Exploring MarioGPT: Open-Ended Text2Level Generation through Large Language Models

Viet Nhat Nguyen, Thi Lien Le
University of Information Technology,
Vietnam National University HCMC
{ 21520378, 21522282 } @gm.uit.edu.vn

Abstract

In this study, we present our exploration of MarioGPT, a fine-tuned GPT2 model trained to generate tile-based game levels, focusing on Super Mario Bros levels discussed in paper (Sudhakaran et al., 2023). Procedural Content Generation (PCG) algorithms offer automated generation of complex and diverse environments. However, generating meaningful content that aligns with specific intentions and constraints remains challenging. Many current PCG algorithms struggle to produce open-ended content. Large Language Models (LLMs) have demonstrated remarkable effectiveness across various domains. These pretrained LLMs can be fine-tuned, leveraging existing knowledge and accelerating training for new tasks. MarioGPT not only generates diverse levels but also supports text prompts for controlled level generation, addressing a key limitation of current PCG techniques. Integration with novel search enables the generation of levels with varying playstyle dynamics, such as player paths, resulting in an increasingly diverse range of open-ended content. Replicating and understanding MarioGPT's capabilities will advance the field of PCG by offering a flexible and controllable approach to creating game levels.

Keywords: PCG, LLMs, Genetic Algorithms, Novelty Search, Transformers

1 Introduction

Curiosity plays a crucial role in motivating players, and to sustain this curiosity, it is important to provide them with fresh and diverse content. Diversification primarily arises from exploring different situations within the established game mechanics, particularly during the level design phase. Currently, this exploration is mainly conducted manually, relying on the intuition of design teams.

Procedural content generation (PCG), especially for levels, is widely employed in game design. Its principle involves automatically creating playable levels using algorithmic techniques. It is good that PCG offers several benefits in the field of game development and beyond (e.g. increased replayability, lower costs). However, controlling the generation process presents a classic challenge. Game design requires the ability to impose specific qualities on the generated levels, such as appropriate difficulty, sufficient diversity, or desired object density on the screen. Exploring the high-dimensional state space of levels is difficult, often requiring the use of optimization metaheuristics like genetic programming or simulated annealing (Togelius et al., 2011). Implementing these algorithms can be costly, slow, and necessitates defining complex fitness metrics.

To address these challenges, in this paper, we conduct a research and re-implement MarioGPT (Sudhakaran et al., 2023), a fine-tuned GPT-2 model trained to generate Mario levels, to see whether given model can demonstrate how LLMs can be combined with PCG techniques enabling the effective creation of new and diverse levels through natural language prompts.

Furthermore, we also conduct a research on how MarioGPT can be combined with novelty search (Lehman and Stanley, 2011), a diversity-seeking algorithm, to produce diverse levels in an open-ended fashion.

2 Background and Related Work

2.1 Neural Network-based Level Generation

More recently, the game development community has embraced statistical learning, particularly generative models, as a central tool in procedural content generation algorithms. Deep learning, in particular, has gained popularity in PCG (Liu et al., 2021). While (Beukman et al., 2022) leveraged neural networks to generate levels, other works (Fontaine et al., 2021; González-Duque et al., 2022; Volz et al., 2018) performed evolution / search in the latent space of a trained generative model. These studies demonstrated that strategically sampling the latent space of the trained generative model led to the generation of a wide range of diverse levels, and the abilities to control characteristics in generated levels.

However, the generative networks used in these methods did not directly generate playable paths within the levels, necessitating the use of an external evaluation agent (e.g., A*) to assess playability. This additional evaluation step can be computationally demanding, especially for larger levels, resulting in increased computational costs. Moreover, to incorporate prompt information, these methods require searching the latent space for levels with specific target characteristics (e.g., levels with certain blocks). This represents a limitation, as it entails searching through the latent space despite the generative models containing a rich set of content. In summary, while deep learning has shown promise in PCG, the generation of levels with specific characteristics and the assessment of playability still come with significant computational costs.

As we'll show in the following sections, MarioGPT offers a distinct advantage over these algorithms as it can generate diverse levels while accurately predicting the paths that an agent would traverse within the level. Moreover, MarioGPT integrates the desired characteristics directly into the generation process, eliminating the necessity of searching for levels with specific characteristics in the latent space. In other words, MarioGPT requires less computational overhead and allows us to just ask for a controllable level generation.

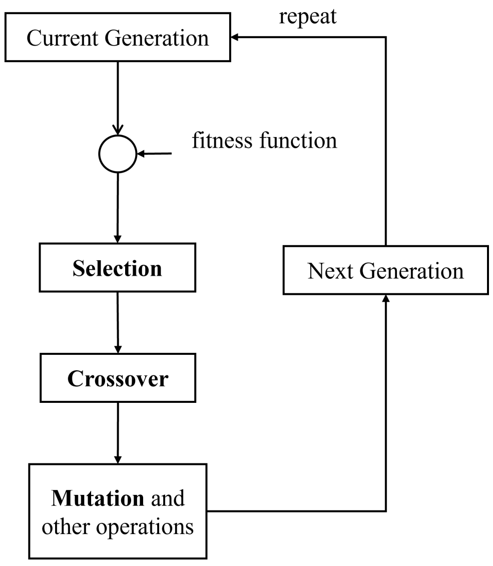


Figure 1: Architecture of a Genetic algorithm

2.2 Open-Endedness Paradigm

The open-endedness paradigm focuses on the development of algorithms that can produce infinite innovation and novel content (Kenneth O. Stanley, 2017). These algorithms are highly regarded in the field of Procedural Content Generation (PCG) as they offer significant benefits to both game designers and players. Designers can leverage open-ended algorithms to create diverse and never-ending content, providing unique and engaging experiences to players. However, PCG algorithms face the challenge of striking a delicate balance between generating content with high diversity while ensuring it remains playable and enjoyable for the players. Achieving this balance is crucial to provide players with compelling and engaging game experiences while offering an endless variety of content to explore. Genetic algorithms (GA), a family of optimization algorithms, are commonly used as the backbone for more open-ended search methods.

