arXiv:2605.18010v2 [cs.CV] 19 Aug 2026

Functionalization via Structure Completion and Motion Rectification

MINGRUI ZHAO, Simon Fraser University, Canada
SAI RAJ KISHORE PERLA, Simon Fraser University, Canada
KAI WANG, Simon Fraser University, Canada and ShanghaiTech University, China
SAURADIP NAG, Simon Fraser University, Canada
DUC ANH NGUYEN, Simon Fraser University, Canada
JIAYI PENG, Simon Fraser University, Canada
RUIQI WANG, Simon Fraser University, Canada
ANGEL X. CHANG, Simon Fraser University, Canada
MANOLIS SAVVA, Simon Fraser University, Canada
ALI MAHDAVI-AMIRI, Simon Fraser University, Canada
HAO ZHANG, Simon Fraser University, Canada

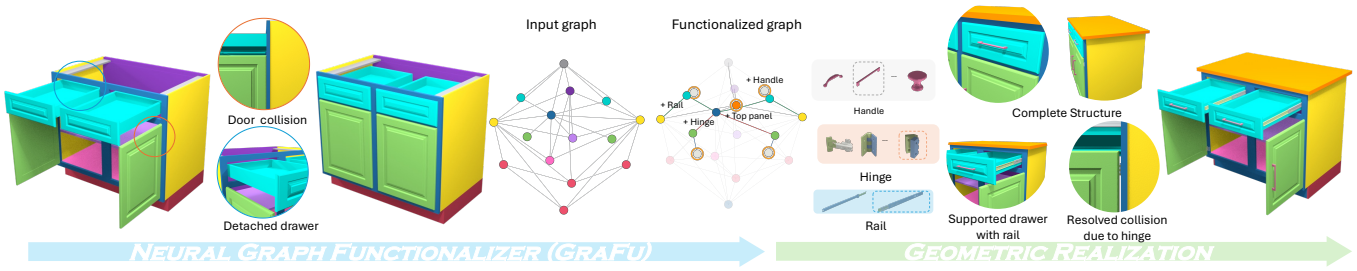


Fig. 1. Our method *functionalizes* a cabinet model that cannot function as it should, due to a) missing connectors for its door (hinge) or drawer (rails) to open and be held in place, and b) a collision between the door and the side panel. Our solution pipeline consists of a neural graph functionalizer (GraFu) for functional graph completion, e.g., adding missing connectors (new graph edges), as well as drawer handles (new graph nodes), followed by geometric realization to put actual hinges and rails in place. The hinge mechanism enforces a motion axis rectification that effectively fixes the initial collision.

Acquisition and creation of 3D assets have been largely view- or appearance-driven. As a result, existing digital 3D models often lack the requisite structural components to function as intended, such as joints, supports, interiors, or interaction elements. At the same time, even human-annotated motions are frequently error-prone, leading to physically implausible behavior. We introduce object *functionalization*, a novel task aimed at transforming visually plausible but non-functional 3D models into functional and physically operable ones. We formulate functionalization as a *graph completion* problem over a new functional graph representation, where labeled nodes represent object parts, labeled edges encode functional and contact relations, and movable nodes carry motion attributes, so that structural functional deficiencies manifest as missing nodes or incorrect edges. We develop a *neural Graph Functionalizer* (GraFu) to complete an incomplete graph representing a non-functional 3D object. The completed graph then drives a geometry realization stage that instantiates predicted connectors and structural elements in 3D, with the compelling side effect of rectifying erroneous human-annotated and predicted motions. To support training and evaluation, focusing on furniture as a rich and challenging target category, we introduce FurFun-233, a dataset of 233 paired non-functional and functionalized furniture models. On PartNet-Mobility ("zero-shot") and HSSD test sets, our method matches state-of-the-art methods in motion prediction accuracy while substantially improving functionality in terms of collision and connectivity. Project page: https://mingrui-zhao.github.io/Functionalization/.

Authors' Contact Information: Mingrui Zhao, Simon Fraser University, Canada, mza143@sfu.ca; Sai Raj Kishore Perla, Simon Fraser University, Canada; Kai Wang, Simon Fraser University, Canada and ShanghaiTech University, China; Sauradip Nag, Simon Fraser University, Canada; Duc Anh Nguyen, Simon Fraser University, Canada; Jiayi Peng, Simon Fraser University, Canada; Ruiqi Wang, Simon Fraser University, Canada; Angel X. Chang, Simon Fraser University, Canada; Manolis Savva, Simon Fraser University, Canada; Ali Mahdavi-Amiri, Simon Fraser University, Canada; Hao Zhang, Simon Fraser University, Canada.

1 Introduction

We live in a 3D world, surrounded by 3D objects that we constantly act on and interact with to carry out our daily activities. These activities define how we function as humans, while the same also applies to the robots we design to assist us. Correspondingly, the 3D objects have been constructed with their intended functionalities to afford the actions of humans or robotic agents alike. For example, all cabinets must have interior shelving and tops for storage and support, drawers have handles or grooves so that they can be pulled, doors have hinges to swing open, while a swivel chair must have a full cylinder assembly to stabilize and rotate its seat.

In reality, however, very few, if any, of the existing 3D assets fully possess the necessary *functional* components such as joints, handles, tracks, or other constituents of a part assembly for them to work. The majority of daily objects such as furniture and appliances do not have interior structures, and worse yet, a great many of them contain self collisions, under- or over-segmented parts, misalignments, etc., which prevent realization of functional motions. Even when the assets were manually created by 3D artists [Chang et al.

Table 1. A rough taxonomy of functional deficiencies related to *incomplete object structures*. Dynamic deficiencies are due to missing motion-enabling elements. Static deficiencies affect usability or interactability.

Category	Deficiency Type	Examples
Dynamic	Missing motion joints or connectors	Revolute: hinges, pins, pistons
		Prismatic: tracks, rails, sliders
Static	Missing interactable parts or elements	Additive: handles, knobs, pull bars
		Subtractive: sockets, grooves, etc.
	Missing support elements	Shelves, dividers, furniture tops

2015; Collins et al. 2022; Deitke et al. 2024], the modeling emphasis has predominantly been placed on their *appearance*, rarely their interior structures, and certainly not their functionalities.

Besides these geometric and topological functional deficiencies, accuracy is another issue. For example, the majority (80%+) of *human-annotated* articulations for the storage furnitures in PartNet-Mobility [Xiang et al. 2020] contain errors in the motion ranges or axes which may cause part collision during articulation. We argue that all of these deficiencies and inaccuracies for existing 3D assets need to be corrected, since the main purpose of reconstructing or generating a 3D object is *not* to merely look at it, but to *use* it, *interact with* it, perhaps to eventually *manufacture* it.

In this paper, we pose the novel problem of object *functionalization*, whose goal is to convert a non-functional 3D object into one that works as intended, by fixing its functional deficiencies through structural and geometric transformations. Given the immense diversity of object functionalities, a single unified solution capable of resolving *all* forms of functional deficiencies would be overly ambitious. As a first step, we focus on a tractable and well-defined subset: structural functional deficiencies stemming from *incomplete object structures*, as summarized in Table 1. Accordingly, our functionalization solution involves adding parts or connectors and adjusting part relationships, while addressing the challenge of devising a unified approach across both the dynamic and static settings.

To this end, we formulate object functionalization as a *graph completion* problem, which offers a unified treatment of both dynamic and static deficiencies. Specifically, we parse an input 3D model, with part segmentation, into a new *functional graph representation*, where every mesh part becomes a labeled node, every functional or contact relation becomes a typed edge (e.g., contact, hinge, rail, etc.), while movable nodes are further annotated with motion attributes. Under this representation, structural functional deficiencies manifest uniformly as missing nodes and/or incorrect edges, regardless of whether they are dynamic (e.g., missing hinges for revolute motion between a door and its frame and missing rails for prismatic motion) or static (e.g., a missing countertop node).

We develop a neural model called a *graph functionalizer* (GraFu), which takes an incomplete graph representing a non-functional object and transforms it into a complete graph representing its functional counterpart. Technically, GraFu is a graph-to-graph translation network comprising a node encoder, a Graph Transformer encoder operating on a fully-connected topology, and a DETR-style (i.e., DEtection TRansformers [Carion et al. 2020]) decoder that predicts new structural nodes, functional edges (hinge, rail, attached), and motion attributes for movable parts.

Graph completion with GraFu is only the first step, which is dedicated to structural (i.e., topological) functionalization without fully resolving geometric or motion precisions. In the second stage, the completed graph drives *geometric realization*, where predicted connectors are instantiated and snapped into place, template meshes are inserted for new structural nodes, and the full pipeline is wrapped in a Blender add-on for interactive review and adjustment. A compelling side effect of such a geometric functionalization is that by grounding part motion in explicit mechanical structures rather than virtual motion axes, any errors in the initial motion annotations, as those in PartNet-Mobility [Xiang et al. 2020], can be *rectified* to produce physically realistic, collision-free motions; see Fig. 1.

To support training GraFu and evaluation, we introduce FurFun-233, for Furniture Functionalization, a dataset of 233 furniture models with paired non-functional and functionalized counterparts, sourced from ShapeNet [Chang et al. 2015], Objaverse [Deitke et al. 2023], ABO [Collins et al. 2022], HSSD [Khanna* et al. 2023], and artist creations. Each model is pre-screened to ensure it contains interior structures, annotated with per-part semantic labels, and paired with a ground-truth functional graph. Functionalized counterparts include manually inserted mechanical components drawn from an expanded gallery of 9 connector types (7 hinges and 2 rails). While our formulation is general, we focus on furniture in this paper, as it represents one of the richest and most challenging categories of functional objects, encompassing both dynamic deficiencies (hinges, rails) and static ones (shelves, handles, countertops).

Our contributions can be summarized as follows:

- A functional graph representation encompassing part and connector/edge labeling and motion attributes.
- A neural graph functionalizer for structure completion.
- A two-stage functionalization pipeline that follows topological completion with geometry realization which can rectify predicted motion attributes via mechanical grounding.
- A high-quality dataset FurFun-233 to support training and evaluation of object functionalization models.

We evaluate our method for its motion prediction using GraFu by comparing against state-of-the-art baselines, Particulate [Li et al. 2025] and SINGAPO [Liu et al. 2024]. On the Storage Furniture subset of PartNet-Mobility (PN-M), GraFu matches their motion prediction accuracy while geometric realization substantially improves physical plausibility in terms of collision resolution and part connectivity. Our method also exhibits strong generalizability with this test since its training set does not contain any PN-M models. On HSSD, our method outperforms both baselines across all three metrics. We further demonstrate robustness via an ablation study in which input node labels are progressively masked up to 100%, showing that GraFu maintains competitive motion prediction and near-perfect connectivity even under severe label noise.

2 Related Work

With rapid advances in 3D generative and spatial AI, robotics, and the pursuit of world models, the value of high-quality 3D assets has

received increasing attention. Take single-view 3D reconstruction as an example, recent works such as CLAY [Zhang et al. 2024], Trellis [Xiang et al. 2025], and Hi3DGen [Ye et al. 2025] unanimously highlight the importance of *high-quality* 3D training data. However, achieving this only through careful data selection and curation is insufficient. Our work seeks to bring object functionalization as a new means of quality upgrades for existing 3D assets.

Broadly speaking, object functionalization falls into the category of shape modeling and editing, a well-studied topic in computer graphics with a variety of criteria or objectives such as symmetry [Mitra et al. 2007], stackability [Li et al. 2012], balance [Prévost et al. 2013], strength [Lu et al. 2014], foldability [Li et al. 2015], detail preservation [Botsch and Sorkine 2008], structure preservation [Mitra et al. 2013], and sketch conformation [Olsen et al. 2009], among others. Umentani et al. [2015] focus on interactive design of static plank-based furniture under frictional and nail-joint constraints, offering real-time guidance to a human designer. It does not address motion-enabling components such as hinges or rails, nor does it automate the functionalization of existing 3D assets. Gil-Ureta et al. [2016] proposes an algorithmic approach to insert kinematic joints, but relies on user-guided insertion signals. To our knowledge, no prior works targeted functionalizing 3D objects via shape completion or geometric transformation.

Shape Completion and Assembly. Shape completion has been extensively studied under different settings and representations from classical hole-filling algorithms [Centin et al. 2015; Ju 2004; Kazhdan et al. 2006; Kazhdan and Hoppe 2013; Liepa 2003; Tekumalla and Cohen 2004; Zhao et al. 2007] to more recent deep learning approaches [Arora et al. 2022; Tesema et al. 2023; Wang et al. 2022; Yan et al. 2022]. However, most of these techniques try to complete a portion of the shape that is missing due to occlusion or flaws in the scanning process as opposed to our technique that recovers a missing part necessary for interaction, movement, or function.

Shape assembly methods are more directly related to our approach, as they typically focus on predicting how parts and joints should be connected. For example, JoinABLe [Willis et al. 2022] is a learning-based assembly method which connects pairs of CAD parts by predicting the correct parametric joints, operating directly on boundary representations without class labels or human guidance. It also contributed the Fusion 360 Gallery assembly dataset.

Schulz et al. [2014] propose a data-driven interactive system to help users design 3D models that are *fabricable* by composing parts from a database of expert-created templates. It automatically aligns, parameterizes, and connects parts, with proper constraints and connectors even including screws and corner braces, so that the resulting models can be manufactured. Such a "fabricabilization" process bears resemblance to one of our functionalization options, namely, hinge/track insertion, but without motion rectification. Generally, while fabricability and functionality are related, they are distinctive modeling goals. Operationally, our work is less about shape assembly but more on structure completion and shape transformation.

In other related work Lin et al. [2017] reconstruct kinematic assemblies from raw 3D scans by inferring part segmentation and joint parameters, and Mitra et al. [2010] analyze and visualize mechanical motion relationships. More recently, Kinematic Kitbashing [Guo et al. 2025] assembles reusable parts into a prescribed kinematic graph by optimizing per-part transforms under an exemplar-based vector distance field energy aggregated across each joint's motion range. However, all of these methods presuppose a complete assembly as input, whereas our work addresses inferring and inserting *missing* functional geometry to enable articulated motion.

Motion Prediction and Generation. Since many object functions are realized through part motions and human-object interactions, works that predict, reconstruct, and/or generate part articulations [Liu et al. 2025] are relevant to functionalization, but their goal is not to functionalize an object by altering its shape or part structure. Recent data-driven methods predict part articulations directly from geometry or images [Le et al. 2024; Li et al. 2025; Liu et al. 2024, 2025], but while effective at inferring *where* and *how* parts move, they do not address *why* a motion may be physically implausible or how to correct it. Our rule-based geometric fixing stage is thus complementary: as shown in our experiments, applying hinge and rail insertion on top of predicted motion axes consistently reduces collisions, demonstrating that learned articulation and explicit mechanical grounding are mutually reinforcing.

