

Text-to-Level Diffusion Models with Various Text Encoders for Super Mario Bros

Jacob Schrum, Olivia Kilday, Emilio Salas, Bess Hagan, Reid Williams

Southwestern University
Georgetown, TX, USA
{schrum2,kilday1,salas2,haganb,williamsr}@southwestern.edu

Abstract

Recent research shows how diffusion models can unconditionally generate tile-based game levels, but use of diffusion models for text-to-level generation is underexplored. There are practical considerations for creating a usable model: caption/level pairs are needed, as is a text embedding model, and a way of generating entire playable levels, rather than individual scenes. We present strategies to automatically assign descriptive captions to an existing dataset, and train diffusion models using both pretrained text encoders and simple transformer models trained from scratch. Captions are automatically assigned to generated scenes so that the degree of overlap between input and output captions can be compared. We also assess the diversity and playability of the resulting level scenes. Results are compared with an unconditional diffusion model and a generative adversarial network, as well as the text-to-level approaches Five-Dollar Model and MarioGPT. Notably, the best diffusion model uses a simple transformer model for text embedding, and takes less time to train than diffusion models employing more complex text encoders, indicating that reliance on larger language models is not necessary. We also present a GUI allowing designers to construct long levels from model-generated scenes.

Introduction

Modern generative AI models use natural language input to create outputs in various modalities, including text, image, sound, and video. These technologies have also been applied to Procedural Content Generation (PCG), “the algorithmic creation of game content with limited or indirect user input” (Shaker, Togelius, and Nelson 2016). The use of generative AI classifies these methods as PCG via Machine Learning (PCGML) (Summerville et al. 2018).

Many models have been used to generate levels for Super Mario Bros. and other tile-based games, including Long Short-Term Memory networks (Summerville and Mateas 2016), Generative Adversarial Networks (GANs) (Volz et al. 2018), Variational Autoencoders (VAEs) (Thakkar et al. 2019), Large Language Models (LLMs) (Sudhakaran et al. 2023), the Five-Dollar Model (FDM) (Merino et al. 2023), and diffusion models (Lee and Simo-Serra 2023). LLMs and FDM are text-guided, whereas diffusion models can be

trained unconditionally or with text guidance. Although the use of text guidance in diffusion models is common in popular models like Stable Diffusion (Rombach et al. 2022), text-guidance seems underexplored in the realm of tile-based game level generation, which is the focus of this paper.

Though it is no surprise that diffusion models can be used for this purpose, there are still many practical considerations in training a working model, including procuring a dataset of adequately descriptive captions, selecting a text embedding model to pair with diffusion, and creating levels of the desired size with the finished model. These issues are explored in this paper. Specifically, our contributions are:

1. A method for automatically assigning captions to Mario level scenes that could be generalized to other domains given sufficient expert knowledge.
2. A method of assessing the quality of text-conditioned generation that depends on the ability to automatically assign captions to scenes.
3. A demonstration of how to use various types of text embedding models, both pretrained and trained from scratch, in a text-to-level diffusion pipeline.
4. A comparison of various text embedding approaches in terms of adherence to input prompts, training time, diversity, and playability, which ultimately concludes that a simple transformer model with a limited vocabulary results in the best diffusion models.
5. A mixed-initiative GUI that can be used to combine model-generated scenes into larger levels.

Related Work

Many generative PCGML models exist. Relevant work is split into unconditional models (no language input) and text conditional models (using natural language).

Unconditional Models

Early work in PCGML used models like Long Short-Term Memory networks (Summerville and Mateas 2016) to generate Mario levels. A survey of other early PCGML approaches came out in 2018 (Summerville et al. 2018)

That same year, Generative Adversarial Networks (GANs) (Goodfellow et al. 2014) for level generation were introduced (Volz et al. 2018). This work also applied latent

variable evolution (Bontrager et al. 2018) to find scenes with desired properties for Mario. Use of GANs for level generation was quickly expanded upon. GANs were combined with Graph Grammars (Gutierrez and Schrum 2020) and interactive evolution (Schrum et al. 2020) to generate Zelda levels. Compositional Pattern Producing Networks were used to combine GAN-generated level scenes into global patterns (Schrum, Volz, and Risi 2020; Schrum et al. 2023). Mixed Integer Programming was used to repair levels discovered by latent quality diversity evolution (Fontaine et al. 2021). The ability to generate samples from a single input was explored in both Mario (Awiszus, Schubert, and Rosenhahn 2020) and Minecraft (Awiszus, Schubert, and Rosenhahn 2021). Levels for multiple games were generated by a single GAN trained to induce a common latent space on data from multiple games (Kumaran, Mott, and Lester 2020).

The concept of searching a latent space to generate levels was also explored with Variational Auto-Encoders (VAEs) (Kingma and Welling 2014). The earliest application of this was to Lode Runner (Thakkar et al. 2019). Later work showed how to blend concepts across multiple games (Sarkar, Yang, and Cooper 2020), and used VAEs for latent quality diversity evolution (Sarkar and Cooper 2021).

Recently, diffusion models (Yang et al. 2023) have risen to prominence. They generate content via an iterative denoising process using a convolutional UNet. Diffusion models predict the presence of noise in a noisy image, so that said noise can be removed to produce a clean image. Trained models start with pure noise and derive quality output from it. A popular example is Stable Diffusion (Rombach et al. 2022), which adds text conditioning to the UNet and combines it with a VAE so that diffusion is performed in a compressed latent space rather than at the scale of the full image.

Stable Diffusion was the basis of research in the game Doom showing how diffusion models can function as semi-playable game engines (Valevski et al. 2024). The model was trained to predict the next screen frame conditioned on actions taken by a Reinforcement Learning agent (actions replaced text embeddings). A similar approach was used to simulate playing Super Mario Bros (Virtuals Protocol 2024).

Though impressive, these models try to reproduce the game experience rather than generate new content, but there are recent examples of generating levels with diffusion. Unconditional diffusion models can indeed generate convincing Mario level scenes when trained on scenes from the original game (Lee and Simo-Serra 2023). Dai et al. (2024) took individual Mario/Minecraft levels and used an unconditional diffusion model to generate new levels at different scales that share the distribution of elements from that one sample.

These methods produce playable levels, but lack control from text guidance. This is why evolution has often been combined with unconditional models to produce desired results, but defining a fitness function is generally more challenging than describing what is desired, so the next section describes PCGML approaches guided by text inputs.

Text-Conditional Models

Despite the frequent association of diffusion models with text guidance (e.g. Stable Diffusion), there is not much work

applying text conditioning to diffusion models for level generation. An exception is recent work on Text-to-game-Map (T2M) models trained as part of the Moonshine system (Nie et al. 2025), though the primary focus of Moonshine is the generation of synthetic captions by an LLM for the sake of training T2M models. The diffusion model from Moonshine relies on a model which we refer to as GTE below, as it is one approach to text embedding that we apply.

