ShapeWalk: Compositional Shape Editing through Language-Guided Chains

Habib Slim
KAUST
habib.slim@kaust.edu.sa

Mohamed Elhoseiny
KAUST
mohamed.elhoseiny@kaust.edu.sa

Figure 1. Visualizing shape chains from ShapeWalk. Our dataset consists of 158K unique shapes connected through 26K edit chains, with an average length of 14 chained shapes. We illustrate the interpolation process by coloring shapes based on their proximity to the starting shape (blue), and the ending shape (orange). Each consecutive pair of shapes is associated with precise language instructions describing the applied edits. For each shape transition, we also provide a precise edit vector \( \theta_{ij} \) describing the parameter changes necessary to transition from one shape to the next.

Abstract

Editing 3D shapes through natural language instructions is a challenging task that requires the comprehension of both language semantics and fine-grained geometric details. To bridge this gap, we introduce ShapeWalk, a carefully designed synthetic dataset designed to advance the field of language-guided shape editing. The dataset consists of 158K unique shapes connected through 26K edit chains, with an average length of 14 chained shapes. Each consecutive pair of shapes is associated with precise language instructions describing the applied edits. We synthesize edit chains by reconstructing and interpolating shapes sampled from a realistic CAD-designed 3D dataset in the parameter space of the GeoCode shape program. We leverage rule-based methods and language models to generate accurate and realistic natural language prompts corresponding to each edit. To illustrate the practicality of our contribution, we train neural editor modules in the latent space of shape autoencoders, and demonstrate the ability of our dataset to enable a variety of language-guided shape edits. Finally, we introduce multi-step editing metrics to benchmark the capacity of our models to perform recursive shape edits. We hope that our work will enable further study of compositional language-guided shape editing, and finds application in 3D CAD design and interactive modeling.
We learn and demonstrate the ability of our dataset to train models in the latent space of shape autoencoders, and show that our method can be applied on shapes unseen during the training of the neural editor. Taking advantage of the ground-truth shape chains provided by our dataset, we introduce multi-step editing metrics inspired from the field of trajectory prediction [4, 26] to evaluate the quality of our models.

Our contributions can be summarized as follows:

- **Chained Shape-Editing Dataset**: We introduce a synthetic dataset of chained edits, with associated language instructions and ground-truth edited shapes. This is the first dataset of its kind, and is designed to facilitate the study of compositional shape editing.

- **Neural Shape Editor**: We introduce a neural editor module trained on our dataset, and demonstrate its ability to perform a variety of shape edits. We show that our method can be applied on shapes unseen during the training of the neural editor.

- **Chained-Editing Metrics**: With the introduction of ground-truth shape chains, we propose multi-step editing metrics to evaluate the ability of neural editors to perform recursively applied edits.

## 2. Related work

**3D Shape Programs.** A significant body of work [9, 11, 12, 14, 19] is dedicated to exploring the use of procedural programs for 3D shape representation. Representing 3D objects as visual programs has many advantages. Representing shapes as visual primitives enhances interpretability [11, 14], as programs are by definition human-readable and thus understandable by human experts. Shape programs are more compact [9, 27] than their usual 3D shape modalities, and can be used to represent shapes more efficiently. Programs can also be composed to create shapes [11, 14], or decomposed into smaller programs. Ideally, shape programs can be edited [13] to modify the underlying geometry they represent.

Our dataset consists of a collection of synthetic shapes created utilizing a backbone mesh-generating shape program. Specifically, we build from GeoCode [17], a 3D shape synthesis technique addressing the challenge of mapping high-fidelity geometry to an editable parameter space. GeoCode introduces a procedural program enabling the generation of high-quality mesh outputs with a balanced blend of interpretability and fine control.

**Language-Guided 3D Shape Editing.** Various works have explored the use of natural language instructions to guide 3D shape editing. Recent efforts [16, 20, 21] propose to leverage pre-trained CLIP [24] models to align latent mappings in the space of a frozen shape autoencoder, and show that our method can be applied on shapes unseen during the training of the neural editor.
3D shapes with a given text prompt. Early efforts in that direction [16] included optimizing meshes to progressively align them with CLIP embeddings, leading to important computational overhead. CLIP-Sculptor [21], a more recent work, leverages a voxelized representation and a discrete latent space conditioned on CLIP’s image-text embeddings to perform fast and fidel shape edits without shape optimization.

Shape Editing Datasets. Other works leverage large text-aligned 3D shape datasets and require significant manual annotation effort. ShapeCrafter [8] generates 3D shapes incrementally from text using a neural network, evolving with additional phrases. ShapeCrafter is designed for recursive shape edition and utilizes a VQ-VAE [23] model to represent shapes as discrete codes. This method exhibits consistent shape-text alignment with gradual evolution. Changel3D [3] introduces ShapeTalk, a large dataset for describing 3D shape differences. The framework facilitates language-based editing of 3D models without requiring 2D to 3D conversion methods, and learns a shape editing model by learning contrasts between sampled shape pairs.