2.3 Genetic Algorithms

2.3.1 Genetic Algorithms (GA)

Genetic algorithms (Sampson, 1976) are a class of optimization algorithms inspired by the principles of natural selection and genetics. They are widely used in various fields, including artificial intelligence, machine learning, and optimization problems.

GA use concepts from *evolutionary biology* to find a global minimum for a optimization problem. The basic idea behind genetic algorithms is to mimic the process of natural evolution by simulating the mechanisms of *selection*, *crossover*, and *mutation*. The algorithm starts with a population of candidate solutions, often represented as individuals or *chromosomes*, that are tested against the objective function. Each candidate solution represents a potential solution to the problem at hand.

In the GA paradigm (Figure 1), a population of candidate solutions (chromosomes), a part of the new populations is selected based on the *survival of the fittest* principle, as determined by an objective of *fitness* function. In order to preserve or enable genetic pluralism, *mutation* and *crossover* occur with certain probability.

The steps involved in creating and implementing a genetic algorithm are as follows:

1. Initialize arbitrarily first generation of chromosomes.
2. Evaluate their fitness (how close a chromosome is to the optimum chromosome).
3. Select members of population (by deterministic or stochastic selection).
4. Implement crossover operation on the reproduced chromosomes.
5. Execute mutation operation with certain probability.

One key advantage of genetic algorithms is their ability to search through a large search space efficiently. They can handle complex optimization problems with numerous variables and constraints. GAs also have the ability to explore multiple regions of the search space simultaneously, allowing them to discover diverse solutions.

However, genetic algorithms have some limitations. GA's are known to be slow in practice, which is not really a surprise, since they are used to solve hard optimization problems. They may require a large number of iterations to converge to an optimal or near-optimal solution. Additionally, the performance of GAs heavily relies on the representation of individuals, the selection and reproduction mechanisms, and the definition of the fitness function.

Despite their limitations, genetic algorithms (GAs) are well-suited for achieving a balance between fitness and diversity in open-ended algorithms within the field of PCG. This is primarily due to their ability to integrate multiple objectives, enabling the exploration of a diverse range of solutions while optimizing for desired fitness criteria.

2.3.2 Novelty Search

In the context of genetic algorithms, novelty search (NS) approaches (Lehman and Stanley, 2011) focus on discovering the most unique solutions in each generation, distinct from what has been previously encountered (i.e., stored in an *archive* of previous solutions).

The motivation behind the proposed goal function definition is the hypothesis behind NS: a fitness gradient may not always be available and/or might be misleading, and it is therefore more appropriate not to follow it. Novelty Search replaces the search for a greater fitness by a search for novel individuals. Thus, instead of an objective function, search employs a *novelty metric*.

The novelty metric measures how different an individual is from other individuals, creating a constant pressure to do something new. The novelty metric must quantify these differences and be tailored to the specific domain. It compares the *behaviors* of newly generated individuals to those stored in an archive of past individuals with high novelty. In steady-state evolutionary algorithms, the current population can also contribute to the archive. The goal is to assess how far the new individual deviates from the rest of the population and predecessors in terms of novelty. A suitable metric should calculate the sparsity in the novelty space, with less novel areas having denser clusters of visited points and receiving lower rewards.

A simple measure of *sparseness* at a point in a behavior space (elicit a chromosome in genotype space) is the mean distance to the k-nearest neighbors of that point, where k is a fixed parameter that is determined experimentally. Intuitively, the smaller the value sparseness, the denser region that point is in, result in the less novel of the individual.

$$\rho(x) = \frac{1}{k}\sum_{i=0}^{k} dist(x, \mu_i) \tag{1}$$

where $\{\mu_1, \mu_2, ..., \mu_k\}$ is the set of the k-closest neighbors to x with respect to the distance metric $dist$, which is a domain-dependent measure of behavioral difference between two individuals in the search space.

In general, the set of individuals used to measure the novelty of new solutions corresponds to the current population, plus an archive of previous individuals, and the commonly metric used to compare behaviors is the Euclidean distance. There are various strategies for managing the archive of past solutions. It can contain either individuals with sufficient novelty score (sparseness), i.e. above some minimal threshold ρ_{min} (Lehman and Stanley, 2011), the most novel individuals at each generation (Liapis et al., 2015), randomly chosen individuals (Lehman and Stanley, 2010), or even not at all (Mouret and Doncieux, 2012).

A key benefit of novelty search is that it keeps track of solutions in an archive and measures diversity by the distance between their behavior characteristics (BCs) compared to that of their k closest neighbors. This makes novelty search very flexible, allowing for the use of many different behavior characteristic types. But what makes NS powerful, and motivated its use in this paper, is that it guides generation towards increasingly diverse solutions in an open-ended fashion.

2.4 Sequence Modelling and Transformers

2.4.1 Sequence Modelling and Large Language Models

Sequence modelling is a powerful technique in the field of machine learning that focuses on capturing and understanding the sequential nature of data. The core idea behind sequence modelling is to model the dependencies and patterns that exist within a sequence of data points. This enables the model to learn and predict the next element in the sequence based on the previous ones. By presenting game levels as a sequence of tokens, we can leverage sequence modelling to generate video game levels, particularly in Mario.

