"Stack It Up!": 3D Stable Structure Generation from 2D Hand-drawn Sketch

Yiqing Xu[1]* **Linfeng Li**[1] **Cunjun Yu**[1] **David Hsu**[1,2]
[1]School of Computing,
[2] Smart Systems Institute,
National University of Singapore

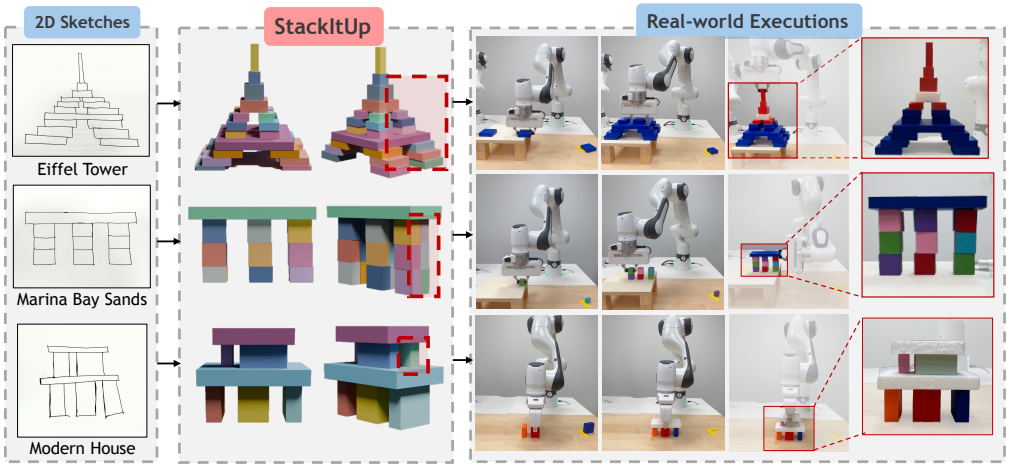


Figure 1: **Demonstration of StackItUp.** StackItUp allows non-experts to specify 3D structure for robot execution using a simple 2D sketch. From a rough front-view drawing, it predicts accurate 3D poses and hidden supports to generate stable structures that resemble the sketch. These poses can be directly used by a robotic arm as goal specification for motion planning and physical execution. Shown: input sketches (left), generated 3D structures with predicted supports highlighted (middle), and real robot executions (right).

Abstract: Imagine a child sketching the Eiffel Tower and asking a robot to bring it to life. Today's robot manipulation systems can't act on such sketches directly—they require precise 3D block poses as goals, which in turn demand structural analysis and expert tools like CAD. We present *StackItUp*, a system that enables non-experts to specify complex 3D structures using only 2D front-view hand-drawn sketches. StackItUp introduces an abstract relation graph to bridge the gap between rough sketches and accurate 3D block arrangements, capturing the symbolic geometric relations (*e.g.*, *left-of*) and stability patterns (*e.g.*, *two-pillar-bridge*) while discarding noisy metric details from sketches. It then grounds this graph to 3D poses using compositional diffusion models and iteratively updates it by predicting hidden internal and rear supports—critical for stability but absent from the sketch. Evaluated on sketches of iconic landmarks and modern house designs, StackItUp consistently produces stable, multilevel 3D structures and outperforms all baselines in both stability and visual resemblance. Our project page is available here.

Keywords: Robotic Goal Specification, Compositional Generative Models

1 Introduction

Imagine a child drawing a simple sketch of Eiffel Tower and asking a robot to build it. A human grasps the idea at a glance, but the robot freezes: the current robotic system requires the manipulation goal to be fully specified as exact 3D object poses—typically crafted in design software and verified

*xuyiqing@comp.nus.edu.sg

by force analysis—before motion planning and execution can even begin [1, 2, 3, 4, 5, 6, 7]. This creates a significant barrier for non-experts, making robot systems inaccessible to everyday users.

We propose *StackItUp*, a system that enables non-experts to specify complex 3D structures for robot manipulation using only a single 2D front-view hand-drawn sketch (Figure 1). The challenge lies in bridging the gap between the rough drawing and a complete 3D goal specification. First, hand-drawn sketches are metrically imprecise—objects may be distorted, misaligned, or physically implausible, making direct pose recovery unreliable. Second, a front-view sketch—precisely because it avoids perspective—is inherently incomplete: it omits interior and rear supports that are crucial for physical stability [8, 9, 10, 11, 12, 13]. To construct such a stable structure, the robot must infer both the missing supporting elements and the metrically accurate 3D poses of all blocks for execution.

Jointly predicting missing blocks and precise 3D poses creates a vast discrete–continuous search space [14, 15, 16, 17]. Our key idea is to introduce an *abstract relation graph* as an intermediate representation to manage this complexity [18, 19, 20]. This graph captures high-level geometric relations (*e.g.*, *left-of*) and stability patterns (*e.g.*, *two-pillar-bridge*), distilling essential structural cues while discarding noisy metric details from the sketch. When hidden supports are required for stability, subgraph matching against a dictionary of stability patterns efficiently reveals likely missing blocks and their relations. Moreover, this symbolic abstraction also enables flexible and scalable 3D pose generation: given any graph, we compose diffusion-based pose generators trained on individual relation types, and perform MCMC sampling over their combined scores to generate a coherent, stable 3D arrangement [18, 19, 21, 22].

StackItUp operates in two stages, as shown in Figure 2. First, it extracts an abstract relation graph from the sketch, capturing geometric relations and stability patterns among visible objects. Then, as illustrated in Figure 3, it grounds the graph to 3D poses using compositional diffusion models, checks stability via simulation, and iteratively updates this graph by adding hidden supports and relations as guided by the stability patterns until the generated 3D arrangement is stable.

We evaluate StackItUp on a diverse set of hand-drawn sketches depicting historical buildings (*e.g.*, Taj Mahal), iconic landmarks (*e.g.*, Marina Bay Sands), and modern architectural designs. As shown in Figure 4, these examples vary widely in appearance, internal support structure, and block dimensions. In all cases, StackItUp produces physically stable, multilevel 3D arrangements that faithfully reflect the sketch intent and outperform all baselines in both stability and visual resemblance.

2 Related Works

Sketch-Based Goal Specification for Robotics. Sketches have long been used in robotic systems to specify goals such as navigation targets, obstacles, or object interactions [23, 24]. More recent work explores sketch-based inputs for high-level manipulation tasks. For example, RT-Sketch [25] leverages hand-drawn inputs for object rearrangement, Sketch2Scene [26] generates detailed 3D game scenes from sketches and text with an emphasis on visual richness, and other works [27, 28] incorporate sketches into multimodal interfaces for task planning. However, these methods primarily focus on scene-level or trajectory goals, and do not address structure-centric reasoning. In contrast, StackItUp targets a fundamentally different challenge: generating physically stable, multi-object 3D structures from rough 2D sketches, which requires inferring both spatial layout and unseen supports.

User-Friendly Input for 3D Structure Generation. Recent systems translate high-level inputs such as natural language into object-centric structures. Blox-Net [29] leverages vision-language models to generate 3D assemblies that are physically plausible and robot-executable. QUERY2CAD [30] refines CAD models through natural language corrections. While these methods highlight the potential of user-friendly inputs for structure generation, they lack fine-grained spatial control and rely on natural language's limited ability to express geometric detail. In contrast, our sketch-based approach captures fine-grained spatial details and integrates explicit stability reasoning, thereby addressing a gap left by natural language–based methods.

3D Structure Generation from Single-View 2D Inputs. A related direction involves generating 3D object structures from single-view 2D inputs [31, 32]. StackGen [33] uses diffusion models to produce stacking layouts from silhouettes. However, it operates in 2D (x–z plane) and optimizes

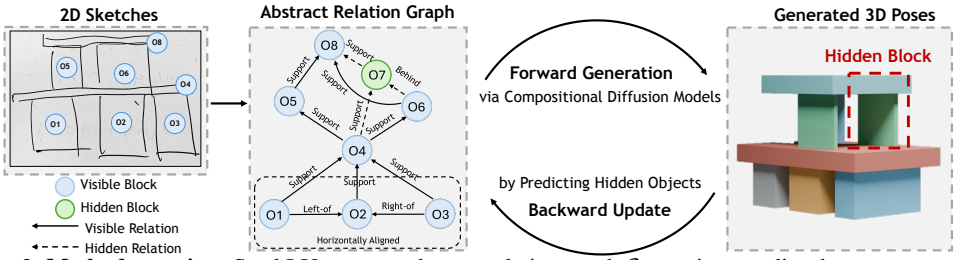


Figure 2: **Method overview.** StackItUp uses an abstract relation graph $\mathcal{G}$ as an intermediate between a rough 2D sketch (left) and the generated 3D block arrangement (right). The graph (middle) encodes high-level geometric and stability relations while abstracting away exact poses. StackItUp first extracts $\mathcal{G}_0$ from visible blocks in the sketch (blue nodes), then iteratively grounds it to 3D poses using compositional diffusion models. If instability is detected, the graph is updated with predicted hidden supports (green nodes), and re-grounded to 3D poses.

for visual similarity, without explicit reasoning about physical stability. Other methods such as Part123 [34] and follow-ups [35, 36, 37] reconstruct metrically accurate 3D models from images. Yet, these methods typically require photorealistic input views and produce unified meshes, making them unsuitable for block-based robot stacking tasks. Moreover, their reliance on fully observed objects limits applicability when key structural elements are occluded or missing.

3 Problem Formulation

StackItUp enables non-experts to specify their desired multi-level 3D structures to robots by drawing a 2D rough sketch, as shown in Figure 1. This sketch $\mathcal{S}$ is a front-view illustration of the desired structure that shows a set of visible blocks with labeled types. From this sketch, StackItUp must choose a set of blocks from a given library $T = \{\tau_1, \ldots, \tau_N\}$—each type τ_i representing a rigid, axis-aligned block of size $g_i = (w_i, l_i, h_i)$—and assign metric poses so that the resulting structure (i) remains static under gravity and (ii) closely resembles the user's sketch. Formally, the output is a set of blocks $\mathcal{O} = \{o_1, \ldots, o_M\}$ with $o_i = (\tau_i, p_i)$, where $\tau_i \in T$ is the block type and $p_i = (x_i, y_i, z_i)$ is the block's centroid. Equivalently, we write $\mathcal{O} = (\mathcal{T}, \mathcal{P})$, with $\mathcal{T} = (\tau_1, \ldots, \tau_M)$ collects the block types and $\mathcal{P} = (p_1, \ldots, p_M)$ for the corresponding poses. The sketch provides type labels for visible blocks $\mathcal{T}_{\text{obs}}$ but inaccurate pose cues $\mathcal{P}_{\text{obs}}$; it may also omit interior or rear supports. The final goal specification $\mathcal{O} = \mathcal{O}_{\text{obs}} \cup \mathcal{O}_{\text{hid}}$ must be physically stable and, when projected to the front view, preserve the abstract geometric relations implied by $\mathcal{S}$.

4 Stack It Up

StackItUp transforms a metrically imprecise 2D front-view sketch $\mathcal{S}$ into a 3D block arrangement $\mathcal{O}$ that is both physically stable and visually resembles the sketch. Central to StackItUp is an abstract relation graph $\mathcal{G}$, where nodes represent blocks and edges capture qualitative geometric relations (*e.g.*, *left-of*) and local stability patterns (*e.g.*, *two-pillar-bridge*), as an intermediate representation between the 2D sketch $\mathcal{S}$ and the 3D block arrangement $\mathcal{O}$ (see Table 1 for a complete list of relations used in this work). With this abstraction, the task splits naturally into two phases (Figure 2):

1. **One-time Graph Extraction.** Extract an initial graph $\mathcal{G}_0$ from sketch $\mathcal{S}$ for the visible blocks.
2. **Iterative Graph Grounding.** Ground the graph $\mathcal{G}_t$ to 3D poses $\mathcal{P}_t$ through compositional diffusion models, test them in simulation, and—when instability is detected—extend the graph with hidden supports, *i.e.*, $\mathcal{G}_{t+1} = \mathcal{G}_t \cup \mathcal{G}_t^{\text{hid}}$. The loop terminates once the structure stands.