ShapeTalk [3] is the most similar work to our proposed dataset, and is the only publicly available dataset specifically designed for language-guided 3D shape-to-shape editing. ShapeTalk is a remarkable contribution in the field of language-guided shape editing, and is one of the first works to leverage large-scale 3D shape datasets to facilitate the study of language-guided shape editing. However, this work has some limitations that we attempt to address in our work. Collecting a dataset of shape differences is a challenging task requiring considerable manual annotation effort, and gathering a large number of 3D shapes may not be feasible for all domains. Furthermore, ShapeTalk is composed of edit contexts (i.e. shape pairs and edit instructions) which are hard to separate into fine-grained, composable edits. Most of the time, singular shape edits do not have an exact ground-truth in the form of a source and target shape pair. We illustrate this important distinction in Figure 2. The availability of singular shape edits with an associated ground-truth would ease benchmarking the quality of shape editing methods, and facilitate the training process of editing models. ShapeTalk also does not provide a mechanism to generate edit chains, which are necessary to study compositional shape editing. Finally, ShapeTalk does not easily enable the study of shape editing in a compositional manner. In contrast, our dataset is 1) synthesized by augmenting a small set of diverse shapes and can easily be scaled up, 2) composed of fine-grained edits and coarse-grained edits with exact ground-truths, and 3) separated into edit chains, which are designed to facilitate the study of compositional shape editing.

3. ShapeWalk

Our dataset contains 158K shapes split into a random and realistic set. With each edge connecting two consecutive shapes, we also produce a text instruction generated using the parameter changes necessary to transition from one shape to the next. We detail the generation method of our dataset here, and summarize the process in Figure 3.

3.1. Dataset

Definition. Our dataset can be defined as a collection of directed graph paths (dipaths), denoted as \( P^{(k)} = (S^{(k)}, E^{(k)}, f^{(k)}) \). For each shape chain \( P^{(k)} \) of length \( l \), we denote:
- \( S^{(k)} = \{ s_0^{(k)} , \ldots , s_{h_l}^{(k)} \} \), the set of distinct shape nodes composing the chain. Each shape is defined by a set of parameters \( \theta_i \subseteq \Theta \), where \( \Theta \) is the linear parameter space of our shape program.
- \( E^{(k)} \subseteq \{ (s_i^{(k)}, s_j^{(k)}) | s_i^{(k)}, s_j^{(k)} \in V^{(k)}, j = i + 1 \} \), the set of edges linking each consecutive pair of shapes.
- \( f^{(k)} : E^{(k)} \mapsto \{ (\overline{\theta_{ij}}, \mu_{ij}) | \overline{\theta_{ij}} \subseteq \Theta, \mu_{ij} \in \Sigma \} \), a function mapping each edge to a vector \( \overline{\theta_{ij}} \), \( \theta_{ij} \) defines the parameter changes, or edits necessary, to go from shape \( i \) to \( j \). \( \mu_{ij} \) is a set of text instructions describing this edit in natural language.

Generation. To generate our dataset, we start by reconstructing a set \( S \subseteq \mathcal{S} \) of realistic 3D CAD shapes into the space of shapes covered by our shape program \( \mathcal{S}_\theta \subseteq \mathcal{S} \). To that end, we utilize shapes from the 3DCoMPAT** [15, 22] dataset, a realistic, industry-based 3D CAD dataset. We employ this dataset to avoid overlap with ShapeNet [5], and to ensure diversity and visual quality of the shapes.

We define a visual similarity function \( d : \mathcal{S} \times \mathcal{S} \mapsto \mathbb{R} \) between two shapes \( s_i, s_j \in \mathcal{S} \) as the feature-wise mean squared error between the original and reconstructed meshes’ renderings, in the space of a pre-trained ResNet-50 [10] encoder \( \phi_R : \mathbb{R}^{b \times w \times h} \mapsto \mathbb{R}^d \).

\[
d(s_i, s_j) = \| \phi_R(f_R(s_i)) - \phi_R(f_R(s_j)) \|_2^2
\]

Where \( f_R : \mathcal{S} \mapsto \mathbb{R}^{b \times w \times h} \) is a rendering function. The set of reconstructed shapes can then be defined as:

\[
\underset{|S_R| = (1-\alpha)|S|}{\arg\min} \sum_{s \in S_R} d(s, \hat{s})
\]

Where \( \hat{s} \) is the reconstructed shape, defined as:

\[
\hat{s} = f_{\theta} \circ \phi_R(s)
\]
Figure 3. Detailing the shape generation pipeline of ShapeWalk. Realistic 3D CAD shapes are reconstructed from the 3DCOMPA** dataset into the GeoCode shape program parameter space, using a mapping function \( \phi_\Theta \). Reconstructed shapes with an error over a fixed threshold are discarded using a visual similarity function \( d \), which is based on rendered feature similarity. Filtered pairs of shapes are then interpolated in the parameter space of the shape program, to generate shape chains \( \mathcal{P}^{(k)} \).

