DiffusionPack: Bin Packing with Custom Human Preferences

Anonymous Author(s)
Affiliation
Address
email

Abstract

The three-dimensional bin packing problem (3D-BPP) is a challenging NP-hard task with real-world applications in logistics and manufacturing. Traditional approaches rely on manually designed heuristics and lack flexibility in handling complex, customizable preferences. We introduce a diffusion-based offline framework that operates in a continuous solution space and supports test-time integration of human preferences. We utilize pretrained large language models (LLMs) as preference interpreters and interaction mediators. LLMs can help understand and interpret natural language preferences into guidance functions and inpainting rules. Our method enables zero-shot adaptation to user-defined requirements, achieving efficient, high-quality packing without retraining. We view this as a step toward agentic systems that unite multimodal planning and human-AI collaboration for real-world decision-making tasks.

1 INTRODUCTION

The regular three-dimensional bin packing problem (3D-BPP) is a fundamental and challenging combinatorial optimization problem. It involves packing a set of cuboid-shaped items into bin(s), usually optimizing for high packing density. This problem is crucial in various industries, including logistics, manufacturing, and supply chain management. Practical applications require not only high packing density but also support for safety considerations and user preferences. For instance, A warehouse manager may require fragile glassware to be placed above sturdier items like books to prevent breakage. They may also prefer glassware to be grouped for easier handling. Such mixed scenarios are vital in practice but hard to model in traditional methods.

A classical 3-D Bin Packing Problem (BPP) involves arranging multiple cuboid items within bin(s) while satisfying constraints such as avoiding overlaps and staying within bin boundaries. The objective of bin packing typically involves maximizing the utilization rate (UR) [Martello et al., 1998]. The Bin Packing problem has two popular forms: 1) Online: items arrive sequentially and decisions must be made quickly with a relatively limited action space, and 2) offline: where all items are known in advance, allowing for global optimization but significantly expanding the planning space, requiring consideration of both the item sequence and their placement positions. Incorporating human preferences introduces additional complexity. Our work targets offline packing into a single bin, guided by user preferences communicated in natural language.

Exact methods, such as the branch-and-bound [Schepler et al., 2022], solve the Bin Packing Problem via mixed-integer programming, yielding optimal solutions given enough time. However, they become painstakingly slow when scaling and are challenging to adapt to incorporate custom preferences. Consequently, the field has largely relied on heuristic and approximation algorithms. Although numerous approaches have been proposed [Dósa, 2007], [Johnson, 1974], their effectiveness is limited due to local decision-making. Conventional heuristics typically treat problem instances in isolation, overlooking shared structural patterns. Moreover, their rigid, handcrafted rules and dependence on discrete placement spaces make it difficult to accommodate custom preferences. These limitations highlight the strong need to exploit such patterns through data-driven and learning-based methods.

Learning-based 3D bin packing methods [Zhao et al., 2022a] outperform traditional approaches under complex constraints like stability. Deep RL has been applied to offline 3D BPP by decomposing it into sequencing and placement tasks [Zhang et al., 2021]. However, RL methods are difficult to train due to the large continuous action space and face two key challenges: designing effective reward functions and balancing exploration with exploitation. Supporting custom packing preferences often requires retraining, and models usually adapt only to preferences seen in the training data. While stability is well studied, preferences such as placing fragile items on top or grouping similar items are still difficult to model. Constraint-aware training can help, but generalizing to new, unseen preferences remains a challenge.

Recently, Diffusion models [Ho et al., 2020] have been applied to combinatorial optimization problems, including the Travelling Salesman Problem and Maximum Independent Set [Sun and Yang, 2023], [Sanokowski et al., 2024]. Building on these advances, we propose learning the distribution of complete packing configurations directly from supervised data. We formulate the 3D Bin Packing Problem as a conditional generation task, where a diffusion model predicts the joint positions of all cuboids given their dimensions. This differs from most learning-based methods, which typically decompose the offline problem into two separate stages: Sequence selection followed by Position prediction. Diffusion models also offer practical advantages, including simpler training compared to reinforcement learning approaches. While diffusion models effectively capture overall arrangement and order, they lack placement accuracy(see Fig.2). We employ keypoints as projection guides to ensure feasible placements. Our method combines diffusion-based generation with keypoint-guided projections to produce valid packing configurations. We train on the cutting stock dataset [Laterre et al., 2018], and we show that the trained model generalizes well to other data distributions. Moreover, as demonstrated in Section X, our method handles complex user preferences at test time without constraint-specific training.

While LLMs are effective at understanding natural language and decomposing complex tasks, they are not well-suited for direct application to NP-hard problems such as bin packing. This is due to the very large optimization space and the strict constraints involved. Instead, we use their strengths in language understanding, problem formulation, and code generation to translate user preferences into optimization objectives. These objectives are then passed to a diffusion-based planner, allowing preferences to be added flexibly at test time.

Our key contributions are:

- A novel bin packing planner that uses diffusion models to predict the object placement directly to ensure feasible and valid packings.
- A flexible framework for integrating human preferences directly at inference, and
- We enable natural language–based packing by integrating large language models, making this the first bin packing approach with a natural language interface, to our knowledge.

2 Related works

2.1 Offline Bin Packing

Exact Methods and Heuristics-based Approaches:
Classic algorithms such as those by [Martello and Toth, 1992], [Lysgaard et al., 2004] systematically explore the search space, pruning branches based on bounds to find near-optimal solutions. Integer Linear Programming (ILP): ILP-based techniques have been utilized since [Dyckhoff, 1981], with more recent improvements by [Cambazard and O'Sullivan, 2010] and [Salem and Kieffer, 2020]. [Crainic et al., 2008] proposed a heuristics-based method where each new item is placed at container corners ("extreme points"), aiming to maximize space utilization. [Fanslau and Bortfeldt, 2010] introduced block-building strategies, wherein items are combined into rectangular blocks, which are subsequently stacked to improve packing efficiency. Diverse strategies such as genetic algorithms, tabu search, and guided local search (e.g., [Faroe et al., 2003]; [Crainic et al., 2009]; [Kang et al.,

2012], [Prasetyo et al., 2018]) have addressed the combinatorial complexity by exploring solution spaces using stochastic or adaptive rules. Due to the complexity of exact methods, heuristics are used, but they are rigid and poorly adapted to custom preferences. Our approach uses a learning-based framework in continuous space, enabling constraint integration at test time.

Learning based Methods:
[Hu et al., 2017] proposed a two-stage method combining deep reinforcement learning for packing order and heuristics for placement. This decomposition strategy has since been widely adopted in subsequent studies (e.g., [Duan et al., 2019]; [Hu et al., 2020]; [Zhang et al., 2021]). Ranked Reward (RR) [Laterre et al., 2018] applies self-play reinforcement learning on the full combinatorial action space. [Jiang et al., 2021] have proposed a multimodal architecture for offline BPP. More recently, [Boliang et al., 2024] integrated generative adversarial networks (GANs) with genetic algorithms to enhance exploration–exploitation balance. All of these approaches treat packing order and placement as separate subproblems, which constrains their capacity for joint preference optimization. Moreover, they typically employ masking strategies to restrict the action space. In contrast, our diffusion-based framework predicts item positions directly, enabling more effective utilization of the continuous action space. Recent works such as [Liu et al., 2023], [Wei et al., 2023], and [Wu et al., 2022] show that diffusion models can implicitly learn spatial layouts for object rearrangement. [Xue et al., 2023] and [Xue et al., 2024] apply diffusion to 2D irregular packing via learned gradient fields, but neither extends to full 3D bin packing nor incorporates human preferences.

