`word2sprite`: Dynamic Generation of Diverse Sprite Assets from Sentence Embeddings

Henry Mei hmei6@gatech.edu
Kelsi Blauvelt kblauvelt3@gatech.edu
Vincent Manna vmanna3@gatech.edu
Raajitha Middi rmiddi3@gatech.edu

Abstract

The five-dollar model by Merino et al. is a minimalist generative model that create human-interpretable, low-dimensional images from text. Experiments in video game pixel maps, character sprites, and emojis were run using small training data to surprising effect. This paper replicates Merino et al. using a more generalized video game sprite dataset inclusive of other non-character categories to explore the usefulness of a single model in generating a broad set of assets in a video game. In addition to evaluating data augmentation strategies and external model validation via pre-trained `CLIP VIT-B/32`, this paper investigates the relationship between alternative model architectures on categorical performance. We show that the addition of improvements to model architecture and training strategy can allow for similar results on a much broader representation space within sprites.

1. Introduction

Generative AI models have made significant advancements in recent years. Specifically text to image models are beneficial to multiple industries in the creative fields, such as advertising, design and gaming. The domain of procedural content generation enabled by machine learning allows for the rapid development of design and content. While many landmark advances in generative models emphasize performance and capability, they often come at the cost of simplicity, accessibility and content generation at the edge. Compute and network availability present restrictions to applying complex models. For example, mobile games need to have satisfactory performance across a wide range of devices and in periods when no internet is available (e.g., while someone is riding the metro).

2. Background

State of the art text to image models include DALL-E [7], Stable Diffusion [8], Midjourney, etc.; however, those models are significant in size and require online interfacing to extract output. Lightweight approaches like the five dollar model are more beneficial for generating low resolution content and require much fewer examples on which to train. Current thinking in procedural content generation include using LSTMs [9], reinforcement learning [4], and transformers [10]. These approaches however all require a large amount of training data. Small source data models that both scales resource-wise and delights the end-user are an open problem that has a diverse set of approaches, including Tree-Based Reconstructive Partitioning [3] and one-shot token based generation [1]. Many of these require more domain details of the game mechanics in development, and the five dollar model is one of few attempts at producing a minimal satisfactory example of rapid iteration in this space.

3. Motivation

This paper explores the current capabilities of a simplified approach over small datasets, by replicating and extending the paper "The Five-Dollar Model: Generating Game Maps and Sprites from Sentence Embeddings" [5]. Merino et al. demonstrate how combining a small language model MiniLM distilled from larger language models (i.e. BERT) via transfer learning with a simple convolution-based deep learning architecture can produce interpretable sprites. We replicate and extend the model from scratch in PyTorch and evaluate performance compared to the original research results. In particular, a more generalized sprite dataset is used to evaluate performance across different types of video game sprites to assess the viability of using a simple foundation model to create a broad set of game assets.

By demonstrating the success of small generative models, developers can create assets in support of procedurally generated games. As levels are dynamic, full personalization likely requires a level of robustness in items, characters, etc. that can only be achieved algorithmically.

4. Data

An open source Pixel Art dataset containing 89,000 pixel art images of various characters and objects from Kaggle is

Figure 1: Example images of the item category.

used to understand performance across different categories of sprites. [2] The images were originally obtained from an online game and were resampled to be 16x16 pixel 3-channel RGB JPEGs.

Table 1: Dataset manifest of files.

Filename	Description
`labels.csv`	Text labels for the images.
`sprites.npy`	`numpy` sprites pickle.
`sprites_labels.npy`	`numpy` labels pickle.

This dataset has been suggested by the author to be useful for various machine learning tasks such as image classification, object detection, and image generation and is licensed under Apache 2.0.

Of particular interest is the category labels:

Table 2: OHE label vector to human-readable category.

Label	Description
[0 0 0 0 1]	Players (side view)
[0 0 0 1 0]	NPCs (cats, blobs, etc.)
[0 0 1 0 0]	Environment Objects (fruits, etc.)
[0 1 0 0 0]	Inventory Items (sword, etc.)
[1 0 0 0 0]	Players (front view)

5. Approach

5.1. Data Preparation

300 samples were selected at random in a stratified random sample across the five label categories and hand-annotated with a sentence description. To reduce the feature space, the RGB images were quantized to 16 discrete colors via K-means in the pixel space. The trained model was saved to perform similar transformation for augmented samples and ensure a consistent treatment of images.