ArtLLM [Wang et al. 2026] autoregressively predicts part layouts and joint structures from point clouds using a multimodal large language model, generating high-fidelity articulated assets; while it advances joint inference and part geometry synthesis, it does not address functional deficiencies in existing assets or insert physical connectors to correct implausible motions. In this spirit, annotated [Xiang et al. 2020] or generated (e.g., [Li et al. 2025; Qiu et al. 2025; Vora et al. 2025]) part motions can serve to initialize motion parameters in our dynamic functionalization setting, which are then improved through functionalization. Motion prediction, and the ensuing simulation, has recently been employed for functional anomaly *detection* [Bhunia et al. 2025] by comparing observed (possibly abnormal) part articulations with anticipated (normal) ones. But the detected anomalies are not corrected.

Interior Modeling. Object interiors are often exposed under part motion, e.g., when opening a cabinet door or drawer. As a result, recent works on geometry and interaction reconstruction such as DRAWER [Xia et al. 2025] and S2O [Iliash et al. 2024] contain modules which complete drawer/cabinet interiors that were hidden from the input videos or missing from the input 3D objects, using a category-specific template approach similar to ours.

For representation learning, aside from volumetric representations such as NeRF [Mildenhall et al. 2020] and 3DGS [Kerbl et al. 2023], object interiors can also be modeled by multi-slice images [Wang et al. 2024] and variants of signed distance functions (SDFs) such as 3-poled SDF [Chen et al. 2022]. But all of these works have targeted reconstruction or generation tasks, e.g., from images or point clouds, not functionalization, for which we simply employ polygonal meshes. Furniture interiors, along with part articulation, can be acquired in conjunction through active 3D reconstruction [Yan et al. 2023], where a robot interacts with a physical piece (e.g., opening/closing a drawer) while scanning the geometry.

Fixing Malfunctions. To our knowledge, the closest work to ours is [Hong et al. 2022], which also resolves functional deficiencies via

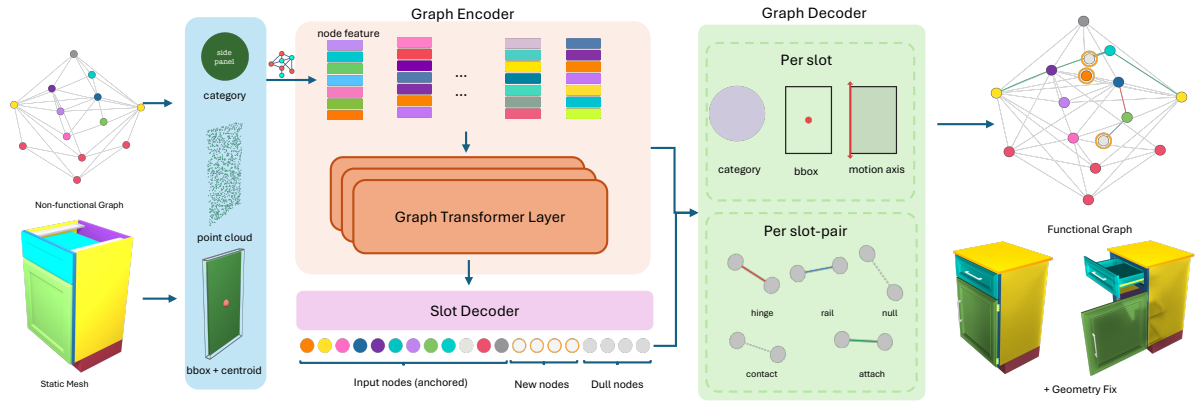


Fig. 2. **Our Pipeline.** Given a non-functional 3D furniture asset, the system encodes each part node using category, point cloud, and bounding box centroid features. A Graph Transformer then models inter-node relationships across the fully-connected graph. The resulting node features are passed to a DETR-style Slot Decoder, which predicts per-slot attributes (category, bounding box, motion axis) and per-slot-pair edge types (hinge, rail, contact, attach, or null) across anchored input nodes, free new-node slots, and dull (padding) slots. The output is a functionalized graph that is subsequently grounded in 3D geometry via the Geometry Fix stage, yielding a physically operable furniture model with instantiated hinge and rail mechanisms.

a discrete set of fixes detected by a physics simulator. However, it focuses solely on motion malfunctions, whereas our work covers both dynamic and static deficiencies. More fundamentally, our motion rectification is not simulation-driven but a natural consequence of inserting proper mechanical connectors, which allows us to correct even the erroneous human annotations in PartNet-Mobility, which usually serves as ground truth. For static deficiencies, no prior work directly addresses the full scope we tackle: works such as [Hržica et al. 2022] and [Badami et al. 2017] address the related but distinct problems of storage space detection and furniture segmentation without modifying geometry, while PhysX-Anything [Cao et al. 2025] and PhysX-3D [Cao et al. 2026] generate simulation-ready assets from scratch rather than functionalizing existing ones. Procedural tools such as the IKEA kitchen planner similarly operate under rigid predefined templates and do not reason about functional deficiencies in arbitrary 3D assets.

3 Method

We address the problem of *functionalization*—transforming non-functional 3D objects, particularly furniture, into physically plausible, functional counterparts. Our approach consists of two main stages: a) The input 3D model is first converted into a graph representation that captures part-level relationships within the object. In this graph, functional deficiencies, missing nodes, or incorrect edges may arise due to imperfections in the input model. A novel **Gra**ph **Fu**nctionalizer (GraFu) module then operates on this non-functional graph to predict the missing functional nodes, correct relational edges, and estimate motion axes for dynamic nodes, thereby producing a complete functional graph. b) The 3D geometry is subsequently updated by cross-referencing the predicted graph with the original input: a suite of *geometric post-fix* algorithms directly embeds hinge and rail mechanisms into the 3D model for newly introduced revolute and prismatic edges, respectively, while template or bounding-box geometries are instantiated for any newly added static nodes. To support practical usability, we additionally provide a Blender add-on that allows users to interactively inspect and refine the output at each stage of the pipeline. In the following, we discuss each of these components in detail. The overall pipeline of our method is illustrated in Figure 2.

3.1 Functional Graph Parsing

For a segmented 3D model, we construct a functional graph $G = (V, E)$, where V denotes the set of annotated part nodes and E denotes the set of inter-part contact relationships. Each node $v \in V$ encodes three attributes: the 3D bounding box of the part, its geometric representation, and a semantic label. Part labels are assigned from a fixed vocabulary: `{top/side/back/bottom panels, face frame, shelf, divider, bar, leg, door, drawer, handle, misc}`. This label set covers the vast majority of structural geometries encountered in practice; parts that do not fit any named category, such as stretchers, are assigned to the `misc` label. Each edge $e \in E$ belongs to one of four types: `{contact, hinge, rail, attached}`. A `contact` edge encodes static structural connectivity, representing joints analogous to glued or nailed panels in physical assemblies. `Hinge` and `rail` edges capture revolute and prismatic kinematic relationships between movable parts and their fixed mounting anchors. Handles are connected to their parent door or drawer via an `attached` edge. For dynamic nodes, a motion axis M is additionally defined in the local coordinate frame of the part. For revolute joints, the motion axis is selected from the bounding box edges of the part, where a positive sign denotes counter-clockwise rotation and a negative sign denotes the opposite. For prismatic joints, the motion axis is assigned as one of the canonical $\pm\{x, y, z\}$ directions.

3.2 GraFu Network

GraFu is a graph-to-graph translation network that transforms a non-functional graph G_u into a functional graph G_f by predicting

missing structural nodes, inferring functional edges (`hinge`, `rail`, `attached`), and estimating motion attributes for dynamic nodes. The architecture comprises three sequential components: a) a *Node Encoder* that extracts per-node feature representations, b) a *Graph Encoder* that models relational dependencies across nodes, and c) a *Graph Decoder* that produces the functionalized output graph G_f. The complete pipeline is illustrated in Figure 2.

Node Encoder. The node encoder maps each node $v \in V$ in the non-functional graph G_u to a fixed-dimensional latent embedding. The *part category*, represented as a one-hot vector, is projected into a latent embedding of dimension $d_{\text{cat}} = 128$ via a lightweight MLP, capturing the semantic identity of each node. The 3D geometry of each part is encoded using a lightweight variant of PointNet++ [Qi et al. 2017], which maps the per-part point cloud to a latent embedding of dimension $d_{\text{pc}} = 64$. Additionally, the centroid of the part bounding box is encoded by a separate lightweight MLP into a latent of dimension $d_{\text{geo}} = 32$. The three embeddings are concatenated to yield a unified per-node latent representation $z_v \in \mathbb{R}^{224}$ for each node $v \in V$. By design, the category embedding d_{cat} constitutes the dominant component of z_v, since part category is the most informative attribute for describing each node.

Graph encoder. To model inter-node relationships in the non-functional graph, we employ a lightweight Graph Transformer that operates over a *fully-connected* graph topology, where all nodes attend to one another and absent edges are assigned `null`-type edge labels. Edge features are constructed by encoding the edge connectivity type as a one-hot vector, augmented with an 8-dimensional geometric descriptor $[\, \|r\|, \|g\|, \hat{r}, g_{\text{axis}} \,]$ that captures the spatial relationship between part bounding boxes. Here, $\hat{r}$ denotes the signed unit vector along the centroid-to-centroid direction, and $g_{\text{axis}} \in \mathbb{R}^3_{\geq 0}$ represents the per-axis surface gap between oriented bounding boxes (set to zero where boxes overlap). The connectivity encoding and geometric descriptor are jointly projected into a 128-dimensional edge feature embedding z_e for each node pair using an MLP. At each Graph Transformer layer, z_e is incorporated into the attention computation as a learned edge bias:

$$\text{Attn}(z_v^Q, z_v^K, z_v^V) = \left(\frac{z_v^Q . (z_v^K)^T}{\sqrt{d}} + z_e\right).z_v^V, \tag{1}$$

where z_v^Q, z_v^K and z_v^V are Query/Key/Value derived from the node embeddings z_v. This formulation enables the model to jointly reason over both node attributes and inter-part relational structure, such as which parts are connected via hinge or rail joints, thereby improving its capacity to predict a functionally coherent output graph.

Graph Decoder. To decode the learned latent representations into a functional graph, we adopt a DETR [Carion et al. 2020]-style Transformer Decoder as our Graph Decoder. Rather than using object queries as in the original DETR formulation, we introduce *node slots* that serve as learned queries over graph nodes. Graph Encoder features are passed to the Graph Decoder via cross-attention. We initialize N *anchor slots*, one per node in the input graph, and set a global maximum slot count of M. To allow the decoder to predict nodes beyond those present in the non-functional input, we introduce K additional *free query slots* alongside the N anchor slots. These free queries are tasked with recovering missing structural nodes

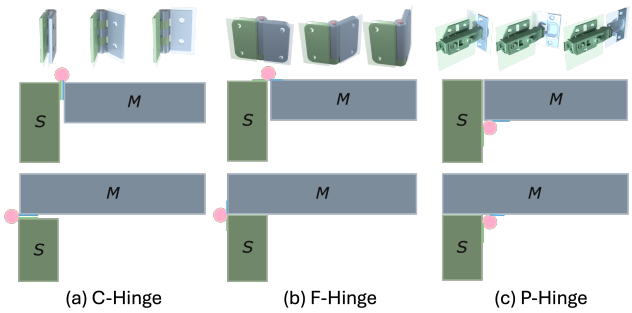


Fig. 3. Various application conditions for Contact-face snapping, Flushing-face snapping, and Perpendicular-face snapping hinges under counter-clockwise rotation motion.

required to make the graph functional; unused slots are zero-padded and treated as null nodes. To provide free queries with global graph context, we pool the Graph Encoder output and add its projected summary as an initialization to each free query embedding. Each decoder slot independently predicts: (i) node validity, (ii) part category, (iii) bounding box, and (iv) centroid. For every pair of nodes, the decoder additionally classifies the connecting edge into one of {`rail`, `hinge`, `attached`, `contact`, `null`}. When a `hinge` or `rail` edge is predicted between dynamic nodes, the motion axis can be estimated by aligning the edge on the target geometry. The first three edge types are directed, encoding a parent-child relation between parts, while `contact` and `null` edges are treated as bidirectional.

Training. To train GraFu, we require paired (non-functional, functional) graph data. To address the scarcity of such supervision, we introduce the FurFun-233 dataset, which provides ground-truth functional graphs. Non-functional counterparts G_u are synthesized from each functional graph G via a set of structure-corrupting augmentations, including random edge collapse, node and edge removal, node and edge label masking, and geometric flips. The resulting paired data constitutes the training corpus for GraFu. The whole network is trained using a combination of an anchor loss, node and edge reconstruction losses, and a graph completion loss, all computed over the synthesized training pairs. Full details of the GraFu architecture, loss formulations, ablation studies, and augmentation strategies are provided in the supplementary material.

3.3 Geometric Fixing

Given the predicted graph and input mesh, Stage 2 realizes the prediction in 3D via dynamic and static geometric fixes. For every predicted hinge or rail edge, the corresponding mechanical geometry is instantiated and integrated into the mesh, with dynamic nodes inheriting the embedded motion animations — mirroring how real-life furniture motion is governed by physical joints rather than virtual axes. For every predicted new node, a template mesh is instantiated and surfaced through a Blender API for the user to visualize, tune, accept, or deny.

Hinge Insertion. Inserting actual hinge geometries into the static mesh offers multiple benefits. First, it ensures physical motion plausibility by keeping all parts connected throughout motion, with the range naturally constrained by the hinge. Second, the hinge carries

its own local revolution system, snapping it between the movable and static parts rectifies the predicted motion axis, making the revolute motion geometrically grounded and physically consistent.

Physically plausible revolute motion requires *bilateral surface contact*. This means that the joint must simultaneously contact both the static frame and movable part at geometrically compatible locations. We identify three hinge types based on surface arrangement (Fig. 3): C-hinge for two contacting faces, F-hinge for flushing faces, and P-hinge for perpendicular interior faces. Together these three types empirically cover all furniture revolute joints, with C-hinge being the most prevalent in practice.

For hinges, we apply a template-based snapping algorithm using a gallery of the three hinge types, each annotated with static/dynamic snap planes and a rotation axis. Given a predicted hinge edge and its two nodes, we detect the presence of contacting, flushing, or interior perpendicular faces to determine the appropriate hinge type, after which a local geometric solver finds the optimal hinge placement.

Given a static-side part S and a dynamic-side part D from the predicted edge, we (i) align the rotation axis to the network-predicted `hinge_axis_signed` on D, (ii) place the hinge along the predicted `hinge_border` face of D, and (iii) snap it to the actual surfaces of S and D by solving a 3×3 linear system for a translation Δ:

$$\mathbf{n}_s \cdot \Delta = t_s, \quad \mathbf{n}_d \cdot \Delta = t_d, \quad \hat{\mathbf{a}} \cdot \Delta = 0,$$

where $(\mathbf{n}_s, \mathbf{n}_d)$ are the world-space snap-plane normals, (t_s, t_d) are robust median ray-cast distances from the snap planes to S and D along $-\mathbf{n}$ and $\hat{\mathbf{a}}$ is the rotation-axis direction.

