

395 A Abstract Relation Library

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

404 A.1 Geometric Relations

405 We define 24 qualitative geometric relations in total. Of these, 13 are front-view (x - z plane) rela-
 406 tions, such as *left-of* and *horizontally-aligned*, which can be computed directly from the bounding
 407 boxes in the sketch. The remaining 11 describe layout in the x - y (depth) plane (e.g., *front-of*, *depth-*
 408 *aligned*), and are only predicted once hidden (occluded) supports are inferred. For each relation, we
 409 specify its arity, associated plane, language description, and the explicit rule used by its classifier
 410 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.	$o_A.\text{right_x} \leq o_B.\text{left_x} + \text{EPS}$; $ \text{right_x} - \text{left_x} < \text{GAP}$; y -projections overlap $> \text{ALPHA} \times \text{min depth}$; 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.	$o.\text{right_x} < \text{table.center_x}$
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.	$o.\text{left_x} > \text{table.center_x}$
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.	$ o.\text{center_x} - \text{table.center_x} < \text{EPS}$ and $ o.\text{center_y} - \text{table.center_y} < \text{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.	$o_A.\text{base_z} \approx o_B.\text{top_z}$ (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.	$o_A.\text{base_z} \approx o_B.\text{top_z}$ (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).	$ \text{center_y}_A - \text{center_y}_B < \text{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.	$o_A.\text{base_z} \approx o_B.\text{top_z}$; $ (x_A, y_A) - (x_B, y_B) < \text{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).	$o_A.\text{base_z} \approx o_B.\text{top_z}$; $ \text{left_x}_A - \text{left_x}_B < \text{EPS}$; $y\text{-overlap} > \text{ALPHA} \times \text{depth}$
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).	$o_A.\text{base_z} \approx o_B.\text{top_z}$; $ \text{right_x}_A - \text{right_x}_B < \text{EPS}$; $y\text{-overlap} > \text{ALPHA} \times \text{depth}$
horizontal-aligned-in-a-line(o_1, \dots, 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 $ \text{center_y}_i - \text{center_y}_j < \text{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.	$ \text{right_x}_A - \text{left_x}_B < \text{EPS}$, $y\text{-overlap} > \text{ALPHA} \times \text{min depth}$
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.	$\text{EPS} \leq \text{gap_x} < \text{D_NEAR}$; $y\text{-projection overlap} > \text{ALPHA} \times \text{min depth}$; 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.	$o_A.\text{back_y} \leq o_B.\text{front_y} + \text{EPS}$; $ \text{back_y} - \text{front_y} < \text{GAP}$; x -projections overlap $> \text{BETA} \times \text{min width}$; 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.	$o.\text{back_y} < \text{table.center_y}$
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.	$o.\text{front_y} > \text{table.center_y}$
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.	$ \text{back_y} - \text{front_y} < \text{EPS}$; x -projection overlap $> \text{ALPHA} \times \text{min width}$
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.	$\text{EPS} \leq \text{gap_y} < \text{D.NEAR}$; x -projection overlap $> \text{ALPHA} \times \text{min width}$
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.	$ \text{center_x}_A - \text{center_x}_B < \text{EPS}$; or for left/right versions, $ \text{left/right_x}_A - \text{left/right_x}_B < \text{EPS}$
depth- aligned-in-a- line(o_1, \dots , 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 $ \text{center_x}_i - \text{center_x}_j < \text{EPS}$; y positions ordered, spacing regular

regular-grid-sparse(o_1, \dots, 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 ($\text{spacing} > \text{touch_eps}$), but rows/cols are regular
regular-grid-compact(o_1, \dots, 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 ($\text{spacing} \leq \text{touch_eps}$); grid fills $\geq 95\%$ of bounding box
random-split-grid-sparse(o_1, \dots, 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, \dots, 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

411 A.2 Stability Patterns

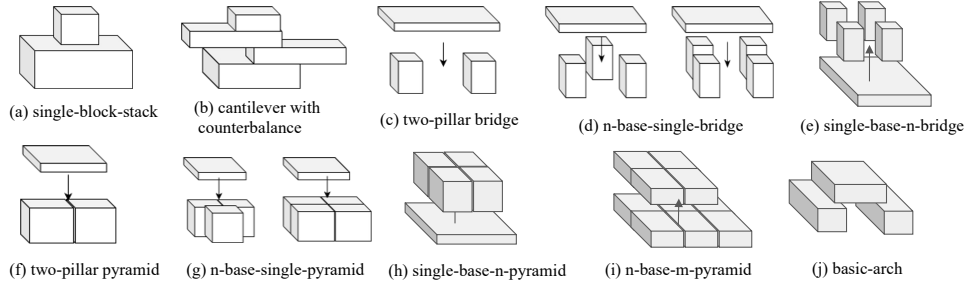


Figure 8: Illustration of the 10 stability patterns.

