Procedural Content Generation in Minecraft via Disentangled Representation Learning Models

Tim Merino[1,*], Yifan Zhang[2] and Julian Togelius[1]

[1]*New York University, Tandon School of Engineering, 6 MetroTech Center, Brooklyn, NY 11201*
[2]*New York University, Stern School of Business, 44 W 4th St, New York, NY 10012*

Abstract

Learning disentangled representations of data is a well-studied problem in machine learning. Disentangled representations offer many advantages for downstream generative tasks, enabling interpretability and granular control by leveraging learned representation spaces with desirable properties. In modern generative pipelines, these representation learning models are typically just a stepping stone for downstream models, such as the popular diffusion model. In this paper, we seek to explore how these representation learning models can be utilized as stand-alone generation and editing models for Procedural Content Generation. We start by extending recent work in Disentangled Representation Learning to a unique domain — the discrete, categorical 3D world of Minecraft. We present a method for learning discrete, disentangled representations of Minecraft terrain by leveraging domain knowledge to incorporate strong inductive biases in our model. We first assume a dual-factor decomposition of Minecraft terrain into “style” and “structure” generative factors. Based on this assumption, we design a dual-codebook Factorized Quantization Variational Autoencoder, using codebook sizes significantly smaller than standard practice. We demonstrate how our model learns disentangled and human-interpretable representations of these generative factors, and attempt to quantify disentanglement using intervention-based metrics. Finally we show how our learned representation enables human-level editing and generation of Minecraft terrain features via human latent authoring.

Keywords

Procedural Content Generation, Controllable Generative AI, Disentangled Representation Learning, Human-in-the-loop Generation

1. Introduction

Learning disentangled representations of data is an active area of research in machine learning. The appeal of disentangled representations can be understood from various lenses. One can view Disentangled Representation Learning (DRL) as effort to build models that think like humans do — building a compositional semantic representation of our observation. [1, 2, 3]. Properly disentangled representations can also provide interpretable, explainable features. As Artificial Intelligence increasingly pervades aspects of everyday life, the need for explainable AI becomes quite pressing and clear. Finally, and most relevant to this work, DRL provides an alternate method for controllable generation, without the arduous requirement of labeled data [4].

The appeals of DRL have attracted researchers across a variety of fields. We largely take inspiration from work in the field of image generation, specifically the sub-field of neural style transfer.

The concept of disentanglement is often cited as lacking a formalized notion of disentanglement [5, 6], and standard metrics with which to measure its presence [6]. One accepted definition states [7]: “Disentangled representation should separate the distinct, independent and informative generative factors of variation in the data. Single latent variables are sensitive to changes in single underlying generative factors, while being relatively invariant to changes in other factors.” The core idea, that a set of explanatory factors of variation underlies real-world data, seems particularly true in procedurally generated game worlds.

Joint AIIDE Workshop on Experimental Artificial Intelligence in Games and Intelligent Narrative Technologies, November 10-11, 2025, Edmonton, Canada.
[*]Corresponding author.
tm3477@nyu.edu (T. Merino); yz9016@stern.nyu.edu (Y. Zhang); julian.togelius@gmail.com (J. Togelius)

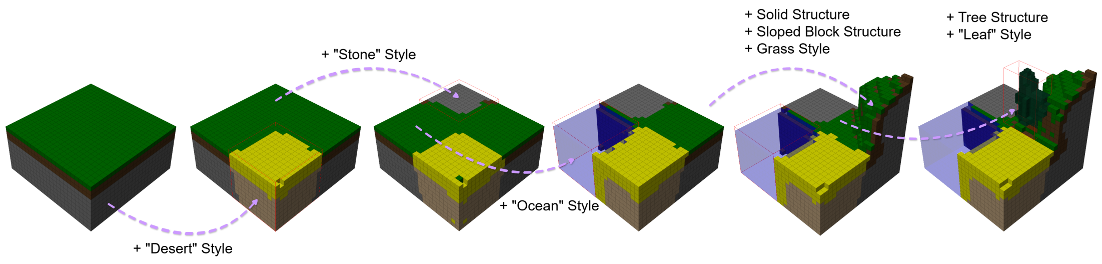


Figure 1: Human map editing and terrain feature generation enabled by our learned disentangled representation. Starting with a flat map, we use style and content transfer by editing the latent representations underlying four corners of the map. The first three steps operate solely on the style representation, while the final uses both structure and style edits to create a grassy mountain and tall tree.

Minecraft is an open world, 3D voxel-based survival game that is extremely popular in the gaming, education, and AI research communities [8, 9, 10, 11, 12]. Core to the game loop is a robust Procedural Content Generation (PCG) system that creates nearly infinite worlds for players to explore. These generated worlds contain a variety of biomes and terrain features, including mountains, caves, rivers, oceans, and deserts. This existing content generator provides an attractive data source for training generative models, easily providing infinite samples to train from. However, at first glance, there is no clear motivation for a Procedural Content Generation via Machine learning (PCGML) approach. If we already have a system that perfectly satisfies the needs of the game it operates in, why do we need another generator? This ”fundamental tension of PCGML” has a simple answer: control.

The field of Generative AI has been making massive leaps in controllable generative models. While text-to-image models such as DALL-E [13] and Stable Diffusion [14] offer undeniably impressive control over realistic images, these models require extensively labeled datasets — something in short supply for video games. Instead of the text-to-asset task, we focus on an alternate method control, using semantically meaningful representations of data to manipulate it in desired ways. By introducing controllability on top existing PCG systems, we introduce new and creative methods of interaction both design and play in simulated worlds.

Building upon recent advances in discrete representation learning, we attempt to answer the call put forth by [5]. First, we introduce a simple inductive bias via a two-stage decoder network, introducing a self-supervised objective for categorical voxel data. Second, we demonstrate the concrete, practical benefits of enforcing a specific notion of style-content disentanglement on the learned representations. Specifically, we demonstrate the potential for utilizing the learned representation for human-in-the-loop generation and editing.

Our contributions can be summarized as follows:

- We introduce a novel Factorized Quantization Variational Autoencoder that learns a disentangled representation of Minecraft voxel terrain data using considerably smaller codebook sizes by leveraging a two-stage architecture.
- We demonstrate that our model learns a meaningful and semantically rich representation of style and structural features, assigning human-interpretable labels to each codebook vector
- We demonstrate the potential applications of this model for controllable, interactive Procedural Content Generation via Machine Learning (PCGML).

2. Background and Related Works

2.1. Disentanglement

InfoGAN [15] is an early application of unsupervised DRL in GAN-based models, using a mutual information regularization objective between the latent variable and observation to encourage disentanglement. Using this approach, they learn interpretable latents that control digit type, rotation, and width of digits on MNIST, as well latents for controlling features such as hair style and emotion in images of celebrity faces.

Later works in image generative models narrow the scope of disentanglement, aiming to specifically to disentangle concepts of "style" from "content" in images [16, 17, 18, 19, 20]. This enables Neural Style Transfer, where learned representations are used to render the content of one image in the style of another. While the concept of style and content disentanglement is well-understood and well explored in image generation, there has been comparatively little exploration of this paradigm in the context of games. [21] applies CycleGAN for cross-domain image-style transfer to render videos of the game Fornite in the style of another game, Player Unknown's Battlegrounds. Other work evaluates style/content disentanglement pretrained Vision Transformer representations across a variety of video games, aiming to find domain-general embeddings of content [22].

While there has been at least one exploration into 3D neural style transfer [23], they limit their domain to binary voxels, using a geometric concept of style (e.g "cube" + "bunny"). We seek to explore content and style disentanglement in a new domain — the 3D categorical voxel world of Minecraft — using analogous concepts of "style" to those found in traditional image style transfer applications.