With \( f_\Theta : \mathcal{S} \rightarrow \Theta \) the shape program mapping shape parameters to a 3D mesh, and \( \phi_\Theta : \mathcal{S} \rightarrow \Theta \) an encoder based on DGCNN \([17, 25]\) which samples points from a shape surface and regresses corresponding shape parameters in \( \Theta \). We reconstruct shapes by first sampling pointclouds from the surface of the original meshes, and then feeding them into the DGCNN \([25]\) encoder fine-tuned \([17]\) to regress the parameters of the shape program. We discard a ratio of \( \alpha = 0.08 \) of the shapes with the highest reconstruction error, which we measure as the feature-wise mean squared error between the original and reconstructed meshes' renderings, in the space of a pre-trained ResNet-50 \([10]\) encoder. This threshold is selected empirically by visually inspecting the shapes in the upper part of the reconstruction error distribution. In our dataset instance building on 3DCOMPA** shapes, we filter out a total of 89 shapes during this process, leading to a set of \( |S_R| = 1113 \) reconstructed shapes. This process can be scaled up to any dataset size and any shape domain, as long as a corresponding shape program is available.

Shape interpolation. After reconstructing a set \( S_R \) of realistic shapes, we generate a set of shape chains \( \mathcal{P}^{(k)} \) by interpolating between the parameters of shape pairs. To that end, we first sample a set of shape pairs \( (s_{\theta_i}, s_{\theta_j}) \in S_P \subseteq S_R \times S_R \), and partition them into \( L = 10 \) levels of proximity. These proximity levels are based on the feature-wise mean squared error between the original and reconstructed meshes’ renderings, in the space of a pre-trained ResNet-50 \([10]\) encoder.

We define the set of shape pairs \( S_P^{(l)} \) at level \( l \in [1, L] \) as:

\[
S_P^{(l)} = \{(s_{\theta_i}, s_{\theta_j}) \in S_P \mid \bar{d}(s_{\theta_i}, s_{\theta_j}) \in \left[\frac{l-1}{L}, \frac{l}{L}\right]\}
\]

Where \( \bar{d} \) is a visual similarity function, min-max normalized across all pairs of shapes.

For each pair \( (s_{\theta_i}, s_{\theta_j}) \in S_P^{(l)} \), we then interpolate between the shape parameters \( \theta_i \) and \( \theta_j \) to generate a set of intermediate shape parameters \( (\theta_1, \ldots, \theta_{(N-2)}, \theta_j) \in \Theta^N \). The number of intermediate shapes \( N \) is defined as the number of differing parameters between \( \theta_i \) and \( \theta_j \):

\[
N = \{|k \mid \theta_i[k] \neq \theta_j[k]\}\]

When the proximity level is low, edits necessary to transition from one shape to another are larger in intensity as the shapes are more different, and smaller when \( l \) is smaller. The ordering of intermediate edits is randomly sampled using a dependency-aware algorithm (for example, parameters relating to armrest height or width are sampled after the addition of armrests).

As illustrated in Figure 1, this generation process leads to plausible interpolated shapes, and to fine-grained and realistic sequences of edits with detailed metadata. We summarize the process of generating realistic shape chains in Figure 3.

Random set. While the realistic set is based on reconstructed shapes from the 3DCOMPA** dataset, the random set is based on a collection of random shapes generated using the shape program. This alternative set aims at covering a large space of the parameter space of the GeoCode shape program. This subset is generated by systematically sampling various combinations of parameters within defined ranges determined by a minimum edit intensity, but does not necessarily result in realistic shapes.

3.2 Text Instructions

Rule-based generation. Given an edit vector \( \vec{e}_{ij} \) describing the parameter changes necessary to transition from shape \( i \) to \( j \), we generate a set of text instructions \( p_{ij} \) describing this edit in natural language. We first map each parameter \( \theta_i[k] \) composing the edit vector to a natural language name, and map the magnitude of the
Figure 4. Dataset chains samples. We feature in this figure shape chains from our dataset with associated generated text instructions. Each shape chain is composed of a sequence of shapes, with each consecutive pair of shapes associated to a set of text instructions describing the edits necessary to transition from one shape to the next. Overall, generated text instructions are accurate and provide a detailed description of the edits applied to the shapes. We include in the supplementary material additional samples of generated shape chains, alongside shape chains generated for two additional table and vase categories.

4. Neural Shape Editing

4.1. Problem

We are concerned with the task of editing a shape \(s_i\) given a sequence of natural language edit instructions. We want to learn an editing function \(f_E : S \times \Sigma \rightarrow S\), where \(S\) is the space of input shapes, and \(\Sigma\) is the space of natural language edit instructions, able to compose edits from a starting shape \(s_1\) to an ending shape \(s_N\), given a sequence of edit instructions \(\{p_t\}_{t=1}^{N}\).

\[
\hat{s}_N = f_E(\phi_T(p_{12}), f_E(\phi_T(p_{23}), \ldots, f_E(\phi_T(p_{kN}), s_1))
\]

Where \(N = |P|\) is the length of the edit chain, and \(\phi_T : \Sigma \rightarrow \mathbb{R}^{D_T}\) is a text encoder. At the end of the edit sequence, we want to recover a shape \(\hat{s}_N\) as close as possible to the ground-truth shape \(s_N\). Our proposed dataset provides a ground-truth for every intermediate shape \(s_k\), which we leverage to train and evaluate our method.
Figure 5. Visualizing synthesized pairs of edits using our latent editors. We illustrate the diverse range of edits generated by our proposed latent mapper, showcasing its ability to capture nuanced variations in the input data, for 3DS2VS latents (top two rows) and PC-AE latents (bottom two rows). Each pair demonstrates the original input on the left and the corresponding synthesized output on the right, with the associated caption. We abbreviate instructions which are too long to fit in the figure, and indicate when text is omitted with an ellipsis. For more subtle changes, we circle the areas of interest in both source and target shapes. Overall, edits are consistent with the input instruction, preserve shape identity while generating plausible output shapes.

