Fine-tuning Pre-Grasps through 3D Object Generation

Almutwakel Hassan
akhassan@andrew.cmu.edu

Chaitanya Chawla
cchawla@andrew.cmu.edu

Eyob Dagnachew
edagnach@andrew.cmu.edu

Keywords: Latent Diffusion Model, 3D Reconstruction, Pre-Grasp Generation

1 Introduction

Three-dimensional (3D) point clouds have emerged as a foundational representation for numerous applications in computer vision [1], [2], robotics [3], and graphics [4], [5], including 3D scene reconstruction, autonomous navigation, and digital human modeling. As interest in high-fidelity 3D generation grows, recent advancements in generative modeling have shown that deep generative models, particularly diffusion models, can produce high-quality 3D content with strong generalization capabilities.

Despite the success of diffusion models in 2D image synthesis, directly applying them to 3D data such as point clouds remains challenging due to the unordered, sparse, and permutation-invariant nature of point cloud representations. Through this work, we tackle the problem of improving robot pre-grasp generation by fine-tuning our model on 3d assets generated by a diffusion model. We propose a latent diffusion model [6] that encodes 3d shapes into compact volumetric 3D latent embeddings and performs denoising in this space to generate high-quality voxel shapes.

In this project, our goal is to first formulate a lightweight, resolution-agnostic latent representation for general voxel generation synthesis. Then, we introduce a framework to fine-tune a bootstrapped grasp generation model on a dataset of 3d generated objects and their respective pre-grasps. Our approach improves the model by tuning on unseen, diverse objects making it robust during real-world deployment.

2 Task and Dataset

2.1 Task Description

In this work, we implement a 3D Latent Diffusion model **from-scratch** including an attention-based UNet model for denoising. The implementation is built using foundational PyTorch layers and publicly accessible OpenAI Neural Network modules for attention mechanisms. The goal is to introduce novel design choices, differentiating our approach from existing methods, potentially leading to a unique and innovative model.

2.2 Dataset

We use 3D assets curated from Objaverse [7]. We train with ~ 50 annotated daily-life objects, such as sofa, sword etc. Most of our objects are low-poly models to facilitate faster training. Fig 5 in the appendix shows some of the objects used in our training.
Our grasp-generation model is pre-trained on the ACRONYM Dataset, consisting of 8872 objects from ShapeNET and roughly 2000 grasps for each object.

We describe our model in detail in section 4.2.

3 Related Work

3.1 Latent Diffusion Models

A clear challenges arises when scaling usage of diffusion models in pixel space. computational expenses being to arise and pose a serious issue for serious applications of diffusion models. Large strides were made to address this through [6], [8], which proposes a shift in the dimensionality in which a diffusion model is operating in. Through the use of autoencoders, the paper introduces the idea of having the diffusion model operation on the latent representation of pixels instead of on the pixels directly.

3.2 3D Reconstruction

Ren et. al. [9] approaches the task of generation of 3D objects through sparse convolution networks to represent and generate 3D shapes. It relies heavily on hierarchical voxel-based latent diffusion through a VAE encoder/decoder model. Chen et. al. [10] builds on the approach of [9] by focusing on the efficiency of the latent space. Sample complex regions of a 3D model and apply a targeted attention mechanism to capture structural detail. However, a key difference between the two models lies in how DORA operates in point-cloud space as opposed to voxel space.

Xiang et. al. [11] proposes a novel latent representation for 3D generation. While not operating on voxel grids, it's unique approach to hierarchical latent spaces presents valuable ideas for structured modeling. More roughly related, [12] trains Neural Radiance Fields in a learned latent space using a VAE. compressing complex 3D scenes into low-dimensions embeddings, it facilitates scalable neural rendering, it's methods of latent space modeling can inform our own VAE-based voxel compression and representation.

3.3 Grasp Synthesis

Grasp synthesis is a major challenge in robotic manipulation. Along with the difficulty of finding a precise location for grasping, the model needs to ensure stable and robust grasps, such that the robot can manipulate the object without it falling. Most of the related works differ primarily on the dataset they choose. There are two schools of thought when generating datasets for pre-grasps. One set of datasets [13], [14], [15] analytically compute the feasibility and quality of the graph using force-closure methods. The other set [16] tests the sampled pre-grasps directly in simulation by adding various force pertubations. Furthermore, previous works such as [4], [17] approach this challenge by training with semantic labels. However, in our project, we leverage a 2D segmentation model to extract relevant objects from the scene and generate grasps on them.

4 Methods

4.1 Baseline

Ren et. al. [9] employ a hierarchical voxel latent diffusion model to generate high-resolution sparse 3D voxel grids. It first trains a sparse structure VAE which learns a compact latent representation for each level of the hierarchy. It then trains a latent diffusion model to generate each level of the hierarchy conditioned on the coarser level above it. At inference time, they run each diffusion model in a cascaded fashion from coarse to fine. They leverage the decoder of the sparse structure VAE at inference time to generate high-resolution voxel grids. Fig 8 shows their method. We show in-depth baseline results in the Appendix C as well as the model architecture used by X-Cube.

