FEDERAL UNIVERSITY OF SANTA CATARINA
TECHNOLOGICAL CENTER
DEPARTMENT OF INFORMATICS AND STATISTICS
COMPUTER SCIENCE COURSE

Luiz Gabriel Slongo

Procedural Generation of Game Maps as a Constraint Satisfaction Problem: An Euler-based Conditional Flow Matching Approach

Florianópolis
2026

Luiz Gabriel Slongo

Procedural Generation of Game Maps as a Constraint Satisfaction Problem: An Euler-based Conditional Flow Matching Approach

Undergraduate Thesis submitted to the Computer Science Program at the Technological Center of the Federal University of Santa Catarina in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.
Advisor: Prof. Elder Rizzon Santos, Ph.D.

Florianópolis
2026

ABSTRACT

This work investigates the application of Euler-based Conditional Flow Matching (CFM) to the procedural generation of tile-based 2D game maps, framed as a Constraint Satisfaction Problem (CSP) in which generated maps must satisfy implicit local and global structural rules learned from a training corpus. A generative pipeline is implemented from first principles in C# with TorchSharp: a two-level U-Net parameterizes a time-dependent vector field trained via the Conditional Flow Matching objective, and inference proceeds through an Euler ordinary differential equation solver at fifty function evaluations per sample. The Video Game Level Corpus Super Mario Bros distribution provides fifteen levels across three biomes (Overworld, Underground, Treetop) as the training set. Development proceeded across four iterations, each targeting the dominant failure mode observed in the previous iteration's output as measured by a six-category failure-mode analyzer. Iteration 1 established the baseline pipeline; Iteration 2 explored class-balanced loss variants and biome conditioning; Iteration 3 tested capacity-and-duration scaling and produced a negative result identifying a fundamental quality ceiling of the CFM plus argmax decoding formulation; Iteration 4 introduced a deterministic post-generation repair pipeline of eleven rule-based primitives. Across thirty generated chunks, the hybrid pipeline reduced three hundred and nine unintended structural violations to zero, with sixty-five intentional Treetop ground gaps preserved as a biome feature.

Keywords: Procedural Content Generation; Conditional Flow Matching; Constraint Satisfaction Problem; Hybrid Neural-Symbolic Systems; Super Mario Bros.

RESUMO

Este trabalho investiga a aplicação de Conditional Flow Matching (CFM) baseado em Euler para a geração procedural de mapas de jogos 2D baseados em tiles, formulada como um Problema de Satisfação de Restrições (CSP) no qual os mapas gerados devem satisfazer regras estruturais locais e globais implícitas, aprendidas a partir de um corpus de treinamento. Uma pipeline generativa é implementada do zero em C# com TorchSharp: uma U-Net de dois níveis parametriza um campo vetorial dependente do tempo, treinado com o objetivo Conditional Flow Matching, e a inferência procede através de um solver de Equações Diferenciais Ordinárias (Euler) com cinquenta avaliações de função por amostra. O corpus Video Game Level Corpus (VGLC), distribuição Super Mario Bros, fornece quinze níveis distribuídos em três biomas (Overworld, Underground, Treetop) como conjunto de treinamento. O desenvolvimento procedeu em quatro iterações, cada uma direcionada à falha dominante observada na saída da iteração anterior, medida por um analisador de modos de falha de seis categorias. A Iteração 1 estabeleceu a pipeline básica; a Iteração 2 explorou variantes de perda class-balanced e condicionamento por bioma; a Iteração 3 testou escalonamento de capacidade e duração, produzindo um resultado negativo que identifica um teto de qualidade fundamental na formulação CFM mais decodificação por argmax; a Iteração 4 introduziu uma pipeline determinística de reparo pós-geração composta por onze primitivas baseadas em regras. Em trinta chunks gerados, a pipeline híbrida reduziu trezentas e nove violações estruturais não-intencionais a zero, preservando sessenta e cinco lacunas intencionais no bioma Treetop como característica de bioma.

Palavras-chave: Geração Procedural de Conteúdo; Conditional Flow Matching; Problema de Satisfação de Restrições; Sistemas Híbridos Neuro-Simbólicos; Super Mario Bros.

CONTENTS

1 INTRODUCTION

Video games have grown into a multi-billion dollar industry, resulting in the birth of gigantic and complex virtual worlds. One of the core technologies that enable this expansion is Procedural Content Generation (PCG), which consists of computational methods for creating game content algorithmically instead of manually. PCG is essential for generating vast environments, ensuring replayability, and reducing the immense cost and time of manual asset creation (HENDRIKX *et al.*, 2013). Among the most popular applications of PCG in this context is the generation of maps, being present in genres ranging from classic open-world survival games like Minecraft, to Sci-Fi games like No Man's Sky.

The procedural generation of a valid game map can be formally defined as a Constraint Satisfaction Problem (CSP) (HENDRIKX *et al.*, 2013). A CSP consists of reaching a state where all constraints are satisfied, making it part of the class of NP-Hard problems (HENDRIKX *et al.*, 2013). That means that as the size of the map and the number and complexity of the constraints increase, the time required to find a guaranteed solution can grow exponentially (RUSSELL, NORVIG, 2022). In the context of 2D tile-based map generation, these constraints operate on two levels:

- Local Constraints: Rules concerning the adjacency of individual tiles. For example, a water tile must be adjacent to a sand or grass tile, but never to a lava tile;
- Global Constraints: Rules that ensure the overall coherence of the map, such as the formation of distinct biomes (forests, deserts, oceans), the connectivity of paths, or the logical placement of key locations.

In face of the possibility of the general CSP becoming computationally impossible, this work finds motivation to search for smarter algorithms. Traditional approaches to solving this CSP, such as the Wave Function Collapse (WFC) algorithm, have proven effective but often rely on explicitly defined rulesets derived from example maps (GUMIN, 2016). While powerful, these methods' need to strictly follow the rulesets hinders their ability to capture bigger-picture design ideas or subtle structural patterns.

In recent years, Deep Generative Models have emerged as a ground-breaking technology, due to their capability of learning complex data distributions from examples. These models, such as Diffusion Probabilistic Models (DDPMs) and Generative Adversarial Networks (GANs),

learn to synthesize new data that mimics the training set (GOODFELLOW *et al.*, 2014; HO, JAIN, ABBEEL, 2020). By training these models on a dataset of existing valid maps, they can learn both the local and global constraints implicitly, without the need for manual rule definition.

While DDPMs have shown great promise, they suffer from a significant drawback: a slow and computationally expensive sampling process (generating a new map, starting from random noise and denoising it step-by-step) that requires hundreds or thousands of iterative steps (HO, JAIN, ABBEEL, 2020). In contrast, this work turns to a more recent and more efficient alternative: Euler-based Flow Matching (LIPMAN *et al.*, 2023). Flow Matching models learn a vector field (a function that determines the direction to move in each point in space) that defines a direct and deterministic path from a simple noise distribution all the way to a valid map (LIPMAN *et al*., 2023). This is analogous to learning the "flow" that guides random particles into a coherent structure (LIPMAN *et al.*, 2023). The "Euler-based" approach refers to the use of the Euler method, a numerical technique used to solve ordinary differential equations (ODEs), to traverse this path, allowing for the generation of high-quality samples in very few steps (LIPMAN *et al.*, 2023).

Therefore, this work proposes and investigates the application of Euler-based Flow Matching as an efficient approach for solving the game map generation CSP. By leveraging its ability to learn how to perform a direct transformation from noise to valid data, we hypothesize that this approach can generate diverse, coherent, and aesthetically plausible maps while drastically reducing the computational overhead associated with other deep generative models.

1.1 OBJECTIVES

This work’s objectives are divided into general and specific objectives.

1.1.1 General Objective

The general objective of this work is to develop, train, and evaluate an Euler-based Flow Matching model for the procedural generation of coherent and diverse maps, framing the task as an implicit solution to a Constraint Satisfaction Problem.

1.1.2 Specific Objectives

The specific objectives are:

- **Objective O1.** To conduct a comprehensive literature review of Procedural Content Generation (PCG), establishing the evolution from classic constraint-based methods (like WFC) to modern deep generative models (like DDPMs and Flow Matching);
- **Objective O2.** To develop a data engineering pipeline to extract, process and convert 2D maps into an adequate format for training, including the division of maps in chunks and the application of one-hot encoding to represent tile geometry and material.
- **Objective O3.** To implement and train an Euler-based Flow Matching model capable of learning the implicit local and global constraints present in the prepared map dataset and generating novel, coherent maps;
- **Objective O4.** To create a control mechanism for the procedural generation, utilizing conditional vectors to guide the model on creating maps with specific characteristics (e.g. presence of lava, biome type, etc.).
- **Objective O5.** To evaluate the implemented Flow Matching model's performance and quality through:
    a) A quantitative measurement of constraint satisfaction via a failure-mode analyzer over generated chunks.
    b) A literature-based positioning against Wave Function Collapse, Denoising Diffusion Probabilistic Models (DDPMs), and Generative Adversarial Networks (GANs) on the dimensions of generation speed, structural fidelity, and output diversity, with an empirical WFC comparison identified as future work.

1.2 FORMAL PROBLEM DEFINITION

This section formalizes the procedural map generation problem as a Constraint Satisfaction Problem, following the framing established by Hendrikx *et al.* (2013) and using the classical CSP tuple representation introduced in Russell and Norvig (2022). A 2D tile-based game map of dimensions H×W is represented as a discrete grid X = {x_{i,j}} for i ∈ {1,...,H}, j

$\in$ {1,...,W}, where each cell x_{i,j}takes a value from a finite tile vocabulary T = {t_1, t_2, ..., t_K}. In this work, K=14, and T is the vocabulary defined in Table 4.1.

The CSP is defined by the tuple (V, D, C) (RUSSELL, NORVIG, 2022):

• Variables V = {v_{i,j} | i $\in$ {1,...,H}, j $\in$ {1,...,W}}: one variable per grid cell.

• Domains D: each v_{i,j} takes a value from T. Domain size |T| = 14.

• Constraints C = C_local $\cup$ C_global:

- C_local: constraints defined over pairs or small neighborhoods of adjacent cells. Example: if v_{i,j} = PipeTopLeft then v_{i,j+1} must be PipeTopRight and v_{i+1,j} must be PipeBodyLeft.

- C_global: constraints defined over the full grid. Example: the bottom row must form a continuous ground surface (biome-dependent).

A solution to the CSP is an assignment X: V $\rightarrow$ T such that every constraint c $\in$ C is satisfied (RUSSELL, NORVIG, 2022). The failure-mode analyzer defined in Section 4.4 quantifies constraint violations for a candidate assignment, providing the primary evaluation metric for this thesis.

This work approaches the CSP not via explicit constraint solving (as the Wave Function Collapse algorithm does, per GUMIN, 2016) but via generative modeling: a deep neural network is trained to approximate the distribution p(X | X satisfies C) implicitly, by learning from a corpus of valid example maps. At inference time, sampling from the learned distribution yields candidates that approximately satisfy the constraints; a deterministic repair stage (Chapter 6) enforces exact satisfaction for the local rules the model cannot reliably guarantee.

2 RELATED WORK

To situate the contributions of this work, it is essential to first survey the landscape of Procedural Content Generation (PCG) and deep generative models. This chapter provides such a review, charting a clear path from the foundational principles of PCG to the cutting-edge techniques that directly motivate this project. The chapter begins by establishing the review methodology in Section 2.1. It then proceeds with a focused analysis of three pivotal publications that, respectively, define the problem space, justify the adoption of deep learning, and introduce the core Flow Matching technique. The chapter culminates in a comparative analysis of these key approaches, identifying the gaps in the current literature that the method proposed in this project aims to address.

2.1 LITERATURE REVIEW METHODOLOGY

This section outlines the systematic methodology employed to identify, screen, and select the relevant literature for this work. The objective was to build a comprehensive understanding of the state-of-the-art in Procedural Content Generation (PCG), focusing on the transition from classic constraint-based methods to modern deep generative models for game map creation. The process was divided into three stages: identifying information sources, executing keyword-based searches, and applying selection criteria to filter the results.

First, the search was conducted across several major academic databases and search engines known for their extensive coverage in computer science and artificial intelligence. The primary sources included the ACM Digital Library, IEEE Xplore, and Google Scholar. Additionally, ArXiv was consulted for pre-print articles to ensure coverage of the latest, cutting-edge research in the rapidly evolving field of generative models.

The search strategy was executed in a multi-stage process to ensure both breadth and depth. The initial phase focused on identifying foundational survey papers using broad search strings such as “procedural content generation survey” and “deep learning for PCG review”. Once these cornerstone papers were identified, a more targeted search was conducted focusing on specific technique and applications. This involved combining technical keywords like “Wave

Function Collapse", "Generative Adversarial Network", "Diffusion Models", and "Flow Matching" with application-specific terms like "level generation" or "map generation". Further searches were performed to find alternative constraint-based methods (e.g., "Answer Set Programming for procedural content generation") and papers on evaluation methodologies (e.g., "evaluating PCG metrics") to establish a comprehensive basis for comparison.

The initial searches yielded a large number of publications. To refine this set to the most pertinent articles, a two-stage filtering process was applied based on a set of inclusion and exclusion criteria. Papers were included if they were: (a) foundational surveys of the PCG field; (b) directly addressed the generation of 2D tile-based maps or levels; (c) proposed or analyzed deep generative models relevant to the task; or (d) were published in high-impact journals or conferences. A strong preference was given to recent publications (post-2020) for topics related to deep learning to ensure the review was current. Conversely, papers were excluded if their primary focus was on non-map content (e.g. narrative, audio), or if they discussed older techniques not directly relevant for comparison. This structured process resulted in the final selection of the key papers analyzed in the following sections.

2.2 A FOUNDATIONAL SURVEY OF PROCEDURAL CONTENT GENERATION FOR GAMES

The work of Hendrikx *et al.* (2013) stands as a seminal contribution to the field, being the first comprehensive survey to formally structure and define the landscape of Procedural Content Generation for Games (PCG-G). The authors were motivated by the exponentially growing unsustainability of manual content creation, which was becoming a bottleneck in game development due to rising resource costs and scalability challenges (HENDRIKX *et al.*, 2013). The paper's primary objective was to use the spreading need to come up with an effective way to generate content that satisfied the hungry gamer community in terms of quality and quantity. The way the authors found to do this was to organize PCG-G research into a cohesive framework, establishing a common terminology and clear classification system for the various types of game content and the methods for generating them.

The main contribution of the survey that relates to this work, is a dual-taxonomy approach. First, it proposes a six-layered pyramid to classify game content (Figure 2.1), starting

from the most basic elements (Game Bits) and moving up through Game Space, Game Systems, Game Scenarios, Game Design, and finally Derived Content (HENDRIKX *et al.*, 2013). For this work, the "Game Space" layer is particularly relevant, as it formally defines the environment where the game takes place, including the generation of indoor and outdoor maps.

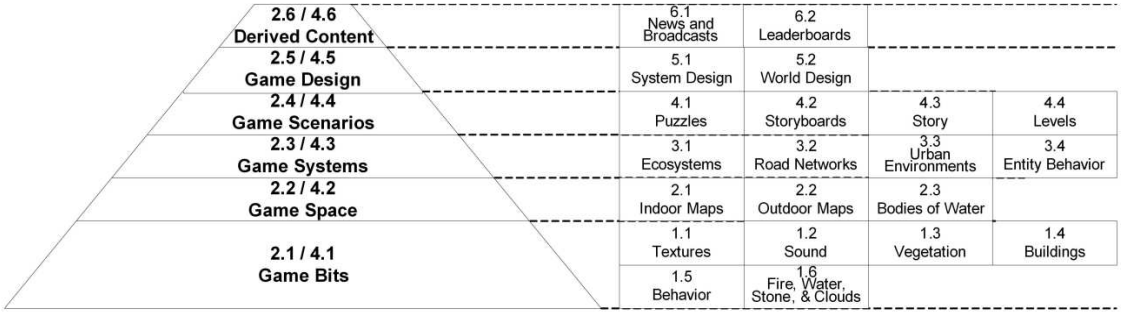


Fig. 2.1. Content hierarchy proposed by Hendrikx *et al.* (2013).

Second, and most crucially, the paper categorizes generation algorithms, placing "Constraint Satisfaction and Planning" under Artificial Intelligence methods (HENDRIKX *et al*., 2013). The authors explicitly define this approach as finding a valid state that satisfies a set of predefined constraints, acknowledging it as an NP-hard problem (HENDRIKX *et al*., 2013). This classification is fundamental for the present study, as it provides a formal academic validation for framing the procedural generation of maps as a Constraint Satisfaction Problem (CSP), which is the core premise of this project. Furthermore, the survey highlights that while constraint-based methods are precise, they struggle with scalability and complex global structures, which are exactly the limitations that modern deep generative models, as the one proposed in this work, aim to overcome. By outlining the state-of-the-art in 2013, Hendrikx *et al.* (2013) also implicitly highlights the limitations of classic methods, particularly the reliance on manually defined rules (which behave as the constraints of a CSP) such as what is performed by the WFC algorithm, which will be described in further detail in the subsequent chapters. Therefore, one could interpret the identification of such limitations by Hendrikx *et al.* (2013) as setting the stage for the subsequent rise of deep learning models capable of learning the CSP constraints implicitly, overcoming the previous manual definition need.