Section 4.1 presents the geometric relation and stability pattern libraries and describes the one-time graph extraction from the sketch; Section 4.2 outlines the iterative forward–backward grounding algorithm that alternates between pose generation and graph update.

4.1 Abstract Relation Graph Extraction

We use an abstract relation graph $\mathcal{G}$ as an intermediate representation between a 2D sketch $\mathcal{S}$ and the final 3D block arrangement $\mathcal{O}$. This graph compactly encodes the qualitative geometric and stability relations that define the structure of multi-layer 3D structures. Formally, $\mathcal{G}$ consists of nodes representing blocks with their types $\mathcal{T}$, and edges denoting abstract relations from a set $\mathcal{R}$. Each relation $r_i(o_1^i, \ldots, o_{k_i}^i; R_i) \in \mathcal{G}$ has type $R_i \in \mathcal{R}$ and arity k_i, applying to a subset of blocks $\{o_1^i, \ldots, o_{k_i}^i\}$.

Table 1: Library of abstract relations, including both geometric relations and stability patterns.

Geometric Relations			
Front View, x-z plane		**Top-down View, x-y plane**	
left-of	left-in	front-of	front-in
right-in	center-in	back-in	touching-along-y
supported-by-partially	supported-by-fully	near-along-y	depth-aligned
horizontal-aligned	vertical-aligned-centroid	depth-aligned-in-a-line	regular-grid-sparse
vertical-aligned-left	vertical-aligned-right	regular-grid-compact	random-split-grid-sparse
horizontal-aligned-in-a-line	touching-along-x	random-split-grid-compact	
near-along-x			
Stability Patterns			
single-block-stack	cantilever-with-counterbalance	two-pillar-single-top-bridge	n-pillar-single-top-bridge
single-base-n-pillar-bridge	two-base-single-overhead-pyramid	n-base-single-overhead-pyramid	single-base-n-overhead-pyramid
n-base-m-overhead-pyramid	basic-arc		

The poses $\{p^i_1, \ldots, p^i_{k_i}\}$ are abstracted away by this intermediate representation. We divide relation types into geometric relations $\mathcal{R}_{\text{geom}}$ (e.g., *left-of*) that capture spatial layout, and stability patterns $\mathcal{R}_{\text{stab}}$ (e.g., *two-pillar-bridge*) that encode physical support. We begin by defining these relations and then describe how to extract the initial graph $\mathcal{G}_0$ from the sketch $\mathcal{S}$.

Geometric relations. We define 24 qualitative geometric relations (top of Table 1). Among these, 13 arise directly from the sketch because they depend only on bounding-box coordinates in the front (x–z) plane(*e.g.*, *left-of*, *touching-along-x*). The remaining 11 involve depths (*e.g.*, *front-of*, *touching-along-y*), therefore requiring x–y information and are only revealed once hidden supports are added. Each relation $R \in \mathcal{R}_{\text{geom}}$ is detected by a rule-based classifier h_R that inspects the bounding boxes projected into their respective planes; full rule sets appear in Appendix A.1.

Local Stability Patterns. Beyond geometric relations, we model 10 local stability patterns (bottom of Table 1), each pattern $R \in \mathcal{R}_{\text{stab}}$ describes a class of stable two-layer subgraphs in which a base tier supports a top tier; arbitrary compositions of these patterns yield stable multi-level structures. Each stability pattern $R \in \mathcal{R}_{\text{stab}}$ has a classifier h_R defined by four descriptors: (i) admissible counts $(\{n_{\text{base}}\}, \{n_{\text{top}}\})_R$ of blocks per tier; (ii) allowed "*supported-by*" subgraphs $\{\mathcal{G}^R_{\text{supp}}\}$ connecting base to top blocks; permissible geometric subgraphs (iii) among base blocks $\{\mathcal{G}^R_{\text{base}}\}$ and (iv) among top blocks $\{\mathcal{G}^R_{\text{top}}\}$, including relations such as *horizontally-aligned* and *regular-grid*. Although these patterns are manually defined, they compose flexibly to cover diverse multi-level structures—including variations in block size, layout, and hidden supports—and the framework remains extensible: new structures (*e.g.*, circle) can be incorporated by adding patterns to this dictionary. Given a candidate graph $\mathcal{G}^{\text{geom}}$ and a partition into top and base blocks, the classifier evaluation $h_R(o^1_{\text{base}}, \ldots, o^n_{\text{base}}, o^1_{\text{top}}, \ldots, o^m_{\text{top}})$ returns 1 when this candidate graph matches an permissible instance for each classifier descriptor. Complete classifier specifications appear in Appendix A.2.

Extracting Abstract Relation Graph from Sketch. The sketch $\mathcal{S}$ depicts the desired 3D structure through enclosed boxes in a front (x–z) plane, where each box corresponds to a visible block $o_i \in \mathcal{O}_{\text{obs}} = \{o_1, o_2, \ldots, o_K\}$ with an labeled type $\tau_i \in \mathcal{T}_{\text{obs}} = \{\tau_1, \tau_2, \ldots, \tau_K\}$. To handle imperfect or roughly drawn sketches, we first regularize the strokes by flood-fill and dilation, fitting an axis-aligned bounding box to each block and recording coarse dimensions $(\hat{w}_i, \hat{h}_i)$ and centroids $(\hat{x}_i, \hat{z}_i)$; these values only reflect qualitative geometric relations, but not the metrically accurate 3D pose $p_i = (x_i, y_i, z_i)$. Enclosed regions are treated as candidate blocks by default; users can mark regions as gaps to allow open structures. Looking ahead, we plan to use cues such as shading to automate gap detection. With the candidate blocks identified, a set of rule-based geometric classifiers $h_R,\ R \in \mathcal{R}_{\text{geom}}$, then operates on the bounding boxes $(\hat{w}_i, \hat{h}_i, \hat{x}_i, \hat{z}_i)_{i=1}^K$ to populate the geometric graph $\mathcal{G}^{\text{geom}}_0$. Next, stability-pattern classifiers $h_R,\ R \in \mathcal{R}_{\text{stab}}$, search $\mathcal{G}^{\text{geom}}_0$ for subgraph matches to known local stability patterns, producing the stability graph $\mathcal{G}^{\text{stab}}_0$. Their union yields the initial graph $\mathcal{G}_0 = \mathcal{G}^{\text{geom}}_0 \cup \mathcal{G}^{\text{stab}}_0$.

4.2 Abstract Relation Graph Grounding

To ground the graph $\mathcal{G}_0$ into stable 3D poses $\mathcal{P}$, we avoid training a single monolithic model conditioned on the entire graph and instead use a set of specialized diffusion models $\{f_{R_0}, \ldots, f_{R_M}\}$, each trained on a relation type $R_i \in \mathcal{R}$. By composing these models based on the relations in $\mathcal{G}_0 = \{r_0, \ldots, r_N\}$, we jointly predict initial poses $\mathcal{P}_0$ for all blocks. However, since $\mathcal{G}_0$—extracted

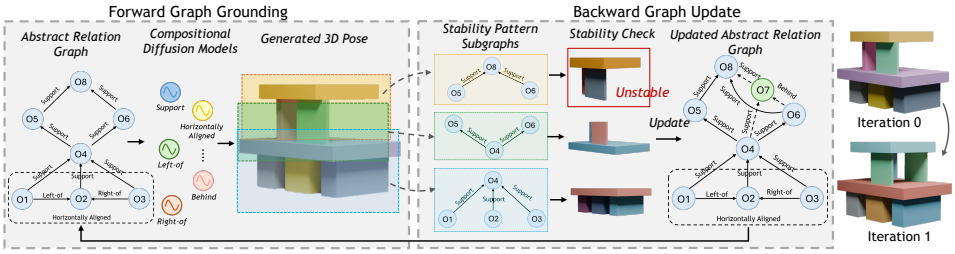


Figure 3: **Iterative graph grounding.** In the forward step (left), given a graph $\mathcal{G}_t$, compositional diffusion models generate 3D block poses. In the backward step (center-left), $\mathcal{G}_t$ is decomposed into subgraphs based on stability patterns, and each is checked via physics simulation. If unstable, new support blocks (green nodes) and relations are added. These updates are aggregated into an updated graph $\mathcal{G}_{t+1}$ for re-grounding. The right column shows the 3D structure before and after the graph update, with an added support ensuring stability.

from a front-view sketch—often omits hidden supports needed for stability, we introduce an iterative forward–backward grounding procedure (see Figure 3). This process alternates between generating poses (forward) and augmenting the graph with missing supports when instability is detected (backward). The remainder of this section first describes the pre-training of relation-specific diffusion models, then details the iterative grounding algorithm.

Training Individual Diffusion Models for Each Abstract Relation. We train a set of specialized diffusion models $\{f_{R_0}, \ldots, f_{R_M}\}$, where each f_R corresponds to an abstract relation $R \in \mathcal{R}_{\text{geom}} \cup \mathcal{R}_{\text{stab}}$. Each relation R is defined by a rule-based classifier h_R that verifies whether a tuple of block poses $\{p_1, \ldots, p_k\}$ satisfies the relation given their geometries $\{g_1, \ldots, g_k\}$. Its diffusion model f_R learns to generate such valid poses by sampling from $q_R(\boldsymbol{p} \mid \boldsymbol{g}) \propto \mathbb{1}[h_R(\boldsymbol{g}, \boldsymbol{p})]$, where $\boldsymbol{p}$ and $\boldsymbol{g}$ are vectorized pose and geometry inputs, and $\mathbb{1}[\cdot]$ is an indicator function enforcing relation satisfaction. To train each f_R, we generate synthetic data of multi-level 3D structures by composing multiple stability patterns. These structures exhibit diverse instances of both geometric and stability relations. We then apply the classifier h_R to extract positive examples for each relation and construct datasets $\mathcal{D}_R$ accordingly. For each R, we construct a denoising diffusion model f_R [38] where the distribution $q_R(\boldsymbol{p} \mid \boldsymbol{g})$ maximizes the likelihood of its dataset. Specifically, for a given sample $(\boldsymbol{g}, \boldsymbol{p}) \in \mathcal{D}_R$, we apply Gaussian noise to $\boldsymbol{p}$ across T time steps and train a denoising network $\epsilon_R(\boldsymbol{p}_t, \boldsymbol{g}, t)$ to recover the original poses. The training loss minimizes the following error:

$$\mathcal{L}_{\text{MSE}} = \mathbb{E}_{(\boldsymbol{g},\boldsymbol{p}),\boldsymbol{\epsilon},t}\left[\left|\boldsymbol{\epsilon} - \epsilon_R\left(\sqrt{\bar{\alpha}_t}\boldsymbol{p} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \boldsymbol{g}, t\right)\right|^2\right], \tag{1}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $t \sim \mathcal{U}(1, T)$, and $\bar{\alpha}_t$ is the diffusion denoising schedule. Sampling from f_R involves reversing the diffusion process using the learned denoiser. Starting from $\boldsymbol{p}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, we apply a learned reverse kernel parameterized by ϵ_R to iteratively obtain $\boldsymbol{p}_{t-1}$ from $\boldsymbol{p}_t$ until reaching $\boldsymbol{p}_0 \sim q_R(\cdot \mid \boldsymbol{g})$, a valid pose that satisfies h_R. See Appendix B for implementation details.

Iterative Pose Generation. We alternate between grounding a graph into 3D poses and updating the graph to recover missing supports. This forward–backward loop combines compositional diffusion models for pose generation with stability-pattern-guided heuristics for efficient graph update.

