Blocked Gibbs meets Diffusion Transformers: Unsupervised Learning for Constraint Optimization

Yudong W. Xu[1,2] **Wenhao Li**[1,2] **Xiaoyu Wang**[1] **Scott Sanner**[1,2] **Elias B. Khalil**[1]
[1] University of Toronto [2] Vector Institute
{wil.xu, chriswenhao.li, cnxiaoyu.wang}@mail.utoronto.ca
ssanner@mie.utoronto.ca, elias.khalil@utoronto.ca

Abstract

Diffusion models have shown promise in learning to solve constraint optimization problems. However, they are mostly restricted to problems with binary variables and rely on graph neural networks, hindering their application to a broader range of problems such as those with general discrete variables or constraint structures that necessitate global rather than local reasoning. We investigate the use of Diffusion Transformers to address the aforementioned limitations. A naive implementation performs poorly due to a fundamental mismatch between the standard diffusion process and constraint solving: while the former applies *small, incremental denoising* across *all* variables, the latter requires *substantially altering specific subsets of variables* to attain feasibility or optimality. Our method, ***Blo**cked **G**ibbs **Di**ffusion **T**ransformer* (BloGDiT), is the first to address this limitation by replacing standard joint Gaussian denoising with blocked Gaussian denoising. BloGDiT uses iterative block resampling and anneals the block size over time to facilitate large, targeted edits within a block of variables. Across Sudoku, Graph Coloring, Maximum Independent Set, and MaxCut, BloGDiT matches or outperforms existing methods, demonstrating that blocked Gibbs-style diffusion provides a highly effective inductive bias for Transformer-based constraint satisfaction and optimization.

1 Introduction

Constraint optimization is fundamental to many real-world applications. Such problems are often solved by iterative methods: a candidate assignment is proposed, evaluated, and then modified. This iteration repeats until a high-quality solution is found, subject to a computation budget. Classical sampling and local search methods can be highly effective, but they typically solve each instance from scratch and rely on hand-designed (assignment) proposal mechanisms. This raises a natural question: *can we learn a high-performance, reusable proposal mechanism whose cost is amortized across problem instances?* Diffusion models [7, 26, 62] provide an appealing answer. Their iterative denoising process can be interpreted as a learned refinement procedure: starting from a noisy assignment, the model repeatedly moves the state toward lower-energy regions. As illustrated in Figure 1, the dynamics of the learned algorithm stand in sharp contrast with pure discrete sampling-based search, which may explore excessively before reaching a good solution.

Recent works have applied diffusion models to combinatorial optimization and discrete reasoning [7, 83] but remain limited in scope. Most focus on binary graph problems and rely heavily on graph neural networks (GNNs) as the backbone architecture [56, 67, 86]. While GNNs are effective for sparse, fixed topologies, their reliance on local message-passing can make it difficult to capture global dependencies and long-range interactions [3, 45]. Many problems involve general non-binary variables and dense global constraints whose natural representations are not well captured by a pairwise constraint graph. To build a general-purpose neural heuristic, the architecture must be able to model complex, global dependencies across diverse problem types.

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

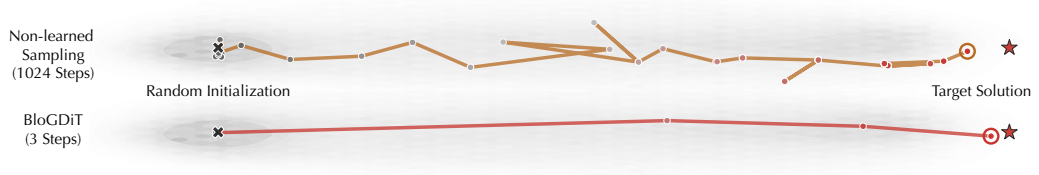

Figure 1: **Intuition for learned diffusion-guided refinement.** We visualize assignment trajectories for a Sudoku instance in a 2D projection of the solution space. While the non-learned baseline [19] takes many exploratory updates (many of which are clustered around the initial assignment), BloGDiT reaches the target in far fewer denoising steps. This illustrates how a learned diffusion model amortizes the cost of search through directed iterative refinement. Projection details are provided in Appendix K.

Diffusion Transformers (DiTs) offer this capability [49]. With their global attention mechanism [71], Transformers can inherently model interactions between any pair of variables and have demonstrated superior scaling behavior in generative modeling [32]. As a starting point, we found that swapping out a backbone GNN for a DiT is insufficient. A naive application of DiT using standard "global" Gaussian diffusion schedules performs poorly on constraint problems. In preliminary experiments, a vanilla DiT generalized poorly and generated suboptimal solutions. We attribute this to a mismatch between global denoising and discrete optimization. In constraint satisfaction, improving a near-feasible solution typically requires *substantially* changing *only a small subset of variables* (e.g., resolving a specific constraint that is currently infeasible). Standard global diffusion, which updates every variable at every step, struggles to perform these surgical edits without disrupting the rest of the assignment.

To bridge this gap, we draw inspiration from *blocked Gibbs sampling* [21, 22, 31] and recent work on Gibbs-style Diffusion [5]. Rather than treating the entire solution as a single joint distribution to be denoised at once, we view the generation process as a sequence of conditional updates. This perspective aligns naturally with *large neighborhood search* [51, 59], a foundational heuristic in operations research in which solutions are improved by iteratively "destroying" (unassigning) a subset of variables and "repairing" (re-optimizing) them while keeping the remaining variables fixed.

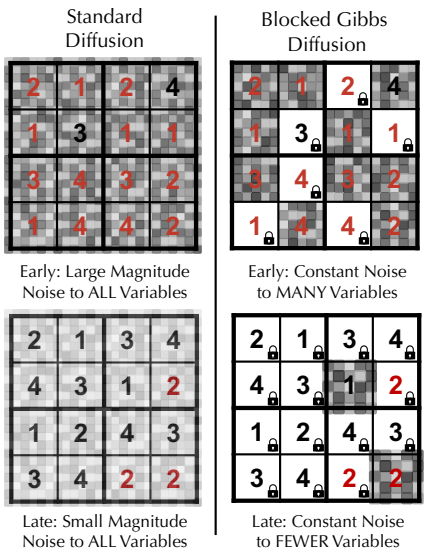

Figure 2: **Intuition for block-wise diffusion using a small Sudoku problem.** As a partial assignment becomes nearly feasible, maximum progress is made by resampling a small subset of variables, rather than applying small denoising updates everywhere.

Contributions. Building on these insights, we propose **BloGDiT**, a **Blo**cked **G**ibbs **Di**ffusion **T**ransformer. Unlike standard diffusion models that operate in the full joint space, BloGDiT operates via iterative *block resampling*. At each step, we apply continuous Gaussian noise to a specific *subset* (block) of variables—effectively treating them as unknown or corrupted—and use the Transformer to denoise this block conditional on the remaining "clean" variables. This allows the model to focus its capacity on repairing sub-regions of the variable assignment. Crucially, we anneal the block size over time: early steps modify large blocks for global exploration, while later steps focus on smaller blocks for precise local refinement (Figure 2).

To train BloGDiT, we derive a mask-augmented joint KL upper bound for unsupervised training on Boltzmann targets and instantiate it with a masked Gaussian logit-space diffusion, yielding a tractable objective with closed-form entropy and forward noise-matching terms.

We evaluate BloGDiT on benchmark instances from four well-studied constraint satisfaction and optimization problems: Sudoku, Graph Coloring, Maximum Independent Set, and MaxCut. These problems vary in the types of constraints they involve, instance sizes, and variable domains (binary

vs. categorical). Compared to a wide variety of methods that span the exact Google OR-Tools solver, pure sampling, and learning-based heuristics, BloGDiT achieves state-of-the-art or competitive performance, validating that the combination of Transformer backbones and blocked Gibbs updates provides an effective inductive bias for general constraint reasoning.

2 Background

2.1 Constraint Programming

A *constraint satisfaction problem* (CSP) is specified by a tuple $(\mathcal{V}, D, C)$, where $\mathcal{V} = \{V^{(1)}, \ldots, V^{(n)}\}$ is a set of variables, $D = \{D^{(1)}, \ldots, D^{(n)}\}$ is a collection of finite discrete domains, and $C = \{c_1, \ldots, c_m\}$ is a set of constraints. Each constraint c_j is a relation over a subset of variables, restricting which joint assignments are allowed. Let $\mathcal{X} \triangleq \prod_{i=1}^{n} D^{(i)}$ denote the Cartesian product over variable domains, i.e., $X = (x^{(1)}, \ldots, x^{(n)}) \in \mathcal{X}$. The goal in a CSP is to find an assignment $X \in \mathcal{X}$ that satisfies all constraints: $c_j(X) = \texttt{true}, \forall j \in [m]$, and a *constraint optimization problem* (COP) adds an additional objective: $\min_{X \in \mathcal{X}} f(X)$. *Constraint programming* (CP) studies modeling languages and algorithms for CSPs/COPs, with a focus on expressive *global constraints* that compactly capture common combinatorial structure [10, 12, 54]. A common example is ALLDIFFERENT, which enforces that a subset of variables take on distinct values [53].

2.2 CP as sampling a Boltzmann distribution

A probabilistic view of CP consists in defining an *energy* $H : \mathcal{X} \to \mathbb{R}$ whose low-energy assignments are desirable (i.e., constraint-feasible), then considering the associated Boltzmann distribution at temperature $\tau > 0$:

$$p_B(X) = \frac{\exp\left(-H(X)/\tau\right)}{Z(\tau)}, \text{where } Z(\tau) = \sum_{X' \in \mathcal{X}} \exp\left(-H(X')/\tau\right).$$

In the most general COP case, the energy combines the objective with constraint penalties:

$$H(X) = f(X) + \sum_{j=1}^{m} \lambda_j \phi_j(X), \tag{1}$$

where each $\phi_j(X) \geq 0$ satisfies $\phi_j(X) = 0$ if and only if constraint c_j is satisfied, and $\lambda_j \geq 0$ is set to ensure feasible solutions are prioritized (e.g., a Lagrangian-style relaxation [76]). For CSPs, one may take $H(X) = \sum_j \lambda_j \phi_j(X)$. As $\tau \to 0$, the Boltzmann distribution concentrates on (assignments near) global minimizers of H. In practice, if one could efficiently sample from $p_B(X)$, then near-optimal solutions are likely to be found.

2.3 Unsupervised diffusion on Boltzmann distributions

In practice, sampling from $p_B(X)$ is often intractable due to the normalization factor $Z(\tau)$. A common approach is therefore to approximate $p_B(X)$ with a learned surrogate $q_\theta(X)$ that can be sampled from efficiently. Diffusion models provide one such approach. However, when data samples are not easily available, which is often the case for complex CSP/COP problems, the standard diffusion objective is not available. We take inspiration from recent approaches that apply diffusion-based learning to sample from Boltzmann densities in an unsupervised setting [56, 75].

Formally, let $q_\theta(X_0)$ denote the (generally intractable) model-implied marginal at time 0, induced by a reverse-time Markov chain with parameters θ. We seek to minimize the reverse KL divergence, $\mathrm{KL}\left(q_\theta(X_0) \| p_B(X_0)\right)$. As in Sanokowski et al. [56] and Wang et al. [75], we upper bound this marginal KL by a joint KL over trajectories $X_{0:T}$: $\mathrm{KL}\left(q_\theta(X_0) \| p_B(X_0)\right) \leq \mathrm{KL}\left(q_\theta(X_{0:T}) \| p(X_{0:T})\right)$, where the forward (corruption) and reverse (generative) trajectory distributions are defined as

$$p(X_{0:T}) = p_B(X_0) \prod_{t=1}^{T} p(X_t \mid X_{t-1}), \quad q_\theta(X_{0:T}) = q(X_T) \prod_{t=1}^{T} q_\theta(X_{t-1} \mid X_t), \tag{2}$$

with $q(X_T)$ a fixed prior noise distribution and $\{p_t\}_{t=1}^{T}$ a fixed forward noising process.

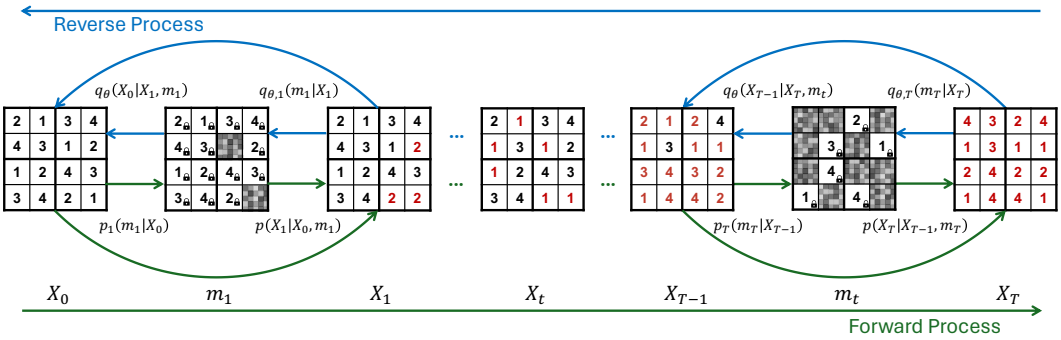

Figure 3: **Illustration of the blocked Gibbs diffusion process on a small Sudoku problem.** The reverse chain (blue, top) is initialized from a high-noise Gaussian prior $Z_T \sim \mathcal{N}(0, s^2 I)$, which decodes to a random discrete assignment X_T. At step t, a binary mask m_t is sampled to select a block of variables to update; the model samples $X_{t-1}^{m_t} \sim q_\theta(\cdot \mid X_t, m_t)$ while copying $X_{t-1}^{\neg m_t} = X_t^{\neg m_t}$ as-is, yielding blocked Gibbs-style resampling moves. The masking rate (block size) is annealed over time, transitioning from coarse global edits to fine local refinements. After T steps, the final sample X_0 is expected to approximate the Boltzmann target $p_B(X_0) \propto \exp(-H(X_0)/\tau)$.

