HiFi-BRep: High-Fidelity Latent Representation for Robust B-Rep Generation

Supplementary Material

A. Implementation Details

This section details the training configurations for our model. Unless stated otherwise, the optimizer, scheduler, and precision settings apply to both the VAE and the DiT components. All attention modules use 12 heads by default. We train in mixed precision (`bfloat16`) and perform inference in `float32`. The latent diffusion model is trained with DDPM for 1,000 diffusion steps using `beta_start`= 1e−4, `beta_end`= 2e−2, the `squaredcos_cap_v2` schedule, and `prediction_type` set to `sample`. At sampling time, we adopt DDIM [33] with 400 steps. Optimization uses AdamW with learning rate 1e−4, weight decay 1e−2, $(\beta_1, \beta_2) = (0.9, 0.99)$, and $\epsilon = 1\text{e}{-8}$. For the objective in Sec. 3.4, we set the loss weights to $(\lambda_{\text{KL}}, \lambda_{\text{len}}, \lambda_{\text{geom}}, \lambda_{\text{adj}}) = (5\times10^{-5}, 1, 25, 5)$. The learning rate warms up linearly over the first 10% of updates from $0.01\times$ the base rate to the base rate, and then follows a cosine decay back to $0.01\times$ the base rate for the remaining steps.

B. Novelty Verification

To assess whether *HiFi-BRep* synthesizes genuinely new shapes rather than recalling training instances, we perform a retrieval–based novelty check following the protocol of DTGBrepGen [20]. For each of 500 randomly generated B-reps, we compute Chamfer Distance (CD) on uniformly sampled surfaces and Light Field Distance (LFD) [3] on multi-view renderings against the entire training corpus, and retrieve the two nearest neighbors under each metric. As illustrated in Fig. 7, the retrieved training shapes remain close under CD/LFD yet exhibit clear differences in patch layout, hole patterns, and junction geometry compared with our generations. These side-by-side comparisons indicate that the model is not merely reproducing training exemplars. Instead, the single-stage validity-aware decoding paired with the high-fidelity latent supports novel but plausible variations within the data manifold.

C. Conditional Generation

We study conditional generation to test whether our model can follow guidance while keeping valid topology. Conditions are injected in the diffusion denoiser with Adaptive LayerNorm (adaLN) as in DiT [27]. Each condition is first encoded to a vector and then used by adaLN to steer the denoising process.

We use the Furniture dataset [40] for class conditioning because it has ten balanced categories with clear visual differences. For class labels, we map each of the ten categories to a 768-dimensional embedding vector. We use CADNet40 [7] for all other conditional generation experiments since it contains realistic industrial parts and supports practical control tasks with a moderate data size. For point clouds, we train a PointNet++ [28] encoder to produce a 768-dimensional feature from 2,048 points. For partial point clouds, we drop approximately 30% of points in a local region to mimic missing scans and use the same encoder. For single view images and wireframe sketches, we use a pretrained DINOv2 [26] to obtain a 1,024-dimensional feature and project it to 768 dimensions. For multi view images, we encode each view with DINOv2, add a simple view position tag, average the view features, and project to 768 dimensions. We fine-tune the ABC-pretrained VAE on Furniture and CADNet40 for 300 epochs each, followed by training a dataset-specific DiT for 300 epochs. Other settings and splits follow the main training. The Furniture training set has 1,440 shapes and the CADNet40 training set has 7,394 shapes. All visual results use test set conditions.

Class-conditioned generation Fig. 8 shows the results on Furniture. The generations express category specific structures while keeping variations within each class.

Point cloud-conditioned generation Fig. 9 shows the results on CADNet40. The outputs follow the geometry of the input points and preserve key features such as holes and fillets.

Partial point cloud-conditioned generation Fig. 10 shows the results when the input point clouds have missing local regions. The generations complete the absent areas and keep the observed parts consistent with the input.

Sketch-conditioned generation Fig. 11 shows the results from wireframe sketches. The outputs align with the silhouette and recover coherent faces and edges.

Single-view conditioned generation Fig. 12 illustrates single-view conditioning. For each input, we sample two candidates, at least one of which is view-consistent, while both typically capture the global shape without multi-view cues.