2.2 Bin Packing with Custom Preferences

Most prior work on bin packing emphasizes geometric compactness, with some addressing stability, load balancing, or stacking preferences. Rule-based methods impose precedence or hierarchy constraints for items to be packed into different bins [Wojciechowski et al., 2024] or use mixed-integer programming for load balancing [Erbayrak et al., 2021]. Learning-based approaches have been utilized to encode these preferences. [Zhu et al., 2024] employs an encoder–decoder model along with branching for stacking constraints. [Zhao et al., 2022b] uses reinforcement learning for online bin packing with stability constraints using packing configuration trees. [Yang et al., 2023] incorporates preferences via energy functions but scales only to small 3D cases(5- 6 objects). These methods are rigid, requiring retraining for every new preference. In contrast, our framework flexibly integrates diverse human preferences at test time without any retraining.

3 Methodology

We address offline bin packing in a single container, representing placements by cuboid centroids. Orientations are not considered, and all cuboids are assumed to be axis-aligned. The objective is to place all cuboids, given their dimensions $(l_1, w_1, h_1), (l_2, w_2, h_2), \ldots, (l_N, w_N, h_N)$, into the bin (L, W, H) such that: 1) every cuboid lies entirely within the container, and 2) no two cuboids overlap. We model the bin packing problem as a conditional generation task, in which centroids are produced based on their dimensions. All cuboid centroids are determined jointly using a diffusion-based optimization process. Importantly, the training process does not involve explicit reward optimization; instead, the model learns to generate placements by imitating teacher-provided data.

3.1 Diffusion Model For Our Approach

Diffusion models transform a sample from a data distribution $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ into white Gaussian noise via a forward Markovian diffusion process $q(\mathbf{x}_t|\mathbf{x}_{t-1}, t) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t I)$, where $t = 1, \ldots, T$ is the diffusion time step, N is the total number of steps, and β_t denotes the noise scale at time step t.

The marginal distribution of the diffusion process at time t can be expressed in closed form as $q(\mathbf{x}_t|\mathbf{x}_0, t) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)I)$, where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$. This closed form enables direct sampling of noisy centroid positions $\mathbf{x}_t$ without running the full forward process. The reverse process aims to recover the centroid distribution by transforming Gaussian noise back into structured centroid sets through a learned denoising model: $p(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c}, t) \approx \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, \mathbf{c}, t), \Sigma_t)$, where only the mean μ_θ is learned, and the covariance is fixed as $\Sigma_t = \tilde{\beta}_t I$, with $\tilde{\beta}_t = \frac{\beta_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}$. Following [Ho et al., 2020], instead of learning the posterior mean directly, the model is trained to

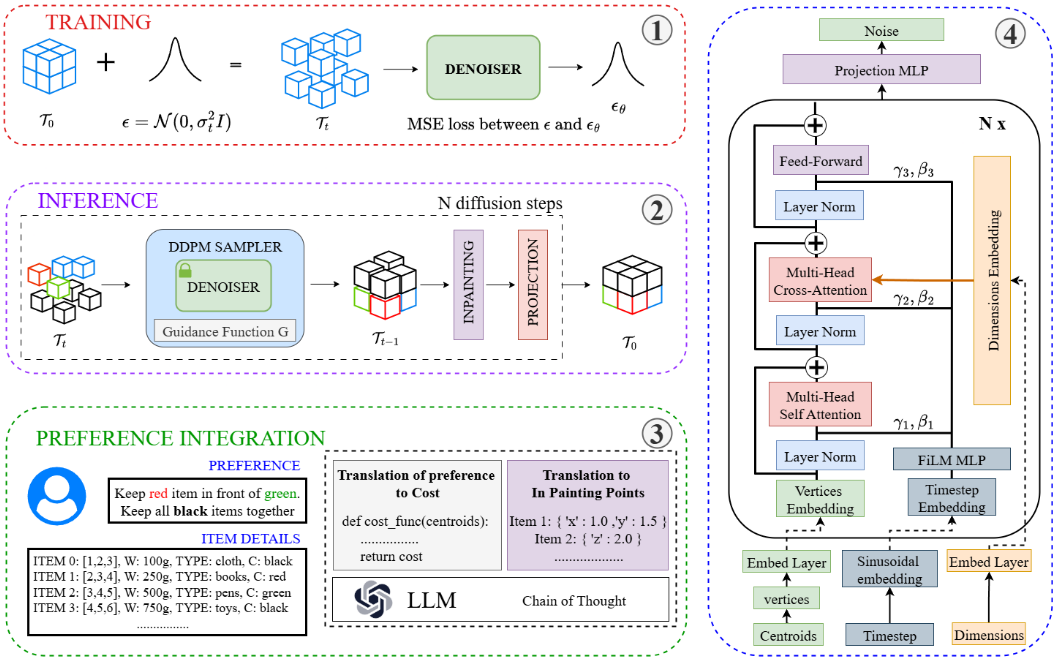

Figure 1: Overview of the proposed framework. **(1) Training:** The diffusion model learns to denoise cuboid centroids with MSE loss. **(2) Inference:** Candidate packings are generated through iterative denoising, projection, and inpainting. **(3) Preference Integration:** User preferences expressed in natural language are translated by an LLM into cost functions and inpainting rules. **(4) Denoiser Architecture:** Attention-based network enabling permutation-invariant packing.

predict the noise ε, with the mean expressed as $\mu_\theta(\mathbf{x}_t, \mathbf{c}, t) = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \varepsilon_\theta(\mathbf{x}_t, \mathbf{c}, t) \right)$, and optimized using the simplified objective $\mathcal{L}(\theta) = \mathbb{E}_{t,\varepsilon,\mathbf{x}_0} \left[\|\varepsilon - \varepsilon_\theta(\mathbf{x}_t, \mathbf{c}, t)\|_2^2 \right]$, where $t \sim \mathcal{U}(1, N)$, $\varepsilon \sim \mathcal{N}(0, I)$, and $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon$.

The sampling step in the reverse diffusion process refers to iteratively drawing samples from $p(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{c}, t)$ until we reach $\mathbf{x}_0$.

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \varepsilon_\theta(\mathbf{x}_t, \mathbf{c}, t) + \sqrt{\tilde{\beta}_t}\, \mathbf{z} \right) \tag{1}$$

We model each packing instance using a conditioning vector $\mathbf{c} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_N\}$, where $\mathbf{c}_i \in \mathbb{R}^3$ encodes the dimensions of the i-th cuboid, and a data sample $\mathbf{x}_0 = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N\}$, where $\mathbf{p}_i \in \mathbb{R}^3$ denotes its centroid. We employ an attention-based architecture that treats each cuboid as a token, enabling the denoiser to capture global contextual information. The training and inference pipelines are illustrated in Fig. 1:1&2, and the denoiser architecture is shown in Fig. 1 3.