Text-to-Level approaches not based on diffusion also appear in the literature. Another model in the Moonshine paper is the Five-Dollar Model (FDM) (Merino et al. 2023), a feed-forward model whose name emphasizes its minimal computational requirements. Previous FDM results indicate it is useful despite its simplicity, but it struggles with overfitting and lack of diversity in outputs.

Level generation with variants of the Large Language Model (LLM) GPT2 from OpenAI (Radford et al. 2019) has been demonstrated in both Sokoban (Todd et al. 2023) and Super Mario Bros (Sudhakaran et al. 2023). We compare against this publicly available MarioGPT model below, though the complexity of the text prompts it understands is less ambitious than what our models are capable of.

Methods

We outline how training scenes are collected and combined with generated captions before training text embedding models, and then text-to-level diffusion models.

Training Data

Full levels are from Super Mario Bros. and the Japanese Super Mario Bros. 2, a.k.a. *The Lost Levels*, which was not initially released outside of Japan. This data comes from the Video Game Level Corpus (VGLC) (Summerville et al. 2016), a repository with data for several games. Despite being widely used (Volz et al. 2018; Sudhakaran et al. 2023; Lee and Simo-Serra 2023), the data has numerous errors and omissions, so we use our own manually cleaned version that is closer to data from the real games¹, and adds back some missing levels. However, we retain the limitation of representing enemies with a single symbol interpreted as a Goomba. We thus have 13 tile types.

As in previous works (Volz et al. 2018; Lee and Simo-Serra 2023), characters for each tile correspond to integers that are one-hot encoded. This approach has proven sufficient for us and others, though vector-based block/tile embeddings have also been used (Awiszus, Schubert, and Rosenhahn 2021; Dai et al. 2024). Such an approach could be useful, but is not explored in this paper.

To extract data, a window slides over each level one tile at a time. Mario levels are 14 tiles high, but because the architectural components of our diffusion model are easier to define when input sizes are powers of 2, we pad the tops of levels to create 16×16 samples.

Creating descriptive captions for each scene is more complicated. The Moonshine system (Nie et al. 2025) mentioned previously uses LLMs to create suitable captions for level data, but we use a deterministic approach. Concepts from

¹<https://github.com/schrum2/TheVGLC>

Mario scenes are manually defined, and scenes are scanned for the presence and quantity of these concepts, resulting in up to one phrase ending in a period for each concept. The full list of concepts and how they are defined is here:

- Floor: Blocks on the bottom row. Can have gaps, or be a void with small floor chunks.
- Ceiling: Blocks in fourth row filling at least half of the row. Can have gaps.
- Pipe: Four correctly arranged tiles of a pipe with neck tiles extended to a solid base or the bottom of the screen.
- Upside down pipe: Pipe with opening at the bottom and neck that extends to a solid top or the top of the screen.
- Coin Line: Adjacent coins in the same row.
- Coin: Coin tiles. Includes coins in lines.
- Cannon: Cannon tiles.
- Question Block: Both types of question block tiles.
- Enemy: Enemy tiles.
- Platform: Adjacent solid tiles in the same row, with the rows above and below being empty/passable.
- Tower: Collection of contiguous blocks with a width less than three and a height of at least three.
- Ascending Staircase: Solid tiles with empty space above where height increases by one for each move to the right. Sequence is at least three columns wide.
- Descending Staircase: Like ascending staircase, but height decreases by one while moving right.
- Rectangular Cluster: Flood-filled rectangular cluster of contiguous solid blocks. Flood fill excludes previously identified structures.
- Irregular Cluster: Remaining flood-filled clusters of at least three contiguous blocks not captured earlier.
- Loose Block: Solid blocks not captured earlier.

Most concepts include a quantity: “one”, “two”, “a few” (3-4), “several” (5-9), or “many” (10 or more). The floor concept distinguishes between “full floor” and one with some number of gaps. If over half of the floor is missing, it is a “giant gap” with some number of “chunks of floor”, though some levels have no floor. Similarly, a ceiling is either “full” or has some number of gaps.

This captioning style is the *regular* approach. However, we also consider *absence* captions, in which every concept missing from a scene is explicitly mentioned, as in “no floor.” The *absence* captions always have the same number of phrases, whereas the number in *regular* captions varies. Examples of each captioning approach are in Figure 1a. To encourage flexibility in using the models, the order of the phrases in training captions is randomized.

We can also assign captions to artificial scenes output by our models, which is useful for assessing model controllability later. When assigning captions to model output, two additional concepts are potentially present:

- Broken Pipes: Portions of a pipe that lack one of the four required tiles, or place them inappropriately.
- Broken Cannons: When a cannon support tile appears without a cannon tile on top.

These concepts are never present in training data.

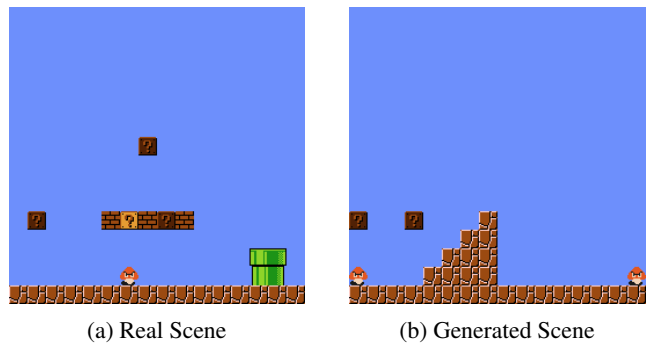


Figure 1: Real and Generated Level Scenes. (a) A scene from the data. Its captions are: *regular*: “full floor. one enemy. a few question blocks. one platform. one pipe.” *absence*: “full floor. no ceiling. one enemy. a few question blocks. no cannons. no coins. no coin lines. one platform. no ascending staircases. no descending staircases. one pipe. no upside down pipes. no towers. no rectangular block clusters. no irregular block clusters. no loose blocks.” *negative*: “ceiling. cannon. coin. coin line. ascending staircase. descending staircase. upside down pipe. tower. rectangular block cluster. irregular block cluster. loose block.” (b) Model-generated scene. Input prompt: “floor with one gap. a few platforms. a few enemies. a few coins. one coin line. a few towers. one ascending staircase. a few question blocks.” Actual caption: “full floor. two enemies. one ascending staircase. two question blocks.” Resulting caption adherence score: 0.478.

Text Embedding Models

The text-conditional models in Related Work depend on pretrained language models to embed text input for the level generator. Leveraging existing models allows for open-ended text input. However, game environments are constrained, so a large vocabulary is not necessary. Therefore, we train a simple model from scratch on limited vocabulary.