In recent years, attention-based large language models (LLMs), a class of sequence models, have taken the world by storm. They leverage the power of deep learning and attention mechanisms to capture intricate patterns and dependencies within sequences, especially in language-related tasks. In MarioGPT architecture, LLMs, i.e. Generative Pre-trained Transformer (GPT) and BERT are leveraged to produce diverse levels, allowing for the generation of coherent and engaging game experiences.

Traditional approaches to sequence modelling, such as recurrent neural networks (RNNs) (Rumelhart et al., 1985) and Long Short-Term Memory (LSTM) networks (Hochreiter and Schmidhuber, 1997), have historically faced limitations. These include the issue of fading memory in the network's state vector and challenges in scalability due to the temporal interdependency of operations. Transformers (Vaswani et al., 2017) have emerged as a powerful solution to address the limitations of traditional sequence modelling approaches. Additionally, Transformers offer improved scalability as they can be parallelized, enabling efficient computation on large datasets. This makes them well-suited for handling sequences with long-range dependencies and achieving good scalability in sequence modelling tasks.

2.4.2 Transformers

Transformers consist of two main components: the encoder and the decoder, however, their structure and operation mechanisms are completely different.

- **Encoder**: In the Transformer architecture, each encoder block consists of identical layers, typically with a fixed number of layers ($N = 6$ in the original paper). Each layer in the encoder block comprises two sub-layers: the Multi-Head Attention and the Feed Forward Network. Additionally, each sub-layer in the encoder block utilizes residual connections. The incorporation of residual connections allows the model to be easily scaled and prevents the issue of gradient vanishing, enabling more efficient training and deeper architectures.

- **Decoder**: Similarly to the encoder, the decoder in the Transformer architecture is also composed of 6 layers. Each layer consists of two sub-layers, with the addition of an extra sub-layer called Masked Multi-Head Attention. This sub-layer takes the encoder's output as input and incorporates a residual connection. The inclusion of the residual connection allows for easier gradient flow and helps improve the model's performance in capturing dependencies and generating accurate predictions during the decoding process.

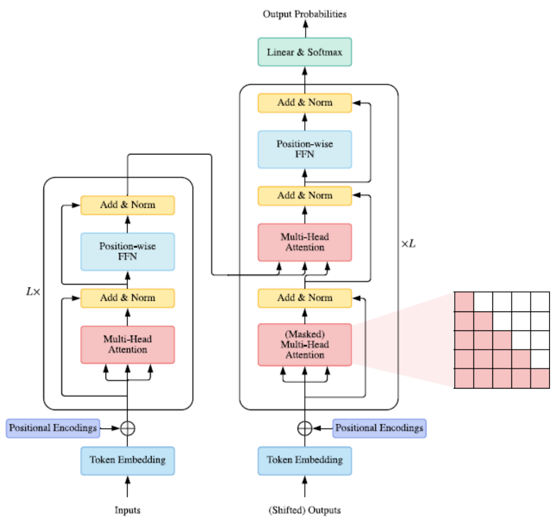


Figure 2: The Transformer - model architecture (re-draw based on (Vaswani et al., 2017))

Multi-Head Attention

Multi-head attention is a key component in the Transformer architecture that enables the model to capture different types of information and dependencies within a sequence. It operates by transforming the input sequence into multiple representations, called attention heads. Each attention head attends to different parts of the input sequence, allowing the model to capture both local and global dependencies.

Multi-Head Attention is one of the crucial mechanisms in the Transformer model. Attention can be understood as a mapping mechanism from a query and a set of key-value pairs to an output, where the *queries*, *keys*, and *values* are all vectors. The output is a weighted sum of the values, with weights calculated based on the query's correspondence to the keys. In other words, we can view the query as a query sentence to find the corresponding sentence embeddings (keys), and we obtain the content or meaning of those sentences as the values. In the paper, the authors denote the query, keys, and values as Q, K, and V, respectively.

In the Multi-Head Attention architecture, each input sequence is transformed linearly using separate Linear layers for Q, K, and V (Figure 3). This enables the model to learn attention weights and capture the meaning of the sentence from multiple perspectives. The attention weights are then combined through the Scaled Dot Product Attention operation.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2}$$

where d_k is the dimension of queries Q and keys K.

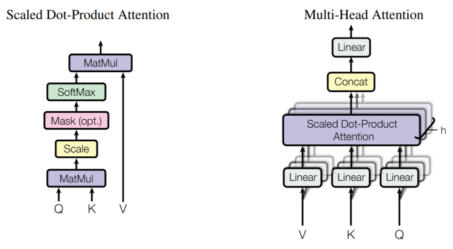


Figure 3: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel. (Vaswani et al., 2017)

Positional Encoding

Due to the lack of a recurrent architecture in the Transformer model, additional information about the relative or absolute positions of words in a sentence is needed. In the paper, the authors employ sine and cosine functions at different frequencies to incorporate positional information.

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(pos/10000^{2i/d_{model}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(pos/10000^{2i/d_{model}}\right) \end{aligned} \tag{3}$$

By using sine and cosine functions at different frequencies, the model is able to encode the relative and absolute positions of words, enabling it to understand the sequential nature of the input data. This positional encoding mechanism enhances the model's ability to capture long-range dependencies and improves its performance.

2.4.3 GPT and BERT

In NS-MarioGPT, we will leverage LLMs model, i.e. GPT-2 (Radford et al., 2019) and BERT (Devlin et al., 2019) for game levels generation.

GPT, a generative model, is composed of a stack of decoder layers, each containing multi-head self-attention mechanisms and position-wise feed-forward networks. This architecture allows GPT to generate coherent and contextually rich text by capturing dependencies and contextual information within the input sequence.