412 We define 10 local stability patterns, each describing a recurring two-tier stable arrangement (e.g.,
413 pillar-bridge, stack). Each pattern’s classifier h_R is characterized by four descriptors: (i) allowed
414 counts of base and top blocks ($n_{\text{base}}, n_{\text{top}}$); (ii) admissible supported-by subgraphs $\mathcal{G}_{\text{supp}}^R$ connect-
415 ing base and top blocks; (iii) required or allowed interrelations among the base blocks $\mathcal{G}_{\text{base}}^R$; and
416 (iv) likewise among top blocks $\mathcal{G}_{\text{top}}^R$. These descriptors allow the automatic extraction of stability
417 patterns from block arrangements, support straightforward synthetic data generation, and guide the
418 compositional assembly of stable multi-level structures. By enumerating these patterns and their
419 descriptors, we enable principled graph growth and hidden block insertion subject to stability con-
420 straints. 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(base ₁ , 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(base _{i} , 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(top _{i} , 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 $m > 1$.

421 B Training of Diffusion-based Pose Generators for Abstract Relations

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

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

Table 5: Method Comparison

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 [28], employing a vision-language model (VLM) to generate 3D block arrangements from natural language descriptions. This setup enables us to compare natural language as a specification modality against 2D sketches.

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

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

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

```
471 You are given a front-view 2D rough hand-drawn sketch that illustrate a
472 desired 3D multi-level stacking structure (in its x-z plane) that
473 are build from rectangle blocks. Generate a detail text
474 description of this sketch, focusing on the relative spatial
475 relations among the objects and the overall appearance, so that
476 someone can generate the 3D structure by specifying the location
477 of the blocks purely based on this textual description.
478
```

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

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

485 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.

```
486 {
487   "type_1_block": [w_1, l_1, h_1], # width (x), length (y), height (
488     z)
489   "type_2_block": [w_2, l_2, h_2],
490   ...,
491   "type_M_block": [w_M, l_M, h_M],
492 }.
```

495 Example answer fragment:

Listing 3: Example output of block selection via VLM.

```
496 - 2 x type_3_block: act as the base platform
497 - 1 x type_7_block: forms the central vertical column
498 - ...
499
```

501 Prompt:

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

```
502 You are now the structural planner.
503
504 Inputs
505
506 Textual scene description (from Step 1) - enclosed in <SCENE> ... </
507 SCENE>.
508 Catalogue of blocks with their exact dimensions - enclosed in <
509 CATALOGUE> ... </CATALOGUE>. The dictionary format is {"
510   type_1_block": [w_1, l_1, h_1], ... } where w = width (x-axis), l
511   = length (y-axis), h = height (z-axis).
512
513 Task
514 Select a set of block types and quantities that can realise the scene
515 while obeying these rules:
516
517
```



```

518 - Use only block types listed in the catalogue; no scaling or custom
519 sizes.
520 - Infer the smallest sufficient number of blocks; you may add hidden
521 support blocks if the scene requires stability.
522 - Match the relative proportions described in <SCENE>.
523 - Ignore absolute coordinates-those will be assigned in Step 3.
524
525 Output format (return nothing else)
526
527 BLOCK_SELECTION = [
528 {"type": "type_k_block", "count": N, "role": "one-sentence purpose" },
529 ...
530 ]
531
532 Example
533 BLOCK_SELECTION = [
534 { "type": "type_3_block", "count": 2, "role": "forms the ground-level
535 platform" },
536 { "type": "type_7_block", "count": 1, "role": "serves as the central
537 vertical column" }
538 ]
539
540 Note that "type_0_block" is the base that is not drawn in the sketch.
541 Always select it.
542
543 Begin.
544

```

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

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

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

555

556 Required output format:

Listing 5: Pose prediction output format via VLM.

```

557 {
558 0: { "type": "type_3_block", "centroid": [-1.2, 0.8, 0.25] },
559 1: { "type": "type_3_block", "centroid": [ 1.2, 0.8, 0.25] },
560 2: { "type": "type_7_block", "centroid": [ 0.0, 0.0, 1.75] },
561 ...
562 }
563

```

565 Prompt used:

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

```

566 You are now the placement engine.
567 Your job is to assign exact 3D positions to every physical block
568 selected in Step 2.
569
570 INPUTS
571
572