The simple architecture we use is a standard transformer encoder that learns token embeddings of length 128. Full details are in the appendix, but we allow multiple transformer encoder layers with multi-headed self-attention. During training, the model is given a sequence of token IDs, and encodes them with an embedding layer. These embeddings are combined with sinusoidal positional encodings and passed through the transformer layers where the attention mechanism enriches the embedded representations with surrounding context. For the sake of training, these embeddings are passed through a final linear layer that outputs logits for each token at each position. Masked Language Modeling is used during training, so we refer to this model as MLM. This means that some input tokens are probabilistically replaced with a special MASK token, but the model must predict the correct tokens from surrounding context. The result is token embeddings that capture semantic information about their context in sentences from the training data.

MLM is a small model trained on a small dataset with a small vocabulary, so it trains quickly and is effective at modeling the restricted grammar in our captions. How-

ever, MLM cannot tolerate tokens not present in its training data. In contrast, pretrained language models have been used in previous text-to-level systems, and *can* accept arbitrary tokens outside our limited vocabulary. The original FDM paper (Merino et al. 2023) used the sentence transformer `multi-qa-MiniLM-L6-cos-v1` (MiniLM). MiniLM maps whole sentences to vectors of length 384, and was designed for semantic search. Later research with Moonshine (Nie et al. 2025) combined FDM and a diffusion model with `gte-large-en-v1.5` (GTE) (Zhang et al. 2024). GTE embeds entire documents into vectors of length 1024. Although these models allow arbitrary tokens, such tokens have little meaning in Mario, and even familiar tokens are used differently in our captions than in natural language. Therefore, further fine-tuning of these sentence transformers could be useful, but this is left for future work.

An alternative approach that is explored takes advantage of the form of our captions: collections of period-separated phrases. The default approach embeds each caption as a single vector, but these phrases can be embedded individually to create multiple vectors to provide to the diffusion model. Both approaches are applied in our experiments.

Diffusion Model

The diffusion model is a conditional UNet with 13 in/out channels: one per tile type. It has three convolutional down-sampling stages and three up-sampling stages. Each contains residual blocks with SiLU activations and skip connections to preserve spatial information and avoid disappearing gradients, as well as cross-attention to allow text-embedding input from the language models described in the previous section. The 13 channels project to 128 channels, then 256, then 512 at the bottleneck before reversing the sequence. An unconditional model can be easily made by removing the cross-attention, and is done for the sake of comparison with an approach similar to that of Lee and Simo-Serra (2023).

We use classifier-free guidance, meaning the model is trained to handle both conditional and unconditional inputs, allowing stronger guidance at inference by interpolating between the two. Specifically, the text conditional model is trained on each sample using both text embeddings and an empty embedding vector. Effectively doubling the samples increases training time in comparison with an unconditional model. We can use `regular` or `absence` captions, but there is also a third option. Instead of indicating the absence of items in the caption, we can train with distinct negative prompts. Because all possible concepts are known, we can take each `regular` caption and make a separate negative prompt listing all concepts that are absent. Now for each sample, a third copy is paired with the negative prompt for negative guidance. This `negative` caption approach takes even longer to train since it triples the number of samples. An example scene with all captions is in Figure 1a.

The diffusion model is given noisy one-hot encoded level scenes as input, and tries to predict the noise that would need to be removed to get the original input. The loss function is a weighted sum of mean squared error (MSE) and categorical cross-entropy (reconstruction loss), as in Lee and Simo-Serra (2023). Detailed diffusion diagram in appendix.

Experiments

We train numerous models, and evaluate them with various metrics. All models were trained on different lab machines sharing the same hardware: Alienware, 13th Gen Intel® Core™ i9-13900F with 24 cores at base speed of 2.0GHz, 32 GB RAM, NVIDIA GeForce RTX 4060 with 8 GB dedicated VRAM and 15.9 GB shared VRAM. These are reasonably powerful gaming PCs. Detailed parameter settings and additional results are available in an appendix at <https://people.southwestern.edu/~schrum2/mario.html> and source code for recreating our results along with selected models are available at <https://github.com/schrum2/MarioDiffusion>.

Dataset Preparation

We use a 90/5/5-split of the 7,687 samples from the two Mario games to get training, validation, and test sets of sizes 6,918, 384, and 385. Care is taken to assure that all three datasets contain representation of all possible concepts. For training both text and diffusion models, data is augmented via random shuffling of phrases within captions. Validation data is used during training to determine the best model to keep, and test data is used for evaluation after training. We also make a set of 100 randomly generated captions not present in the original data for further testing. Each contains random phrases for 1 to 10 randomly selected topics.

Training Text Embedding Models

We train a separate MLM text encoder for each diffusion model that uses one. Models using `regular` and `absence` captions are trained with those caption types, though MLM models for `negative` captions also use `regular` data. Each model is trained for 300 epochs using AdamW with cross-entropy loss, but the lowest validation loss is logged every epoch, and the model with the best validation loss is kept as the final model.

Training Text-Conditioned Diffusion Models

For each text embedding approach and caption style, we train as many models as practical given compute costs. We train models with different random seeds for each caption style: 10 MLM, 10 MiniLM-`single`, 5 MiniLM-`multiple`, 5 GTE-`single`, and 1 GTE-`multiple`.

Models are trained with AdamW for 500 epochs using a cosine learning rate schedule and a warm-up period. To prevent overfitting, the caption adherence score defined later in Equation 1 is computed every 20 epochs across all validation captions, so the final model is whichever one had the best average c-score. The average c-score is a better measure of model performance than the usual validation loss.

Comparison Models

For comparison, we train several other models. We train 30 unconditional diffusion models for 500 epochs in a manner similar to Lee and Simo-Serra (2023). Since there are no captions, best model is determined by the lowest validation loss. We also train 30 Wasserstein GANs (WGANs) following the methodology of Volz et al. (2018), meaning we train for 5,000 epochs and the final model is from the final epoch.

Five-Dollar Models (FDMs) (Merino et al. 2023) are trained using MiniLM and GTE as embedding models with regular and absence captions (30 models per combination). As input, FDM takes a sentence embedding vector and a noise vector of length 5. As a text-conditioned model, FDM checks caption adherence score on validation data every 10 epochs to determine the best final model. However, our experience confirms the observation from previous work that FDM is prone to overfitting, so it is only trained for 100 epochs. Although the caption adherence scores for FDM show a general upward trend (with occasional dips), validation loss increases after an initial dip early in training.

We also compare against MarioGPT’s publicly available model (Sudhakaran et al. 2023), but do not train our own version. MarioGPT’s repertoire of training captions is comparatively limited, based on only 96 combinations (barring the use of arbitrary integer quantities). We use each caption to generate a level 128 blocks long and slice each into 8 scenes, from which 100 scenes are sampled.