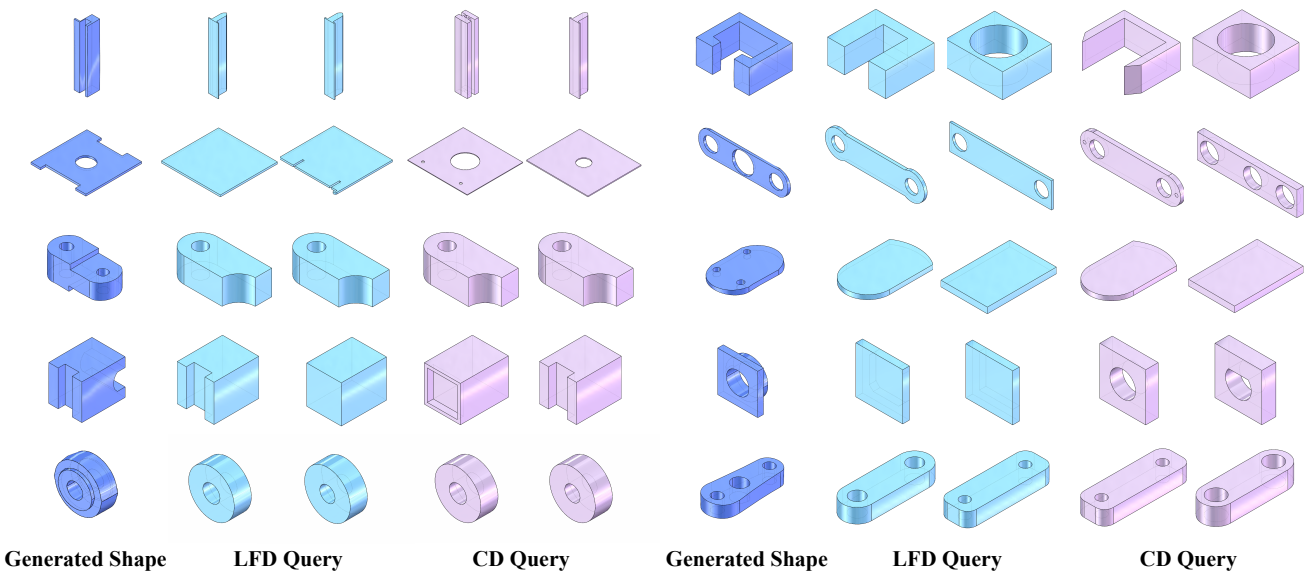

Figure 7. **Retrieval-based novelty check.** For each generated shape (left in each triplet), we show its two nearest neighbors from the training set retrieved by Chamfer Distance (CD) and Light Field Distance (LFD). Despite proximity under both metrics, the generated results display distinct geometric features, supporting that *HiFi-BRep* does not simply memorize training examples.

Multi-view conditioned generation Fig. 13 shows the results from multiple views. The generations are consistent across viewpoints and recover fine details more reliably than single view inputs.

D. Unconditional Generations on ABC

We provide additional unconditional results on the ABC dataset to complement the main experiments. The examples illustrate that our single–stage decoding produces coherent solids with consistent edge–face incidence and fewer topology errors, aligning with the validity gains reported in Sec. 4.2.

E. Data Preprocessing and Postprocessing

We follow the general pipeline of BRepGen [40] and DTGBrepGen [20] and adapt it to our representation. We first remove periodic seams by splitting all torus faces and ring edges so that no periodic faces or edges remain. We then normalize each model to a cube $[-1, 1]^3$ centered at the origin. For each face we record its position box F_p as the two diagonal corners of the face bounding box. For each edge we record its position box E_p in the same way and we take its two endpoints as $\mathcal{V}$. Next we normalize every face and every edge again to $[-1, 1]^3$ in its own local frame and treat them as B-spline primitives. A face is represented by a 6×6 control grid F_z. An edge is represented by six control points E_z. In F_p, E_p, and $\mathcal{V}$, the first point is the minimum corner and the second point is the maximum corner. Each shape becomes a sequence of faces and a sequence of edges. We sort the face sequence by F_p in lexicographic order and sort the edge sequence by E_p in the same way. Unlike BRepGen, which stores sampled points, we store control points. This reduces the parameter count and improves surface smoothness, which is consistent with the benefit of control point representations reported by DTGBrepGen.

In postprocessing we reconstruct loops and assemble a solid from the predicted geometry and topology. We start from the predicted edge–face incidence and collect for each face the set of incident edges. Inside a face we try to build one or more closed non intersecting loops by pairing each edge endpoint with the nearest endpoint that belongs to a different edge. The preprocessing step guarantees that there is no ring edge, so a loop corner must connect two different edges. If we obtain valid loops we accept them as the vertex–edge topology for that face. If loop building fails the current sample is marked invalid. We keep two vertex sets during this step. One set is $\mathcal{V}$. The other set is obtained by de-normalizing the two end samples of each edge. We try both and take the one that succeeds. This fallback is the same as in BRepGen.

