CS230

Example-Guided Diffusion for Minecraft Texture Generation (Generative Modeling)

Markus Hoehn
Stanford University
`hoehn@stanford.edu`

1 Introduction

1.1 Description

Minecraft is a sandbox game whose world is composed of discrete "blocks," each rendered using one or more texture files that are conventionally square. A texture pack is a collection of these low-resolution images, typically 16×16 or 32×32 pixels, that together determine the game's overall visual style. Community-created packs often redraw hundreds of block textures by hand to impose a coherent aesthetic (e.g., Candy, Festive, Plastic). This manual process is time-consuming, labor-intensive, and requires maintaining stylistic consistency across a large set of textures. We propose an example-guided diffusion model that synthesizes new Minecraft textures conditioned on a set of reference examples. Given several textures from a target pack, the model infers its underlying artistic style and automatically generates additional textures that remain faithful to that style, enabling scalable creation of coherent texture packs. Our implementation is publicly available on GitHub.[1]

1.2 Dataset

Our dataset consists of 29 community-created Minecraft texture packs along with the official default textures. After filtering to retain only square textures corresponding to canonical Minecraft blocks, the final dataset contains roughly 27,000 images in total: approximately 1,000 default textures and 26,000 textures from community packs. Most packs provide 16×16 or 32×32 RGBA images, with one pack using 8×8 textures. Each community pack contributes on the order of 700–1,000 valid block textures. All community packs were sourced from the Modrinth repository [1] and span a broad stylistic range (e.g., Plastic, Halloween, Festive, Cartoon, Candy, Norse Mythology, Steampunk, Super Mario). We use all texture packs solely for research purposes and do not redistribute any original assets. To obtain stable RGB inputs for CLIP and the diffusion model, we composite each RGBA texture over a mid-gray background, though we also considered using a bright lime backing for later removal, or applying the default texture's alpha mask to the generated RGB output.

1.3 Evaluation

Qualitatively, we evaluate the model by generating textures for a variety of Minecraft blocks and inspecting their appearance in-game. We assess whether the outputs exhibit a coherent visual style, preserve Minecraft's discrete grid structure and edge sharpness, and look natural when rendered in the game environment. We also include a hold-out evaluation in which entire texture packs are excluded from training; given only a few example textures from a held-out pack, we compare the model's generated textures to the original textures from that pack. Quantitatively, we measure style consistency using CLIP-based metrics, evaluating whether the computed style embeddings of generated textures occupy similar regions in embedding space as those of their non-generated counterparts, and cluster with the reference pack.

[1]https://github.com/MarkOfUs/minecraft-texture-gen

2 Approach

2.1 CLIP Style Embeddings

CLIP maps images and text into a shared embedding space using paired encoders trained with contrastive language–image supervision [2]. We use the OpenCLIP implementation [3] of the `ViT-L-14-336-quickgelu` model, initialized with the weights released in [2], to compute all image and text embeddings.

To assess how CLIP interprets Minecraft textures, we perform a zero-shot retrieval experiment in which each texture is compared against a text set constructed from template variations of every block name in the dataset. As shown in Table 1, CLIP reliably captures coarse color and material cues, but fails to recognize Minecraft-specific objects, often returning semantically unrelated prompts. Accordingly, we use CLIP embeddings primarily as representations of stylistic variation, rather than object identity.

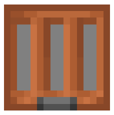
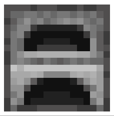
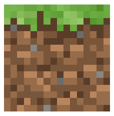
Texture	Image	Top-3 Predicted Prompts	Cosine
`acacia_trapdoor`		minecraft texture of oxidized copper trapdoor minecraft texture of copper trapdoor minecraft texture of oxidized copper door bottom	0.321 0.319 0.316
`furnace_front`		minecraft texture of stonecutter top minecraft texture of comparator minecraft texture of stonecutter bottom	0.297 0.295 0.294
`grass_block_side`		minecraft texture of soul soil minecraft texture of dirt minecraft texture of rooted dirt	0.308 0.307 0.301

Table 1: Zero-shot CLIP predictions for representative Minecraft block textures.

