation. We use an Adam-based optimizer for the network, with a learning rate of 10^{-3}. The current experiment configuration specifies up to 12,000 epochs.

C Details about the evaluation datasets and the reference solutions

C.1 Descriptions of the datasets

TSPLIB-SOP. The TSPLIB-SOP benchmark contains only 41 problem instances and is therefore used solely for evaluation. For training, we use the randomly generated instances provided by Löwens et al. (2022) as the training set. The reference solutions and corresponding cost values are obtained by LKH-3[4].

[4] http://webhotel4.ruc.dk/~keld/research/LKH-3/

C.2 TPC-H

TPC-H is an open-source workflow benchmark for decision support and web commerce, where the applications are in the form of DAG workflows. It is widely used for evaluating workflow scheduling algorithms under the homogeneous processor setting. Under this setting, the execution time of each task is fixed. The total amount of computational resources in the system is limited, and the total resources occupied by concurrently running tasks cannot exceed the system's maximum resource capacity.

C.3 PEGASUS

Pegasus is an open-source workflow benchmark for scientific computation. It is widely adopted as a benchmark for DAG workflow scheduling algorithms under heterogeneous multiprocessor systems. Under this setting, the execution time of each task depends on the computational resource to which it is allocated. We leave task-processor allocation to the widely adopted earliest-finish-time greedy rule and only consider task execution ordering. Pegasus contains several sub-datasets, e.g., GENOME, LIGO, and CYBERSHAKE, corresponding to different scientific computing applications. The number of instances in each sub-dataset is approximately the same (about 100) at the same problem scale. To evaluate the generalization capabilities of different methods, we average the metrics over the test sets collected from each sub-dataset. The execution times of task nodes across workflows from different computing applications are normalized to a common scale.

C.4 REFERENCE SOLUTIONS ON TPC-H AND PEGASUS

Reference selection. For each problem instance $\ell = \{V_\ell, C_\ell, g_\ell\}$, we select the minimum cost produced by the problem-specific heuristics and the topology-preserving metaheuristic as the reference baseline in the experimental results. For training, the corresponding reference solution π_ℓ^{ref} determines its successor permutation matrix $Q_\ell^{\mathrm{ref}} := Q_{\pi_\ell^{\mathrm{ref}}}$, which serves as the supervision target. For evaluation instances, $g_\ell(\pi_\ell^{\mathrm{ref}})$ is the reference baseline cost used to compute the relative gap.

Unlike the SOP, where the cost of a solution can be computed using a linear function, the cost function of the workflow scheduling problem, typically the makespan, generally depends on simulation. The inherent coupling between limited computational resources and partial-order constraints makes the cost functions of these workflow scheduling problems extremely difficult to evaluate using closed-form expressions. Converting the cost function and constraints into a linear programming relaxation may require an exponential number of auxiliary relaxation variables, which leads to long runtimes for exact solvers.Consequently, these problems generally cannot obtain exact reference solutions in the same manner as the SOP. In this work, we combine problem-specific heuristics with a topology-preserving metaheuristic to obtain high-quality feasible reference solutions.

Problem-specific heuristics. The heuristic depends on the specific DAG scheduling problem represented by each dataset. For TPC-H, we use the shortest-time-first heuristic. For Pegasus, we use the classical HEFT heuristic (Topcuoglu et al., 2002).

Topology-preserving metaheuristic. We use the topology-preserving genetic algorithm proposed by Zhu et al. (2016). In Zhu et al. (2016), the crossover and mutation operators preserve topological ordering and thus maintain precedence feasibility. For each optimization instance, we run the search with a population size of 400 for 1,000 generations.

D IMPLEMENTATION DETAILS ABOUT BASELINES

Open-source code is available for the baselines PPO-BiHyb (Wang et al., 2021)[5], Qi et al. (2025)[6], and Ebrahimi et al. (2026)[7].

[5] https://github.com/Thinklab-SJTU/PPO-BiHyb
[6] https://github.com/9erAlpenjaeger/DAG-CAI-DR
[7] https://github.com/sadjadeb/DiffusionRank

For Ebrahimi et al. (2026), it cannot guarantee that its generated candidate solutions satisfy the partial-order constraints, so we apply a post-processing repair to its raw orderings: we apply prioritized topological sorting on the constraint graph G_C, in which the priorities are the items' positions in the raw orderings. Items with positions that are ranked higher in the sorting process have higher priority.