In contrast, BERT follows an encoder-only architecture and consists of transformer encoder layers with self-attention mechanisms and position-wise feed-forward networks. BERT leverages bidirectional training on large-scale unlabeled text data, enabling it to capture fine-grained contextual representations. By incorporating left and right context during training, BERT achieves a deeper understanding of language semantics.

These architectural innovations have enabled Large Language Models (LLMs) to learn from massive datasets. Additionally, such models have also shown to be effective in accelerated learning of down-stream tasks. Fine-tuning LLMs (Devlin et al., 2019) involves using pre-trained model weights as a weight initialization for new tasks. Fine-tuning bidirectional LLMs (Devlin et al., 2019; Sanh et al., 2020) and unimodal LLMs (Radford et al., 2019) have shown impressive results on mask prediction and sequence modelling tasks, respectively.

One particularly relevant use of pretrained / fine-tuned LLMs comes from the method Evolution through Large Models (ELM), proposed in (Lehman et al., 2022). ELM utilizes an LLM diff model (HackTech, 2023), which is trained on code diffs obtained by Github data, giving the model the ability to modify a code snippet based on a particular commit message. This diff model is used as a "mutation operator", for a GA that evolves a population of programs. The wide generative capabilities of the LLM produce incredibly diverse mutations, resulting in novel individuals that vary increasingly over the course of the GA.

3 Open-Ended Level Generation through LLMs

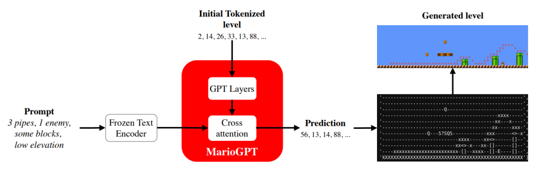


Figure 4: The MarioGPT prediction pipeline involves a fine-tuned GPT2 language model. Levels are represented as strings and tokenized using Byte-Pair Encoding. Prompt information is incorporated using a frozen pretrained bidirectional LLM (BART), with the average hidden states used in the cross-attention layers of the GPT2 architecture along with the level sequence.

In this section, we present our comprehensive approach to open-ended level generation using LLMS, consisting of two main parts. Firstly, we introduce our prompt-conditioned model MarioGPT. This model generates levels encoded as text, based on natural language prompts. Secondly, we provide detailed information on how MarioGPT can be utilized in a novelty-search evolutionary loop discussed in Section 3.4. This loop enables our approach to produce a continuous stream of diverse levels.

Prior to delving into the specifics of the MarioGPT model, we introduce the Mario level representation in the following section.

3.1 Level Representation

Table 1: Unique Mario tiles

Tile Type	Symbol	Visualization
Empty	-	🟦
Unbreakable	X	🟫
Breakable	S	🟧
Question Block	? / Q	🟨
Coin	o	🟡
Enemy	E	🟤
Left pipe top	<	🟩
Right pipe top	>	🟩
Left pipe lower	[	🟩
Right pipe lower	]	🟩
Cannon Top	B	⬛
Cannon Body	b	🟥
Path	x	🟦

Mario levels are represented similarly to previous works (Matthew C. Fontaine and Nikolaidis, 2020; Miguel

González-Duque and Risi, 2022), using levels sourced from the Video Game Level Corpus (VGLC) (Adam James Summerville and Ontañón, 2016). It utilize a relatively small set of path-annotated levels taken from Super Mario Bros (comprising a total of 37 levels). These levels are stitched together to create one extensive level, allowing us to freely sample without being concerned about level boundaries. The levels are represented as strings of tiles, encoding specific game objects as shown in Table 1. The string representation and characters are tokenized into discrete values using a Byte Pair Encoding tokenizer, as employed in the original GPT2 model (Alec Radford and Sutskever, 2019). One limitation of the dataset is the simplified representation of enemies. Although levels contain various enemies with different behaviors and features, the dataset represents them all with the same token.

3.2 MarioGPT Model

MarioGPT, an advanced prompt-conditioned language model, has gained attention in the field of procedural content generation. It is based on the DistilGPT2 transformer architecture, which is known for its lightweight and efficient design (Victor Sanh and Wolf, 2019). By fine-tuning this model, MarioGPT demonstrates remarkable performance in predicting long sequences, particularly in generating levels for Mario games. The approach involves encoding slices of Mario levels as strings, following a similar technique proposed in previous studies (Summerville and Mateas., 2016). The resulting model is capable of generating levels by processing a window of the previous 50 columns as a single vector. This utilization of transformer architectures, known for their wider attention span compared to LSTM and other traditional models used in procedural content generation (Summerville and Mateas., 2016, 2018), leads to improved level quality and increased length.

Architecture: MarioGPT's architecture closely resembles that of DistilGPT2, with the notable addition of utilizing cross-attention weights for prompting. While DistilGPT2 supports context lengths up to 1024, but limiting the context length to 700 yields optimal performance gains. In total, MarioGPT boasts 96 million parameters, comprising 86 million from the original DistilGPT2 parameters and an additional 10 million from cross-attention weights.

Incorporating Prompts: To incorporate prompt information into the level generation process, MarioGPT employ a fine-tuning approach for the cross-attention weights of the attention layers (see Figure 4). Prompts are encoded using BART (Mike Lewis and Zettlemoyer, 2019), a pre-trained language model that remains frozen throughout the process. The prompts undergo encoding by passing them through the frozen language model, and the resulting hidden states from the forward pass are then averaged to derive a single vector. This average hidden state is subsequently combined with the actual level sequence in the cross-attention layers of the GPT2 architecture.