Forward Graph Grounding via Compositional Diffusion Models. Given a $\mathcal{G}$, we aim to sample valid 3D poses $\mathcal{P}$ that satisfy all relations in the graph. Each edge $r \in \mathcal{G}$ corresponds to a relation R applied to a specific block set and is associated with a diffusion model f_R that generates poses by approximating a conditional distribution $q_R(\boldsymbol{p}^r \mid \boldsymbol{g}^r)$ over the poses $\boldsymbol{p}^r$ with geometries $\boldsymbol{g}^r$. To predict poses for the entire graph $\mathcal{G}$, we compose these models into a product distribution: $q^t_{\text{prod}}(\boldsymbol{p} \mid \boldsymbol{g}) := \prod_{r\in\mathcal{G}} q^t_R(\boldsymbol{p}^r \mid \boldsymbol{g}^r)$, where each $q^t_R(\cdot)$ is the noised distribution at diffusion step t for the relation R associated with edge r. Our target is to sample from a **sequence** of product distributions $\{q^t_{\text{prod}}(\boldsymbol{p} \mid \boldsymbol{g})\}_{t=0:T}$ starting from an initial sample $\boldsymbol{p}_T$ drawn from $\mathcal{N}(\mathbf{0}, \mathbf{I})$. However, we cannot directly access the reverse diffusion kernel or exact score function for $q^t_{\text{prod}}(\boldsymbol{p} \mid \boldsymbol{g})$, since it aggregates multiple models [21, 18]. To approximate the transition from $q^t_{\text{prod}}(\boldsymbol{p} \mid \boldsymbol{g})$ to $q^{t-1}_{\text{prod}}(\boldsymbol{p} \mid \boldsymbol{g})$, we adopt a form of annealed MCMC that approximates the composite score function by summing the individual score functions from each model: $\nabla_{\boldsymbol{p}_t} \log q^t_{\text{prod}}(\boldsymbol{p}_t \mid \boldsymbol{g}) \approx \sum_{r\in\mathcal{G}} \epsilon_R(\boldsymbol{p}^r_t, \boldsymbol{g}^r, t)$, where ϵ_R is the learned denoising function for relation R. We apply the unadjusted Langevin algorithm (ULA) to

perform one reverse diffusion step using the composite score function over the graph $\mathcal{G}$:

$$\boldsymbol{p}_{t-1} = \boldsymbol{p}_t - A_t \sum_{r \in \mathcal{G}} \epsilon_R(\boldsymbol{p}_t^r, \boldsymbol{g}^r, t) + B_t \boldsymbol{\xi}, \quad \boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{2}$$

where A_t and B_t are constants from the diffusion schedule. At each noise level t, we run M iterations of ULA sampling using the composite score for the product distribution $q^t_{\text{prod}}(\cdot)$. Starting from $\boldsymbol{p}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, this iterative process gradually refines the sample, producing a final pose $\mathcal{P}$ drawn from $q^0_{\text{prod}}(\cdot)$ that satisfies all relations in $\mathcal{G}$.

Backward Graph Update Guided by Stability Patterns. The initial grounded poses $\mathcal{P}_0$ from $\mathcal{G}_0$ may be unstable due to missing hidden supports. Extending $\mathcal{G}_0$ efficiently is challenging, as the space of possible missing blocks and their placements is combinatorially large. To guide the search, we leverage the descriptors associated with each stability pattern $R \in \mathcal{R}_{\text{stab}}$: permissible object counts $(n_{\text{base}}, n_{\text{top}})_R$, *supported-by* subgraph $\mathcal{G}^R_{\text{supp}}$, and subgraphs of geometric relations among the base and top blocks, *i.e.*, $\mathcal{G}^R_{\text{base}}$ and $\mathcal{G}^R_{\text{top}}$. Since these descriptors constrain the search space and provide strong priors, we propose a heuristic search guided by stability pattern descriptors to predict missing blocks and relations. To repair an unstable subgraph, we add hidden blocks and relations to either (i) extend the subgraph within the same pattern by preserving descriptor validity, or (ii) evolve it into a more complex pattern that satisfies a richer set of descriptors. The types of added blocks $\mathcal{T}_{\text{hid}}$ are sampled heuristically from existing block types, conditioned on the newly added relations. For example, an unstable *single-block-stack* may indicate the top block overhangs the base. This can be resolved by converting it to a *cantilever-with-counterbalance* by changing the relation from *supported-by-fully* to *supported-by-partially*, or by transforming it into a *two-pillar-bridge* by adding a hidden support—of the same type as the base—behind the visible block. In both cases, the guidance of object counts and subgraph constraints from the pattern descriptors enables efficient search.

At each iteration t, we first decompose $(\mathcal{G}_t, \mathcal{P}_t)$ into subgraphs by matching against stability patterns. For each subgraph $(\mathcal{G}^i_t, \mathcal{P}^i_t)$, we simulate its stability. If unstable, we apply the stability-pattern-guided heuristic search to sample missing blocks and relations. All updates are aggregated into a new abstract relation graph $\mathcal{G}_{t+1} = \mathcal{G}_t \cup \mathcal{G}^{\text{hid}}_t$, which is then re-grounded using compositional diffusion models. This forward-backward refinement continues until a stable structure is found or the maximum number of iterations is reached.

5 Experiment

We evaluate StackItUp on 30 hand-drawn sketches illustrating a wide collection of 3D structures, ranging from iconic buildings (*e.g.*, Eiffel Tower), to modern house designs (see Appendix D for all sketches). Generation complexity is quantified by the total number of blocks and the spatial relations in the sketch. We evaluate performance using two metrics: (i) physical stability, measured as the fraction of blocks that remain in place under gravity in simulation; and (ii) resemblance, measured as the fraction of abstract geometric relations from the 2D sketch that are satisfied in the 3D generated structure from its front-view. Our evaluation tests three hypotheses: **H1**: Abstract relation graphs enable zero-shot generation of stable, multi-level 3D structures from sketches; **H2**: Stability-pattern-guided graph updates efficiently recover missing supports; and **H3**: Sketch-based input provides a more expressive design specification than natural language.

5.1 Baselines

We compare StackItUp against two baselines and an ablated variant (details in Appendix C):

End-to-End Diffusion Model. Following StackGen [33], we implement a single transformer-based diffusion model that directly predicts stable 3D block poses from sketches. The model is trained on the same synthetic dataset as our compositional models, using the extracted 2D sketches as inputs.

Direct VLM Prediction. Inspired by Blox-net [29], we use a vision-language model (VLM) to predict 3D structures. Given a sketch, we first prompt the VLM to generate a textual description of the scene, then use the description to produce a corresponding 3D block arrangement.

Ablation: No Hidden Object Prediction. This ablation removes the stability-pattern-guided graph update. It grounds the initial abstract relation graph iteratively using compositional diffusion models when instability is detected, with no addition of hidden supports at each iteration.

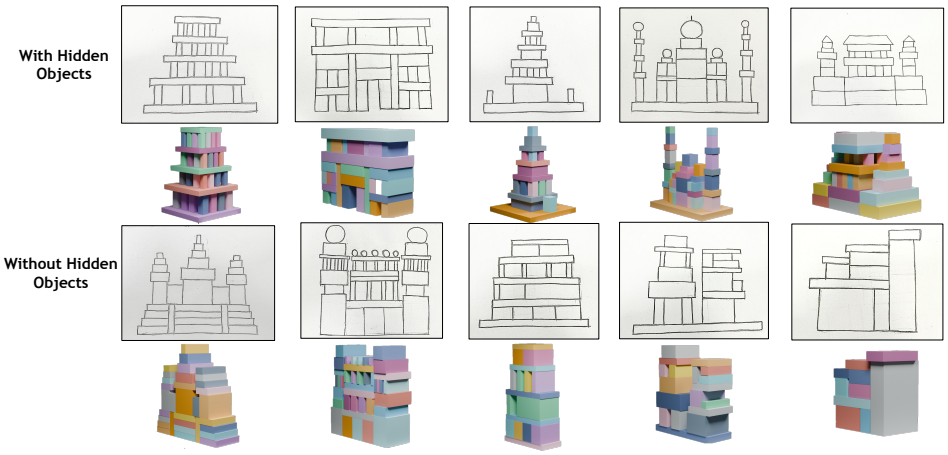


Figure 4: **Qualitative results.** 3D arrangements generated by StackItUp from 2D sketches. Top row: sketches where hidden supports are predicted for stability. Bottom row: sketches that require no hidden supports.

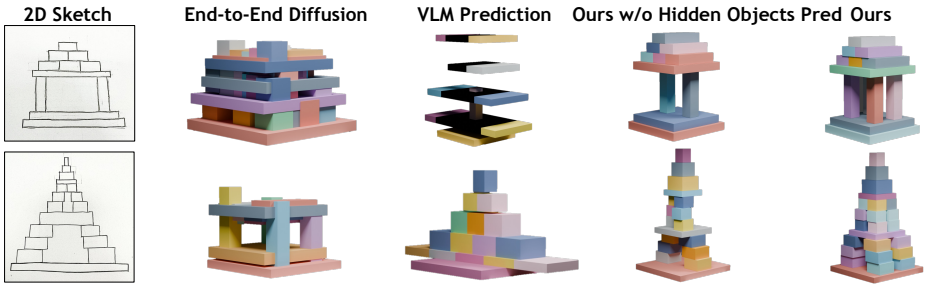


Figure 5: **Qualitative Comparison.**Comparison of 3D block arrangements generated from 2D sketches across methods. StackItUp consistently produces structures that are both visually faithful and physically stable.

5.2 Results

Table 2 summarizes the overall performance across all test cases. Figures 4 and 5 show example outputs from StackItUp and the baselines. StackItUp consistently outperforms all baselines in both physical stability and resemblance to the input sketch, whether or not hidden supports are needed. This validates **H1**, showing that abstract relation graphs enable effective zero-shot generation of stable, multi-level 3D structures from unseen sketches. In contrast, Direct-VLM-Prediction achieves reasonable stability but poor resemblance, highlighting the importance of precise spatial information in sketches for specifying 3D structures (**H3**). End-to-end diffusion model performs poorly on both metrics, likely due to the gap between synthetic training data and real-world sketches. Our ablation variant performs comparably to StackItUp when hidden supports are not required, but its performance degrades significantly when hidden supports are necessary, underscoring the importance of stability-pattern-guided graph updates and supporting **H2**.

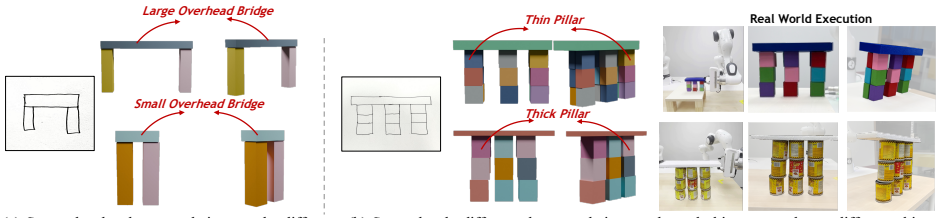
End-to-End Diffusion vs. StackItUp: Abstract Relation Graph Enables Zero-Shot Generalization (H1). Although both models are trained on the same synthetic dataset, StackItUp achieves significantly higher stability and resemblance on test sketches. The test sketches differ substantially from the synthetic training data, causing the end-to-end model to generalize poorly. In contrast, StackItUp demonstrates strong zero-shot generalization by leveraging the abstract relation graph as an intermediate representation, which (i) accurately captures the intended geometric relations from arbitrary sketches and (ii) flexibly composes relevant diffusion models at inference to preserve the user's design intent, thus validating **H1**.

Direct-VLM-Prediction vs. StackItUp: Sketches Better Capture User Intent (H3). While Direct-VLM-Prediction produces stable structures, its low resemblance scores indicate that language descriptions often fail to encode full spatial and geometric intent. In contrast, hand-drawn sketches naturally preserve key geometric and physical relationships that are difficult to express verbally, leading to more faithful reconstructions. This result supports **H3**.

Ablation vs. StackItUp: Stability-Pattern-Guided Updates Improve Robustness (H2). The ab-

Table 2: **Quantitative evaluation of different methods across sketches.** The reported percentages are the mean and standard deviation over 15 diverse test scenes; the best performance in each column is boldfaced.