3 Blocked Gibbs Diffusion

To overcome limitations of existing diffusion models (cf. Section 1), we advocate for a blocked Gibbs diffusion procedure that proposes large, targeted resampling updates within the block. By annealing the block size over time, the diffusion transitions from coarse global edits to local refinements. In this section, we define the mask-augmented forward and reverse processes underlying blocked Gibbs diffusion and derive its training objective. We describe our practical instantiation in Section 4.

3.1 Auxiliary mask variable

To model *block* updates, we introduce an auxiliary binary mask m_t at each step t. Concretely, the i-th variable is associated with a bit mask $m_t^{(i)} \in \{0, 1\}$: if $m_t^{(i)} = 1$, $x^{(i)}$ is *updated* at step t (in both the forward and reverse transitions); if $m_t^{(i)} = 0$, $x^{(i)}$ is *frozen* and copied through the transition unchanged.

Mask-augmented transitions. We augment both the reverse and forward transition kernels with auxiliary m_t variables:

$$q_\theta(X_{t-1} \mid X_t) = \sum\nolimits_{m_t} q_{\theta,t}(m_t \mid X_t) q_\theta(X_{t-1} \mid X_t, m_t), \tag{3}$$

$$p(X_t \mid X_{t-1}) = \sum\nolimits_{m_t} p_t(m_t \mid X_{t-1}) p(X_t \mid X_{t-1}, m_t). \tag{4}$$

where $q_{\theta,t}(m_t \mid X_t)$ and $p_t(m_t \mid X_{t-1})$ are the reverse/forward mask-selection distributions.

The transition distributions leave unmasked variables unchanged. We define this property component-wise: for the i-th variable, the reverse transition is given by:

$$q_\theta(x_{t-1}^{(i)} \mid X_t, m_t) = \begin{cases} \mathbb{I}\{x_{t-1}^{(i)} = x_t^{(i)}\} & \text{if } m_t^{(i)} = 0, \\ q_\theta\left(x_{t-1}^{(i)} \mid X_t\right) & \text{if } m_t^{(i)} = 1, \end{cases}$$

where $\mathbb{I}\{\cdot\}$ is the indicator function. The forward process $p(x_t^{(i)} \mid X_{t-1}, m_t)$ is defined analogously, ensuring $x_t^{(i)} = x_{t-1}^{(i)}$ whenever the variable is unmasked.

Augmented joint distributions. We can equivalently define forward and reverse processes on the augmented space $(X_{0:T}, m_{1:T})$ by factorizing each step through the mask:

$$p(X_{0:T}, m_{1:T}) = p_B(X_0) \prod_{t=1}^{T} p_t(m_t \mid X_{t-1})\, p(X_t \mid X_{t-1}, m_t), \tag{5}$$

$$q_\theta(X_{0:T}, m_{1:T}) = q(X_T) \prod_{t=1}^{T} q_{\theta,t}(m_t \mid X_t)\, q_\theta(X_{t-1} \mid X_t, m_t). \tag{6}$$

Importantly, introducing the auxiliary variables $m_{1:T}$ does not change the marginal trajectory distribution over $X_{0:T}$, as stated in Proposition 3.1 with proof in Appendix B.

Proposition 3.1 (Sampling equivalence under mask augmentation)**.** *Let $(X_{0:T}, m_{1:T})$ be sampled from the augmented joint distribution* (6)*. By discarding $m_{1:T}$, the resulting trajectory $X_{0:T}$ is distributed as the original process $q_\theta(X_{0:T})$ induced by the mask-augmented transitions* (3)*: $X_{0:T} \sim q_\theta(X_{0:T}) \iff \sum_{m_{1:T}} q_\theta(X_{0:T}, m_{1:T}) = q_\theta(X_{0:T})$.*

3.2 Mask-augmented joint upper bound

Our new joint upper bound can be defined similarly as $\mathrm{KL}\left(q_\theta(X_0) \| p_B(X_0)\right) \leq \mathrm{KL}\left(q_\theta(X_{0:T}, m_{1:T}) \| p(X_{0:T}, m_{1:T})\right)$. Substituting (5)–(6) and collecting terms yields the following decomposition, as derived in full in Appendix A:

$$\begin{aligned}\tau\,\mathrm{KL}\left(q_\theta(X_{0:T}, m_{1:T}) \| p(X_{0:T}, m_{1:T})\right) = \underbrace{\mathbb{E}_{q_\theta}\left[H(X_0)\right]}_{\text{Energy}} - \tau \underbrace{\sum_{t=1}^{T} \mathbb{E}_{X_{T:t}, m_{T:t} \sim q_\theta}\left[h\big(q_\theta(X_{t-1} \mid X_t, m_t)\big)\right]}_{\text{Entropy}} \\ - \tau \underbrace{\sum_{t=1}^{T} \mathbb{E}_{X_{T:t}, m_{T:t} \sim q_\theta}\left[\mathbb{E}_{X_{t-1}} \log p(X_t \mid X_{t-1}, m_t)\right]}_{\text{Forward noise matching}} + \tau \underbrace{\mathbb{E}_{X_{0:T}, m_{1:T} \sim q_\theta}\left[\sum_{t=1}^{T} \log \frac{q_{\theta,t}(m_t \mid X_t)}{p_t(m_t \mid X_{t-1})}\right]}_{\text{Mask distribution matching}} + C.\end{aligned} \tag{7}$$

The first term on the right hand side is the energy term which encourages the final sample from the diffusion process X_0 to have low energy as defined in Equation (1). The second term contains the differential entropy $h(q_\theta(X_{t-1} \mid X_t, m_t))$ over $q_\theta(X_{t-1} \mid X_t, m_t)$ which encourages diverse samples at each step of diffusion from the model. The third term ensures that the new samples at each step of the diffusion procedure match the known forward noise distribution; when $p_t(\cdot \mid \cdot, m_t)$ is chosen from a tractable family, the inner expectation is available in closed form. The final term ensures the mask distributions from the forward and reverse processes match.

4 Blocked Gibbs Diffusion Transformer for Constraint Optimization

This section describes the practical instantiation of our masked reverse-time chain. We use a continuous relaxation of the discrete assignments and perform diffusion *in logit space*. Assuming all variables share a common domain size $K \triangleq |D^{(i)}|$, we represent the diffusion state at time t as a matrix of logits $Z_t \in \mathbb{R}^{n \times K}$, where row i corresponds to variable $x^{(i)}$ and column k corresponds to the k-th discrete value. The relaxed assignment is obtained by applying a row-wise softmax, $\tilde{X}_t = \mathrm{softmax}(Z_t) \in [0,1]^{n \times K}$, so that each row $\tilde{x}_t^{(i,:)}$ lies on the simplex Δ^{K-1}. We apply the forward corruption and reverse denoising transitions to Z_t, while energies and constraints are evaluated on the corresponding relaxed variables $\tilde{X}_t$. At sampling time, we output a discrete assignment via $\hat{x}^{(i)} = \arg\max_k \tilde{x}_0^{(i,k)}$. For notational convenience, we identify matrices in $\mathbb{R}^{n \times K}$ with their vectorizations in $\mathbb{R}^{nK}$ and omit explicit $\mathrm{vec}(\cdot)$ operators.

Transformer architecture. We instantiate the denoiser with the Transformer backbone of Xu et al. [79], but perform diffusion over continuous logits rather than variable assignments, similar to Chen et al. [14]. Instead of outputting the next assignment, we output the mean and variance of the logits, which we then decode into variable assignments. The full Transformer architecture including its positional encodings and self-attention mechanism are described in Appendix E.

Energy function. We design constraint-specific energies by defining continuous, differentiable penalties $\phi_j(\tilde{X}) \geq 0$ such that $\phi_j(\tilde{X}) = 0 \iff c_j(\hat{X}) = \texttt{true}$, where $\hat{X}$ denotes the discretization induced by $\tilde{X}$ (e.g., row-wise argmax). The overall energy is the weighted sum of the objective and penalties as in Equation (1).

Noise distribution. We employ a masked Gaussian forward step kernel:

$$p(Z_t \mid Z_{t-1}, m_t) = \mathcal{N}\Big(Z_{t-1}, \sigma^2 \mathrm{diag}(\tilde{m}_t)\Big),$$

where $\sigma > 0$ is fixed, and $\tilde{m}_t \in \{0,1\}^{n\times K}$ is the variable-level mask broadcast to the logit dimensions, i.e., $\tilde{m}_t^{(i,k)} = m_t^{(i)}$. Equivalently, $Z_t = Z_{t-1} + (\tilde{m}_t \odot \epsilon_t)$ where $\epsilon_t \sim \mathcal{N}(0, \sigma^2 I)$, with unmasked coordinates being copied as-is. Repeated application yields a marginal variance that grows with t on the masked coordinates.

We parameterize the masked reverse step with a diagonal-covariance Gaussian:

$$q_\theta(Z_{t-1} \mid Z_t, m_t) = \mathcal{N}\big(\mu_\theta(Z_t, m_t), \mathrm{diag}(\sigma_\theta^2(Z_t, m_t))\big),$$

where the model outputs $\mu_\theta \in \mathbb{R}^{n\times K}$ and $\log \sigma_\theta^2 \in \mathbb{R}^{n\times K}$. To enforce exact copying on unmasked coordinates, we set $\mu_\theta^{\neg\tilde{m}_t} = Z_t^{\neg\tilde{m}_t}, \sigma_\theta^{\neg\tilde{m}_t} = 0$, and only sample the masked coordinates. The corresponding X_{t-1} is obtained by applying per-variable softmax to Z_{t-1}.

Closed form for entropy and noise-matching term. The differential entropy of a diagonal Gaussian restricted to the masked coordinates admits the closed form

$$h\left[q_\theta(Z_{t-1} \mid Z_t, m_t)\right] = \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{K} m_t^{(i)}\Big(\log(2\pi e) + \log \sigma_{\theta,i,k}^2(Z_t, m_t)\Big).$$

where only masked dimensions contribute. Under the masked Gaussian forward kernel, the inner expectation in the forward noise-matching term has a closed form:

$$\mathbb{E}_{q_\theta(Z_{t-1}\mid Z_t, m_t)}\left[-\log p_t(Z_t \mid Z_{t-1}, m_t)\right] = \frac{1}{2\sigma^2}\Big(\|\tilde{m}_t \odot (Z_t - \mu_\theta)\|_2^2 + \mathrm{tr}(\mathrm{diag}(\tilde{m}_t)\Sigma_\theta)\Big) + \mathrm{const.}$$

where $\Sigma_\theta = \mathrm{diag}(\sigma_\theta^2(Z_t, m_t))$ is the covariance matrix and const does not depend on θ.

Base distribution at Z_T. We initialize the chain from a fixed Gaussian base distribution: $Z_T \sim q(Z_T), q(Z_T) = \mathcal{N}(0, s^2 I)$, where s is a fixed hyperparameter (optionally with a constant bias on selected coordinates). In constrained settings with given variable values (e.g., Sudoku, where some variables are fixed), we clamp the corresponding coordinates and sample the remaining ones, which can be viewed as drawing from a conditional base distribution $q(Z_T \mid \text{givens})$.

Mask distribution. At each step $t \in [T]$, we sample a mask $m_t \in \{0,1\}^n$ indicating the subset of variables to update. We choose a *time-dependent, state-agnostic* mask schedule shared by forward and reverse processes,

$$q_{\theta,t}(m_t \mid X_t) \equiv p_t(m_t \mid X_{t-1}) \equiv \pi_t(m_t) = \prod_{i=1}^{n} \mathrm{Bernoulli}\big(m_t^{(i)}; \rho_t\big). \tag{8}$$

with masking rate ρ_t. We parameterize the block schedule by a minimum and maximum masking rate $[\rho_{\min}, \rho_{\max}]$ and anneal ρ_t over time to transition from large to small block updates; exact schedules are given in Appendix C. Because we use the same schedule for both forward and reverse mask variables, the last term in Equation (7) vanishes.

Adaptive mask selection. A practical advantage of the blocked Gibbs formulation is that it exposes an explicit mask-selection step: at each reverse transition during inference, the sampler can decide which subset of variables to update. This makes it possible to replace random block selection with adaptive, state-dependent selection strategies that focus on variables most likely to benefit from updating. In contrast, a standard DiT updates all variables jointly at every denoising step and therefore has no analogous mechanism for prioritizing variables. We use three representative strategies. First is a confidence-margin strategy, inspired by adaptive token selection in masked diffusion [35], which prioritizes variables for which the model is uncertain. The second and third are inspired by classical Large Neighborhood Search [51, 59]: a violation-based strategy, which prioritizes variables involved in violated constraints, and a related-variable strategy, which selects variables that participate in a common constraint. We treat the choice of selection strategy as a hyperparameter and report the best-performing strategy for each benchmark; formal definitions are provided in Appendix D.