Prompts are represented as combinations of specific features, including pipes, enemies, blocks, and elevation, alongside quantitative keywords. The following prompts exemplify the format:

- Pipes: no, little, some, many, [0-1000]
- Enemies: no, little, some, many, [0-1000]
- Blocks: little, some, many, [0-1000]
- Elevation: low, high

Table 2: Prompt Quantiles and corresponding counts within a 50 column window

tile	no	little	some	many
pipes	0	1	2	5
enemies	0	1	3	7
blocks	0	50	75	176

For instance, valid prompts could be ”no pipes, many enemies, low elevation” or ”many pipes, many enemies, many blocks.” The keywords ”no,” ”little,” ”some,” and ”many” are calculated based on quantiles derived from the corresponding count within a 50-column window, as detailed in Table 2. The ”low” and ”high” elevation are determined by the highest unbreakable blocks' height within a specific segment of the level.

3.3 Open-Ended Mario Level Generation with Novelty Search

In the realm of procedural content generation, the focus of MarioGPT extends beyond creating levels with diverse physical features. The ultimate goal is to design levels that evoke a wide range of **player behaviors**, challenging traditional algorithms (Matthew C. Fontaine and Nikolaidis, 2020; Summerville and Mateas., 2018). With MarioGPT, the need for external evaluation agents becomes obsolete as it enables the generation of diverse and controllable levels that closely emulate realistic player paths. To promote diversity in level generation, the novel NS-MarioGPT framework is introduced, incorporating a novelty-search augmented genetic algorithm. Within this framework (illustrated in Figure 5), MarioGPT functions as a mutation operator, facilitating the sampling and mutation of elite levels from a pre-existing archive.

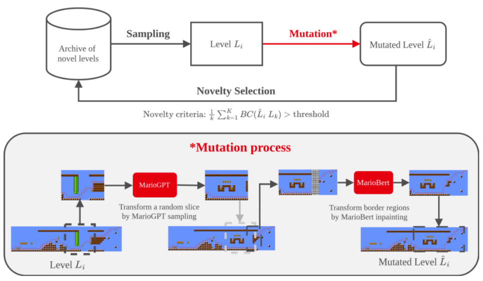

Archive of novel levels
Sampling
Level L_i
Mutation*
Mutated Level $\hat{L}_i$
Novelty Selection
Novelty criteria: $\frac{1}{k}\sum_{k=1}^{K} BC(\hat{L}_i\ L_k) > \text{threshold}$
*Mutation process
MarioGPT
Transform a random slice by MarioGPT sampling
MarioBert
Transform border regions by MarioBert inpainting
Level L_i
Mutated Level $\hat{L}_i$

Figure 5: Novelty search setup and MarioGPT mutation operators. A level is sampled from a set of top elites in the archive, mutated, and, if novel enough, selected to join the archive. The mutation process involves two main steps: (1) Pick a random slice from the level and replace it with a new MarioGPT sample, using a random prompt. (2) Inpaint the border region with MarioBert to preserve path consistency

Novelty Search: The pursuit of novelty drives the retention of mutated levels in the archive, subject to surpassing the novelty score of previous elite levels. The novelty score is calculated based on the mean distance between the behavioral characteristic vectors of the levels and the vectors of the K closest elements in the archive (K-means). The primary objective is to create paths that evoke diverse player behaviors. This is achieved by leveraging predicted player paths as the foundation for defining behavioral characteristics. The focus is on the relative patterns exhibited by these paths, enabling the emergence of paths that exhibit similar behavior in different scenarios. Normalizing the average coordinates of predicted paths ensures a smooth representation, minimizing the impact of minor differences and preventing the archive from being inundated with levels offering minimal variation. The novelty search experiments employ a neighborhood size of 4, resulting in a 100-dimensional representation of behavioral characteristics. With an initial archive size of 30 levels, a balance is struck between diversity and generation efficiency.

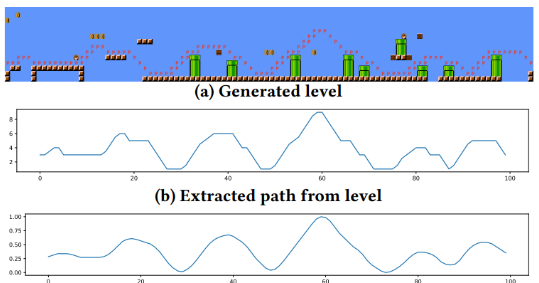


Figure 6: Novelty search behavior characteristic. The behavior characteristic of a level (a) is represented as the moving average, with a window of 5, of predicted path coordinates (b). Its smoother compared to the normal path (b)

The Art of Mutation: Our groundbreaking mutation operation, based on LLMs (Figure 5), revolutionizes the level generation process. Randomly selected slices of levels (ranging from 40 to 80 columns) are transformed using new MarioGPT predictions, guided by randomly generated prompts. While MarioGPT excels at producing levels with diverse agent paths through mutations, path consistency with the rest of the level may be compromised due to its unidirectional nature. To enhance path consistency, we introduce MarioBert, a fine-tuned mask prediction model based on the Bert architecture. Leveraging the strengths of the BERT language model [10] in image in-painting tasks, MarioBert seamlessly fills in the border region after sampling a slice, ensuring a smooth transitionand continuity between the mutated slice and the rest of the level. This innovative combination of MarioGPT and MarioBert in the mutation process adds an extra layer of refinement to the generated levels.

4 Experiments and Results

4.1 Reconstruction and Sampling Quality

We have observed that MarioGPT, utilizing the pretrained GPT2 model, outperforms all other baseline approaches in the context of stack prediction in [1], as demonstrated in Table 3.