2.3 THE REVOLUTION OF DEEP LEARNING IN PCG

While the foundational survey by Hendrikx *et al.* (2013) structured the field of classic PCG, the subsequent years saw a paradigm shift towards data-driven methods, a transition captured by the survey of Liu *et al.* (2021) on deep learning for PCG. This work documents the revolution brought by deep generative models, which moved the field from manually defined rulesets to implicitly learned constraints from existing data (LIU *et al.*, 2021). The paper's primary contribution is to map how different families of deep learning, such as Variational Autoencoders (VAEs) and GANs, were being applied to generate game content, particularly game levels (LIU *et al*., 2021).

A crucial concept highlighted by Liu *et al.* (2021) is the challenge of functionality constraints. The authors argue that generating game content is fundamentally different from other generative tasks; a game level must be functional (i.e., playable), and a failure in functionality renders its utility as "essentially zero". This distinction is instrumental for this work, as it reinforces the necessity of viewing map generation through the lens of a CSP, even when using modern generative models.

The survey further details that early deep learning approaches often treated game levels as simple 2D images (LIU *et al.*, 2021). Models like GANs were trained to replicate the visual and stylistic patterns of existing levels (e.g. given a dataset of multiple valid levels, the model would implicitly learn how to replicate the levels, resulting in ample variety). However the authors emphasize that, unlike natural images, game levels must satisfy strict functionality constraints, such as playability. They review various metrics used to evaluate these models, such as playability rate, visual diversity, they often fail to guarantee functional coherence without complex, domain-specific constraints. This insight directly motivates our investigation into Flow Matching, which offers a more stable training regime and potential for better adherence to global structural constraints compared to the adversarial training of GANs.

2.4 FLOW MATCHING: AN EFFICIENT GENERATIVE MODELING PARADIGM

Building upon the landscape of deep generative models surveyed by Liu *et al.* (2021), the work of Lipman *et al.* (2023) introduces Flow Matching (FM), a novel and efficient paradigm

within the broader framework of Continuous Normalizing Flows (CNFs). This approach directly addresses the primary drawback of Denoising Diffusion Probabilistic Models (DDPMs), their slow and computationally intensive sampling process which requires numerous iterative steps. Flow Matching proposes an alternative by learning a deterministic vector field that defines a direct path from a simple noise distribution to the complex data distribution, which, for this study, would be a valid tile-based 2D map, thereby enabling significantly faster sample generation.

The core idea behind Flow Matching (LIPMAN *et al.*, 2023) is to train a neural network to learn the ideal vector field. In this context, a vector field can be described as a massive list of directions, one for each pixel in the map. Meanwhile, a neural network can be understood as a function. It is given two inputs: the current state of the data (which starts as noise and gradually becomes a map) and a "time" value (from 0 to 1, representing how far along the transformation process it currently is). This vector tells *exactly* how to adjust the value (color/tile type) of every single pixel at that specific moment to move it slightly closer to, together with all of the surrounding pixels, looking like a real map. The goal is to learn the vector field that optimally describes the probability path - the smooth, ideal transformation from pure noise at time 0 to valid map data at time 1. In other terms, the neural network should regress the vector field, that is, to learn to accurately predict the ideal directions at every point and time of this transformation.

Lipman *et al.* (2023) introduced a simulation-free training method called the Conditional Flow Matching (CFM) objective. It does not need to simulate the entire noise-to-map path during each training step, making training much faster. Instead, it learns efficiently by looking at conditional probability paths defined per data sample. This means the network learns from many single examples: for each real map in the training data, it learns the specific directions needed to transform noise into *that* specific map. By learning these individual transformations, the network generalizes to understand the overall transformation process for all maps. Once trained, the model generates new data by solving an Ordinary Differential Equation (ODE) that follows the learned vector field, starting from a random noise sample. The Euler-approach chosen by this work refers to specifically using the simple and fast Euler method to numerically integrate this ODE. The flow the Flow Matching model would follow to generate a valid map step-by-step would be:

1. Start with noise (a grid of random pixel values);
2. Ask the neural network (the vector field): “Based on these current pixel values and time *t = 0,* what small adjustments should I make?”;
3. The network outputs the adjustment vector (directions for every pixel);
4. Apply these tiny adjustments to all pixel values;
5. Slightly increase the time *t;*
6. Repeat steps 2-5 many times until *t = 1*. Solving the ODE is this step-by-step adjustment process. The final state of the pixel values at *t = 1 is* the generated map;

Compared to diffusion models, Flow Matching offers several key advantages. Firstly, the CFM objective leads to more stable and often faster training convergence, meaning the neural network learns reliably and reaches its best performance more quickly during the training phase. Secondly, and most critically for this project, the sampling process is significantly more efficient. Because the learned path is direct, high-quality samples can be generated with a considerably lower Number of Function Evaluations (NFE), the number of times it is needed to ask the neural network for directions, using standard ODE solvers. Thirdly, Flow Matching is not restricted to the probability paths inherent in diffusion processes; Lipman *et al.* (2023) explore the use of paths derived from Optimal Transport (OT), a mathematical theory regarding the most efficient way to morph one shape or distribution into another. These OT paths empirically result in even faster training and sampling due to their straight-line trajectories.

To make the term 'Euler-based' concrete: after training, the learned vector field defines an ordinary differential equation dx/dt = v_θ(x_t, t), where v_θ is the neural network parameterizing the vector field. Generating a map requires integrating this ODE from t=0 (noise) to t=1 (data). The Euler method is the simplest first-order numerical ODE integration scheme: it takes discrete steps of size Δt, computing x_{t+Δt} = x_t + v_θ(x_t, t) · Δt, repeatedly, until t=1 is reached. This work uses the Euler method with 50 uniformly-spaced steps (Δt = 0.02), balancing sampling quality against the number of neural-network forward passes. The choice of Euler over higher-order integrators (Runge-Kutta, Adams-Bashforth) trades marginal accuracy for simplicity of implementation and lower Number of Function Evaluations (NFE), the primary sampling-efficiency metric of Flow Matching approaches.

2.5 SPECIALIZED GENERATIVE APPROACHES: GANS, APPLIED DIFFUSION AND CONSISTENCY MODELS

Beyond the foundational surveys and the general generative architectures discussed above, recent literature has focused on applying specific deep learning families to procedural content generation, highlighting both their potential and their limitations relative to the goals of this work.

Silva *et al.* (2025) provide a comprehensive review of Procedural Game Level Generation using GANs. Their work analyzes the extensive use of GANs to replicate visual patterns in game levels. However they identify critical unresolved challenges, specifically the difficulty GANs face in satisfying strict functional constraints (such as playability and connectivity) without complex post-processing or hybrid search methods. Furthermore, they discuss the issues of training instability and "mode collapse" (in which the model fails to generate sufficiently diverse outputs), reinforcing the need for more stable generative paradigms, such as the Flow Matching approach this work proposes.

In the domain of diffusion models, Ren (2024) demonstrates the efficacy of Denoising Diffusion Probabilistic Models (DDPMs) in the environment of the game Minecraft. The author successfully trained a DDPM to generate coherent 3D structures with voxels (blocks that are the 3D equivalent to tiles), proving that iterative denoising processes *can* capture the spatial dependencies of voxel-based worlds effectively. However, the work also implicitly highlights the computational cost associated with the diffusion sampling process, which requires a large number of steps to converge to the desired high-quality result. This trade-off between output quality versus computational cost and generation speed serves as a direct motivation for this study to explore more efficient alternatives.

Finally, the most recent evolution in efficient sampling is represented by Consistency Models (Song *et al.*, 2023). While diffusion models rely on iterative noise removal, Consistency Models aim to map any point xt on the probability trajectory directly to the data origin x0 in a single step. This is achieved by enforcing a self-consistency property: the model's prediction of the final output must remain invariant regardless of the time step from which the prediction is made. Formally, for any points on the same trajectory, the model must satisfy f(xt, t) = f(xt', t').

While this paradigm offers the theoretical possibility of one-step generation, training these models directly can be unstable and complex. Flow Matching, therefore, serves as a robust middle ground: by learning a deterministic vector field with straight trajectories via Optimal Transport, it enables highly efficient sampling with standard ODE solvers without the rigid constraints and instability often associated with enforcing self-consistency.

In light of this comparative analysis, this work adopts the standard Euler-based Flow Matching framework, specifically leveraging the Conditional Flow Matching (CFM) objective. This objective simplifies the training process by regressing a time-dependent vector field that defines optimal paths between noise and data samples, avoiding computationally expensive simulations of diffusion processes. To parameterize this vector field, we employ a U-Net architecture, due to its encoder-decoder structure with skip connections, which allows it to effectively capture both high-level global structure (e.g. path connectivity) and low-level local details (e.g., tile adjacency). While the core algorithm remains consistent with Lipman *et al.* (2023), the architecture will be adapted to handle discrete tile-based data through *one-hot encoding*, rather than continuous image data. To validate this approach, the proposed model will be empirically tested against a WFC baseline. The comparison will focus on quantitative metrics of sampling efficiency (NFE) and qualitative assessment of global coherence, aiming to demonstrate that Flow Matching can achieve the structural quality of diffusion models with the operational speed required for game development pipelines.

2.6 COMPARATIVE ANALYSIS FRAMEWORK

To synthesize the diverse approaches discussed in this chapter, we establish a comparative framework based on three critical criteria relevant to the goals of this thesis: Constraint Handling, Sampling Efficiency, and Data Requirements.

Constraint Handling evaluates the method's ability to satisfy the game design rules defined in the introduction. This is a qualitative assessment based on the reported capabilities in the literature. An approach is rated as "Excellent" if it consistently produces playable maps that respect both local and global rules without extensive manual intervention. It is rated as "Poor" or "Limited" if it frequently produces invalid states, broken paths, or requires significant post-generation repairs (repairing algorithms).

Sampling Efficiency measures the computational cost and speed of generating a single valid map instance. For deep generative models, this is quantitatively estimated by the NFE. High Efficiency corresponds to methods that generate content in a single pass or very few steps (Low NFE, e.g. < 50), enabling near real-time generation. Low Efficiency corresponds to iterative methods requiring hundreds or thousands of steps (High NFE, e.g. > 500) to converge to a valid result, making them unsuitable for real-time applications.

Data Requirements distinguishes the input necessary to train or configure the algorithm. Single-Example methods like WFC typically require only a single input image or a small set of tiles to extract rules. Dataset-Driven methods like Deep learning approaches (GANs, DDPMs, Flow Matching) require a prepared dataset of valid maps. This involves pre-processing steps such as *chunking*, dividing large maps into smaller training tensors and *one-hot encoding* to represent categorical tile data, as described in the methodology of this work.

Approach	Representative Work	Constraint Handling	Sampling Efficiency	Data Requirement	Key Limitation
Standard WFC	Gumin (2016)	Excellent Local / Poor Global	Very High	Single Example	Rigid output; lack of global structure.
Controllable WFC	Cheng *et al.* (2021)	Improved Global via heuristics	Medium (Backtracking)	Single Example + Heuristics	Complexity of manual rule definition.
GANs	Silva *et al.* (2025)	Learns patterns; struggles with validity	High (Single pass)	Large Dataset	Mode collapse; training instability.
DDPMs	Ren (2024)	Excellent Global & Local	Low (High NFE)	Large Dataset	Very slow generation (1000s of steps).
Flow Matching	Lipman *et al.* (2023)	Excellent Global & Local	High (Low NFE)	Large Dataset	Emerging technique; requires ODE solver.
This work (Hybrid CFM + Repair)	Slongo (2026)	Excellent Global & Local	High (Low NFE)	Large Dataset + Rule Set	Manual rule-set design; corpus-specific

Table 2.1. Comparative analysis of generative approaches for procedural map generation

3 PROPOSED DEVELOPMENT PLAN

Following the theoretical foundation and the comparative analysis of generative models, this chapter presents the methodological approach designed to achieve the objectives stated in Chapter 1. The method treats the procedural generation of 2D game maps as a continuous transformation from a noise distribution to a valid map distribution that satisfies implicit constraints, with each stage of the method chosen and justified on methodological grounds.

The method comprises three connected stages, each addressing one of the specific objectives from Section 1.1.2. Data Engineering (Section 3.2) defines how heterogeneous source maps are converted into a uniform tensor representation suitable for neural training, addressing Objective O2. The Generative Model (Section 3.3) describes the U-Net architecture and the Conditional Flow Matching training objective adopted to learn the constraint distribution implicitly from examples, addressing Objective O3. Inference and Constraint Management (Section 3.4) details the Euler-based ODE solver used to traverse the learned vector field at generation time, together with the conditional vectors used to guide generation toward specific design requirements, addressing Objective O4. Section 3.5 establishes the validation strategy that the method is measured against, addressing Objective O5. Section 3.6 describes the iterative development methodology that organizes how the method is built and evaluated across successive iterations, with each iteration reported in Chapters 4 through 6.

3.1 SOLUTION ARCHITECTURE

The system is designed as a deep learning pipeline where a neural network approximates a time-dependent vector field. This vector field defines the optimal trajectory to transform random noise into a coherent game map. Figure 3.1 illustrates the high-level architecture of the proposed solution, detailing the flow of data from raw assets to the generated output.

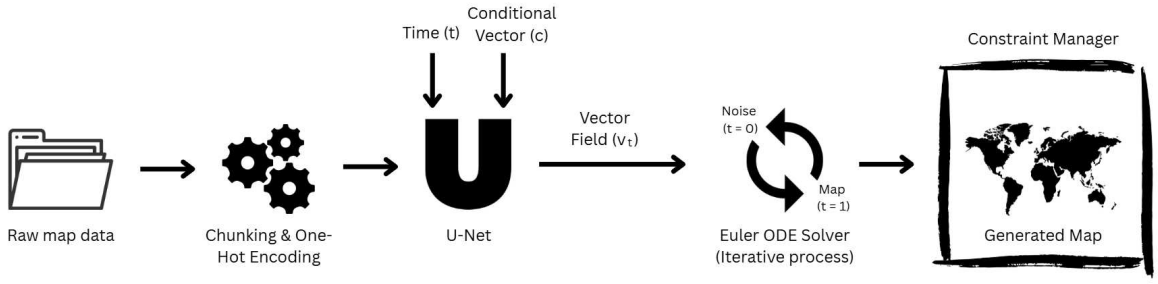


Figure 3.1: High-level architecture of the proposed Conditional Flow Matching generation pipeline.

As illustrated, the process begins with the extraction and preprocessing of tile-based maps into a tensor format suitable for the neural network. The core component is a U-Net architecture trained using the Conditional Flow Matching (CFM) objective. Finally, the inference module utilizes an Euler ODE solver to generate new content, guided by conditional vectors and a constraint manager that ensures the output meets specific design requirements.

3.2 DATA ENGINEERING PIPELINE

To train the model to recognize valid local and global constraints (Objective O2), a robust dataset is required. The data pipeline consists of three steps:

1. Data Acquisition: The dataset will be constructed from 2D tile-based maps (e.g., extracted similar games or procedural generation benchmarks). These maps represent the "valid state" of the Constraint Satisfaction Problem (CSP), where the "energy" (constraint violations) is zero.
2. Chunking: Large game maps will be divided into smaller, fixed-size chunks (14x28 tiles). This standardization is necessary for efficient batch processing during neural network training.
3. One-Hot Encoding: Unlike standard image generation which uses RGB channels, game maps are categorical. Each tile material (e.g., water, grass, sand) and geometry (e.g., square, rectangle, triangle) will be converted into one-hot encoded vectors. Consequently, the input to the model will be a tensor of shape (C, H, W), where C represents the number of unique tile types.

3.3 GENERATIVE MODEL: U-NET AND CONDITIONAL FLOW MATCHING

The core of the proposal is the substitution of the iterative diffusion process with a deterministic Flow Matching mechanism (Objective O3).

3.3.1 Network Architecture (U-Net)

We employ a U-Net architecture as the function approximator for the vector field. Originally designed for segmentation, the U-Net is ideal for this task due to its encoder-decoder structure with skip connections. While the encoder captures global context (global constraints), such as biome distribution and path connectivity, the decoder reconstructs fine details (local constraints), ensuring correct tile adjacency. The network will receive the current time step t and a conditional vector c as additional inputs, injected into the network layers via adaptive normalization (AdaGN) or concatenation.