3.2 Diffusion-Guided Packing With Projections

While diffusion models effectively generate content aligned with the data distribution, enforcing strict predefined constraints remains challenging due to their inherent stochasticity. Training on constraint-satisfying data may help, but it does not ensure constraint adherence. Projected Diffusion Models[Christopher et al., 2024] address this by introducing a projection operator,

$$\mathrm{P}_C(x) = \arg\min_{y \in C} \|y - x\|_2^2,$$

which maps a point x to its nearest feasible counterpart in the constraint set C. To maintain feasibility, each sampling step from Eq. 1 during the inference step is modified as:

$$\mathbf{x}_{t-1} = \mathrm{P}_C \left(\frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \varepsilon_\theta(\mathbf{x}_t, \mathbf{c}, t) + \sqrt{\tilde{\beta}_t}\, \mathbf{z} \right) \right) \tag{2}$$

While [Christopher et al., 2024] uses a gradient-based projection, we introduce a projection operator that adjusts predicted placements to the nearest valid packing keypoints. Such keypoints, widely used in the literature (e.g., corner-point [Martello et al., 2000], extreme-point [Crainic et al., 2008]), guide towards collision-free placements within the container. The projection function incrementally arranges cuboids to avoid overlaps while respecting inpainting preferences. It begins by placing the cuboid closest to the origin at the coordinate $(0, 0, 0)$. Subsequent cuboids are positioned iteratively: for each unplaced cuboid, a set of candidate points is generated from the already placed cuboids. These candidates are then filtered through feasibility checks, ensuring they satisfy spatial constraints and user-defined inpainting rules. Among the feasible options, the cuboid is projected to the location that remains closest to its originally predicted centroid. This process repeats until all cuboids are assigned or no valid configuration exists. Section 4.4 provides an analysis of keypoint strategies. A simplified 2D example using corner-point heuristics as a key point proposal is shown in Fig. 2. For a detailed algorithm, check the supplementary(Algorithm 1).

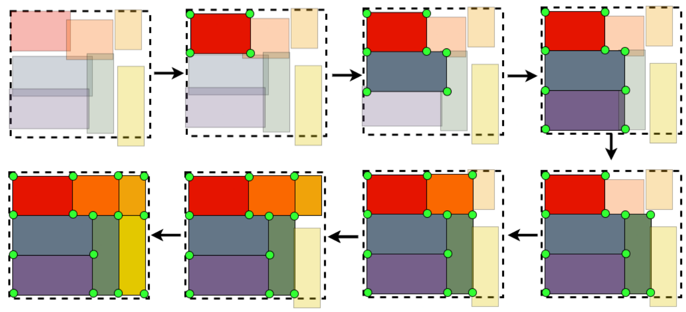

Figure 2: Step-by-step 2D illustration of projection using keypoints. Keypoints are shown in green, projected cuboids in dark, and remaining cuboids in light.

3.3 Selection and Pruning

We sample multiple cuboid configurations by iteratively refining them, removing predicted noise at each step using a trained diffusion model. The projection operator ensures that cuboids fit inside the unit cube without overlapping. We found that applying the projection operator only during the last step of the diffusion process gives the best results in practice(Refer to 4.4). After the final step, each configuration is evaluated using two metrics: support area (indicating physical plausibility) and packing density (measuring space efficiency). Configurations violating spatial feasibility or inpainting preferences are pruned, and the best ones are selected based on high support area and utilization rate. The complete algorithm is detailed in 2.

3.4 Applying preferences for Human-Guided Bin Packing

Diffusion models enable the integration of custom cost functions during sampling. A straightforward mechanism for imposing hard constraints is *inpainting*, where regions subject to fixed preferences are overwritten after each denoising step. This operation can be expressed as

$$\mathbf{x}_{t-1} = (\mathbf{1} - \mathbf{M}) \odot \tilde{\mathbf{x}}_{t-1} + \mathbf{M} \odot \mathbf{y}_0, \tag{3}$$

where $\mathbf{M}$ is a binary mask that specifies which cuboids are associated with hard inpainting constraints, ensuring that their values are preserved during the diffusion process. $\tilde{\mathbf{x}}_{t-1}$ is the unconstrained output from the diffusion update, and $\mathbf{y}_0$ denotes the reference values used to enforce the hard constraints.

Another popular approach is to guide the sampling towards optimizing a specific cost function. Following the motion planning diffusion framework [Carvalho et al., 2023], the transition probability from x_t to x_{t-1}, conditioned on time t and cost O, given as,

$$\log p(x_{t-1} \mid x_t, t, O) = \log \mathcal{N}(z; \mu_t + \Sigma_t g, \Sigma_t), \tag{4}$$

where the guidance term $g = \nabla_{x_{t-1}} \log p(O \mid x_{t-1})\big|_{x_{t-1}=\mu_t}$ is the gradient of the log-likelihood of the cost O with respect to x_{t-1}, evaluated at $x_{t-1} = \mu_t$.

Assuming the cost functions $c_i \in \mathcal{O}$ are locally differentiable, the guidance term g becomes:

$$g = -\sum_i \lambda_i \nabla_{x_{t-1}} c_i(x_{t-1})\Big|_{x_{t-1}=\mu_t}, \tag{5}$$

where λ_i are scalar weights for each cost, and the costs c_i are differentiable with respect to the trajectory. We combine both methods described above and apply custom human preferences by combining in-painting for hard preferences and cost functions for soft preferences.

3.5 Test-time preference integration using LLMs

We enable user preferences to be expressed in natural language. Using LLMs, these preferences are translated into formal constraints and cost functions, which are then integrated into the diffusion-based planner. Our approach proceeds as follows: users specify preferences (e.g., "Cuboid 1 should be on top of Cuboid 4" or "heavier cuboids should be at the bottom"). The LLM, prompted with these constraints and general instructions, generates Python code encoding both cost functions and inpainting rules (Eq.3). Preferences are then represented in two forms: Hard preferences, enforced through inpainting after each denoising step using in-painting rules, and Soft preferences, expressed as differentiable cost functions for gradient-based guidance (Eq. 4 & 5). Candidate solutions produced by the diffusion model are presented to the user. If a solution does not meet user expectations, the instructions can be refined, and the LLM updates the cost functions and inpainting rules, as shown in Fig. 1 (3). Prompts used can be found in Appendix A.4.2 & A.4.1 and example generation by LLM can be found in Appendix A.3 & A.2. We run all experiments with GPT-4o, fixing the temperature at 0.2.

4 Experiments and Results

In this section, we present a thorough analysis of our proposed approach. Our evaluation is structured to address the following key questions:

1. Does the proposed approach generate valid and space-efficient packing configurations?
2. How effectively can it incorporate custom constraints specified at test time?
3. What is the contribution of the Diffusion backbone and projections in the overall pipeline?

4.1 Dataset

Public benchmarks for 3D bin packing are limited and often lack cognitive relevance. To evaluate our approach, we use two widely recognized datasets: the **Cutting Stock Dataset (CSD)** and the **Randomly Sampled Dataset (RSD)**. All cuboid dimensions are normalized relative to the container size $(1, 1, 1)$ to ensure generalizability across different container configurations.

Cutting Stock Dataset (CSD): Following Laterre et al. (2018), Section 3.2, we generate spatially efficient layouts by recursively splitting the container into smaller cuboids, with a bias toward longer dimensions and a minimum size constraint. We train on this dataset. Notably, CSD allows for a 100% occupancy upper bound, providing a clear benchmark for evaluating packing efficiency. CSD-16 and CSD-32 denote datasets with 16 and 32 cuboids, respectively. We employ variable container splits (with cutting dimensions selected from 1–10) to ensure the model learns efficient packing rather than simple filling.