```

```

573 Scene description <SCENE> ... </SCENE> (optional - use for spatial
574 cues)
575 Block selection list <SELECTION> ... </SELECTION> created in Step 2
576 Example: [ { "type": "type_3_block", "count": 2, "role": "ground-
577 level platform" }, { "type": "type_7_block", "count": 1, "role": "
578 central column" } ]
579 Block catalogue with dimensions <CATALOGUE> ... </CATALOGUE> Format: {
580 "type_i_block": [w_i, l_i, h_i], ... } w = width (x-axis), l =
581 length (y-axis), h = height (z-axis)
582 GLOBAL CONSTRAINTS
583 - The whole assembly must fit inside the bounding box
584 x \in [-1.5, 1.5] (width < 3)
585 y \in [-1.0, 1.0] (length < 2)
586 z \in [0.0, 5.0] (height < 5)
587 - Blocks are axis-aligned; do not rotate them.
588 - No two blocks may overlap (touching faces/edges is allowed).
589 - Every block above ground level must rest on, or be supported by,
590 blocks beneath it; hidden support blocks from <SELECTION> may be
591 used.
592 - Centroids are expressed in the same linear units as the catalogue.
593
594 TASK
595 For every physical block instance:
596
597 Assign a unique integer id starting at 0.
598 Specify its block type (exact key from the catalogue).
599 Output the centroid coordinates [x, y, z].
600 OUTPUT FORMAT (return nothing else)
601
602 PLACEMENT = {
603 0: { "type": "type_3_block", "centroid": [-1.2, 0.8, 0.25] },
604 1: { "type": "type_3_block", "centroid": [ 1.2, 0.8, 0.25] },
605 2: { "type": "type_7_block", "centroid": [ 0.0, 0.0, 1.75] },
606 ...
607 }
608
609 Note that "type\_0\_block" is the base that is not drawn in the sketch
610 . Always select it and assigns its centroid to [0, 0, -0.05].
611
612 Begin.

```

614 The second baseline follows the end-to-end paradigm of StackGen [32], training a single
615 transformer-based diffusion model to directly predict the full set of 3D block poses conditioned
616 on sketch input, without any explicit intermediate graph representations.

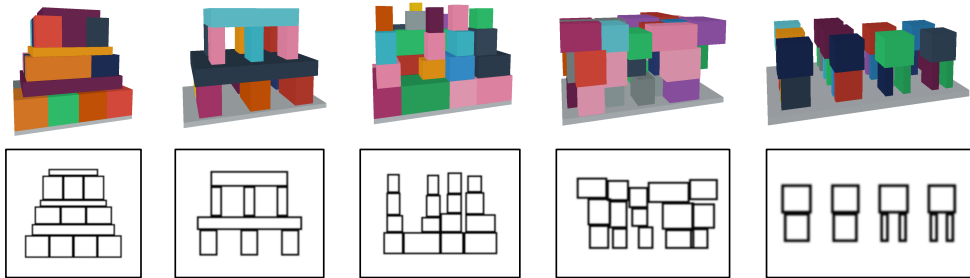


Figure 9: **Illustrative examples of sketch conversion.** For each synthetic 3D block structure (top), we extract the visible blocks from the front-view (x-z plane), and render their outlines as a 2D sketch (bottom). To better resemble human-drawn sketches, we further apply edge dilation and Gaussian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

617 This baseline involves two steps:

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

623 The CNN encoder transforms the input sketch into a compact latent embedding by passing it through
624 several convolutional layers, each followed by batch normalization and ReLU activations, and finally
625 by spatial pooling and a linear layer. This embedding is added to each object’s feature representation
626 before being processed by the transformer to promote cross-block information sharing and contex-
627 tualization.