5 Experiments

The experiments seek to answer the following research questions: **RQ1:** Is diffusion a useful learning paradigm compared to simpler iterative Transformer baselines? **RQ2:** Does blocked Gibbs diffusion improve over standard diffusion Transformers? **RQ3:** Does using the more general Transformer architecture degrade performance relative to methods that are specialized for binary problems? **RQ4:** Can BloGDiT outperform non-learned state-of-the-art methods? Our code is available at `https://github.com/khalil-research/BloGDiT`.

5.1 Problem selection

We evaluate BloGDiT on a diverse set of benchmarks from the literature, following their respective evaluation conventions. For each problem, we define a CP formulation (Appendix F) and a continuous relaxation penalty for each constraint (Table 1), resulting in a problem-specific energy for training. For COPs, we use a weighted sum of the constraint penalties and the objective (cf. Equation (1)).

Sudoku is a CSP defined on a 9×9 grid with variables taking values in $\{1, \ldots, 9\}$. The goal is to fill the empty cells such that each row, column, and 3×3 sub-grid contains each digit exactly once. A single instance consists of a partially filled board ("givens") and a completion that satisfies all constraints. Instance difficulty is determined by the number and placement of givens. We encode Sudoku using ALLDIFFERENT constraints over rows, columns, and sub-grids. Following Xu et al. [79], we train and evaluate on the SATNet dataset [74] of instances with 31–42 given cells, and test for *out-of-distribution* (OOD) generalization on the RRN dataset [47] of harder instances with 17–34 given cells.

Table 1: Discrete constraints $c(X)$ used in our four problems and their continuous penalty functions, $\phi_c(X)$. In the continuous penalties, each discrete variable assignment $x^{(i)}$ is represented by a probability vector approximating its one-hot encoding, with entries $\tilde{x}^{(i,k)} \in [0, 1]$ for $k \in [K]$, where $K = |D^{(i)}|$ is the common domain size.

Discrete Constraint $c(X)$	**Continuous Penalty** $\phi_c(\tilde{X})$
$\text{ALLDIFFERENT}_{m=n}(x^{(1)}, \ldots, x^{(n)})$	$\sum_{k=1}^{K} \left(\lvert 1 - \sum_{i=1}^{n} \tilde{x}^{(i,k)} \rvert\right)$
$x^{(i)} \neq x^{(j)}$	$\sum_{k=1}^{K} (\tilde{x}^{(i,k)} \cdot \tilde{x}^{(j,k)})$
$\forall (i,j) \in E: \; x^{(i)} + x^{(j)} \leq 1$	$\sum_{(i,j) \in E} \left(\tilde{x}^{(i,1)} \cdot \tilde{x}^{(j,1)}\right)$

Graph Coloring asks for an assignment of k colors to vertices such that adjacent vertices have different colors. We generate two sets of instances for $k = 5$ and $k = 10$. Training graphs have 50 vertices for $k = 5$ and 100 vertices for $k = 10$. To test OOD generalization, we evaluate on larger graphs with 100 vertices for $k = 5$ and 200 vertices for $k = 10$. We use pairwise not-equal constraints $x^{(i)} \neq x^{(j)}$ for each edge (i, j) to capture the coloring requirement.

MaxCut is a COP that partitions the vertices of a graph into two sets such that the total count of cut edges is maximized. We train on Erdős–Rényi graphs with $n = 50$ and edge probability sampled uniformly from $(0.05, 0.3)$. We evaluate large-scale generalization on benchmark instances from the GSET dataset [84], which consists of graphs with 800–10,000 vertices. Following existing works, we report the gap to the best known cut sizes [42].

Maximum Independent Set (MIS) is a COP that seeks the largest subset of vertices with no pairwise edges between them. Following Sanokowski et al. [56], we generate graphs using the RB-Model [78]. We report results on RB-small and RB-large with graph sizes between 200-300 and 800-1200 nodes.

5.2 Results

Table 2 summarizes our main results across problems. We compare against classical solvers, learning-based and non-learning-based heuristics. Baseline descriptions and sources of reported results are provided in Appendix I, while computational protocols and their differences are discussed in Appendix J. The best and second-best values are bolded and underlined, respectively. DiT denotes a vanilla diffusion Transformer with standard Gaussian denoising over all variables. BloGDiT denotes our method, using blocked Gibbs diffusion with adaptive mask selection as described in Section 4. OR-Tools is a state-of-the-art exact CP solver and serves as a strong baseline [50], achieving 100% on Sudoku instances. OR-Tools uses tree search along with sophisticated heuristics and thus should be seen as representative of the best algorithmic practices from the CP community; indeed, it has won gold in the international constraint programming competition every year since 2013 [65].

Table 2: Performance comparison. Higher is better for MIS (average objective value), Sudoku, and Graph Coloring (% of instances solved). Lower is better for MaxCut (average absolute gap to best-known solution). Bold indicates the best result and underline indicates the second-best result.

Sudoku	Easy ↑	Hard ↑
OR-Tools	**100**	**100**
SATNet	98.3	3.2
RRN	99.8	28.6
Yang et al. [82]	**100**	32.9
IRED	99.4	62.1
RLSA	4.2	1.8
ConsFormer	**100**	77.7
DiT	**100**	48.6
BloGDiT	**100**	<u>94.1</u>

Graph Coloring	5 Colors		10 Colors	
	$n = 50 \uparrow$	$n = 100 \uparrow$	$n = 100 \uparrow$	$n = 200 \uparrow$
OR-Tools	**83.08**	**57.16**	52.41	10.25
Greedy	32.42	0.00	0.75	0.00
Feasibility Jump	82.83	54.50	35.66	6.00
ANYCSP	79.17	34.83	0.00	0.00
RLSA	37.83	0.42	10.75	2.50
ConsFormer	81.00	47.33	<u>52.60</u>	11.92
DiT	79.75	48.91	50.50	<u>14.25</u>
BloGDiT	<u>83.00</u>	<u>54.67</u>	**53.92**	**18.58**

MIS	RB-small ↑	RB-large ↑
OR-Tools	**20.10**	**42.66**
LwD	19.01	32.32
INTEL	18.47	34.47
DGL	17.36	34.50
LTFT	19.18	37.48
DIFUSCO	18.52	–
RLSA	19.97	<u>40.19</u>
X^2GNN	<u>20.07</u>	39.28
DiffUCO	18.88	38.10
DiT	8.05	0.29
BloGDiT	19.52	37.05

MaxCut	$\lvert V\rvert = 800 \downarrow$	$\lvert V\rvert = 1K \downarrow$	$\lvert V\rvert = 2K \downarrow$	$\lvert V\rvert \geq 3K \downarrow$
OR-Tools	143.89	112.78	365.89	378.62
RUNCSP	185.89	156.56	357.33	401.00
ECO-DQN	65.11	54.67	157.00	428.25
ECORD	8.67	8.78	39.22	187.75
ANYCSP	<u>1.22</u>	<u>2.44</u>	13.11	51.63
RLSA	3.22	4.00	<u>13.0</u>	66.75
X^2GNN	21.11	6.00	22.44	<u>46.38</u>
DiffUCO	4.11	6.33	31.67	116.75
ConsFormer	16.33	12.44	52.11	115.25
DiT	56.11	50.00	101.33	162.00
BloGDiT	**0.11**	**1.33**	**7.88**	**32.75**

RQ1: Diffusion vs. iterative Transformers. We compare with ConsFormer [79] as the baseline (non-diffusion) iterative Transformer model which achieved state-of-the-art result on OOD Sudoku tasks. BloGDiT consistently improves over ConsFormer, with the largest gains in harder OOD settings: Sudoku OOD improves from 77.7% to 94.1%, and Graph Coloring improves across both $k = 5$ and $k = 10$, especially OOD. On MaxCut, BloGDiT also substantially improves over ConsFormer across all GSET size buckets. These results suggest that the diffusion objective provides a more principled framework for learning iterative heuristics, by grounding training in a reverse-KL objective toward a Boltzmann target rather than directly training a Transformer to improve assignments.

RQ2: Blocked Gibbs vs. standard diffusion. We compare against the standard DiT baseline, which uses the same Transformer backbone, training data, unsupervised objective, optimization settings, and inference budget but applies Gaussian denoising to all variables at each step. This isolates the effect of replacing global denoising with block-wise resampling. While DiT is competitive in some in-distribution settings, it degrades more sharply on harder instances: Sudoku OOD drops to 48.6% compared to 94.1% for BloGDiT, and MIS reaches average independent set sizes of only 8.05 and 0.29 on RB-small and RB-large, compared to 19.52 and 37.05 for BloGDiT. BloGDiT also improves over DiT on OOD Graph Coloring and reduces MaxCut gaps across all GSET sizes. These trends support our hypothesis that constraint optimization benefits from targeted block-wise resampling, where most variables can be preserved while selected variables are substantially revised.

RQ3: Transformer vs graph-specialized baselines. We compare against learned baselines for binary graph optimization, such as DIFUSCO [67], DiffUCO [56] and X^2GNN [1], which incorporate graph-specific inductive biases. These methods are designed for binary graph problems, so this comparison tests whether our more general Transformer-based framework sacrifices performance for broader applicability. On MIS, BloGDiT remains competitive but does not uniformly dominate the graph-specialized methods. In contrast, on MaxCut, BloGDiT achieves the best performance across all graph sizes. These results suggest that BloGDiT can remain competitive on binary graph optimization while also supporting non-binary CSPs such as Sudoku and Graph Coloring. To further examine the effect of the transformer backbone, we additionally experiment with a GCN architecture (Table 3). The Transformer achieves substantially better results on larger instances, particularly MIS-RB-large

Table 3: Ablation results. *−AMS* uses random mask selection, *−Entropy* trains without the entropy term, and *w/ GCN* replaces the Transformer backbone with a parameter-matched GCN. Sudoku and Graph Coloring values are percentages of solved instances (↑).

Dataset	BloGDiT	*−AMS*	*−Entropy*	*w/ GCN*
Sudoku-Easy	**100.00**	**100.00**	**100.00**	**100.00**
Sudoku-Hard	**94.12**	88.21	91.92	91.00
Coloring-5 ($n = 50$)	**83.00**	82.92	**83.00**	82.83
Coloring-5 ($n = 100$)	**54.67**	54.58	53.42	53.00
Coloring-10 ($n = 100$)	**53.92**	**53.92**	**53.92**	**53.92**
Coloring-10 ($n = 200$)	**18.58**	17.42	18.25	18.00
MIS-RB-small (size ↑)	**19.52**	15.83	18.99	19.05
MIS-RB-large (size ↑)	**37.05**	17.16	35.03	19.99
MaxCut-800 (gap ↓)	**0.11**	2.11	1.22	4.67
MaxCut-1K (gap ↓)	**1.33**	3.22	2.67	9.56
MaxCut-2K (gap ↓)	**7.88**	18.89	13.33	28.33
MaxCut-≥3K (gap ↓)	**32.75**	88.63	61.00	79.75

Table 4: Masking rate ablation on MIS. The masking rate is annealed from $\rho_{\max}$ to $\rho_{\min}$ following a geometric schedule (cf. Appendix C). Values are mean independent set sizes.

$\rho_{\max}$	$\rho_{\min}$	RB-Small	RB-Large
0.3	0.1	5.45	6.78
0.5	0.1	5.21	9.93
0.5	0.3	4.78	6.14
0.7	0.1	12.46	33.17
0.7	0.3	14.80	29.30
0.7	0.5	17.01	31.05
0.9	0.1	18.67	34.62
0.9	0.3	**19.52**	**37.05**
0.9	0.5	<u>19.49</u>	<u>35.41</u>
0.9	0.7	18.58	4.24

(37.05 vs. 19.99) and MaxCut with $|V| \geq 3$K (32.75 vs. 79.75 gap), suggesting that the Transformer backbone can benefit constraint optimization.

RQ4: Comparison with non-learned solvers. We compare BloGDiT against state-of-the-art non-learned solvers, including OR-Tools [50] and RLSA [19]. On Sudoku, OR-Tools achieves 100% accuracy on both splits and outperforms all learned methods. Nevertheless, BloGDiT matches OR-Tools on the easy split and achieves the strongest learned-method result on the harder OOD split, with 94.1% accuracy. On Graph Coloring, BloGDiT is competitive with OR-Tools for $k = 5$ and outperforms it for $k = 10$, improving accuracy from 10.25% to 18.58% on the hardest instances. RLSA performs poorly on both Sudoku and Graph Coloring, which is expected since it is designed primarily for binary optimization rather than non-binary CSPs. On MIS, OR-Tools obtains the strongest results overall. On MaxCut, however, BloGDiT outperforms both OR-Tools and RLSA across all GSET size buckets. Overall, no method is universally best. BloGDiT provides a learned heuristic that performs strongly across both non-binary CSPs and large-scale binary graph optimization tasks, remaining competitive with or improving over non-learned baselines.