Rail Insertion. Rail insertion follows a similar template-based approach, with each template pre-annotated with static and dynamic snap planes. The rail is oriented to the predicted sliding axis, snapped to the movable part's main body, and scaled to its length. Since most 3D assets lack mechanical compatibility, gaps between the rail and mounting panel are bridged with an additional supporting block. The inserted rail constrains motion range proportionally to the part size and ensures the movable part remains physically connected throughout animation.

Static Functionalization. In addition to functional edge instantiation, our GraFu also predicts new nodes that complement the model structure, such as additional handles, top panel (which is the most commonly missed item in the PartNet-Mobility dataset), shelves and dividers. We develop respective algorithms for each of the categories, detailed in the Supplementary Materials.

Interactive Correction. Upon functional graph prediction, all dynamic functional connectors, top panels and handles are automatically instantiated from the predicted motions and newly added nodes. Human intervention contributes what automation cannot decide, namely personal preference and design aesthetics beyond the automatic installation: template variants, handle styles and positions, additive or subtractive forms, and the interior layout of shelves and dividers. Our Blender API bridges this window, letting users express design intent and further tune or correct functionalized models after automation.

4 Experiments

We evaluate our functionalization output along three axes: *motion correctness*, *motion plausibility*, and *structural completeness*. Motion correctness measures the position and sign of the predicted motion axis. Motion plausibility captures the collision rate and rootedness of the dynamic geometry—whether movable parts detach from or penetrate the static body during articulation. Structural completeness measures whether all required structural elements are present.

FurFun-233 Dataset. To facilitate further investigations on *functionalization*, we introduce FurFun-233, a dataset of **233** furniture assets with paired *non-functional* and *functionalized* versions. Assets are sourced from ShapeNet [Chang et al. 2015], Objaverse [Deitke et al. 2023], ABO [Collins et al. 2022], HSSD [Khanna* et al. 2023], and artist creations, all pre-screened to ensure non-monolithic interiors. Functionalized versions are produced by human annotators via per-part labeling and manual insertion of mechanical components (*e.g.*, handles, hinges, and rails), with appropriate scaling and placement to match the original design intent, and are coupled with ground truth functional graphs.

Datasets and Training. We benchmark on two datasets: the Storage Furniture subset of PartNet-Mobility (PN-M), and 50 randomly sampled models from HSSD. For the PN-M benchmark we train a separate checkpoint that *excludes* all 33 ShapeNet-sourced models to avoid data leakage; for the remaining experiments we train on the FurFun training split (80%). We set M=55, K=4, and use Adam (lr=3e−4) with cosine annealing. Training data is augmented by extracting non-functional graphs from functional ones (see Supplement). Training completes in 1.5 hours on a single RTX 4090.

4.1 Quantitative Results

Baselines and Protocol. We compare against Particulate [Li et al. 2025] and SINGAPO [Liu et al. 2024], as well as ground-truth annotations, on PN-M and HSSD. PN-M segmentations and category labels are mapped to our taxonomy via keyword matching (e.g.,`panel side` maps to the side panel category), covering 67% of the parts, followed by post-verifications to correct any mislabelings; the remainder of PN-M (e.g.,ambiguous `vertical bar` parts) and the HSSD test set are manually segmented at the part level. The input non-functional graph uses contact-only edges, determined by bounding-box intersection between parts.

To ensure a fair comparison for the two baselines, we expose both of them to maximal GT information: for Particulate, we replace its segmented meshes with GT part meshes after shape matching; for SINGAPO, we supply the GT mobility graph (bypassing its LLM-based prediction), inpaint its bounding-box prediction with the GT bboxes during diffusion, and provide GT meshes upon retrieval.

Generalizability. Note that both baselines were previously trained on PN-M, whereas our method is "zero-shot" on this dataset, since our training data does not contain any ShapeNet, and hence PN-M, models. This suggests that a comparison on PN-M is also a test of generalizability for our method.

Metrics. Because PN-M contains substantial model duplication with differing motion annotations, we report two motion-correctness measures: *GT-aligned*, requiring a one-to-one match with the PN-M annotation, and *Reasonable*, accepting any motion observed on other

Table 2. Quantitative evaluation on PN-M (345 models, "zero-shot") and HSSD (50 models). *GT-aligned* matches dataset annotations one-to-one; *Reasonable* is human-verified plausibility (PN-M only). Collision and connectivity cells report **raw** (+**GeoFix**), where the second value in brackets is obtained by applying our dynamic connector insertion on top of the method's predicted motion axes for motion rectification. The *GT annotation* row applies GeoFix on top of ground-truth motion axes annotated by humans, isolating the contribution of geometric realization. *Ours* produces a single integrated result, so no separate raw/+GeoFix split is shown. Best per slot in **bold**, second best underlined (*GT annotation* excluded from ranking).

	PartNet-Mobility				HSSD		
	Motion Correctness ↑		Collision ↓	Connectivity ↑	Motion ↑	Collision ↓	Connectivity ↑
Method	GT-aligned	Reasonable	(%)	(%)	GT-aligned	(%)	(%)
Particulate [Li et al. 2025]	87.6%	**96.5%**	7.9 (5.5)	13.2 (96.5)	27.4%	29.0 (28.7)	62.8 (58.0)
SINGAPO [Liu et al. 2024]	**87.8%**	93%	28.7 (27.0)	48.4 (73.8)	59.4%	28.7 (24.6)	51.7 (98.1)
GT annotation	–	–	19.2 (5.2)	49.1 (97.2)	–	25.0 (25.0)	82.1 (97.7)
Ours	87.0%	96.3%	**4.4**	**98.1**	**85.8%**	**15.1**	**98.6**

GT annotations of the same geometry. For collision, we densely sample 50 frames spanning each part's rest-to-full-extent motion and count frames in which the movable and static parts intersect; the per-object rate is averaged over all movable parts. Connectivity is measured by the distance between the movable and static parts along the motion: if this distance exceeds 5 mm (models are normalized to a unit cube), the part is deemed detached, and we report the fraction of movable parts that remain connected.

Results. Table 2 summarizes the quantitative comparison results. GraFu predicts plausible motion for movable parts, and mechanical-part insertion substantially improves physical motion rationality. The geometric fix is method-agnostic: applied on top of Particulate or SINGAPO motion axes, or human-annotated GT, it consistently improves both collision and connectivity. On PN-M, our method matches the baselines on motion prediction accuracy despite being zero-shot, indicating strong generalizability across datasets.

On HSSD, our method outperforms both baselines across all metrics, while it is worth mentioning that our training data does contain models from HSSD, while the two baselines did not. Particulate tends to predict motion axes that detach the movable part from the static frame, yielding a low collision rate but also low connectivity—our mechanical-part insertion repairs both failure modes.

Robustness to Missing Labels. We further ablate on partially labeled inputs, simulating the case where some part labels are missing or unknown. We mask 10%, 30%, 50%, and 100% of input node labels at inference (Table 3). The performance degrades monotonically with corruption, while remaining in a reasonable band. The quantitative numbers are reported from a five-run average to account for the stochasticity in random node masking.

Static Functionalization. We further evaluate the structural completeness enabled by our static functionalization on the PN-M dataset. Through GraFu prediction and the automatic geometric fix, 100% of the top-less models are completed with a top panel, and 95.4% of the models are paired with graspable handles. Interior completion, invoked upon user request, applies to models containing compartments that can accommodate internal structures; among these, 98.5% are successfully populated with shelves and dividers. We provide further qualitative results, detailed statistics and failure case analysis in the supplementary material.

Table 3. Robustness ablation on PN-M and HSSD under varying levels of missing node labels. At masking rate r, a random fraction r of input node material labels is replaced with unknown.

	PartNet-Mobility			HSSD		
Masking rate	Motion Corr. ↑ (GT-aligned)	Coll. ↓ Rate (%)	Conn. ↑ (%)	Motion Corr. ↑ (GT-aligned)	Coll. ↓ Rate (%)	Conn. ↑ (%)
10%	86.23%	4.49	98.28%	72.68%	14.71	98.72%
30%	85.76%	4.36	97.88%	70.33%	16.02	98.53%
50%	85.44%	4.46	97.83%	62.68%	16.82	96.93%
100%	82.07%	5.3	96.79%	52.87%	21.8	96.10%

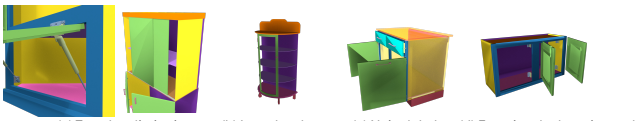
4.2 Qualitative Results

Figure 5 presents qualitative outputs of GraFu and its geometric refinement. From top to bottom, input graph nodes are progressively masked (masked nodes shown in pale gray); the bottom two rows are fully unlabeled. Our method robustly emits functional edges and produces realistic articulation through mechanical-part insertion. Newly introduced nodes are highlighted: rows 1, 4, 5, and 7 gain a top panel; rows 1 and 3 gain handles, all installed automatically by our functionalization pipeline. Figures 6 and 7 further demonstrate our functionalization operations in inserting mechanical parts, bridging connectivity, resolving collisions, structural completion and application to generated objects.

Comparison to LLM-based Methods. Finally, we compare against an LLM-based baseline that uses a commercial LLM to complete non-functional graphs and BlenderMCP to perform mechanical-part insertion. Qualitative results are provided in Supplementary.

5 Conclusion, Limitations, and Future Work

We present object functionalization, a novel 3D modeling task to convert visually plausible but non-functional 3D assets into objects that operate as intended. To our knowledge, this is the first systematic attempt to correct major functional deficiencies in both dynamic and static settings. Our functionalization pipeline operates in two stages, by disentangling relation/topology prediction, via a graph completion neural network (GraFu), and geometry realization driven by the predicted graph which can help refine the predicted motion parameters (e.g., for collision avoidance).



(a) Template limitation (b) Irregular shape (c) Noisy label (d) Functional edge mismatch

Fig. 4. Limitations and failure cases. Our method is bounded by the available template gallery, which does not yet cover piston or sliding-door motions (a); struggles with highly irregular shapes that violate the bounding-box assumptions of our geometric solver (b); inherits errors from contaminated input labels (c); and is susceptible to cascaded failures when GraFu predicts the wrong mounting node for a functional edge (d)

While effective, our approach is still limited on several fronts. First, it can only functionalize parts that have been segmented, which would not cover, for example, the top drawer in Figure 7. This example also reveals that our current method can only add whole parts, not partial structures, such as the missing panels of a drawer. Second, our method does not handle geometric deficiencies that compromise physical validity at the modeling or mesh level (e.g., multiple doors in one segment or shell-only models). Last but not least, our current focuses has been on a limited set of motion types (e.g., no chained motions) and a narrow set of object categories (i.e., furniture) as dictated by the training set we constructed. Figure 4 shows several failure cases that reveal additional limitations.

Despite these limitations, we believe that our first attempt at addressing such an important modeling problem opens the door for further research. Enriching our FurFun dataset and extending our framework to accommodate broader functional mechanisms and additional object categories would increase applicability. Beyond furniture, functionalization has potential applications in fabrication, simulation, and embodied AI, where functional correctness is critical. We hope this work encourages the community to move beyond appearance-driven 3D assets and toward models that are not only visually plausible but also physically usable and functional.

Acknowledgements

We sincerely thank all anonymous reviewers for their careful reading and constructive feedback. We also thank Dr. Francisca Gil-Ureta and Aditya Ganeshan for the insightful discussion.

References

Himanshu Arora, Saurabh Mishra, Shichong Peng, Ke Li, and Ali Mahdavi-Amiri. 2022. Multimodal Shape Completion via Implicit Maximum Likelihood Estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2958–2967.

Ishrat Badami, Manu Tom, Markus Mathias, and Bastian Leibe. 2017. 3D Semantic Segmentation of Modular Furniture using rjMCMC. In *WACV*.

Ankan Kumar Bhunia, Changjian Li, and Hakan Bilen. 2025. Interactive Anomaly Detection for Articulated Objects via Motion Anticipation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=t22zHB6yQ0

Mario Botsch and Olga Sorkine. 2008. On Linear Variational Surface Deformation Methods. *IEEE TVCG* 14, 1 (2008).

Ziang Cao, Zhaoxi Chen, Liang Pan, and Ziwei Liu. 2026. Physx-3d: Physical-grounded 3d asset generation. *Advances in Neural Information Processing Systems* 38 (2026), 93771–93784.

Ziang Cao, Fangzhou Hong, Zhaoxi Chen, Liang Pan, and Ziwei Liu. 2025. PhysX-Anything: Simulation-Ready Physical 3D Assets from Single Image. *arXiv preprint arXiv:2511.13648* (2025).

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

Marco Centin, Nicola Pezzotti, and Alberto Signoroni. 2015. Poisson-driven seamless completion of triangular meshes. *Computer Aided Geometric Design* 35 (2015), 42–55.

Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. 2015. ShapeNet: An information-rich 3D model repository. *arXiv preprint arXiv:1512.03012* (2015).

Weikai Chen, Cheng Lin, Weiyang Li, and Bo Yang. 2022. 3PSDF: Three-Pole Signed Distance Function for Learning Surfaces with Arbitrary Topologies. In *CVPR*.

Jasmine Collins, Shubham Goel, Kenan Deng, Achleshwar Luthra, Leon Xu, Erhan Gundogdu, Xi Zhang, Tomas F Yago Vicente, Thomas Dideriksen, Himanshu Arora, Matthieu Guillaumin, and Jitendra Malik. 2022. ABO: Dataset and benchmarks for real-world 3D object understanding. In *CVPR*. 21126–21136.

Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, et al. 2024. Objaverse-xl: A universe of 10m+ 3d objects. *Advances in Neural Information Processing Systems* 36 (2024).

Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. 2023. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 13142–13153.

Francisca Gil-Ureta, Chelsea Tymms, and Denis Zorin. 2016. Interactive modeling of mechanical objects. In *Computer Graphics Forum*, Vol. 35. Wiley Online Library, 145–155.

Minghao Guo, Victor Zordan, Sheldon Andrews, Wojciech Matusik, Maneesh Agrawala, and Hsueh-Ti Derek Liu. 2025. Kinematic Kitbashing for Modeling Functional Articulated Objects. *arXiv preprint arXiv:2510.13048* (2025).

Yining Hong, Kaichun Mo, Li Yi, Leonidas J Guibas, Antonio Torralba, Joshua B Tenenbaum, and Chuang Gan. 2022. Fixing malfunctional objects with learned physical simulation and functional prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1413–1423.

Mateja Hržica, Petra Pejić, Ivana Hartmann Tolić, and Robert Cupec. 2022. Detection of household furniture storage space in depth images. *Sensors* 22, 18 (2022), 6774.