After we obtain edge–face and vertex–edge topology we compute uniform samples from the predicted control points. We sample each face on a 32×32 grid from F_z and each edge with 32 points from E_z. We then de-normalize faces and edges using F_p and E_p. We refine the geometry with the joint fitting procedure of BRepGen so that topologically connected surfaces and curves meet cleanly. With the refined curves, surfaces, and all topological relations, we use

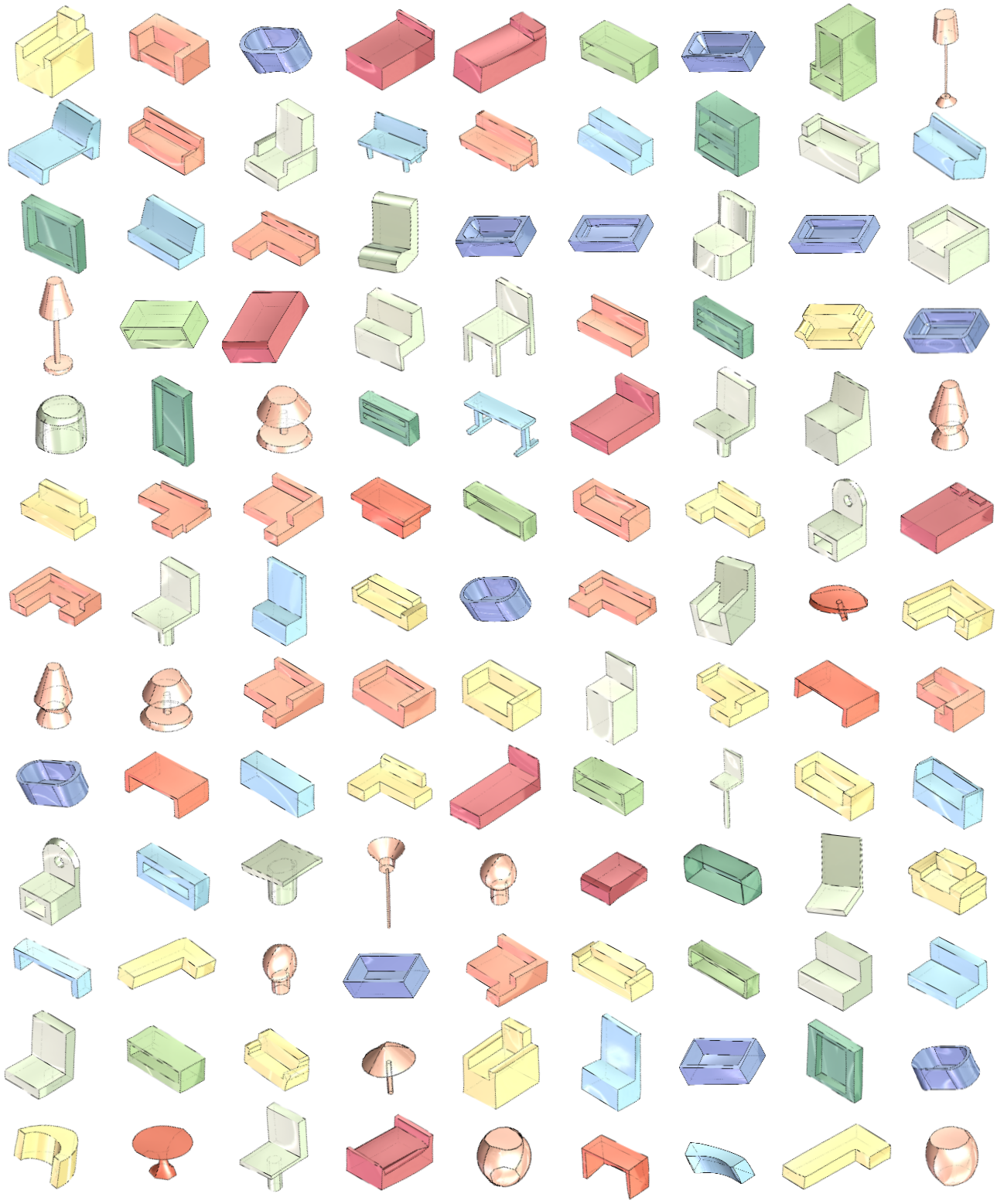

Figure 8. **Class-conditioned generation on Furniture.** Colors indicate categories (ten classes). Samples from different categories are interleaved, showing class-specific traits and intra-class diversity.