5.3 Ablations

Masking rate. We ablate the mask schedule by sweeping the masking rates $\rho_{\max}$ and $\rho_{\min}$. The results are shown in Table 4. The main takeaway is that the schedule should start with large updates and gradually anneal to smaller updates. Small initial update ratios lead to poor performance, suggesting that large early updates are needed to make global changes. At the same time, keeping the update ratio too large until the end is also harmful for generalization: $(\rho_{\max}, \rho_{\min}) = (0.9, 0.7)$ performs well on RB-Small but collapses on RB-Large, indicating that late-stage updates should become smaller to allow fine-grained repair. We extend this analysis in Appendix N.3.

Adaptive mask selection. Table 3 ablates adaptive mask selection (AMS) by replacing it with random selection while keeping the same masking rate. AMS consistently improves performance, with the largest gains appearing on harder optimization settings. On Sudoku-Hard, AMS improves accuracy from 88.21% to 94.12%. On Graph Coloring, the gains are smaller but consistent. The effect is much larger on MIS and MaxCut: on MIS-RB-large, AMS improves the average independent set size from 17.16 to 37.05, and consistently reduces the average optimality gap for all MaxCut sizes. These results indicate that blocked diffusion benefits substantially from selecting which variables to update, rather than choosing blocks blindly.

Role of entropy. Existing CSP/COP benchmarks often reward finding a single high-quality solution, rather than producing a diverse set of samples. This raises a natural question about the role of the entropy term in our training objective (Equation (7)), since this term encourages sample diversity. As Sanokowski et al. [56] note, annealing the Boltzmann temperature toward zero shifts the objective toward energy minimization, reducing the relative importance of entropy in the low-temperature limit. To test whether entropy remains useful in this optimization-focused setting, we train BloGDiT with the entropy term removed. We find that removing entropy generally degrades performance relative to

the full objective (Table 3). This suggests that, although entropy is not part of the energy objective used to evaluate solution quality, it plays an important regularizing role during diffusion training by maintaining exploration and preventing premature collapse to suboptimal regions.

Sampling steps. We ablate the number of sampling steps used at test time while keeping training and all other hyperparameters fixed. We run BloGDiT for $T \in \{25, 50, 100, 200, 500, 1000, 2000\}$ steps and report the resulting solution accuracy on Sudoku. Figure 4 shows that performance improves rapidly in early iterations and exhibits diminishing returns beyond a few hundred steps, i.e., most of the quality gains are achieved with relatively short sampling trajectories, suggesting a trade-off between computation cost and solution quality. We further examine this effect in Appendix N.1

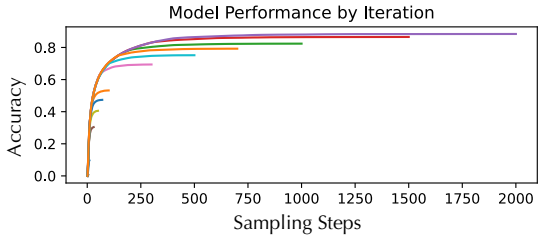

Figure 4: Sudoku test-time accuracy as a function of the number of sampling steps T. Each curve corresponds to a different number of sampling steps with the same model.

6 Related work

Learning-based approaches for discrete optimization. A large body of work studies learning heuristics for constraint problems, including GNN-based methods [69, 70] as well as Transformer-based models for constraint satisfaction [79, 82]. More recently, diffusion models have emerged as an alternative paradigm for combinatorial optimization, with prominent lines focusing on graph-structured problems [67] and unsupervised diffusion frameworks that optimize solutions via energy guidance without requiring supervised solution datasets [56, 57, 86]. In contrast, our goal is to move beyond specifically graph-based backbones by using a Diffusion Transformer.

Gibbs-style sampling in diffusion. A growing body of work investigates diffusion processes that iteratively edit subsets of variables. SDEdit [43] demonstrates the effectiveness of re-noising and re-denoising for editing in continuous domains, while remasking methods develop principled inference-time strategies for discrete diffusion models [72]. Closely related to our approach, Interleaved Gibbs Diffusion [5] implements generation as Gibbs-style conditional updates, and Block Diffusion [6] studies blockwise interpolation between autoregressive and diffusion generation in language modeling. We provide a more detailed discussion of related work in Appendix H.

7 Conclusion

We introduced BloGDiT, a Blocked Gibbs Diffusion Transformer for unsupervised constraint satisfaction and optimization via Boltzmann sampling. Our key insight is that near-feasible solutions typically require sparse edits to a subset of variables, which is poorly matched to standard diffusion updates that denoise all variables at every step. BloGDiT addresses this mismatch by performing block-wise resampling moves with an annealed block-size schedule, enabling global exploration early and localized refinement late. Across four benchmarks, BloGDiT matches or improves over strong neural baselines, suggesting that Gibbs-style conditional updates provide a useful inductive bias for Transformer-based constraint reasoning.

Limitations and future work. An important next step is to incorporate scalability techniques such as those of Sanokowski et al. [57] to reduce training and sampling cost. We operated in the continuous logit space in this work; further investigation of discrete diffusion methods that act directly on the discrete domain, as in Sanokowski et al. [56], would be insightful. In addition, while our approach is motivated by minimizing the reverse KL to a Boltzmann target and produces high-quality approximate samples in practice, it does not provide exact sampling guarantees. Future work could follow recent trends in diffusion [16, 61] and add an MCMC-style correction (e.g., a Metropolis–Hastings accept–reject step [25]) to more faithfully target the true Boltzmann distribution.

Acknowledgments

This work was supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korean government(MSIT) (No. RS-2024-00457882, National AI Research Lab Project).

References

[1] Utku Umur Acikalin, Aaron M Ferber, and Carla P Gomes. Learning to explore and exploit with GNNs for unsupervised combinatorial optimization. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=vaJ4FObpXN.

[2] Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. In *International conference on machine learning*, pages 134–144. PMLR, 2020.

[3] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=i80OPhOCVH2.

[4] Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Pdp: A general neural framework for learning constraint satisfaction solvers. *arXiv preprint arXiv:1903.01969*, 2019.

[5] Gautham Govind Anil, Sachin Yadav, Dheeraj Mysore Nagaraj, Karthikeyan Shanmugam, and Prateek Jain. Interleaved gibbs diffusion for constrained generation. In *ICLR 2025 Workshop on Deep Generative Model in Machine Learning: Theory, Principle and Efficacy*, 2025.

[6] Marianne Arriola, Subham Sekhar Sahoo, Aaron Gokaslan, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Justin T Chiu, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tyEyYT267x.

[7] 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.

[8] Thomas Barrett, William Clements, Jakob Foerster, and Alex Lvovsky. Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3243–3250, 2020.

[9] Thomas D Barrett, Christopher WF Parsonson, and Alexandre Laterre. Learning to solve combinatorial graph partitioning problems via efficient exploration. *arXiv preprint arXiv:2205.14105*, 2022.

[10] Nicolas Beldiceanu, Mats Carlsson, and Jean-Xavier Rampon. Global Constraint Catalog, 2005. URL https://hal.science/hal-00485396. Research Report SICS T2005-08.

[11] Maximilian Böther, Otto Kißig, Martin Taraz, Sarel Cohen, Karen Seidel, and Tobias Friedrich. What's wrong with deep learning in tree search for combinatorial optimization. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=mk0HzdqY7i1.

[12] Quentin Cappart, Tias Guns, Michele Lombardi, Gilles Pesant, and Dimos Tsouros. Combining constraint programming and machine learning: from current progress to future opportunities. *Journal of Artificial Intelligence Research*, 84, 2025.

[13] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020.

[14] Ting Chen, Ruixiang ZHANG, and Geoffrey Hinton. Analog bits: Generating discrete data using diffusion models with self-conditioning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=3itjR9QxFw.

[15] Arnaud Delage-Reid, Gilles Pesant, and Amal Zouaq. Neurosymbolic large neighbourhood search. In *32nd International Conference on Principles and Practice of Constraint Programming (CP 2026)*, pages 18–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2026.

[16] Yilun Du, Conor Durkan, Robin Strudel, Joshua B Tenenbaum, Sander Dieleman, Rob Fergus, Jascha Sohl-Dickstein, Arnaud Doucet, and Will Sussman Grathwohl. Reduce, reuse, recycle: Compositional generation with energy-based diffusion models and mcmc. In *International conference on machine learning*, pages 8489–8510. PMLR, 2023.

[17] Yilun Du, Jiayuan Mao, and Joshua B. Tenenbaum. Learning iterative reasoning through energy diffusion. In *International Conference on Machine Learning (ICML)*, 2024.

[18] Jonas K Falkner, Daniela Thyssens, and Lars Schmidt-Thieme. Large neighborhood search based on neural construction heuristics. *arXiv preprint arXiv:2205.00772*, 2022.

[19] Shengyu Feng and Yiming Yang. Regularized langevin dynamics for combinatorial optimization. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=bbJ0QCujU4.

[20] Shengyu Feng, Zhiqing Sun, and Yiming Yang. Spl-lns: Sampling-enhanced large neighborhood search for solving integer linear programs. *arXiv preprint arXiv:2508.16171*, 2025.

[21] Alan E Gelfand and Adrian FM Smith. Sampling-based approaches to calculating marginal densities. *Journal of the American statistical association*, 85(410):398–409, 1990.

[22] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6): 721–741, 1984.

[23] Fred Glover, Gary Kochenberger, and Yu Du. A tutorial on formulating and using qubo models. *arXiv preprint arXiv:1811.11538*, 2018.

[24] Aric A Hagberg, Daniel A Schult, and Pieter J Swart. Exploring network structure, dynamics, and function using networkx. In *Proceedings of the Python in Science Conference*, pages 11–15. SciPy, 2008.

[25] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97, 1970.

[26] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[27] André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. In *ECAI 2020*, pages 443–450. IOS Press, 2020.

[28] André Hottung, Paula Wong-Chung, and Kevin Tierney. Neural deconstruction search for vehicle routing problems. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=bCmEP1Ltwq.

[29] André Hottung and Kevin Tierney. Neural large neighborhood search for routing problems. *Artificial Intelligence*, 313:103786, 2022. ISSN 0004-3702.

[30] Taoan Huang, Aaron M Ferber, Yuandong Tian, Bistra Dilkina, and Benoit Steiner. Searching large neighborhoods for integer linear programs with contrastive learning. In *International conference on machine learning*, pages 13869–13890. PMLR, 2023.

[31] Claus S Jensen, Uffe Kjærulff, and Augustine Kong. Blocking gibbs sampling in very large probabilistic expert systems. *International Journal of Human-Computer Studies*, 42(6):647–666, 1995.

[32] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[33] Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. *Advances in Neural Information Processing Systems*, 33:6659–6672, 2020.

[34] Nikolaos Karalias, Akbar Rafiey, Yifei Xu, Zhishang Luo, Behrooz Tahmasebi, Connie Jiang, and Stefanie Jegelka. Geometric algorithms for neural combinatorial optimization with constraints. *Advances in Neural Information Processing Systems*, 38:78585–78633, 2025.

[35] Jaeyeon Kim, Kulin Shah, Vasilis Kontonis, Sham M. Kakade, and Sitan Chen. Train for the worst, plan for the best: Understanding token ordering in masked diffusions. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=DjJmre5IkP.

[36] Wenhao Li, Yudong Xu, Scott Sanner, and Elias Boutros Khalil. Tackling the abstraction and reasoning corpus with vision transformers: the importance of 2d representation, positions, and objects. *Transactions on Machine Learning Research*, 2025.

[37] Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. From distribution learning in training to gradient search in testing for combinatorial optimization. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=JtFOugNMv2.

[38] 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.

[39] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. *Advances in neural information processing systems*, 31, 2018.

[40] Yiqiao Liao, Farinaz Koushanfar, and Parinaz Naghizadeh. Test-time adaptation for unsupervised combinatorial optimization. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=VVyGfRp4fG.

[41] Bjørnar Luteberget and Giorgio Sartor. Feasibility jump: an lp-free lagrangian mip heuristic. *Mathematical Programming Computation*, 15(2):365–388, 2023.

[42] Yoshiki Matsuda. Benchmarking the max-cut problem on the simulated bifurcation machine. *Benchmarking the MAX-CUT problem on the Simulated Bifurcation Machine*, 2019.

[43] Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. SDEdit: Guided image synthesis and editing with stochastic differential equations. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=aBsCjcPu_tE.

[44] Sascha Mücke. A simple qubo formulation of sudoku. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1958–1962, 2024.

[45] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1ldO2EFPr.

[46] Emils Ozolins, Karlis Freivalds, Andis Draguns, Eliza Gaile, Ronalds Zakovskis, and Sergejs Kozlovics. Goal-aware neural sat solver. In *2022 International joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2022.

[47] Rasmus Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks. *Advances in neural information processing systems*, 31, 2018.

[48] Seonho Park and Pascal Van Hentenryck. Self-supervised primal-dual learning for constrained optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 4052–4060, 2023.

[49] 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.

[50] Laurent Perron and Frédéric Didier. CP-SAT. URL `https://developers.google.com/optimization/cp/cp_solver/`.

