Diverse Level Generation via Machine Learning of Quality Diversity

Konstantinos Sfikas
Institute of Digital Games
University of Malta
Msida, Malta
konstantinos.sfikas@um.edu.mt

Antonios Liapis
Institute of Digital Games
University of Malta
Msida, Malta
antonios.liapis@um.edu.mt

Georgios N. Yannakakis
Institute of Digital Games
University of Malta
Msida, Malta
georgios.yannakakis@um.edu.mt

Abstract

Can we replicate the power of evolutionary algorithms in discovering good and diverse game content via generative machine learning (ML) techniques? This question could subvert current trends in procedural content generation (PCG) and beyond. By learning the behavior of quality-diversity (QD) evolutionary algorithms through ML, we stand to overcome the computational challenges inherent in QD search and ensure that the benefits of QD search are reproduced by efficient generative models. We introduce a novel, end-to-end methodology named *Machine Learning of Quality Diversity* (MLQD) which is executed in two steps. First, tailored QD evolution creates large and diverse training datasets from the ground up. Second, sophisticated ML architectures such as the Transformer learn the datasets' underlying distributions, resulting in generative models that can emulate QD search via stochastic inference. We test MLQD on the use-case of generating strategy game map sketches, a task characterized by stringent constraints and a multidimensional feature space. Our findings are promising, demonstrating that the Transformer architecture can capture both the diversity and the quality traits of the training sets, successfully reproducing the behavior of a range of tested QD algorithms. This marks a significant advancement in our quest to automate the creation of high-quality, diverse game content, pushing the boundaries of what is possible in PCG and generative AI at large.

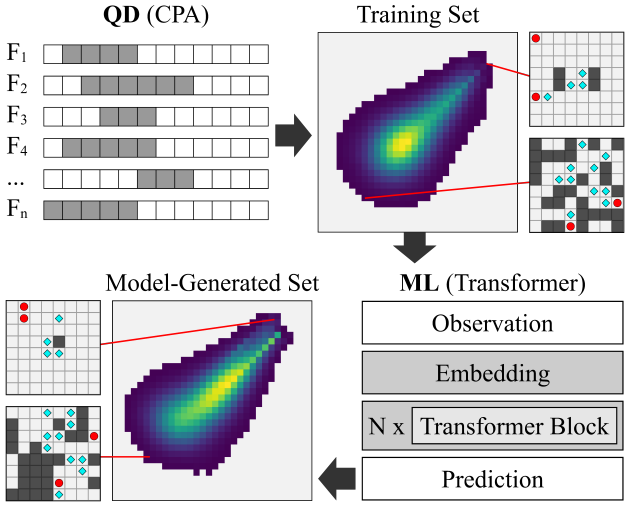

Figure 1: High level representation of MLQD as employed for a level generation task. A QD algorithm (CPA in this example) generates a large and diverse training set of game levels. A generative model (a Transformer in this example) is trained on this set. The model's output resembles the distribution of the training set by machine learning the behavior of QD.

CCS Concepts

• **Applied computing** → **Computer games**; • **Computing methodologies** → **Generative and developmental approaches**; • **Human-centered computing** → *Visualization techniques.*

Keywords

Procedural content generation, machine learning, novelty search, quality diversity, strategy maps

ACM Reference Format:
Konstantinos Sfikas, Antonios Liapis, and Georgios N. Yannakakis. 2025. Diverse Level Generation via Machine Learning of Quality Diversity. In *International Conference on the Foundations of Digital Games (FDG '25), April 15–18, 2025, Graz, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3723498.3723843

FDG '25, Graz, Austria

ACM ISBN /25/04
https://doi.org/10.1145/3723498.3723843

1 Introduction

Procedural Content Generation (PCG) represents a dynamic field [17] within artificial intelligence (AI) and games research. PCG research leverages a diverse array of advanced AI technologies such as machine learning (ML) [30] and evolutionary algorithms [32]. Inspired by research on PCG through quality diversity (PCGQD) [10] and PCG via machine learning (PCGML) [30], we identify an ideal combination of the two that can address the challenges of ML in finding large and diverse game-specific datasets to train on.

The introduced method, *Machine Learning of Quality Diversity* (MLQD), is self-sufficient (see Fig. 1). MLQD operates without training data such as a corpus of real game levels [31], as it can generate as large and controllable a dataset as needed—if provided appropriate genetic operators and quantifiable design metrics. Moreover, the trained generative model can in theory produce diverse content on each call, while evolutionary search may need to traverse a large space of solutions in an inefficient, stochastic manner. In

this paper, we implement MLQD by leveraging a new algorithm for *dataset generation*: Cross-Pollination of Axis-Aligned Archives (CPA) is specifically designed for the rapid generation of large and diverse datasets in high-dimensional feature spaces. For *generative modeling* we train state-of-the-art Transformer [35] models. The goal of this project is to replicate the high quality and diversity of the training sets through ML.

We test MLQD on the challenging task of generating diverse sets of map sketches for strategy games [20]. This use-case is characterized by stringent constraints and a high-dimensional feature space. Within this constrained use-case, we test whether the newly introduced CPA algorithm (adapted for constrained search) can produce good and diverse levels; then, we assess whether Transformers trained on the generated datasets produce levels that are good (i.e. feasible) and diverse (i.e. explore most of the problem space). Finally, we assess whether the distribution of MLQD content matches the distribution of the content it is trained on, and the performance tradeoffs of using ML instead of QD search. Our findings suggest that it is possible to mimic the generative properties of QD algorithms (in this case using constraint satisfaction as quality) through Transformer-based ML models. Our findings pave the way for the advancement of both QD and ML within the context of PCG in games, and beyond.

2 Related Work

MLQD leverages the power and addresses the weaknesses of both QD-based and ML-based PCG paradigms, which we describe below.

2.1 PCG through Quality Diversity

PCGQD [10] tackles the creation of diverse content for games, harnessing a plethora of evolutionary search algorithms around Quality-Diversity (QD) [23]. PCGQD has been applied on many different PCG tasks, powered by many different QD algorithms, as surveyed by [10]. A fundamental strength of QD algorithms lies in their ability to generate diverse sets of solutions, sparsely distributed in a user-controllable behavior space [6]. This capacity is pivotal in PCG, as it can provide designers with a rapid overview of the solution space, while it can enhance content variety and thus replayability for players.

However, applying QD algorithms to PCG has its challenges. First, game content is complex and multi-faceted [18], impeding the operation of QD algorithms which is based on stochastic search guided by (simple and quantifiable) notions of quality and diversity. Moreover, as explained in [34] typical QD algorithms grapple with the "curse of dimensionality," and their application in higher dimensional behavior spaces is restricted by computational bottlenecks.

Significant efforts to alleviate the challenges of QD can be found in the literature [7, 34]. Our approach, however, treats the entire problem of diverse content generation as an ML task. Therefore, here we treat QD algorithms as a stepping stone towards obtaining models that can generate diverse content directly, without the use (or need) of stochastic search.