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

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

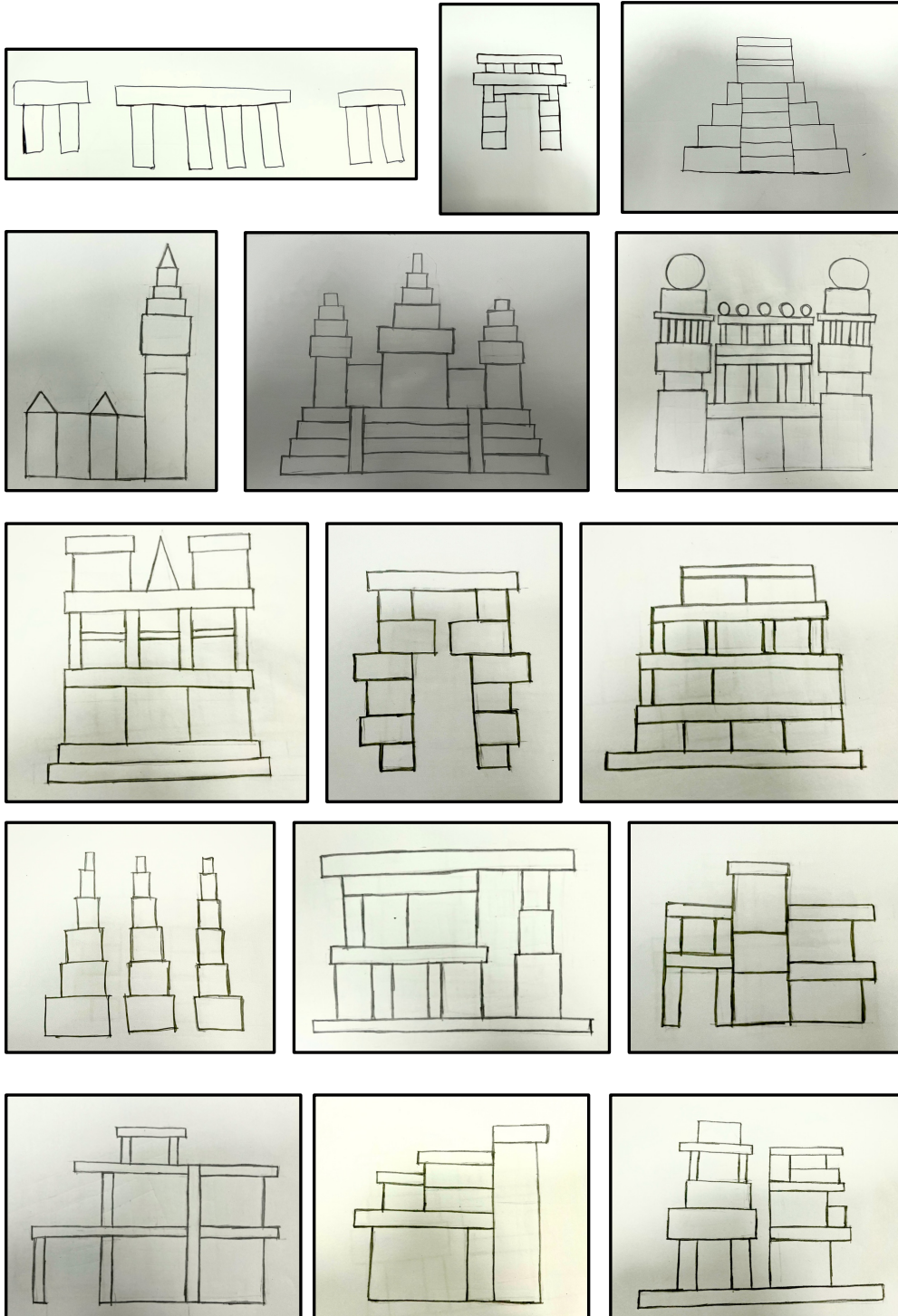
634 For each sample, we project the 3D arrangement to the x-z (front-view) plane. We retain only the
635 visible blocks, sorting them by their y-coordinates (depth), and filtering out those fully occluded by
636 others. The outlines of visible blocks are rendered onto a blank canvas, and their edges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