3.3.2 Training Objective (CFM)

The model is trained using the Conditional Flow Matching objective proposed by Lipman *et al.* (2023). We first introduce the notation used throughout the remaining chapters:

- $x_0 \in \mathbb{R}^{C \times H \times W}$: a sample from a simple noise distribution (typically standard Gaussian, $x_0 \sim \mathcal{N}(0, I)$). Represents the starting point of the generation process at time t=0.
- $x_1 \in \mathbb{R}^{C \times H \times W}$: a real training sample (one-hot encoded map chunk). Represents the target of the generation process at time t=1.
- $t \in [0, 1]$: continuous time parameter interpolating between noise and data.
- x_t: the linear interpolant at time t, computed as $x_t = (1-t) \cdot x_0 + t \cdot x_1$. At t=0, $x_t = x_0$ (pure noise). At t=1, $x_t = x_1$ (pure data).
- $u_t(x_t \mid x_1)$: the target vector field, defined as the direction from noise to data along the interpolation path: $u_t(x_t \mid x_1) = x_1 - x_0$. Note that this quantity is constant with respect to t along a given path; this constancy is what makes the CFM objective simulation-free.

• $v_\theta(x_t, t)$: the neural network parameterizing the vector field, with learned parameters θ.

Instead of simulating the full noise-to-data trajectory, the model learns to regress the target vector field $u_t(x_t \mid x_1)$ that points directly from the current noisy state x_t towards the target data sample x_1. The training objective minimizes the mean squared error between the network prediction and the target: $L(\theta) = E_{(x_0, x_1, t)} [\| v_\theta(x_t, t) - u_t(x_t \mid x_1) \|^2]$ where x_0 is drawn from noise, x_1 from the training corpus, and t uniformly from $[0, 1]$. This simulation-free training allows for faster convergence compared to traditional diffusion training.

3.4 INFERENCE AND CONSTRAINT MANAGEMENT

The generation process is framed not just as image synthesis, but as an iterative planning process to minimize constraint violations, which will be addressed as 'energy' in this work.

The inference process starts with random noise (high entropy/high violation state). The Euler solver iteratively applies the learned vector field to 'move' the map configuration towards a valid state (zero violations). Concretely, the solver executes 50 iterations of the update $x_{t+\Delta t} = x_t + v_\theta(x_t, t, c) \cdot \Delta t$ where v_θ is the U-Net's forward pass parameterizing the vector field, c is the conditional vector defined in Section 3.3, and $\Delta t = 0.02$ corresponds to the uniform step size covering the interval $t \in [0, 1]$. This iterative refinement parallels the concept of annealing, where the system gradually settles into a low-energy configuration.

To ensure the generated maps meet specific design requirements, the generation process is guided by a conditional vector c. This vector encodes high-level attributes (e.g., 'Desert Biome', 'Contains Lava').

Constraint Manager (Proposed Component): To explicitly validate the generated maps, a 'Constraint Manager' module will be implemented. This component will firstly take the generated board as input, then perform a reverse lookup to identify tile rules, and finally, calculate the total number of constraint violations (local adjacency errors, global path failures). This manager provides the quantitative metric for evaluating the model's success in solving the Constraint Satisfaction Problem.

3.5 VALIDATION STRATEGY

The implemented solution will be evaluated based on the comparative framework established in Chapter 2. The primary metrics for validation will be:

1. Sampling Efficiency: Measured by the NFE required to produce a valid map and the total wall-clock time for generation.
2. Constraint Satisfaction: Evaluated by the Constraint Manager, calculating the percentage of valid tile adjacencies (local) and the structural coherence of biomes (global).
3. Diversity: Assessed to ensure the model does not suffer from mode collapse, comparing it against the output of a Wave Function Collapse (WFC) or Genetic Algorithm baseline.

3.6 ITERATIVE DEVELOPMENT APPROACH

Development followed an iterative methodology in which each iteration produced a complete working version of the pipeline, was measured against the failure-mode evaluation framework defined in Section 4.5, and informed the design of the next iteration. The objective of the first iteration is the simplest functional integration of all components proposed above: data pipeline, U-Net architecture, training loop, ODE solver, and evaluation methodology. Subsequent iterations target specific qualitative weaknesses observed in the previous iteration's output, prioritized by impact on the failure modes. Figure 3.2 illustrates the four iterations that constitute this thesis's development, previewing what each iteration will introduce and the chapter in which it is reported in detail.

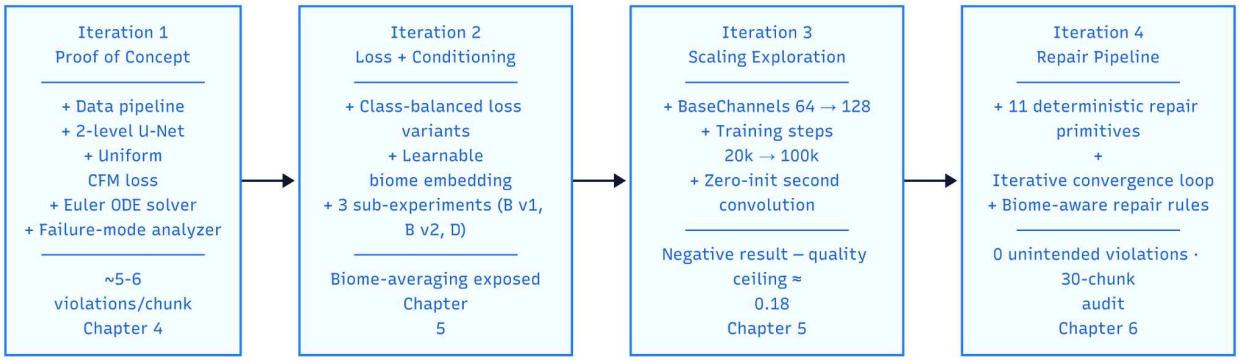


Figure 3.2: Roadmap of the four iterations reported in Chapters 4-6.

This staged approach is preferred over monolithic development because it allows verification of pipeline correctness at each stage before introducing additional complexity, and because it reduces the risk of late discovery of architectural problems.

4 ITERATION 1: PROOF OF CONCEPT DEVELOPMENT AND ANALYSIS

Following the methodology established in Chapter 3, this chapter reports the development of the Conditional Flow Matching pipeline and the results of its first complete end-to-end execution. Section 4.1 describes the technical infrastructure, Section 4.2 the data pipeline, Section 4.3 the network architecture, Section 4.4 the training procedure and evaluation methodology, Section 4.5 the end-to-end results, and Section 4.6 the limitations that motivate subsequent iterations. The scope of Iteration 1 is deliberately constrained to a two-level U-Net with linear-interpolation Flow Matching paths and additive time conditioning; the Optimal Transport variant of Lipman *et al.* (2023), deeper architectures, adaptive normalization, and conditional generation are deferred to subsequent iterations.

4.1 IMPLEMENTATION INFRASTRUCTURE

The workflow is implemented in C# on the .NET 8 platform. Neural network operations are delegated to TorchSharp 0.107.0, a managed wrapper over the libtorch native runtime that exposes the PyTorch tensor API and autograd engine to .NET applications. C# was chosen over Python in order to integrate with the codebase of the organization hosting the project, which is implemented in C#, while still providing access to the same GPU operations available in PyTorch through TorchSharp. The Conditional Flow Matching loss, the U-Net architecture, the Euler ODE solver, and the failure-mode analyzer are all implemented from first principles in new code; no external generative-modeling library is used[1].

The pipeline is implemented as a set of five domain libraries and two command-line entrypoints organized within a four-layer architectural convention used by the parent organization. Figure 4.1 illustrates the assembly dependency structure: a data loading layer consumes the VGLC corpus, a model layer defines the U-Net and its building blocks, a training layer wraps the Conditional Flow Matching loss in an optimizer loop, an inference layer composes the Euler ODE solver, and an evaluation layer implements the failure-mode analysis and visualization. The first command-line entrypoint runs training and writes a checkpoint; the second loads a checkpoint and generates chunks with their failure-mode analysis.

[1] Source code available at: https://github.com/luizslongo/procgen-flow-matching

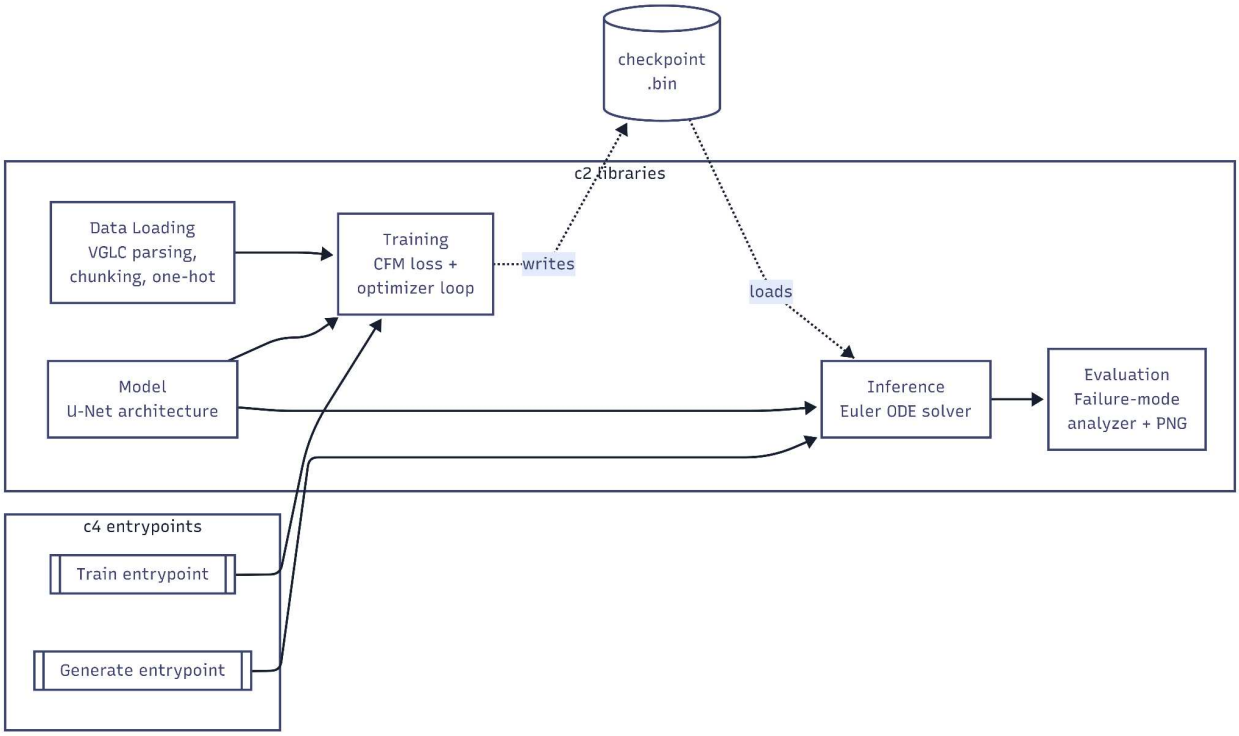


Figure 4.1: Pipeline architecture showing the five domain libraries and their dependency relationships.

Training and inference were performed on a single NVIDIA GeForce RTX 5060 GPU (Blackwell architecture, compute capability sm_120, 8 GB GDDR7), installed in the author's personal workstation. The CfmTrainingLoop selected the CUDA device when available and falls back to CPU otherwise; this fallback was used during early development to verify pipeline correctness on a short 2,000-step run before scaling to the 20,000-step GPU training reported in Section 4.5.2. TorchSharp 0.107.0 was selected specifically because earlier versions of the wrapper, including the 0.105.1 release used during initial CPU development, ship a libtorch native runtime that predates the Blackwell architecture, producing CUDA initialization errors on the RTX 5060. The U-Net's small parameter count (approximately three million parameters in the configuration described in Section 4.3) and the modest chunk dimensions of 14 by 28 keep tensor sizes well below the device's capacity, so neither training nor inference encountered out-of-memory conditions. Generation of ten chunks at NFE 50 completes within seconds on the same device.

4.2 DATA PIPELINE

The training corpus is drawn from the Video Game Level Corpus (VGLC), a public dataset of game levels first published by Summerville *et al.* (2016)[2]. VGLC distributes Super Mario Bros levels as ASCII text files in which each character represents one 16-by-16 pixel tile from the original NES levels. Fifteen levels are used in this work, drawn from the Processed subdirectory of the VGLC Super Mario Bros distribution: mario-1-1, mario-1-2, mario-1-3, mario-2-1, mario-3-1, mario-3-3, mario-4-1, mario-4-2, mario-5-1, mario-5-3, mario-6-1, mario-6-2, mario-6-3, mario-7-1, mario-8-1. Each level has fourteen tile rows; column count varies from 149 to 373 tiles depending on the level.

The VGLC encoding distinguishes thirteen visual tile types from the original Super Mario Bros levels; the present implementation extends this set with a fourteenth Error sentinel at index 0, used to signal an uninitialized or invalid state in code and never expected to appear in valid data. Reserving the zero index for an explicit error value follows a defensive-programming convention adopted by the parent organization's codebase: a tile produced from an uninitialized memory region will be detected as Error rather than silently interpreted as one of the visual types. Table 4.1 lists the complete vocabulary, mapping each tile-type enumeration value to its VGLC ASCII character and to the visual element it represented in the NES game.

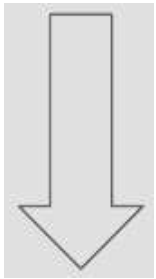
[2] The VGLC is publicly available at: https://github.com/TheVGLC/TheVGLC

Enum Value	Index	VGLC Character	Visual Element in SMB
Error	0	(none)	Uninitialized sentinel; never appears in valid data
Empty	1	-	Air / sky
Solid	2	X	Solid ground or wall block
Breakable	3	S	Breakable brick block
QuestionFull	4	?	Active question block (contains an item)
QuestionEmpty	5	Q	Already-hit question block (item collected)
Enemy	6	E	Enemy character (typically a Goomba)
PipeTopLeft	7	<	Pipe entrance, left half of cap
PipeTopRight	8	>	Pipe entrance, right half of cap
PipeBodyLeft	9	[	Pipe vertical body, left half
PipeBodyRight	10	]	Pipe vertical body, right half
Coin	11	o	Collectible coin
BulletBillLauncher	12	B	Bullet Bill cannon, top piece
BulletBillBody	13	b	Bullet Bill cannon, vertical body

Table 4.1. Tile vocabulary used by the model.

To make the ASCII → tensor → PNG pipeline concrete, Figure 4.2 pairs one representative chunk from a generated batch shown as both its VGLC-encoded ASCII form (the format the model consumes as a one-hot tensor) and its rendered PNG (produced by an auxiliary Python sprite renderer that maps each tile-type index to a 16×16 pixel sprite extracted from the original NES game screenshots). The ASCII form exposes the position-by-position discrete structure the model operates on; the PNG form exposes the visual character the human reader judges. All figures in Chapters 4 through 6 that show generated chunks use the same rendering pipeline.

```
--------------------------
--------------------------
-S----------S------------
--------------------------
--------------------------
-X------------------------
------?---------------------
--------------------------
--------------------------
--------------oS------------
------------------◇--------
SS--SS---X--------[]--------
-------------o----[]--------
XXXXXXXXXXXXXXXXXXXXXXXXXXX
```



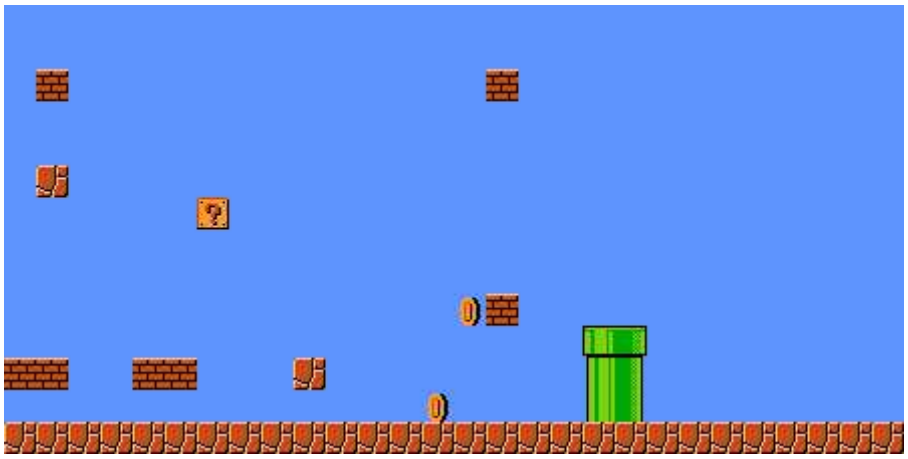


Figure 4.2: Generated PNG matching the ASCII above.

4.3 NETWORK ARCHITECTURE

The vector field that the Conditional Flow Matching objective requires is parameterized by a U-Net. The U-Net is a natural fit for this task because its encoder-decoder structure with skip connections lets the same network simultaneously capture global structure (where pipes are placed, how brick rows align with the ground line) and local detail (which tile sits at each specific position). Figure 4.3 illustrates the architecture used in Iteration 1: a two-level encoder, a single-block bottleneck, a two-level decoder mirroring the encoder structure, and two 1-by-1 channel-adapter convolutions at the input and output boundaries.