2.2. VQ Models

Vector Quantized Variational AutoEncoders (VQVAEs) are discrete representation learning models that extend the classic Variational AutoEncoder [24]. Rather than learning a continuous, compressed representation of input data, VQVAEs encode data using codes from a fixed-size codebook of vectors.

VQVAEs consist of at least two trainable networks: an encoder network E, and a decoder network G. Additionally, the model learns a codebook C, which contains a finite amount ($|C|$) of n-dimensional codebook vectors. The encoder model takes as input x, which is passed through the encoder to produce encoding $z_e = E(x)$. This encoding is then quantized by looking up the nearest codebook vector $z_q = argmin||z_e - z_c||$ in the quantization step. This codebook vector is then used as input to the decoder network, which generates a reconstruction $\hat{x} = G(z_q)$. A straight through estimator allows gradients to flow through the non-differentiable quantization process.

Semantic interpretability of codebook vectors is sometimes cited as a benefit of Vector Quantized models such as VQVAEs. However, without disentanglement-specific regularization or inductive biases, there is no guarantee that these latent vectors correlate with semantically relevant features of the data distribution [5]. [25] experimentally evaluate interpretability of VQ models in model-based Reinforcement Learning. Using Grad-CAM[26], they measure conceptual similarity between patches of the input image that are salient to each code. They find that the codes have no guarantee of uniqueness and limited impact on concept disentanglement, with codes rarely exceeding the embedding similarity of randomly cropped image patches.

In practice, Vector Quantization models are often used as a first-stage for downstream generative tasks. Both Latent Diffusion Models and autoregressive approaches have achieved state-of-the-art image generation performance by utilizing discretized intermediate representations[27, 28, 29]. Codebook sizes for these use-cases can be extreme, exceeding sizes that are feasibly human-interpretable even if codes are semantically meaningful.

To counter the issues introduced by large codebook sizes, [30] propose Factorized Quantization Generative Adversarial Networks (FQGAN). Rather than using a single large codebook, they learn multiple smaller sub-codebooks. Using a disentanglement regularization term, they ensure their sub-codes contain unique and complimentary information about the input. We directly build upon their work, motivated by a need for small and interpretable conceptual codebooks.

2.3. PCGML in Minecraft

This work is most closely related to existing PCGML research in Minecraft. Existing approaches largely leverage Generative Adversarial Networks (GANs) for terrain and structure generation [31, 12, 32]. Additionally, the Generative Design in Minecraft (GDMC) competition [33] has challenged participants to create AI agents that produce functional and aesthetically pleasing settlements in Minecraft. This work diverges in both scope and goal — we omit structures from our dataset, and focus on a new method of controllable generation via manual latent intervention.

3. Method

3.1. Dataset

We create a training dataset from Minecraft's built in procedurally generated world system. We generate a full Minecraft world using a random seed, using the default world generation settings as of Minecraft version 1.12.2. We disable the "generate structures" setting, preventing houses and villages from being generated.

Using the Evocraft API [34], we collect cubes of terrain with a size of 24 voxels by traversing the world in a spiral along the X-Z plane[1]. We stride our sampling window by 36 blocks each step to improve biome diversity. We collect each sample from the surface of the world, centered on the Y-coordinate of the highest non-air block. We collect 11119 samples of terrain to use as our dataset, which includes 42 naturally occurring block types. We one-hot encode this data and apply rotation augmentations around the X and Y axis to improve our models generalization ability. The final shape of each terrain chunk after processing becomes $24 \times 24 \times 24 \times 42$

3.2. Disentanglement

We start by defining the two types of information (or generative factors) we want to disentangle from our data. In this work, we choose two factors commonly used in image generation: style and content.

The specific factors of variation these terms describe varies greatly by dataset and application, To make clear the difference between typical image content and the 3D voxel world, we call our factors *style* and *structure*, returning to terms used by [20]. We define style as describing the categorical block types used in a patch of Minecraft voxels. Minecraft has over one thousand such block types, all rendered using different colors, textures, and occasionally shape (with the exception of *air* which describes an empty voxel). Rather than encoding the exact block types used at each spatial location, we conceptualize style as a palette, or subset, of block types. For example, a "beach" style would describe and collection of *sand* and *water* blocks.

As in [20], we define structure as "the underlying geometry of the scene". In a given patch of voxels, structure describes which spatial locations have blocks present. We can obtain the structural factor of an area by setting all non-*air* blocks to be 1, and all *air* blocks to be 0. This results in a binary matrix that describes which blocks are "on" and which are "off". It is worth noting that this factorization diverges from most DRL research by discarding the factor independence assumption [2].

We note that structure and style (as we have defined them) have some causal relations. For example, the structure of a tree is closely tied to the block types *leaf* and *log*. So while our chosen factors are represent different semantic concepts, they are not statistically independent. As noted by [6], this independence condition is hard to satisfy in real-world datasets. While we may not learn a perfect disentangled representation of these two factors, this factorization allows for intuitive ideas of style transfer and structure transfer, which would be a beneficial feature for level designers, players, and developers.

Unlike other works that explore similar notions of style and concept disentanglement, we do not compress these generative factors into scalar values [35, 36]. We utilize a vector-wise representation structure [2], maintaining the spatial relationships of our data. This leverages the spatial inductive bias of convolutional networks, enabling granular and spatially-localized control in our latent representations.

[1]In Minecraft, the X-Z dimensions represent the horizontal plane, and Y represents vertical position

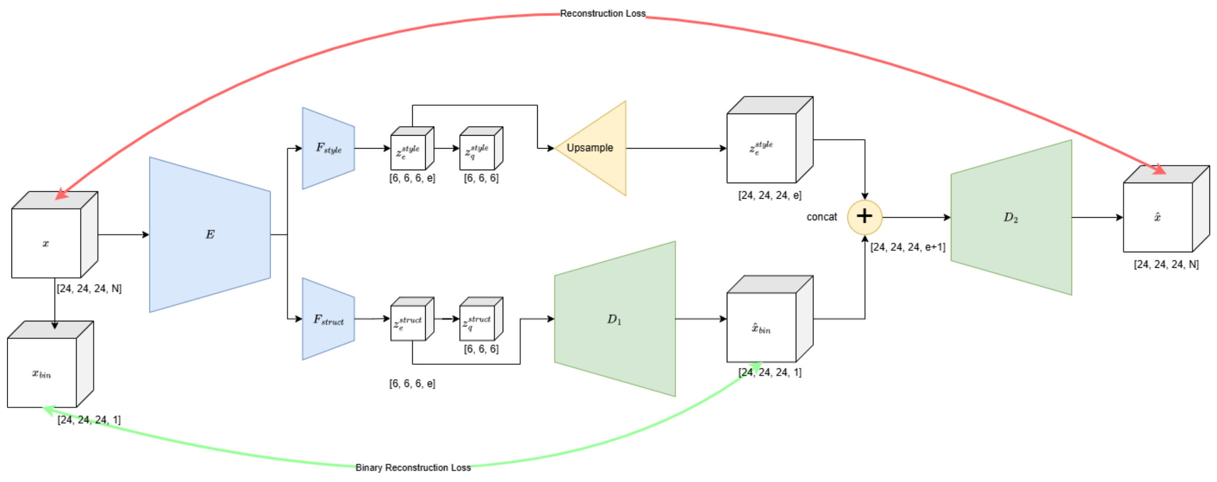


Figure 2: Model diagram

3.3. FQVAE

We design a novel Factorized Quantization Variational Autoencoder model to encode our dataset of terrain chunks into a discrete, disentangled representation. We start with a dual-codebook FQVAE, based on the implementation in [30]. Rather than using representation learning objectives based on frozen pretrained models, we encourage disengagement via additional loss objectives and architectural design.