4.2. Objective

One of the main advantages of training a shape editing model with ShapeWalk compared to other works [2, 8] is the availability of exact ground-truth edited shapes for each edit instruction (see Section 2). We thus formulate the objective function of our learning problem as a simple $L_2$ loss between the latents of the synthesized shape and the ground-truth shape. For all pairs of shapes $(s_i, s_j)$ and their corresponding edit instructions $p_{ij}$, we minimize the following loss:

$$
\mathcal{L} = \| f_E(\phi_E(s_i), p_{ij}) - \phi_E(s_j) \|_2
$$

Where $\phi_{AE} = \phi_E \circ \phi_D$ is a shape autoencoder, and $\phi_E : S \mapsto \mathbb{R}^{D_E}$ denotes its encoder component.

4.3. Models

For $\phi_{AE}$, we experiment with PC-AE [1] and 3D2VS [28] models both pre-trained on ShapeNet [5]. Note that our method is agnostic to the choice of shape autoencoder and can be adapted to a variety of shape representations. For $\phi_T$, we use a pretrained BERT [7] model.

Both $\phi_{AE}$ and $\phi_T$ are frozen during training, and we only train the parameters of our latent mapper $f_E$. 
For PC-AE, we formulate our latent mapper as a neural module which for each tuple \((\phi_E(s_i), \mathbf{p}_i)\) predicts an edit vector \(\tilde{\theta}_{ij} \in \mathbb{R}^{D_{\mathbf{E}}} \) in the feature space of \(\phi_E\), and adds it to the latent of the input shape \(s_i\) to generate the latent of the output shape \(s_j\):

\[
f_E(\phi_E(s_i), \mathbf{p}_i) = \tilde{\theta}_{ij} + \phi_E(s_i)
\]

We utilize two variants of our latent mapper \(f_E\): one in which the edit vector \(\tilde{\theta}_{ij}\) is predicted directly, and one in which the edit vector \(\tilde{\theta}_{ij}\) is predicted as a product of a normalized edit direction \(\tilde{v}_{ij} \in \mathbb{R}^{D_{\mathbf{E}}}\) and a magnitude \(\overline{m}_{ij} \in \mathbb{R}\) separately.

For 3D2VS which encodes shapes as latents sets of higher resolutions, we use a transformer-based latent diffusion model instead to generate edited latents. We concatenate the input shape latent \(\phi_E(s_i)\) with the noised edited shape latent \(\phi_E(s_j + \epsilon_j)\) and feed the BERT text embeddings to the cross-attention layers of the transformer blocks to predict the added noise. We illustrate our architecture for the latent mapper in the supplementary material.

5. Experiments

5.1. Chained Shape Editing

**Comparison Models.** For PC-AE, we experiment with three variants of our latent mapper module:

- **DIRECTGEN** directly predicts the edited shape latent without regressing an edit vector.
- **LATEFUSION** follows the network proposed in [3], and first passes the shape embeddings through an encoder. Encoded shape features are then concatenated with text embeddings and fed into a neural module which finally predicts the edit vector.
- **OURS** is a simple multi-layer perceptron (MLP) which directly takes the context (shape and text embeddings) as input to predict an edit vector.

For all of these variants, we experiment with various bottleneck dimensions and number of layers (in subscript).

We further decompose these variants into coupled and decoupled versions, where we predict a normalized edit direction \(\tilde{v}_{ij} \in \mathbb{R}^{D_{\mathbf{E}}}\) and a magnitude \(\overline{m}_{ij} \in \mathbb{R}\) separately. We provide in appendix the full architecture details and training hyperparameters employed to train our models.

Note that we do not compare with ChangeIt3D [2] as their neural-listener distillation method is not applicable in our context: we train our models with a direct feedback signal from the ground-truth edited shapes.

**Single-step metrics.** We propose to measure the quality of our editing steps both before and after shape reconstruction. Since we experiment on pointclouds, we use the scaled Chamfer Distance between the reconstructed original shape \(s_i\) and the reconstructed edited shape \(s_j\):

\[
d_{\text{CD}} = \frac{1}{|s_j|} \sum_{x \in s_j} \min_{y \in s_i} \|x - y\|_2 + \frac{1}{|s_i|} \sum_{y \in s_i} \min_{x \in s_j} \|x - y\|_2
\]

We also use the \(L_2\) distance between the latents of the original shape \(s_i\) and the edited shape \(s_j\):

\[
d_{L_2} = \|\phi_E(s_i) - \phi_E(s_j)\|_2
\]

**Multi-step metrics.** To extend the single-step metrics to chained shape generation, we take inspiration from the trajectory prediction literature [4, 26] and propose appropriate metrics for our task.