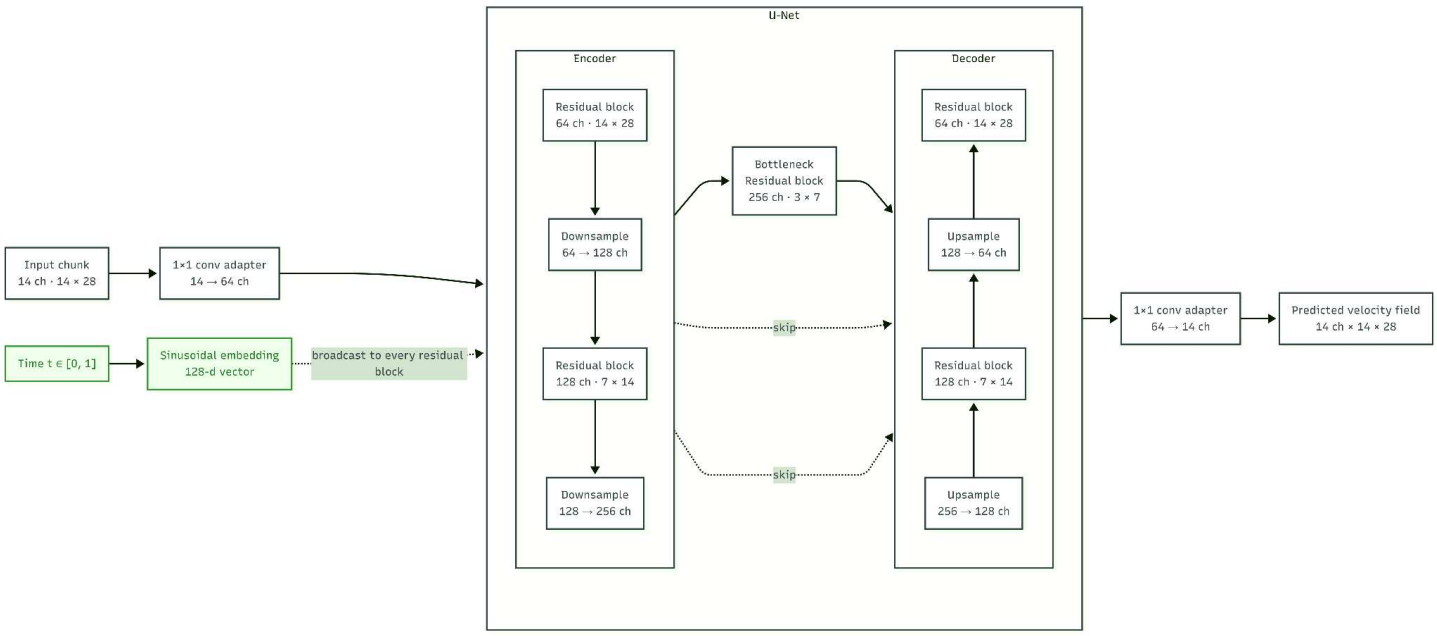


Figure 4.3: Two-level U-Net architecture used as the vector-field approximator.

A two-level depth is the maximum permitted by the chunk dimensions of 14 by 28: a third level of downsampling would compress the bottleneck to a 1-row spatial map, eliminating most of the local context that the residual blocks need to operate on. Channel width doubles at each downsampling step, taking the feature representation from 64 channels at full resolution to 128 at half resolution to 256 at the bottleneck, with the decoder mirroring this schedule on the way back up. The skip connection from each encoder level is concatenated along the channel dimension with the corresponding decoder level's input before the next residual block runs; this

supplies the decoder with both the coarse global signal carried through the bottleneck and the fine spatial detail preserved at the matching encoder depth.

The two 1-by-1 channel-adapter convolutions at the network's boundaries serve as a translation layer between the visible space (the 14-channel one-hot tile vocabulary from Table 4.1) and the working space (the 64-channel internal feature representation). The input adapter projects each pixel's 14-dimensional one-hot vector into a 64-dimensional learned feature vector; the output adapter performs the inverse projection, mapping the final 64-channel feature map back into a 14-channel velocity-field prediction with one channel per tile type.

The internal structure of one residual block is shown in Figure 4.4. The block applies two 3-by-3 convolutions, each followed by GroupNorm and a SiLU activation. Between the two convolutions, the time embedding for the current diffusion timestep is added to the intermediate feature map. An identity shortcut from the block's input is added to the second convolution's output; when the block changes channel width (which happens at the start of each encoder level after a downsampling step, and at the start of each decoder level after an upsampling step), the identity shortcut is replaced by a 1-by-1 convolution that matches the dimensions. The residual structure stabilizes gradient flow through the bottleneck and lets the optimizer treat each block as a refinement of its input rather than a from-scratch reconstruction, a property that aligns naturally with the Conditional Flow Matching objective, which is itself a refinement signal pushing the noisy interpolant toward the data sample.

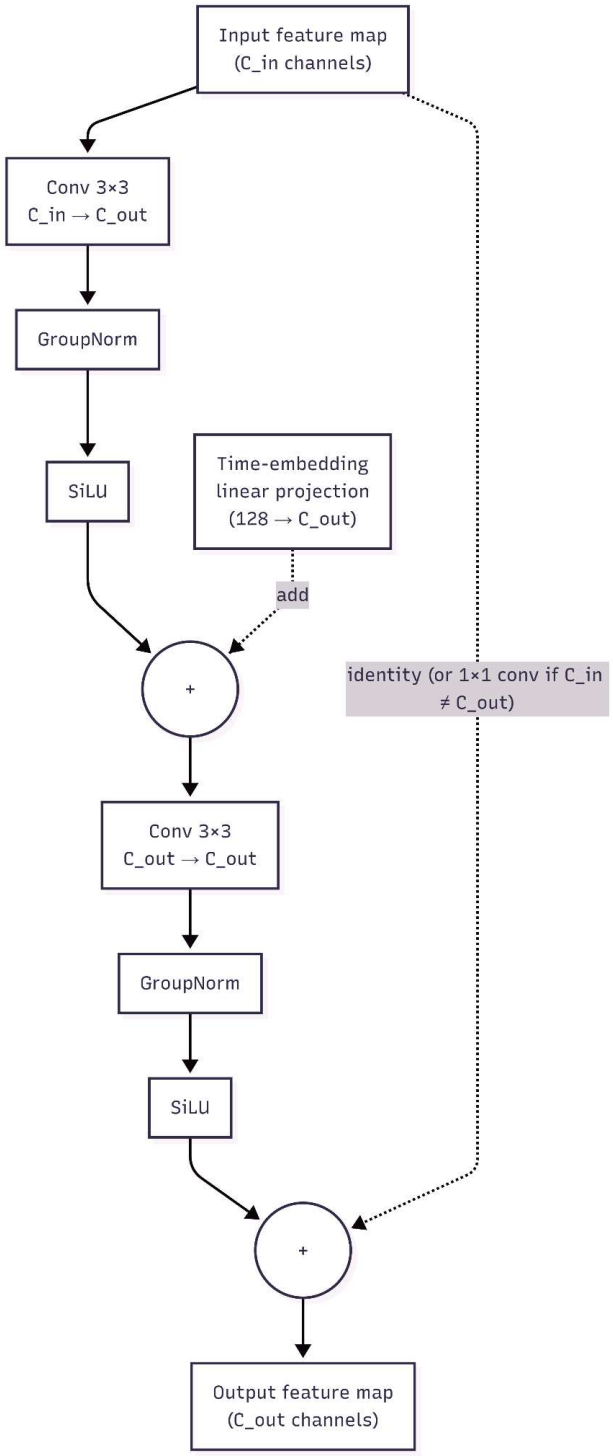


Figure 4.4: Internal structure of one residual convolutional block.

Time conditioning enters the network through a sinusoidal embedding. The embedding takes a scalar time value in the unit interval, scales it by 1000, and computes a 128-dimensional vector whose first 64 entries are sine components and last 64 entries are cosine components at logarithmically spaced frequencies. This same vector is reused across every residual block in the network; a small linear projection inside each block adapts the 128-dimensional time vector to the block's current channel width before adding it to the intermediate feature map. The network thereby knows, at every spatial position and at every block, which diffusion timestep it is currently denoising.

The Iteration 1 U-Net has approximately three million trainable parameters. No conditional vector for biome control (Objective O4) is provided to the network at this stage; a biome-embedding parameter present in the network code was added in preparation for Iteration 2 and is left unused during Iteration 1 training. The Optimal Transport variant of CFM proposed by Lipman *et al.* (2023) and the use of adaptive group normalization for time injection (in place of additive injection) are both deferred to subsequent iterations for the reasons described in Section 4.1.

4.4 TRAINING PROCEDURE AND EVALUATION METHODOLOGY

Training is orchestrated by a loop that loads the chunked dataset described in Section 4.2 into memory once at startup, constructs a fresh U-Net model (Section 4.3), wraps the model parameters with an Adam optimizer at learning rate 5×10^{-5}, and runs a fixed number of optimization steps. Each step samples a batch of 32 chunks uniformly at random with replacement from the corpus, converts the chunks into a one-hot tensor of shape (32, 14, 14, 28), and submits the tensor to a single forward-backward pass governed by the Conditional Flow Matching objective. Iteration 1 runs for 2,000 steps, a budget chosen as the minimum required to verify that the pipeline can be assembled end-to-end without yet examining generation quality. All randomness in the training loop is controlled by a single seed value of 42, applied uniformly to the model's weight initialization, the per-step noise samples drawn inside the loss computation, and the batch-index sequence. The shared seed makes the trained checkpoint reproducible from the same code state and the same VGLC corpus.

The Conditional Flow Matching loss is computed sample-by-sample. For each chunk in the batch, the loss first draws a fresh Gaussian noise tensor x_0, sampled from the multivariate Gaussian distribution $\mathcal{N}(0, I)$ with the same shape as the real chunk, and a fresh time scalar t sampled uniformly from the unit interval [0, 1]. The linear interpolant between the noise and the real chunk x_1 is then computed as $x_t = (1 - t) \cdot x_0 + t \cdot x_1$. The target velocity is the constant displacement along this path, $u = x_1 - x_0$. The U-Net's forward pass returns a predicted velocity field of the same shape as u, and the loss is the mean squared error between the prediction and the target, averaged across all elements of the batch tensor. This is the linear-path simulation-free objective from Lipman *et al.* (2023): no simulation of the full noise-to-data trajectory is required during training because the linear path provides a closed-form target velocity at every interpolation time. Iteration 1 applies uniform weighting across all elements of the batch tensor; a class-balanced variant motivated by the imbalanced tile-frequency distribution of the corpus is deferred to Iteration 2.

The optimizer step applies the gradient with one safeguard: gradient norms are clipped to a maximum L2 magnitude of 0.5 before the Adam update. The clip was introduced after an early training run diverged to NaN at step ~50, when a single batch produced a large gradient that overshot the loss landscape; it is preserved across all subsequent iterations of this work as a stability guard. Loss values are recorded every 50 steps for later analysis (Section 4.5), and model checkpoints are written every 500 steps so an interrupted run can be resumed without retraining from scratch.

The trained model is evaluated by generating chunks from random noise and analyzing each generated chunk for structural correctness against six categories of failure mode, each defined as an independent local scan over the chunk's tile array:

1. Broken horizontal pipe segment: any pipe body left-half whose right neighbor is not the corresponding right-half.

2. Broken pipe-top connection (left side): any pipe-top left-half whose lower neighbor is not the corresponding pipe body left-half.

3. Broken pipe-top connection (right side): symmetric case for the right-half.

4. Broken bullet-bill launcher: any launcher whose lower neighbor is not a launcher-body tile.

5. Floating enemy: any enemy tile not resting on a solid or breakable tile.

6. Discontinuous ground: any position in the chunk's bottom row whose tile is not solid.

The six categories were chosen as the minimum set of local structural rules that any visually valid Super Mario Bros chunk must satisfy; they are independent of any biome-specific consideration, and they can be checked in linear time over the chunk's tile array without requiring a playability simulator. Each analysis produces both a per-category count and a total violation count per chunk; aggregating across a generated batch produces the per-batch totals reported in Section 4.5 and used as the primary evaluation metric throughout this work.

The validation strategy proposed in Chapter 3 (Section 3.5) listed three primary metrics: sampling efficiency, constraint satisfaction, and diversity. Iteration 1 measures only the second of these; constraint satisfaction via the failure-mode violation count defined above.

Sampling efficiency and diversity require comparison against a baseline external generator (Wave Function Collapse or a Genetic Algorithm) which is deferred to subsequent iterations. The Iteration 1 metric thus answers a narrower question than the full validation strategy: does the trained model produce chunks whose structural error rate is low enough to justify continuing with deeper iterations?

4.5 FIRST END-TO-END RESULTS

Iteration 1 was executed in two consecutive training phases that share the same model architecture, the same Conditional Flow Matching loss formulation, and the same hyperparameters apart from training duration and device. Phase 1 ran for 2,000 steps on the workstation's CPU as a correctness verification of the assembled pipeline. Phase 2 ran for 20,000 steps on the RTX 5060 GPU after pipeline correctness was confirmed, producing the trained checkpoint that the qualitative comparisons later in this chapter are drawn from. Both phases are reported below.

4.5.1 Phase 1: CPU Correctness-Verification Run (2,000 Steps)

The first training run intentionally targeted the simplest possible execution path: every pipeline component (data loading, model construction, loss computation, backpropagation, gradient clipping, optimizer update, checkpoint serialization) was exercised on the CPU before

any GPU-specific runtime dependency was introduced. Two earlier training attempts had failed: the first descended to step 200 with a healthy loss of 0.69 before being interrupted, and the second produced NaN losses from step 0 onwards through the entire run. Root-cause analysis attributed both failures to the original training loop having neither a random seed nor gradient clipping: different random weight initializations produced different gradient magnitudes per run, and some initializations triggered immediate gradient explosion. The three corrections described in Section 4.4 (fixed seed 42, global gradient norm clip at 0.5, learning rate lowered from 1×10^{-4} to 5×10^{-5}) eliminated both failure modes; the first run after the corrections completed all 2,000 steps without any NaN value appearing in the loss log.

The corrected training run produced the loss trajectory shown in Figure 4.5. The loss value descends from 1.07 at step 0 to 0.21 at step 1,950, reaching a minimum of 0.13 at step 1,900. The descent is monotonic across coarse time scales but exhibits the per-step variance characteristic of the Conditional Flow Matching objective: each step samples a fresh time scalar t from the unit interval and a fresh noise tensor x_0, so the loss landscape sampled by any one step is different from the previous step's. The trajectory plateaus around step 750 at a value of approximately 0.25 and continues to descend more slowly across the remaining 1,250 steps.

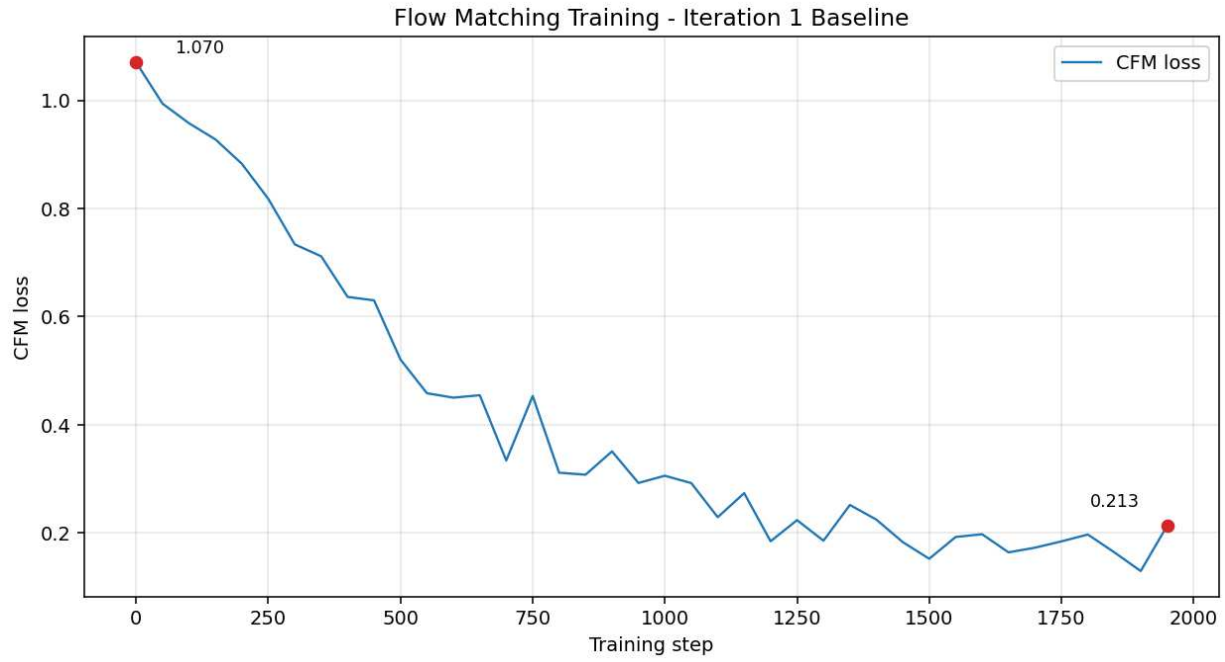


Figure 4.5: Conditional Flow Matching training loss across the 2,000-step Iteration 1 Phase 1 run.