Randomly Sampled Dataset (RSD): To assess generalization, we evaluate on an unseen dataset of randomly sampled cuboids, not used during training. Each cuboid dimension is uniformly sampled from the integer range $[2, 5]$, within a container of size $10 \times 10 \times 10$. To avoid trivial packing configurations, we constrain the sampled cuboid dimensions such that $\frac{\min(L,W,H)}{10} \leq l_t, w_t, h_t \leq \frac{\min(L,W,H)}{2}$. The dimensions are then normalized to fit within the unit container $(1, 1, 1)$. RSD-16 and RSD-32 denote packing sequences containing 16 and 32 sampled cuboids, respectively.

4.2 Model details and Training

We adopt a standard DDPM framework with cosine variance schedule, where the denoiser predicts added noise at each diffusion step and is trained via mean squared error against the true noise. Inputs

Table 1: Performance comparison across different methods and settings. Each setting reports Utilization Rate (UR) and the number of stable configurations generated (SCG). Diffusion-CP, -EP, and -EMS correspond to our approach with CP, EP, and EMS projection functions.

Method	**CSD-16**		**CSD-32**		**RSD-16**		**RSD-32**	
	UR	SCG	UR	SCG	UR	SCG	UR	SCG
Heuristic								
First Fit (EP)	56.5±0.4	100.0	48.2±0.6	100.0	44.0±0.2	100.0	44.7±0.2	100.0
Best Fit (EP)	55.0±0.9	100.0	54.4±1.0	100.0	58.6±0.1	100.0	67.3±0.2	100.0
Key Point	60.7±1.0	100.0	60.1±1.2	100.0	63.0±1.1	100.0	79.9±0.3	100.0
DBLF	59.1±0.2	100.0	58.6±0.4	100.0	61.1±0.1	100.0	69.1±0.1	100.0
Learning-based								
BRKGA	52.6±0.9	100.0	51.2±1.0	100.0	67.4±0.2	100.0	71.1±0.2	100.0
DBLF + GA	61.3±0.4	100.0	56.4±0.5	100.0	68.5±0.3	100.0	77.5±0.1	100.0
Diffusion-CP	86.2±0.4	99.7	70.8±0.7	99.6	65.5±0.2	98.6	75.7±0.1	99.6
Diffusion-EP	89.3±2.7	99.8	79.2±2.5	98.9	65.5±1.7	99.3	79.7±0.8	98.8
Diffusion-EMS	**89.3±2.6**	**99.1**	**80.2±0.1**	**99.8**	**69.0±1.5**	**99.9**	**81.0±0.3**	**99.3**

consist of cuboid centroids and dimensions, together with the diffusion timestep. Our backbone is a modified DiT, adapted to structured 3D inputs for the packing task. Each cuboid is represented by its corner vertices, while dimensions are separately embedded and injected through cross-attention at every block, enforcing geometric consistency. Temporal embeddings are incorporated through sinusoidal encodings and FiLM modulation [Perez et al., 2018], enhancing temporal awareness across layers. The network outputs denoised centroids via a shared MLP head, with no positional encoding applied to preserve permutation invariance. The model comprises six DiT layers with hidden size 256 and eight attention heads. Each cuboid is converted to eight corner vertices, flattened, and projected via a linear layer. Dimension embeddings act as key–value pairs in cross-attention layers, providing geometric conditioning throughout the network. Sinusoidal timestep embeddings $\gamma(t)$ are added to token embeddings and used for FiLM modulation, applying learned scaling and bias parameters to normalized activations. Training is conducted on 1M samples for 500k steps using AdamW (learning rate 3×10^{-5}, batch size 2048). We use 500 training diffusion steps and 250 inference steps. At inference, 128 candidate sequences are generated via DDPM sampling, followed by pruning and selection (see Sec. 3.3). The model predicts denoised centroids independently for each cuboid token through a shared MLP head. All experiments are performed on an Intel i9 CPU with an RTX 6000 Ada GPU (48 GB VRAM). An overview of the architecture is shown in Fig. 1(4). Unless noted otherwise, we adopt a 70:10:20 split for train, validation, and test sets.

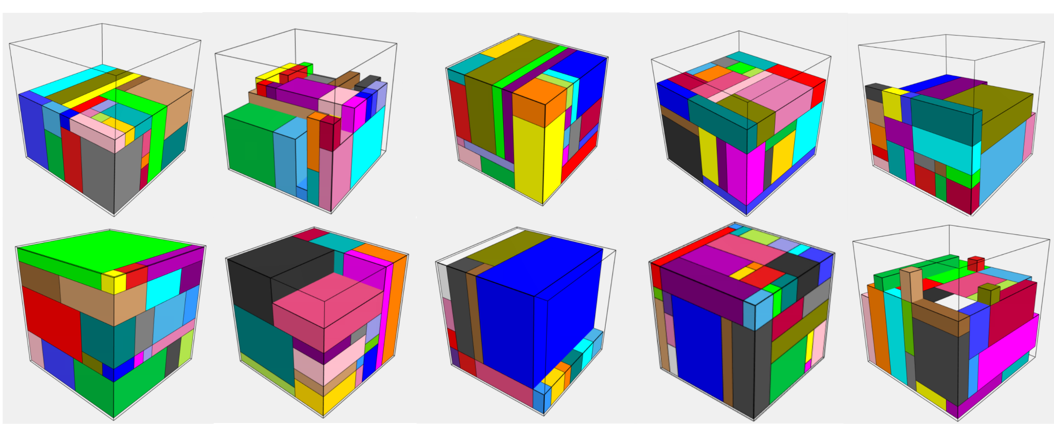

Figure 3: **Packing results for up to 32 cuboids** Our method aims to produce compact arrangements, even when the total cuboid volume does not fully span the container, yielding dense configurations.

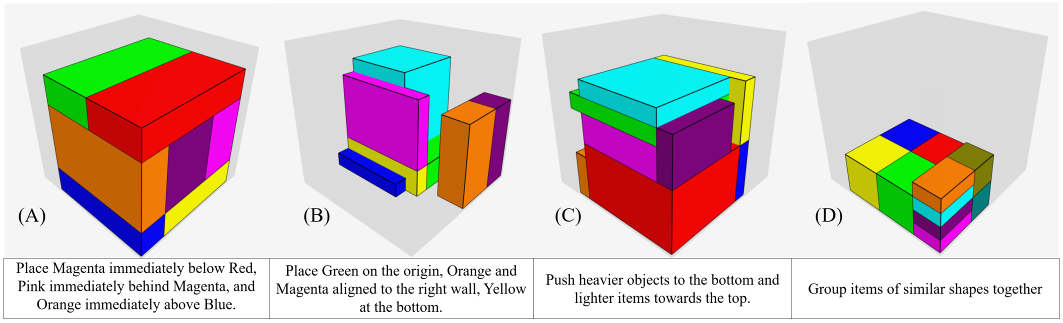

Figure 4: **Packing results with human-defined preferences** Panels A and B apply hard preferences via inpainting, while Panels C and D use guidance functions to steer the packing process iteratively. We present a limited set of cuboids here to keep the illustration clear and concise.