[51] David Pisinger and Stefan Ropke. Large neighborhood search. In *Handbook of metaheuristics*, pages 99–127. Springer, 2018.

[52] Akbar Rafiey, Yifei Xu, and Nikolaos Karalias. Self-supervised combinatorial optimization with constraints via frank-wolfe. *arXiv preprint arXiv:2609.25728*, 2026.

[53] Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In *AAAI*, volume 94, pages 362–367, 1994.

[54] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[55] Subham Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

[56] Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization. In *Forty-first International Conference on Machine Learning*, 2024. URL `https://openreview.net/forum?id=AFfXlKFHXJ`.

[57] Sebastian Sanokowski, Wilhelm Franz Berghammer, Haoyu Peter Wang, Martin Ennemoser, Sepp Hochreiter, and Sebastian Lehner. Scalable discrete diffusion samplers: Combinatorial optimization and statistical physics. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=peNgxpbdxB`.

[58] Martin JA Schuetz, J Kyle Brubaker, and Helmut G Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377, 2022.

[59] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pages 417–431. Springer, 1998.

[60] Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37:103131–103167, 2024.

[61] Anders Sjöberg, Jakob Lindqvist, Magnus Önnheim, Mats Jirstrand, and Lennart Svensson. Mcmc-correction of score-based diffusion models for model composition. *arXiv preprint arXiv:2307.14012*, 2023.

[62] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. pmlr, 2015.

[63] Jialin Song, Yisong Yue, Bistra Dilkina, et al. A general large neighborhood search framework for solving integer linear programs. *Advances in Neural Information Processing Systems*, 33: 20012–20023, 2020.

[64] Nicolas Sonnerat, Pengming Wang, Ira Ktena, Sergey Bartunov, and Vinod Nair. Learning a large neighborhood search algorithm for mixed integer programs. *arXiv preprint arXiv:2107.10201*, 2021.

[65] Peter J Stuckey, Thibaut Feydy, Andreas Schutt, Guido Tack, and Julien Fischer. The minizinc challenge 2008–2013. *AI magazine*, 35(2):55–60, 2014.

[66] Haoran Sun, Katayoon Goshvadi, Azade Nova, Dale Schuurmans, and Hanjun Dai. Revisiting sampling for combinatorial optimization. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 32859–32874. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/sun23c.html.

[67] Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Advances in neural information processing systems*, 36:3706–3731, 2023.

[68] Bo Tang, Elias Boutros Khalil, and Jan Drgona. Learning to optimize for mixed-integer nonlinear programming with feasibility guarantees. In *Workshop on Differentiable Learning of Combinatorial Algorithms*, 2025. URL https://openreview.net/forum?id=o8nHQRJU92.

[69] Jan Toenshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Graph neural networks for maximum constraint satisfaction. *Frontiers in artificial intelligence*, 3:580607, 2021.

[70] Jan Tönshoff, Berke Kisin, Jakob Lindner, and Martin Grohe. One model, any csp: graph neural networks as fast global search heuristics for constraint satisfaction. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, IJCAI '23, 2023. ISBN 978-1-956792-03-4. doi: 10.24963/ijcai.2023/476. URL https://doi.org/10.24963/ijcai.2023/476.

[71] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[72] Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=IJryQAOy0p.

[73] Haoyu Peter Wang, Nan Wu, Hang Yang, Cong Hao, and Pan Li. Unsupervised learning for combinatorial optimization with principled objective relaxation. *Advances in Neural Information Processing Systems*, 35:31444–31458, 2022.

[74] Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, pages 6545–6554. PMLR, 2019.

[75] Yan Wang, Ling Guo, Hao Wu, and Tao Zhou. Energy-based diffusion generator for efficient sampling of boltzmann distributions. *Neural Networks*, page 108126, 2025.

[76] Laurence A Wolsey. *Integer programming*. John Wiley & Sons, 2020.

[77] Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Learning large neighborhood search policy for integer programming. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=IaM7U4J-w3c.

[78] Ke Xu, Frederic Boussemart, Fred Hemery, and Christophe Lecoutre. A simple model to generate hard satisfiable instances. In *Proceedings of the 19th international joint conference on Artificial intelligence*, pages 337–342, 2005.

[79] Yudong Xu, Wenhao Li, Scott Sanner, and Elias Boutros Khalil. Self-supervised transformers as iterative solution improvers for constraint satisfaction. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 69432–69450. PMLR, 13–19 Jul 2025.

[80] Yudong Xu, Wenhao Li, Scott Sanner, and Elias Boutros Khalil. Large neighborhood search meets iterative neural constraint heuristics. In *The 23rd International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 2026.

[81] Zhun Yang, Joohyung Lee, and Chiyoun Park. Injecting logical constraints into neural networks via straight-through estimators. In *International Conference on Machine Learning*, pages 25096–25122. PMLR, 2022.

[82] Zhun Yang, Adam Ishay, and Joohyung Lee. Learning to solve constraint satisfaction problems with recurrent transformer. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=udNhDCr2KQe.

[83] Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Beyond autoregression: Discrete diffusion for complex reasoning and planning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=NRYgUzSPZz.

[84] Yinyu Ye. The gset dataset https://web.stanford.edu/~yyye/yyye/Gset/, 2003.

[85] Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron C Courville, Yoshua Bengio, and Ling Pan. Let the flows tell: Solving graph combinatorial problems with gflownets. *Advances in neural information processing systems*, 36:11952–11969, 2023.

[86] Hang Zhao, Kexiong Yu, Yuhang Huang, Renjiao Yi, Chenyang Zhu, and Kai Xu. Disco: Efficient diffusion solver for large-scale combinatorial optimization problems. *arXiv preprint arXiv:2406.19705*, 2024.

[87] Jianan Zhou, Yaoxin Wu, Zhiguang Cao, Wen Song, Jie Zhang, and Zhenghua Chen. Learning large neighborhood search for vehicle routing in airport ground handling. *IEEE Transactions on knowledge and data engineering*, 35(9):9769–9782, 2023.

A Loss derivation

To derive the objective detailed in Equation (7), we first expand the reverse KL divergence over the joint distribution:

$$\mathrm{KL}\left(q_\theta(X_{0:T}, m_{1:T}) \| p(X_{0:T}, m_{1:T})\right). \tag{9}$$

$$= \mathbb{E}_{q_\theta}\left[\log q(X_T) - \log p_B(X_0)\right] + \sum_{t=1}^{T} \mathbb{E}_{q_\theta}\left[\log q_\theta(X_{t-1} \mid X_t, m_t) - \log p(X_t \mid X_{t-1}, m_t)\right] + \sum_{t=1}^{T} \mathbb{E}_{q_\theta}\left[\log q_{\theta,t}(m_t \mid X_t) - \log p_t(m_t \mid X_{t-1})\right] \tag{10}$$

$$= \underbrace{-\log p_B(x_0)}_{\text{(1) Energy}} + \underbrace{\mathbb{E}_{X_{0:T}, m_{1:T} \sim q_\theta}\left[\sum_{t=1}^{T} \log q_\theta(X_{t-1} \mid X_t, m_t)\right]}_{\text{(2) Reverse entropy / reconstruction}} \tag{11}$$

$$\underbrace{- \mathbb{E}_{X_{0:T}, m_{1:T} \sim q_\theta}\left[\sum_{t=1}^{T} \log p(X_t \mid X_{t-1}, m_t)\right]}_{\text{(3) Forward noise matching}} \tag{12}$$

$$+ \underbrace{\mathbb{E}_{X_{0:T}, m_{1:T} \sim q_\theta}\left[\sum_{t=1}^{T} \log \frac{q_{\theta,t}(m_t \mid X_t)}{p_t(m_t \mid X_{t-1})}\right]}_{\text{(4) Mask divergence}} + C, \tag{13}$$

where C absorbs $\mathbb{E}_{q_\theta}[\log q(X_T)]$, constant w.r.t. θ. We now break down each term further:

Term (1): Energy Using $\log p_B(X_0) = -\beta H(X_0) - \log Z(\beta)$, we can isolate the energy term:

$$-\mathbb{E}_{q_\theta}[\log p_B(X_0)] = \beta\, \mathbb{E}_{q_\theta}[H(X_0)] + \log Z(\beta). \tag{14}$$

where $\log Z(\beta)$ can be absorbed into C as it is constant w.r.t. θ, and $\beta = \frac{1}{\tau}$ can be removed by multiplying τ on both sides of the equation.

Term (2): Reverse entropy / reconstruction.

$$\mathbb{E}_{q_\theta}\left[\sum_{t=1}^{T} \log q_\theta(X_{t-1} \mid X_t, m_t)\right] \tag{15}$$

$$= \sum_{t=1}^{T} \mathbb{E}_{X_{T:t-1}, m_{T:t} \sim q_\theta}\left[\log q_\theta(X_{t-1} \mid X_t, m_t)\right] \tag{16}$$

$$= \sum_{t=1}^{T} \mathbb{E}_{X_{T:t}, m_{T:t} \sim q_\theta}\left[\mathbb{E}_{X_{t-1} \mid X_t, m_t} \log q_\theta(X_{t-1} \mid X_t, m_t)\right] \tag{17}$$

$$= -\sum_{t=1}^{T} \mathbb{E}_{X_{T:t}, m_{T:t} \sim q_\theta}\left[S(q_\theta(X_{t-1} \mid X_t, m_t))\right]. \tag{18}$$

Where $S(q_\theta(X_{t-1} \mid X_t, m_t))$ is the entropy over $q_\theta(X_{t-1} \mid X_t, m_t)$.

Term (3): Forward noise matching.

$$\mathbb{E}_{q_\theta}\left[\sum_{t=1}^{T}\log p(X_t \mid X_{t-1}, m_t)\right] \tag{19}$$

$$=\sum_{t=1}^{T}\mathbb{E}_{X_{T:t-1},m_{T:t}\sim q_\theta}\left[\log p(X_t \mid X_{t-1}, m_t)\right]. \tag{20}$$

$$=\sum_{t=1}^{T}\mathbb{E}_{X_{T:t},m_{T:t}\sim q_\theta}\left[\mathbb{E}_{X_{t-1}|X_t,m_t}\log p(X_t \mid X_{t-1}, m_t)\right]. \tag{21}$$

B Invariant marginal distribution

To validate our approach, we must prove that the marginal distribution of the trajectory $X_{0:T}$ under this augmented system is identical to the target diffusion process $q_\theta(X_{0:T})$ defined in Eq. (2). Recall our proposition from Section 3.1:

Proposition 3.1 (Sampling equivalence under mask augmentation). *Let $(X_{0:T}, m_{1:T})$ be sampled from the augmented joint distribution* (6)*. By discarding $m_{1:T}$, the resulting trajectory $X_{0:T}$ is distributed as the original process $q_\theta(X_{0:T})$ induced by the mask-augmented transitions* (3)*: $X_{0:T} \sim q_\theta(X_{0:T}) \iff \sum_{m_{1:T}} q_\theta(X_{0:T}, m_{1:T}) = q_\theta(X_{0:T})$.*

Proof. We recover the marginal density of $X_{0:T}$ by summing the augmented joint distribution over all possible realizations of the auxiliary mask sequence $\mathbf{m} = \{m_1, \ldots, m_T\}$:

$$\sum_{\mathbf{m}} q_\theta(X_{0:T}, \mathbf{m}) \tag{22}$$

$$= \sum_{m_1,\ldots,m_T}\left[q(X_T)\prod_{t=1}^{T} q_{\theta,t}(m_t \mid X_t) q_\theta(X_{t-1} \mid X_t, m_t)\right]. \tag{23}$$

Because the mask m_t at step t is conditionally independent of masks at other steps given X_t, the summation over the sequence factorizes across time steps. We can interchange the summation and the product:

$$= q(X_T)\prod_{t=1}^{T}\left[\sum_{m_t} q_{\theta,t}(m_t \mid X_t) q_\theta(X_{t-1} \mid X_t, m_t)\right]. \tag{24}$$

The term inside the brackets corresponds exactly to the definition of the mask-augmented transition kernel in Eq. (3). Substituting this back yields:

$$= q(X_T)\prod_{t=1}^{T} q_\theta(X_{t-1} \mid X_t) \tag{25}$$

$$\equiv q_\theta(X_{0:T}). \tag{26}$$

Therefore, discarding the auxiliary variables $m_{1:T}$ yields a sequence distributed exactly according to the original target $q_\theta(X_{0:T})$. □

C Mask annealing schedule

We anneal the masking rate $\rho_t \in [\rho_{\min}, \rho_{\max}]$ over steps $t \in \{0, \ldots, T\}$. We experiment with two schedules:

Linear:

$$\rho_t = \rho_{\min} + (\rho_{\max} - \rho_{\min})\frac{t}{T}.$$

Geometric:

$$\rho_t = \rho_{\min} \left(\frac{\rho_{\max}}{\rho_{\min}} \right)^{t/T}.$$

D Adaptive Mask Selection

We replace the state-agnostic masking distribution in Equation (8) with a *state-dependent* strategy at inference time. Concretely, instead of masking each variable with a common rate ρ_t, we use per-variable rates $\rho_t^{(i)}$ that depend on the current model output. We normalize the rates to preserve the expected mask budget: $\frac{1}{d}\sum_{i=1}^{d} \rho_t^{(i)} \approx \rho_t$. We then sample masks independently,

$$\pi_t(m_t) = \prod_{i=1}^{d} \text{Bernoulli}(m_t^{(i)}; \rho_t^{(i)}). \tag{27}$$

Top-probability margin. Following Kim et al. [35], we estimate the certainty of a variable using the margin between its two most likely values. Let $Z_t^{(i)}$ denote the model's output logits for variable i at reverse step t. We first define $q_\theta(x^{(i)} = j \mid X_t) = \text{softmax}(Z_t^{(i)})_j$. Then, let $j_1^{(i)}$ and $j_2^{(i)}$ be the most- and second-most likely values under $q_\theta(x^{(i)} \mid X_t)$. The top-probability margin certainty score is

$$c_t^{(i)} = \left| q_\theta(x^{(i)} = j_1^{(i)} \mid X_t) - q_\theta(x^{(i)} = j_2^{(i)} \mid X_t) \right|. \tag{28}$$

A small value of $c_t^{(i)}$ indicates that the model is uncertain between two competing assignments for variable i. We then bias mask sampling by replacing the state-agnostic rate ρ_t with a state-dependent masking rate $\rho_t^{(i)}(X_t)$ (dropping the explicit X_t for readability). Concretely, we set

$$\rho_t^{(i)} \propto -c_t^{(i)}$$

This encourages repeatedly revisiting variables where the model has low confidence and may benefit from additional refinement.