Model	Sketch w/o Hidden Objects		Sketch with Hidden Objects	
	Resemblance (%)	Physical Stability (%)	Resemblance (%)	Physical Stability (%)
End-to-End Diffusion	18.2 $_{\pm 7.70}$	24.2$_{\pm 13.5}$	16.2$_{\pm 9.24}$	12.1$_{\pm 8.98}$
Direct VLM Prediction	33.1$_{\pm 22.6}$	63.6$_{\pm 26.8}$	30.2$_{\pm 26.7}$	53.7$_{\pm 35.5}$
Our Ablation (No Hidden Object Pred)	**84.4**$_{\pm 9.45}$	**98.8**$_{\pm 4.53}$	78.8$_{\pm 17.8}$	69.8$_{\pm 33.9}$
StackItUp (Ours)	**84.4**$_{\pm 9.45}$	**98.8**$_{\pm 4.53}$	**79.9**$_{\pm 15.0}$	**94.5**$_{\pm 15.3}$



(a) Same sketch, abstract relation graph, different object poses due to different object geometries.

(b) Same sketch, different abstract relation graphs and object poses due to different object geometries.

Figure 6: **Robustness of StackItUp.** StackItUp adapts the 3D poses to different block geometries. Given a sketch, it adjusts poses to accommodate geometric variations under the same $\mathcal{G}$ (a). If the structure is stable through pose adjustment alone, StackItUp extends $\mathcal{G}$ with hidden objects and re-grounds it to new poses (b).

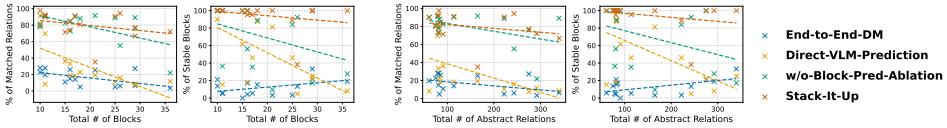


Figure 7: Performance *vs*. sketch complexity, measured by the number of blocks and abstract relations.

lation removes the stability-pattern-guided graph updates and simply re-optimizes the initial graph using diffusion models without inserting hidden supports. The resulting drop in stability for sketches requiring hidden structures demonstrates that stability-pattern-guided updates provide an efficient and effective heuristic for repairing unstable configurations, supporting **H2**.

5.3 Showcase of StackItUp's Robustness

Same Sketch, Different Object Sets: Adapting Object Poses to Satisfy Relations. Given the same sketch, StackItUp adapts object poses to satisfy the abstract relations even when the candidate block dimensions vary (Figure 6(a)). This flexibility comes from using the abstract relation graph, which composes diffusion models online to adjust poses according to the specific object geometries.

Same Sketch, Different Object Sets: Predicting Support Structures for Stability. When the candidate object set changes, StackItUp robustly predicts different hidden supports to maintain structural stability (Figure 6(b)). This is enabled by organizing support structures through stability patterns, which guide the addition of hidden supports when the predicted poses are unstable.

Robustness to Sketch Complexity. We show how resemblance and stability scores change against the number of blocks and spatial relations in Figure 7. As complexity increases, Direct-VLM-Prediction degrades sharply, showing the limitations of language-based specification. The end-to-end diffusion model also performs poorly due to the domain gap between real and synthetic sketches. In contrast, StackItUp consistently maintains high resemblance and stability, enabled by its abstract relation graph and compositional diffusion models. Unlike the ablation, whose stability declines with larger structures, StackItUp effectively predicts hidden supports, demonstrating strong generalization to complex, multi-level designs.

6 Conclusion

StackItUp transforms a rough 2D sketch into a stable 3D block arrangement for robot manipulation. By using an abstract relation graph that captures geometric and stability relations, StackItUp enables compositional 3D pose generation and efficient prediction of hidden supports. Experiments on diverse sketches show that StackItUp reliably produces stable, visually faithful structures, making sketch-based goal specification accessible to non-expert users.

7 Limitation

While StackItUp enables intuitive and robust 3D structure specification from sketches, several limitations point to promising directions for future work.

First, StackItUp currently relies on a single 2D front-view sketch, which limits observability of depth and occluded components. While this format is accessible to non-experts, extending the system to support multi-view sketches would improve accuracy. Our abstract relation graph provides a natural abstraction for this extension: integrating sketches from different views reduces to a graph-matching problem, where the key challenge is inferring correspondence between nodes across views. Second, we assume the block types in the sketch are labeled and that candidate object geometries are known. To make StackItUp deployable in real-world settings, we could relax these assumptions by integrating 3D perception techniques—such as single-view reconstruction or multi-view fusion—to estimate available object geometries. Then, relative proportions in the sketch can be used to infer likely block types. Finally, closing the loop with real-world robot feedback opens an exciting direction for interactive manipulation. Integrating visual or language-based corrections [39, 40, 41], human-in-the-loop refinement [42, 43, 44], or even learning a reward or cost function [45, 46, 47, 48, 49, 50, 51, 52] to discriminate and select among outcomes can make the system more responsive, adaptable, and effective in dynamic environments.

Acknowledgments

This research is supported by the National Research Foundation, Singapore, under its Medium Sized Centre Program, Center for Advanced Robotics Technology Innovation (CARTIN).

References

[1] T. Simeon, J. Cortes, A. Sahbani, and J. Laumond. A Manipulation Planner for Pick and Place Operations Under Continuous Grasps and Placements. In *IEEE Int. Conf. on Robotics & Automation*, 2002.

[2] A. Holladay, J. Barry, L. P. Kaelbling, and T. Lozano-Pérez. Object Placement as Inverse Motion Planning. In *IEEE Int. Conf. on Robotics & Automation*, 2013.

[3] F. Suárez-Ruiz and Q.-C. Pham. A Framework for Fine Robotic Assembly. In *IEEE Int. Conf. on Robotics & Automation*, 2016.

[4] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa. CHOMP: Covariant Hamiltonian Optimization for Motion Planning. *Int. J. Robotics Research*, 32, 2013.

[5] S. Karaman and E. Frazzoli. Sampling-Based Algorithms for Optimal Motion Planning. *Int. J. Robotics Research*, 30, 2011.

[6] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Trans. on Robotics and Automation*, 12, 1996.

[7] S. LaValle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. *Research Report 9811*, 1998.

[8] N. Č. Babič and D. Rebolj. Culture Change in Construction Industry: From 2D Toward BIM Based Construction. *Journal of Information Technology in Construction (ITcon)*, 21, 2016.

[9] C. A. Hunt. The Benefits of Using Building Information Modeling in Structural Engineering. https://digitalcommons.usu.edu/gradreports/319, 2013.

[10] Novel BIM. Problems of 2D Drawings in Construction Projects, 2023.

[11] Mars BIM. Traditional Drawings Vs BIM Technology: Why 2D Plans Are No Longer Enough, 2023.

[12] Tekla. Are You Harming Your Business by Sticking with 2D?, 2023.

[13] Shalin Designs. Importance of Structural 3D Modeling, 2023.

[14] B. Sen, A. Agarwal, G. Singh, B. B., S. Sridhar, and M. Krishna. SCARP: 3D Shape Completion in Arbitrary Poses for Improved Grasping. In *IEEE Int. Conf. on Robotics & Automation*, 2023.

[15] M. Z. Irshad, S. Zakharov, R. Ambrus, T. Kollar, Z. Kira, and A. Gaidon. ShAPO: Implicit Representations for Multi-Object Shape, Appearance, and Pose Optimization. In *Proc. European Conference on Computer Vision*, 2022.

[16] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling. FFRob: Leveraging Symbolic Planning for Efficient Task and Motion Planning. *Int. J. Robotics Research*, 37, 2018.

[17] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling. Sampling-Based Methods for Factored Task and Motion Planning. *Int. J. Robotics Research*, 37, 2018.

[18] Z. Yang, J. Mao, Y. Du, J. Wu, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling. Compositional Diffusion-Based Continuous Constraint Solvers. In *Conference on Robot Learning*, 2023.

[19] Y. Xu, J. Mao, Y. Du, T. Lozano-Pérez, L. P. Kaelbling, and D. Hsu. Set It Up!: Functional Object Arrangement with Compositional Generative Models. In *Proc. Robotics: Science & Systems*, 2024.

[20] Y. Zhu, J. Tremblay, S. Birchfield, and Y. Zhu. Hierarchical Planning for Long-Horizon Manipulation with Geometric and Symbolic Scene Graphs. In *IEEE Int. Conf. on Robotics & Automation*, 2021.

[21] Y. Du, C. Durkan, R. Strudel, J. B. Tenenbaum, S. Dieleman, R. Fergus, J. Sohl-Dickstein, A. Doucet, and W. Grathwohl. Reduce, Reuse, Recycle: Compositional Generation with Energy-Based Diffusion Models and MCMC. In *Proc. Int. Conf. on Machine Learning*, 2023.

[22] A. Sjöberg, J. Lindqvist, M. Önnheim, M. Jirstrand, and L. Svensson. MCMC-Correction of Score-Based Diffusion Models for Model Composition. *arXiv preprint arXiv:2307.14012*, 2023.

[23] C. M. Barber, R. J. Shucksmith, B. MacDonald, and B. C. Wünsche. Sketch-Based Robot Programming. In *IEEE Int. Conf. on Image and Vision Computing New Zealand*, 2010.

[24] D. Porfirio, L. Stegner, M. Cakmak, A. Sauppé, A. Albarghouthi, and B. Mutlu. Sketching Robot Programs on the Fly. In *ACM/IEEE Int. Conf. on Human-Robot Interaction*, 2023.

[25] P. Sundaresan, Q. Vuong, J. Gu, P. Xu, T. Xiao, S. Kirmani, T. Yu, M. Stark, A. Jain, K. Hausman, et al. RT-Sketch: Goal-Conditioned Imitation Learning From Hand-Drawn Sketches. In *Conference on Robot Learning*, 2024.

[26] Y. Xu, Y. Ng, Y. Wang, I. Sa, Y. Duan, Y. Li, P. Ji, and H. Li. Sketch2scene: Automatic generation of interactive 3d game scenes from user's casual sketches. *arXiv preprint arXiv:2408.04567*, 2024.

[27] Y. Cui, S. Niekum, A. Gupta, V. Kumar, and A. Rajeswaran. Can Foundation Models Perform Zero-Shot Task Specification for Robot Manipulation? In *Learning for Dynamics and Control Conference*, 2022.

[28] J. Gu, S. Kirmani, P. Wohlhart, Y. Lu, M. G. Arenas, K. Rao, W. Yu, C. Fu, K. Gopalakrishnan, Z. Xu, et al. Robotic Task Generalization Via Hindsight Trajectory Sketches. In *First Workshop on Out-of-Distribution Generalization in Robotics at CoRL 2023*, 2023.

[29] A. Goldberg, K. Kondap, T. Qiu, Z. Ma, L. Fu, J. Kerr, H. Huang, K. Chen, K. Fang, and K. Goldberg. Blox-Net: Generative Design-for-Robot-Assembly Using VLM Supervision, Physics Simulation, and a Robot with Reset. *arXiv preprint arXiv:2409.17126*, 2024.

[30] A. Badagabettu, S. S. Yarlagadda, and A. B. Farimani. Query2CAD: Generating CAD Models Using Natural Language Queries. *arXiv preprint arXiv:2406.00144*, 2024.

[31] Y. Hong, K. Zhang, J. Gu, S. Bi, Y. Zhou, D. Liu, F. Liu, K. Sunkavalli, T. Bui, and H. Tan. LRM: Large Reconstruction Model for Single Image to 3D. In *Int. Conf. on Learning Representations*, 2024.

[32] J. Tang, J. Ren, H. Zhou, Z. Liu, and G. Zeng. DreamGaussian: Generative Gaussian Splatting for Efficient 3D Content Creation. In *Int. Conf. on Learning Representations*, 2024.

[33] L. Sun, T. Yoneda, S. W. Wheeler, T. Jiang, and M. R. Walter. Stackgen: Generating Stable Structures From Silhouettes Via Diffusion. *arXiv preprint arXiv:2409.18098*, 2024.

[34] A. Liu, C. Lin, Y. Liu, X. Long, Z. Dou, H.-X. Guo, P. Luo, and W. Wang. Part123: Part-Aware 3D Reconstruction From a Single-View Image. In *ACM SIGGRAPH Conference Papers*, 2024.