4.2 Our Method

In the following section, we describe the main methods that are adopted in our project:

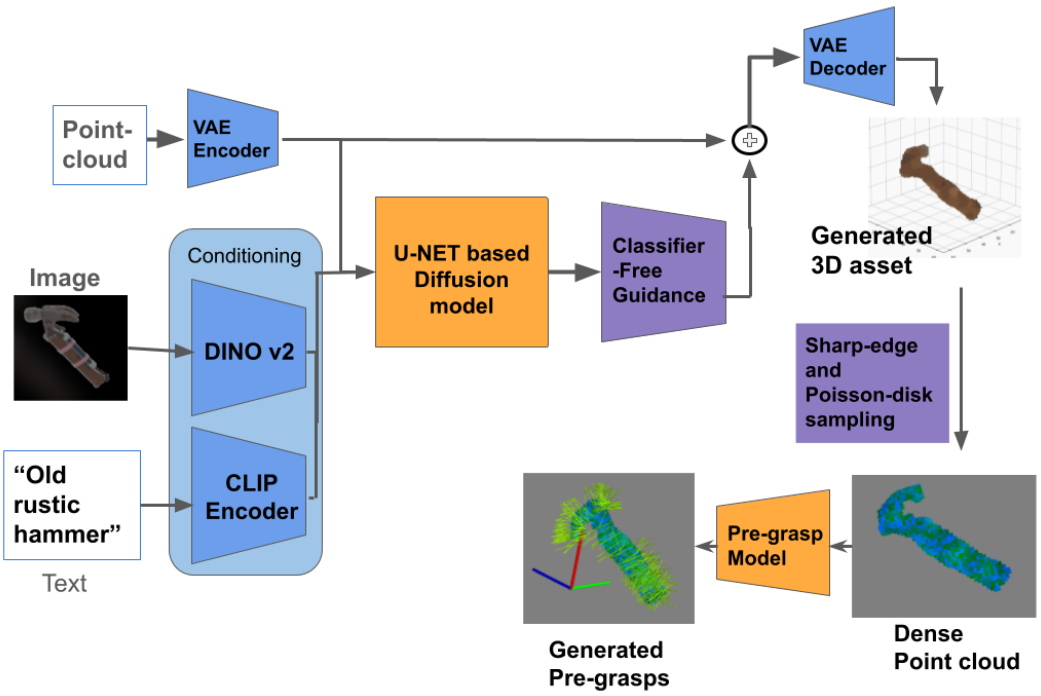


Figure 1: Our Model Architecture

4.2.1 Model Architecture

Our architecture builds upon a combined framework inspired by both XCube and TRELLIS, incorporating both 3D convolutional autoencoders and attention-based 3D UNets within a latent diffusion setting. Initially, a 3D convolutional VAE compresses high-dimensional voxel grids into a lower-dimensional latent space, capturing significant structural features at reduced resolutions (e.g. 32^3 compressed to 8^3). The diffusion process then occurs within this latent space using a 3D UNet architecture that employs both convolutional and self-attention blocks, efficiently modeling local and global structural features. The attention modules integrate positional embeddings and hierarchical spatial information, enhancing the network's ability to model complex spatial dependencies effectively. Our framework emphasizes efficiency, using sparse convolutional techniques adapted from XCube to manage voxel sparsity, particularly critical at lower resolutions.

Figure 1 illustrates our detailed architectural design.

4.2.2 Text and Image Conditioning

We implement robust text conditioning to facilitate precise and meaningful 3D object generation from textual prompts. Textual inputs are encoded using a pre-trained CLIP model, which produces embeddings that capture rich semantic information. These embeddings guide the diffusion model through cross-attention conditioning in the UNet layers at multiple scales. This conditioning ensures that the textual semantics are heavily involved in the denoising process, which enables accurate translation from text descriptions into the 3D voxel structures.
In addition to text, we utilize image conditioning to enhance the precision of generated 3D voxel outputs based on visual inputs. Image conditioning is implemented using a pre-trained vision-language transformer, DINOv2, as used in TRELLIS, which extracts rich visual embeddings from input images. These visual embeddings condition the diffusion model similarly through cross-attention mechanisms within the UNet, similar to how text conditioning is added. This conditioning allows the model to leverage detailed visual context, ensuring fidelity between input images and generated 3D representations.

4.2.3 Text to Image to 3D

Despite the success of text conditioning models and its successful usage in text-to-3D, image-to-3D often finds more success due to its ability to ground 3d generations in image inputs. The inputs provide more information, which reduces the level of creativity needed in the model. For text-to-3D, the TRELLIS authors advise that text to image, followed by image to 3D is a much more robust pipeline, due to text to 3D models having limited creative capabilities which are bottlenecked by 3d dataset limitations.

4.2.4 Classifier-Free Guidance

Classifier-Free Guidance (CFG) significantly enhances the fidelity and alignment of generated voxel outputs with the conditioning inputs, and enables fine-grained control over generation quality. During training, we occasionally omit conditioning information, allowing the model to learn both conditional and unconditional generation distributions. At inference time, guidance scaling factors are introduced to interpolate between conditional and unconditional predictions dynamically. This approach allows adjusting the intensity of text and image conditioning, balancing creativity and fidelity, resulting in more accurate and visually coherent 3D generations.

Classifier-Free Guidance adjusts the diffusion model's prediction by interpolating between conditional and unconditional outputs at each denoising step. Let $\epsilon_\theta(z_t, c)$ be the predicted noise at timestep t conditioned on input c, and $\epsilon_\theta(z_t, \emptyset)$ the unconditioned prediction. The final guided prediction is:

$$\hat{\epsilon}_\theta(z_t, c) = \epsilon_\theta(z_t, \emptyset) + s \cdot (\epsilon_\theta(z_t, c) - \epsilon_\theta(z_t, \emptyset))$$

where s is a guidance scale hyperparameter. This expression is applied during each denoising step to amplify the influence of the conditioning input.