Ten chunks were generated from the trained checkpoint using the Euler ODE solver at NFE 50, the default value adopted for Iteration 1. Each chunk was then analyzed independently for structural correctness against the six failure-mode categories defined in Section 4.4. The per-category counts aggregated across all ten chunks are shown in Table 4.2.

Failure mode	**Count**	**Percentage**
DiscontinuousGround	46	59.7
FloatingEnemy	12	15.6
BrokenPipeHorizontal	6	7.8
BrokenPipeTopLeft	6	7.8
BrokenPipeTopRight	4	5.2
BrokenBulletBill	3	3.9
Total	**77**	**100.0**

Table 4.2: Iteration 1 Phase 1 aggregate failure-mode counts across 10 generated chunks.

The mean violation count is 7.7 per chunk and the mean violation rate per tile position is 1.96 percent. The single best chunk produced one violation and the worst produced twelve, indicating that the model is capable of producing nearly-valid outputs but is not yet consistent across the full sample.

Two of the ten chunks are reproduced below as paired ASCII renderings and PNG visualizations to illustrate the visual character of the generation. The PNG renderings are produced by a small auxiliary pipeline that complements the C# generation code: a Python script extracts one 16-by-16 pixel sprite per tile type from the original VGLC level screenshots (writing thirteen sprite PNGs to a sprites directory), and the chunk-rendering routine takes a generated tile-index map and composes a 28-by-14 grid of sprites into a single chunk PNG (448 by 224 pixels at one tile per 16 pixels). The ASCII view exposes the position-by-position structure that the failure-mode analyzer scores; the PNG view exposes the visual character that a human reader would judge.

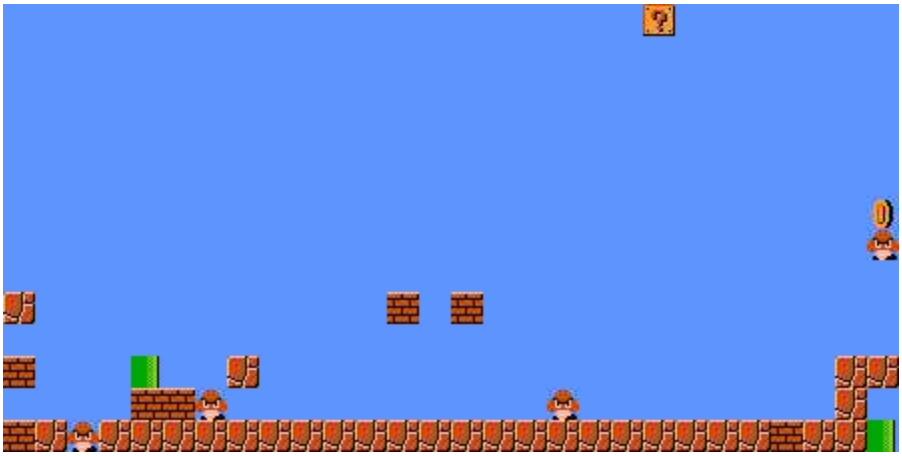


Figure 4.6: Iteration 1 Phase 1 chunk 9, the lowest-violation sample.

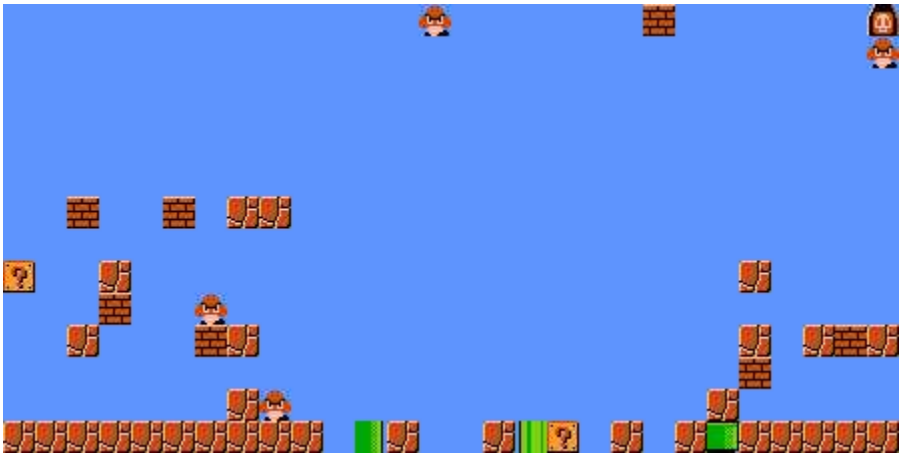


Figure 4.7: Iteration 1 Phase 1 chunk 7, a representative average-quality sample (eight violations).

Several patterns are visible across the ten generated chunks. The bottom row consistently contains a horizontal band of Solid tiles, indicating that the model has learned the prior "ground belongs at the bottom of the chunk." The upper rows are predominantly Empty, indicating that the model has learned the symmetric prior "sky belongs at the top of the chunk." Discrete tile types appear correctly: bricks, question blocks, coins, enemies, and pipe pieces all materialize as the right tile-type value at the right pixel location of an argmax over the 14-channel output. However, multi-tile structures are not consistently assembled: chunk 7 contains an isolated PipeBodyLeft without a matching PipeBodyRight next to it, a PipeTopRight appears at the bottom-right of the same chunk without a paired PipeTopLeft, and chunk 9's ground row is

interrupted at column 24 by a stray PipeBodyRight tile. These structural inconsistencies account for the 7.7-violation-per-chunk average.

4.5.2 Phase 2: GPU Production Run (20,000 Steps)

After Phase 1 confirmed pipeline correctness, three infrastructure changes were made to enable a longer training run on the GPU: the TorchSharp wrapper was upgraded from 0.105.1 to 0.107.0 to add Blackwell sm_120 support for the RTX 5060, a tensor-device mismatch in the time-embedding code path was corrected so the embedding builds on the same device as the model parameters, and the training step count was raised from 2,000 to 20,000. The model architecture, loss formulation, and remaining hyperparameters were left unchanged from Phase 1. The longer run completed on the GPU in approximately ten minutes and produced the trained checkpoint used for the qualitative comparisons reported throughout the remainder of this work.

The Phase 2 loss descended from 1.07 at step 0 to 0.028 at step 20,000, a 97 percent reduction. The Phase 2 loss log was lost when subsequent training runs in Iteration 2 reused the same default log filename without first archiving the previous run, but the start and end values were recorded at training-completion time and are reproduced here for completeness. The checkpoint snapshots saved every 500 steps during the run are preserved on disk, so the trained weights themselves are available even though the per-step loss curve is not.

Phase 2 produced chunks that are visually recognizable as Super Mario Bros level segments. Figure 4.8 shows one such chunk: a well-formed two-cell-wide pipe assembly sits on a near-complete ground row of Solid tiles, with scattered Breakable blocks at the upper edges and predominantly Empty sky above. The pipe assembly is structurally coherent in the sense the Iteration 1 Phase 1 chunks rarely achieved: the PipeTopLeft and PipeTopRight tiles sit side by side on a single row, and the PipeBodyLeft and PipeBodyRight tiles sit directly below them in matching columns.

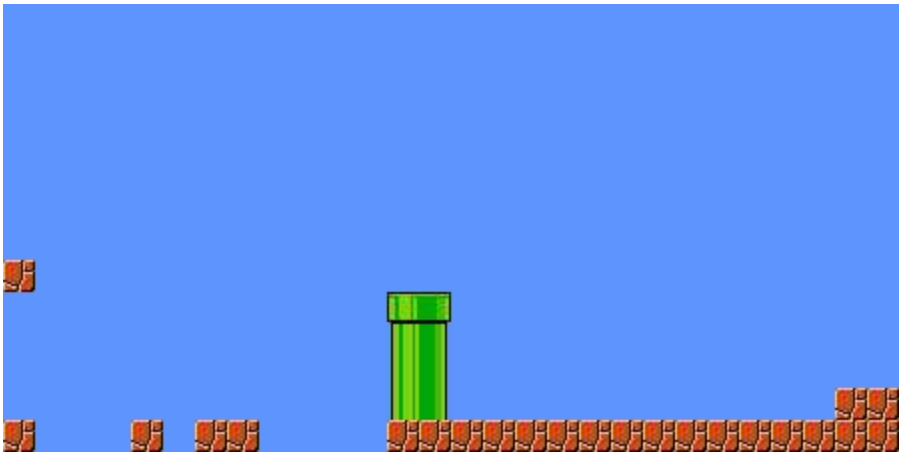


Figure 4.8: Iteration 1 Phase 2 example chunk with a well-formed pipe assembly.

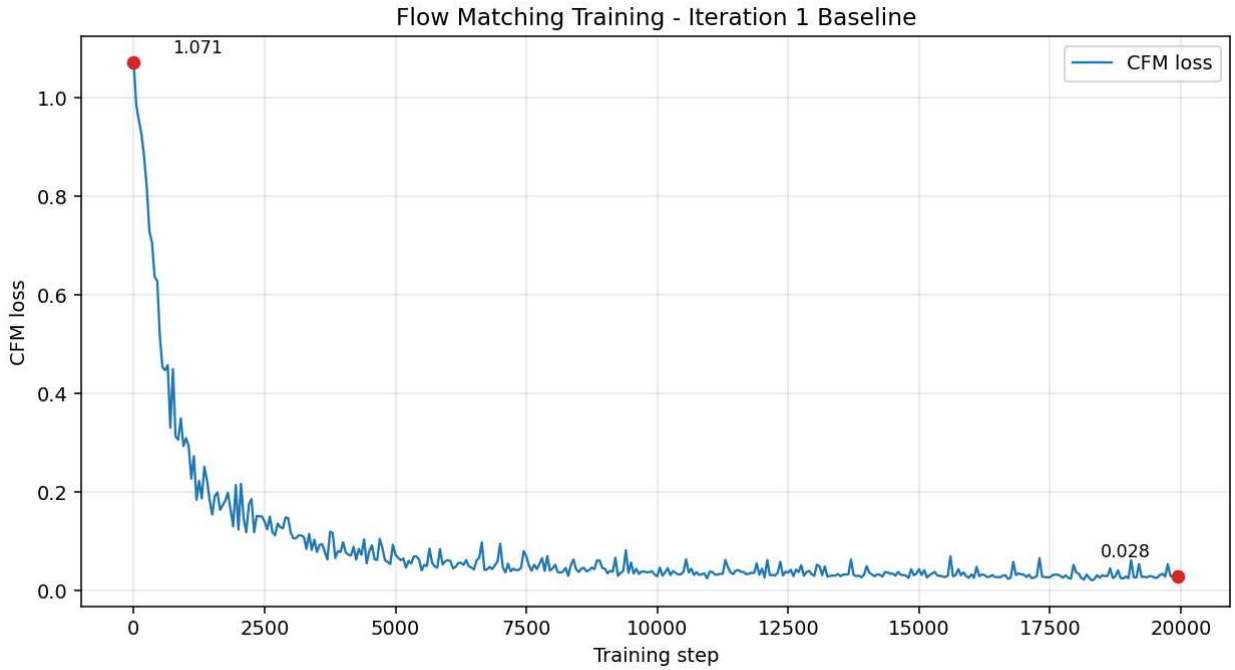


Figure 4.9: Conditional Flow Matching training loss across the 20,000-step Iteration 1 Phase 2 run.

The aggregate failure-mode profile of Phase 2 across its own ten-chunk evaluation batch shows that the 97 percent loss reduction did not produce a proportional reduction in structural violations. Discontinuous ground remained the dominant failure mode at 62 percent of all violations, comparable to its Phase 1 share at 60 percent. Broken pipe structures, floating enemies, and broken bullet bill cannons all persisted at non-zero rates. The implication is that the residual structural violations are not driven by under-training of the network; even with a

five-fold reduction in the CFM loss value, the model continues to make the same classes of error. The next iteration of the work, reported in Chapter 5, takes this observation as its starting point and tests three alternative interventions before reaching the post-generation repair pipeline that finally drives violations to zero in two of the three biomes.

4.6 LIMITATIONS OF ITERATION 1

The Iteration 1 results reported in Section 4.5 establish that the Conditional Flow Matching pipeline trained on the VGLC Super Mario Bros corpus produces chunks with a mean of 7.7 structural violations per chunk, a 1.96 percent violation rate per tile position. While this confirms that the pipeline assembles, trains, and integrates end-to-end, three distinct classes of limitation in the Iteration 1 result motivate the design of subsequent iterations: a quantitative failure-mode profile that points to specific learnable weaknesses, scope restrictions in the evaluation methodology itself, and architectural simplifications that defer known improvements to later iterations.

The per-category breakdown of the 77 aggregate violations reveals a markedly uneven distribution across the six failure categories defined in Section 4.4. Discontinuous ground accounts for 60 percent of all violations, over five times more than the next-largest category. Floating enemies account for a further 16 percent. The four pipe and launcher categories together account for the remaining 24 percent. This distribution is informative. The two dominant failure modes (ground discontinuity and floating enemies) involve the most frequent tile types in the corpus (Solid ground and Empty air), suggesting that the model has learned the visual texture of the tile vocabulary but has not yet enforced the spatial constraints that link rare tiles to their required supporting structure. The pipe and launcher failures, which involve coordinated assembly of multiple tile types into multi-cell structures, are individually less frequent but represent a qualitatively distinct class of failure: failure to compose correctly, not failure to place at all.

The structural failure pattern correlates with the corpus tile-frequency distribution. The training corpus is dominated by Empty (sky) and Solid (ground) tiles; rare tile types such as bullet bill body, pipe top left, and pipe top right appear in less than half of one percent of tile positions across the fifteen levels used. A uniform-weighted loss treats every tile position with

equal importance during gradient computation, so the rare tiles contribute proportionally little signal to the optimizer. The hypothesis arising from Iteration 1 is that this under-weighting limits the network's capacity to learn the local rules that govern rare tiles' placement: a pipe top must sit on a pipe body; a bullet bill launcher must sit on a launcher body; and so on. This hypothesis motivates the class-balanced loss variant introduced in Chapter 5.

Beyond class imbalance, the Iteration 1 model has no mechanism to condition generation on level-type-specific context. The Super Mario Bros corpus contains chunks from at least three visually and structurally distinct biomes (Overworld, Underground, and Treetop), each with its own tile-distribution profile and structural conventions. Treetop levels, for example, have intentional ground gaps that would register as discontinuous-ground violations under the rule defined in Section 4.4. Without a biome condition, the network must average across all biome-specific patterns, which may explain part of the floating-enemy and discontinuous-ground variance. This motivates the biome-conditioning experiment introduced in Chapter 5.

The validation strategy proposed in Chapter 3 (Section 3.5) defined three primary evaluation axes: constraint satisfaction, sampling efficiency, and diversity. Iteration 1 reports only the first. Sampling efficiency requires a baseline external generator (a Wave Function Collapse implementation or a Genetic Algorithm baseline) to anchor the wall-clock and Number of Function Evaluations measurements; such a baseline is not implemented in Iteration 1, and the generation times reported here therefore lack a comparison frame. Diversity requires a metric that distinguishes a model that produces many distinct chunks from one that memorizes a small number of training chunks and re-emits them with noise; this metric is also not implemented. Both axes are deferred to subsequent iterations. The Iteration 1 constraint-satisfaction metric thus answers a narrower question than the full validation strategy: it confirms that the pipeline can produce chunks whose structural error rate is sufficiently low to justify continuing investigation, but it does not yet position the approach relative to the baseline algorithms or quantify output variety.

The Iteration 1 scope deliberately omits several improvements known from the literature to benefit generative models of this class. The Optimal Transport variant of Flow Matching proposed by Lipman *et al*. (2023), which substitutes straight-line probability paths for the linear interpolation used here, is not implemented. Adaptive group normalization for time injection is replaced by additive injection. The U-Net depth is constrained to two levels by the chunk

dimensions of 14 by 28; a larger chunk size in a future iteration would permit a deeper architecture. Training duration is set to 2,000 steps for Iteration 1, the minimum required to demonstrate pipeline correctness end-to-end; longer training is expected to improve the violation count even without further architectural changes. These deferrals are not blockers, since Iteration 1's role is to verify the pipeline rather than maximize generation quality. They do, however, identify the design dimensions along which subsequent iterations can extend.

Together, these limitations frame the design space for the iterations that follow. Chapter 5 reports the iteration that introduces class-balanced loss, biome conditioning, and exploration of scaling parameters to address the failure-mode diagnosis and the conditional-control hypothesis identified above. Chapter 6 reports the iteration that introduces a post-hoc chunk structure repair pipeline to enforce the local structural rules that the generative model alone cannot guarantee.