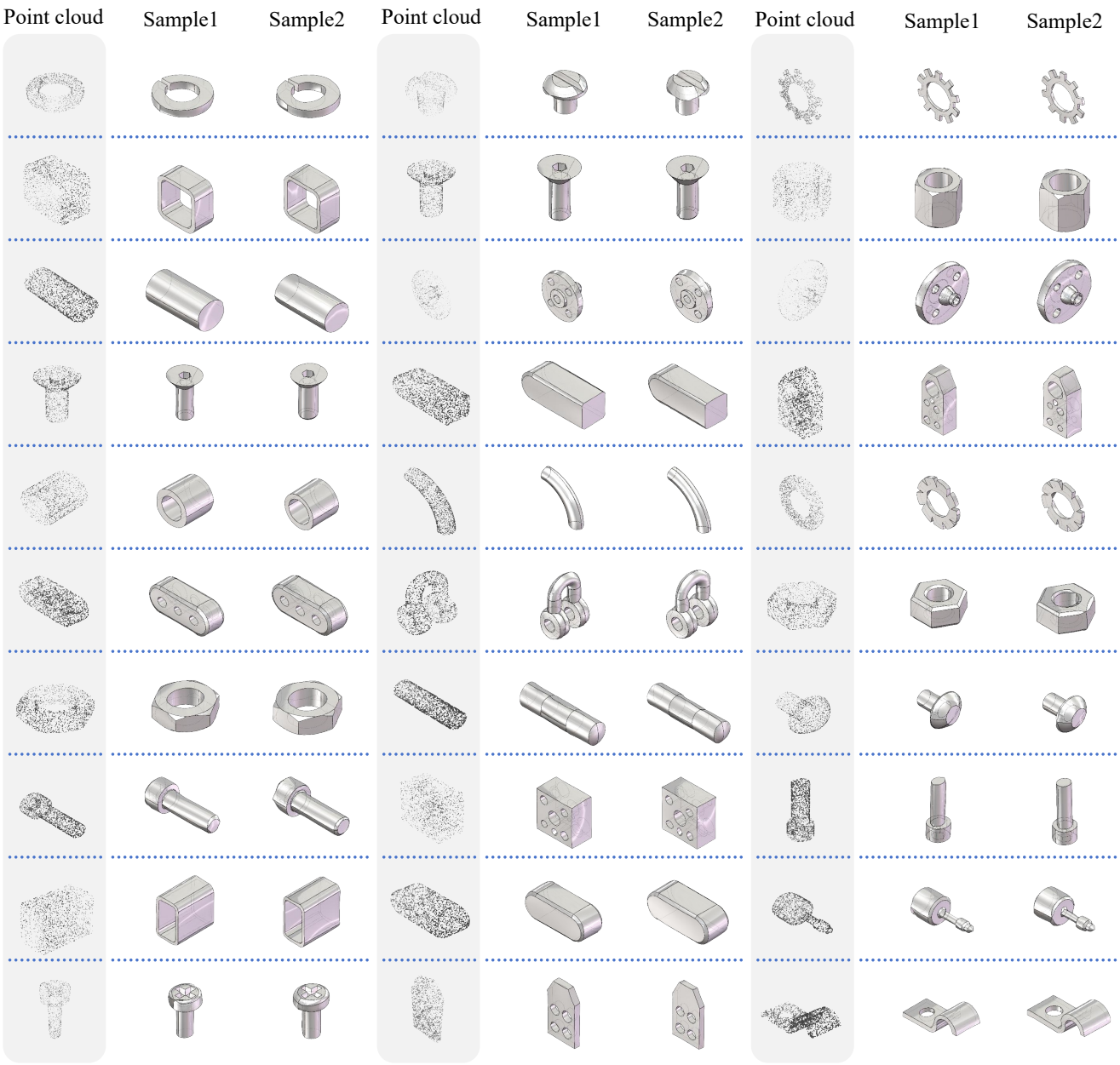

Figure 9. **Point cloud-conditioned generation on CADNet40.** Given a 2,048 point input, the model produces solids that match the global shape while allowing small variations.

the OpenCascade kernel to build the final B-rep. Any exceptions during this step are treated as a compile failure. A sample is considered valid only if the kernel can construct the model, confirm that it is closed, and compute a finite volume. If any of these checks fails the sample is invalid. These checks certify closure and watertightness. Topological legality is already enforced before a model can compile: every edge is incident to exactly two faces and two vertices, each face loop closes head to tail, and no extraneous vertices or edges are present. The extra step of sampling from