Denys Iliash, Hanxiao Jiang, Yiming Zhang, Manolis Savva, and Angel X Chang. 2024. S2O: Static to openable enhancement for articulated 3D objects. *arXiv preprint arXiv:2409.18896* (2024).

Tao Ju. 2004. Robust repair of polygonal models. *ACM Transactions on Graphics (TOG)* 23, 3 (2004), 888–895.

Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, Vol. 7.

Michael Kazhdan and Hugues Hoppe. 2013. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)* 32, 3 (2013), 1–13.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4 (July 2023). https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

Mukul Khanna*, Yongsen Mao*, Hanxiao Jiang, Sanjay Haresh, Brennan Shacklett, Dhruv Batra, Alexander Clegg, Eric Undersander, Angel X. Chang, and Manolis Savva. 2023. Habitat Synthetic Scenes Dataset (HSSD-200): An Analysis of 3D Scene Scale and Realism Tradeoffs for ObjectGoal Navigation. *arXiv preprint* (2023). arXiv:2306.11290 [cs.CV]

Long Le, Jason Xie, William Liang, Hung-Ju Wang, Yue Yang, Yecheng Jason Ma, Kyle Vedder, Arjun Krishna, Dinesh Jayaraman, and Eric Eaton. 2024. Articulate-Anything: Automatic Modeling of Articulated Objects via a Vision-Language Foundation Model. *arXiv preprint arXiv:2410.13882* (2024).

Honghua Li, Ibraheem Alhashim, Hao Zhang, Ariel Shamir, and Daniel Cohen-Or. 2012. Stackabilization. *ACM TOG* 31 (2012), article 158. Issue 6.

Honghua Li, Ruizhen Hu, Ibraheem Alhashim, and Hao Zhang. 2015. Foldabilizing furniture. *ACM TOG* 34 (2015), article 90. Issue 4.

Ruining Li, Yuxin Yao, Chuanxia Zheng, Christian Rupprecht, Joan Lasenby, Shangzhe Wu, and Andrea Vedaldi. 2025. Particulate: Feed-Forward 3D Object Articulation. arXiv:2512.11798 [cs.CV] https://arxiv.org/abs/2512.11798

Peter Liepa. 2003. Filling holes in meshes. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. 200–205.

Minmin Lin, Tianjia Shao, Youyi Zheng, Niloy Jyoti Mitra, and Kun Zhou. 2017. Recovering functional mechanical assemblies from raw scans. *IEEE transactions on visualization and computer graphics* 24, 3 (2017), 1354–1367.

Jiayi Liu, Denys Iliash, Angel X Chang, Manolis Savva, and Ali Mahdavi-Amiri. 2024. SINGAPO: Single Image Controlled Generation of Articulated Parts in Object. *arXiv preprint arXiv:2410.16499* (2024).

Jiayi Liu, Manolis Savva, and Ali Mahdavi-Amiri. 2025. Survey on Modeling of Human-made Articulated Objects. In *Eurographics STAR (State of The Art Report), Computer Graphics Forum*.

Lin Lu, Andrei Sharf, Haisen Zhao, Yuan Wei, Qingnan Fan, Xuelin Chen, Yann Savoye, Changhe Tu, Daniel Cohen-Or, and Baoquan Chen. 2014. Build-to-Last: Strength to Weight 3D Printed Objects. *ACM TOG* 33, 4 (2014), 97:1–97:10.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*.

Niloy Mitra, Michael Wand, Hao Zhang, Daniel Cohen-Or, and Martin Bokeloh. 2013. Structure-aware shape processing. In *Eurographics State-of-the-art Report (STAR)*.

Niloy J. Mitra, Leonidas Guibas, and Mark Pauly. 2007. Symmetrization.

Niloy J Mitra, Yong-Liang Yang, Dong-Ming Yan, Wilmot Li, Maneesh Agrawala, et al. 2010. Illustrating how mechanical assemblies work. *ACM Transactions on Graphics-TOG* 29, 4 (2010), 58.

Luke Olsen, Faramarz F. Samavati, Mario Costa Sousa, and Joaquim A. Jorge. 2009. Sketch-based modeling: A survey. *Computers & Graphics* 33 (2009), 85–103. Issue 1.

Romain Prévost, Emily Whiting, Sylvain Lefebvre, and Olga Sorkine-Hornung. 2013. Make it stand: balancing shapes for 3D fabrication. *ACM TOG* 32 (2013), article 81. Issue 4.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. 2017. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems* 30 (2017).

Xiaowen Qiu, Jincheng Yang, Yian Wang, Zhehuan Chen, Yufei Wang, Tsun-Hsuan Wang, Zhou Xian, and Chuang Gan. 2025. Articulate AnyMesh: Open-vocabulary 3D Articulated Objects Modeling. *arXiv preprint arXiv:2502.02590* (2025).

Adriana Schulz, Ariel Shamir, David I. W. Levin, Pitchaya Sitthi-amorn, and Wojciech Matusik. 2014. Design and Fabrication by Example. *ACM TOG* 33 (2014). Issue 4.

Lavanya Sita Tekumalla and Elaine Cohen. 2004. A hole-filling algorithm for triangular meshes. *School of Computing, University of Utah, UUCS-04-019, UT, USA* 2 (2004).

Keneni W Tesema, Lyndon Hill, Mark W Jones, Muneeb I Ahmad, and Gary KL Tam. 2023. Point cloud completion: A survey. *IEEE Transactions on Visualization and Computer Graphics* 30, 10 (2023), 6880–6899.

Nobuyuki Umentani, Takeo Igarashi, and Niloy J Mitra. 2015. Guided exploration of physically valid shapes for furniture design. *Commun. ACM* 58, 9 (2015), 116–124.

Aditya Vora, Sauradip Nag, Kai Wang, and Hao Zhang. 2025. Articulate That Object Part (ATOP): 3D Part Articulation via Text and Motion Personalization. arXiv:2502.07278 [cs.CV] https://arxiv.org/abs/2502.07278

Kai Wang, Paul Guerrero, Vladimir G Kim, Siddhartha Chaudhuri, Minhyuk Sung, and Daniel Ritchie. 2022. The shape part slot machine: Contact-based reasoning for generating 3D shapes from parts. In *European Conference on Computer Vision*. Springer, 610–626.

Penghao Wang, Siyuan Xie, Hongyu Yan, Xianghui Yang, Jingwei Huang, Chunchao Guo, and Jiayuan Gu. 2026. ArtLLM: Generating Articulated Assets via 3D LLM. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2026).

Yizhi Wang, Wallace Lira, Wenqi Wang, Ali Mahdavi-Amiri, , and Hao Zhang. 2024. Slice3D: Multi-Slice, Occlusion-Revealing, Single View 3D Reconstruction. In *CVPR*.

Karl D.D. Willis, Pradeep Kumar Jayaraman, Hang Chu, Yunsheng Tian, Yifei Li, Daniele Grandi, Aditya Sanghi, Linh Tran, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. 2022. JoinABLe: Learning Bottom-up Assembly of Parametric CAD Joints. In *CVPR*.

Hongchi Xia, Entong Su, Marius Memmel, Arhan Jain, Raymond Yu, Numfor Mbiziwo-Tiapo, Ali Farhadi, Abhishek Gupta, Shenlong Wang, and Wei-Chiu Ma. 2025. DRAWER: Digital Reconstruction and Articulation With Environment Realism. In *CVPR*.

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. 2020. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 11097–11107.

Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. 2025. Structured 3D Latents for Scalable and Versatile 3D Generation. In *CVPR*.

Xingguang Yan, Liqiang Lin, Niloy J Mitra, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. 2022. Shapeformer: Transformer-based shape completion via sparse representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 6239–6249.

Zihao Yan, Fubao Su, Mingyang Wang, Ruizhen Hu, Hao Zhang, and Hui Huang. 2023. Interaction-Driven Active 3D Reconstruction with Object Interiors. *ACM Transactions on Graphics (Special Issue of SIGGRAPH Asia)* 42, 6 (2023).

Yunhan Yang, Yufan Zhou, Yuan-Chen Guo, Zi-Xin Zou, Yukun Huang, Ying-Tian Liu, Hao Xu, Ding Liang, Yan-Pei Cao, and Xihui Liu. 2025. Omnipart: Part-aware 3d generation with semantic decoupling and structural cohesion. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers*. 1–12.

Chongjie Ye, Yushuang Wu, Ziteng Lu, Jiahao Chang, Xiaoyang Guo, Jiaqing Zhou, Hao Zhao, and Xiaoguang Han. 2025. Hi3DGen: High-fidelity 3D Geometry Generation from Images via Normal Bridging. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 25050–25061.

Longwen Zhang, Ziyu Wang, Qixuan Zhang, Qiwei Qiu, Anqi Pang, Haoran Jiang, Wei Yang, Lan Xu, and Jingyi Yu. 2024. CLAY: A Controllable Large-scale Generative Model for Creating High-quality 3D Assets. *ACM Transactions on Graphics (TOG)* 43, 4 (2024), Article 120.

Wei Zhao, Shuming Gao, and Hongwei Lin. 2007. A robust hole-filling algorithm for triangular mesh. *The Visual Computer* 23, 12 (2007), 987–997.

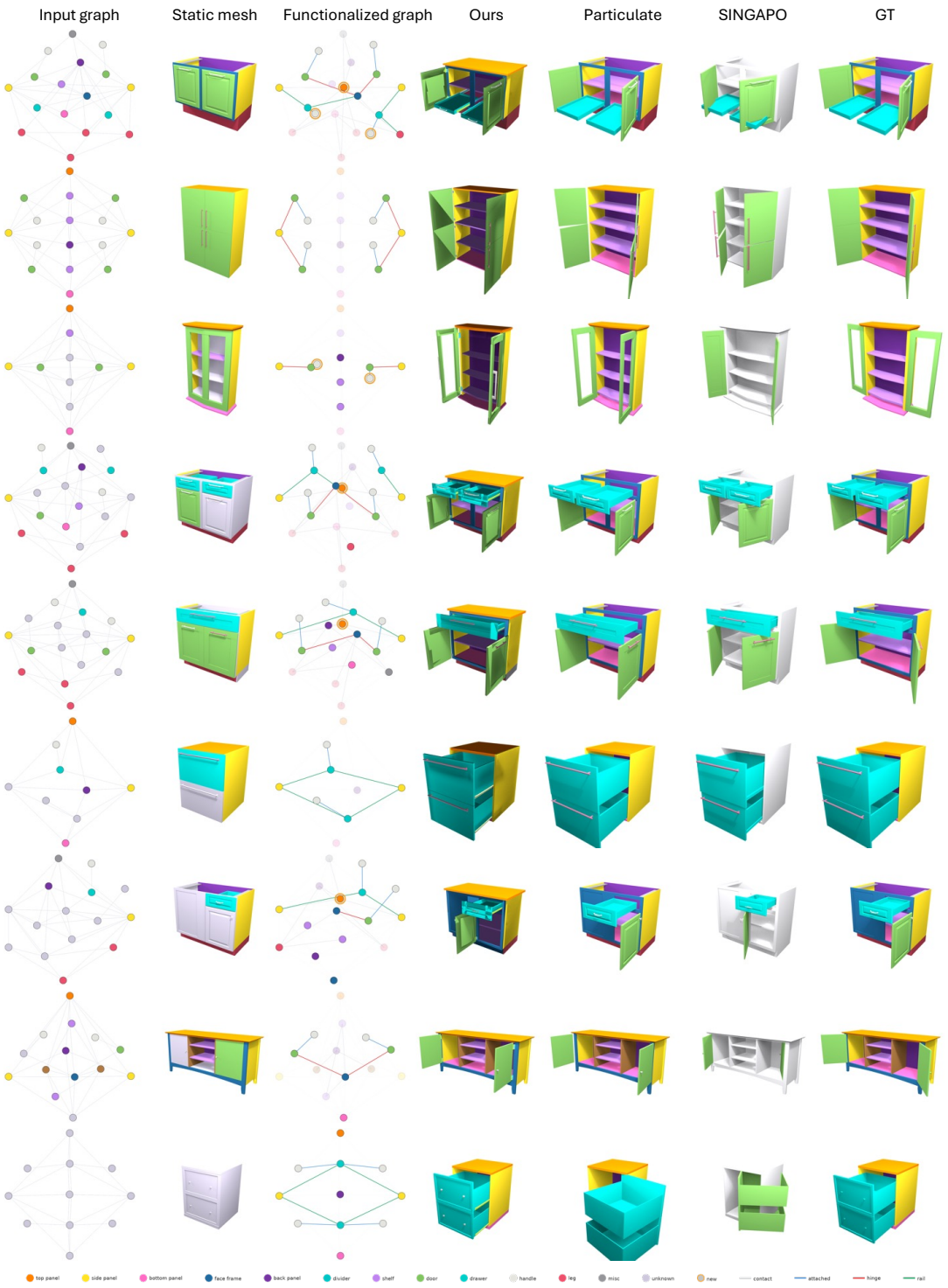


Fig. 5. Qualitative results and comparisons to Particulate and SINGAPO on test samples from PN-M (top 7 rows) and HSSD (bottom 2 rows). For our results, we mainly show GraFu graph completion with added nodes and edges highlighted in the functionalized graphs (third column). From top-down, the input graphs (left column) are increasingly "corrupted" with less nodes semantically labeled, shown as grey vs. colored nodes. All results are obtained from automatic functionalization executions.