4.2.5 Sharp Edge Sampling (SES)

Chen et. al. [10] introduces Sharp Edge Sampling to enhance reconstruction of 3D shapes, by focusing on capturing fine-grained geometric details often missed in uniform sampling methods.
Salient Edge Detection: Given a triangular mesh, SES identifies salient edges by computing the dihedral angle between two adjacent faces. If the given angle:

$$\theta_{ij} = arccos(n_i.n_j)$$

is greater than a given threshold δ, then the edge is considered salient.
Importance Sampling: From the set of salient edges ϵ_s, the method collects unique vertices to form a salient vertex set V_s. If $|V_s| > N_s$, where N_s is the desired number of salient points, then we downsample V_s using Farthest Point Sampling. Otherwise, we uniformly sample more points along the salient edges ϵ_s

4.2.6 Grasp synthesis

For synthesizing grasps from a given RGB-D input, we fine-tune Contact-Graspnet [18] on a new dataset, containing our generated 3D assets and grasps generated from Contact-Graspnet.

5 Experiments

5.1 VAE Ablations

Our VAE iterative development resulted in ablations between two main final implementations of the VAEs we used.

Implementation 1 was heavily inspired by XCube's VAE implementation but adapted to our dense representation. It includes attention at the bottleneck, two 3d convolutions at each resolution level, average pooling for downscaling, and a sophisticated mask learning layer for upscaling.

Implementation 2 has one convolution per level instead of 2, and does simple pooling downscaling and unpooling upscaling. It adds sliding window attention at the very end for detail improvements. Otherwise it matches the architecture of version 1. Implementation 2 was found to perform significantly better qualitatively and quantitatively, resulting in sharper outputs as opposed to blob-like ones. It also was 4x more parameter efficient, resulting in major efficiency and memory gains.

Figure 2 shows the two VAE architectures and the results obtained from both the architectures.

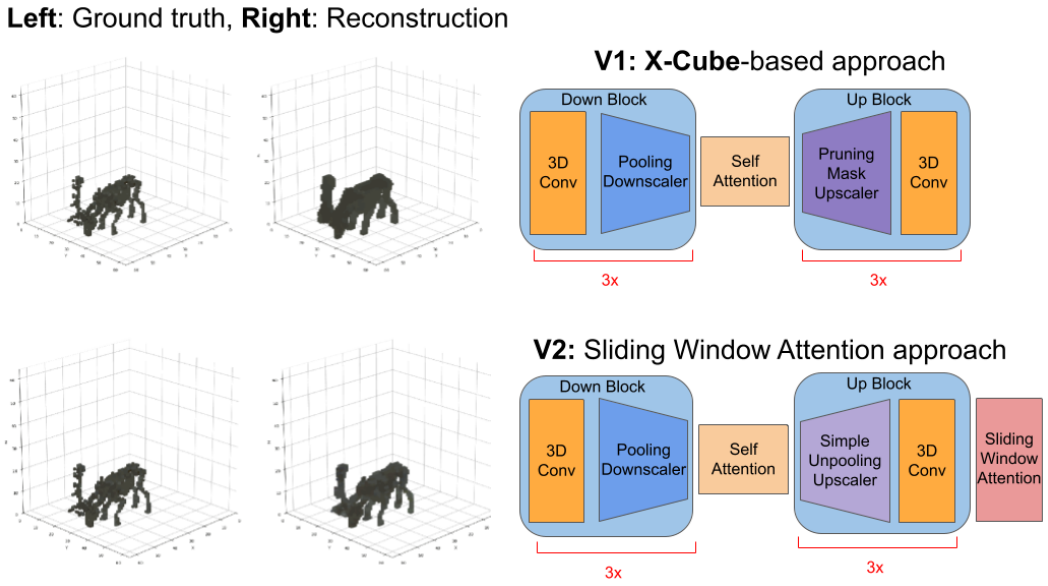


Figure 2: VAE Ablations and their results

5.2 3D Asset Quality

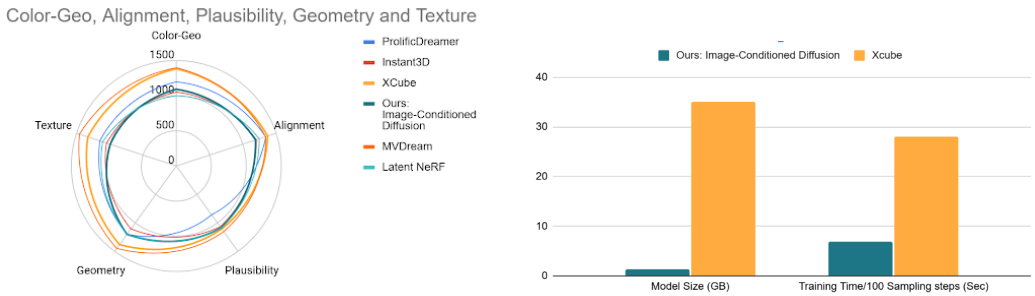
To evaluate the quality of our generated 3D assets, we adopted the GPT-4V-based human-aligned evaluation framework proposed in Wu et al. (CVPR 2024). This framework enables scalable, interpretable, and holistic comparisons between models using rendered RGB and normal maps. Specifically, we assessed our model across five key metrics:

- **Text-Asset Alignment** – How well the generated 3D object reflects the semantics of the input text prompt.
- **3D Plausibility** – Whether the shape is structurally coherent and could plausibly exist in the physical world.
- **Texture-Geometry Coherency** – The consistency between geometric structures and texture patterns, such as textures aligning properly with object surfaces.
- **Texture Detail** – Clarity, sharpness, and realism of the textures rendered on the asset.
- **Geometry Detail** – The complexity and fidelity of the 3D shape's structure, including fine-grained features.