2.2 PCG via Machine Learning

PCGML studies the generation of game content using machine learning models trained on existing content [30]. As with PCGQD, PCGML has leveraged a plethora of ML algorithms to generate many types of game content, as surveyed by [11]. The main advantage of PCGML is its capacity to efficiently generate new content that nevertheless retains desirable characteristics found in existing data.

A significant hurdle for PCGML is the scarcity of comprehensive datasets [30]. Unlike other ML areas such as image generation and text generation, game content is much more limited in volume, or even inaccessible due to copyright concerns. This could be mitigated by leveraging pre-trained Large Language Models (LLMs), which can be fine-tuned with smaller datasets. While this method is currently explored [27], pre-trained LLMs hardly follow ethical research protocols [14] and moreover raise two concerns for their applicability. First, fine-tuning and inference with LLMs demands substantial computational resources. Second, the availability of such models to the broader public may not continue indefinitely.

In our study, we address the issue of data scarcity through a novel QD approach that efficiently generates large and diverse training sets. We can thus fully exploit the capabilities of the Transformer architecture [35] (the foundational technology behind numerous LLMs), while retaining flexibility regarding the model's parameters and computational resources.

3 Machine Learning of Quality Diversity

In the context of PCGQD and PCGML, we formulate a methodology for replicating the behavior of QD algorithms through efficient ML models, thus extending the reach of QD to demanding use cases such as real-time diverse content generation. To meet this goal, we need (a) access to extensive datasets of diverse content and (b) ML architectures and sufficient computational resources to train the models. Our contributions address (a) efficiently generating a large and diverse dataset through evolutionary QD search, and (b) leveraging state-of-the-art Transformer architectures for learning the patterns of the dataset in an unsupervised manner. We emphasize that MLQD is *self-sufficient* as it does not require access to external datasets, which may be sparse or copyrighted [30], and it does not require pre-trained models which may not (a) be tailored to the problem at hand, (b) be open-source or transparent, (c) be ethically sourced [14]. This does not mean, however, that MLQD does not require clever algorithmic design (in terms of genetic representation and variation operations) and expert knowledge to design appropriate metrics for behavior characterization. The very same benefit of controllable exploration for the designer can become a challenge in cases where designing appropriate metrics is difficult—not unlike how designing a good fitness function can be more expensive than the optimization itself. Approaches of automatically discovering diversity metrics [5] could make MLQD more self-sufficient in that regard; however, in this paper we leverage hand-crafted metrics.

In this paper, we leverage a novel stochastic algorithm for efficient generation of diverse training data (see Section 3.1), and customize a Transformer architecture to generate new tile-based content (see Section 3.2).

3.1 Cross-Pollination of Axis-Aligned Archives

Cross-Pollination of Axis-Aligned Archives (CPA) aims to diversify solutions in complex behavior spaces by mitigating the "curse of dimensionality". We note that CPA is not a QD algorithm on its own:

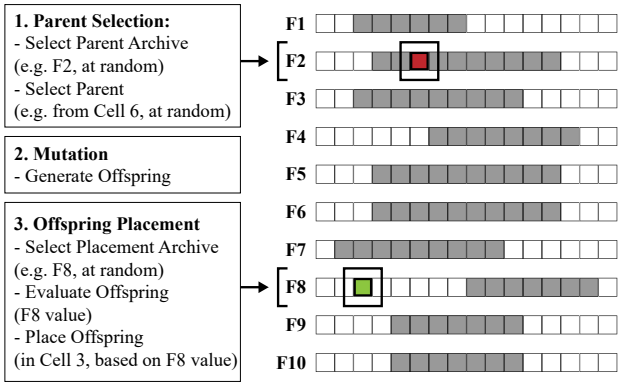

Figure 2: Core algorithmic steps of CPA, in a hypothetical 10-dimensional behavior space (F), where each archive has been subdivided into 16 cells.

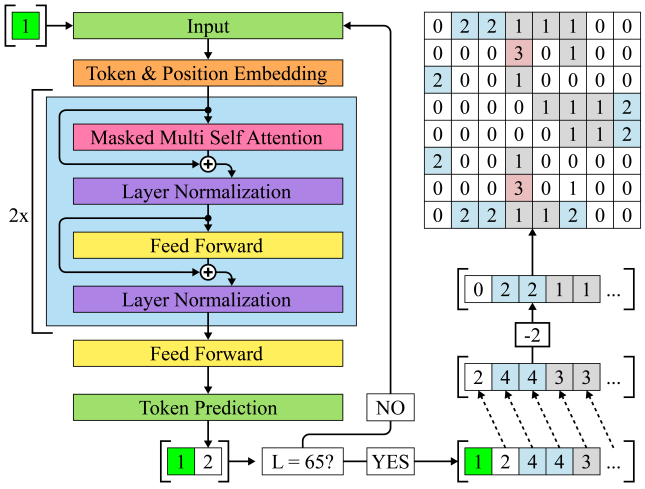

Figure 3: The Transformer architecture, and a visual example of how it generates strategy maps during inference.

its goal is to solely explore a problem space defined by different (quantifiable) design features, similar to novelty search [15]; it can become a QD algorithm by adding constraint satisfaction as proxy of quality (see Section 5.1.1). CPA utilizes N one-dimensional archives, one for each quantifiable design feature of the artifact. Through "cross-pollination," offspring are exchanged between archives, indirectly boosting diversification across the entire feature space. The key benefit of this approach is that generated individuals are only evaluated along a single feature, thus shrinking the computational overhead of high-dimensional feature spaces. Unlike typical evolutionary computation methods that focus on the solutions of the final population, CPA maintains an *evolutionary history record* (EHR) which stores all generated individuals throughout evolution, thus enabling the collection of large datasets without compromising computational efficiency.

The algorithm's operation is visualized in Fig. 2 and summarized below. CPA establishes N one-dimensional archives, one for each feature of the behavior space: those are selected by the designer. Each archive is then divided into discrete cells for specific value ranges of the design feature. During the *initialization* phase of CPA, the following steps are repeated: An individual is generated via random initialization, and is stored in the EHR. Then, a placement archive is selected at random and the individual is evaluated along the feature of that archive. Finally, the individual is placed at the corresponding cell, based on its feature value, always replacing any existing occupant of that cell. The *core operation* phase of CPA starts with randomly selecting an archive and then randomly selecting a parent from within it. The parent produces an offspring through mutation; the offspring is stored in the EHR, and is then placed in a (random) archive following the same approach as in the *initialization* phase.

3.2 Transformer architecture

In this paper we employ the Transformer architecture used in [24, 25] (see Fig. 3), as it performs very well in challenging tasks—given sufficient data and compute. This architecture is primarily aimed at sequence generation, while the artifacts in our use case (see Section 4) are 2D arrays. To address this apparent incompatibility, we follow a common approach in the literature [28, 29] and treat the levels as flattened, 1D arrays during training and inference (see Section 4.2).