3.3.1. Encoder Network

Our encoder network serves as a base feature extractor, learning general voxel features and outputting a sequence feature vectors of size c. This network resembles the encoder network of a standard VQVAE, using 3D convolutional residual blocks and gradually downsampling the spatial dimensions of our data. Our encoder has a downsampling factor of 4, with self-attention performed at the lowest resolution. This yields a $6 \times 6 \times 6 \times k$ feature representation. Following this feature-extractor base, we add two feature adapter heads, F_{style} and F_{struct} to translate base voxel features into style and structure features, as done in [30].

Each feature adapter head contains two more 3D convolution residual blocks, which transform the encoders features into codebook-specific feature vectors z_e^{struct} and z_e^{style}. Each encoded representation has dimensions $6 \times 6 \times 6 \times n$, where n is the codebook vector size.

3.3.2. Quantizer and Codebooks

We define two separate codebooks for style and structure, C_{style} and C_{struct} respectively. For quantization, we use exponential moving average updates (EMA) for both codebooks, which we find leads to much higher codebook utilization than updating via the original VQVAE loss function [37].

The size of our codebooks is significantly smaller than codebooks typically used in image VQVAE models, such as the 2886 codes used in VQDiffusion [38] or the 8192 codes used in DALL-E [39]. We limit our codebook sizes for three main reasons. First, as discussed by [30], two smaller codebooks provide combinatorially large “conceptual codebook” by considering unique combinations of codes. Thus, despite the difference in size, our representational capacity is not as low as it may appear. Second, we are motivated by findings in [25] that the majority of codes do not encode semantically similar concepts. We hypothesize that by restricting the number of available codes for each concept codebook, we can encourage semantically consistent representations by necessity.

Third, and most importantly, small codebook sizes are needed for human-facing interactive systems. The goal of this work is to enable human-in-the-loop generation and editing, using labeled codebook vectors. With large codebook sizes, it is infeasible for a human to parse and understand thousands of labeled codes. Further, large codebooks introduce user fatigue, as it becomes tedious to navigate through and select the appropriate code to induce a user's desired modification to the output. For our final model, we set $|C_{style}| = 16$ and $|C_{struct}| = 32$.

3.3.3. Decoder

Despite the impressive results achieved in by [30] using pretrained vision models to create a disentanglement objective, models that can serve a similar purpose are, to our knowledge, non-existent for 3D Minecraft voxel data. Without the representation learning objective, our model maintains the considerably weaker disentanglement objective $\mathscr{L}_{disentangle} = \frac{1}{N}\sum_{i,j}(z_q^{struct}{\top}z_q^{style})^2$. This pushes the involved style and structure codes towards orthogonality, but says nothing of the types of information encoded in each codebook.

To fill this gap, we create a two-stage decoder network. By reconstructing the structure and style information separately and sequentially, we encourage our model to encode meaningful semantic information into the respective codebooks. Our decoder can be thought of as two separate networks, D_1 and D_2, which function at different stages of reconstruction.

In the first stage, the D_1 takes as input only the structure codebook's latent representation z_q^{struct}. This stage attempts to reconstruct a binarized representation of input x, referred to as x_{bin}, which has an easily computable ground truth via the binarization process described in Section 3.2. To return to the original spatial dimensions of x, D_1 uses transposed 3D convolutional residual layers. $\hat{x}_{bin} = D_1(z_q^{struct})$ We use a binary reconstruction loss to train the first stage of our decoder network: $\mathscr{L}_{bin} = \mathscr{L}_{BCE}(x_{bin}, \hat{x}_{bin})$

This self-supervised binary reconstruction objective teaches our model to first construct the "scaffolding" of the reconstruction, providing blocks to the second stage to be "painted" with block styles.

Before entering the second stage, we up-sample our style codebook vectors z_q^{style} to match the spatial dimensions of $\hat{x}_{bin}$. We then channel-wise concatenate the up-sampled style codes and binary reconstruction. This concatenated tensor is passed as input to the second stage, where D_2 attempts to generate the fully reconstructed input $\hat{x}$ using further 3D convolutional layers.

$\hat{x} = D_2(concat(sg(\hat{x}_{bin}), upsample(z_q^{style})))$ We compute a categorical cross-entropy reconstruction loss between $\hat{x}$ and input terrain chunk x. We use a stop-gradient operation to prevent the gradient from the full reconstruction from flowing through the first stage of the decoder, which should only care learn the binary reconstruction task. $\mathscr{L}_{recon} = \mathscr{L}_{CCE}(x, \hat{x})$ This ensures that our style codes contain the additional information needed to paint our binarized reconstruction with the appropriate block types to achieve full reconstruction of x.

We omit the adversarial and perceptual loss terms of [30], finding the former detrimental to model performance, and the latter impossible due to lack of suitable models. This gives us a FQVAE, rather than an FQGAN. Our final training objective becomes: $\mathscr{L}_{FQVAE} = \lambda_1\mathscr{L}_{recon} + \lambda_2\mathscr{L}_{bin} + \lambda_3\mathscr{L}_{disentangle}$

3.4. Training

Our final model is trained for 15,000 steps on a single NVIDIA 3090, using a batch size of 8 and an Adam optimizer with $\beta_1 = 0.9, \beta_1 = 0.95, lr = 0.0001$.

4. Interpretability

One stated benefit of DRL is interpretability of learned representations. For our goal of controllable, human-interactive systems, this is a requirement: a designer must understand the semantic meaning of their selected code in order produce an expected change in the decoded output. To enable our model for human-in-the-loop interactive pipelines, we leverage this interpretability to label our latent codes in a way that a human can understand.

4.1. Labeling Codes

Both the encoder and decoder networks use self-attention layers, providing global context to each code. As a consequence, there is no single “meaning” of each code, as each realization in voxel space will be at least partially informed by other codes in the latent representation. Instead, we attempt to assign labels by computing informative metrics to the relevant generative factor over voxel-space representations. We compute these metrics with respect to the reconstructed output of our decoder $\hat{x}_{bin}$ and $\hat{x}$, as this is the representation we can control with the latent representations.

For our metrics, we make the simplifying assumption that each codebook vector is solely responsible for the corresponding 4x4x4 area in $\hat{x}_{bin}$, due to the spatial downsampling factor of our encoder. For example, $z_q^{struct}[0,0,0]$ has corresponding structural information in $\hat{x}_{bin}[0:4, 0:4, 0:4]$.

The labeled codebooks used for experiments are found in Tables 1 and 2 of the Appendix.

For editing and generation via latent intervention, a useful label for structural codebooks is a “blueprint”. This blueprint is a binary 4x4x4 array, which conveys the expected arrangement of non-air blocks in $\hat{x}$ when this code is used.

To compute this, we first construct a dictionary of size $|C_{struct}|$. For each structural code index, we collect all of the corresponding 4x4x4 chunks in $\hat{x}_{bin}$ across our entire encoded dataset. We visualize the top 5 most frequent patterns for each code, along with their frequency, to act as the label for each structure code (Table 2).

Labeling the style codes follows a similar process, constructing a dictionary style code indices and corresponding $4 \times 4 \times 4$ areas from the full reconstruction $\hat{x}$.

We compute block-type distribution across all chunks. Based on the block type distribution, we manually assign a text label that describes the expected block types for this style code. Where possible, we use labels corresponding to Minecraft’s biome system. For example, codes with a majority of water blocks are assigned the label “ocean”, while codes with a majority of sand blocks are assigned “desert”.

5. Experiments

We first explore whether the inductive bias of our model design succeeds in learning a disentangled representation. Fundamentally, we aim to learn a representation where style and structure are completely disentangled. Ideally, the effects of modifying the two latent representations are fully disjoint: editing the style codes causes a change in block types, but does not affect the topography, while changes to structure codes change the terrains shape but leave block types unchanged.