We performed pairwise comparisons using GPT-4V with customized prompts and rendered views, following the methodology described in the paper. GPT-4V was able to provide both a selection between models and natural language explanations for its choices. Each comparison result was aggregated using an Elo rating system, allowing us to holistically rank models across the five dimensions. This evaluation allowed us to bypass the need for ground truth 3D data, which is rarely available for generative 3D tasks, while maintaining alignment with human judgment.

5.3 Inference Time and Model Size, Device used

Our model displays **compactness and efficiency** when compared to our XCube baseline, this is reflected in both it's size and runtime characteristics. We trained and evaluated the model on an



(a) **Holistic evaluation**: XCube scores higher than a majority of models whereas our latent diffusion model scores similarly with a majority of the other models used

(b) Co-training consistently outperforms isolated training as Humanoid B demonstrations increase, achieving good success rates even in low-data regimes.

NVIDIA RTX 4090 GPU.In terms of size, our latent diffusion model occupies just **1.4 GB**, significantly smaller than the baseline XCube model, which is approximately **30 GB**. This benefit manifests itself in better storage and memory usage, making it more suitable for downstream tasks like robotic grasp generation. Inference performance also a mjor benefit towards our model . For a standard evaluation of **100 sampling steps**, our model required an average of **6 seconds per object**, whereas XCube took about **30 seconds** for the same task.

5.4 Pre-grasp Generation

Figure 4 shows some of the antipodal grasps on 3d assets generated from our model. We provide an in-depth analysis of the quality of generated grasps in the appendix D.1. Note that our current model only predicts grasps from one side of the asset, i.e. the side facing the camera. Furthermore, our current approach doesn't consider any semantic information, leading to generating inaccurate grasps, as seen in the third and fourth object.

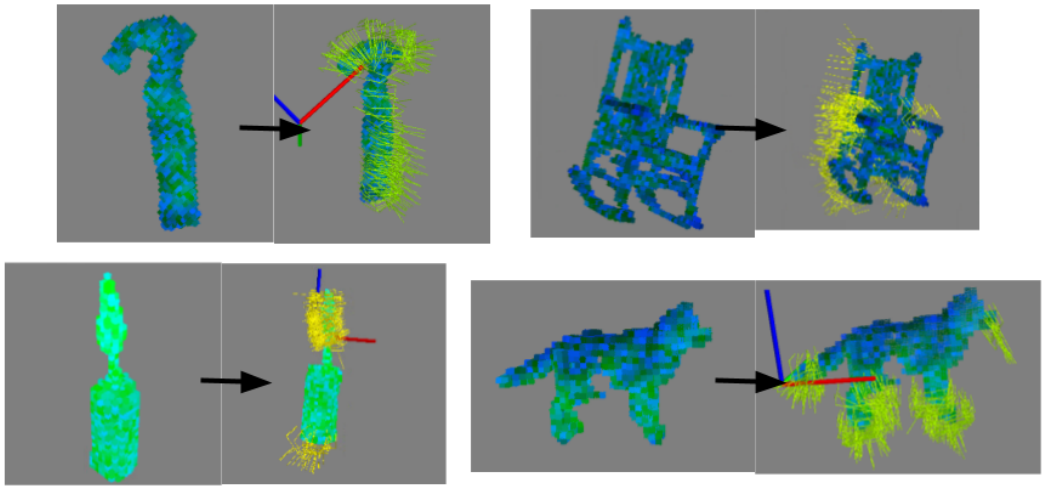


Figure 4: Generated Grasps on our generated 3d assets

6 Code Overview

As this is a from-scratch implementation, all of the code shared in the repository's main directory files is implemented by us. We reference latent diffusion open-source implementations provided by OpenAI. There are additional modules for evaluation and robotic grasping, which are built on top of open-source repositories. Here we describe the most important parts of the code and refer the reader to the appendix for a further discussion.

autoencoders.py (lines 1-148)

- Lines 8-42: vae_loss function that implements a weighted VAE loss suitable for sparse voxel data
- Lines 45-123: AutoencoderKL3D class implementing a 3D variational autoencoder
  - Lines 47-85: Encoder architecture with proper downsampling for 3D volumes
  - Lines 87-114: Decoder architecture with upsampling to reconstruct 3D volumes
  - Lines 116-122: Core VAE functions (encode, reparameterize, decode, forward)

train_diffusion.py (lines 1-488)

- Lines 8-24: loss_logger class for tracking training metrics
- Lines 27-45: encode and decode functions to interface between diffusion and VAE models
- Lines 48-145: train_one_epoch function implementing the diffusion model training loop
- Lines 147-195: val_one_epoch function for validation
- Lines 197-306: sample_N_images function for generating samples from the diffusion model
- Lines 308-488: Main training execution with model initialization, dataset loading, and training loop

models.py (lines 1-1373)