For the Transformer, its input layer takes sequences of integers and its output predicts the immediate next token. The first hidden layer is a token and position embedding layer, which projects each input token into an N-dimensional embedding space. The core of the model consists of several stacked Transformer decoder layers. Each decoder layer uses self-attention mechanisms with several attention heads followed by a feed-forward network, allowing the model to weigh the importance of different tokens within the same sequence differently, and improving its ability to understand context and relationships between tokens. Finally, the model's output is processed by a dense layer configured to predict the next token in the sequence by outputting a probability distribution across all possible tokens in the vocabulary.

4 Use-Case: Game Level Generation

Low-resolution tile-based game maps have been a staple testbed for constrained QD [1] and constrained novelty search [21]. In this paper, we use map sketches of 8×8 tiles (see Fig. 4) as the target domain [19], as they are simple enough to generate rapidly, have a multitude of evaluation metrics established in the literature [19], and can be up-scaled to create complex realistic maps suitable for modern games [20]. Moreover, map sketches have been used in several studies [19, 21], allowing us to leverage existing methods for e.g. initialization and variation. While different map sizes have been tested for map sketch generation [21], we focus on the smallest map size explored in past studies (8×8 tiles), as computing pathfinding metrics (see Section 4.1.5) is faster and constraint satisfaction is less challenging than in larger maps.

4.1 Evolutionary Search Process

For experiments in this paper, we compare different evolutionary search methods for generating training sets for ML (see Section 5.1).

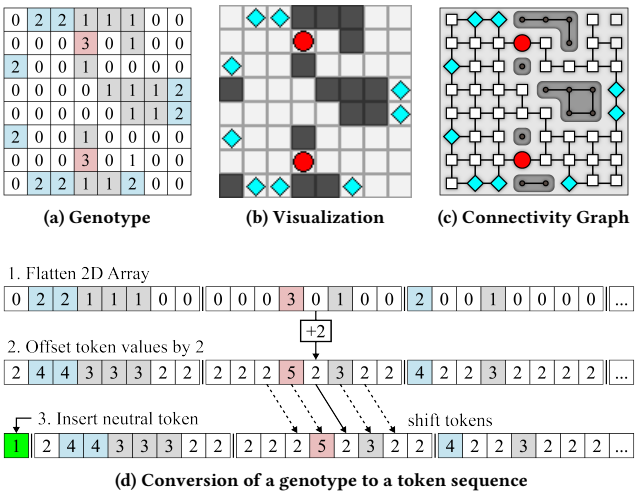

Figure 4: Example of a generated feasible level for our use-case, showing different ways to represent and parse the level. Fig. 4d shows the process of converting a genotype into a token sequence as part of the Transformer's training set.

However, all methods use the following steps for initialization, mutation, and evaluation, building on literature on constrained divergent search [21] for map sketches.

4.1.1 Representation. Map sketches are represented as 2D arrays of integers (Fig. 4) where each integer (0, 1, 2, 3) corresponds to a tile type: Floor, Wall, Resource, and Base, respectively. Walls block movement, while other tiles allow passage. Pathfinding is limited to orthogonal connections between passable tiles, allowing walls to block more paths. Bases and resources are considered "special", due to their functional importance in strategy games.

4.1.2 Constraints. Following [21], we test all maps for the following three constraints: (a) there must be exactly 2 bases; (b) there must be between 4 and 10 resources; (c) all special tiles (bases and resources) must be connected. Maps that fail any constraint are assigned a feasibility score, with higher values indicating that they are closer to being feasible. Since initialization and mutation operators used in our experiments ensure (a) and (b), we assess the feasibility score only in terms of disconnected paths among bases and disconnected paths between bases and resources, as summarized in Eq. (1), where b and r are the number of bases and resources, respectively, while c_b and c_r are the number of connected base pairs and connected base-resource pairs, respectively.

$$F_{inf} = \frac{1}{2}\frac{c_b}{b\cdot(b-1)} + \frac{1}{2}\frac{c_r}{r\cdot b} \tag{1}$$

4.1.3 Initialization. Initialization follows [21], ensuring that the entire initial population consists of feasible solutions. Starting from a map consisting only of floor tiles, 2 random floor tiles are changed to bases, while between 4 and 10 floor tiles are changed to resources. This ensures that the number of special tiles (bases and resources) meet the constraints, while all bases and resources are connected (the third constraint) since there are no impassable wall tiles in these initial maps.

Table 1: 10 custom metrics used for measuring the diversity of the population across all tested algorithms, in addition to the six metrics from [19]. N is the total number of map tiles, while other metrics are described in Section 4.1.5.

Floor Tiles Ratio	$F_{1-3} = c/N$
Walls Ratio	
Resources Ratio	
Horizontal Symmetry	$F_{4-7} = \frac{\sum_{n=1}^{N} Q_{C_n,C'_n}}{N}$
Vertical Symmetry	
Diagonal Symmetry	
Antidiagonal Symmetry	
Wall Islands	$F_8 = 2{\cdot}I_w/N$
Passable Graph Diameter	$F_9 = d_g/(N-1)$
Bases Distance	$F_{10} = d_{b1\to b2}/(N-1)$

4.1.4 Mutation. Evolutionary variation operators are limited to mutation in this work, following the method of [21] which consistently upholds constraints on the number of bases and resources. Mutation is applied to a random portion of the total tiles on the map (between 5% to 20%). Two possible changes are applied to these tiles: any tile may be swapped with an adjacent one, or a Wall or Floor tile may be converted to Floor or Wall respectively.

4.1.5 Behavior Characterization. To highlight the power of our proposed approach, we select as many features as we could identify for these maps in relevant literature [1, 19]. The resulting list includes 16 behavior characterizations bounded within $[0, 1]$, which can be calculated on the map itself, without requiring simulations [33]. We leverage six metrics tailored to strategy game maps from [19] (resource safety, resource safety balance, base safety, base safety balance, exploration, exploration balance as F_{11}-F_{16} in that order), and introduce 10 more tile-based metrics described in Table 1. F_1-F_3 assess tile-type ratios through a shared formula, where c is the number of tiles of the considered type and N the total number of tiles ($N = 64$ in this paper). F_4-F_7 assess symmetries found to be important by human users [20]. In their shared formula, C'_n is the symmetric tile of C_n (vertically, horizontally, diagonally or antidiagonally) and Q_{C_n,C'_n} is 1 if $C_n = C'_n$ or 0 otherwise. F_8 evaluates the ratio of graph-islands (I_w) formed by wall tiles (i.e. connected groups of wall tiles, shown in dark gray in Fig. 4c) over the maximum possible number of graph-islands ($N/2$). F_9 evaluates the ratio of the passable graph diameter d_g, i.e. the maximum value of shortest paths between any tile in the map, over the maximum possible distance ($N-1$). F_{10} evaluates the ratio of the distance between bases ($d_{b1\to b2}$) over the maximum possible distance ($N-1$).