Most-critical variables. Motivated by Large Neighborhood Search [51], we also consider a mask-selection strategy that focuses updates on variables currently involved in constraint violations. At each step, we compute a violation score v_i for every variable based on the current assignment by counting how much that variable contributes to the violated constraints. For example, in graph coloring, a vertex receives a higher score if many of its incident edges have both endpoints assigned the same color.

We then sample the mask with probabilities proportional to these violation scores, normalized to preserve the expected mask budget:

$$\rho_{t,i} \propto v_i(X_t)$$

Thus, variables that contribute more to the current constraint violation are more likely to be masked and resampled.

Related variables. We also consider another simple strategy inspired by Large Neighborhood Search [59]. Rather than selecting variables independently, this strategy selects groups of related variables that participate in common constraints.

Let $\mathcal{C} = \{c_1, \ldots, c_M\}$ denote the set of constraints, and let $\text{vars}(c_k) \subseteq [d]$ be the variables appearing in constraint c_k. At reverse step t, we sample a Bernoulli mask over constraints,

$$m_{t,k}^{\text{constr}} \sim \text{Bernoulli}(\eta_t), \qquad k \in [M],$$

and define the variable destroy set as the union of variables appearing in the selected constraints:

$$S_t = \bigcup_{k: m_{t,k}^{\text{constr}} = 1} \text{vars}(c_k). \quad m_t^{(i)} = \mathbb{I}\{i \in S_t\}.$$

We choose the constraint-level sampling rate η_t so that the resulting expected variable-level destruction rate is approximately ρ_t.

E Transformer architecture

We adopt the same Transformer architecture as Xu et al. [79]. This section summarizes the input representation and attention-based update rule used to model CSP assignments.

E.1 Input representation

We represent a CSP assignment as a set of n tokens, one per variable. Throughout, we focus on tasks where all variables share the same discrete domain D with $|D| = K$.

Value embeddings. Given a current assignment $\{x_i = v_i\}_{i=1}^n$, each variable token receives a learnable embedding $\mathbf{e}(v_i)$ for its current value $v_i \in D$.

Absolute positional encoding (APE). To encode variable identity, we add an absolute positional encoding $\mathrm{APE}(x_i)$. For variables with multi-dimensional indices $(i_1, \ldots, i_k)$ (e.g., Sudoku), we concatenate positional encodings per dimension inspired by vision Transformers[13, 36]:

$$\mathrm{APE}(x_{i_1,\ldots,i_k}) = \mathrm{Concat}(\mathrm{PE}(i_1), \ldots, \mathrm{PE}(i_k)). \tag{29}$$

Constraint graph as relative positional encoding (RPE). We construct a binary constraint graph $G = (V, E)$ where vertices correspond to variables and $(i, j) \in E$ if variables i and j co-occur in any constraint. We incorporate this structure via an additive bias on attention logits:

$$\mathrm{RPE}(i, j) = c \cdot \mathbb{I}[(i, j) \notin E], \qquad c \le 0, \tag{30}$$

which reduces (or masks) attention between unrelated variables. We use either a fixed c or a learned scalar.

E.2 Transformer update rule

Self-attention with constraint bias. At layer ℓ, a standard multi-head self-attention block computes

$$\mathbf{A}_{ij} = \frac{(\mathbf{W}^Q \mathbf{h}_i^{(\ell)})^\top (\mathbf{W}^K \mathbf{h}_j^{(\ell)})}{\sqrt{d}} + \mathrm{RPE}(i, j), \qquad \alpha_{ij} = \mathrm{softmax}_j(\mathbf{A}_{ij}), \qquad \mathbf{z}_i = \sum_j \alpha_{ij} \mathbf{W}^V \mathbf{h}_j^{(\ell)}.$$

We then apply a position-wise feedforward network with standard residual connections and normalization (omitted for brevity).

Output. For each variable $i \in S$, the final-layer representation $\mathbf{h}_i^{(L)}$ is mapped to the parameters of a diagonal Gaussian distribution:

$$\boldsymbol{\mu}_i = \mathbf{W}_\mu \mathbf{h}_i^{(L)} + \mathbf{b}_\mu, \qquad \log \boldsymbol{\sigma}_i^2 = \mathbf{W}_{\log \sigma^2} \mathbf{h}_i^{(L)} + \mathbf{b}_{\log \sigma^2},$$

where $\boldsymbol{\mu}_i, \log \boldsymbol{\sigma}_i^2 \in \mathbb{R}^K$. We then sample

$$\mathbf{z}_i = \boldsymbol{\mu}_i + \boldsymbol{\sigma}_i \odot \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \qquad \boldsymbol{\sigma}_i = \exp\left(\tfrac{1}{2} \log \boldsymbol{\sigma}_i^2\right),$$

using the reparameterization trick. The sampled latent $\mathbf{z}_i$ is decoded into the current solution, and used for the next diffusion step.

E.3 GCN backbone ablation

We construct a GCN variant of BloGDiT by replacing multi-head attention with graph convolution. Both backbones use the constraint graph $G = (V, E)$ defined above, including self-loops. For Sudoku, neighboring variables are cells sharing a row, column, or 3×3 sub-grid; for Graph Coloring, MIS, and MaxCut, we use the corresponding problem graph. Omitting the shared normalization and dropout for brevity, the GCN aggregation at layer ℓ is

$$\mathbf{z}_i = \mathbf{W}_{\mathrm{gcn}}^{(\ell)} \sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{\mathbf{h}_j^{(\ell)}}{\sqrt{d_i d_j}} + \mathbf{b}_{\mathrm{gcn}}^{(\ell)},$$

where $\mathcal{N}(i)$ denotes the neighbors of variable i and $d_i = |\mathcal{N}(i) \cup \{i\}|$. The degree normalization controls the contribution of nodes with different numbers of neighbors. The remaining block structure and Gaussian output heads follow the Transformer architecture. Absolute positional encodings are disabled for the GCN.

Model size. We increase the GCN hidden dimension to 276 while keeping the number of layers fixed for each benchmark. The total trainable parameter counts differ by less than 0.3% (Table 5). Both variants are trained from scratch using the same training instances, diffusion objective, masking schedule, optimizer, and training budget, and are evaluated under the same inference budget.

Table 5: Total trainable parameter counts for the Transformer and GCN variants of BloGDiT.

Benchmark	Transformer	GCN
Sudoku	1,623,703	1,621,247
Coloring-5	927,887	926,547
Coloring-10	1,624,089	1,622,077
MIS	926,729	924,057
MaxCut	926,729	924,057

F Constraint programming formulations

This section summarizes the CP formulations used to define energies for each benchmark.

Sudoku. Let $X_{i,j} \in \{1, \dots, 9\}$ denote the value in cell (i, j) for $i, j \in \{1, \dots, 9\}$. Sudoku constraints enforce uniqueness in every row, column, and 3×3 subgrid:

$$\textsc{AllDifferent}(X_{i,1}, \dots, X_{i,9}) \ \forall i, \qquad \textsc{AllDifferent}(X_{1,j}, \dots, X_{9,j}) \ \forall j,$$

and for each subgrid indexed by $(r, c) \in \{0, 1, 2\}^2$,

$$\textsc{AllDifferent}\big(\{X_{3r+a,3c+b}\}_{a,b\in\{1,2,3\}}\big).$$

Given clues are enforced by fixing the corresponding variables.

Graph Coloring. Given a graph $G = (V, E)$ and k colors, assign each vertex $v \in V$ a variable $X_v \in [k]$. The constraints require adjacent vertices to have different colors:

$$X_u \neq X_v \qquad \forall (u, v) \in E.$$

MaxCut. Given a graph $G = (V, E)$, introduce binary variables $X_v \in \{0, 1\}$ indicating the side of the cut. MaxCut can be viewed as a *2-coloring MaxCSP*: each edge (u, v) corresponds to an inequality constraint $X_u \neq X_v$, and the objective is to satisfy as many such constraints as possible:

$$\max_X \sum_{(u,v)\in E} \mathbb{I}[X_u \neq X_v],$$

which we write in minimization form for the energy:

$$\min_X \ - \sum_{(u,v)\in E} \mathbb{I}[X_u \neq X_v].$$

Maximum Independent Set. Given a graph $G = (V, E)$, introduce binary variables $X_v \in \{0, 1\}$ indicating whether vertex v is selected. The independent set constraints are

$$X_u + X_v \leq 1 \qquad \forall (u, v) \in E,$$

and the objective is to maximize set size:

$$\max_X \sum_{v\in V} X_v, \qquad \text{equivalently} \qquad \min_X \ - \sum_{v\in V} X_v.$$

G Training details

Our models were trained on single NVIDIA H100 GPU nodes. The typical training time for a model ranges from 3 to 6 hours (wall clock). For all models, we used AdamW as the optimizer, with learning

rate set to 0.0001, and train for 5000 epochs. For optimization tasks, we follow Sanokowski et al. [56] and set λ_j in Equation (1) to 1.01. We anneal τ in Equation (7) across training epochs linearly from 1 towards 0. Our approach requires unrolling the reverse diffusion steps for each gradient update, this restricts the number of diffusion steps we can back propagate through. The DiT baselines were trained by unrolling five reverse diffusion steps per gradient update. Because BloGDiT is time-agnostic (shared parameters across steps), we use a cheaper single-step estimator: at each gradient update, we sample a block size corresponding to a reverse timestep and apply one reverse transition. At inference time, the trained transition is applied iteratively according to the problem-specific sampling budget.

Table 6: Hyperparameters for the best performing models.

	Sudoku	Graph-coloring-5	Graph-coloring-10	MIS	MaxCut
Batch Size	512	128	128	128	128
Dropout	0.2	0.1	0.1	0.1	0.1
Layer Count	7	4	7	4	4
Head Count	3	3	3	3	3
Embedding Size	128	128	128	128	128
Mask Selection	margin	related	related	critical	margin
$\rho_{\max}$	0.9	0.9	0.9	0.9	0.9
$\rho_{\min}$	0.5	0.5	0.5	0.3	0.5

H Extended related work

Constraint Programming solvers. Classical constraint programming provides expressive modeling abstractions for finite-domain variables, global constraints, and propagation-based search [54]. Global constraints such as ALLDIFFERENT are especially important because they compactly encode structured combinatorial relations and admit specialized filtering algorithms [10, 53]. Modern solvers such as CP-SAT combine constraint propagation, SAT-style reasoning, integer programming techniques, and highly engineered heuristics [50]. These solvers are extremely effective, but their proposal and repair mechanisms are largely hand-designed and are applied anew to each instance.

Iterative methods for constraint optimization. Constraint optimization problems are often solved heuristically through iterative procedures that repeatedly propose, evaluate, and modify candidate assignments. A central classical example of iterative refinement is Large Neighborhood Search (LNS) [51, 59]. LNS maintains an incumbent solution and alternates between a destroy step and a repair step. The destroy operator removes or unassigns selected components of the current solution, and the repair operator reconstructs the missing components to produce a new complete assignment. Recent work has explored learning-based LNS heuristics, including learned destroy and repair operators [15, 18, 20, 27–30, 63, 64, 77, 87].

This view is closely aligned with our motivation: in many constraint problems, once an assignment is nearly feasible, progress often requires changing a selected subset of variables while preserving the rest. BloGDiT borrows this intuition by masking a block of variables and denoising that block conditional on the unmasked variables.

Another line of work implements discrete sampling for solving combinatorial optimization problems. Sun et al. [66] show that modern gradient-based discrete MCMC and parallel neighborhood exploration can produce competitive training-free heuristics for several CO problems. More recently, Feng and Yang [19] propose Regularized Langevin Dynamics, which modifies discrete Langevin dynamics and achieves state-of-the-art results on several binary problems.