- Lines 13-85: Core neural network building blocks (GroupNorm32, conv_nd, linear, etc.)
- Lines 86-106: timestep_embedding function for encoding diffusion timesteps
- Lines 107-155: checkpoint function and implementation for memory-efficient backprop
- Lines 188-225: TimestepBlock and TimestepEmbedSequential for conditioning on timesteps
- Lines 226-277: Upsample and Downsample classes for multi-dimensional feature maps
- Lines 278-391: ResBlock class implementing residual blocks with timestep conditioning
- Lines 392-463: AttentionBlock class for self-attention in diffusion models
- Lines 464-544: QKVAttention implementations for efficient attention computation
- Lines 959-1221: UNetModel3D class implementing a 3D UNet architecture for diffusion
- Lines 1222-1373: Helper functions for creating UNet models of different sizes (UNet-Big3D, UNet3D, UNetSmall3D)

7 Timeline

Table 1 provides details about how much time we spent on each part of the project.

Task	Time Spent
Reading Related Works	$10hrs$ to $12hrs$
Reading XCube and Contact-Graspnet Documentation	$2hrs$ to $3hrs$
Setting up and running XCube and Contact-Graspnet locally	$4hrs$ to $5hrs$
Writing Latent-Diffusion Models from scratch	$55hrs$ to $60hrs$
Modifying Contact-Graspnet to our data format	$4hrs$ to $6hrs$
Running experiments for various ablations	$10hrs$ to $12hrs$
Compiling results	$2hrs$ to $3hrs$
Preparing Report	$2hrs$ to $3hrs$

Table 1: Time spent on various tasks

8 Research Log

8.1 From-scratch Development

The bulk of the time we spent on this project was spent in the development of our model implementation. We began early and incorporated elements from class lectures slowly and iteratively into our model implementation for the 3d latent diffusion pipeline. Namely, we took inspiration from 2d latent diffusion, and added another spatial dimension, which resulted in custom implementations of everything from ResNet blocks, QKVAttention, cross attention blocks, transformer modules, relative position embeddings, dataloaders, training architectures, and even custom voxel-based visualizers (using matplotlib). Many of these improvements were built on quick training iterations using a toy dataset of 3d digits to determine how well the model was learning, and what was lacking. We iterated heavily on the architecture of the UNet model as well as the 3D variational autoencoder, testing both lower dimensional latent spaces and 3D latent spaces. Through iterations, we landed on an approach with 3D VAEs using attention, as other representations and model types did not learn well enough from the small dataset we had to work with. Due to time and GPU memory limitations, we could not scale up our model to full resolution as we originally planned in our proposal.

We also spent a lot of time setting up and using X-CUBE, as the dependencies used by X-CUBE are particularly challenging to set up and use, including custom NVIDIA libraries that replace PyTorch. We developed our own scripts to run inference using their codebase, so that we could port over results for our evaluation.

8.2 Selecting Metric

Another major challenge in our project was identifying interpretable and reliable evaluation metrics for assessing the quality of 3D assets generated from text prompts. Unlike traditional supervised learning settings, the lack of ground truth 3D models corresponding to our prompts made it difficult to find ways to apply standard metrics like PSNR or Chamfer Distance in a directly interpretable way. This absence of reference data significantly complicated our efforts to benchmark our model's performance.

Initially, we explored multiple quantitative metrics, such as CLIP Score, FID with DINO features, ,and normal-map-based metrics for surface detail. However, these metrics had their own limitations—some required ground truth geometry, while others, like CLIP Score, only assessed superficial text-to-shape alignment without capturing geometric plausibility or texture coherence.

We investigated 3DGen-Bench, which proposed 3DGen-Score as a comprehensive evaluation suite. This seemed promising, as it provided a single model architecture able to comprehensively and quantitatively review the quality of 3d model architectures, potentially making the evaluation pipeline very simple. However 3DGen-Score was available only as a compiled .pkl file with no open access to the model architecture or training code, making integration and interpretation challenging. Without transparency into the model's behavior or inputs, its use as a black-box evaluator risked undermining the reproducibility and reliability of our results.

Ultimately, we pivoted to using GPT-4V(ision)-based evaluation, as described in [Wu et al., CVPR 2024]. GPT-4V provided a scalable, human-aligned framework capable of performing pairwise comparisons across similar model quality metrics to 3DGen-Bench like text-asset alignment, texture detail, geometry detail, texture-geometry coherence, and 3D plausibility. What made GPT-4V especially compelling was its interpretability—each decision was accompanied by a natural-language explanation, making the evaluation process both transparent and extensible. Additionally, since GPT-4V relies on rendered views (RGB and normal maps) rather than full 3D data, it bypassed the need for ground truth models entirely.

This shift not only allowed us to perform holistic model comparisons using Elo ratings but also significantly streamlined our evaluation pipeline. It also helped bridge the gap between objective and subjective quality assessments, aligning closely with human perception. Despite its reliance

on API access and probabilistic outputs, GPT-4V proved to be the most feasible and interpretable option given our constraints.

In summary, the journey to find a suitable evaluation strategy was non-trivial and iterative. While many traditional metrics fell short due to lack of ground truth, leveraging GPT-4V as a human-aligned evaluator allowed us to overcome this fundamental bottleneck and conduct meaningful comparisons across generative models.