4.3 Evaluation

This subsection focuses on two aspects: (i) the effectiveness of *DiffusionPack* as a bin-packing planner, and (ii) its ability to integrate preferences into planning. We evaluate our method on two subsets (see Subsection 4.1): an unconstrained setting measuring utilization rate, and a constrained setting emphasizing human-imposed preferences. Utilization rate as $\mathrm{UR} = \frac{\sum_{i=1}^{N} V_i}{A_{\text{base}} \cdot H_{\max}}$, where V_i is the volume of cuboid i, A_{base} is the container base area, and $H_{\max}$ is the maximum packing height. Each evaluation set contains 1000 sequences, and results are averaged over 2 random seeds. We also report the number of stable generations(SCG) (where the support area for each cuboid must be >70/

4.3.1 3D bin packing without preferences

We compare our method against heuristics, genetic algorithms, and learning-based approaches (Table 1). Our method effectively learns global packing layouts across both full and partial sequences(Fig. 3, highlighting its ability to capture features critical for global optimization. Our method outperforms or matches most baselines. `CSD` provides a more reliable benchmark, as all items are guaranteed packable, unlike `RSD`, which may include infeasible sequences. Results are reported for 16 and 32 cuboids to evaluate the generalization variable number of cuboids.

4.3.2 3D bin packing with custom human preferences

We evaluate our method's ability to integrate custom preferences, defining success by constraint satisfaction. Using 100 unseen samples in each dataset from 4.1, we test zero-shot generalization across diverse constraint scenarios. This set includes human preferences, such as preferred placement regions, heavier items at the bottom, grouping similar items, and avoiding stacking on fragile ones. It also includes relative spatial constraints like proximity, alignment, and separation. Quantitative results are in Table 2. Fig. 4(A&B) show hard constraint enforcement, while (C&D) illustrate results guided by soft preferences.

Table 2: Performance on constraints dataset

Dataset	**Success Rate (%)**	**Utilization Rate (%)**
CSD-16	81%	70.4 ± 1.1
RSD-16	72%	61.2 ± 2.0
CSD-32	76%	69.6 ± 1.8
RSD-32	68%	52.3 ± 2.2

Table 3: Ablations on projection after diffusion steps

Method	**CSD-16**	**CSD-32**	**RSD-16**	**RSD-32**
Last-2 steps	83.2 ± 3.7	60.6 ± 3.7	64.3 ± 1.8	74.2 ± 0.4
Last-5 steps	70.4 ± 1.8	51.4 ± 0.2	62.3 ± 1.1	69.3 ± 0.6
Last-10 steps	61.1 ± 0.5	44.3 ± 1.3	58.2 ± 1.3	63.2 ± 0.0
Each step	51.0 ± 0.2	41.0 ± 4.2	54.1 ± 2.3	59.5 ± 2.3
Final step	87.3 ± 0.2	70.8 ± 4.2	64.9 ± 1.8	75.5 ± 0.1

4.4 Ablation Studies

Additional ablation studies are conducted to thoroughly analyze the impact of various components in our method. These components include the effect of the projection operator. When and how many times to apply the projection operator (Table 3). Also, we explore different keypoint generating methods that can be used for the projection operator (Table 1).

Ablations on Projection Operator

1. **Every step:** Apply projection at all t.
2. **Last k steps:** Apply projection for $t \in \{T-k+1, \ldots, T\}$
3. **Final step only:** Apply projection only at $t = T$.

For these ablations, we adopt EMS as the projection operator. Applying projection at every step biases the sampler, reduces diversity, and overconstrains placements, resulting in lower densities. Applying projection from the beginning is less sensible because early in sampling, cuboid positions have a large variance. The late-projection strategy aims to let the diffusion model explore broadly, and then perform small, local feasibility corrections once samples are near-converged. Projection during the last k steps may disrupt the diffusion process by pushing samples outside the Markov chain distribution. Restricting projection to the final step preserves guidance from the diffusion process while providing valid corrections with minimal interference.

Variants of projection-point selection.

We evaluate the effect of different selection strategies for candidate projection points (used inside the projection operator). We explore the following methods for getting the keypoints.

1. **Corner points(CP)**: Points are axis-aligned corners generated from already-placed cuboids.[Martello et al., 2000]
2. **Extreme-point (EP):** Points formed by surfaces of already placed cuboids and container boundaries. [Crainic et al., 2008]
3. **Empty Maximal Space (EMS)**: Candidate placements are proposed within the set of maximal empty spaces, defined as the largest axis-aligned rectangular regions that remain unoccupied after previous placements.[Ha et al., 2017]

EMS proves most effective as a projection operator, likely due to guidance towards unoccupied regions. This property is often exploited in heuristic methods. In contrast, CP and EP are not that efficient: EP explores fewer waypoints, while CP behaves greedily, restricting placements to corner alignments.

4.5 Discussion and Limitations

The strong performance of our hybrid approach is an outcome of combining the global planning strength of diffusion models with the precision of heuristic-based projection. Diffusion models capture overall order and placement distribution but lack fine-grained accuracy. Heuristics, while suboptimal alone, offer fast, valid placements aligned with container boundaries. Together, they yield more effective results than either method individually. Additionally, the continuous nature of diffusion enables flexible constraint integration. However, the method has limitations. Training becomes costly when scaling beyond 32 items, making scalability a key area for future work. In addition, LLM-generated costs may be incorrect or non-differentiable, leading to suboptimal handling of preferences. Performance also depends on how users phrase instructions, and conflicting constraints can prevent all requirements from being satisfied.

5 Conclusion

We propose a diffusion-based bin packing framework that directly generates feasible, constraint-aware layouts with support for custom human-defined rules. Unlike two-stage methods, our permutation-invariant model learns global layouts end-to-end. The ability to adapt to new preferences in a zero-shot manner without retraining is a key contribution. This makes the approach well-suited for dynamic, real-world applications where user requirements can change frequently. This work serves as a proof of concept, with future directions including scaling to more objects, supporting a wider range of preferences, and benchmarking against stronger baselines.

References

Zhang Boliang, Yu Yao, H. Kan, and Wuman Luo. A gan-based genetic algorithm for solving the 3d bin packing problem. *Scientific Reports*, 14, 04 2024. doi: 10.1038/s41598-024-56699-7.

Hadrien Cambazard and Barry O'Sullivan. Propagating the bin packing constraint using linear programming. volume 6308, pages 129–136, 09 2010. ISBN 978-3-642-15395-2. doi: 10.1007/978-3-642-15396-9_13.

Joao Carvalho, An T Le, Mark Baierl, Dorothea Koert, and Jan Peters. Motion planning diffusion: Learning and planning of robot motions with diffusion models. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1916–1923. IEEE, 2023.

Jacob K Christopher, Stephen Baek, and Nando Fioretto. Constrained synthesis with projected diffusion models. *Advances in Neural Information Processing Systems*, 37:89307–89333, 2024.

Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. Extreme point-based heuristics for three-dimensional bin packing. *INFORMS J. on Computing*, 20(3):368–384, July 2008. ISSN 1526-5528. doi: 10.1287/ijoc.1070.0250. URL `https://doi.org/10.1287/ijoc.1070.0250`.

Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. Ts2pack: A two-level tabu search for the three-dimensional bin packing problem. *European Journal of Operational Research*, 195:744–760, 06 2009. doi: 10.1016/j.ejor.2007.06.063.

György Dósa. The tight bound of first fit decreasing bin-packing algorithm is ffd(i) 11/9opt(i) + 6/9. In *Proceedings of the First International Conference on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, ESCAPE'07, page 1–11, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3540744495.

Lu Duan, Haoyuan Hu, Yu Qian, Yu Gong, Xiaodong Zhang, Jiangwen Wei, and Yinghui Xu. A multi-task selected learning approach for solving 3d flexible bin packing problem. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1386–1394, 2019.