Given a shape distance function \(d : S \times S \rightarrow \mathbb{R}\), a chain of ground-truth reconstructed shapes \(\{s_1, \ldots, s_N\}\), and a set of recursively generated shapes \(\{\tilde{s}_1, \ldots, \tilde{s}_N\}\), we define the average edit error as the average distance between the generated shapes and the corresponding ground-truth shapes:

\[
A_d = \frac{1}{N} \sum_{t=1}^{N} d(s_t, \tilde{s}_t)
\]

And the final edit error as the distance between the last shape in the chain and the corresponding ground-truth shape:

\[
F_d = d(s_N, \tilde{s}_N)
\]

We report results with both metrics for \(d = d_{\text{CD}}\) and \(d = d_{L_2}\).

**Results.** We provide qualitative results of edited pairs generated by our latent mapper in Figure 5. Overall, we observe that our method is able to generate plausible edits that are consistent with the input instruction, and preserve the identity of the input shape. However, our method is limited by the quality of the shape autoencoder. We notice that the PC-AE editor is unable to perform very fine-grained edits which are not properly captured by the autoencoder, while visible in the ground-truth shapes. We explore the possible causes of these limitations in Section 5.2.

We provide quantitative results for our chained editing metrics averaged across \(|\mathcal{P}| \in \{10, 15, 20\}\) in Table 1, and detail per-chain length results in the appendix. Directly predicting the edited shape latent (DIRECTGEN) leads to poor edit predictions across all metrics, although increasing the number of layers may improve the results. Overall, decoupling the edit direction and magnitude leads to better results. Our coupled OURS\(_{512\times4}\) variant performs better than the coupled LATEFUSION\(_{512}\) variant on the Chamfer Distance, but worse on \(L_2\) distance.
We use an MLP-based shape latent encoder to encode a pair of shapes. To edit, we train a binary classifier to discriminate from a realistic set, alongside the average accuracy for the full random set, as well as on subsets of the data corresponding to specific types of edits.

Results. In Table 2, we provide accuracy results for shape edit recognition, depending on the edit type. We also provide the global accuracy on the random and realistic sets. We observe that our classifier is able to recognize edits corresponding to shape parameters associated with a high degree of shape variation (like seat height and width/height of the shape), but struggles with parameters which are more subtle (like legs roundness, handle cushions). Our classifier is able to detect edits related to global width/height/depth adjustments with an accuracy of 97.8%. This accuracy drops to 94.4% for legs bending/curvature, which is a more subtle change in the shape, and to 58.7% for legs roundness/indentation. We also note that the model does not perform well on edits related to the number of parts, i.e. adding or removing a number $k$ of legs or backrest rails.

We impute these discrepancies to several factors. The PC-AE shape autoencoder we employ may be unable to properly reconstruct fine-grained details in the ground-truth and edited shapes, which renders the task of detecting subtle changes more difficult. Pointcloud resolution is also a limitation, as fine-grained parameters like leg indentation may not be properly sampled. Finally, learning a latent mapper and a classifier that can accurately count the number of parts to add or remove may also be a difficult task of its own, and require more domain-specific inductive biases.

6. Conclusion

In conclusion, this paper introduces the ShapeWalk dataset, designed for advancing compositional shape editing guided by natural language instructions. The dataset comprises 158K unique shapes connected through 26K edit chains, synthesized from a realistic CAD-designed 3D dataset. Language instructions for applied edits are provided, alongside exact ground-truth edited shapes. Our method requires zero human annotation effort, and can be scaled up indefinitely. The usefulness of the dataset is demonstrated by training neural editor modules in the latent space of shape autoencoders [1]. Evaluation metrics inspired by trajectory prediction literature [4, 26] are introduced, offering a quantitative assessment of the quality of recursively edited shapes.
References


ShapeWalk: Compositional Shape Editing through Language-Guided Chains

Supplementary Material

Code and Dataset

Our dataset, generation code, model checkpoints and training scripts will be made available upon publication at:

https://shapewalk.github.io/

Appendix Overview

This supplementary document is organized as follows:

• Section 1 provides additional analysis of our latent editor models.
• Section 2 provides additional implementation details about our dataset generation and model training processes.
• Section 3 gives shape reconstruction examples.
• Section 4 showcases edit chain samples from our dataset, including edit chains for additional table and vase classes, and dataset statistics.
• Section 5 illustrates the sampling process for the 3D2VS latent diffusion model.

1. Analysis

We differentiate in the additional results in this section between CD — REC and CD — REAL metrics, which respectively measure the distance to the reconstructed edited shape and the real edited shape. The latter metric is more relevant for our task, as it measures the final editing ability of the model (reported in the main paper).

1.1. Chained Shape Editing

We provide detailed results for the chained shape editing task described in the main paper, in Table 2. We report baseline results for our proposed chained shape editing task, for chain lengths |P| ∈ {10, 15, 20}. Overall, while no clear trend is observed for the Chamfer Distance, we observe that the average edit error for L₂ distance decreases with longer chains. This could be explained by the fact that longer chains contain more sequences of fine-grained edits which are less likely to make the feature representations diverge abruptly. On the other hand, shorter chains are more likely to contain sequences of high-magnitude edits, which amplify the cumulative prediction error. In that sense, a parallel could be made with the long-horizon prediction problem in trajectory forecasting, where the prediction error accumulates over time [2].

1.2. Oracle Editing

We provide detailed results for the chained shape editing task when oracle magnitudes and directions are used, in Table 2. Overall, the biggest gain for PC-AE based models is observed when using oracle directions. This suggests that the main bottleneck for our models is the edit direction prediction task, which is more challenging than predicting the magnitude from the input prompt.