The lack of ground truth generative factors in our dataset prevents us from easily leveraging supervised disentanglement metrics. We instead use unsupervised information-based disentanglement metrics using mutual information to evaluate the modularity of our representations [2].

5.1. Mutual Information

Normalized Mutual Information (NMI) has been used in the field of DRL as a basis for both supervised and unsupervised measures of disentanglement [36, 35].

In the supervised setting, mutual information typically measures shared information between latents and source generative factors. Due to the lack of ground truth factors, we rely on a simpler (and perhaps less informative) formulation, computing the mutual information between our two codebooks

$$NMI(Z_{style}; Z_{struct}) = \frac{D_{KL}(p(Z^i_{style}, Z^j_{struct}||p(Z^i_{style})p(Z^j_{struct}))}{\frac{1}{2}(H(Z^i_{style}) + H(Z^j_{struct}))}$$

A normalized mutual information score of 0 indicates that the two codebooks are completely disentangled, while a 1 represents complete entanglement. As noted in Section 3.2, there are inherent relationships between block types and their common topographies that make eliminating mutual information impossible.

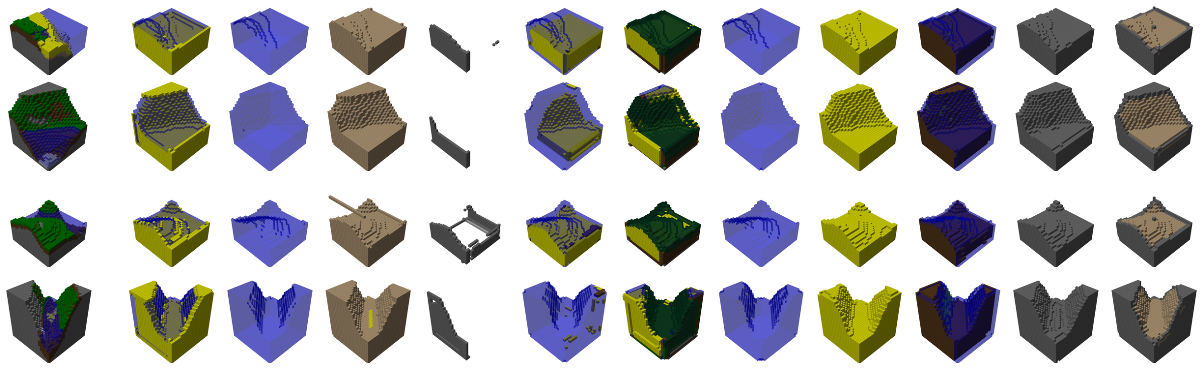


Figure 3: Left column: Original map reconstructions $\hat{x}$. Columns to the right show reconstructions where all values in z_q^{style} are replaced with a single style code (1, 2, 3, etc.). Style codes 0, 3, 10, 13, and 14 are ommited from the visualization due to similarity.

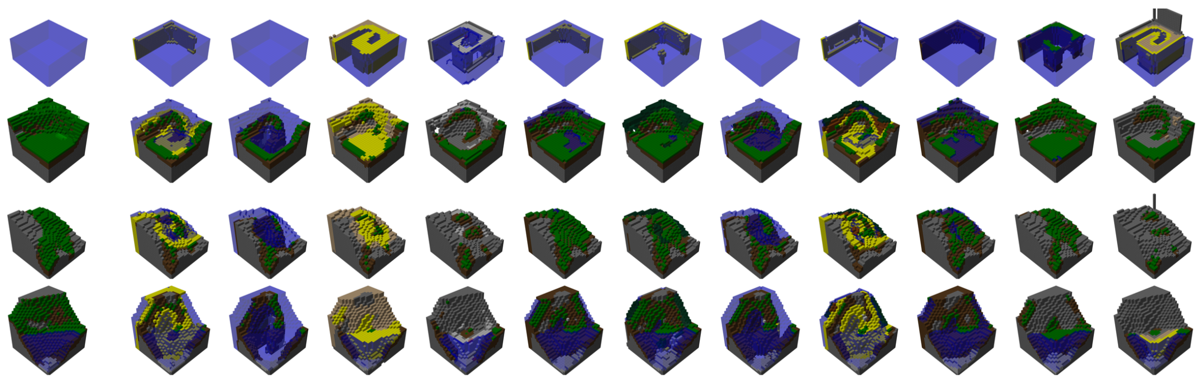


Figure 4: Left column: Original map reconstructions $\hat{x}$. Right: reconstructions where original style codes in z_q^{style} are replaced with a single code value in a spiral pattern

Our best model achieves a NMI of 0.1933. While this is low, indicating mostly disentangled features, it is not zero; indicating some persistent entanglement between the features. Exploratory analysis indicates this is largely due to a small set of highly entangled style and structure codes, often corresponding to causal relationships inherent to Minecraft. for example, structure codes for solid chunks of blocks have high mutual information with the "Stone" style, as the bottom half of our dataset largely consists of solid chunks of *stone* blocks.

5.2. Qualitative analysis

Figures, 3, and 4 show the result of "style swapping" on chunks of Minecraft terrain, demonstrating the disentanglement of style and structural information. By modifying the underlying style latent representation of existing maps, we are able to realize an intuitive style swap behavior in Minecraft voxels. We show that these latent representations are disjoint - modifications to style largely maintain the geometric information of the original map.

Figure 3 demonstrates an extreme case, using a single style code for the entirety of the map. We see predictable results that correspond to our labeled style codebook, with the "beach" style code generating *sand* blocks below *water*. While some style codes generate implausible results (like mountains made out of leaves), it demonstrates that our model has learned a generalizable way of applying style codes to unrelated voxel shapes, despite these homogeneous style encodings not appearing in the dataset.

Figure 4 demonstrates a more fine-grained application, closer to the detailed editing capabilities we wish to enable. We apply each style code in a spiral pattern, resulting in the map's original style codes appearing between unrelated style codes. Despite this, our model decodes these mixed latents in the desired way, even generating spirals of land in the ocean.

5.3. Human-in-the-loop experiments

We demonstrate how our learned representations enable human-in-the-loop interaction through a series of small-scale human experiments. Figure 1 illustrates the concept used in these experiments; by editing a map's latent representation, we can make large visual or structural changes to a map with just a few steps. By combining solid chunks (Structure code 12) and sloped chunks (Structure code 31) with the "grass" style code, we create a grassy mountain. Next, we stack tree-like structure codes (Structure code 19) with the "tree" style to create a tall tree. We extend this concept into "design tasks" and "generative tasks" to demonstrate the capabilities of this class of models.

5.3.1. Design Tasks

We showcase three tasks, designed to demonstrate the capabilities of the model for editing existing Minecraft terrain via both style and structural modifications. These tasks were selected to represent plausible edits to an existing map that a human designer (e.g. a developer or player) may wish to make to an existing piece of the Minecraft game world.

The first task "Dig a tunnel", represents a structure-based task. Figure 5 demonstrates how this can be accomplished using only structure latent edits, while Figure 6 extends this to create a secret mountain base by incorporating additional style edits.

The second design task,"Add a river" represents a style-based task. Figure 7 demonstrates how this task accomplished can be accomplished using only style-latent edits, while Figure 8, show how the task can be creatively extended using a combination of style and structure edits to create a flowing waterfall.

Our final design task is "Create a floating island". This task requires both style and structure edits, and also tests our model's generalization, as floating islands are typically rare in generated Minecraft worlds. Figure 9 shows two variations using the same starting map, where we edit the latent representation to create a floating platform of land with a tree on top. We show a grass and desert variant of the floating island.