Harald Dyckhoff. A new linear programming approach to the cutting stock problem. *Operations Research*, 29:1092–1104, 12 1981. doi: 10.1287/opre.29.6.1092.

Seda Erbayrak, Vildan Özkır, and U. Mahir Yıldırım. Multi-objective 3d bin packing problem with load balance and product family concerns. *Computers Industrial Engineering*, 159:107518, 2021. ISSN 0360-8352. doi: https://doi.org/10.1016/j.cie.2021.107518. URL `https://www.sciencedirect.com/science/article/pii/S0360835221004228`.

Tobias Fanslau and Andreas Bortfeldt. A tree search algorithm for solving the container loading problem. *INFORMS Journal on Computing*, 22:222–235, 05 2010. doi: 10.1287/ijoc.1090.0338.

Oluf Faroe, David Pisinger, and Martin Zachariasen. Guided local search for the three-dimensional bin-packing problem. *INFORMS Journal on Computing*, 15(3):267–283, 2003. doi: 10.1287/ijoc.15.3.267.16080. URL `https://doi.org/10.1287/ijoc.15.3.267.16080`.

Chi Trung Ha, Trung Thanh Nguyen, Lam Thu Bui, and Ran Wang. An online packing heuristic for the three-dimensional container loading problem in dynamic environments and the physical internet. In *Applications of Evolutionary Computation - 20th European Conference, EvoApplications 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings, Part II*, volume 10200, pages 140–155. Springer, 2017. URL `https://doi.org/10.1007/978-3-319-55792-2_10`.

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

Haoyuan Hu, Xiaodong Zhang, Xiaowei Yan, Longfei Wang, and Yinghui Xu. Solving a new 3d bin packing problem with deep reinforcement learning method. *arXiv preprint arXiv:1708.05930*, 2017.

Ruizhen Hu, Juzhan Xu, Bin Chen, Minglun Gong, Hao Zhang, and Hui Huang. Tap-net: transport-and-pack using reinforcement learning. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020.

Yuan Jiang, Zhiguang Cao, and Jie Zhang. Solving 3d bin packing problem via multimodal deep reinforcement learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, page 1548–1550, Richland, SC, 2021. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450383073.

D. S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8(3): 272–314, 1974.

Jay Kang, Ilkyeong Moon, and Hongfeng Wang. A hybrid genetic algorithm with a new packing strategy for the three-dimensional bin packing problem. *Applied Mathematics and Computation*, 219:1287–1299, 10 2012. doi: 10.1016/j.amc.2012.07.036.

Alexandre Laterre, Yunguan Fu, Mohamed Khalil Jabri, Alain-Sam Cohen, David Kas, Karl Hajjar, Torbjorn S Dahl, Amine Kerkeni, and Karim Beguir. Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization. *arXiv preprint arXiv:1807.01672*, 2018.

Weiyu Liu, Yilun Du, Tucker Hermans, Sonia Chernova, and Chris Paxton. Structdiffusion: Language-guided creation of physically-valid structures using unseen objects. In *Robotics: Science and Systems*, 2023.

Jens Lysgaard, Adam Letchford, and Richard Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100:423–445, 06 2004. doi: 10.1007/s10107-003-0481-8.

Silvano Martello and Paolo Toth. Generalized assignment problems. In Toshihide Ibaraki, Yasuyoshi Inagaki, Kazuo Iwama, Takao Nishizeki, and Masafumi Yamashita, editors, *Algorithms and Computation*, pages 351–369, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg. ISBN 978-3-540-47501-9.

Silvano Martello, David Pisinger, and Daniele Vigo. The three-dimensional bin packing problem. *Operations Research*, 48, 02 1998. doi: 10.1287/opre.48.2.256.12386.

Silvano Martello, David Pisinger, and Daniele Vigo. The three-dimensional bin packing problem. *Oper. Res.*, 48(2):256–267, March 2000. ISSN 0030-364X. doi: 10.1287/opre.48.2.256.12386. URL `https://doi.org/10.1287/opre.48.2.256.12386`.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Hari Prasetyo, Anandistya Putri, and Gusti Fauza. Biased random key genetic algorithm design with multiple populations to solve capacitated vehicle routing problem with time windows. volume 1977, page 020052, 06 2018. doi: 10.1063/1.5042908.

Khadija Salem and Yann Kieffer. *New Symmetry-less ILP Formulation for the Classical One Dimensional Bin-Packing Problem*, pages 423–434. 08 2020. ISBN 978-3-030-58149-7. doi: 10.1007/978-3-030-58150-3_34.

Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Xavier Schepler, André Rossi, Evgeny Gurevsky, and Alexandre Dolgui. Solving robust bin-packing problems with a branch-and-bound approach. *European Journal of Operational Research*, 297(3): 831–843, 2022. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2021.05.041. URL `https://www.sciencedirect.com/science/article/pii/S0377221721004690`.

Zhiqing Sun and Yiming Yang. Difusco: graph-based diffusion solvers for combinatorial optimization. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Qiuhong Anna Wei, Sijie Ding, Jeong Joon Park, Rahul Sajnani, Adrien Poulenard, Srinath Sridhar, and Leonidas Guibas. Lego-net: Learning regular rearrangements of objects in rooms. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19037–19047, 2023.

Piotr Wojciechowski, K. Subramani, Alvaro Velasquez, and Bugra Caskurlu. Priority-based bin packing with subset constraints. *Discrete Applied Mathematics*, 342:64–75, 2024. ISSN 0166-218X. doi: https://doi.org/10.1016/j.dam.2023.08.019. URL https://www.sciencedirect.com/science/article/pii/S0166218X2300327X.

Mingdong Wu, Fangwei Zhong, Yulong Xia, and Hao Dong. Targf: learning target gradient field to rearrange objects without explicit goal specification. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, 2022.

Tianyang Xue, Mingdong Wu, Lin Lu, Haoxuan Wang, Hao Dong, and Baoquan Chen. Learning gradient fields for scalable and generalizable irregular packing. In *SIGGRAPH Asia 2023 Conference Papers*, SA '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400703157. doi: 10.1145/3610548.3618235. URL https://doi.org/10.1145/3610548.3618235.

Tianyang Xue, Lin Lu, Yang Liu, Mingdong Wu, Hao Dong, Yanbin Zhang, Renmin Han, and Baoquan Chen. Gfpack++: Improving 2d irregular packing by learning gradient field with attention. *arXiv preprint arXiv:2406.07579*, 2024.

Zhutian Yang, Jiayuan Mao, Yilun Du, Jiajun Wu, Joshua B Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Compositional diffusion-based continuous constraint solvers. In *Conference on Robot Learning*, pages 3242–3265. PMLR, 2023.

Jingwei Zhang, Bin Zi, and Xiaoyu Ge. Attend2pack: Bin packing through deep reinforcement learning with attention. *arXiv preprint arXiv:2107.04333*, 2021.

Hang Zhao, Qijin She, Chenyang Zhu, Yin Yang, and Kai Xu. Online 3d bin packing with constrained deep reinforcement learning, 2022a. URL https://arxiv.org/abs/2006.14978.

Hang Zhao, Yang Yu, and Kai Xu. Learning efficient online 3d bin packing on packing configuration trees. In *International Conference on Learning Representations*, 2022b. URL https://api.semanticscholar.org/CorpusID:251649027.