[35] R. Liu, R. Wu, B. Van Hoorick, P. Tokmakov, S. Zakharov, and C. Vondrick. Zero-1-to-3: Zero-Shot One Image to 3D Object. In *IEEE Int. Conf. on Computer Vision*, 2023.

[36] M. Liu, C. Xu, H. Jin, L. Chen, M. Varma T, Z. Xu, and H. Su. One-2-3-45: Any Single Image to 3D Mesh in 45 Seconds Without Per-Shape Optimization. *Advances in Neural Information Processing Systems*, 2023.

[37] M. Liu, R. Shi, L. Chen, Z. Zhang, C. Xu, X. Wei, H. Chen, C. Zeng, J. Gu, and H. Su. One-2-3-45++: Fast Single Image to 3D Objects with Consistent Multi-View Generation and 3D Diffusion. In *IEEE Conf. on Computer Vision & Pattern Recognition*, 2024.

[38] J. Ho, A. Jain, and P. Abbeel. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, 2020.

[39] K. Lu, C. Ma, C. Hori, and D. Romeres. KitchenVLA: Iterative Vision-Language Corrections for Robotic Execution of Human Tasks. In *IEEE International Conference on Robotics and Automation Workshop on Safely Leveraging Vision-Language Foundation Models in Robotics (SafeLVMs@ICRA)*, May 2025. URL https://www.merl.com/publications/TR2025-068.

[40] Y. Zeng and Y. Xu. Learning reward for physical skills using large language model. *arXiv preprint arXiv:2310.14092*, 2023.

[41] P. Sharma, B. Sundaralingam, V. Blukis, C. Paxton, T. Hermans, A. Torralba, J. Andreas, and D. Fox. Correcting robot plans with natural language feedback. *arXiv preprint arXiv:2204.05186*, 2022.

[42] A. P. Dani, I. Salehi, G. Rotithor, D. Trombetta, and H. Ravichandar. Human-in-the-loop robot control for human-robot collaboration: Human intention estimation and safe trajectory tracking control for collaborative tasks. *IEEE Control Systems Magazine*, 40(6):29–56, 2020.

[43] P. Slade, C. Atkeson, J. M. Donelan, H. Houdijk, K. A. Ingraham, M. Kim, K. Kong, K. L. Poggensee, R. Riener, M. Steinert, et al. On human-in-the-loop optimization of human–robot interaction. *Nature*, 633(8031):779–788, 2024.

[44] C. Jiang, Y. Xu, and D. Hsu. Llms for robotic object disambiguation. *arXiv preprint arXiv:2401.03388*, 2024.

[45] M. Palan, N. C. Landolfi, G. Shevchuk, and D. Sadigh. Learning reward functions by integrating human demonstrations and preferences. *arXiv preprint arXiv:1906.08928*, 2019.

[46] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, 2017.

[47] Y. Xu and D. Hsu. How to tidy up a table: Fusing visual and semantic commonsense reasoning for robotic tasks with vague objectives. *arXiv preprint arXiv:2307.11319*, 2023.

[48] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. 2008.

[49] M. Wulfmeier, P. Ondruska, and I. Posner. Maximum entropy deep inverse reinforcement learning. In *Advances in Neural Information Processing Systems*, 2016.

[50] J. Fu, K. Luo, and S. Levine. Learning robust rewards with adversarial inverse reinforcement learning. In *Int. Conf. on Learning Representations*, 2018.

[51] Y. Xu, W. Gao, and D. Hsu. Receding horizon inverse reinforcement learning. In *Advances in Neural Information Processing Systems*, 2022.

[52] Y. Xu, F. Doshi-Velez, and D. Hsu. On the effective horizon of inverse reinforcement learning. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*, pages 2208–2216, 2025.

[53] P. Vincent. A Connection Between Score Matching and Denoising Autoencoders. *Neural Comput.*, 23(7):1661–1674, 2011.

[54] D. Coleman, I. Sucan, S. Chitta, and N. Correll. Reducing the barrier to entry of complex robotic software: a MoveIt! case study. *Journal of Software Engineering for Robotics*, 2014.

A Abstract Relation Library

The abstract relation library comprises a suite of qualitative geometric relations and local stability patterns used to compactly represent multi-layer block structures. Each relation is defined as a classifier h_R that examines block dimensions and relative poses to determine if a specific relation holds. This systematic encoding enables straightforward extraction of relation graphs from block arrangements and provides a foundation for generating synthetic data and training the class-conditional diffusion models f_R. Table 1 lists the symbolic names and arities of all relations. Below, we detail the implementation of each classifier, following the rules that also underlie our synthetic data generation.

A.1 Geometric Relations

We define 24 qualitative geometric relations in total. Of these, 13 are front-view (x–z plane) relations, such as *left-of* and *horizontally-aligned*, which can be computed directly from the bounding boxes in the sketch. The remaining 11 describe layout in the x–y (depth) plane (e.g., *front-of*, *depth-aligned*), and are only predicted once hidden (occluded) supports are inferred. For each relation, we specify its arity, associated plane, language description, and the explicit rule used by its classifier h_R.

Relation	**Arity**	**Plane**	**Description**	**Implementation of h_R**
left-of(o_A, o_B)	2	x-z	Block o_A is positioned entirely to the left of block o_B on the same support level; they do not overlap along the x-axis, but their front-to-back positions overlap substantially, meaning they are laterally offset and adjacent as seen from the front.	o_A.right_x $\leq o_B$.left_x + EPS; $\lvert$right_x $-$ left_x$\rvert <$ GAP; y-projections overlap $>$ ALPHA $\times$ min depth; same z-level
left-in(o, table)	2	x-z	Block o is located entirely to the left side of the table, with its rightmost side not crossing the table center.	o.right_x $<$ table.center_x
right-in(o, table)	2	x-z	Block o is located entirely to the right side of the table, with its leftmost side not crossing the table center.	o.left_x $>$ table.center_x
center-in(o, table)	2	x-z	Block o is centered on the table, having its center positioned at or very close to the table's origin.	$\lvert o$.center_x-table.center_x$\rvert <$ EPS and $\lvert o$.center_y-table.center_y$\rvert <$ EPS
supported-by-partially(o_A, o_B)	2	x-z	Block o_A sits on top of (is immediately above) block o_B but its base is only partially resting on o_B's top surface—so some but not all of its footprint is supported.	o_A.base_z $\approx o_B$.top_z (within EPS), o_A's xy footprint overlaps with but is not contained in o_B's
supported-by-fully(o_A, o_B)	2	x-z	Block o_A sits on top of (is immediately above) block o_B, and its entire base lies within (is fully supported by) o_B's top surface.	o_A.base_z $\approx o_B$.top_z (within EPS), o_A's xy footprint is fully contained in o_B's

horizontal-aligned(o_A, o_B)	2	x-z	Blocks o_A and o_B have the same front-back (y) coordinate—either at their centers or at matching front/back surfaces—indicating that they're horizontally aligned across the table (as seen from the front).	$\lvert\text{center\_y}_A - \text{center\_y}_B\rvert <$ EPS; or front/back surfaces match
vertical-aligned-centroid(o_A, o_B)	2	x-z	Block o_A is stacked directly above block o_B; both their x and y centroids align, so they form a straight vertical column.	o_A.base_z $\approx o_B$.top_z; $\lvert(x_A, y_A) - (x_B, y_B)\rvert <$ EPS
vertical-aligned-left(o_A, o_B)	2	x-z	Blocks o_A and o_B are precisely stacked so that their left (x) sides line up, and they overlap significantly along the y axis (as seen from above).	o_A.base_z $\approx o_B$.top_z; $\lvert\text{left_x}_A - \text{left_x}_B\rvert <$EPS; y-overlap >ALPHA$\times$depth
vertical-aligned-right(o_A, o_B)	2	x-z	Blocks o_A and o_B are stacked so their right (x) sides line up exactly, with significant y-overlap (depth).	o_A.base_z $\approx o_B$.top_z; $\lvert\text{right_x}_A - \text{right_x}_B\rvert <$EPS; y-overlap >ALPHA$\times$depth
horizontal-aligned-in-a-line($o_1, \ldots, o_n$)	n	x-z	Several blocks are arranged in a perfectly straight row (line) horizontally (left to right) with exactly matched y positions and equal z; typical for bridges/beams.	All $\lvert\text{center\_y}_i - \text{center\_y}_j\rvert <$ EPS; x positions ordered, spacing regular
touching-along-x(o_A, o_B)	2	x-z	Blocks o_A and o_B are side by side and touch exactly at their adjoining left/right faces, with a strong front-back (y) overlap.	$\lvert\text{right_x}_A - \text{left_x}_B\rvert <$EPS, y-overlap >ALPHA$\times$min depth
near-along-x(o_A, o_B)	2	x-z	Blocks o_A and o_B are on the same level and are placed close to each other side-by-side along the left-right (x) direction, but with a small gap (not touching). They overlap substantially in the front-back (y) direction, meaning they are nearly "neighbors" from the front view but not actually in contact with each other.	EPS $\leq$ gap_x $<$ D_NEAR; y-projection overlap $>$ ALPHA $\times$ min depth; same z-level

front-of(o_A, o_B)	2	x-y	Block o_A is positioned entirely in front of block o_B on the same level; their sides overlap along left/right, but o_A is closer to the front (observer), not overlapping with o_B in the y direction.	o_A.back_y $\leq o_B$.front_y + EPS; $\lvert$back_y $-$ front_y$\rvert <$ GAP; x-projections overlap $>$ BETA $\times$ min width; same z-level
front-in(o, table)	2	x-y	Block o is positioned entirely in front of the table's center (the $y = 0$ axis), meaning its entire back face is still in front of the table origin. The block lies between the observer and the center of the table, not straddling or exceeding the central axis.	o.back_y $<$ table.center_y
back-in(o, table)	2	x-y	Block o is located entirely behind the table's origin, with its frontmost point behind the $y = 0$ axis.	o.front_y $>$ table.center_y
touching-along-y(o_A, o_B)	2	x-y	Blocks o_A and o_B are placed side by side in the front-back (y) axis, so that one's back directly meets the other's front, with strong overlap along the left-right (x) axis; they "touch" along their y faces.	$\lvert$back_y $-$ front_y$\rvert <$ EPS; x-projection overlap $>$ ALPHA$\times$min width
near-along-y(o_A, o_B)	2	x-y	Blocks o_A and o_B are positioned nearly touching in the front-back (y) direction—separated by a small gap, but otherwise closely aligned, and substantially overlapping along the x axis.	EPS $\leq$ gap_y $<$ D_NEAR; x-projection overlap $>$ ALPHA$\times$min width
depth-aligned(o_A, o_B)	2	x-y	Blocks o_A and o_B have aligned depth (front-back) placements: their centers, or edges, in the left-right (x) axis coincide; often used for checking columnar or grid-like arrangements.	$\lvert$center_x$_A -$center_x$_B\rvert <$ EPS; or for left/right versions, $\lvert$left/right_x$_A -$left/right_x$_B\rvert <$ EPS
depth-aligned-in-a-line($o_1, \ldots, o_n$)	n	x-y	A group of n blocks, arranged in a straight line along the y (front-back) axis, each with the same x position (column formation), often with similar or equal spacing between centers.	All $\lvert$center_x$_i -$center_x$_j\rvert <$ EPS; y positions ordered, spacing regular

regular-grid-sparse(o_1, …, o_n)	n	x-y	A set of blocks forms a 2D grid in the x-y plane, where the left-right and front-back spacings between blocks are consistent but relatively wide—leaving space between adjacent blocks.	All blocks similar size, grouped into rows/columns by x/y; adjacent grid pairs are more than just touching (spacing>touch_eps), but rows/cols are regular
regular-grid-compact(o_1, …, o_n)	n	x-y	A set of blocks arranged in a closely packed, regular 2D grid, so that each block touches its neighbors both horizontally and vertically without gaps, and fills nearly all of the bounding rectangle.	All blocks similar size, assigned to rows/columns by x/y; all adjacent blocks touch (spacing $\leq$ touch_eps); grid fills $\geq$95% of bounding box
random-split-grid-sparse(o_1, …, o_n)	n	x-y	A group of blocks covers much—but not all—of a region in the x-y plane, forming a loosely connected configuration that is not fully regular, but no large gaps exist; may result from a random split/composition.	Sum of all block areas $\geq$90% of total bounding box; block positions irregular
random-split-grid-compact(o_1, …, o_n)	n	x-y	A group of blocks forms a compact area in the x-y plane, filling almost all available space but without the strict regularity of a grid.	Sum of all block areas $\geq$ 90% of bounding box; positions irregular, but very little wasted space

A.2 Stability Patterns

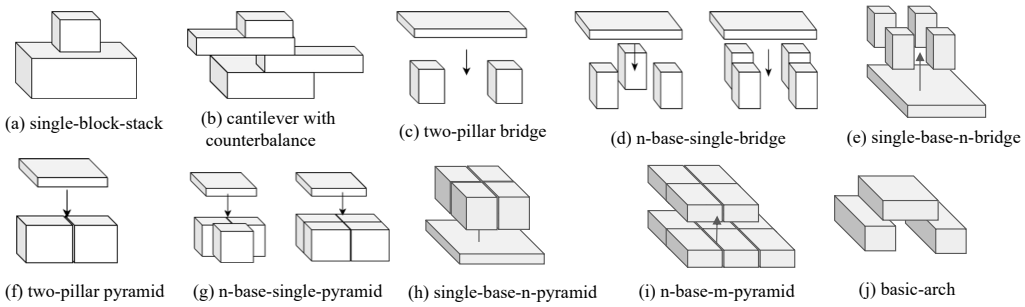


Figure 8: Illustration of the ten stability patterns.