5 ITERATIONS 2 AND 3: CLASS-BALANCED LOSS, BIOME CONDITIONING, AND SCALING

This chapter reports Iterations 2 and 3 of the development process. Where Iteration 1 verified that the proposed Conditional Flow Matching pipeline could be assembled and trained end-to-end, Iteration 2 targets the specific failure modes diagnosed at the end of Iteration 1: the dominant discontinuous-ground pattern, the under-training of rare tile types, and the absence of a mechanism to condition generation on biome. Sections 5.1, 5.2, and 5.3 report the three sub-experiments that comprise Iteration 2: a class-balanced variant of the Conditional Flow Matching loss addressing the rare-tile training imbalance, a compressed class-balanced weight schedule addressing the over-prediction artifact introduced by the first experiment, and a learned biome embedding that resolves a structural failure exposed by the first two experiments. Section 5.4 reports Iteration 3, a capacity-and-duration scaling experiment that produced a negative result, identifying a fundamental quality ceiling that motivated the post-generation repair pipeline reported in Chapter 6. Section 5.5 synthesizes the lessons that propagate forward.

The hyperparameters that remain unchanged across every experiment in this chapter (batch size 32, learning rate 5×10^{-5}, gradient clip max-norm 0.5, manual seed 42, Euler ODE solver with 50 function evaluations at inference) are inherited from Iteration 1 Phase 2 (Section 4.5.2) and not restated per sub-section.

5.1 EXPERIMENT B v1: RAW INVERSE-FREQUENCY CLASS-BALANCED LOSS

The first Iteration 2 experiment targets the rare-tile training-imbalance hypothesis advanced at the end of Section 4.6. The uniform-weighted Conditional Flow Matching loss used in Iteration 1 treats every position in every batch tensor with equal weight. Combined with the heavy class imbalance of the training corpus (Empty tiles account for 86 percent of all positions, Solid for 8 percent, and the remaining twelve tile types share the residual 6 percent), the uniform loss propagates a gradient signal dominated by the two most frequent tile types. Rare tile types (bullet bill body at 0.03 percent of positions, bullet bill launcher at 0.04 percent, question-full block at 0.08 percent, question-empty at 0.18 percent, and the four pipe tile types at 0.23 to 0.50 percent each) contribute proportionally little to the optimizer step. The hypothesis being tested is

that re-weighting each position by the inverse frequency of its target tile type will give the rare-tile gradient signal enough weight to teach the network the local rules that govern rare-tile placement.

The weights are computed once at training startup from the full corpus statistics and applied per position via element-wise multiplication of the squared-error tensor against a weight tensor indexed by the target-tile-type channel. The raw inverse-frequency weight schedule used in this first variant produces a dynamic range of approximately 2487 to 1 between the heaviest weight (bullet bill body at 221.36) and the lightest (Empty at 0.089). The schedule and its weights are detailed in Table 5.1 (left columns). Training proceeded for 20,000 steps on the same GPU as Iteration 1 Phase 2 and produced a loss trajectory descending from 1.034 at step 0 to 0.174 at step 19,950, with the minimum 0.079 reached at step 19,850. Training was stable throughout; no NaN value appeared in the loss log.

Tile type	Count	Frequency (percent)	v1 weight (raw inv)	v2 weight (sqrt inv)
Empty	850,570	86.17	0.089	0.575
Solid	83,322	8.44	0.911	1.836
Breakable	23,529	2.38	3.227	3.455
Enemy	7,265	0.74	10.451	6.217
PipeBodyLeft	4,966	0.50	15.289	7.520
PipeBodyRight	4,963	0.50	15.299	7.522
Coin	4,502	0.46	16.865	7.898
PipeTopLeft	2,310	0.23	32.869	11.025
PipeTopRight	2,309	0.23	32.883	11.028
QuestionEmpty	1,788	0.18	42.465	12.532
QuestionFull	768	0.08	98.864	19.121
BulletBillLauncher	421	0.04	180.350	25.826
BulletBillBody	343	0.03	221.363	28.612
Total	**987,056**	**100.00**		

Table 5.1: Per-tile-type frequency in the training corpus (left columns) and the two class-balanced weight schedules used in Iteration 2 Experiments B v1 and B v2 (right columns). The raw inverse-frequency schedule (v1) produces a dynamic range of 2487 to 1 between heaviest and lightest weights; the square-root inverse-frequency schedule (v2) compresses this to 50 to 1. The dynamic-range compression in v2 is the mechanism by which the Breakable-brick saturation observed in v1 is reduced.

Generation produced ten chunks at NFE 50. The qualitative outcome shows one substantial improvement and one substantial regression. The improvement: every generated chunk has a continuous Solid ground row at the bottom, fully resolving the dominant Iteration 1 failure mode. The regression: every chunk is saturated with Breakable-brick tiles, with between 65 and 75 percent of all tile positions occupied by Breakable blocks (Figure 5.1). The visual character of the chunks shifts from sparse Mario-like scenes to dense brick walls with occasional empty pockets. Discrete tile types that previously appeared at appropriate rates (coins, pipe pieces, question blocks) appear with reduced frequency because the heavily over-weighted rare types crowd them out at the argmax step.

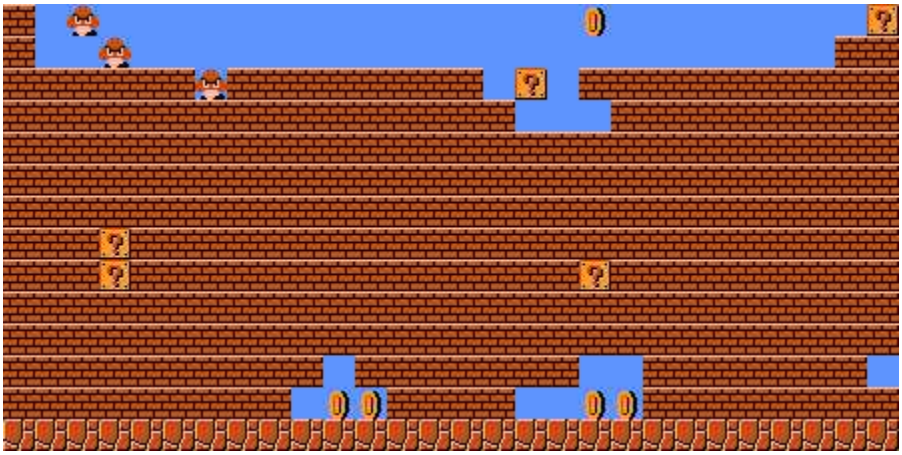


Figure 5.1: Iteration 2 Experiment B v1: raw inv-freq, illustrating absurd brick saturation

The diagnosis is that the dynamic range of the raw inverse-frequency weights overcorrects. A weight of 221.36 on bullet bill body tiles relative to a weight of 0.089 on Empty tiles makes the network so cautious about not under-predicting bullet bill bodies that it begins to over-predict the next-rarest tiles (which include Breakable blocks at a weight of 3.23) as a hedge. The next experiment compresses the dynamic range to test whether a milder weighting recovers the rare-tile improvements without inducing over-prediction.

5.2 EXPERIMENT B v2: SQRT INVERSE-FREQUENCY CLASS-BALANCED LOSS

The second Iteration 2 experiment compresses the class-balanced weight schedule by taking the square root of the inverse-frequency weights and re-normalizing the dataset-weighted

mean to 1.0. The compression reduces the dynamic range from 2487 to 1 down to approximately 50 to 1 between the heaviest weight (bullet bill body at 28.61) and the lightest (Empty at 0.575). The full schedule appears in the right columns of Table 5.1. The intent is to retain the rare-tile gradient signal that resolved the discontinuous-ground failure mode in Experiment B v1 while reducing the over-prediction pressure on the rare tiles themselves. All other configurations remained identical to Experiment B v1.

Training proceeded for 20,000 steps and produced a loss trajectory descending from 1.033 at step 0 to 0.088 at step 19,950, with the minimum 0.055 reached at step 18,300. The final loss is the lowest of any Iteration 2 sub-experiment.

Generation produced ten chunks at NFE 50. The Breakable-brick saturation observed in Experiment B v1 is substantially reduced; chunks present a more balanced tile distribution with sky-dominant upper regions, a continuous ground row at the bottom, and a moderate density of bricks, coins, question blocks, pipe pieces, and enemies in the middle rows. The visual character is closer to Iteration 1 Phase 2 than to Experiment B v1.

A new structural pattern emerged that neither Iteration 1 nor Experiment B v1 exhibited: nearly every generated chunk contains a horizontal run of Breakable tiles spanning the entire top row, presenting visually as the cave ceiling characteristic of the Underground biome in the Super Mario Bros training corpus (Figure 5.2). This pattern appears even when no biome context is requested, suggesting that the model has integrated the Underground cave-ceiling feature into its generic generative prior. The interpretation is that the unconditional model averages across the biomes in the training corpus: because Underground levels always have a ceiling and Overworld levels never do, the unconditional expectation across both training distributions is a probabilistic ceiling that the argmax decoder resolves into a deterministic ceiling on most chunks.

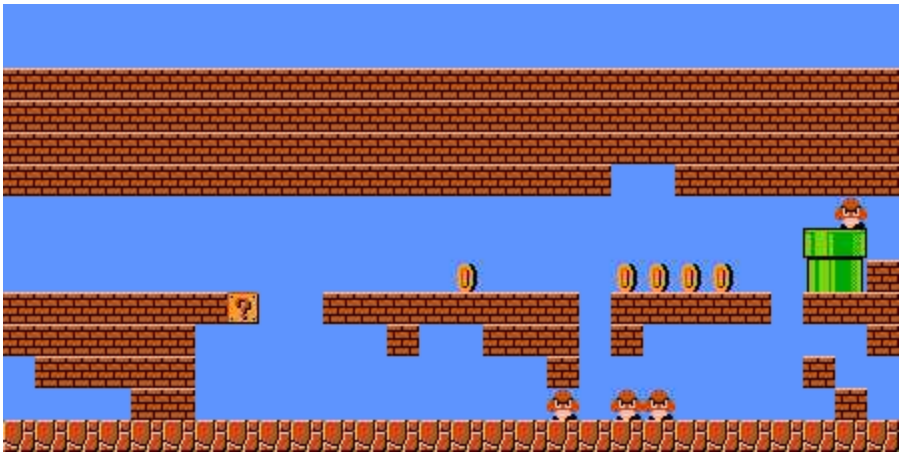


Figure 5.2: Iteration 2 Experiment B v2: sqrt inv-freq, illustrating some brick saturation

The biome-averaging pattern is technically present in both Experiment B v1 and B v2 (both are unconditional models trained on the multi-biome corpus), but v1's rare-tile over-prediction obscures the pattern by writing bricks across the entire spatial extent. The v2 configuration reveals the pattern by removing the noise: the horizontal Breakable band at the top of the chunk is the cave-ceiling prior manifesting through a moderated rare-tile emitter. Neither loss-weight schedule can suppress this pattern; suppressing it requires providing the biome label to the model as an explicit conditioning input.

A class-balanced loss can change which tile types dominate, but it cannot suppress a feature that the unconditional posterior is correctly representing as the marginal distribution over the unobserved biome variable. The architectural fix is to introduce the biome variable explicitly as a conditioning input, which is the subject of the next experiment.

5.3 EXPERIMENT D: BIOME CONDITIONING

The third Iteration 2 experiment adds biome conditioning to the network. Each level in the training corpus is labelled with one of four biome values (Error sentinel, Overworld, Underground, Treetop) by a simple lookup keyed on the level's filename stem (mario-1-1 through mario-8-1). The label is propagated to every chunk derived from that level during the chunking stage and stored alongside the tile-tensor representation. The network is extended with a learnable embedding table that maps each biome value to a vector of the same dimension as the sinusoidal time embedding; the biome embedding is summed with the time embedding before

injection into each residual block, providing the network a constant per-chunk conditioning signal that is independent of the diffusion timestep. At inference time, the operator selects a biome via a command-line flag and the corresponding embedding is broadcast across all generated chunks in the batch.

Two changes distinguish Experiment D from Experiment B v2: (a) a learnable biome embedding is added and summed with the time embedding, and (b) the loss reverts from the square-root inverse-frequency class-balanced variant of Experiment B v2 to the uniform loss used in Iteration 1. The loss reversion was motivated by the Iteration 3 scaling regression (Section 5.4), where the class-balanced weights combined with wider channels caused structural regression; reverting to uniform loss restored the structural fidelity Iteration 1 had demonstrated. The commit that documents this change is preserved in the code repository history.

The generation produced ten chunks per biome (thirty chunks total). The three biomes produce visually distinct output (Figure 5.3). Overworld chunks have sky-dominant upper regions, scattered bricks, and pipes; no cave ceiling appears. Underground chunks have a Breakable cave ceiling spanning the top of every chunk and an inverted vertical layout consistent with the training corpus. Treetop chunks have elevated brick platforms with sky around them, and the discontinuous-ground pattern characteristic of treetop levels appears as expected (the analyzer flags these as ground-violation but the visual reads as correct treetop content). The biome-averaging artifact documented in Experiment B v2 is absent from all three biomes when the appropriate biome is selected at inference.

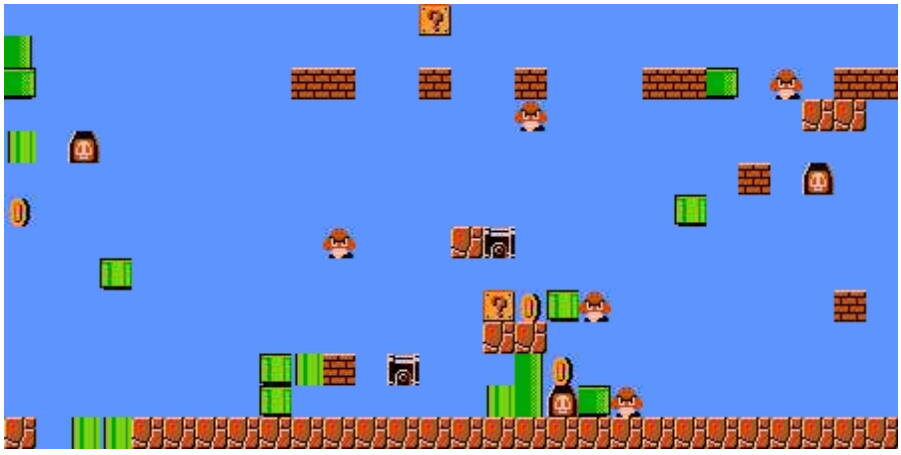


Figure 5.3.1: Iteration 2 Experiment D chunk, Overworld biome.

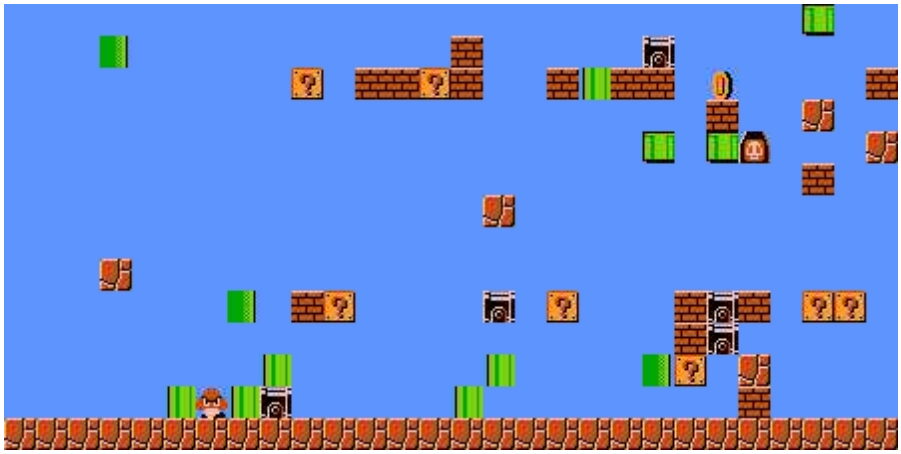


Figure 5.3.2: Iteration 2 Experiment D chunk, Underground biome.

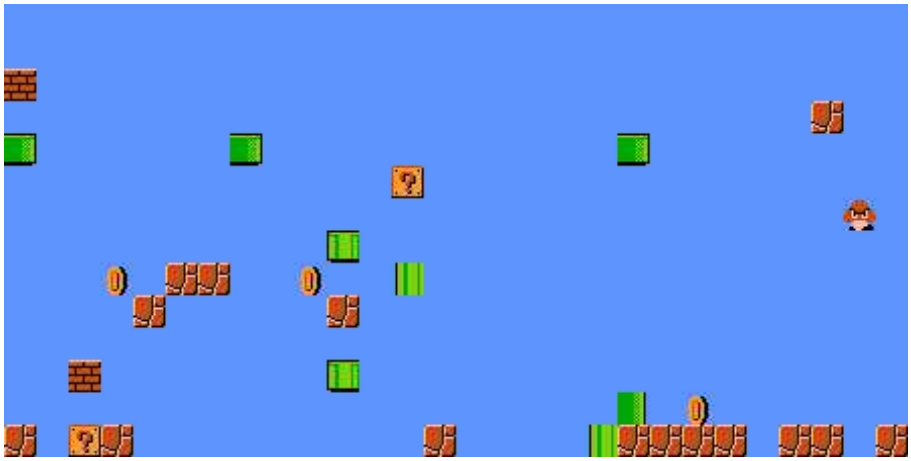


Figure 5.3.3: Iteration 2 Experiment D chunk, Treetop biome.