A broad line of work learns iterative neural heuristics for discrete reasoning and optimization. Many graph-based methods refine assignments through message passing and have been applied to constraint reasoning tasks [47, 69, 70]. Yang et al. [82] and Xu et al. [79, 80] implement recurrent Transformers as iterative heuristics. X^2GNN [1] combines the exploration of multiple candidate solutions with stochastic GNN-based iterative refinement.

Unsupervised methods for constraint optimization. A common approach to learning optimization heuristics without labeled solutions is to construct training losses directly from the problem

objective and constraint violations. Such methods use continuous relaxations or gradient estimators to enable gradient-based learning, and have been developed for SAT and MaxCSP [4, 46, 69], graph optimization [33, 58, 73], and more general constraint satisfaction and constrained optimization problems [48, 68, 79–81]. Other approaches construct differentiable losses through geometric decompositions into feasible solutions [34, 52]. Test-time adaptation has also been explored to improve unsupervised neural optimization on individual problem instances [40]. Similarly, BloGDiT uses differentiable problem energies as part of a Boltzmann-based diffusion objective to learn iterative blocked resampling.

Diffusion and energy-based methods as iterative heuristics. Diffusion models define generative processes through gradual noising and learned denoising [26, 62]. Discrete diffusion extends this idea to categorical and structured domains [7], while continuous relaxations such as analog bits generate discrete variables by diffusing in a continuous representation space [14]. Diffusion Transformers replace specialized denoisers with Transformer backbones and have shown strong scaling behavior in generative modeling [49].

Recent work has adapted diffusion models to combinatorial optimization. DIFUSCO uses graph-based diffusion for graph optimization problems [67], and DISCO studies efficient diffusion heuristics for large-scale combinatorial optimization [86]. DiffUCO formulates unsupervised neural combinatorial optimization as approximate sampling from a Boltzmann distribution, thereby avoiding the need for supervised solution labels [56]; later work develops scalable discrete diffusion samplers for combinatorial optimization and statistical physics [57]. Energy-based diffusion generators have also been used [17, 75]. T2T [37] combines diffusion-based solution distribution learning with gradient-guided search at test time, iteratively perturbing and reconstructing candidate solutions. Fast T2T [38] introduces optimization consistency models to reduce the number of denoising steps while retaining objective-guided search. In contrast, BloGDiT performs Gibbs-style blocked conditional resampling, updating selected variables while preserving the rest of the assignment, learning the reverse transitions through an unsupervised Boltzmann objective.

Masked, blocked, and Gibbs-style generation. Classical Gibbs sampling updates variables, or blocks of variables, conditional on the rest of the current state [21, 22, 31]. This is natural for constraint solving because the unmodified part of a nearly feasible solution provides useful context for repairing the violated part.

Several recent generative models also use partial updates. SDEdit shows that re-noising and re-denoising can perform controlled edits in continuous domains [43]. Masked diffusion language models generate discrete sequences through iterative masking and denoising [55, 60], and remasking improves inference-time scaling by deciding which variables should be revised [72]. Work on token ordering in masked diffusion further shows that the choice of variables to reveal or update can strongly affect generation quality [35]. These ideas are conceptually aligned with BloGDiT's adaptive mask selection, where variables are prioritized according to model uncertainty or constraint violation.

Two recent methods are particularly close and are useful points of contrast. Interleaved Gibbs Diffusion performs Gibbs-style conditional updates for constrained generation [5], but updates one variable at a time and uses supervised training. BloGDiT instead updates a block at each iteration and trains without labels using a Boltzmann objective. Block Diffusion is also related, but it is a language modeling method whose blocks are fixed sequential segments of a token sequence; it runs a complete diffusion process within each block before moving to the next [6]. In contrast, BloGDiT samples a different block at each diffusion step, anneals the block size over time, and uses each block update as a learned repair move for constraint optimization.

I Baseline methods

Our baseline comparisons build on results reported in prior work on the same benchmarks, supplemented by additional evaluations. We include both classical optimization methods and learned heuristics to provide context for solution quality and computational cost. Below, we briefly describe the baselines and identify the sources of their reported results.

Sudoku. We compare with SATNet [74], a differentiable MAXSAT solver based on solving an SDP relaxation; RRN [47], a recurrent message-passing model for multi-step relational reasoning; the

Recurrent Transformer [82], which extends Transformers with recurrence for end-to-end CSP solving; iRED [17], which formulates reasoning as energy-based optimization and performs iterative inference via energy diffusion; RLSA [19], where we formulate Sudoku as a QUBO following Section I.1; and ConsFormer[79].

Graph Coloring. We compare with the greedy coloring algorithm implemented by NetworkX [24]; Feasibility jump, which is a local search heuristic [41]; ANYCSP [70], which proposes a universal GNN trained to act as an end-to-end heuristic for CSPs; RLSA [19], where we formulate graph coloring as a QUBO following Section I.2; and ConsFormer[79].

MaxCut. We compare with RUNCSP [69], an unsupervised GNN approach for maximum constraint satisfaction over binary CSPs; ECO-DQN [8], an RL method that improves solutions by exploratory local modifications during inference; ECORD [9], an RL approach that amortizes expensive GNN computation; X^2GNN [1], an unsupervised GNN-based optimization method that combines exploration of candidate solutions with stochastic iterative refinement; RLSA [19], ANYCSP [70], DiffUCO [56], and ConsFormer[79].

MIS. Following Sanokowski et al. [56], we compare with INTEL [39] and DGL [11], which use a Graph Convolutional Network to predict node membership likelihoods and then performs guided tree search; LwD [2], which learns a policy for deferring decisions to construct MIS in a variable number of stages; LTFT [85], which trains conditional GFlowNets to sequentially construct solutions and sample from an unnormalized objective over the solution space; DIFUSCO [67], a supervised graph-based diffusion solver that learns to generate combinatorial solutions from reference solutions; RLSA [19], X^2GNN [1], and DiffUCO [56].

Table 7: Sources of baseline results. "New" denotes numbers obtained by running the method under the evaluation protocol used in this paper. Otherwise, numbers are taken from the cited source.

Benchmark	Methods	Source
Sudoku	OR-Tools, SATNet, RRN, Recurrent Transformer, IRED, ConsFormer	Xu et al. [79]
	RLSA	New
Graph Coloring	OR-Tools, Greedy, Feasibility Jump, ANYCSP, ConsFormer	Xu et al. [79]
	RLSA	New
MIS	LwD, INTEL, DGL, LTFT, DiffUCO	Sanokowski et al. [56]
	DIFUSCO, RLSA	Feng and Yang [19]
	X^2GNN	Acikalin et al. [1]
	OR-Tools	New
MaxCut	RUNCSP, ECO-DQN, ECORD, DiffUCO	Sanokowski et al. [56]
	OR-Tools, ANYCSP, ConsFormer	Xu et al. [79]
	X^2GNN	Acikalin et al. [1]
	RLSA	New

I.1 Sudoku QUBO formulation

Following Mücke [44], we represent a Sudoku assignment with binary variables $X_{r,c,d} \in \{0, 1\}$, where $X_{r,c,d} = 1$ indicates that digit d is placed in cell (r, c). The QUBO energy is

$$E(X) = -\sum_{r,c,d} X_{r,c,d} + \lambda \sum_{\{(r,c,d),(r',c',d')\}\in\mathcal{C}} X_{r,c,d} X_{r',c',d'},$$

where $\mathcal{C}$ is the set of conflicting assignment pairs. Two assignments conflict if they place the same digit in the same row, column, or 3×3 box, or if they assign different digits to the same cell. We use $\lambda = 3$, as in Mücke [44]. Given clues are incorporated by clamping the corresponding variables: if cell (r, c) is given as digit d, then $X_{r,c,d} = 1$ and $X_{r,c,d'} = 0$ for all $d' \neq d$.

I.2 Graph Coloring QUBO formulation

Following Glover et al. [23], let $G = (V, E)$ be an undirected graph and let K be the number of available colors. We introduce binary variables $X_{v,k} \in \{0, 1\}$, where $X_{v,k} = 1$ indicates that vertex

v is assigned color k. The QUBO energy is

$$E(X) = \sum_{v \in V} \left(1 - \sum_{k=1}^{K} X_{v,k}\right)^2 + \sum_{(u,v) \in E} \sum_{k=1}^{K} X_{u,k} X_{v,k}.$$

The first term enforces that each vertex receives exactly one color, while the second penalizes adjacent vertices assigned the same color.

J Computational efficiency

No prior work evaluates all four benchmarks under a single unified protocol, and different benchmark families follow different reporting conventions. We therefore follow the standard evaluation convention for each benchmark and report the corresponding wall-clock budgets and sampling steps in Table 8. For Sudoku, we use a fixed budget of 2000 reverse diffusion steps following Xu et al. [79]. For Graph Coloring, all methods are evaluated under a strict 10-second wall-clock limit per instance, following Xu et al. [79]. For MaxCut, we use a 180-second wall-clock limit and report the best result over 20 independent runs, following Sanokowski et al. [56]. For MIS, comparison is less standardized across prior work, so we use a fixed budget of 1000 reverse diffusion steps and report the measured runtime, for reference, OR-Tools results averaged 0.39 seconds per instance for RB-small, and 18.43 seconds for RB-large.

Dataset	Runtime (Seconds)	Inference Steps
Sudoku-SATNet	0.15	2000
Sudoku-RRN	0.15	2000
Graph-Coloring-5 ($n = 50$)	10	5399
Graph-Coloring-5 ($n = 100$)	10	5439
Graph-Coloring-10 ($n = 100$)	10	3728
Graph-Coloring-10 ($n = 200$)	10	3931
MaxCut-800	180	84853
MaxCut-1K	180	75963
MaxCut-2K	180	23569
MaxCut-$\geq$3K	180	6840
MIS-RB-small	0.44	1000
MIS-RB-large	4.81	1000

Table 8: Computation cost at inference time. Sudoku and MIS use fixed sampling-step budgets, and the table reports the resulting average runtime per instance. Graph Coloring and MaxCut use fixed wall-clock budgets, and the table reports the average number of sampling steps completed within that budget.

J.1 MaxCut GSET computation protocols

All methods except X^2GNN use a 180-second budget and report the best of 20 runs following Sanokowski et al. [56]. For X^2GNN, using the values reported in Acikalin et al. [1], we compute the average absolute gaps to the same best-known reference values used for our results. X^2GNN uses different training distributions, inference budgets, and restart settings.

J.2 Sudoku and MIS runtime comparisons

We additionally report wall-clock runtimes for Sudoku and MIS. Table 9 compares the average solution quality and runtime per instance on the RB benchmarks. The runtimes of learned baselines are taken from the corresponding literature. As these methods may use different hardware and implementations, the reported runtimes are intended to provide additional context rather than a strictly equal hardware comparison. Table 10 compares BloGDiT with OR-Tools and RLSA on Sudoku. OR-Tools achieves the best performance in both accuracy and runtime. In contrast, BloGDiT substantially outperforms RLSA in accuracy while requiring less inference time.

Table 9: Runtime comparison on MIS. We report the average independent set size (↑) and wall-clock time per instance in seconds (↓).

	RB-small		RB-large	
Method	Size ↑	Time ↓	Size ↑	Time ↓
OR-Tools	20.10	0.39	42.66	18.43
RLSA	19.97	0.07	40.19	0.22
INTEL	18.47	1.57	34.47	2.43
DGL	17.36	1.53	34.50	2.87
LTFT	19.18	0.06	37.48	0.52
DIFUSCO	18.52	1.93	–	–
X^2GNN	20.07	1.94	39.28	9.43
DiffUCO	18.88	0.11	38.10	0.59
BloGDiT	19.52	0.44	37.05	4.81

Table 10: Runtime comparison on Sudoku. We report the percentage of solved instances (↑) and average wall-clock time per instance in seconds (↓).

	SATNet		RRN	
Method	Solved ↑	Time ↓	Solved ↑	Time ↓
OR-Tools	100.0	0.004	100.0	0.004
RLSA	4.2	0.38	1.8	0.38
BloGDiT	100.0	0.15	94.1	0.15

K Details for obtaining 2D projected trajectory visualization

To visualize the solving process, we project trajectories into a shared 2D subspace defined only over the *unfixed Sudoku cells*. For a puzzle with mask $m \in \{0,1\}^{81}$, where $m_i = 1$ indicates a given cell, each solution state is represented as a matrix $X \in [0,1]^{81\times 9}$ of per-cell categorical probabilities. We discard the fixed cells and flatten the remaining U unfixed rows into a vector in $\mathbb{R}^{9U}$. To provide context for the projection, we construct a static background cloud of N reduced states consisting of: (i) random assignments on the unfixed cells, and (ii) partially corrupted versions of the target solution, where a random fraction of unfixed cells is replaced by random digits. We then fit a single 2D dimensionality reduction map f on this static set using PCA, and project all trajectories with the same fitted map. Trajectory points are colored by accuracy, computed as the number of cells whose decoded digit matches the target solution. The resulting figure shows a hexbin/KDE background over the projected static cloud, together with the projected paths and the noise/target anchors. This visualization illustrates broad differences in trajectory behavior, but the 2D geometry and background density should not be interpreted as the true energy landscape.