We define 10 local stability patterns, each describing a recurring two-tier stable arrangement (e.g., pillar-bridge, stack). Each pattern's classifier h_R is characterized by four descriptors: (i) allowed counts of base and top blocks $(n_{\text{base}}, n_{\text{top}})$; (ii) admissible supported-by subgraphs $\mathcal{G}^R_{\text{supp}}$ connecting base and top blocks; (iii) required or allowed interrelations among the base blocks $\mathcal{G}^R_{\text{base}}$; and (iv) likewise among top blocks $\mathcal{G}^R_{\text{top}}$. These descriptors allow the automatic extraction of stability patterns from block arrangements, support straightforward synthetic data generation, and guide the compositional assembly of stable multi-level structures. By enumerating these patterns and their descriptors, we enable principled graph growth and hidden block insertion subject to stability constraints. Detailed specification for each stability classifier appears below.

Stability Pattern	Natural Language Description	Block Counts (base, top)	Subgraph Pattern Constraints
Notes on Subgraph Patterns: **(a) Supported-by subgraph**: Describes which "supported-by-fully" and "supported-by-partially" relations between base and top blocks must hold. **(b) Base geometric subgraph**: Specifies geometric layout constraints (touching, regular grid, separation, etc.) among base blocks. **(c) Top geometric subgraph**: Specifies geometric layout constraints among top blocks (when there are multiple).			
single-block-stack	A simple stack: one block sits fully above and is supported by another, forming a minimal stable column or pillar.	$(\{1\}, \{1\})$	(a) supported-by-fully(top, base) (b) none (c) none
cantilever-with-counterbalance	Two blocks stacked where the top block overhangs the base, but its center of mass remains safely above the base.	$(\{1\}, \{1\})$	(a) supported-by-partially(top, base) (COM of top within base) (b) none (c) none
two-pillar-single-top-bridge	Two upright, separated base blocks (pillars) jointly support a single horizontal top block (lintel), which bridges across them.	$(\{2\}, \{1\})$	(a) supported-by-fully(top, base_1) and supported-by-fully(top, base_2) (b) bases must not touch: not-touching($\text{base}_1, \text{base}_2$). They should be either horizontal-aligned or depth-aligned. (c) none
n-pillar-single-top-bridge	A single bridge block fully supported by n separated pillar blocks below, forming a "wide" bridge.	$(\{n\}, \{1\})$	(a) supported-by-fully(top, base_i) for all $i = 1..n$ (b) bases must not touch: all pairs not-touching($\text{base}_i, \text{base}_j$), $i \neq j$. The pillars should be arranged in regular-grid-sparse. (c) none
single-base-n-pillar-bridge	A single large base block with n separated vertical pillar blocks supported on top, each independent.	$(\{1\}, \{n\})$	(a) supported-by-fully(top_i, base) for all $i = 1..n$ (b) none (c) tops must not touch: all pairs not-touching($\text{top}_i, \text{top}_j$), $i \neq j$. The pillars should be arranged in regular-grid-sparse.

two-base-single-overhead-pyramid	Two base blocks, placed touching each other, jointly support a single overhead block centered above.	$(\{2\}, \{1\})$	(a) supported-by-fully(top, base$_1$) and supported-by-fully(top, base$_2$) (b) bases must touch along x or y: touching-along-x or touching-along-y (c) none
n-base-single-overhead-pyramid	n base blocks form a touching or tightly packed group (typically a regular grid), all together supporting a single overhead block.	$(\{n\}, \{1\})$	(a) supported-by-fully(top, base$_i$) for all $i = 1..n$ (b) bases form a regular grid or are all pairwise touching: regular-grid-compact or all pairwise touching (c) none
single-base-n-overhead-pyramid	A single base supports n top blocks closely packed together (usually in a regular row or grid), like a multi-top step of a pyramid.	$(\{1\}, \{n\})$	(a) supported-by-fully(top$_i$, base) for all i (b) none (c) tops form a regular grid or are all pairwise touching: regular-grid-compact$(top_1, ..., top_n)$ or all pairwise touching
n-base-m-overhead-pyramid	n base blocks in a compact/touching pattern collectively support m overhead blocks in a similarly compact arrangement; a multi-unit platform or tier.	$(\{n\}, \{m\})$	(a) supported-by-fully/partially(top$_j$, base$_i$) for each (i, j) where direct support exists (b) bases: regular grid or all touching, regular-grid-compact$(base_1, ..., base_n)$ (c) tops: regular grid or all touching, regular-grid-compact$(top_1, ..., top_m)$
basic-arc	Two separated base blocks act as pillars to partially support a top "keystone" block, often at an angle, forming the core of an arch.	$(\{2\}, \{1\})$	(a) supported-by-partially(top, base$_1$) and supported-by-partially(top, base$_2$) (b) bases must not touch: not-touching(base$_1$, base$_2$) (c) none

Table 4: Dictionary of local stability patterns. **Block Counts**: $(\{n\}, \{m\})$ means n base blocks stabilize m top blocks. **Subgraph Pattern Constraints**: (a) Required supported-by relations between base/top, (b) required geometric relations among base blocks, (c) required geometric relations among top blocks if $m > 1$.

B Training of Diffusion-based Pose Generators for Abstract Relations

For each distinct abstract relation in our library, we train a dedicated diffusion model to generate object poses that fulfill the specified spatial or stability constraint. Each model is tasked with predicting the denoising direction necessary to recover normalized object poses, ensuring the given structural relation holds.

Model Input and Data Preparation. To enable consistent learning, object geometries and poses are preprocessed and normalized: each object's 3D bounding box is scaled relative to the reference container region. The resulting input features comprise shape descriptors (width, height, depth, etc.)

Table 5: Comparison of methods and their user input mode, use of intermediate graphs, and how to search for hidden objects.

Method	User Input Mode	Intermediate Graph	Search for Hidden Objects
Direct VLM Prediction	Language Description	No	VLM informed search
End-to-end Diffusion Model	2D hand-drawn Sketch	No	N.A.
Our Ablation	2D hand-drawn Sketch	Yes	N.A.
Stack It Up (Ours)	**2D hand-drawn Sketch**	**Yes**	**Stability pattern guided search**

and normalized 3D pose coordinates. These, along with a diffusion timestep, form the input tuple for the network.

Modular Encoder Design. Our architecture integrates three specialized encoders:

- **Shape Encoder:** Implements a two-stage neural network, compressing raw geometry into a 256-dimensional latent representation using SiLU activations.
- **Pose Encoder:** With a configuration paralleling the shape encoder, this module transforms the normalized bounding box positions and dimensions into hidden feature space.
- **Temporal Encoder:** Timestep information is embedded via a sinusoidal encoder followed by linear and Mish-activated layers, mapping to and from a higher intermediate dimension to facilitate time-aware conditioning.

Backbone Architectures. The core relational reasoning is handled by one of two network backbones, selected based on relation arity:

- **MLP Backbone:** Fixed-arity relations employ a multi-layer perceptron that processes concatenated encodings, mapping directly to noise prediction in pose space. The MLP consists of linear and SiLU layers.
- **Transformer Backbone:** Variable-arity relations are addressed with a transformer-based network, which ingests sequences of object features (padded and masked as necessary, with positional encodings) for relations spanning multiple objects. The transformer output is post-processed and projected to the target pose distribution.

Pose Decoding and Reconstruction. The pose decoder reverses the encoding process, converting the hidden noise representations produced by the backbone into 3D pose refinements or direct pose predictions as appropriate for each object.

Learning Objective and Inference Procedure. The model is trained end-to-end, optimizing mean squared error (L2 loss) between the predicted and true denoising directions at each diffusion step. During inference, a cosine noise schedule over 1500 diffusion steps is applied. For scenarios involving multiple overlapping relations, noise estimates from individual relations are combined—either averaged or weighted—prior to decoding, allowing for joint enforcement of multiple spatial or stability constraints during the generation process.

C Baseline Implementation

We implemented two baselines and one ablated variant of StackItUp, as summarized in Table 5. The baselines differ in (i) user input modality, (ii) whether they use an intermediate abstract relation graph, and (iii) their ability to infer hidden supporting blocks. We provide implementation details for each below.

C.1 Direct VLM Prediction

The Direct VLM Prediction baseline adapts principles from Blox-Net [29], employing a vision-language model (VLM) to directly generate 3D block arrangements from natural language descrip-

tions. Unlike Blox-Net, this baseline does not use simulator-based filtering; instead, it relies on a single-pass VLM output. The purpose is to assess whether text alone can reliably convey structural intent. While we initially considered using human-written descriptions, their level of detail varied widely—from vague phrases to precise layouts—making comparisons inconsistent. To ensure consistency, we used VLM-generated descriptions, which are typically rich and precise, capturing layout, level count, and intra-level arrangements. This setup allows us to compare natural language as a specification modality against 2D sketches.

Given a hand-drawn sketch, the pipeline proceeds as follows:

1. Scene Description Generation: We prompt GPT-4.1 with the 2D sketch to produce a detailed, structured textual description, specifically requesting explicit statements of spatial relations and overall assembly appearance. The instruction emphasizes capturing all necessary details for faithfully reconstructing the depicted 3D structure from language alone. We use the following prompt:

Listing 1: Prompt used for scene description generation (Step 1).

```
You a given a front-view 2D rough hand-drawn sketch that illustrate a
    desired 3D mutli-level stacking structure (in its x-z plane) that
    are build from rectangle blocks. Generate a detail text
    description of this sketch, focusing on the relative spatial
    relations among the objects and the overall appearance, so that
    someone can generate the 3D structure by specifying the location
    of the blocks purely based on this textual description.
```

2. Block Type Selection: Using the generated scene description and a catalog of available block types (each with known dimensions), the VLM is asked to:

- Select a combination of block types and their quantities required to realize the described scene, including both visible and potentially hidden support blocks for stability.
- Briefly annotate each type's structural role within the assembly.