Sentences were transformed to 384-length embedding vectors via a pretrained `qa-MiniLM-L6-cos-v1` model. `qa-MiniLM-L6-cos-v1` is a popular, state-of-the-art sentence transformer model that produces normalized embeddings and can be scored via dot product metrics such as cosine similarity or euclidean distances. `qa-MiniLM-L6-cos-v1` is trained on 215M question/answer pairs from multiple sources and is a particularly good fit due to its contrastive learning objective. Sprite generation involves creating unique assets, and having meaningful subtlety and differentiation is important. [6]

A small length-5 noise vector is added to generate diversity.

5.2. Data Augmentation

5.2.1 Multiplicative Noise

Instead of additive noise of $\mathcal{N}(0, 0.15)$ which more often than not created single digit variances in the embedding vector, multiplicative noise of $\mathcal{N}(1, 0.01)$ was used to create greater diversity. The motivation for this is greater noise fuzzing should introduce better generalization and be a source of regularization.

Hadamard (elementwise) product is used to add noise across the entire embedding vector, and 3 noisy embeddings per ground truth observation are generated.

5.2.2 Mixup

Similar to SMOTE, Mixup is an augmentation technique that generates new training samples by linearly interpolating between pairs of existing samples and their labels to improve the generalization and robustness of neural networks. Data is used for regularization in this technique. [12] To create additional observations, embedding arithmetic assumptions were made and synthetic samples created by taking a $\lambda = 0.5$ soft update to effectively average across two images and their embeddings:

$$\mathbf{E} \leftarrow \lambda \cdot \mathbf{E}_{\mathbf{original}} + (1 - \lambda) \cdot \mathbf{E}_{\mathbf{sampled}} \quad (1)$$

$$\mathbf{X} \leftarrow \lambda \cdot \mathbf{X}_{\mathbf{original}} + (1 - \lambda) \cdot \mathbf{X}_{\mathbf{sampled}} \quad (2)$$

where $\mathbf{E}$ and $\mathbf{X}$ represent the embedding and image matrices, respectively. We generated one extra image-label pair.

5.2.3 GPT

An additional embedding is constructed for each ground truth sample by prompting GPT models to generate alternate labels that are then run through the sentence transformer to create a corresponding embedding vector. Initial implementation used `gpt-4o-mini` and was converted to `gpt-4o` for final production runs to minimize the cost of API requests. GPT models require tokenization as a pre-processing step, and both models fortunately share

`cl100k_im` byte pair encoding (BPE) via the `tiktoken` library.

BPE splits text into sub-word tokens that allow the consuming model to appropriately represent rare and unknown words by decomposing them to more common tokens. The tokenizer is trained on a large corpus of text to create a stable vocabulary representation. For alternate label generation, prompts and roles were specifically engineered to use common words.

The following prompt was used to encourage uniqueness in the labels that were generated:

> Each string in the list provided is a description of a pixel art image. For each string, write a corresponding alternate description that uses different words, sentence structure, and length. Use common words when writing the alternate labels and do not simply correct spelling and punctuation or use a different regional spelling (e.g., gray versus grey). Each alternate description should be different from the original string in a significant way. Your output list should have the same number of strings as the input list.

Similarly, a role was engineered to return an appropriate formatted list of labels to be safely parsed as a literal:

> You are a helpful assistant with excellent attention to detail. You only output python lists of strings according to the instructions you are given. Output the list on a single line, without any newlines. Make sure every list is closed properly. Each output string is significantly different than the input.

5.2.4 Embedding Interpolation

Again taking advantage of embedding arithmetic, synthetic embeddings were generated by taking n-step interpolations between the original embedding vector and the GPT-generate alternate embeddings. This contributed to an additional synthetic sample per original ground truth observation.