5.3.2. Generative Task

While the design tasks showcase the ability for our model to edit existing Minecraft terrain (using its encoded representations), we seek to push this further and generate entire maps from scratch. To demonstrate this, we attempt to recreate existing terrain "blind" (i.e, not looking at the model's true latent representation of the map), using only our labeled style and structure codebooks and the visual features of the map as reference. We first select two maps from the dataset that contain interesting structure and style features, which serve as a ground truth for the experiment. Then, we attempt to recreate these maps using entirely human-authored latent representations. We allow for multiple refinement passes, to get as close to the target map in voxel space. Figures 10 and 11 show the results of this experiment.

In both cases, our manually constructed latents result in reconstructions that are close to the original sample. To check whether this is just the result of memorization, we compare our constructed latent to the map's ground truth representation. The first map uses 14 unique style codes and 24 unique structure codes, and the second terrain chunk uses 15 unique style codes and 28 unique structure codes. In comparison, our constructed latents use 4 unique structure and 5 unique style codes for the first map, and 4 unique structure and 3 unique style codes for the second.

6. Discussion

We proposed a Factorized Quantization VAE model for disentangling style and structure in 3D categorical voxel worlds. Using simple self-supervised reconstruction objectives, our model learns a suitable disentanglement of these concepts in in the discrete learned representation space. Despite discarding the typical assumption of statistical independence in the disentangled factors, we show that it is still possible to learn a meaningful disentanglement for downstream applications.

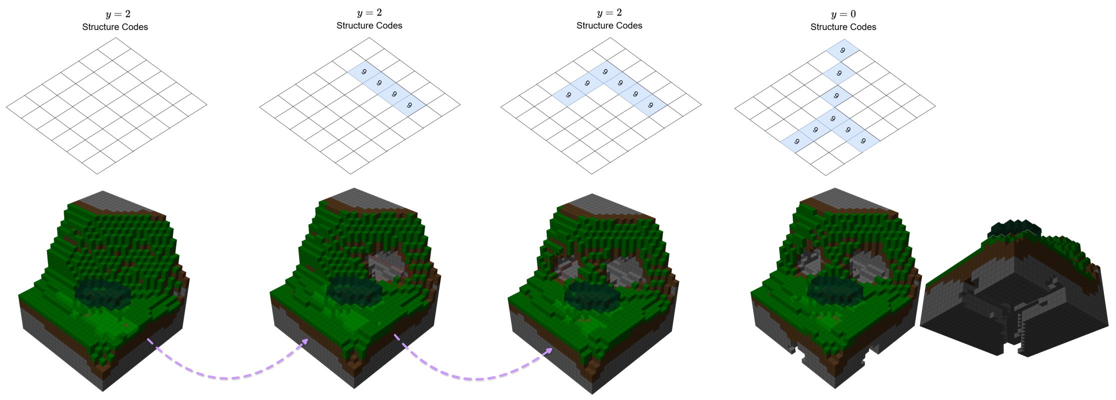


Figure 5: Design task 1: "Dig a tunnel". Top row: visualization of human latent editing steps. Bottom row: resulting reconstruction. White latent cells indicate the original z_q values, colored cells represented edited z_q values

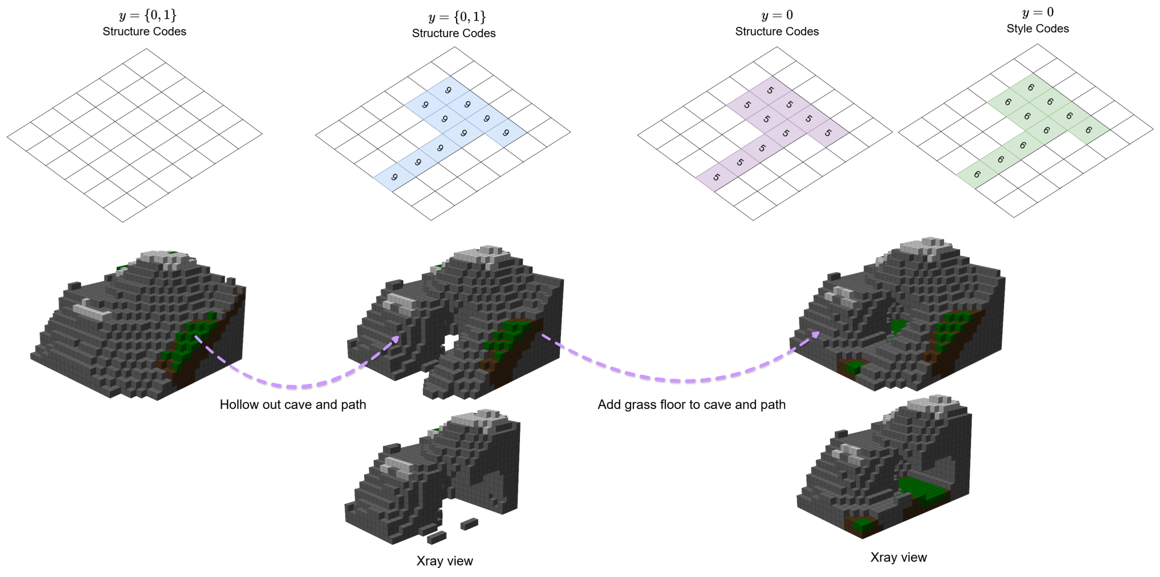


Figure 6: Design task 1: "Dig a tunnel". Top row: visualization of human latent editing steps. Center row: resulting reconstruction. Bottom row: "x-ray" view into the mountain.

Unlike most VQVAE models, we achieve style-structure disentanglement with a comparatively tiny codebook size. This enables semantic labeling approach to construct a small, human-interpretable codebook. Our assigned labels correspond to expected changes in the reconstructed output, allowing for direct human intervention in the representation space to modify encoded maps. We demonstrate that the labeled codebook we construct is interpretable enough to fully realize a creative vision by interacting solely with the latent space. By essentially "building in miniature", we are able to turn a much more compact representation into a full 24^3 chunk of Minecraft terrain, which can easily be spawned into game worlds to enhance the gameplay experience. This human-centered application of VQVAE models — generating content using humans to author the latent representation from scratch — is to our knowledge not explored in existing research.

7. Limitations

While we successfully demonstrate some use-cases, our small scale qualitative results do not constitute a robust exploration of our models generative ability. Early analysis of unsupervised DRL cast some doubt on the usefulness of disentanglement for downstream learning tasks [5]. While later work

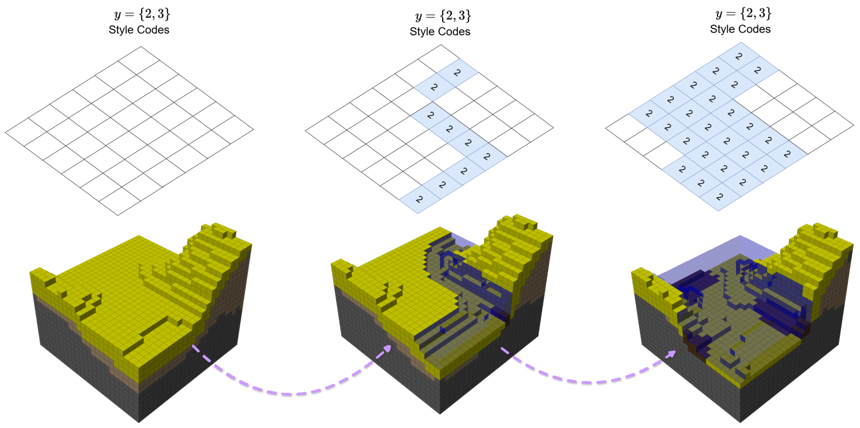


Figure 7: Design task 2: "Add a river". Top row: visualization of human latent editing steps. Bottom row: resulting reconstruction.