To facilitate this, block types and their dimensions are supplied in a structured dictionary format:

Listing 2: Format of candidate block dimensions into the VLM.

```
{
    "type_1_block": [w_1, l_1, h_1], # width (x), length (y), height (
        z)
    "type_2_block": [w_2, l_2, h_2],
    ...,
    "type_M_block": [w_M, l_M, h_M],
}.
```

Example answer fragment:

Listing 3: Example output of block selection via VLM.

```
- 2 x type_3_block: act as the base platform
- 1 x type_7_block: forms the central vertical column
- ...
```

Prompt:

Listing 4: Prompt used for block selection (Step 2).

```
You are now the structural planner.

Inputs

Textual scene description (from Step 1) - enclosed in <SCENE> ... </
    SCENE>.
```

```
Catalogue of blocks with their exact dimensions - enclosed in <
    CATALOGUE> ... </CATALOGUE>. The dictionary format is {"
    type_1_block": [w_1, l_1, h_1], ... } where w = width (x-axis), l
    = length (y-axis), h = height (z-axis).

Task
Select a set of block types and quantities that can realise the scene
    while obeying these rules:

- Use only block types listed in the catalogue; no scaling or custom
    sizes.
- Infer the smallest sufficient number of blocks; you may add hidden
    support blocks if the scene requires stability.
- Match the relative proportions described in <SCENE>.
- Ignore absolute coordinates-those will be assigned in Step 3.

Output format  (return nothing else)

BLOCK_SELECTION = [
{"type": "type_k_block", "count": N, "role": "one-sentence purpose" },
...
]

Example
BLOCK_SELECTION = [
{ "type": "type_3_block", "count": 2, "role": "forms the ground-level
    platform" },
{ "type": "type_7_block", "count": 1, "role": "serves as the central
    vertical column" }
]

Note that "type_0_block" is the base that is not drawn in the sketch.
    Always select it.

Begin.
```

3. Block Placement Prediction: Finally, the VLM assigns 3D coordinates to every block instance, respecting the following constraints:

- The entire assembly must fit within a fixed bounding box.
- Block instances are axis-aligned and individually identified.
- No two blocks occupy the same space (collisions are disallowed).
- Support and stability are enforced by allowing the VLM to introduce hidden blocks, as needed.
- The resulting output is a mapping between each block (with type) and its 3D centroid.

The centroid definition, bounding box, and other spatial conventions follow those of our main model for comparability.

Required output format:

Listing 5: Pose prediction output format via VLM.

```
{
0: { "type": "type_3_block", "centroid": [-1.2,  0.8, 0.25] },
1: { "type": "type_3_block", "centroid": [ 1.2,  0.8, 0.25] },
2: { "type": "type_7_block", "centroid": [ 0.0,  0.0, 1.75] },
...
}
```

Prompt used:

Listing 6: Prompt used for block pose generation (Step 3).

```
You are now the placement engine.
Your job is to assign exact 3D positions to every physical block
    selected in Step 2.

INPUTS

Scene description <SCENE> ... </SCENE>   (optional - use for spatial
    cues)
Block selection list <SELECTION> ... </SELECTION>   created in Step 2
    Example: [ { "type": "type_3_block", "count": 2, "role": "ground-
    level platform" }, { "type": "type_7_block", "count": 1, "role": "
    central column" } ]
Block catalogue with dimensions <CATALOGUE> ... </CATALOGUE> Format: {
     "type_i_block": [w_i, l_i, h_i], ... } w = width (x-axis), l =
    length (y-axis), h = height (z-axis)
GLOBAL CONSTRAINTS
- The whole assembly must fit inside the bounding box
x \in [-1.5,  1.5]   (width < 3)
y \in [-1.0,  1.0]   (length < 2)
z \in [ 0.0,  5.0]   (height < 5)
- Blocks are axis-aligned; do not rotate them.
- No two blocks may overlap (touching faces/edges is allowed).
- Every block above ground level must rest on, or be supported by,
    blocks beneath it; hidden support blocks from <SELECTION> may be
    used.
- Centroids are expressed in the same linear units as the catalogue.

TASK
For every physical block instance:

Assign a unique integer id starting at 0.
Specify its block type (exact key from the catalogue).
Output the centroid coordinates [x, y, z].
OUTPUT FORMAT  (return nothing else)

PLACEMENT = {
0: { "type": "type_3_block", "centroid": [-1.2,  0.8, 0.25] },
1: { "type": "type_3_block", "centroid": [ 1.2,  0.8, 0.25] },
2: { "type": "type_7_block", "centroid": [ 0.0,  0.0, 1.75] },
...
}

Note that "type\_0\_block" is the base that is not drawn in the sketch
    . Always select it and assigns its centroid to [0, 0, -0.05].

Begin.
```

The second baseline follows the end-to-end paradigm of StackGen [33], training a single transformer-based diffusion model to directly predict the full set of 3D block poses conditioned on sketch input, without any explicit intermediate graph representations.

This baseline involves two steps:

1. **Block Type Selection:** A small convolutional neural network (CNN) is trained to predict the set and count of block types required, given the input sketch and candidate block dimensions.
2. **Block Pose Regression:** A large transformer-based diffusion model generates the 3D poses (centroids) for all selected blocks, conditioned on the sketch and selected types.

The CNN encoder transforms the input sketch into a compact latent embedding by passing it through several convolutional layers, each followed by batch normalization and ReLU activations, and finally

by spatial pooling and a linear layer. This embedding is added to each object's feature representation before being processed by the transformer to promote cross-block information sharing and contextualization.

Within the diffusion model, sketch embeddings are broadcast and added to the positional and geometric embeddings of each block, and position encoding is applied. Batch data is padded and appropriately masked to handle varying object counts per sample.

Training and Data: Both models are trained on synthetic data generated by composing multiple local stability patterns. However, only the 3D block arrangements and dimensions are available initially. To create paired sketch inputs, we generate a 2D front-view sketch for each 3D structure.

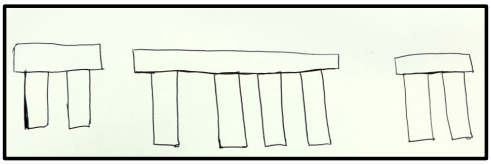
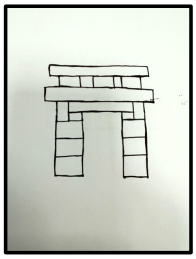
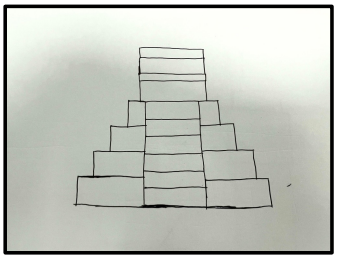
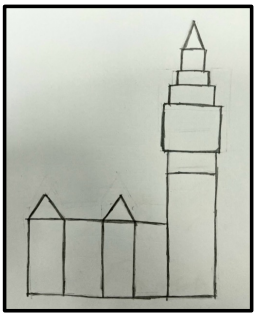
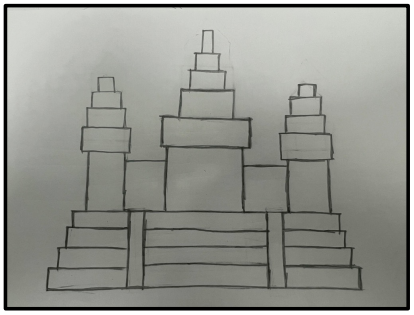
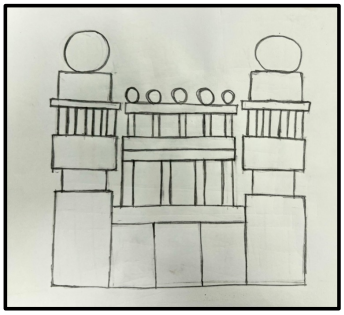
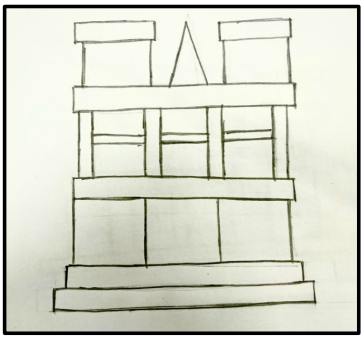
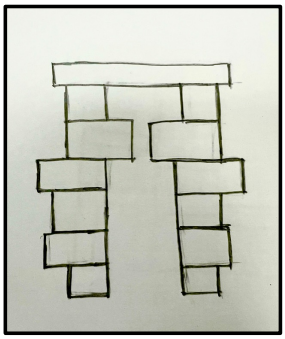
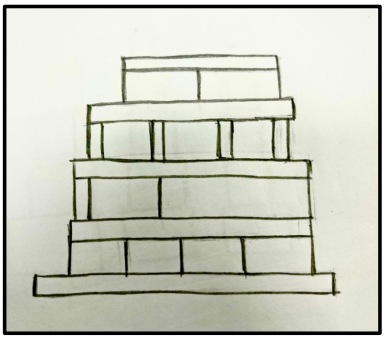
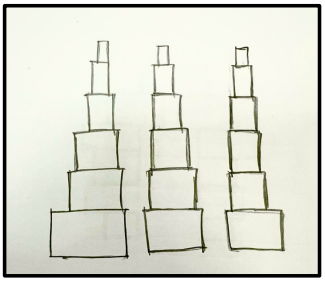
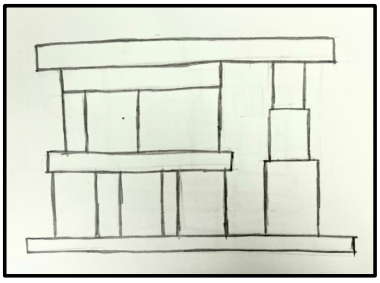
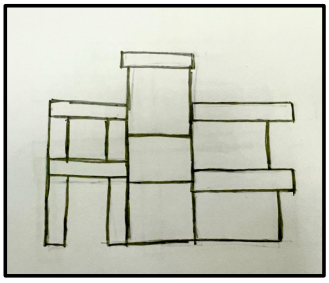
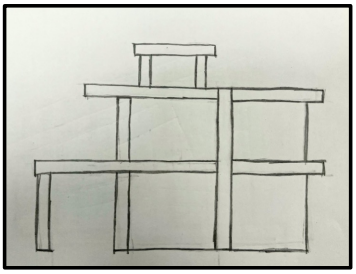
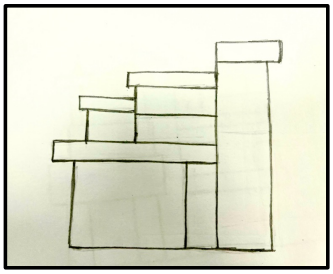
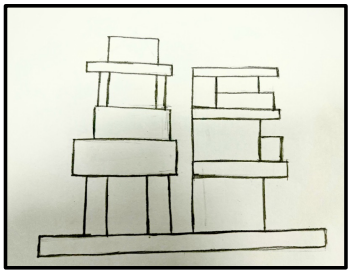
For each sample, we project the 3D arrangement to the x-z (front-view) plane. We retain only the visible blocks, sorting them by their y-coordinates (depth), and filtering out those fully occluded by others. The outlines of visible blocks are rendered onto a blank canvas, and their edges are dilated and blurred to resemble hand-drawn sketches and reduce the domain gap. Padding is added to maintain consistent framing. Figure 9 illustrates representative examples of these generated sketches.

C.2 Our Ablation: No Hidden Object Prediction

To assess the importance of hidden support prediction, we ablate the stability-pattern-guided backward graph update in StackItUp. In this variant, the relation graph representing the 3D structure is not expanded with additional hidden support blocks. Instead, when an arrangement generated from the abstract relation graph is found to be unstable in physical simulation, we re-ground the same graph using our compositional diffusion models and attempt pose adjustments using only the initially specified blocks. This allows us to study whether iterative re-sampling alone is sufficient to achieve global stability or if explicit reasoning over hidden supports is necessary.

D Hand-drawn Test Cases

We evaluate all methods on a set of 30 hand-drawn sketches spanning a range of stacking challenges. The complete set of test cases is shown below, across two pages.