1.3. Predicting Edit Magnitudes

Our decoupled latent editor models predict both a normalized edit direction ̂v_{i,j} and an edit magnitude ̂m_{i,j} for each edit. We use this property to analyze the correlation between the predicted edit magnitudes and the ground truth edit intensities extracted from our metadata. We extract decoupled edit vector predictions from our models on the random subset, and compare them to the ground truth edit intensities. In order to do that, we min-max normalize the predicted edit magnitudes and bin them in the [1, 9] interval. Ground-truth edit intensities are extracted from the ground-truth edit vectors, similarly by discretizing the magnitude of changes in the same interval. We plot confusion matrices comparing these predicted discretized edit magnitudes to the ground truth edit intensities in Figure 1. We show results for three decoupled baselines: LATEFUSION1024, LATEFUSION512, and OURS512×4.

Overall, we observe a strong correlation between the predicted edit magnitudes and the ground truth edit intensities across all models. However all models tend to deviate from the ideal diagonal magnitude predictions. We remark also that the OURS512×4 model tends to deviate from the diagonal and avoids predicting the maximum possible magnitude range. One way to alleviate these biases could be to introduce explicit supervision to the edit magnitude prediction task using the ground truth edit magnitude labels. For example, a contrastive loss could incentivize the model to predict edit magnitudes that align with the ground truth ranking of edit intensities.

2. Implementation Details

2.1. Shape Chain Generation

We detail in Figure 9 the process of sampling parameter chains from the parameter tree. Each parameter vector 但仍 can be represented as a dependency tree. We start by sampling a parameter tree 但仍 using regressed parameters from a real source shape in the realistic subset. We then successively sample parameters to edit by interpolating towards a target shape in the realistic setting.

In the random setting, the source shape and subsequent parameters are sampled randomly.
The generated instruction is parallel to the edit vector, we map the edit intensity and the parameters with a continuous domain, we sample a random range. After generating each parameter vector \( \mathbf{x} \), we observe when using oracle directions.

### Table 1. Chained Shape Editing Ablation

We report detailed baseline results for our proposed chained shape editing task, for chain lengths \( |P| \in \{10, 15, 20\} \), using the PC-AE trained latent editors. Both the average final error and average edit error are reported for the Chamfer Distance (CD) and \( EAL \) distance (\( EAL \)) metrics. We differentiate between distances to the reconstructed edited shape (CD – RE) and distances to the real edited shape (CD – REAL), which is the most relevant metric for our task. We highlight in grey the model we select for our qualitative results.

| Model        | \( |P| = 10 \) | \( |P| = 15 \) | \( |P| = 20 \) |
|--------------|----------------|----------------|----------------|
|              | \( \text{FID}_{	ext{rec}} \) | \( \text{Act}_{	ext{rec}} \) | \( \text{Act}_{	ext{rec}} \) | \( \text{FID}_{	ext{rec}} \) | \( \text{Act}_{	ext{rec}} \) | \( \text{Act}_{	ext{rec}} \) | \( \text{FID}_{	ext{rec}} \) | \( \text{Act}_{	ext{rec}} \) | \( \text{Act}_{	ext{rec}} \) |
| LATEFUSION   | 1.751          | 1.431          | 3.006          | 2.756          | 1.813          | 1.707          | 2.139          | 1.678          | 2.041          | 2.925          | 1.906          | 1.241          | 1.791          | 0.844          | 2.628          | 2.182          | 1.054          | 0.885          |
| LATEFUSION\(_{w1}\) | 1.802          | 1.413          | 3.011          | 2.801          | 2.018          | 1.770          | 2.000          | 1.499          | 2.624          | 2.755          | 1.999          | 1.221          | 1.414          | 1.065          | 2.361          | 2.272          | 1.059          | 0.860          |
| LATEFUSION\(_{w2}\) | 1.887          | 1.486          | 3.026          | 2.928          | 2.879          | 1.810          | 2.034          | 1.380          | 2.852          | 2.718          | 1.342          | 1.210          | 1.737          | 1.018          | 2.743          | 2.106          | 1.106          | 0.818          |
| OUR\(_{w1}\)   | 2.148          | 1.605          | 3.680          | 3.109          | 2.311          | 2.012          | 2.473          | 1.664          | 3.547          | 2.956          | 1.961          | 1.351          | 1.571          | 0.992          | 2.599          | 2.341          | 1.207          | 0.817          |
| OUR\(_{w2}\)   | 1.751          | 1.494          | 3.236          | 2.859          | 2.168          | 1.884          | 2.376          | 1.560          | 3.098          | 2.870          | 1.452          | 1.271          | 1.659          | 0.867          | 2.638          | 2.466          | 1.166          | 0.808          |
| LATEFUSION\(_{w1}\) | 1.733          | 1.452          | 3.066          | 2.794          | 2.002          | 1.714          | 1.708          | 1.200          | 2.707          | 2.667          | 1.212          | 1.184          | 1.902          | 1.649          | 2.279          | 2.242          | 0.989          | 0.817          |
| LATEFUSION\(_{w2}\) | 1.687          | 1.430          | 2.694          | 2.098          | 1.883          | 1.672          | 1.978          | 1.328          | 3.069          | 2.749          | 1.300          | 1.190          | 2.149          | 1.707          | 3.224          | 2.678          | 1.170          | 0.900          |
| LATEFUSION\(_{w3}\) | 2.057          | 1.555          | 3.071          | 2.824          | 2.043          | 1.743          | 2.386          | 1.650          | 3.342          | 2.967          | 1.445          | 1.305          | 2.307          | 1.394          | 3.514          | 2.734          | 1.170          | 0.923          |
| OUR\(_{w1}\)   | 1.627          | 1.358          | 3.243          | 2.858          | 2.887          | 1.808          | 1.902          | 1.351          | 3.018          | 2.679          | 1.359          | 1.234          | 1.733          | 0.865          | 2.717          | 2.214          | 1.035          | 0.826          |
| OUR\(_{w2}\)   | 1.647          | 1.438          | 3.028          | 2.826          | 1.941          | 1.705          | 1.698          | 1.200          | 2.713          | 2.582          | 1.328          | 1.211          | 1.450          | 0.884          | 2.268          | 2.163          | 1.053          | 0.821          |