4.2 Machine Learning Process

Based on the general method of Section 3.2, we specify the parameters of the Transformer used for our experiments.

4.2.1 Model parameters. Since we are generating map sketches of 8×8 tiles, we set the maximum input length of the model to 65 to match the total number of map tiles (64) plus the initial neutral

token. The vocabulary size is set to 6, matching the 4 tile types plus the neutral symbol and the mask symbol. The dimensionality of the token and position embedding layer and the size of the feed-forward layer are both set to 256: these values were determined from a grid-search (based on cross-entropy loss on the validation set) to find the optimal values considering options of 4, 16, 64 and 256 for both parameters. The Transformer model has 2 decoder layers and 2 attention heads, after exploring the performance of 1 and 2 for both parameters.

4.2.2 Dataset Pre-processing. As mentioned in Section 3.2, the 2D maps are flattened into 1D arrays during training and inference, to accommodate the Transformer's mode of operation. As shown in Fig. 4d, the genotype (2D array) is first flattened into a 1D array. Then, all tokens are offset by 2, so that values of (2, 3, 4, 5) represent floor, wall, resource and base tiles accordingly. The neutral token of value 1 is inserted at the beginning of the array, shifting all other values to the right, and extending the sequence size to 65. The number 0 is reserved as a masking token during inference.

4.2.3 Training. For a collected dataset of D datapoints, we shuffle them before assigning them to training, validation and test sets. The training set contains 80% of the data, the validation set contains 15% of the data and the test set is only 5% of the data. The validation set is used for early stopping, if 3 consecutive epochs return an increasing validation error. The validation set is also used to select the optimal parameters for the grid-search method described in Section 4.2.1.

4.2.4 Inference. Inference takes place as visualized in Fig. 3: first, we provide the model with the initial token (1) and iteratively let the model predict the next token, adding it to the input sequence for the next prediction. Out of the probabilities provided for the next token by the model, we select one by applying top-p search, as in [12], with a cutoff point of 0.9. As soon as the output sequence is complete (i.e. its length is 65 tokens), it is converted back to its proper form of a 2D array, following the reverse process from Section 4.2.2.

5 Experimental Protocol

In this paper, our experiments explore the impact that the training set has on the MLQD method, specifically regarding the sensitivity of the ML methods on different degrees of diversity in the training data produced from different QD search processes. In order to provide a fair comparison, we use three different QD search approaches to generate datasets of 10^6 maps in their evolutionary history records and repeat the process 10 times in independent runs. The details for the three QD search processes for generating the data are detailed in Section 5.1. The datasets are split into training data ($8{\cdot}10^5$), validation data ($1.5{\cdot}10^5$), and test data ($0.5{\cdot}10^5$) and used to train Transformer-based models to generate new maps. The hypotheses regarding the performance of the models, and metrics for evaluating them, are detailed in Section 5.2.

5.1 QD Methods for Dataset Generation

We test the performance of MLQD algorithms, using different datasets for training, in order to assess the level of diversity required by the Transformer architectures. Since our use case includes hard constraints that would result in unplayable content (see Section 4.1.2), we leverage the feasible-infeasible population paradigm [13] and test three evolutionary search methods for dataset generation. All algorithms' operation has been adjusted to store every *feasible* individual in the evolutionary history record. We let each algorithm run until 10^6 feasible individuals have been collected, and repeat the process 10 times. We note here that we leverage algorithms that optimize *diversity* (CPA, novelty search) and convert them to QD algorithms by adding constraint satisfaction as a minimum criterion for *quality* [10], rather than optimizing quality as a separate objective as in e.g. [16]. All of the below constrained search algorithms follow the two-population approach of [13].

5.1.1 Feasible-Infeasible CPA. Feasible-Infeasible CPA (FI-CPA) uses two groups of archives, separating feasible from infeasible individuals. The feasible archive group includes 16 one-dimensional archives, corresponding to the 16 features detailed in Section 4.1.5. There is a single one-dimensional archive in the infeasible archive group, diversifying infeasible individuals based on their feasibility score F_{inf} of Eq. (1). Each archive is divided into 65 equal bins, evenly spaced within the range $[0, 1]$. The initial population size is set to 1, 105 individuals, equal to the total bin count across all archives. During initialization, generated individuals are evaluated for feasibility; feasible ones are placed in one of the 16 archives of the feasible group based on CPA procedures, while infeasible ones are placed in the infeasible archive. During its operation, FI-CPA alternates between selecting a parent from the feasible or the infeasible group. Offspring of either parent are evaluated for feasibility and placed either in the infeasible archive or a random archive (triggering evaluation for that metric) among the 16 in the feasible group. Feasible offspring are always added to the EHR.

5.1.2 Feasible-Infeasible Novelty Search. The Feasible-Infeasible Novelty Search (FINS) algorithm is a variant of Novelty Search [15] tailored for constrained problems. FINS was used for map sketch evolution [21] and serves as a baseline for constrained diversity search. We introduce minor modifications to the version in [21], by storing all feasible individuals generated in the EHR and by limiting the size of the novel archive. FINS operates on two populations which may vary in size as evolution progresses. The infeasible population selects parents to maximize their feasibility score of Eq. (1), while the feasible population selects parents to maximize the novelty score. The novelty score is calculated as the average distance from the k nearest neighbors in the feasible population and the (feasible) novel archive. The novel archive stores the most novel individuals per generation, and in our case is limited to 3, 000 entries; in our version of FINS, the five most novel feasible individuals in each generation are added to the novel archive, replacing the oldest members when the archive reaches capacity. Following [21], we set $k = 20$ in our experiments. Distance is measured as the Euclidean distance of the 16 metrics detailed in Section 4.1.5.

FINS is population-based; in every run, we seed it with 1, 105 individuals as per the initialization process of Section 4.1.3, which are assigned to either the feasible or the infeasible population. FINS uses minimal elitism, retaining the most novel feasible individual and the least infeasible individual from their respective populations. In each generation, 1, 103 parents are selected (50% from the feasible population via roulette-wheel selection based on their novelty score,

and 50% from the infeasible population based on their feasibility score). The selection process produces 1, 103 offspring. All feasible offspring are stored in the EHR and evaluated on their novelty score. The five most novel offspring are placed in the archive as described above. Infeasible offspring are assigned F_{inf} of Eq. (1). Aside from the elite individuals, offspring replace the previous populations.

5.1.3 FI-Random. We employ a random strategy as our worst-case baseline. FI-Random is a deliberately impaired version of FINS that attempts to maximize feasibility on the infeasible population but performs random search on the feasible population. All components remain the same as in FINS, except for the absence of a novel archive since feasible individuals are selected randomly for mutation, As in all other methods, every feasible offspring is added to the EHR.