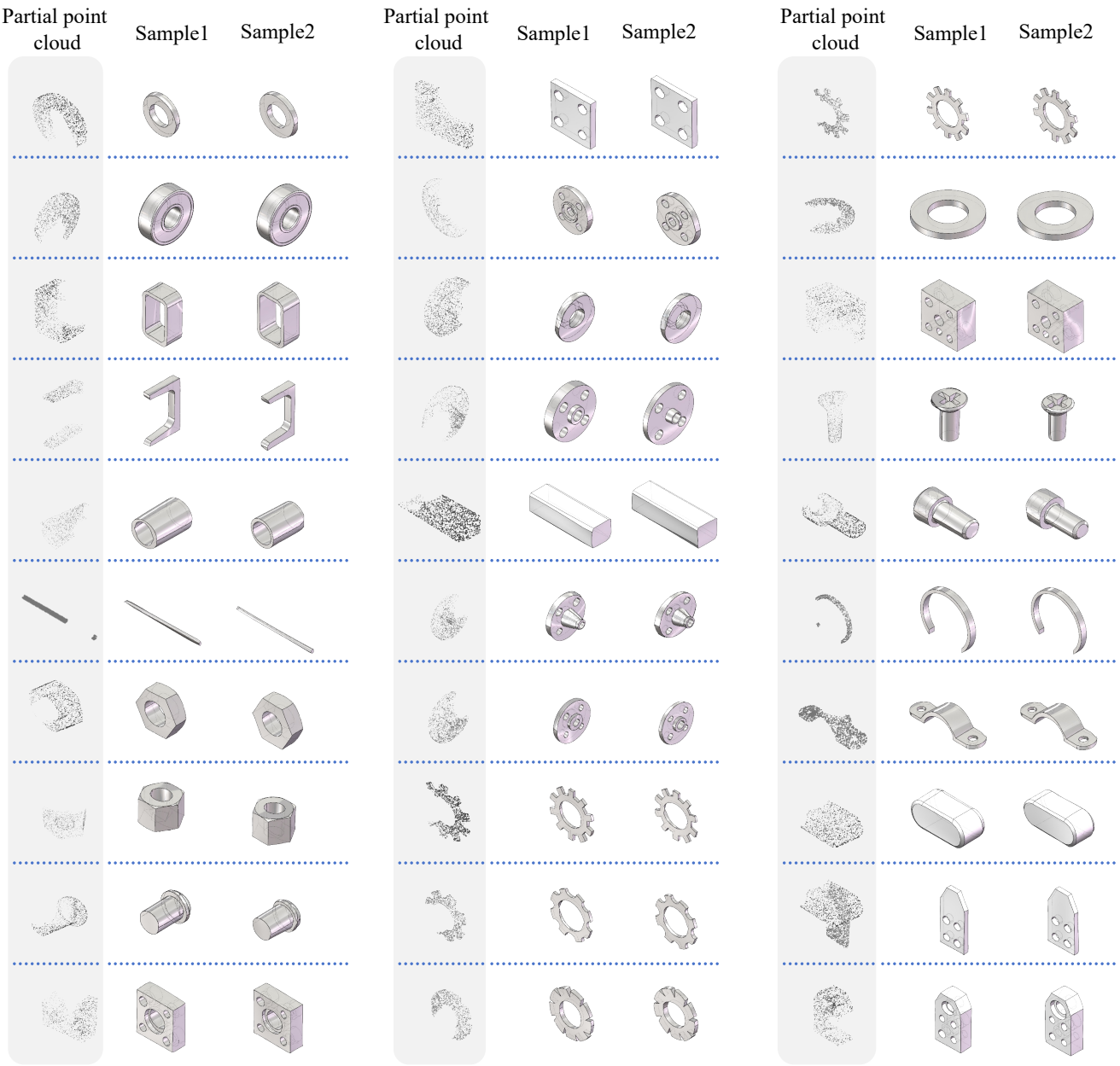

Figure 10. **Partial point cloud-conditioned generation.** The model generates plausible completions for missing regions while respecting visible geometry.

control points and fitting curves and surfaces explains why our postprocessing is slightly slower than BRepGen even though both pipelines share the same assembly procedure.