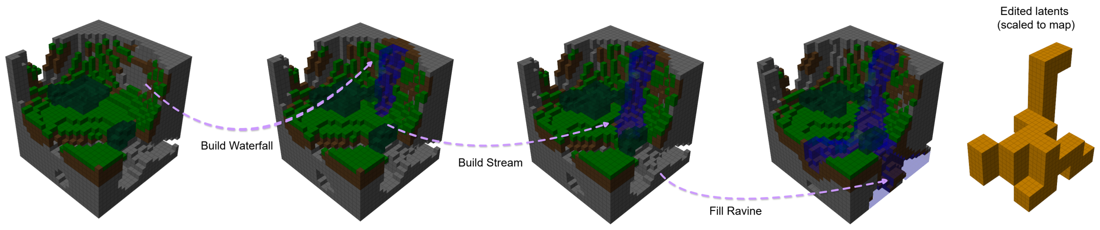


Figure 8: Design task 2: "Add a river". Visualized latent edits omitted due to complexity. Rightmost figure shows the map areas corresponding to edited latent codes.

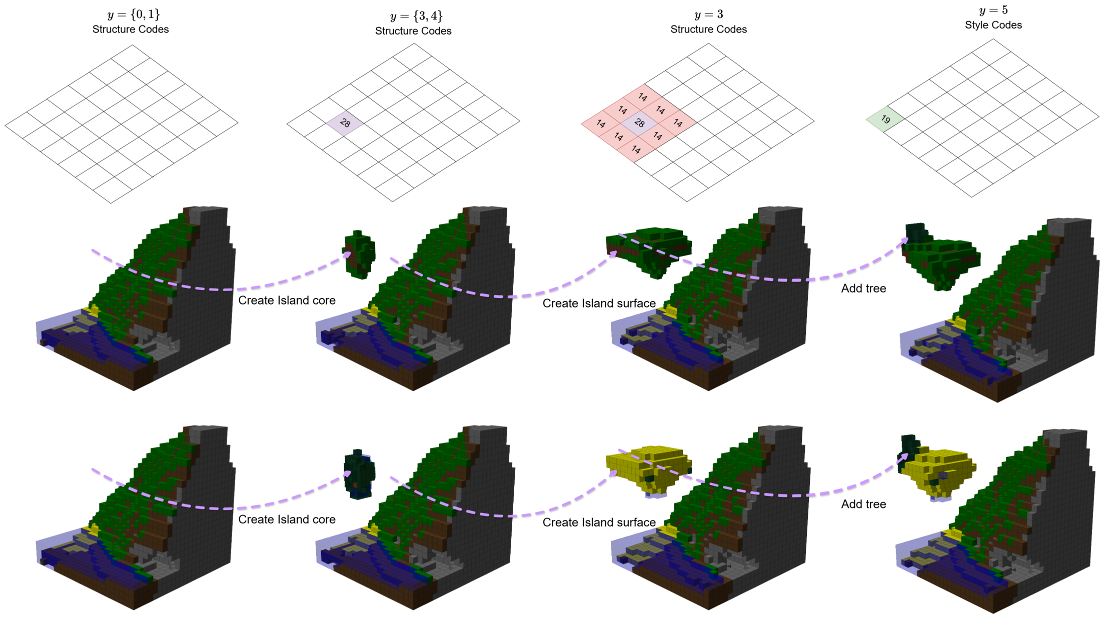
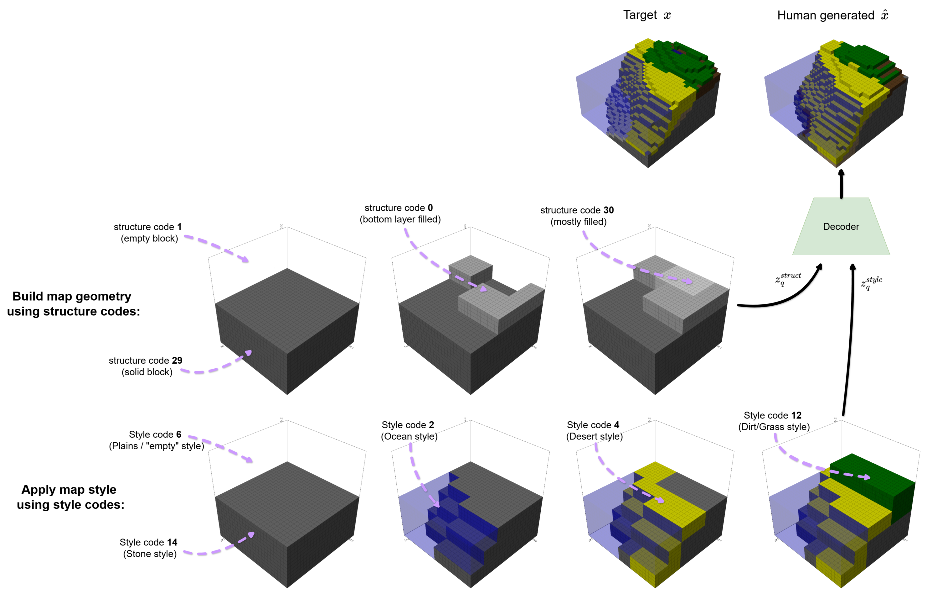


Figure 9: Design task 3: "Create a floating island". Top row: visualization of human latent editing steps. Bottom rows: resulting reconstructions. Center row shows the grassy version of the island, while the bottom row shows a desert variant.

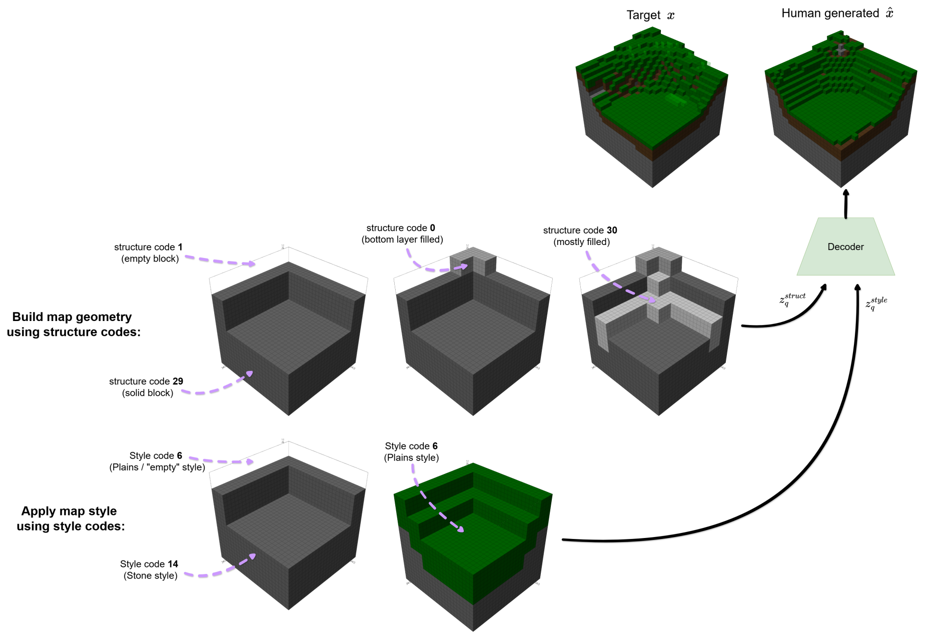
presents a more optimistic perspective of disentangled representations, specifically on the axes of explainability and generalization, we do not present any empirical results that prove our system's benefit for downstream generative tasks. A promising area of future work is to apply this disentangled representation to a generative Latent Diffusion Models.