5.3. Five Dollar Model Architecture

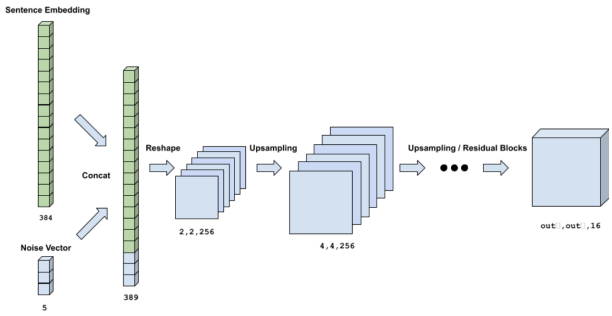


Figure 2: Five dollar model architecture.

The proposed five dollar model architecture was motivated by the need to be lightweight and train on small datasets (Fig. 2). The details of each layer are described in the following sections:

5.3.1 Embedding with Noise Input

The 384-length embedding vectors previously described, with the 5-length noise vector sampled from a uniform distribution for diversity, are concatenated as input to the model. The embedding vector enables the model to learn to translate sentence embeddings to low-resolution images. Each embedding is encoded based on the image caption (label).

5.3.2 Linear Layer and Upsampling

The input data is reshaped by a linear layer and then upsampled prior to the residual block. By upsampling, the feature vector will be the expected output size through the convolution layers of the residual blocks thus persisting the embedding patterns through training.

5.3.3 Residual Block

Residual blocks are stacked and composed of a sequential convolutional layer with kernel size 7 and padding of 3, followed by ReLu activation and then batch normalization, and repeated. In the original model, 5 residual blocks were used for training, with the first two performing upsampling (scaled by 2) prior to each block. The use of these residual blocks allow for skip connections, which aid in model training when evidence of vanishing/exploding gradients appears. The relationship of the total number of residual blocks with the depth and capacity of the model are further investigated as an experiment.

5.3.4 Output

Merino et al. used negative log-likelihood loss and the appropriate transformations to make it equivalent to cross-entropy loss. This paper uses cross-entropy loss on raw logits for brevity. In order to train on the new 16x16 shape data, additional upsampling after the residual blocks is performed to ensure the final convolution layer produces the expected output image size. Unfortunately, the standard PyTorch upsampling function negatively impacted training time, so upsampling was performed in a separate function outside of PyTorch in order to maintain better iteration speeds for experiments. The kernel of the final convolutional layer was modified from 9 to 17 in order to output the expected image size.

5.3.5 Optimizer & Scheduler

The original model and our work both use Adam optimizers; however, while Merino et al. used learning rate without decay, this paper implements a smooth and continuously decreasing learning rate to help stabilize the training process and avoid local minima and saddle points. An initial higher learning rate allows for exploration of a broader region of the loss landscape early in training. The learning rate then gradually decreases, allowing the model to make finer adjustments and exploit the regions that show promise. This cosine annealing strategy is combined with warm restarts to allow the model to explore new regions of the loss landscape and improving performance further in the sparse mapping of embeddings to images.

Mathematically, the learning rate at time t can be computed using the following formula:

$$\text{lr}(t) = \text{lr}_{\text{min}} + \frac{1}{2}(\text{lr}_{\text{max}} - \text{lr}_{\text{min}}) \left(1 + \cos\left(\frac{t}{T}\pi\right)\right)$$

where lr_{min} and lr_{max} are the minimum and maximum learning rates, respectively; t is the current iteration or epoch; and T is the total number of iterations or epochs for the cosine annealing period.

5.4. Challenges

We anticipated challenges with implementation extensions to the original paper model; however, we found ourselves starting at greater disadvantage than we expected.

5.4.1 Implementation

Previous work by Merino et al. was available publicly (via GitHub repository `TimMerino1710/five-dollar-model`). Unfortunately, the code in the repository was not usable in the reproduction of their paper. The original model was written in Tensorflow and required significant effort to rewrite in PyTorch. Key components of the paper such as data augmentation, CLIP score, and image generation from model output are missing.

5.4.2 GPT Lack of Label Diversity

Both `gpt-4o` and `gpt-4o-mini` models were unable to reliably create unique alternate labels. With bias towards short labels, the models oftentimes returned identical labels. Various prompt engineering experiments were done, but repeated trials were ultimately necessary maximize label diversity.