5.2 Hypotheses

We aim to identify how the training data impacts the quality and diversity of the generated output of a Transformer model. We expect that more diverse datasets that uniformly distribute their points across ad-hoc feature dimensions, such as those produced by the CPA method, lead to better generative models that match the diversity of the training set. Specifically, we aim to address the following hypotheses:

- H1 **Constraint Satisfaction:** The Transformer model captures feasibility constraints, generating (mostly) feasible solutions.
- H2 **Generalization:** The Transformer model generalizes well, producing unique content that is also not present in the training data.
- H3 **Behavior Space Diversity:** The Transformer model can generate behaviorally diverse output.
- H4 **Behavior Space Distribution Similarity:** Output from the Transformer model matches the (behavioral) distribution of its training set.
- H5 **Efficiency Comparison:** Trained Transformer models are faster to generate solutions than evolutionary search.

Each hypothesis is tested using a combination of quantitative metrics and comparative analyses as follows.

5.2.1 Metrics for H1. We calculate the dataset's *feasibility ratio* as the number of feasible generated solutions over the total number of generated solutions.

5.2.2 Metrics for H2. We capture generalization capacity of the Transformer model through two metrics: *Unseen ratio* as the ratio of solutions not found in the training set, and *Unique ratio* as the ratio of solutions without duplicates in the generated set. Here, we consider exact matches between tested maps for these calculations, as even a small difference in the map may lead to major gameplay imbalances.

5.2.3 Metrics for H3. We assess diversity using two metrics: one captures the extent of the feature space that is explored, and the other captures the dataset's distribution on the feature space. Given that we wish to see how the training data affects the Transformer, we compare the QD-evolved training data ($8 \cdot 10^5$) with the same number of ML-generated samples. To measure the extent of the feature space, we use the notion of a *hypervolume* as the volume of the bounding hypercube of the 16-dimensional feature vector described in Section 4.1.5. For the distribution of the dataset ϕ, we follow [9] and evaluate its *exploration uniformity* $U(\phi)$ as the similarity between the probability distribution of the feature vector set of ϕ and a uniform distribution. It is calculated as $U(\phi) = 1 - \mathrm{J}(\mathbf{P}_\phi, \mathbf{Q})$, where $\mathbf{P}_\phi$ is the estimated distribution of ϕ, $\mathbf{Q}$ the uniform distribution, and J their Jensen-Shannon Divergence [8]. We estimate the density $\mathbf{P}_\phi$ using Kernel Density Estimation [4] with a 0.23 bandwidth (from preliminary testing) and a Gaussian kernel. Given the large dataset size of ϕ and for the sake of computational efficiency, we derive $U(\phi)$ as the average from 1, 000 re-runs of a subset of ϕ consisting of 10^3 datapoints sampled randomly from ϕ.

Table 2: Feasibility ratio (H1) and computation time in minutes (H5) for producing 10^6 feasible maps. Results are averaged from 10 runs, and include 95% confidence interval.

	Dataset (QD)	Transformer (ML)
Feasibility ratio		
FI-CPA	39.6%±0.26%	87.5%±2.05%
FINS	45.0%±0.07%	93.6%±3.24%
FI-Random	44.3%±0.21%	88.9%±4.57%
Computation time		
FI-CPA	37.5±0.38	115.4±2.24
FINS	196.8±1.57	105.0±3.47
FI-Random	25.4±0.68	111.4±5.80

5.2.4 Metrics for H4. We measure the similarity between the distribution of a ML-generated dataset ϕ_{ML} and its QD-evolved training set ϕ_{QD}. We adapt the exploration uniformity of Section 5.2.3 and measure *distribution similarity* $S(\phi_{ML}, \phi_{QD}) = 1 - \mathrm{J}(\mathbf{P}_{\phi_{ML}}, \mathbf{P}_{\phi_{QD}})$, where $\mathbf{P}_{\phi_{ML}}$ is the estimated distribution of ϕ_{ML}, $\mathbf{P}_{\phi_{QD}}$ is the estimated distribution of ϕ_{QD}, and J their Jensen-Shannon Divergence. As above, we measure this $S(\phi_{ML}, \phi_{QD})$ as the average from 1, 000 re-runs of a random subset of ϕ_{ML} and a random subset of ϕ_{QD} with 10^3 datapoints each.

5.2.5 Metrics for H5. We measure computational efficiency based on the time each method takes to generate 10^6 feasible individuals. Computation time is measured on a desktop computer equipped with an I9-12900K processor, 64GB of DDR5 RAM and an RTX3070 GPU (8GB GDDR6 RAM).

6 Results

We report our findings based on the hypotheses tested (see Section 5.2). Statistical significance, where reported, is established at 95% confidence ($p < 0.05$). When reporting significant differences across multiple comparisons, the Bonferroni correction is applied [3].

6.1 Testing Constraint Satisfaction

Table 2 reports the feasibility ratio of the QD-evolved and the ML-generated data. It is important to note that for the QD datasets, we measure the number of total individuals generated by the time the EHR reached 10^6 feasible solutions. For the ML-generated data, we follow a similar approach and generate content until 10^6 feasible solutions are generated, at which point we measure how many individuals were generated at that point. Unsurprisingly, Table 2

Table 3: Ratio of unseen and unique maps in ML-generated output (H2), and distribution similarity with the training set (H4). Results are calculated on the first $8 \cdot 10^5$ feasible ML-generated individuals, to match the size of the training set. Results are averaged from 10 runs, and include the 95% confidence interval.

Transformer	Unseen Ratio	Unique Ratio	$S(\phi_{ML}, \phi_{QD})$
FI-CPA	99.1% ± 0.21%	98.00% ± 0.43%	0.84 ± 0.02
FINS	100.0% ± 0.00%	100.00% ± 0.00%	0.82 ± 0.04
FI-Random	100.0% ± 0.00%	99.99% ± 0.01%	0.89 ± 0.04

shows that the constrained problem of map generation poses a challenge to evolutionary search, and only around 40% of offspring are feasible. All FI-based evolutionary algorithms, however, tend to tackle the constrained problem at a similar level. CPA yields a significantly lower feasibility ratio than the other methods, since both FINS and FI-Random apply selection pressure based on feasibility in their infeasible populations. On the other hand, since the Transformer is only trained on feasible individuals, its feasibility ratio is much higher than for QD search. Again, differences are not that pronounced depending on the training set, although the Transformer trained on FINS-generated data has a significantly higher feasibility ratio than that trained on FI-CPA. We observe that overall the likelihood of generating an infeasible individual by the Transformer is less than 15% (and less than 7% at the best case). This means that the model may yet produce a level that is unplayable (e.g. has only one base, or bases are not reachable), which could be frustrating if shown to a human player. Since the likelihood of this occurring is non-zero, it would be important to test for constraint satisfaction of ML-generated content before showing it to a human user, and re- generate-and-test [32] if the content is infeasible. Based on the above, H1 is validated as the ML-generated content is *mostly* playable.