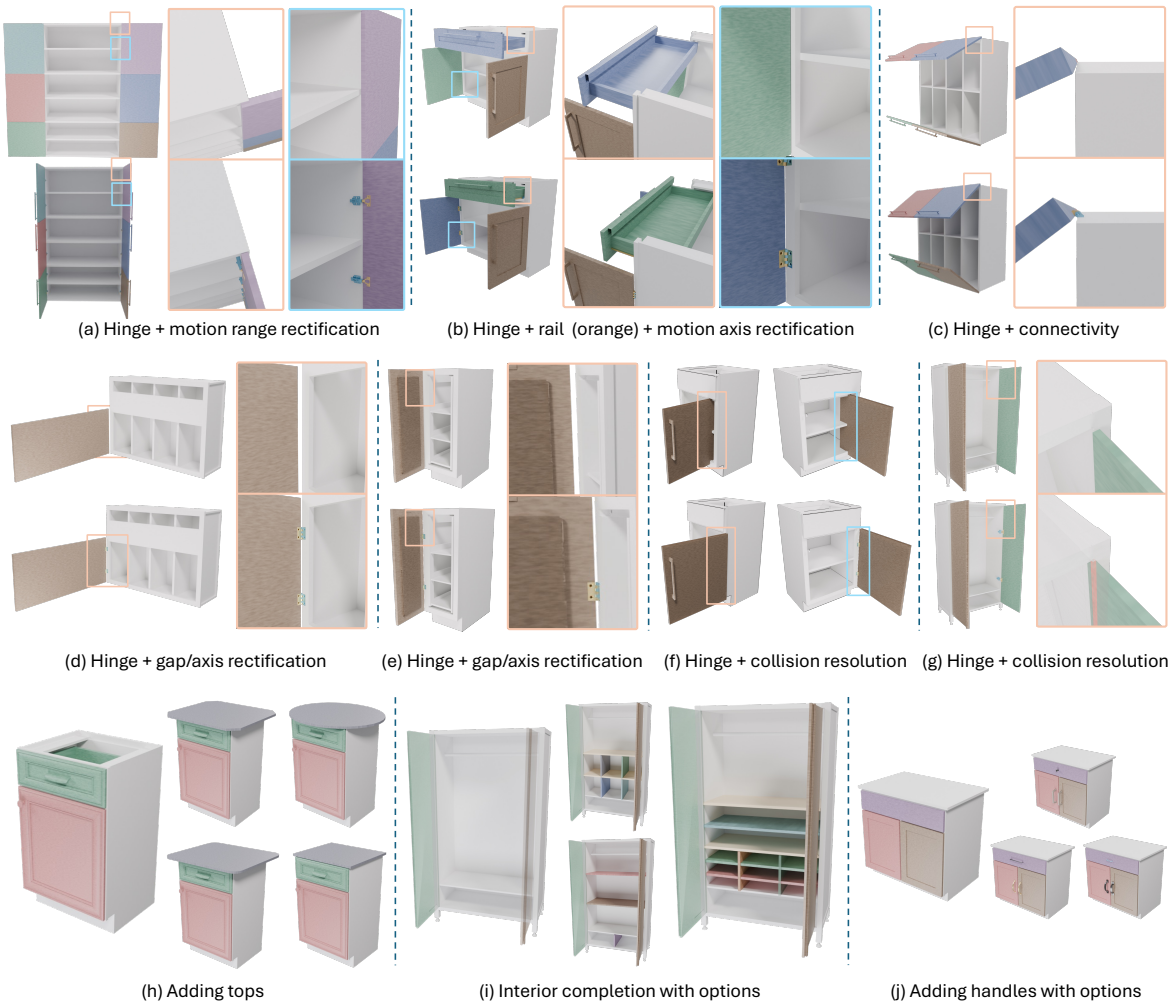


Fig. 6. A gallery of qualitative results from our object functionalization tool. Dynamic functionalization is shown in (a-g), with motion axis read from GT annotations and geometric fix automatically executed. The top is before functionalization and bottom is after. We highlight the fixes in both zoom-ins and sub-figure captions. Static functionalizations (h-j) with user-in-the-loop design are shown in the bottom row. Users can alter design styles of top panel, interior, and handles through the Blender Add-On.



Fig. 7. Dynamic and static functionalizations applied to a 3D model obtained from single-view 3D reconstruction by Omniparts [Yang et al. 2025]. The motion axes are read from human annotations, with automatically executed dynamic functionalization and user-designed interiors.

Functionalization via Structure Completion and Motion Rectification

(Supplementary Material)

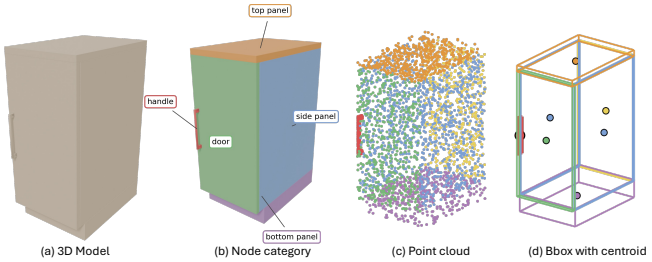


Fig. 8. The node attributes encoded by GraFu. For (a) an input model, its (b) per-part category, (c) surface point cloud and its (d) bounding box and centroids, are encoded and concatenated to form a node feature.

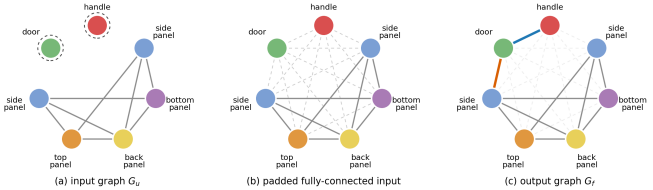


Fig. 9. Edge padding mechanism on input graphs.

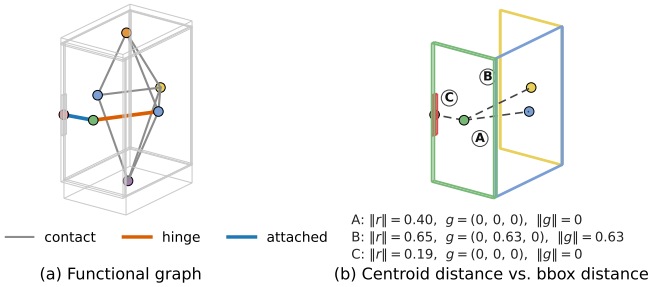


Fig. 10. Bounding box distance reflects faithful contact distance between nodes, providing complementary spatial awareness besides Euclidean centroid distance.

A GraFu Architecture

This section details the GraFu architecture. GraFu is composed of three modules: a node encoder, a graph encoder, and a graph decoder. Conceptually, the node encoder converts the per-part information, including category label, bounding box with centroid, and surface point cloud, into a compact latent. The graph encoder then relates these latents to one another with respect to the input graph topology. Finally, the graph decoder, conditioned on the encoded graph, decodes the final functional graph. Each module is detailed below.

A.1 Node Encoder

Figure 8 illustrates the three node attributes encoded by the node encoder. Each node represents a geometric part segmented out from the input mesh, the associated attributes are defined as follows:

- **Category**, from the preset list of 13 part labels plus an unknown type, $n_{\text{cat}} \in \{0,1\}^{14}$. The unknown type absorbs unlabeled parts, both from label-masking augmentation and from missing labels at test time.
- **Bounding box and centroid.** The axis-aligned bounding box and centroid of the part, which together capture its extent and position: $n_{\text{geo}} = [\, n_{\text{bbox}} \,\|\, n_{\text{centroid}} \,] \in \mathbb{R}^9$, where $n_{\text{bbox}} := (bbox_{\text{min}}, bbox_{\text{max}}) \in \mathbb{R}^6$ stores the two corner coordinates and $n_{\text{centroid}} \in \mathbb{R}^3$ the part's centroid. These give the network a coarse notion of part size and placement.
- **Point cloud**, 512 points sampled from the part surface to provide geometric cues, $n_{\text{pc}} \in \mathbb{R}^{512\times3}$.

n_{cat} is encoded by a linear projection into $\mathbb{R}^{128}$; n_{geo} by a two-layer MLP into $\mathbb{R}^{32}$; and n_{pc} by a lightweight two-stage PointNet++ [Qi et al. 2017] into $\mathbb{R}^{64}$. The three embeddings are concatenated into the node latent $z_v \in \mathbb{R}^{224}$.

A.2 Graph Encoder

The graph encoder allows the encoded node features to attend to one another with respect to the graph topology, yielding a relation-aware graph encoding. With node encoding described above, the design choices here concern how the inter-node relations (i.e., edges) are represented and learned. Each node pair is described by the following attributes:

- **Edge type.** $e_{cat} \in \{0,1\}^5$. One of five types in [`contact`, `hinge`, `rail`, `attach`, `null`], represented as a one-hot vector.
- **Node distance.** $r \in \mathbb{R}^3$. The Euclidean distance between the two centroids, together with the signed unit direction.
- **Bounding-box distance.** $g \in \mathbb{R}^3_{0+}$. The distance between the two nodes' bounding boxes along each axis, lower-bounded by zero under overlap.

We now elaborate on two design choices introduced above. First, the `null` edge: we pad the input graph to a fully connected graph by assigning `null` edges to the empty edge slots during graph encoding. This allows new functional edges to form on newly added nodes or disjoint node pairs at input. Edge creation reduces to a *reclassification* of `null` edge types rather than the insertion of new edge entries at inference time. An example is shown in Figure 9, where the two isolated nodes (a) at input time are padded with null edges (b), and later reclassified to functional edges upon inference (c).

Second, the bounding-box distance, which further consolidates spatial awareness between nodes. An example is shown in Figure 10, which shows the pairwise distance between the door (green) towards the side panel (blue), the back panel (yellow) and the handle (red). The centroid distance gives non-zero measurements for all three pairs, though the door is indeed contacting two of them at the rest state. Using the bbox distance reflects the correct distance rankings among these three pairs.

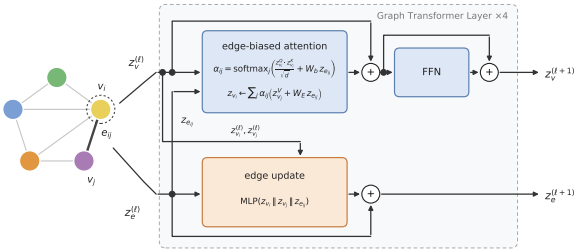
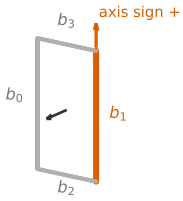


Fig. 11. Graph encoder architecture. It takes 4 cascaded graph transformer layers that cross-attend each pair of nodes based on their edge connection.



(a) Hinge revolute axis prediction (b) Rail prismatic axis prediction

Fig. 12. Motion prediction on hinge and rail functional edges. For hinge motion prediction, the revolute axis is classified as one of the four bounding box borders (gray borders) at the thin edge (black arrow), with additional sign prediction for the revolving direction. For rail edge prismatic axis prediction, the axis is classified between $\{\pm x, \pm y, \pm z\}$.

The edge type is embedded, the geometric term $[\,\|r\|, \|g\|, \hat{r}, g\,]$ is encoded by a small MLP, and their concatenation is projected to a 128-dimensional edge feature z_e. Together with the node features, z_e enters a stack of four Graph Transformer layers (Figure 11). Each layer is a pre-norm transformer block over all node pairs, made edge-aware at two points: z_e contributes a per-head bias $W_b z_e$ to the attention score and a term $W_E z_e$ to the attended value, so an edge steers both how strongly its endpoints attend to each other and what content they exchange. Each layer then re-encodes every edge from its two endpoint features, so node and edge representations refine each other layer by layer.

A.3 Graph Decoder

The graph decoder is a DETR-style [Carion et al. 2020] transformer decoder that decodes a fixed budget of M=55 slots into the output graph through four layers of slot self-attention and cross-attention to the graph encoding. The slots take three roles:

- **Anchored slots** (N, one per input node), initialized from the corresponding encoder feature; they always decode to a real node, since functionalization never removes input parts.
- **Free slots** (K=4), learned queries conditioned on a pooled summary of the graph encoding; they instantiate the missing parts.
- **Padding slots** ($M{-}N{-}K$), supervised to be non-existent.

The inference goal of the decoder is to, first, predict the node attributes for each slot, and second, predict the edge relations between predicted nodes.

Per slot, the decoder predicts node existence, category, centroid, and box half-extents. Per slot pair, it classifies the connecting edge into the five types from the two slot features and their pairwise geometric terms. For motion, GraFu classifies rather than regresses: a hinge's rotation axis is a 4-way border choice plus a 2-way axis sign in the door's thin-axis-aligned local frame, a drawer's prismatic axis is a 6-way signed axis (Figure 12). The discrete prediction serves only as a coarse initialization and is subsequently grounded by the geometry realization stage.

After all nodes are inferred, Hungarian matching is performed between predicted slots and ground-truth nodes, allowing loss calculation. The loss formulation and data augmentation are detailed in their respective sections of this supplement.

B Loss Function

Given a training pair (G_u, G_f) where G_u is the non-functional input graph and G_f is the ground-truth functional graph, GraFu decodes a slot graph $\hat{G} = \{\hat{s}_i\}_{i=1}^{M}$ with M output slots. The first N slots are *anchored* to the input nodes and the next K are *free* queries; the remaining $M{-}N{-}K$ slots are zero-padded (and supervised to be null). We optimize a weighted sum of per-slot, per-pair, and motion losses, where the per-pair terms are evaluated after Hungarian matching between the predicted slot set and the GT node set.

B.1 Per-slot Loss

Set assignment. All M slots are matched jointly to the GT nodes by the Hungarian assignment

$$\sigma = \arg\min_{\sigma} \sum_{i} \left[\, \lambda_{\text{cls}}\, \text{CE}(\hat{m}_i, m_{\sigma(i)}) + \lambda_{\text{pos}}\, \|\hat{c}_i - c_{\sigma(i)}\|_2 \,\right] \quad (2)$$

with $(\lambda_{\text{cls}}, \lambda_{\text{pos}}) = (1, 2)$, where m_i, c_i denote the GT category and centroid. The correspondence between the input nodes and the first N slots is preserved by the anchor supervision below, which makes the identity match cost-optimal and leaves the GT nodes absent from the input to be claimed by the free slots.

Attribute supervision. The attributes of each matched slot, including its category, centroid, and bounding box, should conform to the ground truth. Correct category prediction is essential for functional-graph reasoning, while centroid and box alignment keeps the Hungarian assignment faithful and stabilizes training. Three losses enforce this over the matched pairs:

$$\mathcal{L}_{\text{mat}} = \text{CE}\big(\hat{m}_i,\ m_{\sigma(i)}\big), \quad (3)$$

$$\mathcal{L}_{\text{cent}} = \big\|\hat{c}_i - c_{\sigma(i)}\big\|_2^2, \quad (4)$$

$$\mathcal{L}_{\text{bbox}} = \big\|\hat{h}_i - h_{\sigma(i)}\big\|_2^2, \quad (5)$$

where $\hat{h}_i, h_i$ are the predicted and GT box half-extents, and each term is averaged over the matched slots of a batch.

The attribute loss combines the three terms as

$$\mathcal{L}_{\text{attr}} = \mathcal{L}_{\text{mat}} + \mathcal{L}_{\text{cent}} + 0.5\, \mathcal{L}_{\text{bbox}} \quad (6)$$

Existence supervision. Existence supervision is what lets GraFu fire new nodes and hence perform structural completion. Three

terms address it at different granularities. A general existence loss applies to all M slots,

$$\mathcal{L}_{\text{exist}} = \text{BCE}\big(\hat{e}_i,\ \mathbb{1}\,[i \text{ anchored or matched}]\big), \tag{7}$$

Because anchored slots and padding slots occupies a large portion in all node sets, a dedicated free-slot term is added to signify existence signals from the K free slots.

$$\mathcal{L}_{\text{free}} = \text{BCE}_{\text{pw}}(\hat{e}_i, \cdot)\,,\ i \notin \text{anchored}$$

with a positive-class weight (pw) of at least 10; this is the model's primary signal that missing nodes must be added back. Finally, a cardinality regularizer calibrates the *total* amount of completion,

$$\mathcal{L}_{\text{count}} = \Big(\big|\textstyle\sum_{i\in\text{free}} \sigma(\hat{e}_i) - N^{\text{gt}}_{\text{free}}\big|\Big)^{1/2}, \tag{8}$$

where $N^{\text{gt}}_{\text{free}}$ counts the GT nodes not covered by any anchored slot. It penalizes over-firing and under-firing induced by incorrect free slots prediction.