5.4.3 Data Leakage

Initial model efforts seemed too good to be true with validation curve mirroring training curve closely. This behavior is potentially explained by homogeneity in the training dataset. On closer observation, it was discovered that no differentiation between original and augmented samples was being made for the train and test datasets, allowing the model to evaluate itself using near-original augmentations such as multiplicative noise (Fig. 3).

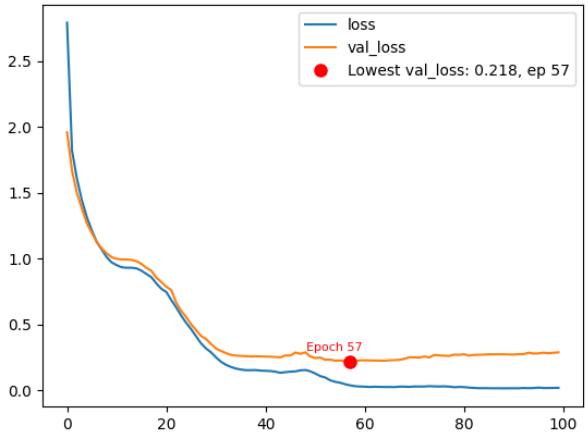


Figure 3: Epochs vs. loss curve that is too good to be true.

After employing a stratified train-test splitting strategy where a sample and all derivative augmentations are grouped together, the loss curves become much more... realistic.

6. Experiments & Results

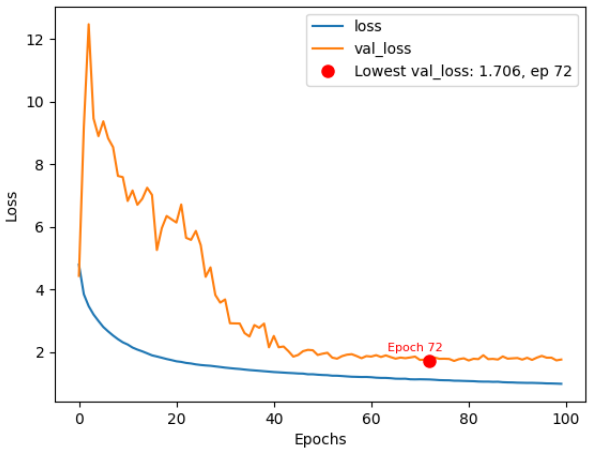
6.1. Hyperparameter Tuning

Table 3: Gridsearch results of hyperparameters tuned.

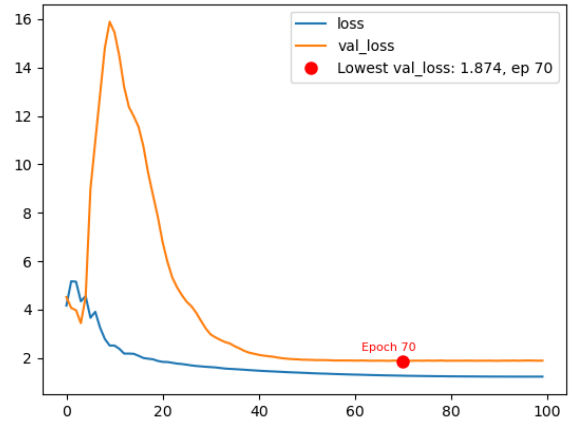
Hyperparam	Range	Best
learning rate	[1e-5, 1e-2]	1e-4
batch size	[64, 512]	128
residual blocks	[4, 11]	5
scheduler	assorted	`CosineAnnealing`

Gridsearch was performed on some key hyperparameters such as learning rate, batch size, and residual blocks in addition to a set of various schedulers (including having no learning rate decay or adjustment). Compared to the best hyperparameters of the original paper, we find that the broader set of sprite data benefited from a smaller learning rate (1e-4 vs 5e-4), smaller batch size (128 vs 256), larger number of residual blocks (5 vs 3), and scheduler

(`CosineAnnealingLR` vs None) (Table 3). This follows our intuition that adding significantly more variance in training data, which encourages more careful discovery with a smaller learning rate, dynamic convergence through a scheduler, regularization to tame overfitting through smaller batch size, and increased representation capacity through more residual blocks (Fig. 4).