D.1 Positional heatmap diffusion and Successor heatmap diffusion

Next, we introduce two heatmap-based baseline methods, implemented based on the Fast-T2T (Li et al., 2024) backbone. Li et al. (2024) is a discrete diffusion method for COP. It does not initially resolve partial-order constrained problems, i.e., DAG scheduling, and its predicted variable-wise probability heatmap cannot be directly decoded into a solution satisfying partial-order constraints for DAG scheduling. We conduct the following modifications for the heatmap decoding.

Positional Heatmap Diffusion The clean target is $P_{\pi^{\mathrm{ref}}} \in \{0,1\}^{n\times n}$, with rows representing items and columns representing positions. To decode the final heatmap, items are processed in row order. Each item is assigned to the highest-scored position that is not assigned so far. We can finally obtain a permutation matrix, which is converted into a raw ordering $\widehat{\pi}$ according to the positional interpretation. This candidate solution may not satisfy the partial-order constraints. So, before computing the application-specific cost g, a topological sorting repair is applied to $\widehat{\pi}$ to produce the repaired solution: items with positions that are ranked higher in the sorting process have higher priority.

Successor Heatmap Diffusion The clean target is Q^{ref} under the successor interpretation: $(Q^{\mathrm{ref}})_{i,j} = 1$ means that item j immediately follows item i. Noising and denoising operate only on the active $(n-1)\times(n-1)$ block. Decoding constructs a solution directly by a greedy topological walk, starting at source 0. At each step, it considers the unplaced items whose direct predecessors in G_C have all been placed, and chooses the item with the largest heatmap score from the most recently selected item. Repeating this rule until all items are placed gives a topological ordering of G_C. The heatmap thus supplies local preferences among currently eligible successors. The resulting feasible solution is passed to the evaluation routine. Precedence is already enforced during this walk.

Ablation without feasible solution decoding (w/o FSD). For the ablation without feasible solution decoding, we apply the Hungarian algorithm to the model output S_θ to obtain an $(n-1)\times(n-1)$ successor permutation matrix, and then convert this matrix into a raw ordering. Since the raw ordering is not guaranteed to satisfy the partial-order constraints, we apply prioritized topological sorting as a post-processing repair. This repair procedure iteratively sorts the node earliest in the raw ordering. The repaired ordering is used for cost evaluation, while the feasibility rate is computed from the raw ordering before repair.

E Wall-clock time of the feasible solution decoding from the DSMs

In the feasible solution decoding procedure from the DSMs, the number of feasible permutation matrices that can be extracted determines the time required for feasible solution decoding. We record the wall-clock time of feasible solution decoding for several problem scales in Table 4. In the early stage of training, the output of the diffusion model has not yet converged and is relatively uniform. At this time, the number of feasible permutation matrices that can be extracted is larger, and the time for feasible solution decoding is longer. In the later stage of training, the output of the diffusion model becomes more concentrated. At this time, the number of feasible permutation matrices that can be extracted is smaller, and the time for feasible solution decoding is significantly shorter.

F Limitation

In our experiments, the supervision targets are generated by heuristics and a metaheuristic, instead of a precise optimal solution, and may therefore be suboptimal. Thus, training may suffer from biases

Table 4: Wall-clock time of feasible solution decoding before training and after the full 12,000 epochs of training.

Scenario	Before training.	After training.
TPC-H-50 ($\bar{n}$=500)	0.68 s	0.03 s
TPC-H-100 ($\bar{n}$=912)	1.26 s	0.41 s
TPC-H-150 ($\bar{n}$=1385)	2.03 s	0.62 s

and quality limitations of these targets. The reported gap measures performance relative to the reference baseline, not distance to the global optimum; a small gap does not certify near-optimality. Evaluation against precise optimal solutions remains future work. However, this does not affect the fact that our method can learn to generate solutions that are competitive with the heuristic reference solutions.