To evaluate the ability of MarioGPT to generate playable levels, we need to address the following two questions:

1. Are the levels generated by MarioGPT playable?

Table 3: Training Reconstruction Accuracy – Validation Set

Model	Tile Acc.	Path Acc.	Promptable?
LSTM	46%	39%	NO
from-scratch-MarioGPT	31%	23%	YES
adapter-MarioGPT	21%	11%	YES
MarioGPT	93%	91%	YES

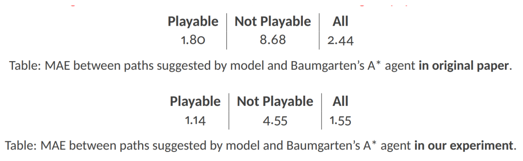
2. Is MarioGPT memorizing dataset instead of learning?

Next, let's examine the results for the two aforementioned questions:

(1) Measuring Playability of Levels

To evaluate the playability of MarioGPT-generated levels, we employed Robin Baumgarten's A* agent, as described in previous research (Khalifa, 2009; Julian Togelius and Baumgarten, 2010). Our study involved testing the agent on 250 generated levels. The results revealed that 83.75% of the levels could be completed by the A* agent, indicating their playability. Remarkably, only one of the successful levels required a retry by the agent. We also compared the paths generated by MarioGPT with those taken by the A* agent to assess their feasibility.

Figure 7 provides insights into the mean absolute error (MAE) between the suggested path generated by MarioGPT and the actual path taken by the agent. The MAE for playable levels was 1.8 tiles, indicating that the paths generated by the model and the paths taken by the agent were relatively close. However, for non-playable levels, the average difference in paths increased significantly to 8.68 tiles. This discrepancy suggests that the paths generated by the model may not be feasible for the agent in these cases.



Playable	Not Playable	All
1.80	8.68	2.44

Table: MAE between paths suggested by model and Baumgarten's A* agent **in original paper**.

Playable	Not Playable	All
1.14	4.55	1.55

Table: MAE between paths suggested by model and Baumgarten's A* agent **in our experiment**.

Figure 7: Mean average error (MAE) between paths suggested by model and Baumgarten's A* agent. Results are averaged over 5 runs per level to account for minor stochastic variation in agent simulation. MAEs are computed between x coordinates of path trajectories for every point on the y axis (which goes across the level) both trajectories have visited.

Considering the overall MAE of 2.44 tiles for paths across all levels, we can conclude that the generated paths by the model generally align with those taken by an actual agent. This demonstrates that incorporating the model's path generation with the level design is an effective approach for producing high-quality levels in terms of playability.

Observing Figures 8, it becomes apparent that paths generated by MarioGPT tend to exhibit more airtime compared to Baumgarten's agent. The model's paths show a weaker consideration for "gravity." This difference may be attributed to the nature of path annotations in the training set used for the model. In the referenced study by Summerville et al. (Adam James Summerville and Ontañón, 2016), an A* path solver was employed to find paths through the level. In contrast, an actual agent, like the one used for comparison in our study, adheres more strictly to game physics, including ”gravity,” and must avoid enemies within the level.

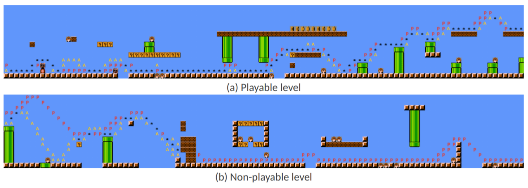


Figure 8: A* agent (denoted as A), and model suggestion. Positions where both trajectories overlap are marked with *

We posit that these issues could be partially attributed to the training data, where paths were derived from a solver rather than an actual agent. Future research could address these concerns by annotating the training data with trajectories obtained from actual agents, potentially mitigating the aforementioned discrepancies.

(2) Is MarioGPT memorizing?

The issue of memorization dynamics in large language models (LLMs), specifically within transformer architectures, remains an intriguing challenge that has captured the attention of researchers. Although LLMs are undeniably powerful, they occasionally succumb to overfitting, resulting in the regurgitation of training data rather than generating novel content. One popular technique to address this concern is the incorporation of randomness through a tunable ”temperature” parameter (Eric Jang and Poole, 2016).

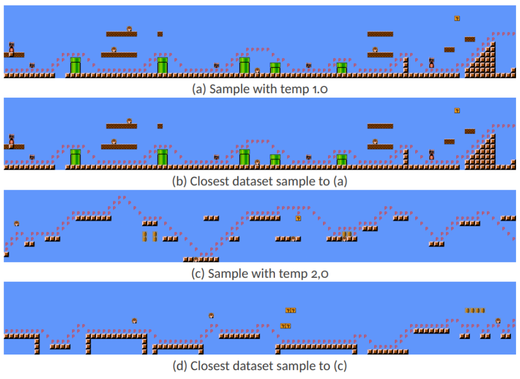
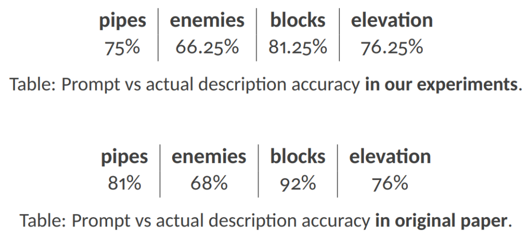

(a) Sample with temp 1.0
(b) Closest dataset sample to (a)
(c) Sample with temp 2.0
(d) Closest dataset sample to (c)

Figure 9: Generated levels vs closest in dataset. Temperature of 1.0 ends up spitting out almost exactly what is in the dataset, while increasing temperature improves sample diversity.

To assess whether MarioGPT falls prey to memorization by generating levels that mirror the training set, we conducted experiments using different temperature parameters. The generated samples were then compared to the most similar levels in the training dataset. Figure 9 showcases the outcomes of this evaluation, revealing that increasing the temperature leads to more diverse but potentially lower-quality samples. In our experiments, we opted for a temperature range of 2.0-2.4 when generating levels. This range strikes a balance between producing diverse samples and ensuring a certain level of quality.