(a) Sprite Data

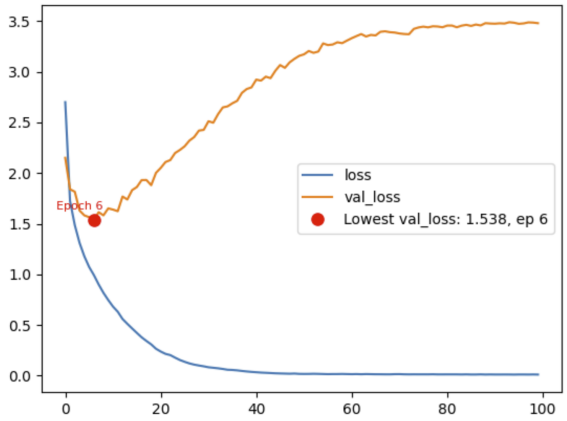


(b) Original Paper Dataset

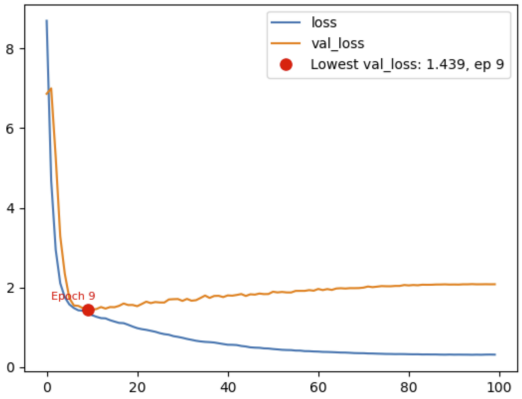
Figure 4: Epochs vs. loss curve for best model hyperparams using original paper sprite data and new sprite data.

6.2. Number of Residual Blocks

In order for the PixelArt data to be used to train the model, upsampling and kernel adjustment was made in the final convolutional layer of the model. Without upsampling the features prior to the final layer, training could only run with a kernel size of 1 in that configuration. This would basically simplify the convolutional layer as a fully connected layer, without the ability to extract features, but pass through the data one to one. When the model was trained with a kernel one configuration the training results were fairly poor (Fig. 5a), with the model validation loss high, unable to learn the new data. Which motivated the hypothesis that the final convolutional layer was important to maintain for learning. After including the upsample step, so the kernel could be larger, we had a loss curve that was trending down (Fig. 5b). However the loss seemed to hit a plateau.



(a) Convolution Layer with Kernel 1, Loss v Epochs



(b) Using Upsampling and Kernel 17, Loss v Epochs

Figure 5: Loss curve prior to upsampling with kernel=1 for the last layer and after updating the model to use a larger kernel on the final layer and upsample. Appeared to have an issue with the gradients potentially exploding.

Additional implementations to the modified model architecture discovered to be beneficial for the pixelart dataset. First parameters were initialized with Xavier initialization for more efficient training and to ensure equal variance across the initial weights in the layers. Using initialization the loss curves no longer started to be below the training loss for the first few epochs, since it would be expected that the validation loss be significantly higher (the model's early epoch predictions should be bad) at the beginning of training.

Table 4: Performance by number of residual blocks.

# of Blocks	Best Loss	Epoch
2	1.832	56
4	1.737	76
5	1.706	72
9	1.959	98
11	2.321	96

Second, Dropout was included in the residual blocks in order to force the model to better generalize to the data. Otherwise the model had exploding gradients when it reached a certain loss value where the validation loss actually jumped back up as previously seen in (Fig. 5b). Dropout will use a mask to select certain neurons to ignore from contributing to the following layer, this helps prevent over fitting, which is crucial when working with smaller datasets. And finally, after training over multiple values of residual blocks, (Table. 4), 5 residual blocks was found to be optimal.

Inference results from the best model captured some of the representative objects or characters expected, but not all. The model was consistent on expected color, but for some sprites the model still output blobs. Example outputs from the model can be seen in (Fig. 6)

Figure 6: Best Model results for PixelArt style objects and characters produced during inference on validation. Examples show 3 rows of, where each row the top pixelated image is the ”true” image and the bottom is the generated image by word2sprite.