Measuring Performance

Caption Adherence Score We focus on the ability of text-to-level models to produce scenes matching their input prompts. As indicated earlier, we can automatically assign captions to any level scene, including output from trained models. Output captions can be compared to their corresponding input prompts to define a caption score (c-score):

$$\text{c-score}(p, c) = \frac{\sum_{t \in T} \text{match}(\text{phrase}(t, p), \text{phrase}(t, c))}{|T|} \quad (1)$$

$$\text{match}(p_t, c_t) = \begin{cases} 1.0 & \text{for } p_t = c_t \\ 1.0 - \frac{|\text{qu}(p_t) - \text{qu}(c_t)|}{|Q| - 1} & \text{for } \text{co}(p_t) \wedge \text{co}(c_t) \\ 0.1 & \text{for } p_t \neq \emptyset \wedge c_t \neq \emptyset \\ -1.0 & \text{otherwise} \end{cases} \quad (2)$$

p is a prompt. c is a caption describing the scene produced from p . T is the set of caption concepts for Mario levels. $\text{phrase}(t, s)$ returns the phrase in s associated with concept t , or \emptyset if there is no such phrase or it starts with “no” (possible for absence captions). $\text{co}(s_t)$ is for *countable*, and indicates whether s_t describes a quantity (phrases like “full floor” do not). $\text{qu}(s_t)$ returns an integer that orders quantities: “one” = 0 up to “many” = 4. Q is the set of quantities.

So, $\text{match}(p_t, c_t)$ takes two phrases about the same concept, and returns 1.0 if they are identical (first case), a value from 0.0 to 1.0 if there is a partial match (second and third cases), and -1.0 if one phrase indicates the concept is completely absent, but the other does not. If both phrases indicate that some quantity of the concept is present, then smaller differences in quantities result in higher results. The third case returns 0.1 when one phrase has a quantity and the other does not, which only happens when comparing a full floor or ceiling to one with some amount of empty space.

The $\text{c-score}(p, c)$ calculation is simply the average $\text{match}(p_t, c_t)$ score across all topics, which means it is

scaled from the range -1.0 to 1.0. An example comparison and resulting c-score are in Figure 1b.

End Time and Best Time A model’s training time is also relevant. We define *End Time* as the time required to complete the final epoch, and *Best Time* as the time required to reach the checkpoint with the best validation performance. *Best Time* is relevant since early stopping could hopefully end training shortly after a model reaches its best performance. Note that times for MLM diffusion models include the additional training time for the actual MLM text encoders.

Average Minimum Edit Distance This metric measures the variety of levels within a set. Levels are compared in terms of edit distance. The average minimum edit distance $\text{AMED}_{\text{self}}$ across a set of level scenes S is

$$\text{AMED}_{\text{self}}(S) = \frac{\sum_{s \in S} \text{argmin}_{x \in (S - \{s\})} \text{dist}(s, x)}{|S|} \quad (3)$$

A high $\text{AMED}_{\text{self}}$ score means most levels are very different from each other. A low score means most levels are similar to other levels in the set. As a set of limited possibilities grows, closer neighbors are more likely to be found, so comparison of sets with equal sizes is important for fairness.

Average minimum edit distance can also be defined with respect to real game data to define $\text{AMED}_{\text{real}}$:

$$\text{AMED}_{\text{real}}(S) = \frac{\sum_{s \in S} \text{argmin}_{x \in R} \text{dist}(s, x)}{|S|} \quad (4)$$

R is the set of real game samples. High $\text{AMED}_{\text{real}}$ scores indicate large differences between generated and real levels.

Solvability A level scene is considered solvable if Robin Baumgarten’s A* agent (Togelius, Karakovskiy, and Baumgarten 2010) can beat it, though this widely used agent is not perfect (Šosvald et al. 2021), and we have observed unusual failure cases. Furthermore, it is not always the agent’s fault that a scene cannot be beaten. Although complete levels are beatable, slicing them into 16×16 samples sometimes results in the loss of a platform or other element that is required to traverse the remainder of the scene. Only 7,160 of the 7,687 samples are solvable, approximately 93%.

Level Integrity In model output, tiles for pipes and cannons are sometimes arranged incorrectly. The broken pipe issue was first recognized when generating Mario levels with GANs (Volz et al. 2018). There are trivial ways to repair broken features or change the data encoding so that they cannot appear (Schrum, Volz, and Risi 2020), but forcing the models to learn how to build such structures provides us another way to assess them. Therefore, the percentage of generated scenes with broken pipes and cannons is reported.

Results

Caption Adherence Score Caption adherence across the test set prompts from real game scenes is in Figure 2. Most models earn scores above 0.9 within 500 epochs.

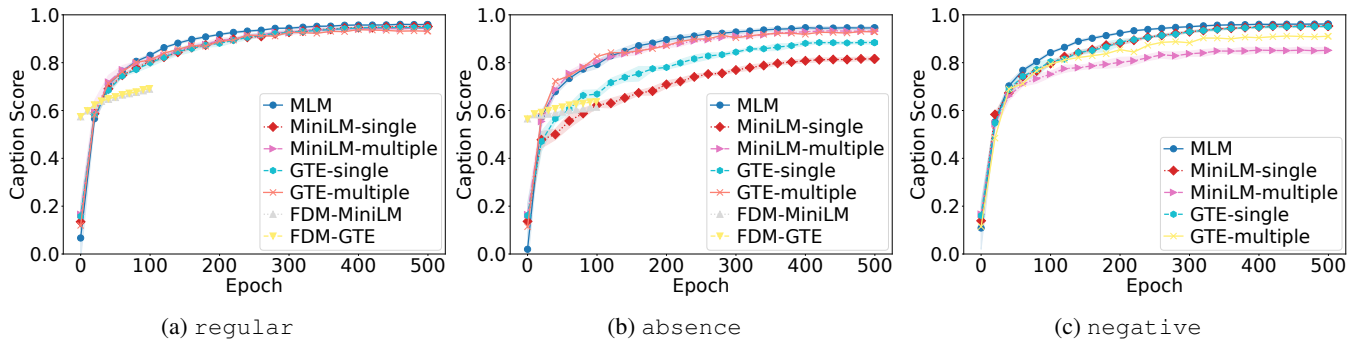


Figure 2: Average Caption Adherence Score by Epoch on Real Game Test Set Captions. The number of runs being averaged over varies by model type, though GTE-multiple data are based on a single run each. When there is more than one run, 95% confidence intervals are shown, though they are often extremely narrow. Recall that FDM models are trained for a shorter period due to their overfitting problem. (a) regular models are all good, though MLM peaks faster. FDM starts high but lags behind. (b) absence results are more varied, with GTE-single and MiniLM-single performing poorly, and MLM once again on top. FDM results are flatter. (c) negative results are also varied, with MLM once again on top and poor performance from MiniLM-multiple and GTE-multiple. FDM does not train with negative captions.