The existence loss combines the three granularities as

$$\mathcal{L}_{\text{ext}} = \mathcal{L}_{\text{exist}} + 3\,\mathcal{L}_{\text{free}} + \mathcal{L}_{\text{count}} \tag{9}$$

Anchor supervision. Input nodes must be fully preserved through the graph-to-graph translation. Rather than hard-constraining the assignment, we let GraFu learn this softly through two pose-consistency terms that pull each anchored slot back onto its *input-node* pose,

$$\begin{aligned} \mathcal{L}_{\text{anchor}} &= \big\|\hat{c}_i - c_i^{\text{in}}\big\|_2^2, \\ \mathcal{L}_{\text{anchor-bbox}} &= \big\|\hat{h}_i - h_i^{\text{in}}\big\|_2^2, \end{aligned} \qquad i \in \text{anchored}, \tag{10}$$

whose targets are the input poses themselves, independent of the assignment.

The anchor loss is

$$\mathcal{L}_{\text{anc}} = 0.3\,(\mathcal{L}_{\text{anchor}} + \mathcal{L}_{\text{anchor-bbox}}) \tag{11}$$

and the per-slot loss aggregates the three groups,

$$\mathcal{L}_{\text{slot}} = \mathcal{L}_{\text{attr}} + \mathcal{L}_{\text{ext}} + \mathcal{L}_{\text{anc}} \tag{12}$$

B.2 Per-pair Loss

Edge classification. For every pair (i, j) of real slots, the decoder emits an edge-type distribution $\hat{r}_{ij}$ over five classes {`contact`, `hinge`, `rail`, `attached`, `null`}, with the GT edge matrix constructed through the matched correspondence σ. Because the fully connected pair set is dominated by `null` pairs while the functional edges are rare, the edge loss is a class-weighted cross-entropy,

$$\mathcal{L}_{\text{edge}} = \text{CE}_{\omega}\big(\hat{r}_{ij},\ r_{ij}\big), \qquad \omega = (1,\ 5,\ 5,\ 3,\ 0.2), \tag{13}$$

which suppresses the trivial null majority and emphasizes the `hinge`, `rail`, and `attached` classes that carry the functionalization.

Exclusive-relation supervision. Two relations are exactly one by nature: a handle attaches to exactly one door or drawer, and a door hinges onto exactly one static panel. The per-pair cross-entropy above calibrates each pair in isolation but never compares candidates against one another, whereas inference must select a single winner by an argmax over candidates. Two dedicated heads therefore score each candidate pair, ρ_{ij}, alongside a learned abstain logit $\rho_{\varnothing}$, and are supervised with a cross-entropy over the across-candidate softmax that inference selects with:

$$\mathcal{L}_{\text{parent}} = -\log \frac{\exp(\rho_{ij^*})}{\sum_{j\in C_i}\exp(\rho_{ij}) + \exp(\rho_{\varnothing})}, \tag{14}$$

where C_i are the candidate parents (door and drawer slots) of handle slot i and j^* its GT parent; $\mathcal{L}_{\text{ht}}$ is the analogue on the door side, over candidate static panels. Beyond ranking, the abstain logit gives the selection a calibrated "no parent" option for handles whose parent is absent from the prediction. The per-pair loss combines the terms as $\mathcal{L}_{\text{pair}} = \mathcal{L}_{\text{edge}} + 0.5\,\mathcal{L}_{\text{parent}} + 0.5\,\mathcal{L}_{\text{ht}}$.

B.3 Motion Loss

For each GT `hinge` edge and `rail` edge, the motion heads predict pose-invariant joint attributes as discrete classifications: the 4-way hinge border $\hat{f}_{ij}^{(4)}$, the 2-way axis sign $\hat{\epsilon}_{ij}^{(2)}$, and the 6-way signed rail axis $\hat{a}_{ij}^{(6)}$ for drawers:

$$\mathcal{L}_{\text{motion}} = \text{CE}\big(\hat{f}_{ij}^{(4)}, f_{ij}\big) + \text{CE}\big(\hat{\epsilon}_{ij}^{(2)}, \epsilon_{ij}\big) + \text{CE}\big(\hat{a}_{ij}^{(6)}, a_{ij}\big), \tag{15}$$

where the first two terms apply to hinge edges and the third to rail edges, each averaged over its edges in a batch.

C Loss Ablation

Setup. We probe the contribution of each supervisory signal by holding out one loss component (or a tightly coupled pair, when two losses jointly supervise the same quantity) on top of the winning configuration and retraining from scratch under identical optimisation settings (85 epochs, cosine-annealed learning rate with $T_{\max}$=450, same train/val/test split as the main model). The six variants we evaluate are:

- $-w_{\text{ht}}$: removes $\mathcal{L}_{\text{hinge\_target}}$. This loss trains the directed head that, for each door, picks exactly one static mounting panel; removing it disables the door-to-panel selection signal.
- $-w_{\text{parent}}$: removes $\mathcal{L}_{\text{parent}}$. This loss trains the directed head that, for each handle, picks exactly one parent door or drawer; removing it disables the handle-to-parent selection signal.
- $-w_{\text{cent}}$: removes the centroid loss inside $\mathcal{L}_{\text{attr}}$. This loss supervises slot centroids; removing it leaves bbox half-extents as the only direct geometry supervision.
- $-w_{\text{bbox}}$+$w_{\text{anchor\_bbox}}$: removes both bbox losses (free-slot and anchor-side). These jointly carry all bbox supervision; removing them leaves the model with no signal about part size.
- $-w_{\text{anchor}}$+$w_{\text{anchor\_bbox}}$: removes both anchor pose losses (centroid and bbox). These jointly tie anchored slots to their input-node poses; removing them lets the encoder freely drift the identity transform on existing parts.
- $-w_{\text{free\_exist}}$+w_{count}: removes the two free-slot firing supervision. Together, they regulate when and how many free slots fire; removing them frees the model to predict any number of hallucinated nodes.

We pair each loss with another only when both supervise the same underlying quantity from different angles, so that the ablation isolates *a function* (e.g. bbox supervision, anchor-pose preservation,

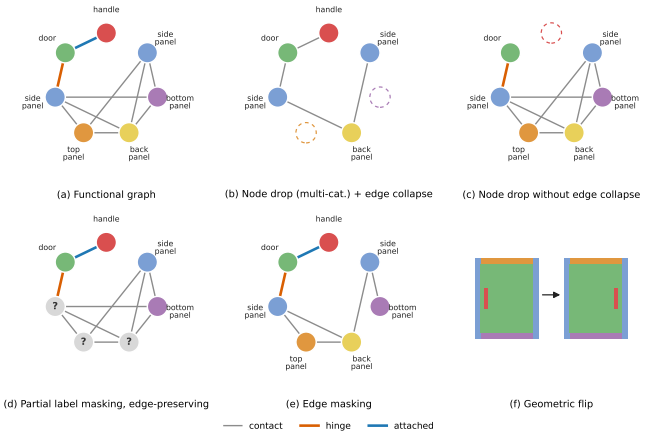


Fig. 13. Example data augmentation strategies.

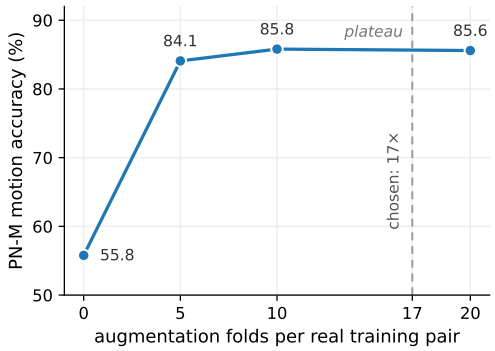


Fig. 14. Data augmentation fold ablation experiment.

or hallucination control) rather than an arbitrary single weight. Table 4 reports the resulting test-set metrics against the full setup

Conclusion. The full setup is the best overall configuration. Looking across Table 4, it attains the highest hinge edge F1, hinge axis-sign accuracy, and joint motion accuracy (the signature metrics for our task), and ranks at-or-near best on the remaining metrics. Importantly, no single ablation improves more than one metric in isolation while leaving the others unaffected, ruling out the possibility that any loss is redundant.

D Data Augmentation

Beyond the natural (G_u, G_f) pairs obtained from FurFun-233 (where G_u is the rest-state contact-only graph and G_f the functional counterpart), we synthesize additional input variants by corrupting the functional graph G_f via various structure-corrupting strategies plus a random geometric flip. Illustrative examples are shown in Figure 13. Per training step, a strategy is sampled from a weighted categorical distribution (Table 5); the sampled corruption is then applied to G_f to produce a synthetic G_u^{aug}, while the supervision target remains the original G_f. Empirically, we augment the unfunctional graphs to 17 folds for training.

Category drops. Five strategies remove all nodes of a specific structural category (handles, top panels, bottom panels, doors, drawers). Each forces the free-slot existence and material heads to recover the missing category from the surrounding contact graph and the global pooled summary fed into the free queries. A sixth strategy removes two or three of the above categories simultaneously, training the decoder to handle compound completion where multiple structural roles are missing at once. A seventh strategy randomly removes a handful of arbitrary nodes regardless of category. In all cases the dropped node's incident edges are deleted with it, and the surviving edges are relabeled as plain `contact`.

Label-masking corruptions. Two strategies anonymize the input labels at complementary severities. The first replaces *every* node's category with the *unknown* token, and with all of its edges relabelled as plain `contact`. The second masks a random 30–60% of the categories while preserving the surviving edges' functional types, pairing partial semantics with an intact relational structure. Both train the material head to recover part categories from geometry and topology without explicit semantic labels.

Edge-preserving partial-functional drops. Three strategies contribute to *edge-preserving* family: every surviving edge keeps its functional type, training the model on partial-functional inputs. They are:

- `mask_random_edges_keep_types` removes a random 25–50% of the edges while keeping all nodes;
- `drop_nodes_keep_edge_types` removes a random 30–50% of the nodes, keep the remaining edges;
- `fully_functional_drop` removes a random number of non-dynamic nodes and their incidental edges such that none of the functional edges are removed. All remaining edges are preserved.

Geometric flip. Independent of the corruption strategy, with probability $p = 0.5$ we apply a random axis-aligned mirror across X or Y. Coordinates and bbox extents are negated on the mirrored axis, point-cloud samples are flipped, and the motion labels are remapped. The mirror doubles the effective data and enhances the geometric diversity without collecting new data.

Sampling weights. Table 5 lists the categorical sampling weight of each strategy; weights are renormalized to a probability distribution at sampling time.

Validation. The 17-fold data augmentation is empirically chosen based on FurFun validation set performance. Here we conduct a post-validation on PN-M dataset, benchmarking on its GT-aligned motion accuracy. We train GraFu with 0, 5, 10 and 20-fold data augmentation and test on PN-M testset. The motion accuracy are 55.78%, 84.08%, 85.8% and 85.59%, respectively. The accuracy change is visualized in Figure 14. It can be seen that data augmentation provides substantial support in GraFu learning from 5-fold, and plateaued from 10 fold. Our choice of 17-fold is inbetween this plateaued range.

Table 4. Loss hold-out ablation on the FurFun test split. Each column zeros one loss term (or pair). **Bold** = best per row; ↓ / ↑ indicates desirable direction. Every variant degrades at least one signature metric, and no single loss is dispensable.

Metric	**Full**	$-w_{\text{ht}}$	$-w_{\text{parent}}$	$-w_{\text{cent}}$	$-w_{\text{bbox+anc.bbox}}$	$-w_{\text{anc+anc.bbox}}$	$-w_{\text{free\_exist+count}}$
Node existence F1 ↑	0.972	0.959	**0.974**	0.961	0.970	0.969	0.964
Material accuracy ↑	0.957	0.947	0.958	0.949	**0.961**	0.954	**0.961**
Centroid MAE (m) ↓	0.060	0.048	**0.046**	0.092	0.060	0.079	0.056
Bbox MAE (m) ↓	0.081	0.078	**0.069**	0.114	*0.298*	0.108	0.075
Edge acc. (real) ↑	0.770	0.637	0.649	0.646	0.725	0.704	**0.777**
Edge F1 contact ↑	0.884	0.836	0.858	0.835	**0.896**	0.844	0.892
Edge F1 hinge ↑	**0.656**	*0.409*	0.567	0.565	0.573	0.580	0.652
Edge F1 rail ↑	0.447	0.377	0.407	0.395	0.431	0.398	**0.479**
Edge F1 attached ↑	0.855	0.778	*0.004*	0.749	0.791	0.835	0.823
Hinge border-4 acc. ↑	0.922	0.918	0.930	0.877	0.926	0.866	**0.940**
Hinge axis-sign acc.↑	**0.975**	0.925	0.899	0.897	0.940	0.925	0.954
Rail axis acc. ↑	0.997	0.992	0.992	**1.000**	**1.000**	**1.000**	0.996
Joint motion (hb4×has) ↑	**0.899**	0.849	0.836	0.786	0.870	0.801	0.897

Table 5. Categorical sampling weights for augmentation strategies.

Family	Strategy	Weight
Category drop	drop_handles	0.15
	drop_top	0.15
	drop_bottom	0.10
	drop_doors	0.08
	drop_drawers	0.08
	drop_multi_cat	0.10
	random	0.05
Label masking	fully_anonymized	0.05
	mask_materials_keep_edges	0.04
Edge-preserving	mask_random_edges_keep_types	0.10
	drop_nodes_keep_edge_types	0.05
	fully_functional_drop	0.20

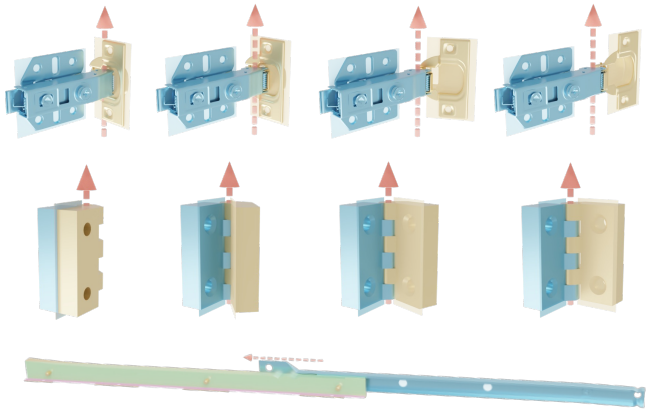


Fig. 15. Mechanical part annotations. Row 1: P-hinge with non-linear motion; Row 2: C-hinge with linear revolute motion. Cyan and gold indicate geometry and snap plane for static and dynamic parts, respectively. Row 3: Sliding rail. Magenta and lime indicate two snapping planes.