8.3 Pre-Grasp Generation

Finally, while selecting pre-grasp generation, we went through various models, but found out most to be out-of-date and incompatible with the current nvidia-drivers. After trying models like AnyDexGrasp, DexNet, and Contact-GraspNet, we narrowed down to Contact-GraspNet. Another major issue while inferring current model on our generated 3d voxel grids arose due to a different scaling method in the original dataset. We had to scale and shift our generated assets into the origin to obtain meaningful pre-grasp candidates from the model.

9 Conclusion

We introduce a self-supervised framework to continuously improve pre-grasp generation quality using 3D asset generation through a latent diffusion model. We ablate on various design architecture choices as well as different evaluation metrics for both the 3d asset generation pipeline and the grasp generation pipeline.

As future work:
We want to scale up the resolution and dataset size of our model to better compare with X-CUBE. Furthermore, use native 3d sparse data structures to improve efficiency.
And on the aspect of pre-grasp generation, we want to add a self-supervised critic (e.g., collision checker, grasp quality metric) to validate or rank generated grasps. Also, add semantic information about the object to the model. Finally, confirm pre-grasp quality in simulation by producing force perturbations and test it on real hardware.

References

[1] A. Ramesh et al. Hierarchical text-conditional image generation with clip latents. *arXiv:2204.06125*, 2022.

[2] B. Poole et al. Dreamfusion: Text-to-3d using 2d diffusion. In *arXiv:2209.14988*, 2022.

[3] H. G. Singh, A. Loquercio, C. Sferrazza, J. Wu, H. Qi, P. Abbeel, and J. Malik. Hand-object interaction pretraining from videos. *arXiv preprint arXiv:2409.08273*, 2024.

[4] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[5] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[6] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.

[7] M. Deitke, D. Schwenk, J. Salvador, L. Weihs, O. Michel, E. VanderBilt, L. Schmidt, K. Ehsani, A. Kembhavi, and A. Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13142–13153, 2023.

[8] A. Blattmann, R. Rombach, H. Ling, T. Dockhorn, S. W. Kim, S. Fidler, and K. Kreis. Align your latents: High-resolution video synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 22563–22575, 2023.

[9] X. Ren, J. Huang, X. Zeng, K. Museth, S. Fidler, and F. Williams. Xcube: Large-scale 3d generative modeling using sparse voxel hierarchies. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4209–4219, 2024.

[10] R. Chen, J. Zhang, Y. Liang, G. Luo, W. Li, J. Liu, X. Li, X. Long, J. Feng, and P. Tan. Dora: Sampling and benchmarking for 3d shape variational auto-encoders. *arXiv preprint arXiv:2412.17808*, 2024.

[11] J. Xiang, Z. Lv, S. Xu, Y. Deng, R. Wang, B. Zhang, D. Chen, X. Tong, and J. Yang. Structured 3d latents for scalable and versatile 3d generation. *arXiv preprint arXiv:2412.01506*, 2024.

[12] G. Metzer, E. Richardson, O. Patashnik, R. Giryes, and D. Cohen-Or. Latent-nerf for shape-guided generation of 3d shapes and textures. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12663–12673, 2023.

[13] J. Mahler, M. Matl, X. Liu, A. Li, D. Gealy, and K. Goldberg. Dex-net 3.0: Computing robust vacuum suction grasp targets in point clouds using a new analytic model and deep learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5620–5627, 2018. doi:10.1109/ICRA.2018.8460887.

[14] J. Mahler, F. T. Pokorny, B. Hou, M. Roderick, M. Laskey, M. Aubry, K. Kohlhoff, T. Kröger, J. Kuffner, and K. Goldberg. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1957–1964, 2016. doi:10.1109/ICRA.2016.7487342.

[15] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics, 2017. URL `https://arxiv.org/abs/1703.09312`.

[16] H.-S. Fang, C. Wang, M. Gou, and C. Lu. Graspnet-1billion: A large-scale benchmark for general object grasping. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11441–11450, 2020. doi:10.1109/CVPR42600.2020.01146.

[17] P. Ni, W. Zhang, X. Zhu, and Q. Cao. Pointnet++ grasping: Learning an end-to-end spatial grasp generation algorithm from sparse point clouds. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3619–3625. IEEE, 2020.

[18] M. Sundermeyer, A. Mousavian, R. Triebel, and D. Fox. Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13438–13444. IEEE, 2021.

Figure 5: **Objaverse Samples**

A Background

Latent diffusion models or LDMs are a class of generative models that operate by learning a denoising process in a compressed latent space rather than pixel space, significantly reducing computational cost while preserving semantic fidelity [6]. Given an input data distribution $x_0 \sim p_{\text{data}}(x)$, an encoder $\mathcal{E}$ maps inputs to a latent space: $z_0 = \mathcal{E}(x_0)$. A forward diffusion process is then applied to z_0 by progressively adding Gaussian noise:

$$z_t = \sqrt{\bar{\alpha}_t} z_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I}),$$

where $\{\bar{\alpha}_t\}_{t=1}^{T}$ is a predefined variance schedule. The reverse process is modeled using a neural network $\epsilon_\theta(z_t, t)$ trained to predict the added noise ϵ, optimizing the variational objective:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{z_0,\epsilon,t}\left[\|\epsilon - \epsilon_\theta(z_t, t)\|^2\right].$$

Once trained, a decoder $\mathcal{D}$ reconstructs data via $x_0 \approx \mathcal{D}(z_0)$. We adopt this framework to generate high-fidelity 3D object structures while amortizing the cost of operating in high-dimensional spaces.