The quest for improvement in combating memorization in LLMs, including MarioGPT, presents numerous exciting opportunities. One approach involves enriching the dataset itself, as a greater variety of samples reduces the likelihood of overfitting. Additionally, enhancing MarioGPT’s sampling capabilities can be achieved by exploring alternative search methods beyond temperature-based sampling. Promising alternatives include constrained beam search and dataset augmented search (Matthew C. Fontaine and Nikolaidis, 2020), both of which aim to amplify diversity while preserving a higher standard of quality during the generation process.

4.2 Guided Level Generation via Prompting



pipes	enemies	blocks	elevation
75%	66.25%	81.25%	76.25%

Table: Prompt vs actual description accuracy **in our experiments**.

pipes	enemies	blocks	elevation
81%	68%	92%	76%

Table: Prompt vs actual description accuracy **in original paper**.

Figure 10: Prompt vs actual description accuracy.

Harnessing the potential of prompts, we are able to steer MarioGPT towards controlled and diverse level generation. To empirically evaluate the prompting ability of MarioGPT, we generated 250 samples using various prompt combinations and assessed how accurately the generated levels matched the provided prompts. The results showcased a high degree of alignment between the generated levels and their corresponding prompts (refer to Figure 10). Notably, MarioGPT exhibited the highest accuracy when generating block-related prompts, while its accuracy was relatively lower when dealing with prompts related to enemies. This disparity can be attributed to the higher abundance of block tiles in the training dataset compared to enemy tiles.

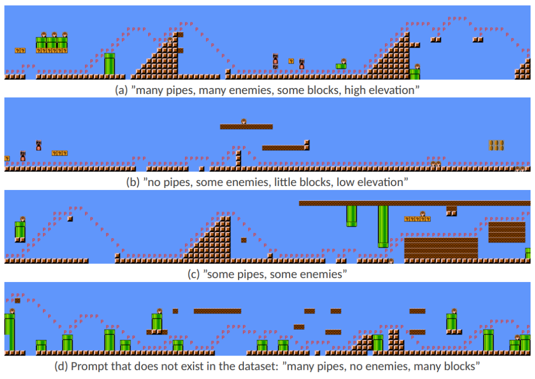

(a) "many pipes, many enemies, some blocks, high elevation"
(b) "no pipes, some enemies, little blocks, low elevation"
(c) "some pipes, some enemies"
(d) Prompt that does not exist in the dataset: "many pipes, no enemies, many blocks"

Figure 11: Multiple prompt generations from a single seed block. For the majority of cases, MarioGPT is able to successfully generate levels that follow the text prompt. Rarely failure cases happen

To further validate the system, we visually examined selected prompt-conditioned generations, as illustrated in Figure 11. Remarkably, MarioGPT demonstrated the ability to generate levels based on text descriptions that were not explicitly represented in the training dataset.

In future research endeavors, we aim to explore additional techniques for incorporating prompt importance. This may involve editing levels with specific tiles to create more diverse samples or exploring prompt tuning strategies (Khalifa, 2009). These approaches hold the potential to further enhance the alignment between prompts and generated levels, fostering even more fine-grained control over the level generation process.

4.3 Generating Diverse Levels with Novelty Search

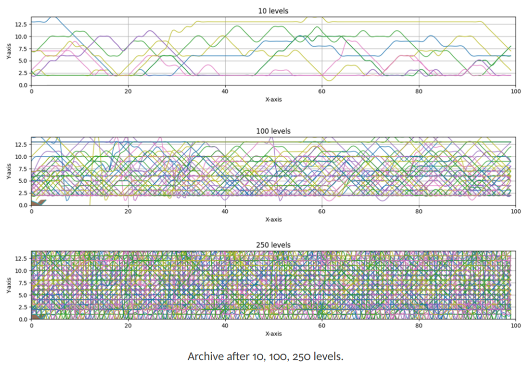


Figure 12: Generated path populations during novelty search. Each line is a path through a level. Over time, NS-MarioGPT fills more and more of the space of all possible paths

By combining a large language model (LLM) as described in Section 3.2, and implementing novelty search as outlined in Section 3.4, we have unlocked the ability to continuously generate diverse levels in an open-ended manner. Specifically, NS-MarioGPT can generate a collection of levels that exhibit a wide range of predicted agent paths.

Figure 12 provides a comprehensive view of the predicted paths overlaid on a level grid as more levels are added to the archive during novelty search. As time progresses, the space of possible predicted agent paths becomes progressively filled, indicating that the addition of increasingly diverse levels through mutation contributes to a richer variety of paths. Notably, as more levels enter the archive, a growing portion of the grid's tiles and empty spaces become occupied, suggesting that NS-MarioGPT is successfully uncovering a wide array of levels that produce diverse paths. However, it is worth noting that there still exist overlapping paths within the archive, implying that similar paths continue to be included. Addressing this issue could involve exploring more sophisticated time series distance metrics that consider path patterns.