F. Generality of the Proposed Generator

We discuss the ability to handle common B-rep cases where two faces share more than one edge, and we compare the modeling assumptions with HoLa [24] and DTGBrepGen [20]. HoLa is an excellent near one-stage method. It first performs an implicit intersection test between face pairs and then generates the corresponding half-edge pairs. This design ties edge creation to a single intersection per face pair. In practice it means one shared edge per face pair unless extra logic is added. Our approach predicts a set of unique edges and a set of unique faces and then assigns each edge to its two incident faces with an explicit incidence matrix. There is no built-in cap on how many edges two faces may share. DTGBrepGen adopts a fixed cap of five shared

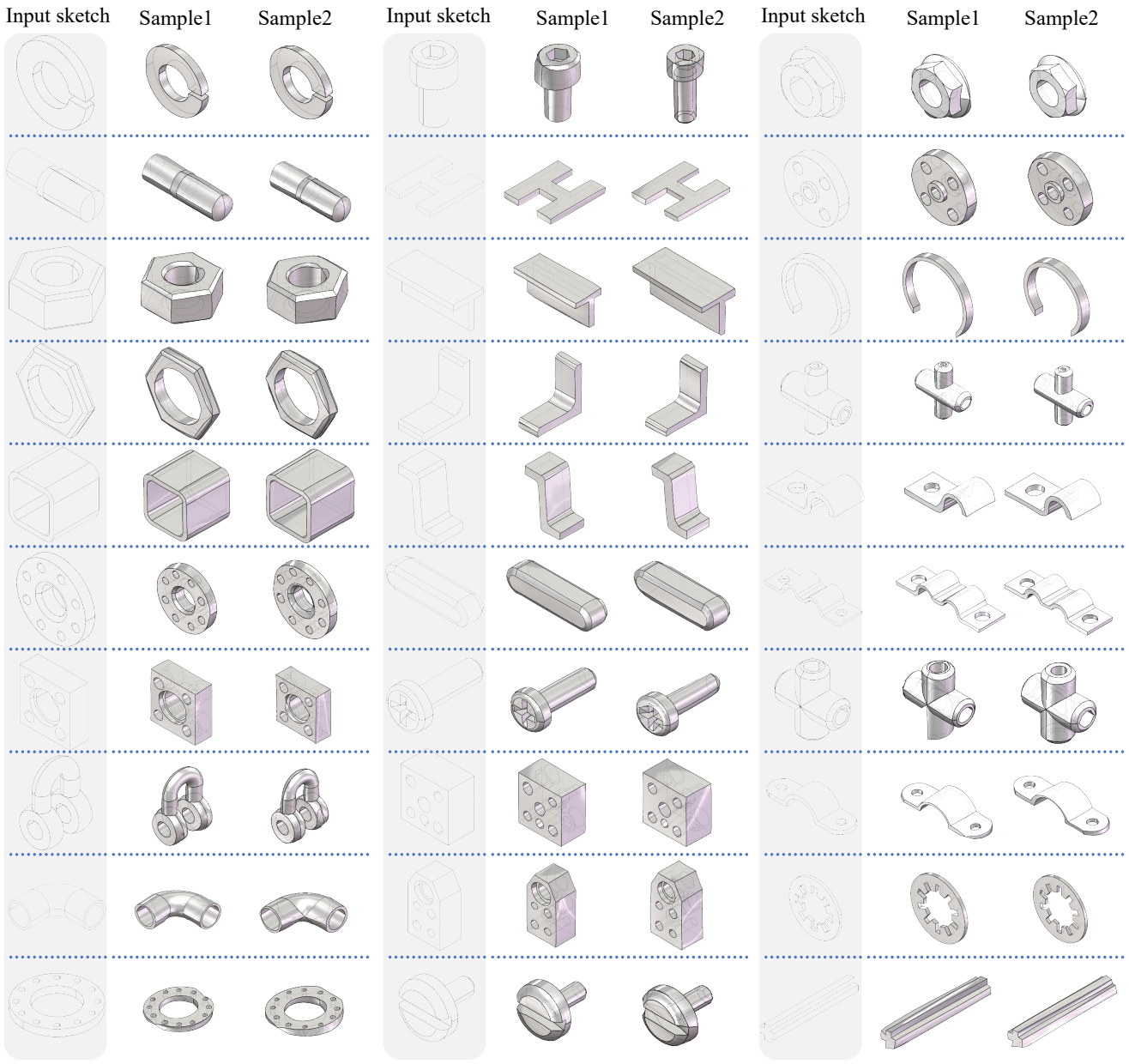

Figure 11. **Sketch-conditioned generation.** Given a sketch, the model generates solids that respect the sketch while maintaining topological consistency.

edges per face pair, which in our statistics covers 99.9% of ABC. The cap is practical, yet the distribution is long tailed, so a cap can become a modeling limit in rare but valid cases.

We measure the maximum number of shared edges between any face pair in each ABC model. The phenomenon is common rather than rare. After splitting periodic faces and ring edges at seams, about 63.37% of models have a maximum shared count greater than one. Without the split, the share is still about 23.71%. Tab. 4 lists the raw counts to show the tail. In short, modeling edges and faces as separate sets with explicit incidence lets the generator cover these frequent multi-edge cases without special rules, while keeping the decoding stage single step and validity aware.