639 C.2 Our Ablation: No Hidden Object Prediction

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

647 D Hand-drawn Test Cases

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



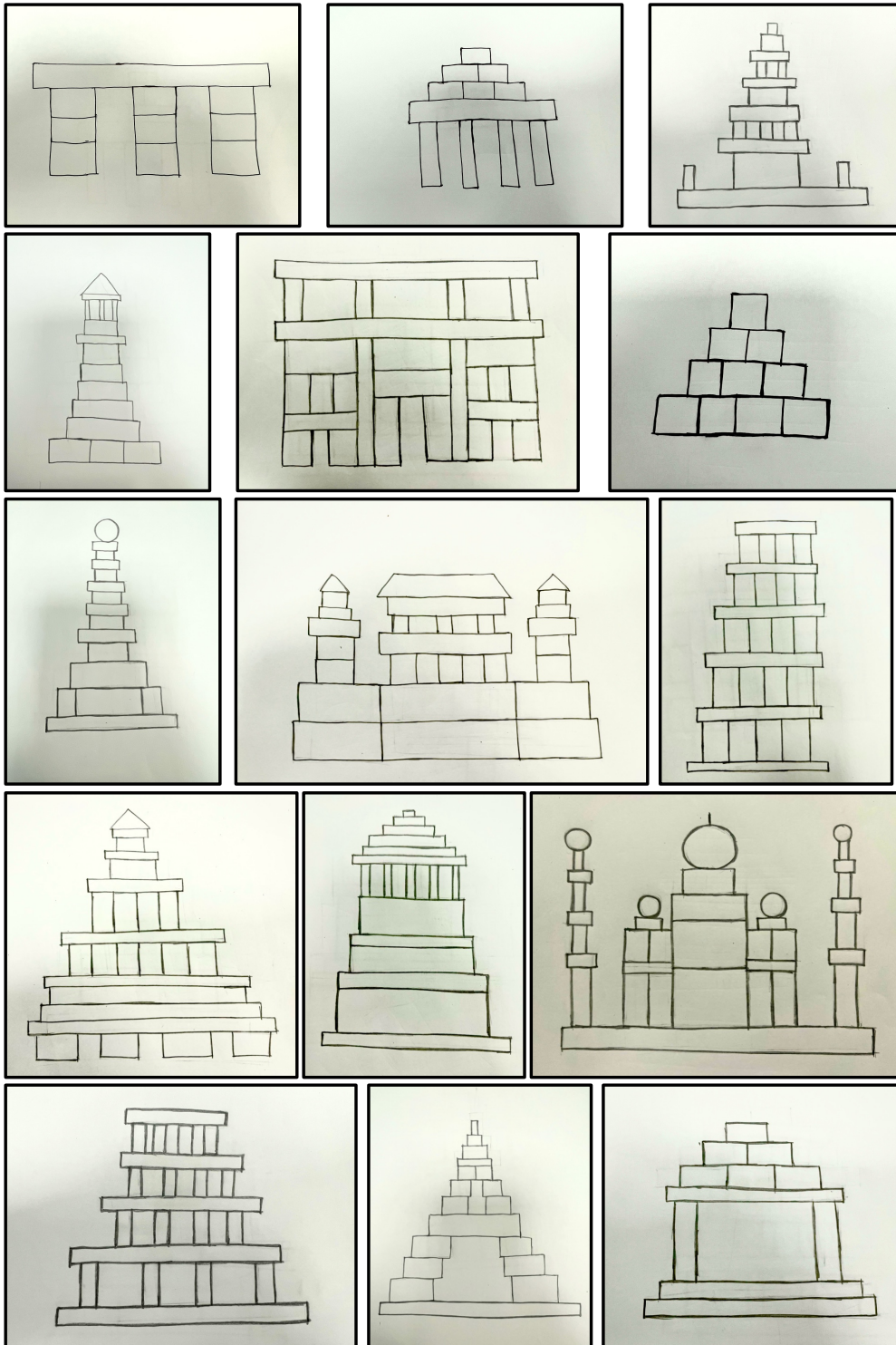


Figure 10: **Hand-drawn test cases.** The full set of 30 human-drawn 2D sketches used for evaluation, presented over two pages (15 per page). The test cases cover a variety of multi-level stacking scenarios and structural challenges.

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 [38], an explicit estimator of the gradient of the log-density for the perturbed data distribution $p_t(x)$ at time t :

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

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

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

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

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

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

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

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:

$$\nabla_x \log q_{\text{prod}}^t(x) \simeq \sum_{r \in \mathcal{G}} w_r \nabla_x \log q_r^t(x), \quad (7)$$

where q_r^t 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! [39].

683 We assume that StackItUp’s output, $\mathcal{O} = \{o_1, \dots, o_M\}$, is sorted such that lower-level objects
 684 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
 685 orientation:

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

686 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
 687 bracket with a known pose: by aligning the object with the bracket on a flat surface, its pose can be
 688 determined. Alternatively, vision-based methods can be used to estimate object poses.
 689 For each object type τ' , we define a relative grasp pose $\mathbf{H}_{\tau'}^{\text{grasp}}$ between the gripper frame and the
 690 object frame. We also specify a sequence of approach poses $\mathcal{H}_{\tau'}^{\text{pick}}$ for picking (*e.g.*, moving the
 691 gripper above the object, then descending to align with it). To pick up an object $o_i = (\tau_i, p_i)$, the
 692 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
 693 gripper. For placement, we define pre-placement and post-placement pose sequences, $\mathcal{H}_{\tau'}^{\text{pre-place}}$
 694 and $\mathcal{H}^{\text{post-place}}$, respectively, which are executed before and after opening the gripper.
 695 To assemble the target configuration \mathcal{O} , the robot performs the pick-and-place routine sequentially
 696 for each object.