Figure 13 showcases the levels with the highest and lowest novelty scores from the archive. The level with the lowest novelty, as depicted in Figure 9 (a), exhibits a path that is highly prevalent in the archive, evident by its striking similarity to the second-lowest level shown in Figure 9 (c). Conversely, the levels with higher novelty, displayed in Figure 9 (b) and Figure 9 (d), demonstrate more unique patterns but share a similar pattern towards the end. This suggests that one

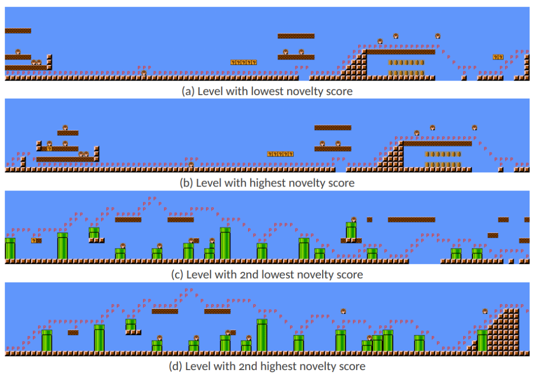

(a) Level with lowest novelty score
(b) Level with highest novelty score
(c) Level with 2nd lowest novelty score
(d) Level with 2nd highest novelty score

Figure 13: Comparison of most and least novel levels in the archive. The two least novel levels are very similar to each other, while the most novel levels have more distinct path patterns.

level was likely created through the mutation of the other.

While NS-MarioGPT successfully discovers a multitude of diverse levels through its simple mutation process, the exploration of more complex functions is worth considering. For instance, incorporating crossover, a common mutation technique employed in genetic algorithms, could increase mutation diversity and potentially result in even greater level diversity.

5 Conclusion

In this study, we explored the capabilities of MarioGPT, a fine-tuned GPT2 language model, in generating diverse levels while being guided by language prompts. MarioGPT proves to be a valuable tool in the field of Procedural Content Generation (PCG), where balancing controllability and diversity is a challenging endeavor. The findings indicate that MarioGPT excels in several aspects: (1) accurately predicting player interactions in generated levels, (2) producing playable and diverse environments, and (3) reducing the reliance on costly external agent interactions, with approximately 88% of the generated levels being playable. Moreover, when coupled with diversity-driven algorithms like novelty search, MarioGPT demonstrates the ability to create open-ended and functional content.

One major advantage of leveraging an existing language model like MarioGPT is the opportunity to benefit from the extensive research and advancements made in this domain. Moving forward, there are plans to capitalize on the scalability of language models by training MarioGPT on larger and more detailed annotated levels. Additionally, the incorporation of human feedback into the level generation process through reinforcement learning from human feedback (RLHF) holds great promise. This approach enables users to continuously refine the characteristics of the generated levels to align with their desired specifications. Ultimately, the aim is for MarioGPT to pave the way for more controllable and diverse PCG systems.

References

Michael Mateas Adam James Summerville, Sam Snodgrass and Santiago Ontañón. 2016. The vglc: The video game level corpus.

Rewon Child David Luan Dario Amodei Alec Radford, Jeff Wu and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. (2019).

Michael Beukman, Christopher W Cleghorn, and Steven James. 2022. Procedural content generation using neuroevolution and novelty search for diverse video game levels. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1028–1037.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Shixiang Gu Eric Jang and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax.

Matthew C Fontaine, Ruilin Liu, Ahmed Khalifa, Jignesh Modi, Julian Togelius, Amy K Hoover, and Stefanos Nikolaidis. 2021. Illuminating mario scenes in the latent space of a generative adversarial network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 5922–5930.

Miguel González-Duque, Rasmus Berg Palm, Søren Hauberg, and Sebastian Risi. 2022. Mario plays on a manifold: Generating functional content in latent space through differential geometry. In *2022 IEEE Conference on Games (CoG)*, pages 385–392. IEEE.

Sadiq HackTech. 2023. Diff models – a new way to edit code.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Sergey Karakovskiy Julian Togelius and Robin Baumgarten. 2010. The 2009 mario ai competition. in ieee congress on evolutionary computation. 1–8.

Joel Lehman Kenneth O. Stanley. 2017. Open-endedness: The last grand challenge you've never heard of.

Ahmed. Khalifa. 2009. The mario ai framework.

Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O. Stanley. 2022. Evolution through large models.

Joel Lehman and Kenneth O Stanley. 2010. Efficiently evolving programs through the search for novelty. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 837–844.

Joel Lehman and Kenneth O Stanley. 2011. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223.

Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. 2015. Constrained novelty search: A study on game content generation. *Evolutionary computation*, 23(1):101–129.

Jialin Liu, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N Yannakakis, and Julian Togelius. 2021. Deep learning for procedural content generation. *Neural Computing and Applications*, 33(1):19–37.

Ahmed Khalifa Jignesh Modi Julian Togelius Amy K. Hoover Matthew C. Fontaine, Ruilin Liu and Stefanos Nikolaidis. 2020. Illuminating mario scenes in the latent space of a generative adversarial network.

Søren Hauberg Miguel González-Duque, Rasmus Berg Palm and Sebastian Risi. 2022. Mario plays on a manifold: Generating functional content in latent space through differential geometry.

Naman Goyal Marjan Ghazvininejad Abdelrahman Mohamed Omer Levy Ves Stoyanov Mike Lewis, Yinhan Liu and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.

J-B Mouret and Stéphane Doncieux. 2012. Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evolutionary computation*, 20(1):91–133.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Rumelhart et al. 1985. Learning internal representations by error propagation.

Jeffrey R Sampson. 1976. Adaptation in natural and artificial systems (john h. holland).

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Shyam Sudhakaran, Miguel González-Duque, Claire Glanois, Matthias Freiberger, Elias Najarro, and Sebastian Risi. 2023. Mariogpt: Open-ended text2level generation through large language models.

Adam Summerville and Michael Mateas. 2016. Super mario as a string: Platformer level generation via lstms.

Adam Summerville and Michael Mateas. 2018. Evolving mario levels in the latent space of a deep convolutional generative adversarial network.

Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. 2011. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Julien Chaumond Victor Sanh, Lysandre Debut and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M Lucas, Adam Smith, and Sebastian Risi. 2018. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the genetic and evolutionary computation conference*, pages 221–228.