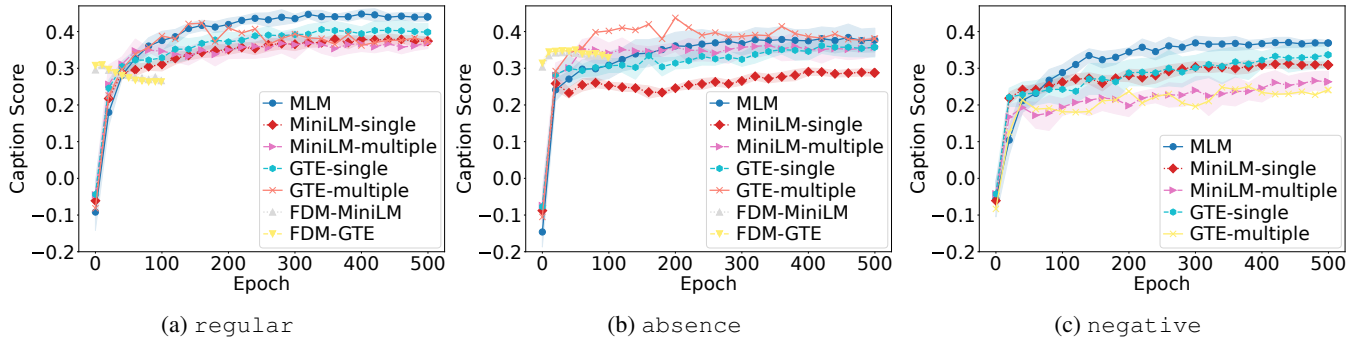


Figure 3: Average Caption Adherence Score by Epoch on Random Captions. The 95% confidence intervals are more evident here. (a) regular models have the highest scores. MLM is the best among them. FDM performance plummets quickly and GTE-multiple is erratic. (b) absence data mostly has worse scores, though GTE-multiple has a spike before dipping to meet the other models. Only MiniLM-single is exceptionally low. FDM models maintain performance better in this case. (c) negative performance is the worst, though MLM is the best in this group. FDM does not train with negative captions.

Among text-conditioned diffusion models, the only exceptions are MiniLM-single-absence, GTE-single-absence, MiniLM-multiple-negative, and GTE-multiple-negative. FDM models start high, but get stuck around 0.6 to 0.7 depending on the specific model.

Results across all real data are qualitatively similar (appendix). In contrast, models have difficulty with random captions (Figure 3). The highest score is under 0.5, and is achieved by MLM-regular. Figure 1b shows an example scene generated by MLM-regular. Early in training, GTE-multiple-regular and GTE-multiple-absence reach the maximum score achieved by MLM-regular before dropping down. Overfitting by FDM is evident, as performance peaks and then drops within about 30 epochs. The worst scores are from MiniLM-single-absence, MiniLM-multiple-negative, and GTE-multiple-negative. The way some scores drop indicate that random captions may have served better to determine the best final model than validation captions.

End Time and Best Time Figure 4a shows average *End Time* and *Best Time* for each model. The text-conditioned diffusion models with the shortest training times are MiniLM-single-regular, MiniLM-single-absence, MLM-regular, and MLM-absence with times from 12.58 to 14.1 hours. However, MiniLM’s c-scores on random captions were worse than MLM, so the small time difference is not worth the performance drop.

In terms of *Best Time*, the same models are fastest, but in different order: MiniLM-single-regular, MLM-regular, MLM-absence, and MiniLM-single-absence. Times range from 11.44 to 12.51 hours. For these models, the best epoch came slightly before the final epoch. GTE-multiple-negative is the only model where this difference was huge: 126.3 vs. 73.97 hours.

In general, negative captions take more time for little gain. Although some GTE models were comparable to MLM in terms of caption score, they take longer to train, even in terms of *Best Time*. Pretrained sentence transformers that

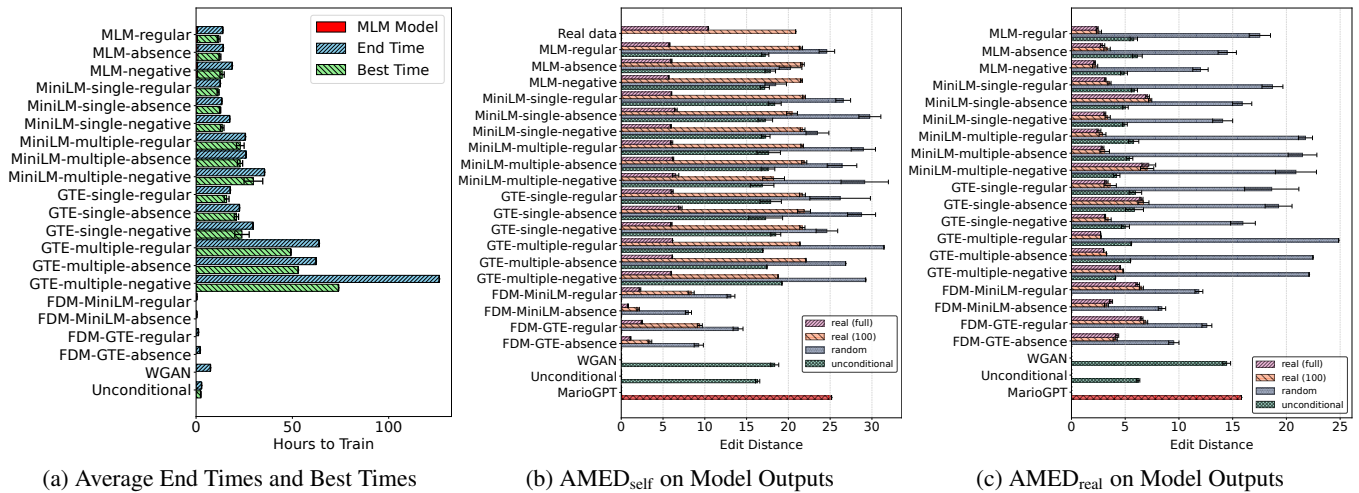


Figure 4: Bars have 95% confidence intervals when multiple models were trained. (a) Time to train the text encoder for MLM diffusion models is shown in red on the left, but is barely visible. *Best Time* is not shown for WGAN or FDM. (b) $AMED_{self}$ compares the diversity of model-generated scenes. *real (full)* is generated from all real data prompts. *real (100)* is from a selection of 100 real prompts. *random* prompts are distinct from real prompts. *unconditional* scenes had no prompts. Diversity for Real data is also shown, and data from MarioGPT. WGAN and Unconditional cannot produce scenes from prompts. FDM has no *unconditional* scenes. (c) $AMED_{real}$ compares model-generated scenes in terms of their distinctness from the real data.

use multiple phrase embeddings take longer to train than their single embedding counterparts.

The models that train the quickest are unconditional diffusion, WGAN, and FDM. However, WGAN and unconditional diffusion offer no text guidance, and FDM performance is much worse, so the extra speed is of little benefit.