Despite achieving good reconstruction quality, our model struggles with low frequency blocks,



Target x
Human generated $\hat{x}$
structure code **1** (empty block)
structure code **0** (bottom layer filled)
structure code **30** (mostly filled)
Decoder
z_q^{struct}
z_q^{style}
Build map geometry using structure codes:
structure code **29** (solid block)
Style code **6** (Plains / "empty" style)
Style code **2** (Ocean style)
Style code **4** (Desert style)
Style code **12** (Dirt/Grass style)
Apply map style using style codes:
Style code **14** (Stone style)

Figure 10: Generative task: recreate an existing map. Visualized is the series of edits to generate constructed latent representations z_q^{struct} (center row) and z_q^{style} (bottom row). We scale these representations to voxel space, with each latent code in the $6 \times 6 \times 6$ latent representation represented as a colored $4 \times 4 \times 4$ chunk of voxels. The top row shows the resulting reconstruction using our constructed latent representations (right), compared to the original target map (left)



Target x
Human generated $\hat{x}$
structure code **1** (empty block)
structure code **0** (bottom layer filled)
structure code **30** (mostly filled)
Decoder
z_q^{struct}
z_q^{style}
Build map geometry using structure codes:
structure code **29** (solid block)
Style code **6** (Plains / "empty" style)
Style code **6** (Plains style)
Apply map style using style codes:
Style code **14** (Stone style)

Figure 11: Generative task: recreate an existing map.

which often appear as decoration. Our dataset contains 42 unique block types in our dataset, but our style codes only broadly capture a small subset that most frequently appear in the game world. We experiment with weighting lower frequency block types, but this did not completely fix the issue. This may become more problematic when training on datasets with much higher categorical values, such as ones that capture the full set of Minecraft blocks. Existing work has explored Minecraft specific block embedding methods [12], which may provide solution to this issue, though we leave experimentation with embedding methods to future work.

References

[1] Y.-F. Wu, M. Lee, S. Ahn, Neural language of thought models, arXiv preprint arXiv:2402.01203 (2024).

[2] X. Wang, H. Chen, S. Tang, Z. Wu, W. Zhu, Disentangled representation learning, IEEE Transactions on Pattern Analysis and Machine Intelligence (2024).

[3] S. Edelman, Representation is representation of similarities, Behavioral and brain sciences 21 (1998) 449–467.

[4] B. Liu, Y. Zhu, X. Yang, A. Elgammal, Pivqgan: Posture and identity disentangled image-to-image translation via vector quantization (????).

[5] F. Locatello, S. Bauer, M. Lucic, G. Raetsch, S. Gelly, B. Schölkopf, O. Bachem, Challenging common assumptions in the unsupervised learning of disentangled representations, in: international conference on machine learning, PMLR, 2019, pp. 4114–4124.

[6] M.-A. Carbonneau, J. Zaidi, J. Boilard, G. Gagnon, Measuring disentanglement: A review of metrics, IEEE transactions on neural networks and learning systems 35 (2022) 8747–8761.

[7] Y. Bengio, A. Courville, P. Vincent, Representation learning: A review and new perspectives, IEEE transactions on pattern analysis and machine intelligence 35 (2013) 1798–1828.

[8] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, A. Anandkumar, Voyager: An open-ended embodied agent with large language models, 2023. URL: https://arxiv.org/abs/2305.16291. arXiv:2305.16291.

[9] W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. Veloso, R. Salakhutdinov, Minerl: a large-scale dataset of minecraft demonstrations, in: Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI'19, AAAI Press, 2019, p. 2442–2448.

[10] S. Earle, F. Kokkinos, Y. Nie, J. Togelius, R. Raileanu, Dreamcraft: Text-guided generation of functional 3d environments in minecraft, 2024. URL: https://arxiv.org/abs/2404.15538. arXiv:2404.15538.

[11] T. Merino, M. Charity, J. Togelius, Interactive latent variable evolution for the generation of minecraft structures, in: Proceedings of the 18th International Conference on the Foundations of Digital Games, FDG '23, Association for Computing Machinery, New York, NY, USA, 2023. URL: https://doi.org/10.1145/3582437.3587208. doi:10.1145/3582437.3587208.

[12] M. Awiszus, F. Schubert, B. Rosenhahn, World-gan: a generative model for minecraft worlds, 2021 IEEE Conference on Games (CoG) (2021) 1–8. URL: https://api.semanticscholar.org/CorpusID:235485259.

[13] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, M. Chen, Hierarchical text-conditional image generation with clip latents, 2022. URL: https://arxiv.org/abs/2204.06125. arXiv:2204.06125.

[14] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, B. Ommer, High-resolution image synthesis with latent diffusion models, 2022. URL: https://arxiv.org/abs/2112.10752. arXiv:2112.10752.

[15] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, P. Abbeel, Infogan: Interpretable representation learning by information maximizing generative adversarial nets, Advances in neural information processing systems 29 (2016).

[16] L. A. Gatys, A. S. Ecker, M. Bethge, A neural algorithm of artistic style, arXiv preprint arXiv:1508.06576 (2015).

[17] T. Karras, S. Laine, T. Aila, A style-based generator architecture for generative adversarial networks, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, pp. 4401–4410.

[18] H. Kazemi, S. M. Iranmanesh, N. Nasrabadi, Style and content disentanglement in generative adversarial networks, in: 2019 IEEE Winter Conference on Applications of Computer Vision (WACV), IEEE, 2019, pp. 848–856.

[19] D. Kotovenko, A. Sanakoyeu, S. Lang, B. Ommer, Content and style disentanglement for artistic style transfer, in: Proceedings of the IEEE/CVF international conference on computer vision, 2019, pp. 4422–4431.

[20] X. Wang, A. Gupta, Generative image modeling using style and structure adversarial networks,

in: European conference on computer vision, Springer, 2016, pp. 318–335.

[21] C. Trivedi, Turning fortnite into pubg with deep learning (cyclegan), Towards Data Science, available at: https://towardsdatascience. com/turning-fortnite-into-pubg-with-deep-learning-cyclegan-2f9d339dcdb0 (Jun. 18, 2018) (2018).

[22] C. A. Trivedi, K. Makantasis, A. Liapis, G. N. Yannakakis, Towards general game representations: Decomposing games pixels into content and style, ArXiv abs/2307.11141 (2023). URL: https://api.semanticscholar.org/CorpusID:260091544.

[23] T. Friedrich, B. Hammer, S. Menzel, Voxel-based three-dimensional neural style transfer, in: Advances in Computational Intelligence: 16th International Work-Conference on Artificial Neural Networks, IWANN 2021, Virtual Event, June 16–18, 2021, Proceedings, Part I 16, Springer, 2021, pp. 334–346.

[24] A. Van Den Oord, O. Vinyals, et al., Neural discrete representation learning, Advances in neural information processing systems 30 (2017).

[25] K. Eaton, J. Balloch, J. Kim, M. Riedl, The interpretability of codebooks in model-based reinforcement learning is limited, arXiv preprint arXiv:2407.19532 (2024).

[26] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, D. Batra, Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization, CoRR abs/1610.02391 (2016). URL: http://arxiv.org/abs/1610.02391. arXiv:1610.02391.

[27] P. Esser, R. Rombach, B. Ommer, Taming transformers for high-resolution image synthesis, 2021. URL: https://arxiv.org/abs/2012.09841. arXiv:2012.09841.

[28] S. Bond-Taylor, P. Hessey, H. Sasaki, T. P. Breckon, C. G. Willcocks, Unleashing transformers: Parallel token prediction with discrete absorbing diffusion for fast high-resolution image generation from vector-quantized codes, 2021. URL: https://arxiv.org/abs/2111.12701. arXiv:2111.12701.

[29] P. Sun, Y. Jiang, S. Chen, S. Zhang, B. Peng, P. Luo, Z. Yuan, Autoregressive model beats diffusion: Llama for scalable image generation, arXiv preprint arXiv:2406.06525 (2024).