E Geometric Fix Algorithms

Given the functional graph predicted by GraFu, the geometric fixing stage grounds it in 3D. *Dynamic* functionalization inserts real mechanism templates for every predicted `hinge` and `rail` edge, rectifying the motion axis in the process; *static* functionalization repairs non-moving defects, including missing countertops, absent handles, and empty interiors.

E.1 Dynamic Functionalization

Rather than inferring mechanical parameters from data, we use animated hinge and rail templates, each annotated once under a shared protocol. This turns the continuous placement problem into a *discrete selection* of a template per installation condition, followed by *geometric fitting*.

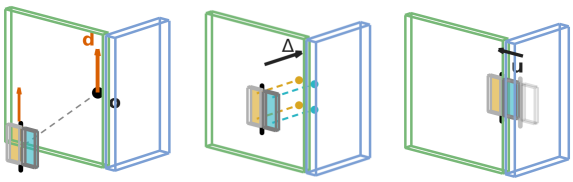
E.1.1 Template Annotation. All connectors follow the same annotation protocol, so the annotations transfer to new template instances. Each template is partitioned into a *static end*, which contacts the static mounting panel, and a *dynamic end*, which mounts on the movable part. The annotation comprises two elements (Figure 15): a **motion axis** and a pair of **snap planes** $\{\Pi_s, \Pi_d\}$, one per end.

For hinges, the motion axis lies on the pin, oriented by the right-hand rule so that the positive direction indicates counter-clockwise rotation; for rails, it indicates the sliding direction. Each snap plane is a zero-thickness rectangle covering its mounting region; during snapping, its distance to the target mounting surface serves as the fitting objective.

These elements constitute the minimum annotation; templates with specific installation rules carry additional planes, such as the magenta plane in Figure 15, which encodes that rail installation requires exact bottom support beneath the drawer.

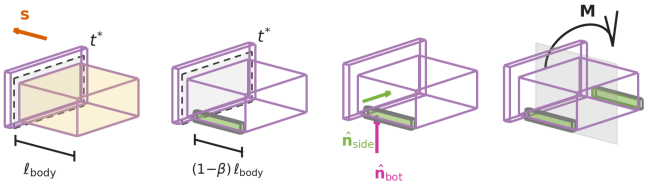
E.1.2 Revolute Motion: Hinge Attachment. Physically plausible revolute motion requires bilateral contact: the hinge must meet both the static part $\mathcal{S}$ and the movable part $\mathcal{M}$ along geometrically compatible surfaces.

We first select a compatible hinge class from the spatial relation between $\mathcal{M}$ and $\mathcal{S}$, highlighted in Figure 3 of the main paper. Each



(a) Pose initialization (b) Search-and-snap (c) Co-border alignment

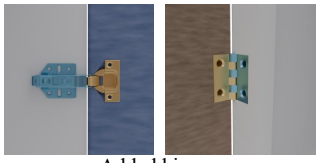
Fig. 16. Hinge insertion procedures



(a) Body Detection (b) Scale (c) Snap (d) Mirror

Fig. 17. Rail insertion procedures

class implies a different pair of snapping targets: a C-hinge seeks contacting faces, an F-hinge seeks flush faces, and a P-hinge seeks perpendicular faces, all detected on bounding-box proxies of $\mathcal{M}$ and $\mathcal{S}$. When several classes apply, we prefer C- over F- over P-hinges.



Added hinges

Insertion is initialized from a motion axis $\mathbf{a} = (\mathbf{o}, \mathbf{d})$, with origin $\mathbf{o}$ and direction $\mathbf{d}$. This can be read from GraFu predictions, GT annotations, or other motion-prediction networks (e.g., [Li et al. 2025; Liu et al. 2024]). The axis serves only as initialization and a coarse estimate suffices.

Fitting proceeds in three steps, shown in Figure 16.

Pose Initialization. The template is translated so its pin passes through $\mathbf{o}$ and rotated by the minimal rotation aligning its motion axis with $\mathbf{d}$. A flip about the axis is applied when needed so that the Π_s faces $\mathcal{S}$ and the Π_d faces $\mathcal{M}$, and the template is scaled so that the hinge leaves do not protrude from their mounting faces.

Search and Snap. We then snap the annotated planes onto the actual meshes. Let $\mathbf{x}_s, \mathbf{x}_d$ denote the snap-plane centroids, $\mathbf{n}_s, \mathbf{n}_d$ their normals, and $\mathbf{p}_s, \mathbf{p}_m$ the contact points located on $\mathcal{S}$ and $\mathcal{M}$ by casting rays along the plane normals. The snap translation $\boldsymbol{\delta}$ solves

$$\boldsymbol{\delta}\cdot\mathbf{n}_s = (\mathbf{p}_s - \mathbf{x}_s)\cdot\mathbf{n}_s, \quad \boldsymbol{\delta}\cdot\mathbf{n}_d = (\mathbf{p}_m - \mathbf{x}_d)\cdot\mathbf{n}_d, \quad \boldsymbol{\delta}\cdot\mathbf{d} = 0. \tag{16}$$

The two contact constraints leave $\boldsymbol{\delta}$ free along $\mathbf{d}$, which indicates the hinge's position along the door border. The third constraint removes this freedom; the axial position is instead set explicitly, placing the i-th of N hinges at height $i/(N{+}1)\cdot h$ on a door of height h.

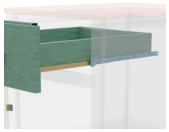
Co-bordering. When $\mathbf{n}_s \parallel \mathbf{n}_d$, the two contact constraints in eq. (16) become linearly dependent, leaving one more free direction $\mathbf{u} = \mathbf{n}_s \times \mathbf{d}$, in the mounting plane and perpendicular to the axis. We resolve it by aligning borders. Writing $e_{\mathbf{o}}(X)$ for the extreme of $\{\, v\cdot\mathbf{u} : v \in X \,\}$ nearest the pin projection $\mathbf{o}\cdot\mathbf{u}$, the correction is

$$\Delta_{\text{co-border}} = \big(e_{\mathbf{o}}(\partial\mathcal{M}) - e_{\mathbf{o}}(\partial\Pi_d)\big)\,\mathbf{u}, \tag{17}$$

which brings the dynamic snap plane's pin-side edge onto the movable part's hinge-side border, doing so places the pin exactly at the shared edge of the movable and static parts.

E.1.3 Prismatic Motion: Sliding Rail Attachment. A drawer slides on a pair of rails mounted between its container body and the cabinet walls. Placing them reduces to three questions: how long the rail may be? Where does it contact the drawer? How does the pair stay symmetric? The proposed fitting solution proceeds in four steps (Figure 17), followed by an optional support block.

Body-Region Detection. The rail length should be proportional to the drawer length, capped behind the pulling board. The rail length determines the motion range of the drawer, and it should conform to the container's length. The first step is to determine the container body region. A dense ray grid over the drawer's flank yields its outer-width profile $W(t)$ along the sliding axis $\mathbf{s}$ (Figure 17a). The body width is the profile's median, and the pulling board contributes only as outliers.



Added rail

$$T_{\text{body}} = \big\{\, t : |W(t) - \text{med}_{t'}\, W(t')| < \tau_W \,\big\}, \tag{18}$$

with τ_W a geometric tolerance. Its boundary t^* toward the pulling board separates body from board, and ℓ_{body} is the body's length along $\mathbf{s}$.

Scaling. The template is rotated minimally to align its prismatic axis with $\mathbf{s}$, rolled about $\mathbf{s}$ so its static snap plane faces the cabinet wall, and scaled uniformly to $(1-\beta)\,\ell_{\text{body}}$ (Figure 16b). Empirically, we set $\beta = 0.1$. The end of the rail template is aligned with the drawer end.

Snapping. Exact contact is recovered by directional from snap plane normals, analogy to hinge template snapping (Figure 16c). For rails with bottom support, the equation write as

$$\mathbf{n}_{\text{bot}} = +\hat{\mathbf{z}}, \qquad \mathbf{n}_{\text{side}} = \pm(\mathbf{s}\times\hat{\mathbf{z}}), \tag{19}$$

The dynamic snap plane contacts the drawer's flanks and the bottom-support plane contacts the underside of the drawer bottom. Every cast restricted to T_{body} so pulling-board geometry cannot bias the fit. Center-mount templates carry no bottom support plane and are centered vertically on the drawer side instead.

Symmetric Mirroring. We optimize one rail only; the other is its reflection across the drawer's sagittal plane (Figure 17d),

$$\mathbf{M}_{\text{mirror}} = \mathbf{I} - 2\,\mathbf{n}_\perp\mathbf{n}_\perp^\top, \qquad \mathbf{n}_\perp = \mathbf{s}\times\hat{\mathbf{z}}, \tag{20}$$

which enforces exact bilateral symmetry.

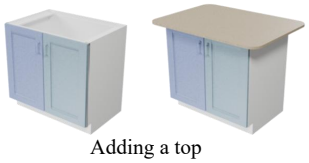
Lateral Support Block. A rail needs a mounting point behind its static snapping plane, which the cabinet does not always provide flush. When a lateral gap remains, a thin support block is extruded from the static snap plane to the wall; rails already contacting with the wall skip this step.

E.2 Static Functionalization

Static repairs missing structural parts such as countertops, handles, and interiors such as shelves and dividers.

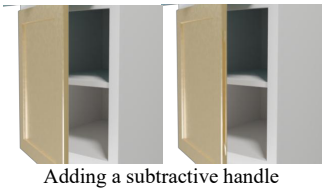
E.2.1 Adding Countertops. We frequently observe base cabinets in PartNet-Mobility [Xiang et al. 2020] that are modeled without an explicit countertop. This is often reasonable when the cabinet is intended to be placed beneath a shared counter surface or paired with appliances (*e.g.*, a stove). However, when the asset is considered in isolation, the missing top surface makes it unusable as a standalone cabinet. We therefore treat *missing countertops* as a static malfunction and repair them as part of our functionalization pipeline.

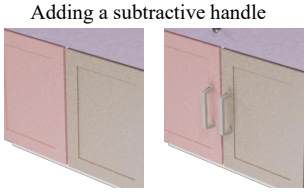
To check if an object has a countertop, from a top view, we look at how much space it covers on the floor and how much of that space is covered near the top of the object. If a large part of the floor area is also covered near the top, we say a countertop exists. Algorithmically, we project the mesh onto the ground plane and create two binary occupancy maps: $\mathbf{O}_{all}$ for the entire mesh and $\mathbf{O}_{top}$ for geometry near the maximum height. We then compute the coverage ratio $c = \texttt{area}(\mathbf{O}_{top})/\texttt{area}(\mathbf{O}_{all})$. If c exceeds a threshold α, we declare a countertop present; otherwise, it is considered missing. When a countertop is missing, we generate one by first estimating a planar outline and then extruding it into a slab. The outline is obtained by slicing the furniture near the top, fitting an axis-aligned rectangle to the rim points (with optional overhang), and extruding it by a user-defined thickness to form the final panel.


Adding a top

E.2.2 Adding Interaction Elements. Interaction elements refer to geometries that enable a user to grasp and operate movable parts. We consider two complementary types. *Additive* elements correspond to externally attached handles that protrude from the drawer/door surface. *Subtractive* elements correspond to *recessed* grips, where a portion of the drawer/door geometry is carved to form a finger groove (with no external attachment).

When *attaching* interaction elements, our pipeline handles both additive handles and recessed grips. The main steps are choosing a placement region and setting handle orientation and size. For drawers, additive handles go at the front center, and recessed grips are carved on a chosen side. For doors, we use the motion axis to find the hinge side, placing handles on the opposite side, additive at mid-height with an offset, recessed carved into the panel thickness.


Adding a subtractive handle


Adding additive handles

For additive handles, we add a small *snap plane* to define the mounting position. During attachment, the handle is oriented toward the target, rays are cast from the snap plane to the surface, and the handle is moved until it sits flush. The handle is also scaled to match the part. For recessed grips, we subdivide the target face and deform vertices into a U-shaped groove, with profile (sharpness, depth, size) controllable by users.

E.2.3 Interior Generation. While the previous components address exterior defects, interior structure equally defines the functionality of cabinet-like furniture(e.g., wardrobes), yet is frequently missing from both human-made repositories [Deitke et al. 2023; Xiang et al. 2020] and automatically generated models [Yang et al. 2025]. We repair it in two steps: detect the empty interior space as a set of box-shaped *compartments*, then furnish selected compartments with shelves and dividers.

Compartment Detection. A compartment is a maximal empty box inside the storage furniture. The predicted functional graph makes finding them direct: the labelled wall panels bound the interior cavity by their inner faces, and the labelled interior structure, including shelves, dividers, and drawers, marks the space already taken. Subtracting the occupied regions from the cavity and merging what remains into maximal axis-aligned boxes yields the compartments; slots too small to furnish are discarded.


Adding interiors

Panel Generation. Each selected compartment hosts N equally spaced panels of user-set thickness, horizontal for shelves and vertical for dividers, with an optional inset from the cabinet front. Generation is per compartment, so different compartments can host different configurations within the same asset.

This supports generating interiors from scratch as well as augmenting existing ones. Although we do not propose an explicit interior detector, compartment detection itself indicates whether an asset contains meaningful interior capacity. The primary limitation is the box representation: irregularly shaped interiors are not directly modeled.

F Per-case Evaluation

Table 2 of the main text reports quantitative results on a per-part basis. Here we complement it with a per-model evaluation on the PN-M test set.

F.1 Motion Prediction

Of the 345 PN-M test models, 266 (77.1%) are assigned motions that align with the ground-truth annotation, and a further 62 (18.0%) receive reasonable motions that differ from the annotation; 17 (4.9%) are incorrect. Figure 18 shows eight randomly selected GT-aligned examples, and Figure 19 six randomly selected examples that are plausible but not GT-aligned, along with the reference models that allow the prediction to be marked as reasonable.

Figure 20 shows five representative failure cases in three groups: motions outside our current taxonomy (e.g. sliding doors), incorrect functional-edge predictions, and models with very large node counts, which strain the inference in general.

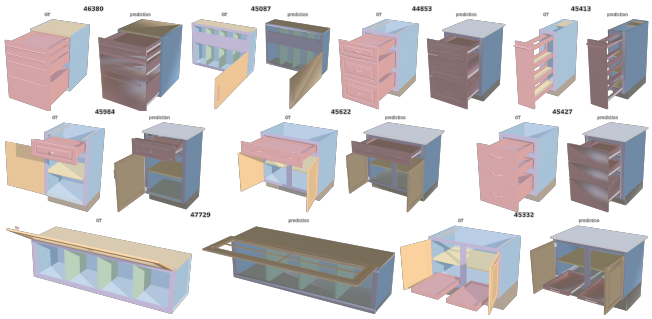


Fig. 18. GT-aligned motion prediction result showcase. For each pair, GT is on the left, and the prediction is on the right. Model ID is shown above.