Average Minimum Edit Distance To compare the diversity of generated levels, four sets of scenes are created by most models: scenes from the full set of real game scene captions, 100 samples from this set, scenes from random captions not in the original data, and unconditionally generated scenes. Except for *real (full)* data, these sets each contain 100 scenes to allow fair comparison. WGAN and unconditional diffusion cannot generate scenes from captions, so they only have unconditional samples. FDM can technically create unconditional samples from an empty embedding vector, but it was not intended to, and the results are so terrible that we do not include them. MarioGPT is a special case, since its scenes were not generated unconditionally, but also were not generated by our captions. We compare against 100 size 16×16 scenes from MarioGPT as described earlier.

Figure 4b shows $AMED_{self}$ scores. In general, *real (full)* is small because comparing against more scenes makes it more likely to find a similar scene. This is why *real (100)* was needed for fairness, and is always higher. Across text-conditioned diffusion models, *real (full)* is around 6 tiles, and *real (100)* is around 22 tiles, except for GTE-multiple-negative and MiniLM-multiple-negative around 19 tiles. These two models also had poor caption adherence scores. For reference, *Real data* shows that $AMED_{self}$ is 10.4077 tiles on the full set of game data and 20.87 tiles in the 100 sample case. The collection of all real game data shows more diversity than what models

produce using all real game captions, but the diversity across the evenly spaced set of 100 samples is about the same.

For *random* captions, diversity varies more but is generally higher than for real captions, though there are exceptions: MLM-absence and MLM-negative. Some models with lower caption adherence have higher diversity from random captions; poor level structure can result in high edit distances, since tiles will be in weird places. Unconditional samples are generally less diverse than caption-generated samples, the only exception being GTE-multiple-negative. WGAN and unconditional diffusion have comparable scores to the unconditional samples from text-conditioned diffusion models. FDM’s $AMED_{self}$ scores are extremely low in all categories. MarioGPT’s score is around 25, which is higher than *real (100)* for all models but lower than the *random* score of most.

Figure 4c compares $AMED_{real}$ results. Samples from random captions have high distances, but distances from real captions are much smaller, with unconditional samples usually between. *real (full)* and *real (100)* are generally closer to each other, around 3-5 tiles, though some FDM results are higher. In other words, for captions seen during training, models often create nearly identical scenes, which is probably for the best, but it means that alternate scenes that share the caption are not generated. This is a limitation in how well models generalize. Thankfully, high *random* scores indicate that generation of novel scenes is possible, though being too novel carries the risk of being disorganized, as occurs with WGAN results. WGAN samples have higher $AMED_{real}$ scores because they struggle to fit the data. In contrast, samples from unconditional diffusion have $AMED_{real}$ scores similar to unconditional samples from text-conditioned diffusion models. FDM is once again an anomaly, since its *ram-*

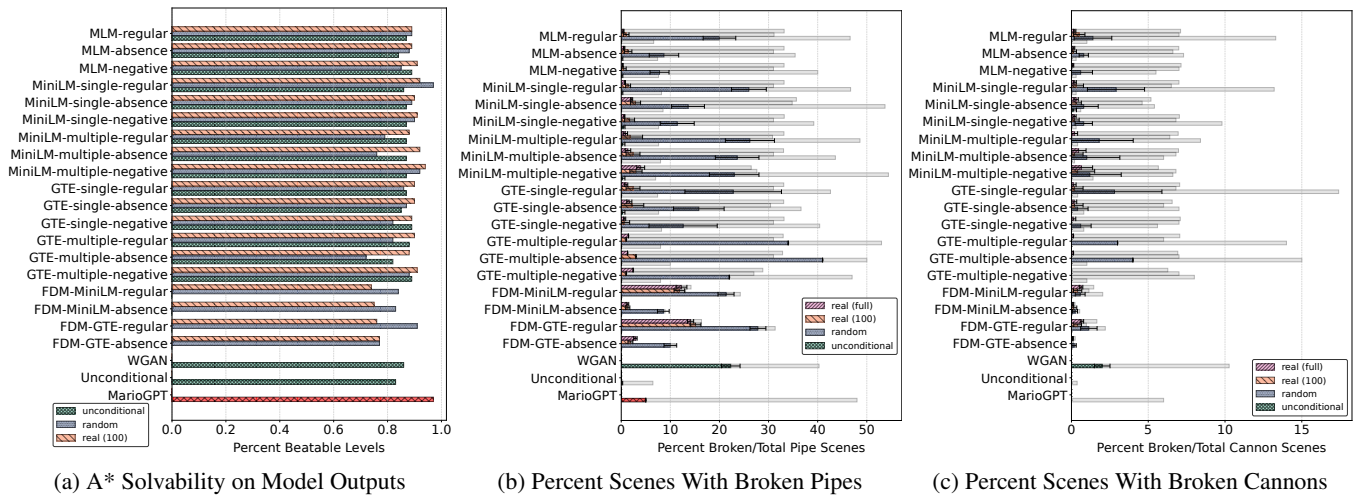


Figure 5: (a) A* results show the percentage of beatable scenes. Only one model and 100 scenes of each type are used, so there are no error bars. (b) For each dataset, colored bars with 95% confidence intervals show the percentage of scenes that contain a broken pipe. The gray bar extending from each bar represents the scenes with pipes of any kind: regular, broken, or upside down. Producing few broken pipes is more impressive if valid pipes are also generated, as indicated by larger gray regions. Most models produce few broken pipes, but more with random captions. FDM-regular models produce more broken pipes than FDM-absence. WGAN produces many broken pipes. (c) Broken cannon data is presented similarly. Many datasets have no broken cannons, though cannons in general are rarer than pipes.

dom scores are much lower. MarioGPT’s scores are higher than most, though the highest random scores of certain diffusion models are more novel.

Solvability Figure 5a shows the percentage of beatable scenes from each model. Since simulation takes time, we only apply A* to 100 samples from real captions per one model of each type as opposed to all model outputs across all real captions. Even the worst, GTE-multiple-absence with random captions, produces 72% beatable scenes. FDM results from real captions are between 74-77%. The highest score is a tie at 97% between MarioGPT and MiniLM-single-regular’s random caption samples. Most diffusion models have scores between 82% and 94%.

Level Integrity Figure 5b shows the percentage of generated scenes that contain one or more broken pipes and the percentage of scenes with any kind of pipe. For datasets with 100 samples, the percentage is also the count. Most models produce few scenes with broken pipes using real captions, but many using random captions. Captions produce many valid pipes too. FDM creates more broken pipes with regular captions than absence captions. MarioGPT has 5 broken pipe scenes, which is more than diffusion models on real captions, though less than diffusion models on random captions. WGAN produces many broken pipes, but unconditional scenes from diffusion models have almost no broken pipes, though they have fewer valid pipes as well.