As we can see, the default CLIP model does not accurately identify what a Minecraft texture represents, but it does capture coarse color and stylistic information. To isolate style, we compute a style embedding for each texture x using L2-normalized CLIP image embeddings:

$$\Delta(x) \;=\; \frac{\mathrm{CLIP}(x)}{\|\mathrm{CLIP}(x)\|} \;-\; \frac{\mathrm{CLIP}(x_{\text{default}})}{\|\mathrm{CLIP}(x_{\text{default}})\|}.$$

where $\mathrm{CLIP}(\cdot)$ denotes the CLIP image encoder. Only the CLIP embeddings are normalized; the resulting style embeddings $\Delta(x)$ are left unnormalized because their magnitude reflects the strength of the stylistic deviation from the default texture. Taking the mean of these style embeddings across all textures in a pack, followed by a final normalization step, yields a compact pack-level style embedding.

To quantify which high-level styles CLIP associates with each pack, we compare every pack's normalized mean style embedding to a bank of text labels. Starting from a manually curated list of base terms covering franchises (super mario bros, wind waker, fallout, mass effect), seasonal themes (christmas, halloween), and generic style words (pastel, cartoon, blocky, rustic, steampunk), we automatically expand each term with suffixes such as "aesthetic," "vibe," "style," and "texture," yielding 310 candidate descriptors. We encode all labels with the CLIP text encoder, normalize them, and compute cosine similarity between each label and every pack's normalized mean style embedding.

Several packs exhibit strong alignment between their mean style embeddings and the expected stylistic descriptors. PastelCraft ranks pastel vibe, pastel style, and pastel aesthetic as its top–3 terms; the Candy Texture Pack similarly ranks candyland style and candy aesthetic in its top–2. Seasonal packs behave as expected: Festive Mash-Up surfaces christmas style, christmas vibe, and christmas aesthetic at the top, while Halloween Mash-Up ranks spooky, spooky style, and spooky texture in its top–3. Franchise packs also show clear matches: Super Mario Mash-Up ranks super mario bros style and nintendo vibe highest; Wind Waker Edition ranks wind waker aesthetic first;

Fallout Mash-Up ranks fallout style, fallout vibe, and fallout aesthetic in its top–3; and Mass Effect Mash-Up analogously ranks mass effect vibe, mass effect aesthetic, and mass effect style in its top–3. These results show that CLIP style embeddings consistently track high-level aesthetic themes, despite lacking Minecraft-specific semantics.

We also visualize the distribution of these style embeddings using UMAP [4], which provides a 2D projection illustrating how similar, or distinct, different packs are in CLIP style space.

2.2 Stable Diffusion and LoRA Adaptation

We experiment with two latent diffusion backbones: Stable Diffusion 1.5 (SD 1.5) [5] and Stable Diffusion XL (SDXL) [6]. Since neither model contains Minecraft-specific visual knowledge, unmodified SD 1.5 and SDXL perform poorly out of the box, producing blurry or semantically incorrect textures. We also tested a general pixel-art LoRA, PixelArtRedmond [7], but found that generic pixel-art priors do not transfer cleanly to Minecraft's highly structured, low-resolution textures.

To introduce Minecraft-specific structure, we trained LoRA adapters [8] for both SD 1.5 and SDXL. For SD 1.5, we trained a rank–32 LoRA on the full set of 1,020 default Minecraft block textures using (image, caption) pairs derived from normalized block names. For SDXL, we trained a rank–4 LoRA using the same dataset. All LoRA training was performed using the official Hugging Face diffusers implementation [9] and the provided training scripts examples/text_to_image/train_text_to_image_lora.py and examples/train_text_to_image_lora_sdxl.py from the Diffusers repository.