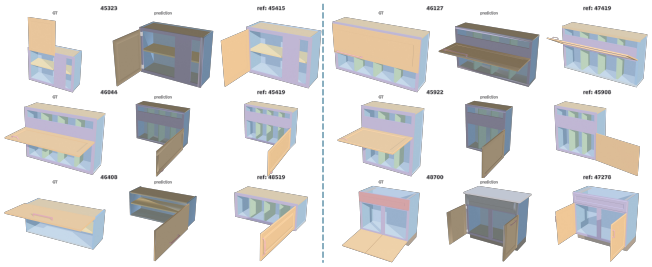


Fig. 19. Reasonable motion prediction result showcase. For each grid, left: GT; middle: GraFu prediction; right: reference GT sample used to identify the reasonable motion. Model ID is noted above.

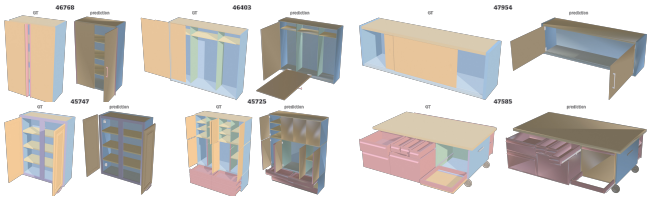


Fig. 20. Motion prediction failure cases. Our mechanical joint template gallery does not yet support sliding door motion (Row 1). Other cases include incorrectly formed functional edges between the dynamic node and the mounting static node, leading to incorrect geometric instantiation (Row 2 Column 1); and the high node-count models where some dynamic nodes are left static.

F.2 Top Panel Completion

Missing top panels are the most frequent functional deficiency in PN-M. GraFu detects and repairs them automatically using the prior learnt from the disentangled FurFun-233 dataset, with near-perfect accuracy. Figure 21 shows 100 pairs randomly sampled from the 164 completed models.

F.3 Handle Insertion

When handle nodes are already present in the input graph, they act as a prior for motion prediction. The position of a handle indicates how the corresponding movable part actuates. Row 7 of Figure 5 in the main text illustrates this behaviour, where GraFu predicts the door to open away from the handle. This is the more natural motion in practice even though it contradicts the GT annotation. When no handle is present, GraFu predicts the door motion without this cue, and we insert a handle consistent with the predicted axis.

Figure 22 presents a gallery of newly predicted handle nodes together with their automatic installation. In the Blender add-on, users can further select the handle style, choose an additive or subtractive form, add or remove handles, and adjust handle positions.

We next evaluate handle completion quantitatively on the PN-M test set. We regard a movable part as interactable once it is coupled with at least one handle. On input, 310 (89.9%) models are fully equipped with additive handle nodes; after GraFu prediction this rises to 329 (95.4%). The remaining 16 (4.6%) cases have two causes. First, the model currently issues $K = 4$ free-slot queries, which caps the number of handle nodes that can be instantiated on models with many movable parts. Second, cross-distribution generalization: five cases contain recessed drawer units behind a door, a structural and geometric configuration not represented in the training data. FurFun also contains a large proportion of subtractive handles, such as recessed sockets and grooves, so no strong prior that every movable part must carry an additive handle is enforced during training.

Ideally, the model would probe the geometry of a movable part to determine whether an interactable feature, additive or subtractive, is already present, and insert a new handle only when it is not. The current design does not operate at this geometric granularity, and we regard this as a promising direction for future work. Representative failure cases are shown in Figure 23.

F.4 Interior Generation

We expose interior generation as a user-in-the-loop feature, driven entirely by our interior completion algorithm. Figure 6(i) of the main text shows user-designed results obtained through the Blender add-on; below we evaluate the fully automatic setting on PN-M.

The algorithm first detects the compartments available in a model and then inserts divider or shelf geometries into them. We run it on all PN-M test models, inserting one to three dividers or shelves at random into each detected compartment; Figure 24 shows the resulting completions. Of the 345 models, 77 contain no compartment suitable for insertion and are correctly identified as inapplicable. Among the remaining 268, the algorithm produces reasonable completions for 264 (98.5%). The four exceptions have irregular compartment shapes that require non-rectilinear geometry (Figure 25), which the current geometry library does not yet provide.

F.5 Out-of-distribution Graphs

We further stress-test the pipeline on challenging input graphs, ones that deviate substantially from the training distribution. For each graph, a feature vector is constructed from the total node count, the per-category node counts, the average node degree and the overall geometric bounding box, and measure the disparity between two graphs as the Euclidean distance between their feature vectors.

For each evaluation set, PN-M and HSSD, we report the five test models with the largest z-scored distance from the corresponding training distribution, shown in Figures 26 and 27 together with their nearest training-set neighbour on the right. These cases are the most challenging in their respective test sets, and the residual errors are correspondingly concentrated here, namely dynamic nodes that



Fig. 21. 100 randomly sampled PN-M models with missing tops. In each cell, top: input PN-M model; bottom: functionalized model with mechanical parts inserted and top panel added. Please zoom in for details.

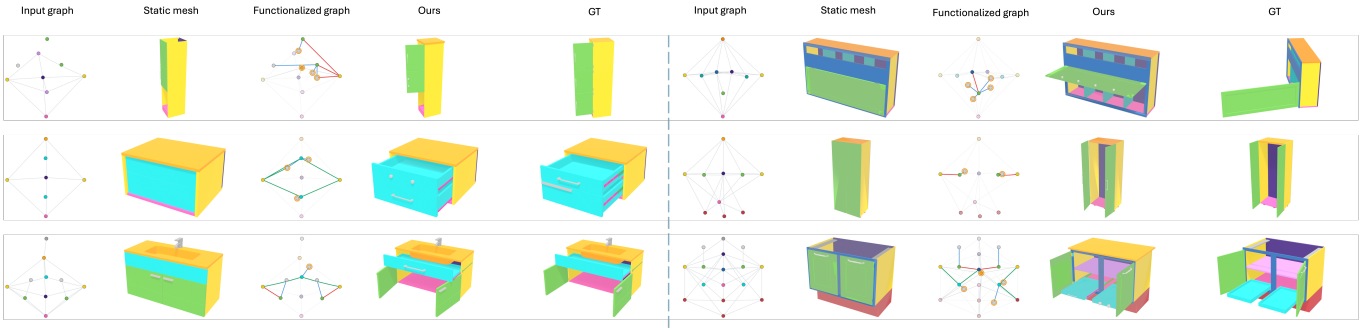


Fig. 22. Gallery of handle insertion results. All new handle nodes are predicted by GraFu and installed automatically. Please zoom in for details.

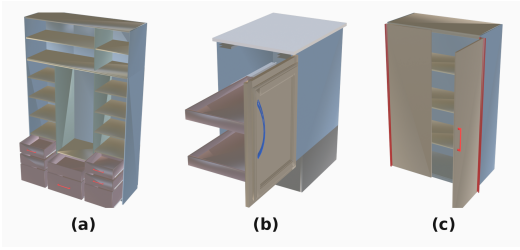


Fig. 23. Representative failure cases in handle prediction; existing handles are rendered in blue and newly added handles are rendered in red. (a) The number of required new nodes exceeds the free-slot budget $K = 4$. Only 4 new handles are added and the rest are handle-less (b) Unusual, out-of-distribution structures such as recessed drawer units behind a door. The two recessed drawers miss the new handle nodes. (c) Incorrect motion prediction. The static door did not have new handle inserted due to lack of prior motion.

are left static and occasional incorrect motion predictions. Overall, even at extreme node counts and far from the training distribution, the pipeline generalizes well, recovering most functional relations and instantiating the corresponding mechanical joints correctly.

G Blender Add-on

We provide a Blender add-on that wraps the geometric realization stage into an interactive tool: automatic functionalization first, per-joint refinement after. From a single prediction the user obtains a fully functionalized, animated model in one pass, then inspects, switches, or refines every installed connector and synthesized node directly in the 3D viewport.

Pipeline integration. The add-on consumes the artifacts produced by inference: the predicted functional graph and the per-part meshes. Loading a model imports the parts coloured by predicted category and automatically installs the predicted hinge, rail, top panel and handles. The interior can be installed on request. The output is an animated `.blend` file whose doors and drawers animate through the installed mechanisms over a closed–open–closed cycle, and with structure completed.

Hinges. Eight templates span the three mechanical classes of C-, P- and F-hinges. Per joint, the user keeps the default collision-tested class selection or forces a class, switches the template variant, and adjusts scale and the number of hinges per door.

Rails. Each drawer offers a centre-mount or corner-mount rail pair, optionally backed by support slabs or dividers that bridge the rails to the cabinet walls.

Handles. Handles are managed per parent door or drawer, including parts for which even no new handle nodes was predicted. The user chooses among 18 templates (bar, knob, and cup styles) or a recessed carved form, and sets the count and placement offsets; multiple handles distribute at equal partitions and shrink automatically to fit. Placement takes the predicted motion as a prior, mounting door handles opposite the hinge axis and centring drawer handles on the front face, so no manual snapping is required. Handles are automatically installed when a new node is predicted, and users can reject the install, or initiate new installations on movable parts that have no new handle nodes predicted.

Tops. A coverage test reports whether a top panel is missing; a single click then synthesizes one above the silhouette of the body parts, matching the cabinet footprint.

Interior. Interior structure completion detects compartments that appear as unoccupied boxes, and the user furnishes selected ones with a chosen orientation, panel count, thickness, and front inset. Different compartments can host different configurations within the same session.

H Comparison with LLM-based method

We compare our geometric realization algorithms against BlenderMCP, a tool that connects a vision-language model (VLM) to Blender. We use Claude 4.5 Sonnet as the VLM, and prompt it to functionalize the object under a range of instructions with different amounts of detail. We show the results in Figure 28. When prompted only with "functionalize the object", the VLM fails to apply any meaningful edits. We thus exclude these results. When prompted with the same edit types as ours, the VLM still fails to make accurate edits, leading to inaccurately positioned parts, as well as floating geometry (a-d, left image of e). Only after additional detailed instructions can the VLM perform satisfactory edits (middle image of e). In contrast, our method performs the edits consistently.

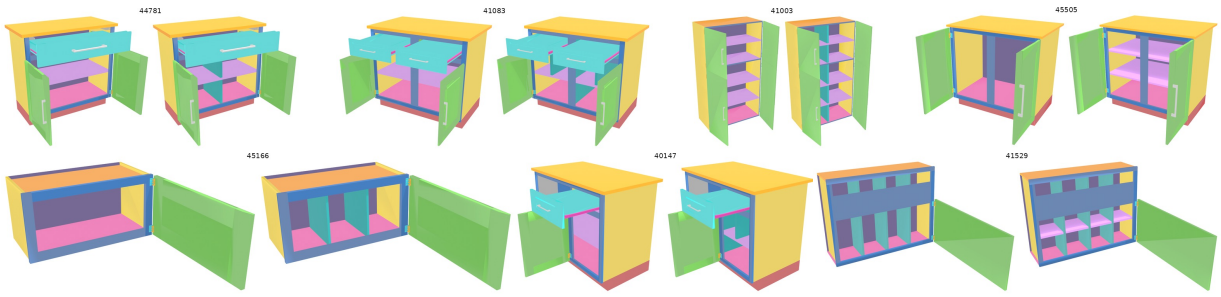


Fig. 24. Gallery of interior completion results. In each pair, the original model is shown on the left and the automatic interior completion on the right.

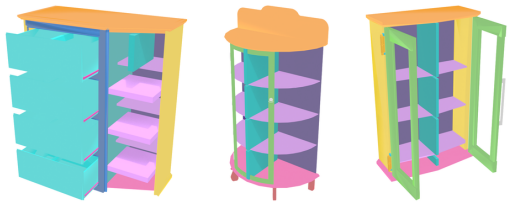


Fig. 25. Our interior completion algorithm does not yet support irregularly shaped shelves and dividers, such as circular or slanted ones.

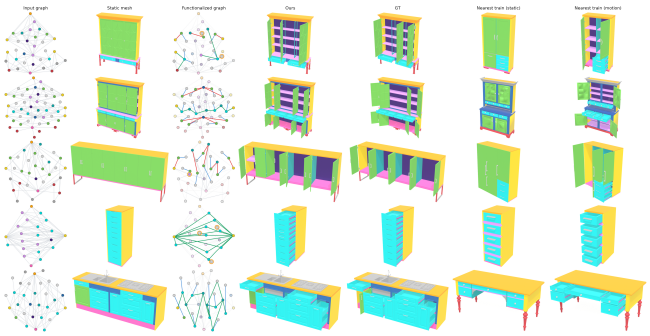


Fig. 26. Top five highest-disparity input graphs from the HSSD test set.

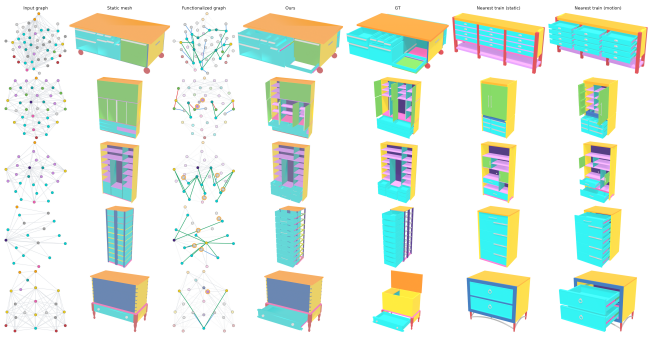


Fig. 27. Top five highest-disparity input graphs from the PN-M test set.

We further tested on Google Gemini on graph completion results. We provide 20 ground truth functional graphs as the context, and provide detailed instructions in defining nodes, edges and functionality criteria. Upon inference, we provide an unfunctional graph and a model's render, then explicitly query for graph completion. The commercial VLM performed poorly even with sufficient context, with the returned graph containing many isolated nodes.

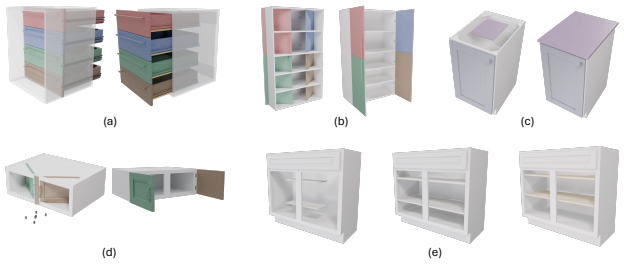


Fig. 28. Comparison against BlenderMCP. Left: BlenderMCP, right: Ours. (a) Adding rails to the drawer. (b)(d) Adding hinges to the cabinet. (c) Adding countertops. (e) Adding interior shelves. For each case, multiple prompts are provided from coarse "functionalize it" to detailed step-by-step instructions. (e) shows two BlenderMCP attempts, where the middle case undergoes iterative prompt tuning.