Weiwang Zhu, Sheng Chen, Min Dai, and Jiping Tao. Solving a 3d bin packing problem with stacking constraints. *Computers Industrial Engineering*, 188:109814, 2024. ISSN 0360-8352. doi: https://doi.org/10.1016/j.cie.2023.109814. URL https://www.sciencedirect.com/science/article/pii/S0360835223008380.

A Technical Appendices and Supplementary Material

A.1 Algorithms

Algorithm 1 PROJECTION FUNCTION OUTLINE

Require: $\mathbf{C} \in \mathbb{R}^{BS \times L \times 3}$ (centroids), $\mathbf{D} \in \mathbb{R}^{BS \times L \times 3}$ (dimensions), $\mathcal{P}$ = inpainting preference set
Ensure: $\widehat{\mathbf{C}} \in \mathbb{R}^{BS \times L \times 3}$ (adjusted centroids, no overlaps)

Initialize $\widehat{\mathbf{C}} \leftarrow \mathbf{0}$
for $b \in \{1, \ldots, BS\}$ **do**
    $\mathcal{B} \leftarrow \varnothing$ ▷ Placed cuboids set
    $\mathcal{U} \leftarrow \{1, \ldots, L\}$ ▷ Unplaced indices
    Place initial cuboid:
    $i^* \leftarrow \arg\min_{i \in \mathcal{U}} \|\mathbf{C}_{b,i}\|_2$
    $\widehat{\mathbf{C}}_{b,i^*} \leftarrow (0, 0, 0)$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{i^*\}, \mathcal{U} \leftarrow \mathcal{U} \setminus \{i^*\}$
    Place other cuboid:
    while $\mathcal{U} \neq \varnothing$ **do**
        $\mathcal{Q} \leftarrow$ candidate points generated from $\mathcal{B}$
        $\mathcal{F} \leftarrow$ SelectFeasible$(\mathcal{Q})$
        for $j \in \mathcal{U}$ **do**
            for $\mathbf{q} \in \mathcal{Q}$ **do**
                $\mathbf{p} \leftarrow$ ProjectCuboid$(j, \mathbf{q})$
                if $\mathbf{p}$ respects $\mathcal{P}$ **then**
                    $\mathcal{F} \leftarrow \mathcal{F} \cup \{(j, \mathbf{p}, \|\mathbf{p} - \mathbf{C}_{b,j}\|_2)\}$
                end if
            end for
        end for
        if $\mathcal{F}$ is empty **then**
            Mark remaining $\mathcal{U}$ as unplaceable
            break
        else
            ▷ Select cuboid j^* with minimal distance d
            $(j^*, \mathbf{p}^*) \leftarrow \arg\min_{(j,\mathbf{p},d) \in \mathcal{F}} d$
            $\widehat{\mathbf{C}}_{b,j^*} \leftarrow \mathbf{p}^*$ ▷ Assign $\mathbf{p}^*$ to cuboid j^*
            $\mathcal{B} \leftarrow \mathcal{B} \cup \{j^*\}, \quad \mathcal{U} \leftarrow \mathcal{U} \setminus \{j^*\}$
        end if
    end while
end for
return $\widehat{\mathbf{C}}$

Algorithm 2 Diffusion-Guided Bin Packing with preferences

Require: Number of Samples B, Dimensions c, Denoising Model ε_ϑ, total steps N, Guidance costs G, Guidance weights λ, projection operator $\mathcal{P}_C$, Hard preferences H
Ensure: Final feasible cuboid configuration x_0

`Candidates` $\leftarrow$ [ ]
for $i = 1$ to B **do**
  Sample noise $x_T \sim \mathcal{N}(0, I)$
  for $t = N$ **to** 1 **do**
    $\mu_t \leftarrow \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \varepsilon_\theta(x_t, c, t) \right)$
    $g \leftarrow -\sum_k \lambda_k \nabla_{x_{t-1}} G_k(x_{t-1}) \Big|_{x_{t-1}=\mu_t}$
    $\mu_t \leftarrow \mu_t + \Sigma_t g$
    Sample $z \sim \mathcal{N}(0, I)$
    $x_{t-1} \leftarrow \mu_t + \sqrt{\tilde{\beta}_t} \cdot z$
    $x_{t-1} \leftarrow \text{Inpaint}(x_{t-1}, H)$
    if $t < T_{\text{proj}}$ **then**
      $x_{t-1} \leftarrow \mathcal{P}_C(x_{t-1}, H)$
    end if
  end for
  `Candidates`$[i] \leftarrow x_0^{(i)}$
end for
Evaluate metrics for each candidate $x_0^{(i)}$**:**
  – Compute support area for each cuboid
  – Compute Utilization rate (UR)
Prune: Discard unstable configurations
Select best:

$$x_0^* \leftarrow \arg \max_{x_0^{(i)} \in \texttt{Stable Candidates}} \text{UR}\left(x_0^{(i)}\right)$$

return x_0^*

A.2 Example Cost Functions generated by LLMs

Soft constraints are incorporated into our framework by defining cost functions that quantify how well a packing configuration satisfies certain preferences or requirements. Below, we provide examples of such cost functions generated by LLMs, illustrating how different types of soft constraints can be mathematically formulated and integrated into the model.

A.2.1 Push heavier objects to the bottom and lighter items towards the top

This cost penalizes large (heavy) items placed high along the vertical (z) axis. The goal is to encourage heavier items to be positioned closer to the bottom of the bin.

Let:

- $c_i = (x_i, y_i, z_i)$ denote the centroid of item i,
- $d_i = (w_i, h_i, l_i)$ denote the dimensions (width, height, depth),
- $v_i = w_i \cdot h_i \cdot l_i$ be the volume (proxy for weight) of item i.

Then the cost for item i is given by:

$$\text{cost}_i = z_i \cdot v_i$$

This penalizes high-z placement of large-volume items. The total cost over a sequence of n items is:

$$\text{TotalCost} = \sum_{i=1}^{n} z_i \cdot (w_i \cdot h_i \cdot l_i)$$

A.2.2 Group items of similar shapes together

This cost encourages items with similar dimensions to be spatially close, penalizing configurations where similar items are far apart.

Let:

- $c_i \in \mathbb{R}^3$ denote the centroid of item i,
- $d_i \in \mathbb{R}^3$ denote the dimensions (width, height, depth) of item i,
- $\hat{d}_i = \frac{d_i}{\|d_i\|}$ be the normalized dimension vector of item i,
- $s_{ij} = \hat{d}_i^\top \hat{d}_j$ be the cosine similarity between dimensions of items i and j,
- $d_{ij}^2 = \|c_i - c_j\|^2$ be the squared Euclidean distance between centroids of items i and j.

The pairwise grouping cost between items i and j is defined as:

$$\text{cost}_{ij} = \alpha \cdot s_{ij} \cdot d_{ij}^2$$

where α is a scaling factor (set to 10 in our implementation).

The total cost for item i is:

$$\text{cost}_i = \sum_{j=1}^{n} \text{cost}_{ij}$$

And the total cost over all n items is:

$$\text{TotalCost} = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha \cdot (\hat{d}_i^\top \hat{d}_j) \cdot \|c_i - c_j\|^2$$

This penalizes configurations where similar items (high s_{ij}) are placed far apart (high d_{ij}^2).

A.3 Example of inpainting rules generated by LLMs

Following the instructions, inpainting enforces specific data points at each sampling step. Examples demonstrate how inpainting the positions of cuboids effectively applies hard constraints.