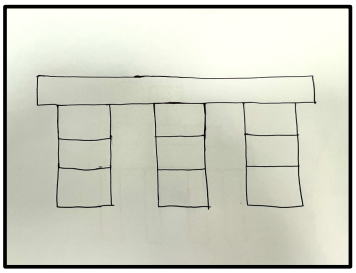
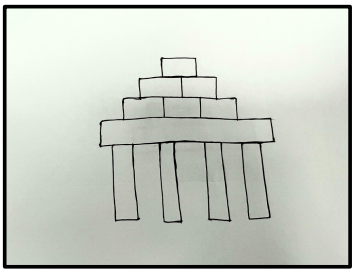
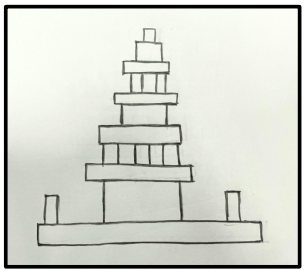
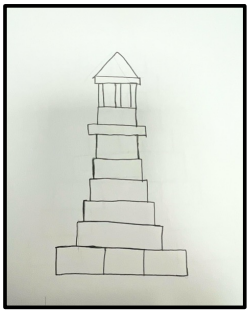
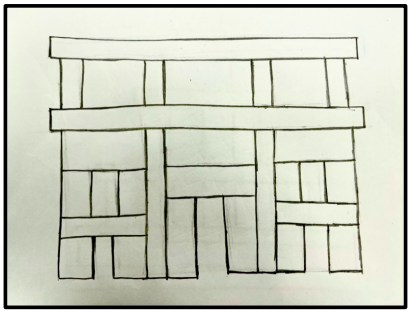
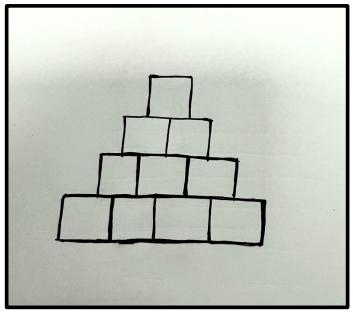
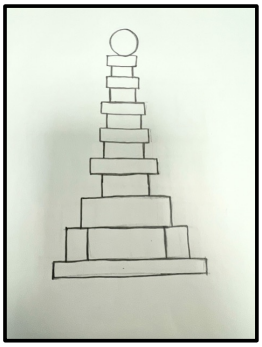
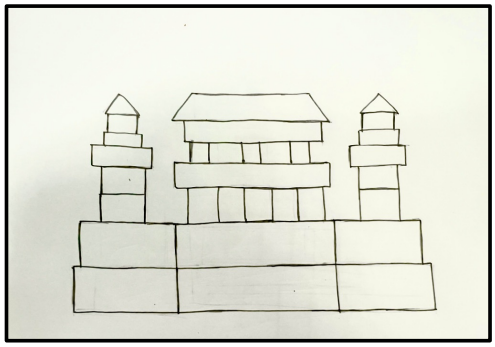
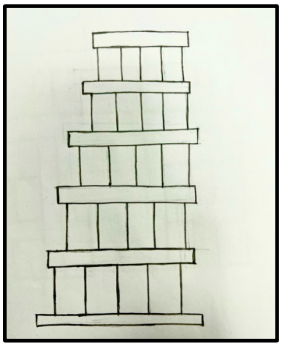
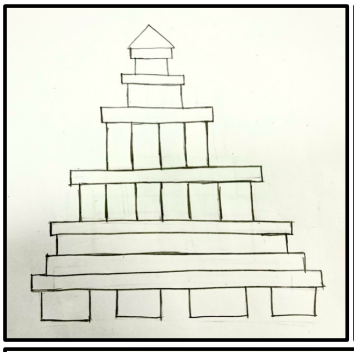
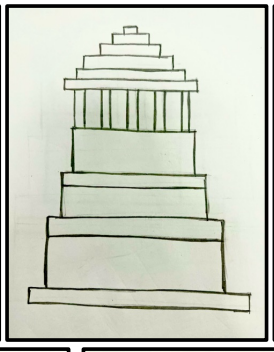
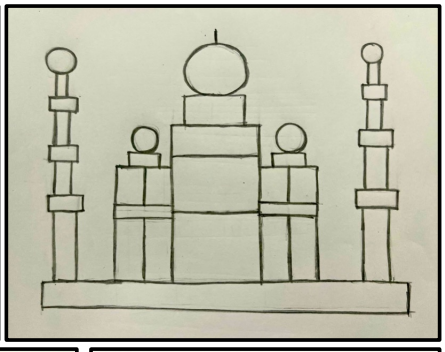
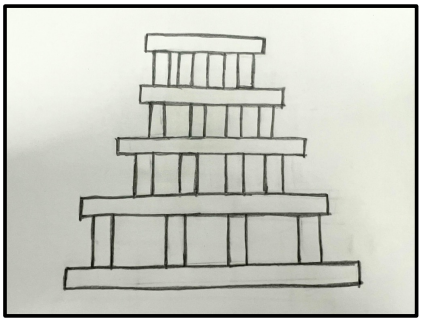
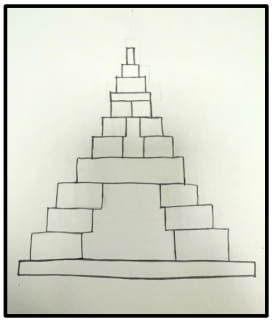
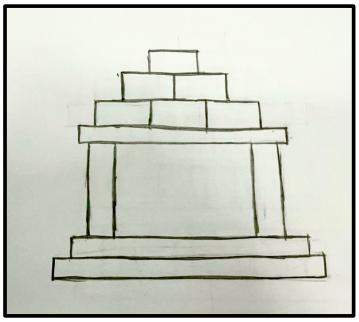
















Figure 10: **Hand-drawn test cases.** The full set of 30 human-drawn 2D sketches used for evaluation, presented over two pages (15 per page). The test cases cover a variety of multi-level stacking scenarios and structural challenges.

E Joint Pose Prediction with Composite Diffusion Scores and ULA

To jointly satisfy multiple abstract relations among objects, we combine the score functions from several trained diffusion models and perform inference using a composite score. This approach enables simultaneous pose generation that respects all specified abstract relations.

Diffusion Reverse Step as Score-Based Sampling. A single reverse step of a diffusion model at noise level t updates the input x_t by

$$x_{t+1} = x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}t}}\epsilon\theta(x_t, t) + \beta_t\xi, \quad \xi \sim \mathcal{N}(0, I),$$

where ϵ_θ denotes the neural network's noise prediction, β_t is the step's noise parameter, and ξ is standard Gaussian noise. Notably, the quantity $\frac{\epsilon_\theta(x_t,t)}{\sqrt{1-\bar{\alpha}t}}$ is, by denoising score matching theory [53], an explicit estimator of the gradient of the log-density for the perturbed data distribution $p_t(x)$ at time t:

$$\nabla_x \log q_t(x) \simeq \frac{\epsilon\theta(x_t, t)}{\sqrt{1-\bar{\alpha}t}}. \tag{3}$$

Thus, the reverse step can be rewritten in Langevin form:

$$x_{t+1} = x_t - \beta_t\nabla_x \log q_t(x) + \beta_t\xi. \tag{4}$$

Connection to ULA. The unadjusted Langevin algorithm (ULA) samples from $q_t(x)$ according to

$$x_{t+1} = x_t - \eta\nabla_x \log q_t(x) + \sqrt{2\eta}\xi, \quad \xi \sim \mathcal{N}(0, I), \tag{5}$$

for step size η. If we set $\eta = \beta_t$, this becomes

$$x_{t+1} = x_t - \beta_t\nabla_x \log q_t(x) + \sqrt{2}\beta_t\xi. \tag{6}$$

The only difference from the diffusion reverse update is a multiplicative factor of $\sqrt{2}$ in the noise term. Consequently, running the reverse diffusion process is equivalent to a ULA sampler with a reduced noise temperature; one can recover exact ULA sampling by scaling the variance of the added noise by a factor of 2 at each step, or equivalently scaling the standard deviation by $\sqrt{2}$.

Composing Scores from Multiple Relations. When enforcing multiple spatial relations jointly, we aggregate (e.g., sum) the individual score estimates from relevant diffusion models at each step to create a composite score:

$$\nabla_x \log q^t_{\text{prod}}(x) \simeq \sum_{r\in\mathcal{G}} w_r\nabla_x \log q^t_r(x), \tag{7}$$

where q^t_r is the noisy distribution associated with relation r, and w_r are optional weights. We then perform sampling updates using this composite gradient, applying ULA theory as above.

Implementation in Practice. In our experiments, joint ULA sampling is implemented by replacing the noise term in the standard reverse diffusion step with one scaled by $\sqrt{2}$, and substituting the composite score for the individual model score. Alternatively, if using the original (diffusion) noise schedule, the resulting samples correspond to a lower-temperature (less stochastic) variant of the fully tempered ULA trajectory.

F Real Robot Execution of Planned Poses

We conduct our real-robot experiments using a Franka Research 3 (FR3) robotic arm. The robot performs motion planning to reach target gripper poses while avoiding collisions using MoveIt! [54].

We assume that StackItUp's output, $\mathcal{O} = \{o_1, \ldots, o_M\}$, is sorted such that lower-level objects have smaller indices. Each object $o_i = (\tau_i, p_i = (x_i, y_i, z_i))$ has a goal pose $\mathbf{H}_i^{\text{goal}}$ with identity orientation:

$$\mathbf{H}_i^{\text{goal}} = \begin{bmatrix} 1 & 0 & 0 & x_i \\ 0 & 1 & 0 & y_i \\ 0 & 0 & 1 & z_i \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{8}$$

We assume the initial pose $\mathbf{H}_i^{\text{init}}$ of each object o_i is known. In our setup, we use an L-shaped bracket with a known pose: by aligning the object with the bracket on a flat surface, its pose can be determined. Alternatively, vision-based methods can be used to estimate object poses. In practice, our current setup simplifies deployment by assuming known block types, predefined grasp poses, and an L-block calibration to fix initial poses, which reduces planning uncertainty. A natural extension is a fully vision-based pipeline: 3D perception could recover block geometry from point clouds and generalize to unseen shapes, though this would require robust closed-loop control to mitigate compounding uncertainty.

For each object type τ', we define a relative grasp pose $\mathbf{H}_{\tau'}^{\text{grasp}}$ between the gripper frame and the object frame. We also specify a sequence of approach poses $\mathcal{H}_{\tau'}^{\text{pick}}$ for picking (*e.g.*, moving the gripper above the object, then descending to align with it). To pick up an object $o_i = (\tau_i, p_i)$, the robot executes the sequence of poses $\mathbf{H}_i^{\text{init}} \mathbf{H}_{\tau_i}^{\text{grasp}} \mathbf{H}$ for each $\mathbf{H} \in \mathcal{H}_{\tau_i}^{\text{pick}}$, followed by closing the gripper. For placement, we define pre-placement and post-placement pose sequences, $\mathcal{H}_{\tau'}^{\text{pre-place}}$ and $\mathcal{H}^{\text{post-place}}$, respectively, which are executed before and after opening the gripper.

To assemble the target configuration $\mathcal{O}$, the robot performs the pick-and-place routine sequentially for each object.

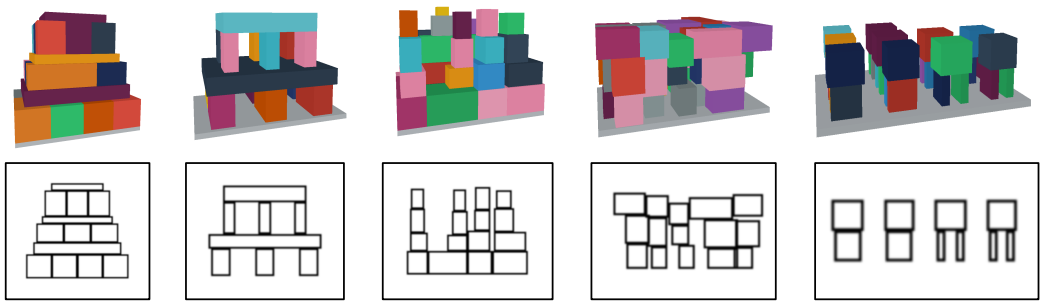


Figure 9: **Illustrative examples of sketch conversion.** For each synthetic 3D block structure (top), we extract the visible blocks from the front-view (x-z plane), and render their outlines as a 2D sketch (bottom). To better resemble human-drawn sketches, we further apply edge dilation and Gaussian blur. These converted sketches serve as input representations for our end-to-end diffusion model baseline.