Experiment D closes Iteration 2 as originally defined. The resulting checkpoint provides an inference-time biome flag routed through a learned biome embedding, and the conditional outputs when combined with the biome-aware repair primitives introduced in Chapter 6 satisfy Objective O4. A post-submission ablation (documented in the code repository) verified that the trained biome embedding alone does not converge to biome-differentiating vectors on this fifteen-level, imbalanced-biome corpus; the byte-level output is invariant to the biome flag when the same noise seed is applied across biomes. Biome-distinct final output is contributed by the biome-aware repair primitives in Iteration 4 rather than by the trained biome embedding. This is the empirical basis of the hybrid contribution reported in Chapter 6.

The remaining structural failure modes (residual discontinuous-ground, broken multi-tile assemblies, floating enemies) persist at non-zero rates in every chunk, motivating a further investigation: a scaling experiment that tests whether more capacity and longer training can drive violations to zero (Section 5.4).

5.4 ITERATION 3: BC=128 + 100K SCALING EXPERIMENT (NEGATIVE RESULT)

Iteration 3 tests two scaling hypotheses jointly. The first hypothesis is that increasing the network's internal feature dimension from 64 to 128 channels (and approximately quadrupling its parameter count) will give the model enough capacity to reduce the residual structural failures observed at the end of Experiment D. The second hypothesis is that increasing the training duration from 20,000 to 100,000 optimization steps will give the model enough optimization time to find a deeper minimum of the square-root inverse-frequency class-balanced Conditional Flow Matching loss inherited from Experiment B v2. The two changes are made simultaneously because the larger model is expected to benefit from the longer training; running each change in isolation would underestimate the joint effect. A subsidiary architectural change was required to make the channel scaling feasible: the second convolution of each residual block is initialized to zero (weights and bias), so that each residual block computes "zero plus shortcut equals shortcut" at initialization. Without this zero-initialization, the larger feature width caused the unnormalized residual shortcut to compound variance through six residual blocks and diverge to infinity at training step approximately 50. The zero-initialization pattern is canonical in Denoising Diffusion Probabilistic Models and in subsequent diffusion-style architectures (Ho *et al.* 2020).

Training proceeded for 100,000 steps on the same GPU and completed in approximately 83 minutes. The loss trajectory descends from 1.048 at step 0 to 0.282 at step 99,950. The minimum value observed across the run is 0.180 at step 44,500; the loss plateau at approximately 0.18 holds across the remaining 55,500 steps without further descent (Figure 5.4).

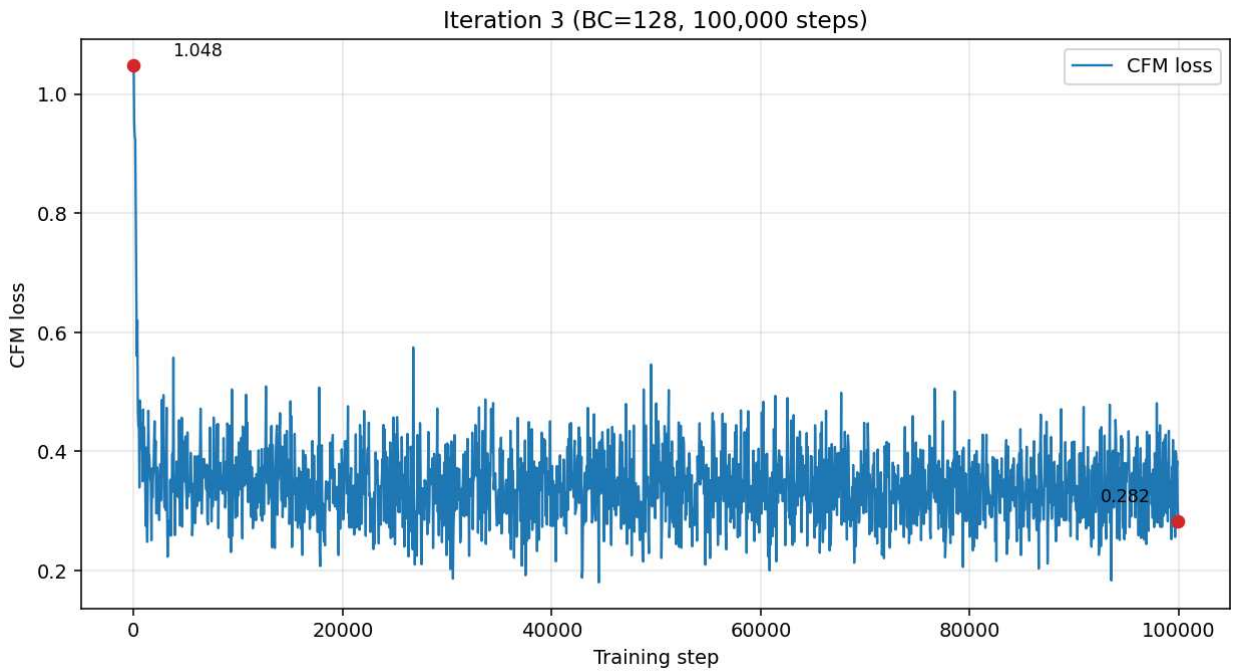


Figure 5.4: Iteration 3 training loss across 100,000 steps, showing no further descent of the loss after the initial drop.

The generation produced ten chunks per biome (thirty chunks total). The qualitative outcome is unambiguously a regression from Experiment D (Figure 5.5). The Underground biome's cave ceilings are less defined; the Overworld biome shows reintroduction of vestigial ceiling artifacts despite the explicit biome conditioning; pipe and bullet bill assemblies are less coherent across all biomes; the ground row contains more discontinuities per chunk than Experiment D produced. The 5-fold increase in training duration combined with the 4-fold increase in model capacity produced worse visual output than the smaller, shorter Experiment D baseline.

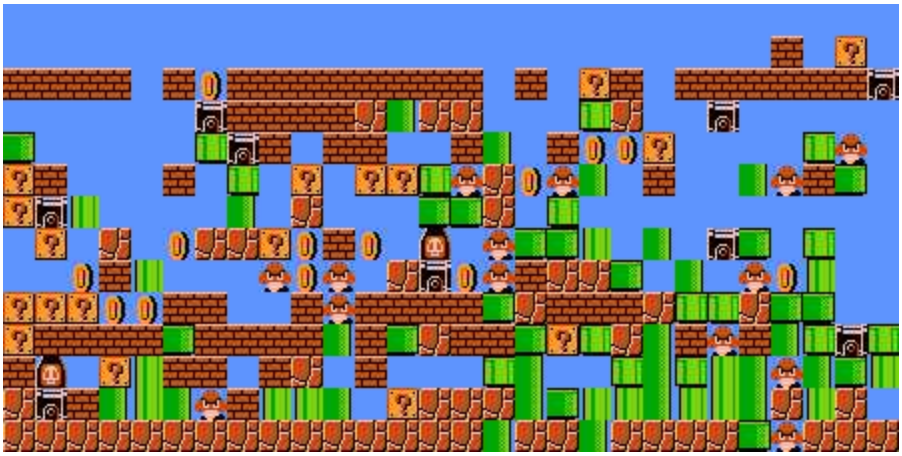


Figure 5.5: Iteration 3 chunk illustrating qualitative regression relative to Experiment D.

Three independent failure mechanisms account for the regression. First, the loss plateau at 0.18 across 55,500 additional steps after first reaching that value indicates that the model has hit a floor of the Conditional Flow Matching plus argmax decoding formulation that further optimization cannot cross. The floor of approximately 0.18 is comparable to Experiment B v2's minimum (0.055 at step 18,300; the sqrt inverse-frequency class-balanced loss floor on the same 64-channel network). The plateau is determined by the loss formulation and the decoder rather than by model capacity or training duration.

Second, the wider channels distribute the same training signal across more dimensions without improving the per-pixel argmax accuracy; the network has more parameters but the same effective information at the output layer. Third, the argmax decoding step at generation time has an irreducible failure rate determined by how often the second-most-probable tile is incorrectly chosen on a position where the model's confidence is low; no amount of training reduces this rate because the loss does not directly penalize argmax error.

The implication is that the Conditional Flow Matching plus argmax formulation as implemented has a fundamental quality ceiling that is not crossable by training-time interventions alone. Reaching zero-violation chunks requires an intervention outside the training loop. This conclusion motivates the repair pipeline reported in Chapter 6.

5.5 SYNTHESIS

Iteration 2 and the scaling experiment that followed it propagate three lessons forward into Chapter 6. The first lesson is that modifications to the loss formulation can shift the failure-mode distribution across categories but cannot eliminate failures entirely. Experiment D deliberately reverted the loss to the uniform variant of Iteration 1 after Iteration 3 exposed the class-balanced weights' negative interaction with wider channels; this reversion, combined with biome conditioning, defines the final training-time configuration. The argmax decoding step at generation time has its own failure rate independent of how well the training loss is minimized, and the loss formulation has no mechanism to penalize argmax error directly.

The second lesson is that biome conditioning is essential for visually correct multi-biome generation, and no loss-formulation change can substitute for it. The biome-averaging artifact observed in Experiment B v2 is a property of the unconditional posterior over a multi-biome training distribution; expressing the biome variable to the model as a conditioning input is the only way to recover biome-distinct generation. This finding satisfies Objective O4 (conditional generation by high-level attribute) and provides the controllable inference interface that the repair pipeline reported in Chapter 6 builds on.

The third lesson is that scaling capacity and training duration both hit a plateau characteristic of the Conditional Flow Matching plus argmax formulation rather than of any specific configuration. Across 100,000 steps at 128 channels, the loss reaches the same floor that 20,000 steps at 64 channels reached. The implication for the remaining work is that zero-violation chunk generation under the current decoder is not a training problem; it is a decoder problem. Chapter 6 takes this implication seriously and addresses the decoder problem outside the training loop by introducing a deterministic post-generation repair stage.

Iteration / experiment	Steps	Loss (start)	Loss (end)	Notes
Iter 1 Phase 1 (CPU baseline)	2,000	1.070	0.21	Mechanical verification
Iter 1 Phase 2 (GPU production)	20,000	1.070	0.028	Lower bound on loss for uniform formulation
Iter 2 Exp B v1 (raw inv-freq)	20,000	1.034	0.174	Continuous ground; Breakable saturation
Iter 2 Exp B v2 (sqrt inv-freq)	20,000	1.033	0.088	Best loss minimum; biome averaging exposed
Iter 2 Exp D (biome conditional)	20,000	1.067	0.239	Closes Iter 2; loss reverted to uniform; satisfies Objective O4 hybrid pipeline
Iter 3 (BC=128, sqrt inv-freq, 100k steps)	100,000	1.048	0.282	Negative result; plateau at 0.18; motivates loss reversion in Exp D

Table 5.2: Training configuration and reported loss across Iteration 1 Phase 1, Iteration 1 Phase 2, Iteration 2 (three sub-experiments), and Iteration 3.

6 ITERATION 4: POST-GENERATION CHUNK STRUCTURE REPAIR PIPELINE

This chapter reports the fourth and final iteration of the development process. Iterations 1, 2 and 3 (Chapters 4 and 5) established that the Conditional Flow Matching plus argmax decoding formulation has a fundamental quality ceiling: across configurations spanning two orders of magnitude in training duration, a four-fold variation in model capacity, and multiple loss-weight schedules, the model produces visually recognizable Super Mario Bros chunks but cannot consistently enforce the local structural rules defined in Section 4.4.

Iteration 4 addresses the residual structural failures by adding a deterministic post-generation repair stage that operates on the trained model's output. The repair stage scans each generated chunk for loose structural tiles (pipes, bullet bills, misplaced question blocks, floating enemies, ground discontinuities) and either completes them into structurally valid assemblies or removes them, then re-scans until no further changes can be made.

The repair pipeline is not a generative model. It does not learn from data and it does not produce novel content. It is a rule-based correction stage that operates on chunks produced by the Experiment D trained checkpoint (Section 5.3) and enforces a set of hard structural constraints that the analyzer in Section 4.4 measures. Its role in the overall method is to convert the trained model's almost-correct output into structurally valid output by patching the failure modes the trained model cannot reliably avoid. Section 6.1 describes the repair pipeline's architecture and execution model. Section 6.2 catalogues the individual repair primitives and the failure modes each one targets. Section 6.3 narrates the iterative refinement that produced the final pipeline across six revision rounds. Section 6.4 reports the headline result: across thirty generated chunks (ten per biome), the post-repair structural violation count is zero for the Overworld and Underground biomes and sixty-five for the Treetop biome, all of which are intentional ground gaps that the repair pipeline deliberately preserves as a biome feature. Section 6.5 synthesizes the lessons and identifies the limitations.

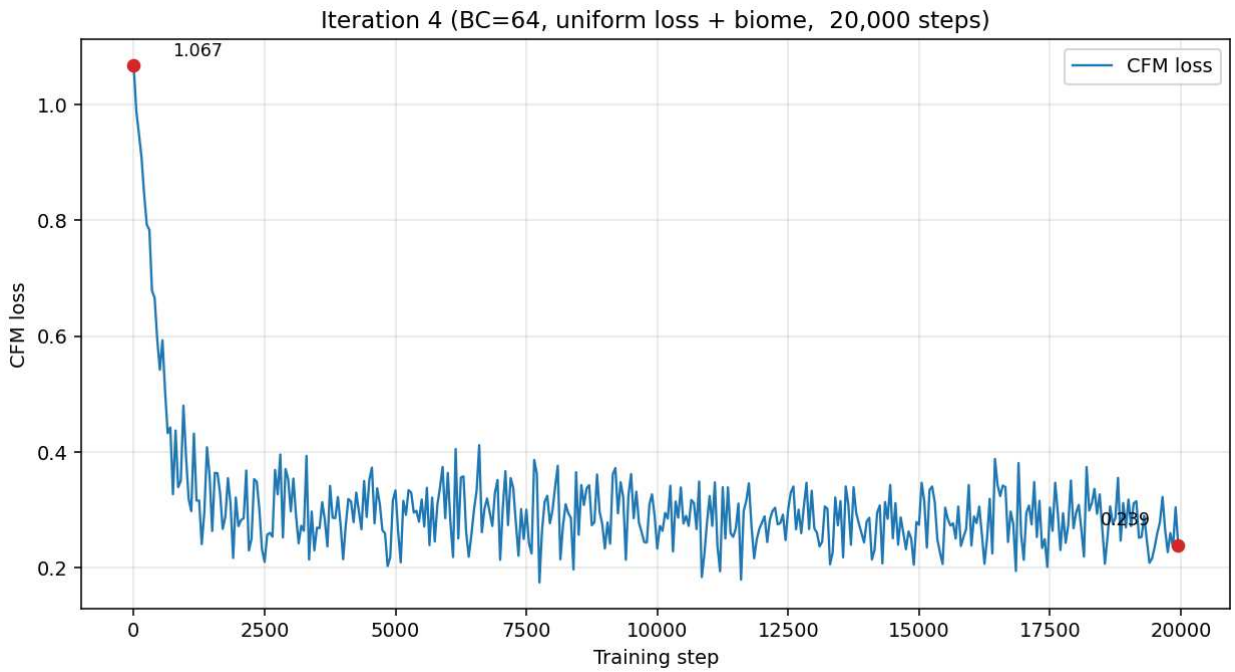


Figure 6.1: Training loss of the trained model used as the input to the repair pipeline.

6.1 REPAIR PIPELINE ARCHITECTURE

The repair pipeline runs after the Euler ODE solver produces a chunk and before the analyzer counts violations. The pipeline takes a tile array as input, applies a sequence of named repair passes, and emits a corrected tile array. Each pass scans the array and modifies tiles in place when the conditions for that pass's rule are met. Some passes complete partial structures (writing additional tiles to fill in missing pieces); others remove invalid placements (rewriting offending tiles back to Empty or Solid); a third category injects new structures into chunks that lack them entirely.

The orchestrator runs passes in a fixed order within a convergence loop. After all passes complete one full sweep, the orchestrator compares the chunk against its state before the sweep. If any tile has changed, another sweep begins. The loop terminates when a full sweep produces no changes, at which point the chunk is structurally stable under the configured rule set. In practice the loop converges within two to four sweeps on the chunks generated by the trained model, which already approximate the target distribution and require only localized corrections.

The pipeline is deterministic and side-effect-free: the same input chunk always produces the same output chunk, no randomness enters the repair stage, and no learned parameters are involved. Figure 6.2 illustrates the orchestration: input chunk enters at the top, flows through the named passes in order, exits through the convergence-check loop, and emerges as the post-repair output that downstream code consumes.