L Additional MIS Baselines

We additionally compare BloGDiT with T2T, Fast T2T, and X^2GNN on the ER[700–800] MIS benchmark. These methods also improve solutions through iterative perturbation or reconstruction, but differ in their learning objectives and architectures.

Table 11 reports the average independent set size and runtime per instance. X^2GNN achieves the largest independent set size, while Fast T2T has the lowest runtime. BloGDiT outperforms T2T, Fast T2T, and OR-Tools in solution quality, but requires more inference time than T2T and Fast T2T. Compared with X^2GNN, BloGDiT achieves a smaller independent set size at a lower runtime. These results illustrate the trade-off between solution quality and computational cost across the different approaches.

Table 11: Additional MIS baseline comparison on ER[700–800]. We report the average independent set size (↑) and runtime per instance in seconds (↓).

Method	Size ↑	Time (s) ↓
T2T	41.37	13.92
Fast T2T	41.73	**2.74**
X^2GNN	**44.91**	57.18
OR-Tools	41.97	120.00
BloGDiT	42.20	32.90

M MaxCut per instance performance

The convention for MaxCut is to conduct 20 parallel runs and report the best cut found [9, 70, 79]. In Table 12, we additionally detail the mean and standard deviation of the 20 runs for each instance for ablation. We see that BloGDiT obtains a higher mean cut value than the −AMS and −Entropy variants on 19 and 21 of the 35 instances, respectively. The advantage is most notable on the largest instances: BloGDiT outperforms both ablations on all instances with at least 5,000 vertices. However, the improvements are not uniform across instances, and the complete model does not consistently reduce run-to-run variability.

Table 12: Per-instance MaxCut values reported as mean ± standard deviation across runs for each ablation. The highest mean among the three variants is shown in bold.

$\|V\|$	ID	Best Known	BloGDiT	−AMS	−Entropy
800	G1	11624	11609.88 ± 17.34	$\mathbf{11619.95 \pm 6.95}$	11613.15 ± 13.28
800	G14	3064	$\mathbf{3060.12 \pm 2.15}$	3057.65 ± 1.76	3057.45 ± 1.54
800	G15	3050	$\mathbf{3048.82 \pm 0.95}$	3046.40 ± 1.19	3046.10 ± 3.96
800	G16	3052	$\mathbf{3049.00 \pm 2.24}$	3044.60 ± 2.60	3044.45 ± 2.28
800	G17	3047	$\mathbf{3042.65 \pm 2.21}$	3039.60 ± 1.88	3040.15 ± 2.37
800	G2	11620	11606.82 ± 9.61	11609.20 ± 7.54	$\mathbf{11609.90 \pm 9.47}$
800	G3	11622	11614.47 ± 8.18	$\mathbf{11617.55 \pm 3.72}$	11616.95 ± 7.14
800	G4	11646	11630.65 ± 10.17	11640.80 ± 4.51	$\mathbf{11641.30 \pm 6.25}$
800	G5	11631	11621.71 ± 11.26	$\mathbf{11626.85 \pm 1.66}$	11622.20 ± 5.43
1000	G43	6660	6650.18 ± 10.38	$\mathbf{6657.90 \pm 2.88}$	6653.80 ± 6.99
1000	G44	6650	6644.12 ± 8.34	$\mathbf{6647.30 \pm 1.72}$	6646.50 ± 5.78
1000	G45	6654	6640.06 ± 6.56	$\mathbf{6648.85 \pm 6.34}$	6645.90 ± 4.47
1000	G46	6649	6638.06 ± 5.49	$\mathbf{6644.10 \pm 3.73}$	6641.85 ± 5.33
1000	G47	6657	6647.24 ± 5.61	$\mathbf{6653.15 \pm 4.25}$	6648.35 ± 6.78
1000	G51	3848	$\mathbf{3843.06 \pm 2.44}$	3838.20 ± 2.04	3839.75 ± 1.41
1000	G52	3851	$\mathbf{3845.65 \pm 2.80}$	3839.30 ± 2.13	3839.95 ± 2.26
1000	G53	3850	$\mathbf{3845.29 \pm 2.57}$	3839.55 ± 1.70	3841.15 ± 1.93
1000	G54	3852	$\mathbf{3844.76 \pm 3.23}$	3841.05 ± 2.50	3842.60 ± 3.27
2000	G22	13359	13318.59 ± 19.17	$\mathbf{13340.75 \pm 15.12}$	13333.10 ± 17.48
2000	G23	13344	13312.53 ± 13.64	13322.45 ± 6.46	$\mathbf{13325.50 \pm 8.82}$
2000	G24	13337	13313.82 ± 9.52	$\mathbf{13320.70 \pm 6.41}$	13316.50 ± 8.78
2000	G25	13340	13316.71 ± 8.32	$\mathbf{13317.05 \pm 5.65}$	13316.60 ± 9.08
2000	G26	13328	13303.47 ± 11.14	13307.00 ± 9.27	$\mathbf{13307.60 \pm 8.19}$
2000	G35	7687	$\mathbf{7667.76 \pm 4.74}$	7646.10 ± 4.20	7651.15 ± 5.90
2000	G36	7680	$\mathbf{7659.29 \pm 3.93}$	7638.35 ± 4.85	7647.20 ± 2.33
2000	G37	7691	$\mathbf{7670.59 \pm 4.93}$	7652.45 ± 5.58	7661.30 ± 7.53
2000	G38	7688	$\mathbf{7668.12 \pm 3.67}$	7649.25 ± 7.20	7651.60 ± 5.06
3000	G48	6000	$\mathbf{5969.18 \pm 49.46}$	5945.50 ± 56.26	5925.20 ± 46.97
3000	G49	6000	$\mathbf{5954.00 \pm 40.09}$	5942.50 ± 31.52	5932.50 ± 30.23
3000	G50	5880	5852.71 ± 24.83	$\mathbf{5853.60 \pm 24.22}$	5841.50 ± 17.48
5000	G55	10299	$\mathbf{10264.06 \pm 11.13}$	10258.55 ± 7.88	10256.05 ± 10.23
5000	G58	19293	$\mathbf{19223.18 \pm 5.50}$	19145.15 ± 12.88	19151.55 ± 12.22
7000	G60	14188	$\mathbf{14138.06 \pm 7.45}$	14081.05 ± 11.24	14118.55 ± 8.65
7000	G63	27045	$\mathbf{26936.06 \pm 12.48}$	26657.45 ± 20.50	26825.40 ± 12.12
10000	G70	9591	$\mathbf{9508.18 \pm 8.65}$	9467.95 ± 7.49	9491.35 ± 7.81

N Additional ablations

N.1 Inference budget sensitivity

We further evaluate BloGDiT using $0.5T$, T, and $2T$ inference steps, where T denotes the reference sampling budget for each benchmark (Appendix J). As shown in Table 13, Graph Coloring and MIS generally benefit from additional steps, although the improvements are modest. MaxCut exhibits a less consistent trend: increasing the number of steps sometimes degrades performance. One possible explanation is that extending the annealing schedule introduces additional large-block updates, which may disrupt high-quality assignments. These results suggest that increasing the inference budget does not necessarily improve solution quality.

Table 13: Effect of the number of inference steps. We evaluate using half ($0.5T$) and twice ($2T$) the reference number of steps (T). Higher is better for Graph Coloring (percentage solved) and MIS (average independent set size); lower is better for MaxCut (gap to best-known solution).

Dataset	$0.5T$	T	$2T$
Coloring-5 ($n = 50$)	82.58	83.00	83.00
Coloring-5 ($n = 100$)	52.33	54.67	54.92
Coloring-10 ($n = 100$)	53.92	53.92	53.92
Coloring-10 ($n = 200$)	17.92	18.58	18.83
MIS-RB-small	19.03	19.52	19.53
MIS-RB-large	36.49	37.05	37.51
MIS-ER	40.74	42.20	42.59
MaxCut-800	0.22	0.11	0.56
MaxCut-1K	0.44	1.33	0.56
MaxCut-2K	5.55	7.88	6.11
MaxCut-$\geq$3K	44.13	32.75	23.75

N.2 Adaptive mask-selection

We study the effect of the mask-selection strategy by comparing random selection with the three adaptive strategies described in Appendix D. Table 14 reports the results across all four problems. The choice of mask-selection strategy can have a substantial effect on performance. Confidence-based selection achieves the highest accuracy on Sudoku-Hard, while selecting variables involved in constraint violations performs best on MIS. On Graph Coloring, related-variable selection generally performs best, although the differences are relatively small. For MaxCut, both confidence-based and critical-variable selection achieve substantially smaller gaps than random masking. These results suggest that adaptive mask selection benefits from problem-specific information, while strategies that are poorly matched to the problem may perform worse than random selection.

Table 14: Sensitivity to the mask-selection strategy. We compare random selection with related-variable, most-critical-variable, and confidence-margin selection. Higher is better for Sudoku and Graph Coloring (% solved) and MIS (average independent set size); lower is better for MaxCut (gap to best-known solution).

Dataset	Random	Related	Critical	Confidence
Sudoku-Easy	100.00	100.00	100.00	100.00
Sudoku-Hard	88.21	80.53	78.27	94.12
Coloring-5 ($n = 50$)	82.92	83.00	82.33	82.25
Coloring-5 ($n = 100$)	54.58	54.67	50.33	51.42
Coloring-10 ($n = 100$)	53.92	53.92	53.92	53.92
Coloring-10 ($n = 200$)	17.42	18.58	16.92	16.83
MIS-RB-small (size $\uparrow$)	15.83	16.22	19.52	18.46
MIS-RB-large (size $\uparrow$)	17.16	20.99	37.05	34.08
MaxCut-800 (gap $\downarrow$)	2.11	11.11	2.22	0.11
MaxCut-1K (gap $\downarrow$)	3.22	11.56	1.22	1.33
MaxCut-2K (gap $\downarrow$)	18.89	38.89	7.11	7.88
MaxCut-$\geq$3K (gap $\downarrow$)	88.63	117.25	34.13	32.75

N.3 Masking rate and realized changes on Sudoku

We extend the masking rate ablation in Section 5.3 to Sudoku, varying $\rho_{\max}$ and $\rho_{\min}$ while following the same geometric annealing schedule. We additionally measure the percentage of masked variables whose discrete assignments change after each reverse diffusion step. This allows us to distinguish the masking rate from the actual number of variables modified by the model.

Table 15 reports accuracy on Sudoku-RRN and the realized change rates at selected inference steps. Similar to MIS, schedules that begin with large blocks and anneal to moderately smaller blocks achieve the best performance. The realized change rate also decreases substantially over inference, indicating that the model increasingly preserves assignments even among the selected variables. In particular, for the best-performing schedule $(\rho_{\max}, \rho_{\min}) = (0.9, 0.5)$, the percentage of masked variables that change decreases from 88.86% at the first step to 1.00% at the final step.

These results suggest that relatively large masking rates do not necessarily lead to equally large changes in the assignment. At the same time, excessively small final masking rates can restrict the variables available for repair, whereas large final masking rates may disrupt high-quality assignments.

Table 15: Masking-rate ablation on Sudoku-RRN. We report solution accuracy (%) and the percentage of masked variables whose discrete assignments change at selected inference steps. The masking rate is annealed geometrically from $\rho_{\max}$ to $\rho_{\min}$.

$\rho_{\max}$	$\rho_{\min}$	Accuracy %	Step 1	Step 2	Step 100	Step 1000	Step 2000
0.3	0.1	30.10	88.11	62.86	3.82	0.83	0.48
0.5	0.1	57.18	87.92	52.61	3.74	0.65	0.23
0.5	0.3	67.52	88.10	52.68	3.46	1.07	0.62
0.7	0.1	81.79	88.29	32.71	3.51	0.41	0.14
0.7	0.3	88.58	88.59	33.43	5.73	0.71	0.33
0.7	0.5	93.36	88.60	33.99	9.60	3.77	1.27
0.9	0.1	87.81	85.09	28.18	14.09	0.39	0.12
0.9	0.3	91.73	86.40	25.65	17.52	1.50	0.35
0.9	0.5	**94.12**	88.86	14.47	6.20	7.39	1.00
0.9	0.7	83.54	88.49	15.44	5.76	6.37	5.32

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: We describe all claims in our abstract as well as provide a list of contributions in our introduction.

   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: see Section 7.

   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: See Appendix B.

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 the code in supplementary materials and detail the hyper parameters in Appendix G.

   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: We provide the code in supplementary materials.

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 discuss training details in Appendix G.

   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: [Yes]

   Justification: For MaxCut, which uses multi-start inference as part of the standard evaluation protocol, Appendix M reports mean $\pm$ standard deviation across 20 independent inference trajectories for each instance and ablation variant. For the remaining benchmarks, we follow the standard reporting protocol of prior work and report point estimates under fixed evaluation budgets.

   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: discussed in Appendix G.

   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 agree and confirm our work follows the 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: The goal of the paper is to advance heuristic solvers for constraint optimization problems. Solving these problems will bring a positive social impact as they are important for everyday life as discussed in the introduction.

    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: [N/A]

    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 cite the relevant papers which our paper compares against.

    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: [Yes]

    Justification: We provide instructions for running our code.

    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: [N/A]

    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: [N/A]

    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: [N/A]

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.