6.2 Testing Generalization

Table 3 captures how the model generalizes in terms of creating content unseen in the training set and unique (without duplicates in the ML-generated set). Transformers trained on FINS and FI-Random always produce unseen maps across all runs, and for FINS-based datasets even always produce unique solutions (among a sample of $8 \cdot 10^5$ maps). The FI-CPA datasets lead Transformers to sometimes produce data in the training set (1% chance) or data that the player may have seen already (2% chance). This likelihood is very low, but non-zero, likely because the more niche maps in the FI-CPA dataset are on the borders of the feasible space (see Section 6.3) where only few maps satisfy the constraints. Trying to match the diversity of the training set, the Transformer trained on FI-CPA may revisit past maps. Despite the small chance of non-unique maps, the findings validate H2: all Transformers show generalization capabilities, and can generate both unseen and unique content with a very high likelihood.

Table 4: Behavior space diversity (H3) of training sets and trained models' output. The first $8 \cdot 10^5$ feasible ML-generated maps are compared to the training sets, to match their size. Results are averaged from 10 runs, and include the 95% confidence interval.

	Training Set (QD)	Transformer (ML)
Hypervolume ($\times 10^3$)		
FI-CPA	2.8 ± 0.3	2.3 ± 0.3
FINS	1.0 ± 0.0	0.9 ± 0.1
FI-Random	0.8 ± 0.0	0.7 ± 0.1
Exploration uniformity: $U(\phi)$		
FI-CPA	0.292 ± 0.0006	0.282 ± 0.0029
FINS	0.279 ± 0.0004	0.263 ± 0.0044
FI-Random	0.277 ± 0.0004	0.269 ± 0.0038

6.3 Testing Behavior Space Diversity

Table 4 compares the behavior space diversity metrics (see Section 5.2.3) of the training datasets evolved via QD variants and the content generated by Transformers trained on each dataset. Here, we see the power of FI-CPA as its EHR datasets have a significantly higher hypervolume and exploration uniformity than both FINS and FI-Random. This means that FI-CPA explores more of the edges of the feasible space, and also produces maps that are more evenly distributed along the 16-dimensional feature vector used to partition the space. We see that both diversity metrics drop in the ML-generated dataset compared to the training set, with significant differences between model-generated set and training set (a relative drop as high as 17% for hypervolume, on FI-CPA, and as high as 5.5% for $U(\phi)$, on FINS). However, the diversity of the Transformer trained on the more diverse training set (FI-CPA) is still significantly better than respective ones using FINS and FI-Random. Moreover, targeted search by FINS does have an effect, as the Transformer trained on FINS has significantly higher hypervolume than the Transformer trained on FI-Random. Considering all 30 training sets and 30 ML-generated sets (from 10 independent runs), there is a significant correlation ($\rho = 0.956$ for hypervolume and $\rho = 0.737$ for $U(\phi)$) between the training set's and the ML-generated outputs' behavior space diversity, validating H3.

6.4 Testing Behavior Space Distribution Similarity

Table 3 includes the distribution similarity between the QD training set and the ML-generated set, calculated as $S(\phi_{ML}, \phi_{QD})$ in Section 5.2.4. High values of $S(\phi_{ML}, \phi_{QD})$ mean that the distribution of the ML-generated set matches the training set; we observe that this is true regardless of the evolutionary algorithm used to produce the training set. There are no significant differences in $S(\phi_{ML}, \phi_{QD})$ between QD dataset generation methods. However, we note that this finding strongly favors FI-CPA, as its training sets are significantly more diverse than other methods (see Section 6.3). Thus, ML-generated datasets that match this distribution generate maps along all axes of exploration. By comparison, even if $S(\phi_{ML}, \phi_{QD})$ is high for FI-Random, the hypervolume of its training set is very small and thus the ML-generated content will not be particularly

diverse. That said, H4 is validated: regardless of training set, distribution similarity remains high.

6.5 Testing Efficiency

Table 2 shows the compute time for reaching 10^6 feasible results, either via evolutionary search or via inference on the Transformer. In terms of evolutionary search, we see that FI-Random is significantly faster than other methods, as expected since it only performs constraint satisfaction checks but no evaluations on feasible maps. FI-CPA is fairy fast, as it performs only one feasibility check and one feature evaluation per feasible individual, while FINS is very slow as it has to measure 16-dimensional distances from all individuals in the feasible population. While on the one hand evolution is performing very fast with FI-CPA while generating very diverse results (see Section 6.3), Transformers seem to be surprisingly slow. On average, it takes 6 milliseconds to generate a feasible map with a Transformer (regardless of training set). This is not a major time delay, especially considering that most generated maps will be feasible on the first try (see Section 6.1), but we expected Transformers to be faster than the training set generation process. For a real-time interface, this is likely an acceptable delay for a player or a designer. However, it is difficult to argue that H5 is validated in terms of the efficiency boost of the Transformer compared to the search-based QD approach. We revisit this limitation in Section 7.

6.6 Qualitative Results

Table 5 shows representative results generated by a single run of the FI-CPA algorithm, alongside those produced by the Transformer model trained on its output. The selected results correspond to the extreme (minimum or maximum) values of each behavior characterization ($F_1 - F_{16}$).

A key observation is that both FI-CPA and its trained Transformer model successfully identify solutions that reach the extreme values of most metrics. This is particularly noteworthy for metrics with complex computations, such as those in $F_{11} - F_{16}$. Overall, the Transformer's extreme values per feature closely match those in its training set. Interestingly, in some cases, the Transformer surpasses its training set, achieving even more extreme values: specifically for $\max F_2$, $\max F_8$, $\min F_1$, $\min F_5$, and $\min F_{15}$. These results suggest that the Transformer occasionally ventures slightly beyond the behavior space hypervolume of its training data.

Additionally, in certain cases, the Transformer's outputs are nearly identical to their respective training samples. This is particularly evident in $\min F_1$, $\max F_5$, and $\min F_9$, where the Transformer's generated samples, while technically different from their corresponding ones in the training set (i.e., not all tiles match), are visually and functionally almost indistinguishable. This phenomenon may stem from the difficulty of finding genotypically diverse solutions at the selected extremes. In other words, there may be fewer solutions that exist in those regions, and therefore it is harder to establish diversity. Conversely, in scenarios where a similar pattern could be intuitively expected — such as the minimum values of symmetries — the Transformer's outputs differ significantly from the training set, suggesting a greater genotypic variability.

In any case, this study's focus on behavioral diversity means that genotypic variability was only examined in a simplistic manner (see Section 5.2.2). However, the qualitative results showcase that a fine-grained perspective on genotypic diversity may be worth pursuing in parallel to behavioral diversity in future work.