Figure 12. **Single-view conditioned generation.** For each input image, we present two generated candidates, at least one of which consistently aligns with the conditioning view while capturing the overall shape with minor variations.

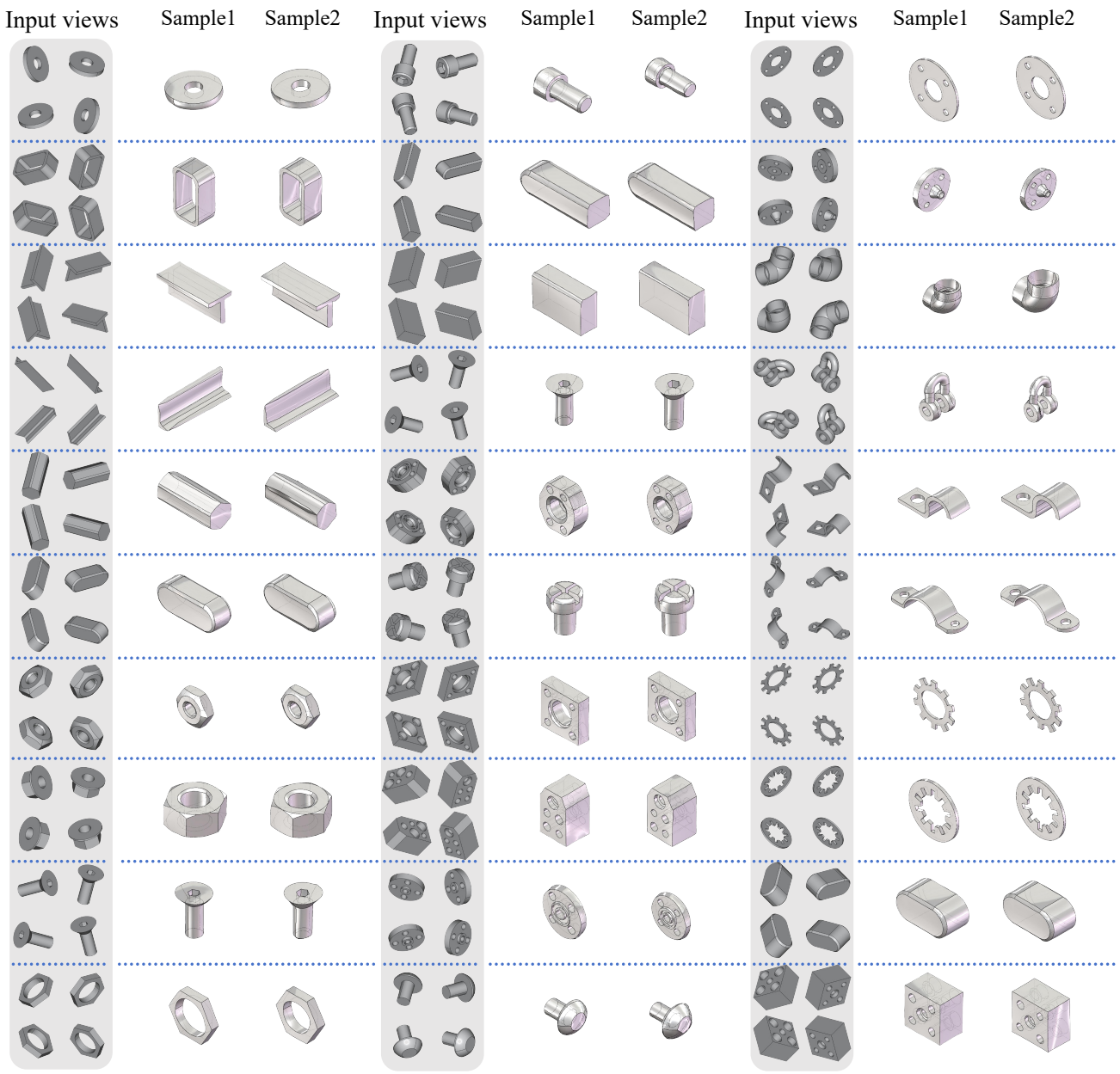

Figure 13. **Multi-view conditioned generation.** With several views, the outputs align across viewpoints and preserve more fine details.