Across all models, the SDXL Minecraft LoRA produced the most stylistically coherent results, capturing sharp edges, grid structure, and block-appropriate material cues more reliably than SD 1.5. However, both models showed limited ability to fully acquire Minecraft semantics through LoRA alone—likely due to the low adaptation rank (often recommended for pixel-art domains, but insufficient here) and the substantial domain gap between natural-image training data and Minecraft textures. These limitations motivate further development of the diffusion backbone before integrating higher-level conditioning mechanisms such as IP-Adapter.

2.3 IP-Adapter Style Conditioning

Looking ahead, we plan to incorporate image-based style conditioning using IP-Adapter [10], which provides a general mechanism for injecting visual features into text-to-image diffusion models. Although IP-Adapter is not explicitly designed for style transfer, recent work such as InstantStyle [11] demonstrates that adapter-based conditioning can preserve content while guiding generation toward a target artistic style.

In our setting, the adapter is conditioned not on raw reference images, but on the CLIP-derived style embeddings introduced in Section 2.1. Given a set of example textures from a target pack, we compute a normalized mean style embedding:

$$s = \frac{\frac{1}{k}\sum_{i=1}^{k} \Delta(x_i)}{\left\| \frac{1}{k}\sum_{i=1}^{k} \Delta(x_i) \right\|}.$$

During inference, the diffusion model receives (i) the default Minecraft texture x_{def} for the block, which serves as the image input to the UNet and the IP-Adapter, and (ii) a text prompt containing the block name (e.g., "minecraft texture of spruce trapdoor"). The IP-Adapter additionally receives the pack-level style embedding s.

For training, the model is given the default texture x_{def}, the block-name text caption, and the target style embedding s. It is trained to reconstruct the corresponding target-pack texture $\hat{x}$ using the objective:

$$\mathcal{L} = \|\hat{x} - x_{\text{target}}\|_1 \; + \; \lambda \, \|\text{CLIP}(\hat{x}) - \text{CLIP}(x_{\text{target}})\|_2^2,$$

where λ is a scalar hyperparameter controlling the strength of the CLIP loss, and $\text{CLIP}(\cdot)$ denotes the CLIP image encoder. The first term encourages pixel-level fidelity, while the second enforces perceptual and stylistic alignment in CLIP space. This component will be integrated once the underlying diffusion–LoRA backbone is sufficiently stable.

References

[1] Modrinth. https://modrinth.com. Accessed: 2025-01-01.

[2] Alec Radford, Jong Wook Kim, Chris Hallacy, A. Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, 2021.

[3] Gabriel Ilharco, Mitchell Wortsman, Ross Wightman, Cade Gordon, Nicholas Carlini, Rohan Taori, Achal Dave, Vaishaal Shankar, Hongseok Namkoong, John Miller, Hannaneh Hajishirzi, Ali Farhadi, and Ludwig Schmidt. Openclip, July 2021. URL https://doi.org/10.5281/zenodo.5143773. If you use this software, please cite it as below.

[4] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2020. URL https://arxiv.org/abs/1802.03426.

[5] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695, June 2022.

[6] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis, 2023. URL https://arxiv.org/abs/2307.01952.

[7] ArtificialGuyBR. Pixelartredmond 1.5v pixel-art lora for stable diffusion. https://huggingface.co/artificialguybr/pixelartredmond-1-5v-pixel-art-loras-for-sd-1-5, 2023.

[8] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

[9] Patrick von Platen, Suraj Patil, Anton Lozhkov, Pedro Cuenca, Nathan Lambert, Kashif Rasul, Mishig Davaadorj, Dhruv Nair, Sayak Paul, William Berman, Yiyi Xu, Steven Liu, and Thomas Wolf. Diffusers: State-of-the-art diffusion models. https://github.com/huggingface/diffusers, 2022.

[10] Hu Ye, Jun Zhang, Sibo Liu, Xiao Han, and Wei Yang. Ip-adapter: Text compatible image prompt adapter for text-to-image diffusion models. 2023.

[11] Haofan Wang, Qixun Wang, Xu Bai, Zekui Qin, and Anthony Chen. Instantstyle: Free lunch towards style-preserving in text-to-image generation. *arXiv preprint arXiv:2404.02733*, 2024.