7 Discussion

This paper introduced MLQD as a way of generating large, diverse, and feasible datasets via QD evolutionary search for training powerful ML models that can generate diverse content on demand via stochastic inference. The use case showed how a constrained version of the proposed CPA diversification mechanism can generate large datasets for training by collecting generated content throughout the evolutionary process. Tests on the models' performance for different training sets showed that the chosen Transformer architecture can generate (mostly) feasible maps that are unique and novel. We also note the impact of the training set: more diverse training sets lead to more diverse generators, as the Transformers universally match the distribution of the data (in terms of ad-hoc design features) they are trained on. Finally, the Transformer produces a feasible individual within 6 milliseconds on average: given the small size of the content (a map of 64 tiles), this is not particularly fast but could likely be improved on a machine that is more tailored for GPU calculations than the one used in these experiments.

We note that the inference times was an unexpected finding and a limitation of our approach. When conducting this research we assumed that stochastic generation via Transformers would be much more computationally efficient than e.g. evolution. While CPA is explicitly designed to be computationally efficient, we expected Transformers to be even more so. It is worth noting that the ML model generates good and diverse content nearly every time (at 6 milliseconds), while evolution may need extensive compute before producing something complex enough for use due to the search process. We also note that the experiment uses a simple artifact (a map of 64 tiles) due to the extensive heuristics and evolutionary operators found in the literature [21] that made the experiment design faster. However, we should explore how MLQD operates (especially in terms of inference time) on more complex game content such as larger and more complex maps, or in generated rulesets that would require simulation-based evaluations [32] when evolving a training set. Other directions for future work could assess the impact the generative model has on MLQD, exploring different Transformer architectures or other ML algorithms, including "inexpensive" language-based models as in [22].

Finally, while MLQD aims to train stochastic generators of diverse content—largely addressing issues with ML models producing "generic" output—there is more potential for ML as a generator trained on evolutionary search processes. A generative model trained on parent-offspring pairs could learn to predict an offspring from its parent guided by changes in terms of feasibility or feature values as ground truth discovered through the stochastic search of a QD algorithm. Such an ML-controlled variation operator could be used in tandem with an evolutionary process or as a controllable variation operator, e.g. for mixed-initiative level design tools to modify a user's creation [20] towards a user-defined goal.

While we argue that the main strength of MLQD is that it is *self-sufficient*, critics can argue that this statement overlooks the human effort of designing appropriate diversity metrics. Admittedly,

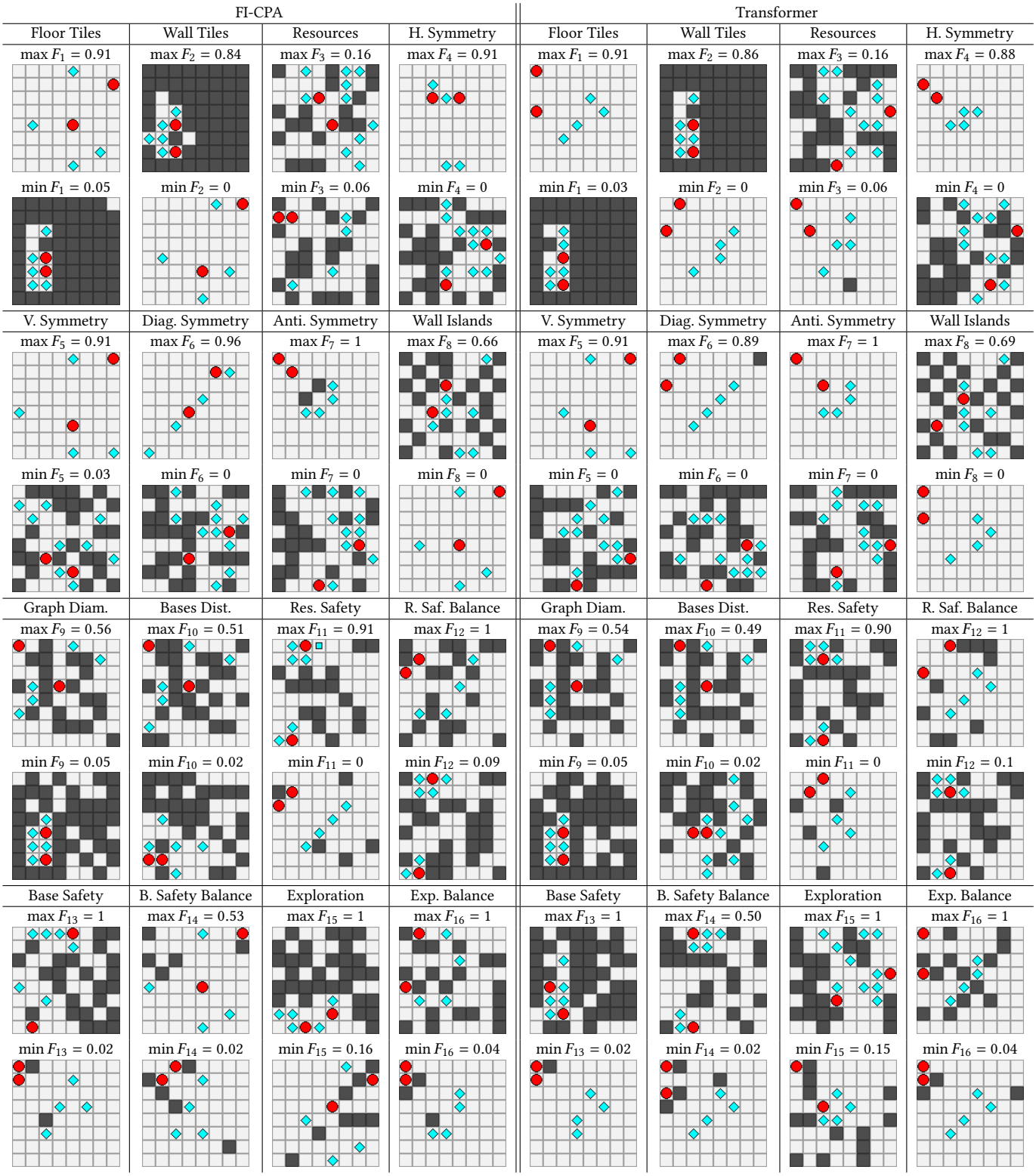

Table 5: Indicative samples from the FI-CPA algorithm's first run (left) and the Transformer model trained on the output of that run (right). Each sample represents an extreme (minimum or maximum) value of a behavior characterization ($F_1 - F_{16}$) from its respective dataset.

one of the reasons for using a well-studied level generation problem [21] as a use-case is because the literature already provided behavior characterizations [19] and a straightforward initialization and variation process [21] for this problem. While MLQD does require human expert knowledge and intuition, arguably all PCG research hinges on human expertise: from devising generative scripts for constructive approaches, to collecting a corpus of levels for PCGML, to designing genetic operators and fitness functions for search-based PCG [26]. Future work, however, could explore how MLQD can operate more freely: e.g. with more freeform representations, such as game descriptions in natural language and evolved via LLM-based genetic operators [2], and with automatically defined behavior characterizations e.g. via dimensionality reduction methods as in [5].