6.3. CLIP & Subjective Scoring

Because the model is generating image output as opposed to classifying labels, a multi-modal `clip-vit-base-patch32` pre-trained model is used to generate embeddings for both the image and label in a sample pair. Cosine similarity can then be used as a way to quantify the correlation of the two as a form of showing how good the generative model is at faithfully following the label input. As Merino et al. discovered, CLIP scores are not particularly representative even in ground truth data. Thus, this paper uses CLIP scores to compute a normalized advantage: how much better are clip scores relative to random. Human ratings for binary accuracy was also performed.

Table 5: Performance by label class.

Label Class	Avg Loss	Adv.	Rating
Players (side)	2.1971	2.1583	41%
NPCs	1.9376	4.6124	57%
Environment Objects	1.7114	4.7091	61%
Inventory Items	1.1955	5.6285	87%
Players (front)	1.6348	4.0861	83%

Intuitively, the model struggled with side view perspectives, likely because perspective shift destroys details in low resolution imagery. Generally, the simpler the concept the more likely the model is to accurately generate the appropriate image. Inventory items, such as swords, books, keys, were created with roughly 87% human interpretation accuracy, which suggests the power of small models.

7. Potential Future Work

Future work might include text to image generation for images with more complex spatial data. Including training the model to understand prompt requests to localize certain areas of the image to place unique items in the learned categories. This would be a good opportunity to experiment with introducing something like self attention, as was done in work by Attention GAN [11]. Then given a certain scene imagery for a game, the designer could use the five dollar model to place more detail into the renderings.

8. Work Division

Work was dividing into 3 phases initially, data labelling, data augmentation, and model implementation. The three phases were then integrated to produce an end-to-end model, followed by developing visualization for how the model trained, debugging, experimentation, and paper writing. The team met weekly and more frequently closer to the due date and stayed in touch with work progress async via Discord.

Table 6: Contributions of team members.

Student Name	Contributed Aspects	Details
Henry Mei	Data Augmentation, Model Implementation, Hyperparameter Tuning, Experimentation, Writing	Implemented data augmentation/validation; contributed to implementation of PyTorch model, hyperparameter tuning, experiments, results analysis, and paper writing.
Kelsi Blauvelt	Model Implementation, Hyperparameter Tuning, Experimentation, Visualization, Writing	Contributed to implementation of the PyTorch model, hyperparameter tuning, experiments, results analysis, and paper writing; constructed visualizations.
Vincent Manna	Data Labeling, Hyperparameter Tuning	Co-labeled data with Raajitha; contributed to hyperparameter tuning.
Raajitha Middi	Data Labeling, Hyperparameter Tuning	Co-labeled data with Vincent; contributed to hyperparameter tuning and learning curve.

References

[1] Maren Awiszus, Frederik Schubert, and Bodo Rosenhahn. Toad-gan: Coherent style level generation from a single example, 2020. 1

[2] Ebrahim Elgazar. Pixel art dataset: 89,000 sprite images. http://www.kaggle.com/datasets/ebrahimelgazar/pixel-art, Feb 2024. 2

[3] Emily Halina and Matthew Guzdial. Tree-based reconstructive partitioning: A novel low-data level generation approach, 2023. 1

[4] Ahmed Khalifa, Philip Bontrager, Sam Earle, and Julian Togelius. Pcgrl: Procedural content generation via reinforcement learning, 2020. 1

[5] Timothy Merino, Roman Negri, Dipika Rajesh, M Charity, and Julian Togelius. The five-dollar model: Generating game maps and sprites from sentence embeddings, 2023. 1

[6] Hugging Face multi-qa-MiniLM-L6-cos v1. 2

[7] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation, 2021. 1

[8] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022. 1

[9] Adam Summerville and Michael Mateas. Super mario as a string: Platformer level generation via lstms, 2016. 1

[10] Graham Todd, Sam Earle, Muhammad Umair Nasir, Michael Cerny Green, and Julian Togelius. Level generation through large language models. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*, FDG 2023. ACM, Apr. 2023. 1

[11] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. Attngan: Fine-grained text to image generation with attentional generative adversarial networks, 2017. 6

[12] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization, 2018. 2