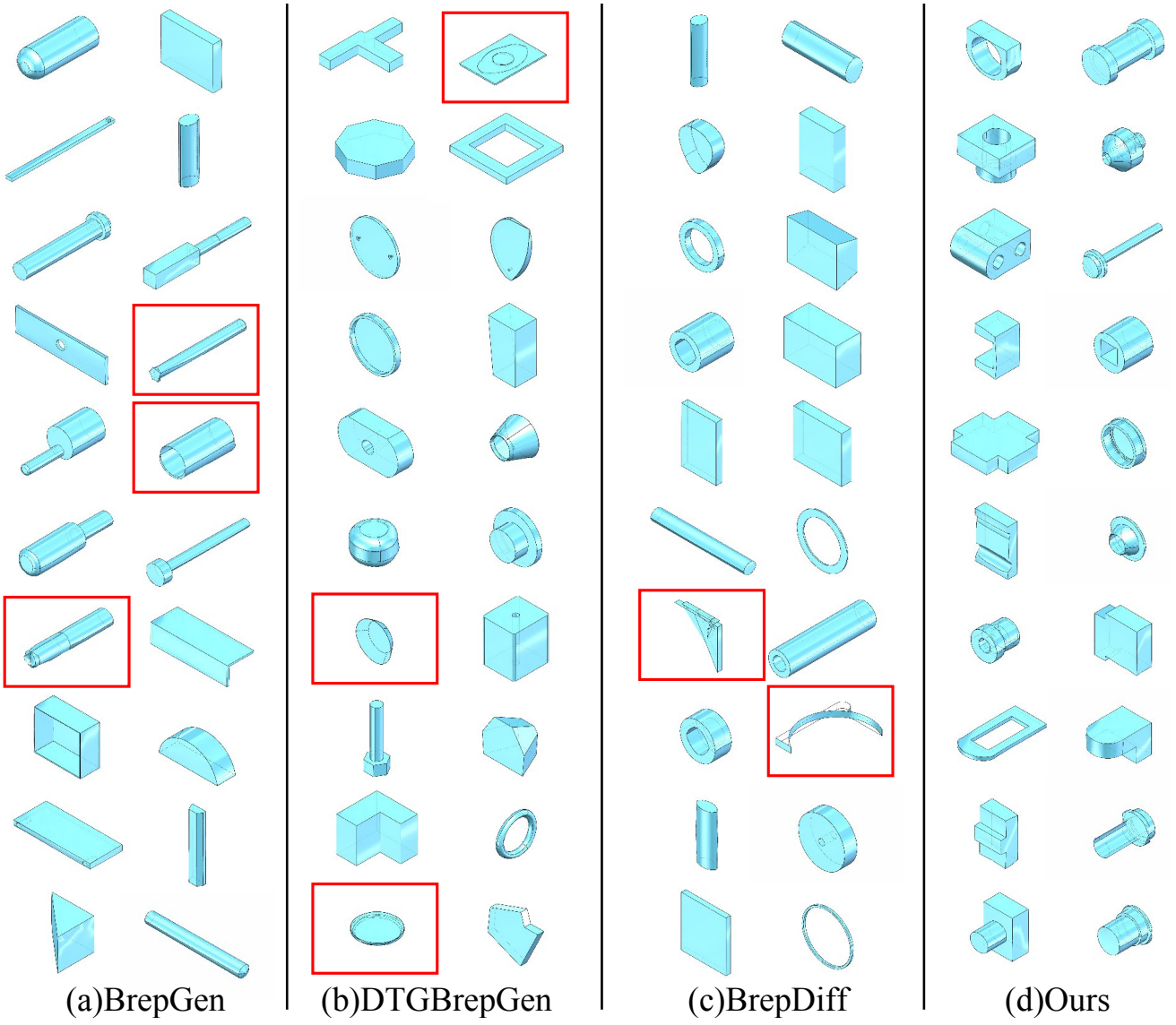

Figure 14. **Unconditional generations on ABC.** Columns (a–d) show BRepGen, DTGBrepGen, BrepDiff, and *HiFi-BRep* respectively. Red boxes highlight typical artifacts observed in baselines, including open shells or missing faces around holes and fillets, non-manifold junctions, and degenerate sliver plates or rods.

Table 4. **Maximum shared edges per face pair in ABC (counts).** Left: before splitting periodic faces/edges. Right: after splitting. The distribution is long tailed, so multiple shared edges are common.

max shared	count
0	847
1	142020
2	40735
3	1830
4	509
5	110
6	47
7	15
8	22
9	4
10	5
11	2
12	2

Before splitting

max shared	count
1	68188
2	114404
3	2736
4	737
5	48
6	26
7	2
8	4
9	1
11	1
16	1

After splitting