Figure 5c shows similar results for broken cannons. Cannons are rarer, and broken ones rarer still, though more broken cannons are associated with random captions and WGAN. Unconditional samples have almost no broken cannons, but very few valid ones either.

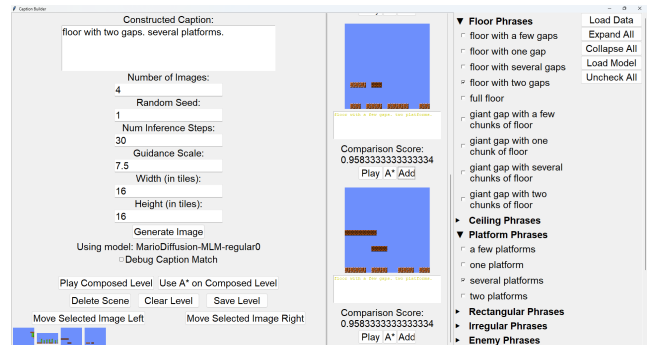


Figure 6: Interactive Level Building Interface. Phrases are selected from the right which create a caption in the upper-left. Then samples can be generated which appear in the center column. These can be added to a level in the lower-left which can be played by a human or A* agent.

Larger Levels

Although the diffusion models are trained on 16×16 samples, the output size can be any value during inference, making it possible to generate levels of arbitrary length. However, there are two problems. First, input prompts are calibrated for smaller scenes, so it is not clear what one should expect from larger scenes, or how to assess them. Secondly, longer levels seem to often be unbeatable due to levels having massive gaps that cannot be jumped over.

It may be possible to address these issues with a dataset consisting of longer levels, but confirming this is a task for future work. In the meantime, human designers can still benefit from tools we have designed by incorporating diffusion

models into a mixed-initiative system with a GUI for building complete levels from diffusion-generated scenes. The interface (Figure 6) supplies checkboxes for valid phrases organized by topic (floor, coins, etc.) so that creating descriptive captions is easy to do without worrying about spelling or the use of unknown vocabulary. Once a caption is constructed, level scenes can be generated with a chosen model. Parameters like the random seed, number of samples, number of inference steps, guidance scale, and scene width can be set. Generated scenes have automatically generated captions that are color-coded to visually indicate differences/similarities to the user-supplied prompt. Caption adherence score is also displayed. Individual scenes can be combined into a larger level. Scenes are added to the end of the level by default, but they can be moved or deleted by the user. Both constructed levels and individual scenes can be tested via human play or with A*. Finally, ASCII text versions of constructed levels can be saved for future use. Examples of longer composed levels are in the online appendix, as is a video demonstrating how to use the GUI: <https://people.southwestern.edu/~schrum2/mario.html>

Co-creative level design systems have been explored in the academic literature before (Larsson, Font, and Alvarez 2022), including for Mario (Guzdial, Liao, and Riedl 2018; Schrum et al. 2020). Evolutionary computation and various types of AI models are common in such work, but the ability to mix and match components produced from text prompts is new, and opens up possibilities for numerous human subject studies. Experimental verification of the benefits of such a system is still needed, but the authors found it easy to create scenes of different sizes and combine them into fun and interesting levels of arbitrary length. The generation of multiple scenes per caption along with the ability to change the random seed and guidance scale make it easier to find scenes that match a desired caption, and users may also be serendipitously inspired by scenes they create, even when they do not match the input caption. We hope to study the experiences of users interacting with this system in future work.

Limitations

Our approach to diffusion-based PCGML requires a sufficiently sized dataset of level scenes and enough expert knowledge and programming skill to assign adequate captions to such scenes algorithmically. However, we believe this is a modest barrier and are actively expanding our research to other games. Although we used NVIDIA GPUs, they were not excessively expensive; a decent gaming PC is able to train our models. Long levels are less likely to be beatable, but our GUI provides an effective way to make beatable long levels with slight additional effort.

Discussion and Conclusion

It was surprising that a small and basic transformer architecture with little training on a small dataset with limited vocabulary produced the best results when combined with our diffusion model. It was also disappointing that attempts to enhance the approach had little effect or a detrimental effect. Models take longer to train with `negative` prompts,

and are not better. The `absence` captions are more complicated, but offer no benefit. `MiniLM` and `GTE` are more powerful and general language models, but do not produce definitively better results, which is especially damning in the case of `GTE`, whose models take much longer to train. We thought `multiple` sentence embeddings could provide richer context for diffusion, but they simply increased training time. Of course, it is not bad that a simpler model can be so effective, though we wonder if other pretrained language models or alternative architectures might be better.

There must be some way to break the average caption score barrier of 0.5 on random captions. In fairness, some of the random captions are very unusual. Actual captions in the dataset include “two ascending staircases.” and “two question blocks. two enemies. two cannons.” These captions do not mention a floor. Although levels without floors exist, such levels tend to have many platforms to support entities like cannons and enemies.

We tested larger UNet architectures in our preliminary experiments, but they only seemed to increase training time with no clear benefit. We admit that a more systematic exploration of different architectures and hyperparameters could lead to improvements. However, we suspect the biggest limitation is the training data, but part of that problem could be solved with our automatic captioning system. Although our model may not always produce scenes with desired captions, it can produce scenes with captions that do not exist in the dataset. If such scenes were to be automatically captioned and added to the training dataset, it may be possible to gradually accumulate enough scenes to get near complete coverage of the space of captions. Our results suggest that once a caption is in the dataset, our model would have no trouble generating a scene that matches it.

MarioGPT does well in several metrics, though our best diffusion models are comparable or superior in certain instances. However, it is unclear what the best comparison is: real, random, or unconditional. As stated above, our diffusion models do well with captions they’ve seen during training; MarioGPT’s limited caption options makes it less likely to see an unfamiliar caption. It would be interesting to compare diffusion with simpler captions, or MarioGPT with more complex captions.

For now, we conclude, having presented a method to make captions for Mario level scenes which allows for the training of a transformer text encoder and diffusion model that can make realistic Mario scenes, which can be combined into larger levels using our GUI.

Acknowledgments

This paper was initially drafted without AI, but ChatGPT was later used to improve clarity/concision. We thank donors to Southwestern University’s SURF program for support.

References

Awiszus, M.; Schubert, F.; and Rosenhahn, B. 2020. TOAD-GAN: Coherent Style Level Generation from a Single Example. In *Artificial Intelligence and Interactive Digital Entertainment*. AAAI.