8 Conclusion

This paper introduces the notion of MLQD: a self-sufficient way of building a ML generative model capable of producing feasible and diverse content on demand. The MLQD pipeline is validated using a high-performing Transformer architecture combined with an efficient way of producing diverse content via Cross-Pollination of Axis-Aligned Archives. Results on an established testbed for QD search (as constrained diversity) indicate that the Transformer can capture the diversity of the training set and—provided a large enough dataset—can generate feasible, novel, and unique content on demand. Future work should validate these findings in a more complex domain, with more computationally heavy evaluations, and with different ML architectures.

Acknowledgments

The authors would like to thank Daniele Gravina for early discussions and insights on the QD problem. This project has received funding from the European Union's Horizon Europe programme under grant agreement No 101070524.

References

[1] Alberto Alvarez, Steve Dahlskog, Jose Font, and Julian Togelius. 2019. Empowering quality diversity in dungeon design with interactive constrained MAP-Elites. In *Proceedings of the IEEE Conference on Games*.

[2] Jill Baumann and Oliver Kramer. 2024. Evolutionary Multi-Objective Optimization of Large Language Model Prompts for Balancing Sentiments. In *Applications of Evolutionary Computation*. Vol. 14635, LNCS. Springer.

[3] J Martin Bland and Douglas G Altman. 1995. Multiple significance tests: the Bonferroni method. *BMJ* 310, 6973 (1995), 170.

[4] Yen-Chi Chen. 2017. A tutorial on kernel density estimation and recent advances. *Biostatistics & Epidemiology* 1, 1 (2017), 161–187.

[5] Antoine Cully. 2019. Autonomous skill discovery with quality-diversity and unsupervised descriptors. In *Proceedings of the Genetic and Evolutionary Computation Conference*.

[6] Stephane Doncieux, Alban Laflaquière, and Alexandre Coninx. 2019. Novelty search: a theoretical perspective. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 99–106.

[7] Matthew C. Fontaine, Julian Togelius, Stefanos Nikolaidis, and Amy K. Hoover. 2020. Covariance matrix adaptation for the rapid illumination of behavior space. In *Proceedings of the Genetic and Evolutionary Computation Conference*.

[8] Bent Fuglede and Flemming Topsoe. 2004. Jensen-Shannon divergence and Hilbert space embedding. In *Proceedings of the International Symposium on Information Theory*. IEEE, 31.

[9] Jorge Gomes, Pedro Mariano, and Anders Lyhne Christensen. 2015. Devising effective novelty search algorithms: A comprehensive empirical study. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 943–950.

[10] Daniele Gravina, Ahmed Khalifa, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. 2019. Procedural Content Generation through Quality-Diversity. In *Proceedings of IEEE Conference on Games*.

[11] Matthew Guzdial, Sam Snodgrass, and Adam J. Summerville. 2022. *Procedural Content Generation via Machine Learning*. Springer.

[12] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751* (2019).

[13] Steven Orla Kimbrough, Gary J Koehler, Ming Lu, and David Harlan Wood. 2008. On a feasible-infeasible two-population (FI-2pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch. *European Journal of Operational Research* 190, 2 (2008), 310–327.

[14] Carolyn E. Lamb and Daniel G. Brown. 2023. Should we have seen the coming storm? Transformers, society, and CC. In *Proceedings of the International Conference on Computational Creativity*.

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

[16] Joel Lehman and Kenneth O Stanley. 2011. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 211–218.

[17] Antonios Liapis. 2020. 10 Years of the PCG workshop: Past and Future Trends. In *Proceedings of the FDG Workshop on Procedural Content Generation*.

[18] Antonios Liapis, Georgios N. Yannakakis, Mark J. Nelson, Mike Preuss, and Rafael Bidarra. 2019. Orchestrating Game Generation. *IEEE Transactions on Games* 11, 1 (2019), 48–68.

[19] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. 2013. Generating Map Sketches for Strategy Games. In *Proceedings of Applications of Evolutionary Computation*. Vol. 7835, LNCS. Springer, 264–273.

[20] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. 2013. Sentient Sketchbook: Computer-Aided Game Level Authoring. In *Proceedings of the 8th Conference on the Foundations of Digital Games*. 213–220.

[21] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. 2015. Constrained Novelty Search: A Study on Game Content Generation. *Evolutionary Computation* 23, 1 (2015), 101–129.

[22] Timothy Merino, Roman Negri, Dipika Rajesh, M Charity, and Julian Togelius. 2023. The five-dollar model: generating game maps and sprites from sentence embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

[23] Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. 2016. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI* 3 (2016).

[24] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. https://openai.com/research/language-unsupervised. Accessed 1 March 2024.

[25] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. https://openai.com/index/better-language-models/. Accessed 1 March 2025.

[26] Noor Shaker, Julian Togelius, and Mark J. Nelson. 2016. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer.

[27] Shyam Sudhakaran, Miguel González-Duque, Claire Glanois, Matthias Freiberger, Elias Najarro, and Sebastian Risi. 2023. Prompt-Guided Level Generation. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*.

[28] Shyam Sudhakaran, Miguel González-Duque, Claire Glanois, Matthias Freiberger, Elias Najarro, and Sebastian Risi. 2023. MarioGPT: Open-Ended Text2Level Generation through Large Language Models. In *Proceedings of the Neural Information Processing Systems Conference*.

[29] Adam Summerville and Michael Mateas. 2016. Super Mario as a string: Platformer level generation via LSTMs. In *Proceedings of 1st International Joint Conference of DiGRA and FDG*.

[30] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. 2018. Procedural content generation via machine learning (PCGML). *IEEE Transactions on Games* 10, 3 (2018), 257–270.

[31] Adam Summerville, Sam Snodgrass, Michael Mateas, and Santiago Ontanon. 2016. The VGLC: The Video Game Level Corpus. In *Proceedings of the FDG workshop on Procedural Content Generation*.

[32] Julian Togelius, Georgios Yannakakis, Kenneth 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 (2011).

[33] Alberto Uriarte and Santiago Ontanon. 2014. Game-Tree Search over High-Level Game States in RTS Games. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference*.

[34] Vassilis Vassiliades, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. 2018. Using Centroidal Voronoi Tessellations to Scale Up the Multidimensional Archive of Phenotypic Elites Algorithm. *IEEE Transactions on Evolutionary Computation* 22, 4 (2018), 623–630.

[35] 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 (2017).