Table 2. Oracle editing results. We report detailed baseline results when oracle magnitudes and directions are used for the chained shape editing task, on all LATEFUSION models. We differentiate between distances to the reconstructed edited shape (CD – RE) and distances to the real edited shape (CD – REAL), which is the most relevant metric for our task. Overall, the biggest gain for PC-AE based models is observed when using oracle directions.

When boolean parameters with children are triggered, we sample a random value for each child parameter. For scalar parameters with a continuous domain, we sample a random value from a discretized version of the parameter’s range. After generating each parameter vector \( \theta \), we pass it through a geometry checker (implemented in [12]) ensuring that the resulting shape is valid. If the shape is invalid, the whole chain is discarded and we start over from a different starting pair. Otherwise, the corresponding mesh is synthesized using the shape program \( \phi_{\theta} \) and added to the chain.

### 2.2. Text Instructions Generation

We detail in Figure 2 the process of synthesizing text instructions for the chained shape editing task. Starting from an edit vector, we map the edit intensity and the parameter name to a predefined vocabulary set which is randomly sampled. Optionally, the generated instruction is paraphrased using a pre-trained language model. Our instruction generation method is completely automatic and does not require any human annotation or additional data.

### 2.3. Architecture details

#### Rendered View Feature Extractor

We use a ResNet-50 [7] model pre-trained on ImageNet [5] to extract rendered view features from the input shapes. Since all synthetic meshes are centered at the origin and aligned, we do not require multi-view features to ensure invariance to rotation. Every mesh is rotated around the z-axis by an angle \( \theta = \frac{\pi}{4} \), assigned a base RGB color and rendered from a static viewpoint using the trimesh [4] library.

#### Autoencoders

We encode pointclouds using a PC-AE [1] model pre-trained on the ShapeNet [3] dataset with a latent dimension \( d = 256 \). We also use a 3D2VS [17] model also pre-trained on ShapeNet with a latent dimension \( d = 512 \times 8 \). Note that our model is agnostic to the choice of autoencoder model, and can be adapted to any other shape or pointcloud representation.

#### PC-AE Latent Editors

We detail here the architecture of our latent editor models.

- LATEFUSION\(_{w1}\) is composed of a shape latent encoder,
Comparing predicted edit magnitude to ground truth edit intensity. We plot confusion matrices comparing the predicted edit magnitudes binned in the interval [1, 9] to the ground truth edit intensities extracted from our metadata. We show results for three decoupled baselines: \textsc{LateFusion1024}, \textsc{LateFusion512}, and \textsc{Ours512×4}. Overall, we observe a strong correlation between the predicted edit magnitudes and the ground truth edit intensities. However, a bias can be observed in the \textsc{Ours512×4} model, which tends to deviate from the diagonal predictions, and avoids predicting the maximum possible magnitude range.

Synthesizing edit instructions. Our method generates synthetic text instructions by first applying rule-based generation to translate parameter changes into natural language. The edit intensity is mapped to a natural language intensity depending on the parameter type. To add diversity, the vocabulary describing the edit is randomly sampled, edit directions are randomly inverted, and a paraphrasing transformer model is optionally employed to augment the final instructions.

3D2VS Latent Editors. We fine-tune a latent diffusion model to encode the language instructions into a 768-dimensional feature vector. We employ the base uncased model with 12 layers, 12 self-attention heads, and 110M parameters. Training with larger instances of BERT or with LLM-extracted text features could potentially improve the generalization ability of our models on unseen language instructions.
2.4. Training

Shape Sampling. We sample $N = 4096$ points from each synthesized mesh using the triangle point sampling scheme [16]. Pointclouds sampled from synthetic shapes are centered at the origin, normalized to the unit sphere, and rescaled alongside each axis to align with ShapeNet statistics. A minor downside of normalizing shapes is that the model will only be able to learn to apply edits relative to the scale of the input shapes.

PC-AE Latent Editor. We train our models using the Adam optimizer [10] with a learning rate of $1 \times 10^{-4}$, and an effective batch size of 128. The learning rate is linearly increased from $1 \times 10^{-6}$ to $1 \times 10^{-4}$ during the first 8 epochs, and then gradually decayed back to $1 \times 10^{-6}$ following a cosine annealing schedule [11]. We train our models on two NVIDIA V100 GPUs with 32GB of memory each. Training and sampling strategies are the same as in [9, 17]. All text and pointcloud features are pre-extracted and cached to disk to speed up training.