B Detailed Description of Metrics

Attempting to evaluate generative models for 3D point clouds can pose to be challenging given that there are no clear metrics widely used for this task. While some works use CLIP, this mainly only checks for alignment between the generated image and the text prompt used to generate it, neglecting other key qualities towards the quality of the generated 3D model. While human user studies are a reasonable way to rate these arbitrary metrics, they are expensive and difficult to scale. To tackle these challenges, we leverage the capabilities of GPT-4V.

B.1 Meta-Prompt

Firstly, we use a "meta-prompt" system using GPT-4V to generate diverse and customizable text prompts, addressing the need for varied input prompts that reflect real-world user inputs and evaluation focuses. These prompts are structured with components like subjects, properties, and compositions to enable controlled generation.

B.2 LLM-based Evaluation

Secondly, GPT-4V is employed as a 3D Assets Evaluator to compare pairs of 3D shapes generated from the same text prompt. The 3D shapes are represented by 2D visual renderings (RGB and normal images) to make them suitable for GPT-4V's analysis. An instruction template guides GPT-4V in the comparison process based on user-defined criteria. To enhance robustness, the evaluation process uses an ensemble approach, where results from multiple perturbed inputs are combined. The Elo rating system is then used to rank the text-to-3D models based on the pairwise comparisons. We describe these metrics in more detail in section 5.

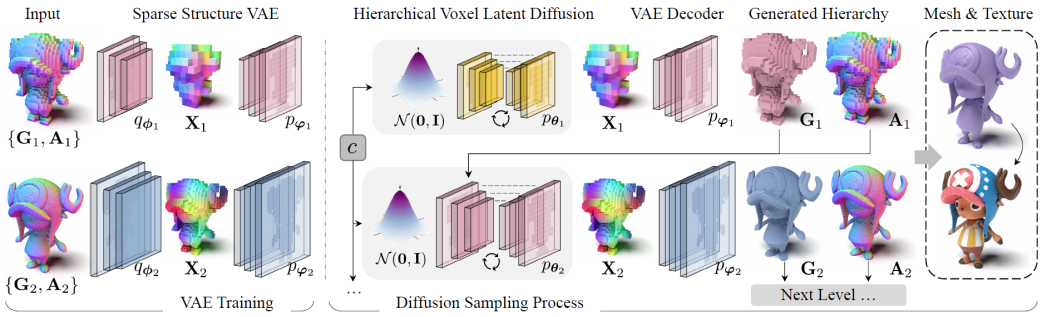


Figure 6: **X-Cube**

B.3 Grasping Metrics

For measuring grasp success, we adopt the **Precision@k** (top-k grasp accuracy), since each object has multiple grasp predictions. A grasp is considered successful if it has more than a given threshold of voxel grid between the grippers. Ideally, a grasp's success should be measured in simulation by applying random forces to the object, however that is beyond the scope of the project.

C Baseline Results

C.1 Model Architecture

Figure 8 depicts the model architecture for XCube.

C.2 Qualitative Analysis of X-CUBE

Prompts: Sofa, Polished oak chair

Prompts: Elephant, 3 bundled balloons

Prompts: Dog building with sand, rotary phone

The X-CUBE model outputs impressive results at a high level of detail, with strong prompt adherence for simple prompts as demonstrated by the Sofa and Polished oak chair prompts. However, the model starts to produce inaccuracies when sophisticated shapes with fine or thin details are requested, such as the elephant or the bundled balloons. The model performs worst when requesting complex prompts with multiple objects or stylistic details, as demonstrated by the dog in sand prompt and the fancy rotary phone prompt.

C.3 Further results from our image conditioned model

Figure 7 shows our image conditioning aspect of the model.

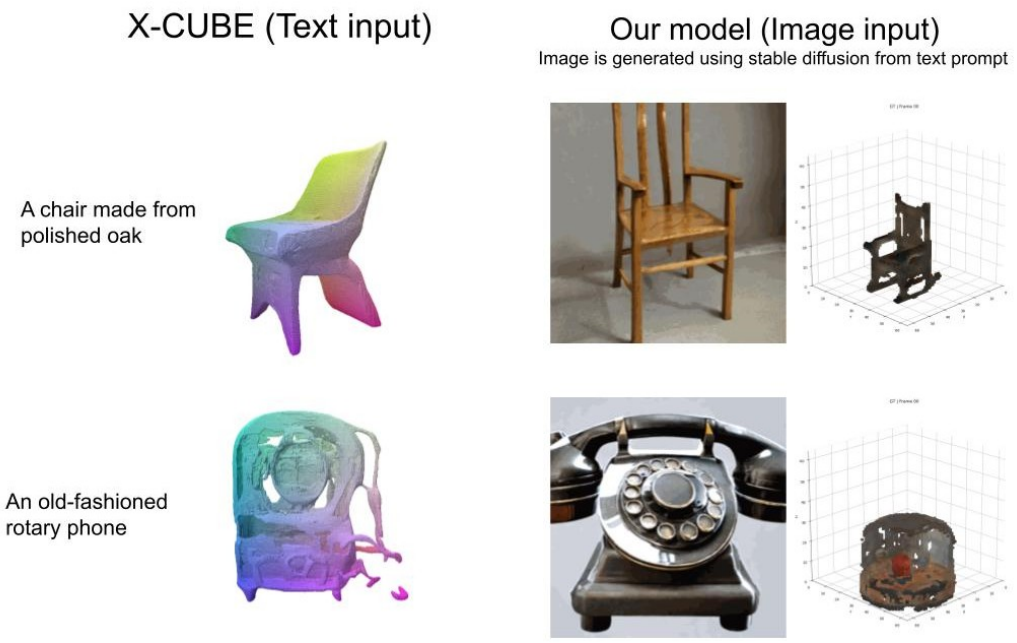


Figure 7: Image and text conditioning

D Further Code Review

gaussian_diffusion.py (lines 1-151)