A.3.1 Place Magenta immediately below Red, Pink immediately behind Magenta, and Orange immediately above Blue.

Given:
Centroids $\{\mathbf{c}_i\}$ and dimensions $\{\mathbf{d}_i\}$ for each object
Let $\mathbf{c}_i = (x_i, y_i, z_i)$ and $\mathbf{d}_i = (w_i, h_i, d_i)$

Apply Constraints:

- **Place Magenta below Red:**
  $\mathbf{c}_{\text{Magenta}} \leftarrow \mathbf{c}_{\text{Red}} - (0,\ 0,\ \frac{h_{\text{Red}} + h_{\text{Magenta}}}{2})$
- **Place Pink behind Magenta:**
  $\mathbf{c}_{\text{Pink}} \leftarrow \mathbf{c}_{\text{Magenta}} - (0,\ \frac{d_{\text{Magenta}} + d_{\text{Pink}}}{2},\ 0)$
- **Place Orange above Blue:**
  $\mathbf{c}_{\text{Orange}} \leftarrow \mathbf{c}_{\text{Blue}} + (0,\ 0,\ \frac{h_{\text{Blue}} + h_{\text{Orange}}}{2})$

A.3.2 Place Green on the origin, Orange and Magenta aligned to the right wall, Yellow at the bottom.

Given:
Centroids $\{\mathbf{c}_i\}$ and dimensions $\{\mathbf{d}_i\}$ for each object
Let $\mathbf{c}_i = (x_i, y_i, z_i)$ and $\mathbf{d}_i = (w_i, h_i, d_i)$
Unit cube bounds: $x, y, z \in [0, 1]$

Apply Constraints:

- **Place Green at origin:**
  $\mathbf{c}_{\text{Green}} \leftarrow \left(\frac{w_{\text{Green}}}{2}, \frac{d_{\text{Green}}}{2}, \frac{h_{\text{Green}}}{2}\right)$
- **Align Orange to right wall:**
  $\mathbf{c}_{\text{Orange},x} \leftarrow 1 - \frac{w_{\text{Orange}}}{2}$ (keep y, z unchanged)
- **Align Magenta to right wall:**
  $\mathbf{c}_{\text{Magenta},x} \leftarrow 1 - \frac{w_{\text{Magenta}}}{2}$ (keep y, z unchanged)
- **Place Yellow at bottom:**
  $\mathbf{c}_{\text{Yellow},z} \leftarrow \frac{h_{\text{Yellow}}}{2}$ (keep x, y unchanged)

A.4 PROMPTS

A.4.1 INPAINTING RULES PROMPT

AI Code Generation Task

You are an AI code generation assistant. Your task is to generate a **Python function** that updates 3D centroids of cuboids based on **natural language spatial commands**.

Inputs to the Function

1. `centroids`: A tensor of shape $(BS, seq\_len, 3)$
   Each entry is (x, y, z) coordinates of a cuboid's center.
2. `dimensions`: A tensor of shape $(BS, seq\_len, 3)$
   Each entry is $(width, height, depth)$ of a cuboid.
3. `index_dict`: Python dictionary mapping string labels (e.g., "red", "blue") to indices in `centroids` and `dimensions`.

Goal
Update `centroids` to enforce spatial constraints from the natural language command.

Rules and Behavior

- Translate spatial relations (e.g., "above", "on the left of") into mathematical updates on `centroids`.
- Use `index_dict` to resolve object labels.
- Express relations as relative offsets using both `centroids` and `dimensions`.
- Support batch processing over BS.

EXAMPLES of Spatial Relationship Logic and COORDINATE SYSTEM being used
Spatial Relationship Logic

- **above (z-axis)**

$$\texttt{pos[:, A, 2]} = \texttt{pos[:, B, 2]} + \frac{\texttt{Dim[:, B, 2]}}{2} + \frac{\texttt{Dim[:, A, 2]}}{2}$$

- **on the left of (x-axis)**

$$\texttt{pos[:, A, 0]} = \texttt{pos[:, B, 0]} - \frac{\texttt{Dim[:, B, 0]}}{2} - \frac{\texttt{DIm[:, A, 0]}}{2}$$

- **in front of (y-axis)**

$$\texttt{pos[:, A, 1]} = \texttt{pos[:, B, 1]} - \frac{\texttt{Dim[:, B, 1]}}{2} - \frac{\texttt{Dim[:, A, 1]}}{2}$$

Suggested Function Signature

```
def project_centroids(centroids, dimensions, index_dict):
    # Your implementation here for the command
    return centroids
```

Output Format

```
{
  'projection_code': "Your code goes here"
}
```

A.4.2 GUIDANCE FUNCTIONS PROMPT

AI Code Generation Task

You are an AI code generation assistant. Your task is to generate a **Python function** that updates 3D centroids of cuboids based on **natural language spatial commands**.

Inputs to the Function

1. `centroids`: A tensor of shape $(BS, seq\_len, 3)$
   Each entry is (x, y, z) coordinates of a cuboid's center.
2. `dimensions`: A tensor of shape $(BS, seq\_len, 3)$
   Each entry is $(width, height, depth)$ of a cuboid.
3. `index_dict`: Python dictionary mapping string labels (e.g., "red", "blue") to indices in `centroids` and `dimensions`.

Goal
A **differentiable cost function** that enforces the same spatial relationships using a `torch`-compatible loss, suitable for use with gradient-based guidance during diffusion.

Rules and Behavior

- Use `index_dict` to look up object indices (never hardcode).
- Ensure both functions are vectorized and batch-compatible.
- The cost function must return a tensor of shape (BS, seq_len).
- Use `F.relu()` or `torch.square()` to keep losses non-negative.
- Cost should be exactly zero when the spatial relation is perfectly satisfied.

Goal: `cost_function`
The function must return a tensor of shape (BS, seq_len) derived from `centroids` and `dimensions`. It should be:

- Continuous
- Designed so gradients push centroids toward satisfying the relation
- Usable inside a diffusion model loop for gradient-based guidance

Suggested Function Signature

```
def cost_function(centroids, dimensions, index_dict):
    # Your implementation here
    return costs  # tensor of shape (BS, seq_len)
```

Output Format

```
{
  'cost_function_code': "Cost function goes here"
}
```

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: Yes. The abstract and introduction clearly state the development of a diffusion-based planner augmented with LLMs for preference translation, and the experiments validate these claims.

2. **Limitations**

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

   Answer: [Yes]

   Justification: Section 4.5 clearly discuss the limitations of our approach.

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: [NA]

   Justification: [NA]

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: Most of the information needed to reproduce our results is provided in the main text, with prompts included in the appendices. Since this is a workshop paper, we plan to release the code after completing additional baselines and more extensive evaluations.

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: [No]

   Justification: Most of the information needed to reproduce our results is provided in the main text, with prompts included in the appendices. Since this is a workshop paper, we plan to release the code after completing additional baselines and more extensive evaluations.

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: [NA]

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: All metrics are reported across at least 2 seeds.

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:

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: [NA]

10. **Broader impacts**

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

    Answer: [NA]

    Justification: [NA]

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., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: [NA]

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: [NA]

13. **New assets**

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

    Answer: [NA]

    Justification: [NA]

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: [NA]

    Justification: [NA]

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: [NA]

    Justification: [NA]

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 rigorousness, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: We explain how we use LLMs in the methodology section.