[30] Z. Bai, J. Gao, Z. Gao, P. Wang, Z. Zhang, T. He, M. Z. Shou, Factorized visual tokenization and generation, arXiv preprint arXiv:2411.16681 (2024).

[31] A. Staaij, M. Preuss, C. Salge, Terrain-adaptive pcgml in minecraft, 2024, pp. 1–8. doi:10.1109/CoG60054.2024.10645652.

[32] T. Merino, M. Charity, J. Togelius, Interactive latent variable evolution for the generation of minecraft structures, in: Proceedings of the 18th International Conference on the Foundations of Digital Games, FDG '23, Association for Computing Machinery, New York, NY, USA, 2023. URL: https://doi.org/10.1145/3582437.3587208. doi:10.1145/3582437.3587208.

[33] C. Salge, M. C. Green, R. Canaan, J. Togelius, Generative design in minecraft (gdmc) settlement generation competition, in: Proceedings of the 13th International Conference on the Foundations of Digital Games, 2018, pp. 1–10.

[34] D. Grbic, R. B. Palm, E. Najarro, C. Glanois, S. Risi, Evocraft: A new challenge for open-endedness, in: Applications of Evolutionary Computation: 24th International Conference, EvoApplications 2021, Held as Part of EvoStar 2021, Virtual Event, April 7–9, 2021, Proceedings 24, Springer, 2021, pp. 325–340.

[35] G. Baykal, M. Kandemir, G. Unal, Disentanglement with factor quantized variational autoencoders, arXiv preprint arXiv:2409.14851 (2024).

[36] K. Hsu, W. Dorrell, J. Whittington, J. Wu, C. Finn, Disentanglement via latent quantization, Advances in Neural Information Processing Systems 36 (2023) 45463–45488.

[37] A. Van Den Oord, O. Vinyals, et al., Neural discrete representation learning, Advances in neural information processing systems 30 (2017).

[38] S. Gu, D. Chen, J. Bao, F. Wen, B. Zhang, D. Chen, L. Yuan, B. Guo, Vector quantized diffusion model for text-to-image synthesis. 2022 ieee, in: CVF Conference on Computer Vision and Pattern Recognition (CVPR), volume 2, 2021, p. 4.

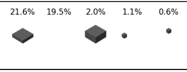
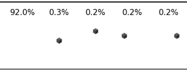
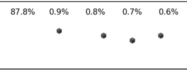
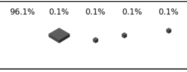
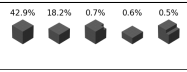
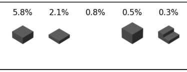
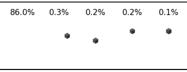
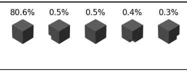
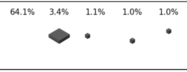
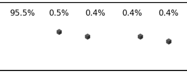
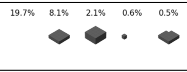
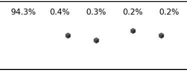
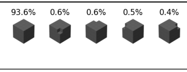
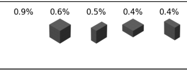
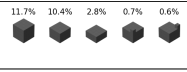
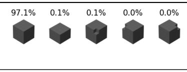
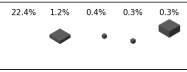
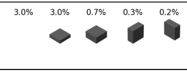
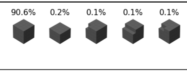
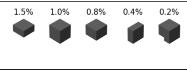
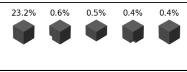
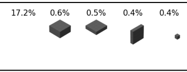
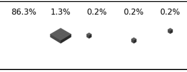
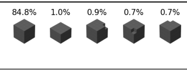
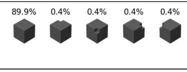
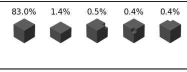
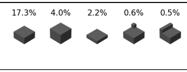
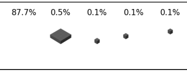
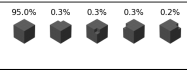
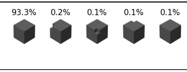
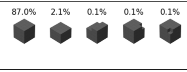
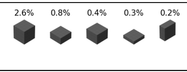
[39] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, I. Sutskever, Zero-shot text-to-image generation, 2021. URL: https://arxiv.org/abs/2102.12092. arXiv:2102.12092.

A. Appendix

Table 1
Labeled Style Codes

Code Index	Label	Most Frequent Blocks + Frequency (> 10%)
0	Stone 1	Stone (95.07%)
1	Beach	Sand (68.48%), Water (10.6%)
2	Ocean	Water (95.71%)
3	Stone 2	Stone (98.78%)
4	Desert	Sandstone (54.33%), Sand (32.81%)
5	Gravel Stone Water	Stone (54.68%), Gravel (22.77%), Water (14.81%)
6	Plains	Dirt (41.21%), Grass (36.38%), Sand (12.85%)
7	Tree	Leaves (53.49%), Stone (11.70%)
8	Pond	Water (65.75%), Dirt (18.32%),
9	Trees, and Sand	Leaves (23.62%), Water (18.70%), Log (18.44%), Sand (18.42%)
10	Stone 3	Stone (92.70%)
11	Dirt/Grass	Dirt (67.68%), Grass (13.60%), Stone (11.28%)
12	Stone/Dirt/Grass	Stone (43.79%), Dirt (35.31%), Grass (17.05%)
13	Stone/Dirt	Stone (57.07%), Dirt (35.66%)
14	Stone 4	Stone (95.07%)
15	Stone / Sandstone	Stone (64.24%), Sandstone (64.24%)

Table 2
Top 5 most frequent binary structure patterns for each structure code.

Index	Structure Blueprint
0	 21.6% 19.5% 2.0% 1.1% 0.6%
1	 92.0% 0.3% 0.2% 0.2% 0.2%
2	 87.8% 0.9% 0.8% 0.7% 0.6%
3	 96.1% 0.1% 0.1% 0.1% 0.1%
4	 42.9% 18.2% 0.7% 0.6% 0.5%
5	 5.8% 2.1% 0.8% 0.5% 0.3%
6	 86.0% 0.3% 0.2% 0.2% 0.1%
7	 80.6% 0.5% 0.5% 0.4% 0.3%
8	 64.1% 3.4% 1.1% 1.0% 1.0%
9	 95.5% 0.5% 0.4% 0.4% 0.4%
10	 19.7% 8.1% 2.1% 0.6% 0.5%
11	 94.3% 0.4% 0.3% 0.2% 0.2%
12	 93.6% 0.6% 0.6% 0.5% 0.4%
13	 0.9% 0.6% 0.5% 0.4% 0.4%
14	 11.7% 10.4% 2.8% 0.7% 0.6%
15	 97.1% 0.1% 0.1% 0.0% 0.0%
16	 22.4% 1.2% 0.4% 0.3% 0.3%
17	 3.0% 3.0% 0.7% 0.3% 0.2%
18	 90.6% 0.2% 0.1% 0.1% 0.1%
19	 1.5% 1.0% 0.8% 0.4% 0.2%
20	 23.2% 0.6% 0.5% 0.4% 0.4%
21	 17.2% 0.6% 0.5% 0.4% 0.4%
22	 86.3% 1.3% 0.2% 0.2% 0.2%
23	 84.8% 1.0% 0.9% 0.7% 0.7%
24	 89.9% 0.4% 0.4% 0.4% 0.4%
25	 83.0% 1.4% 0.5% 0.4% 0.4%
26	 17.3% 4.0% 2.2% 0.6% 0.5%
27	 87.7% 0.5% 0.1% 0.1% 0.1%
28	 95.0% 0.3% 0.3% 0.3% 0.2%
29	 93.3% 0.2% 0.1% 0.1% 0.1%
30	 87.0% 2.1% 0.1% 0.1% 0.1%
31	 2.6% 0.8% 0.4% 0.3% 0.2%

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.