- Awiszus, M.; Schubert, F.; and Rosenhahn, B. 2021. WorldGAN: a Generative Model for Minecraft Worlds. In *Conference on Games*, 1–8. IEEE.
- Bontrager, P.; Lin, W.; Togelius, J.; and Risi, S. 2018. Deep Interactive Evolution. In *European Conference on the Applications of Evolutionary Computation (EvoApplications)*.
- Dai, S.; Zhu, X.; Li, N.; Dai, T.; and Wang, Z. 2024. Procedural Level Generation with Diffusion Models from a Single Example. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(9): 10021–10029.
- Fontaine, M.; Hsu, Y.-C.; Zhang, Y.; and Nikolaidis, S. 2021. On the Importance of Environments for Human-Robot Coordination. In *Proceedings of Robotics: Science and Systems*.
- Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative Adversarial Nets. In *Neural Information Processing Systems*, 2672–2680.
- Gutierrez, J.; and Schrum, J. 2020. Generative Adversarial Network Rooms in Generative Graph Grammar Dungeons for The Legend of Zelda. In *Congress on Evolutionary Computation*. IEEE.
- Guzdial, M.; Liao, N.; and Riedl, M. 2018. Co-Creative Level Design via Machine Learning. In *Proceedings of the Experimental AI in Games (EXAG) Workshop at AIIDE*.
- Kingma, D. P.; and Welling, M. 2014. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*.
- Kumaran, V.; Mott, B. W.; and Lester, J. C. 2020. Generating Game Levels for Multiple Distinct Games with a Common Latent Space. In *Artificial Intelligence and Interactive Digital Entertainment*. AAAI.
- Larsson, T.; Font, J.; and Alvarez, A. 2022. Towards AI as a Creative Colleague in Game Level Design. In *Artificial Intelligence and Interactive Digital Entertainment*. AAAI.
- Lee, H. J.; and Simo-Serra, E. 2023. Using Unconditional Diffusion Models in Level Generation for Super Mario Bros. In *International Conference on Machine Vision and Applications*, 1–5.
- Merino, T.; Negri, R.; Rajesh, D.; Charity, M.; and Togelius, J. 2023. The Five-Dollar Model: Generating Game Maps and Sprites From Sentence Embeddings. In *Artificial Intelligence and Interactive Digital Entertainment*. AAAI.
- Nie, Y.; Middleton, M.; Merino, T.; Kanagaraja, N.; Kumar, A.; Zhuang, Z.; and Togelius, J. 2025. Moonshine: Distilling Game Content Generators into Steerable Generative Models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(13): 14344–14351.
- Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*.
- Rombach, R.; Blattmann, A.; Lorenz, D.; Esser, P.; and Ommer, B. 2022. High-Resolution Image Synthesis with Latent Diffusion Models. In *Computer Vision and Pattern Recognition*, 10674–10685. IEEE.
- Sarkar, A.; and Cooper, S. 2021. Generating and Blending Game Levels via Quality-Diversity in the Latent Space of a Variational Autoencoder. In *Proceedings of the Foundations of Digital Games*.
- Sarkar, A.; Yang, Z.; and Cooper, S. 2020. Conditional Level Generation and Game Blending. In *Proceedings of the Experimental AI in Games (EXAG) Workshop at AIIDE*.
- Schrum, J.; Capps, B.; Steckel, K.; Volz, V.; and Risi, S. 2023. Hybrid Encoding for Generating Large Scale Game Level Patterns With Local Variations. *IEEE Transactions on Games*, 15(1): 46–55.
- Schrum, J.; Gutierrez, J.; Volz, V.; Liu, J.; Lucas, S.; and Risi, S. 2020. Interactive Evolution and Exploration Within Latent Level-Design Space of Generative Adversarial Networks. In *Genetic and Evolutionary Computation Conference*. ACM.
- Schrum, J.; Volz, V.; and Risi, S. 2020. CPPN2GAN: Combining Compositional Pattern Producing Networks and GANs for Large-scale Pattern Generation. In *Genetic and Evolutionary Computation Conference*. ACM.
- Shaker, N.; Togelius, J.; and Nelson, M. J. 2016. *Procedural Content Generation in Games*. Springer.
- Sudhakaran, S.; González-Duque, M.; Freiburger, M.; Glanois, C.; Najarro, E.; and Risi, S. 2023. MarioGPT: Open-Ended Text2Level Generation Through Large Language Models. In *Neural Information Processing Systems*.
- Summerville, A.; and Mateas, M. 2016. Super Mario as a String: Platformer Level Generation via LSTMs. In *1st International Joint Conference of DiGRA and FDG*.
- Summerville, A.; Snodgrass, S.; Guzdial, M.; Holmgård, C.; Hoover, A. K.; Isaksen, A.; Nealen, A.; and Togelius, J. 2018. Procedural Content Generation via Machine Learning (PCGML). *IEEE Transactions on Games*, 10(3): 257–270.
- Summerville, A. J.; Snodgrass, S.; Mateas, M.; and Ontañón, S. 2016. The VGLC: The Video Game Level Corpus. In *Procedural Content Generation in Games*. ACM.
- Thakkar, S.; Cao, C.; Wang, L.; Choi, T. J.; and Togelius, J. 2019. Autoencoder and Evolutionary Algorithm for Level Generation in Lode Runner. In *Conference on Games*, 1–4. IEEE.
- Todd, G.; Earle, S.; Nasir, M. U.; Green, M. C.; and Togelius, J. 2023. Level Generation Through Large Language Models. In *Foundations of Digital Games*. ACM.
- Togelius, J.; Karakovskiy, S.; and Baumgarten, R. 2010. The 2009 Mario AI Competition. *Congress on Evolutionary Computation*, 1–8.
- Valevski, D.; Leviathan, Y.; Arar, M.; and Fruchter, S. 2024. Diffusion Models Are Real-Time Game Engines. arXiv:2408.14837.
- Virtuals Protocol. 2024. Video Game Generation: A Practical Study Using Mario. Preprint.
- Volz, V.; Schrum, J.; Liu, J.; Lucas, S. M.; Smith, A. M.; and Risi, S. 2018. Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network. In *Genetic and Evolutionary Computation Conference*. ACM.
- Šosvald, D.; Töpfer, M.; Holan, J.; Černý, V.; and Gemrot, J. 2021. Super Mario A-Star Agent Revisited. In *International*

Conference on Tools with Artificial Intelligence, 1008–1012. IEEE.

Yang, L.; Zhang, Z.; Song, Y.; Hong, S.; Xu, R.; Zhao, Y.; Zhang, W.; Cui, B.; and Yang, M.-H. 2023. Diffusion Models: A Comprehensive Survey of Methods and Applications. *ACM Computing Surveys*, 56(4).

Zhang, X.; Zhang, Y.; Long, D.; Xie, W.; Dai, Z.; Tang, J.; Lin, H.; Yang, B.; Xie, P.; Huang, F.; Zhang, M.; Li, W.; and Zhang, M. 2024. mGTE: Generalized Long-Context Text Representation and Reranking Models for Multilingual Text Retrieval. In *Empirical Methods in Natural Language Processing: Industry Track*, 1393–1412. Association for Computational Linguistics.