3. Shape Reconstructions

We illustrate in Figure 12 the top-8 and bottom-8 shape reconstructions from the 3DCoMPaT [13] dataset into the GeoCode representation, ranked by rendered view feature similarity. Overall, we observe that the reconstructions are of high quality, and that the model is able to capture the main shape features of the input shapes. The worst reconstructions are discarded from the shape interpolation process as they do not generally lead to realistic reconstructed shapes.
Figure 6. **ShapeWalk dataset statistics.** We plot the distribution of the number of shapes per chain (left), of the intensity of edits (middle), and of the complexity of edits (right). For the edit complexity, note that the frequency is provided in log-scale. The vast majority of edits in the realistic subset are granular, affecting a single shape parameter.

In order to further investigate the quality of the reconstructions, we compute t-SNE embeddings [15] of the rendered view features of the input shapes and their corresponding reconstructions in the space of $\phi_B$, the rendered view feature extractor. We show the results in Figure 5. While reconstructed (in purple) and source shapes (in yellow) from the 3DCoMPaT [13] dataset form two distinct clusters, we observe that the reconstructions are generally close to the input shapes in the feature space. The filtered reconstructions (in blue) are more likely to be outliers in the feature space, and are thus discarded from the shape interpolation process to avoid unrealistic shape interpolations.

4. **Dataset Insights**

4.1. Shape Chains

In Figures 10, and 11, we showcase sampled shape chains from our dataset truncated to the first $N = 9$ shapes. For each chain edge, we show the corresponding language instruction describing the parameter changes necessary to transition from one shape to the next.

4.2. Dataset Statistics

In Figure 6, we plot the distribution of the number of shapes per chain (left), of the intensity of edits (middle), and of the complexity of edits (right) in the realistic subset\(^2\). The edit complexity corresponds to the number of shape parameters affected by the edit. For edit complexity, the frequency is provided in log-scale. We observe that the vast majority of edits in the realistic subset are granular, and affect a single shape parameter. Chain lengths range from $N = 6$ to $N = 29$ shapes, with an expected length around $N = 15$ shapes. The intensity of edits is not uniformly distributed and skews heavily towards low-intensity edits, which is expected.

---

\(^2\)Note that we do not show statistics for the random subset as all parameters including chain length and edit intensity are static and determined at generation.

5. **Diffusion-based Editor Generation Process**

We illustrate in Figure 7 the process of generating a shape chain using a diffusion-based latent editor model. We showcase the generation of a shape chain from a source shape to a target shape using our latent editor operating in the space of the 3D2VS autoencoder, illustrated in Figure 3. We show 32 intermediate shapes generated by the diffusion model for each sampling step.
Figure 7. **Diffusion-based shape chain generation.** We illustrate in this figure the process of generating a shape using a diffusion-based latent editor model, for a source shape (blue) and a text instruction, to obtain a final target shape (orange) after $T = 32$ diffusion steps.
Figure 8. **Chain samples for additional classes.** We showcase in this figure additional chain samples for the *realistic* subset, for the *table* and *vase* classes. Our method is able to generate realistic shape chains for a variety of classes, given support for the corresponding shape programs. In our case, GeoCode [12] supports the classes we use in this work. Extending our method to new classes will require extending these underlying shape programs.
Figure 9. **Parameter tree sampling.** We detail in this figure the process of sampling parameter chains from the parameter tree. Each parameter vector $\theta_i$ can be represented as a dependency tree. We start by sampling a random parameter tree $\theta_0$ from the root node of the tree (or using regressed parameters from a real shape in the realistic subset). Edit parameters are then sampled randomly (or using interpolation for the realistic set) from the tree until the desired chain length is reached (or until the target shape is reached).
Figure 10. **Additional dataset samples.** We showcase sampled chains from our dataset truncated to the first $N = 9$ shapes. We color shapes based on their proximity to the starting shape (blue), and the ending shape (orange). For each chain edge, we show the corresponding language instruction describing the parameter changes necessary to transition from one shape to the next. All shapes are normalized to the unit cube and centered at the origin.
Figure 11. **Additional dataset samples.** We showcase sampled chains from our dataset truncated to the first $N = 9$ shapes. We color shapes based on their proximity to the starting shape (blue), and the ending shape (orange). For each chain edge, we show the corresponding language instruction describing the parameter changes necessary to transition from one shape to the next. All shapes are normalized to the unit cube and centered at the origin.
Figure 12. **Shape reconstruction examples.** We illustrate in this figure the original shapes from the 3DCoMPaT [13] dataset (yellow), and their corresponding reconstructions into the GeoCode representation (purple). In the top two rows, we show the top-8 reconstructions ranked by rendered view feature similarity, and the bottom-8 reconstructions ranked by rendered view feature similarity in the bottom two rows. While top reconstructions are generally of high quality, the bottom reconstructions are discarded from the shape interpolation process as they do not generally lead to realistic reconstructed shapes. We discard these shapes to avoid unrealistic shape interpolations in our shape chain generation process.