- Lines 7-57: GaussianDiffusion class implementing the diffusion process
  - Lines 37-56: get_all_scalars method for computing diffusion schedule parameters
  - Lines 58-68: sample_from_forward_process method for adding noise to the input
  - Lines 70-151: sample_from_reverse_process method for generating samples by denoising

train_autoencoder.py (lines 1-271)

- Lines 6-48: Dataset and model initialization for the 3D VAE
- Lines 50-116: Main training loop with custom loss functions for sparse voxel reconstruction
- Lines 117-192: Training epoch implementation with BCE, KL, and color loss components
- Lines 194-271: Validation and model saving logic

test_diffusion.py (lines 1-65)

- Lines 10-22: Model loading and configuration
- Lines 24-37: Testing both the diffusion model and VAE
- Lines 39-65: Sample generation and visualization

test_autoencoder.py (lines 1-86)

- Lines 4-14: Helper functions for encoding and decoding with the VAE
- Lines 17-38: Model loading and testing code

- Lines 40-86: Testing code to evaluate reconstruction quality and visualization

utils.py (lines 1-527)

- Lines 5-17: calculate_psnr and calculate_struct_loss metrics for model evaluation
- Lines 19-58: visualize_batch function for rendering 2D batch data
- Lines 59-135: visualize_sequence function for creating animated visualizations
- Lines 137-215: visualize_voxel_array functions for 3D voxel visualization
- Lines 216-303: visualize_voxel_array_cubes function for rendering 3D voxels as cubes
- Lines 304-511: Advanced visualization functions with profiling and optimization
- Lines 513-527: visualize_voxel_arrays function for rendering multiple 3D arrays

datasets.py (lines 1-333)

- Lines 5-35: Mnist2dDataset class for 2D MNIST data
- Lines 36-91: Mnist3dSeqDataset class for sequential 3D MNIST data
- Lines 92-188: Mnist3dSeqV3Dataset class for colored 3D MNIST with metadata
- Lines 190-254: get_metadata and get_dataset functions for dataset configuration
- Lines 270-275: Utility functions for loading model checkpoints

models_new.py (lines 1-2467)

- Lines 1-200: Improved core building blocks and updated attention mechanisms
- Lines 200-450: Enhanced 3D-specific operations and layer implementations
- Lines 450-800: Modified ResBlock and AttentionBlock implementations for 3D volumes
- Lines 800-1200: UNetModel3DS class with improved spatiotemporal processing capabilities
- Lines 1200-1600: UNet4D and UNetBig4D models for handling 4D data (3D volumes + time dimension)
- Lines 1600-2000: UNet3DS and specialized model variants for 3D sequence modeling
- Lines 2000-2467: Factory functions for creating models of different capacities and configurations

D.1 In-depth Analysis of Pre-grasp Quality

We measure the quality of the generated pre-grasps on three metrics -
1. Point density between grippers
2. Orientation between surface normal and grasp normal
3. Distance between surface and grasp

In the chosen metrics XCube outperforms our model and ablations significantly. This leads us to the conclusion that the pre-grasp model finds it easier to generate grasps for dense point-clouds as produced by XCube. However, our method can be trained much more quickly on a diverse variety of objects, thus leading to a better performance during real-world deployment.

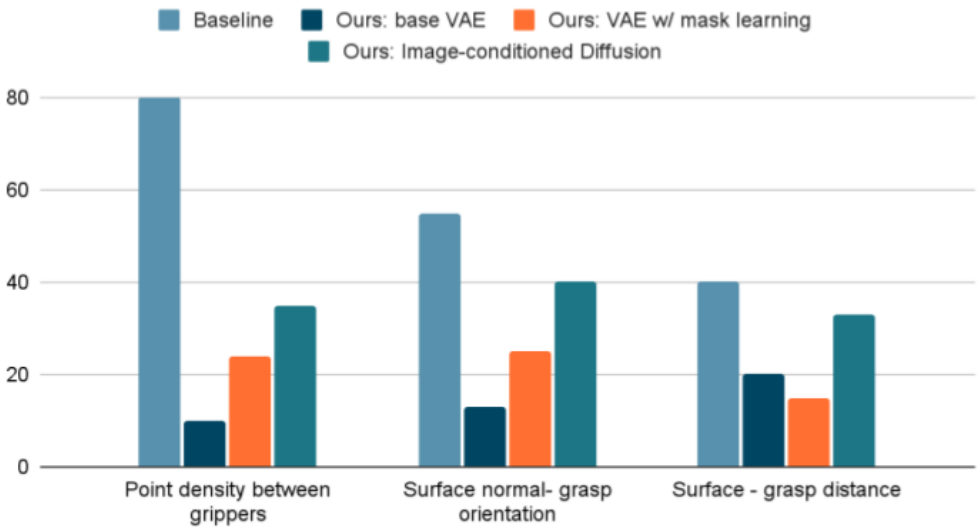


Figure 8: Qualitative Analysis of the generated pre-grasps