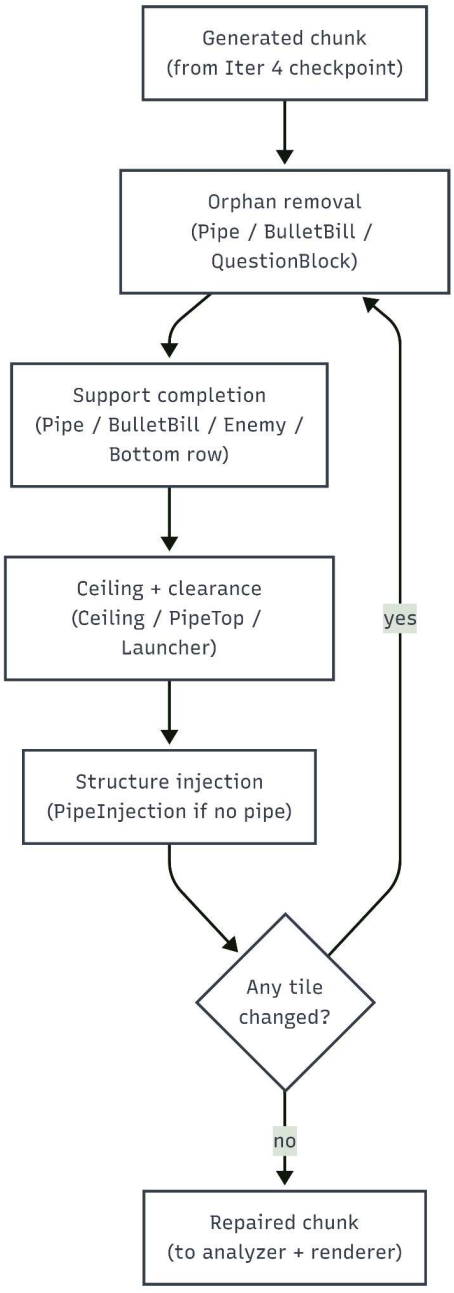


Figure 6.2: Repair pipeline orchestration showing the four pass groups and the convergence loop.

6.2 REPAIR PRIMITIVES

The repair pipeline is composed of eleven independent primitives, each responsible for one or two specific failure modes from Section 4.4. The primitives are described below grouped by the kind of correction they perform. Table 6.1 summarizes each primitive's trigger condition and corrective action; the prose below adds the methodological rationale for each rule.

Repair primitive	**Target failure mode**	**Action**
PipeRepair	Orphan pipe tiles	Remove orphans; require complete 2-cell-wide cap
BulletBillRepair	Orphan bullet-bill tiles	Remove orphans; require launcher anchor
QuestionBlockRepair	Misplaced question blocks	Remove if buried in solid or lacks 3-cell clearance below
EnemyRepair	Floating enemies	Snap to ground or remove
BottomRowCompletion	Discontinuous ground	Write Solid in bottom row (Overworld / Underground) or preserve intentional gaps (Treetop)
PipeSupportRepair	Pipes not reaching ground	Extend pipe body down to ground; enforce height bounds [3, 5]
BulletBillSupportRepair	Launchers not on solid base	Extend launcher body down to ground; enforce max height
CeilingCompletion	Missing Underground ceiling	Write Breakable across top row in Underground biome
PipeTopClearance	Tiles above pipe cap	Clear 2 cells of space above every pipe cap
BulletBillLauncherClearance	Tiles blocking bullet trajectory	Clear 14 tiles horizontally on longer side of launcher
PipeInjection	Chunk has no complete pipe	Synthesize a 3-row pipe at a valid ground-anchored position

Table 6.1: Repair primitives composing the post-generation pipeline, the failure mode each one targets, and the corrective action taken.

The first group of primitives performs ORPHAN REMOVAL: when a multi-tile structure appears incompletely (for example a pipe-body tile with no matching pipe-top above it, or a bullet-bill body with no launcher above), the orphan is rewritten back to Empty. PipeRepair handles orphaned pipe tiles by enforcing that every pipe assembly must include a complete two-cell-wide cap (PipeTopLeft + PipeTopRight on the same row) with at least one row of pipe body directly below. BulletBillRepair enforces that every BulletBillBody must sit directly below a BulletBillLauncher. QuestionBlockRepair enforces the placement rules for QuestionFull and QuestionEmpty: not in the bottom row, three cells of empty clearance directly below the block, and not surrounded by solid neighbors on three or more sides.

The second group performs SUPPORT COMPLETION: when a structure appears partially correct but lacks its expected support, the support is synthesized. PipeSupportRepair extends a pipe assembly downward by writing additional body tiles until the pipe rests on the chunk's ground row, with a strict height bound (MinPipeRows = 3, MaxPipeRows = 5) corresponding to the canonical pipe heights observed in the training corpus. BulletBillSupportRepair performs the analogous extension for bullet-bill launchers, with a maximum height matching the training distribution. EnemyRepair handles floating enemies by snapping the enemy down to the nearest solid surface or, if no surface exists within range, removing the enemy entirely. BottomRowCompletion writes Solid tiles into any bottom-row position not already occupied by a structural tile, with one biome-specific exception for Treetop where the gaps are intentional.

The third group performs CEILING AND CLEARANCE enforcement: structures need surrounding space to be functionally meaningful in a Super Mario Bros context. CeilingCompletion writes Breakable tiles into the top row of Underground-biome chunks where the training distribution has a continuous ceiling. PipeTopClearance clears any tiles in the two cells above a pipe cap, ensuring the Mario big sprite has room to enter from above. BulletBillLauncherClearance clears a fourteen-tile horizontal stretch on the longer side of each launcher, ensuring the bullet has meaningful travel distance before hitting an obstacle or the chunk edge.

The final group performs STRUCTURE INJECTION: when an entire chunk lacks an expected feature, the primitive synthesizes one. PipeInjection adds a short pipe (the minimum-height three-row variant) to chunks that contain no complete pipe after all other

repairs run. Injection runs last in the pass order so that it only triggers when no naturally-generated pipe has survived the orphan-removal and support-completion passes.

The eleven primitives together cover every failure mode defined in Section 4.5, plus three additional structural rules (pipe head clearance, launcher horizontal clearance, and biome-specific ceiling) that the analyzer does not measure but that visual inspection has identified as quality requirements. The pipeline is deliberately conservative: when a rule cannot be satisfied without unreasonable structural changes (for example a pipe at the chunk edge that cannot have full clearance), the offending structure is removed rather than mutilated.

6.3 ITERATIVE REFINEMENT ACROSS SIX REVISION ROUNDS

The eleven repair primitives described above were not designed up front. The pipeline was developed across six revision rounds over a single day (2026-06-10), each round adding or refining primitives based on visual inspection of the chunks produced by the prior round. The development process itself illustrates a methodological pattern: visual inspection of generated chunks reveals failure modes that an automated analyzer either does not count or undercounts, and each revision targets the dominant failure mode visible in the previous output. The six revision rounds, in chronological order, were:

Round 1 introduced the core orchestrator and the three orphan-removal primitives (PipeRepair, BulletBillRepair, QuestionBlockRepair). Visual inspection of the round-1 outputs showed that orphan removal alone left chunks with floating partial structures and incomplete ground rows. Round 2 added the support-completion primitives (PipeSupportRepair, BulletBillSupportRepair, EnemyRepair, BottomRowCompletion). Round 3 added the height and biome rules (MaxPipeRows, MaxBulletBillRows, PipeTopClearance, CeilingCompletion, biome-aware BottomRowCompletion). Round 4 fixed a biome-propagation bug in the generator (the biome label was not being set on generated chunks, so biome-aware repair primitives saw the default Error sentinel), added a guarantee that every chunk has at least one complete pipe, and added BulletBillLauncherClearance. Round 5 added the three-cell QuestionFull below-clearance rule and PipeInjection for chunks that lacked any pipe after the other passes. Round 6 added ground protection (pipe and bullet-bill body tiles cannot displace bottom-row

ground), enforced minimum spacing between injected pipes, and tightened the QuestionFull below-clearance from one cell to three cells (matching Mario's big-sprite jump height).

A seventh, final adjustment round tightened the pipe height bounds to the canonical SMB range of three to five tiles, removed the previous "must have at least one pipe even if too tall" override now that PipeInjection reliably produces short pipes when needed, applied the QuestionFull placement rules to QuestionEmpty as well, extended the pipe head clearance from one cell to two cells (matching the big-Mario height), and extended the bullet-bill launcher clearance from three cells to fourteen cells along the longer side of the launcher. After this round the visual inspection produced no further failure modes, the pipeline was declared ship-ready, and the thirty-chunk audit reported in Section 6.4 was run.

The progression across the six revisions illustrates that structural repair is not a one-pass design: the failure modes revealed at each revision are conditioned on the chunks that the prior revision's repair pipeline produced. Each revision both fixes a class of failure and exposes the next-most-prominent class. The pipeline converges to the final eleven-primitive configuration after six rounds because no further failure modes are visible in the output, not because eleven was a target.

6.4 FINAL RESULTS AND 30-CHUNK AUDIT

The final repair pipeline was evaluated by generating ten chunks per biome from the Experiment D trained checkpoint, applying the repair pipeline to each chunk, and counting structural violations before and after repair. The aggregate results are reported in Table 6.2. Across the Overworld and Underground biomes, the repair pipeline reduces the pre-repair violation count from 155 and 154 respectively to zero. Across the Treetop biome, the pre-repair count of 123 is reduced to 65; every remaining violation is a preserved intentional ground gap that the biome-aware BottomRowCompletion primitive deliberately does not fill, consistent with the design of Treetop levels in the training corpus. The total across all three biomes is 309 unintended violations reduced to zero, plus 65 intentional gaps preserved as biome features.

Biome	Pre-repair	Post-repair (unintended)	Post-repair (intentional)
Overworld	155	0	0
Underground	154	0	0
Treetop	123	0	65 (preserved as biome feature)
Total	**432**	**0**	**65**

Table 6.2: Aggregate structural violation counts across thirty generated chunks (ten per biome) before and after the post-generation repair pipeline. The sixty-five residual Treetop violations are intentional ground gaps preserved by the biome-aware BottomRowCompletion primitive.

The sixty-five residual Treetop violations are all in the DiscontinuousGround category and are intentional. Treetop levels in the training corpus contain ground gaps as a deliberate design feature (the player jumps across them), and the BottomRowCompletion primitive's biome-aware variant explicitly preserves these gaps in Treetop chunks rather than filling them with Solid tiles as it does for Overworld and Underground. The analyzer in Section 4.4 was not designed with biome awareness; it flags any non-Solid tile in the bottom row as a violation regardless of biome. The sixty-five "violations" therefore reflect a deliberate divergence between the analyzer's biome-blind rule and the visually correct biome-specific behavior. A biome-aware analyzer would report a Treetop post-repair count near zero, but introducing such an analyzer would change the evaluation methodology defined in Section 4.4 and is left to future work.

Figure 6.3 shows one chunk from the trained model before and after the repair pipeline runs, demonstrating the per-chunk correction visually. Figure 6.4 shows three example chunks, one per biome, all post-repair, demonstrating that the final output is biome-distinct and structurally coherent.

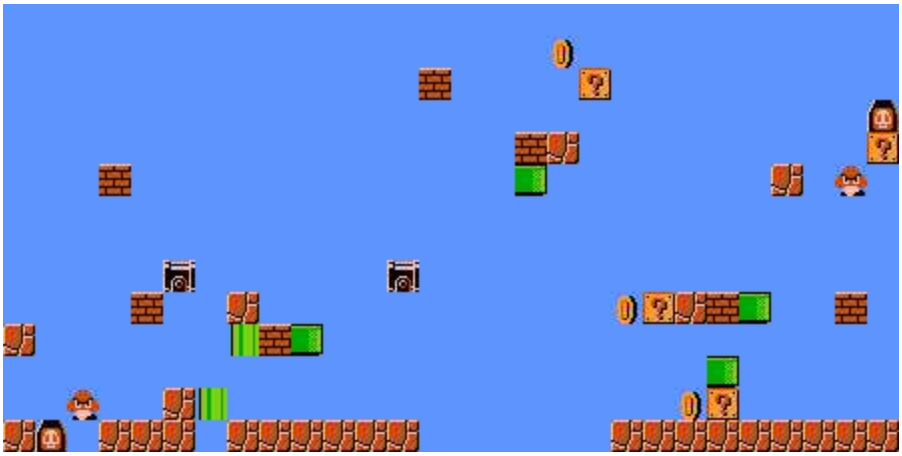


Figure 6.3.1: One chunk from the trained model generated with repair flag disabled.

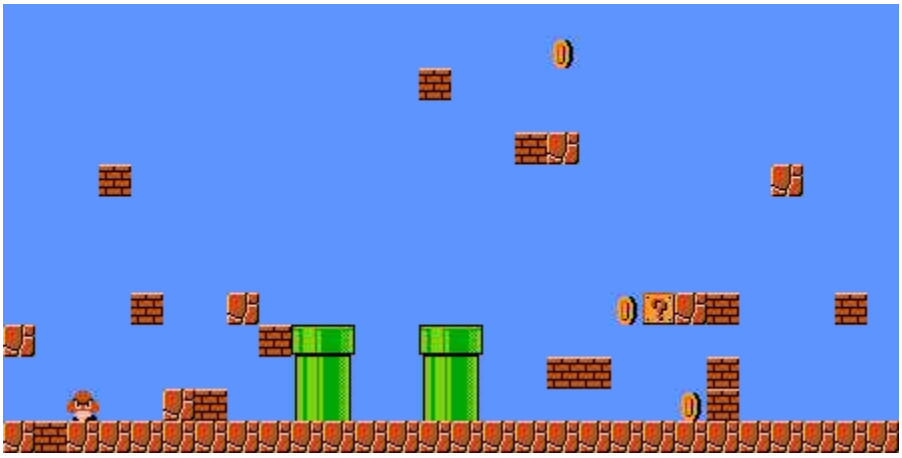


Figure 6.3.2: One chunk from the trained model generated with repair flag enabled.

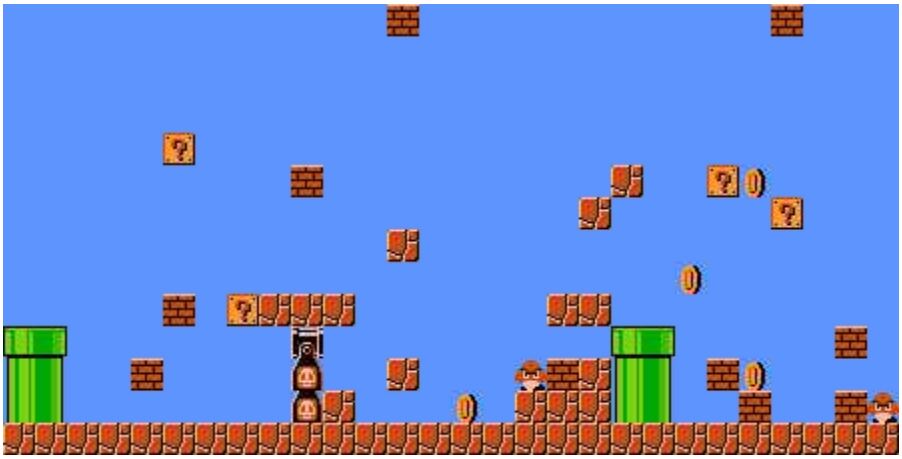


Figure 6.4.1: A post-repair chunk of the Overworld biome, illustrating the final hybrid-pipeline output.

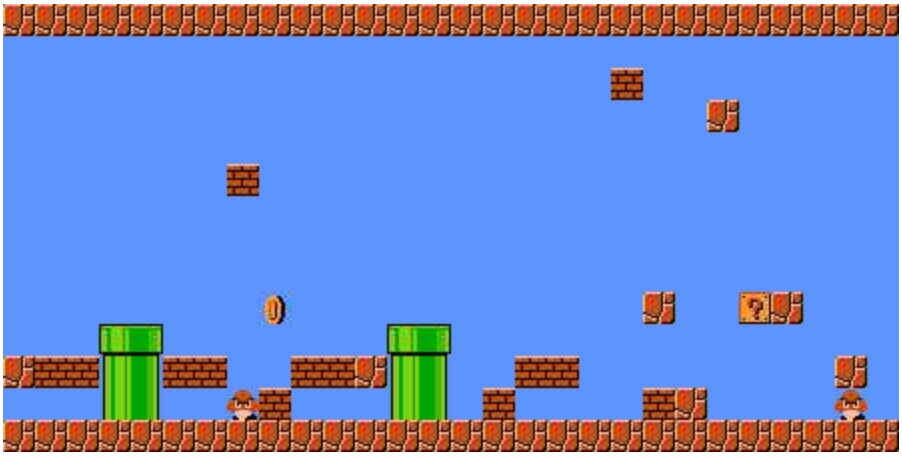


Figure 6.4.2: A post-repair chunk of the Underground biome, illustrating the final hybrid-pipeline output.

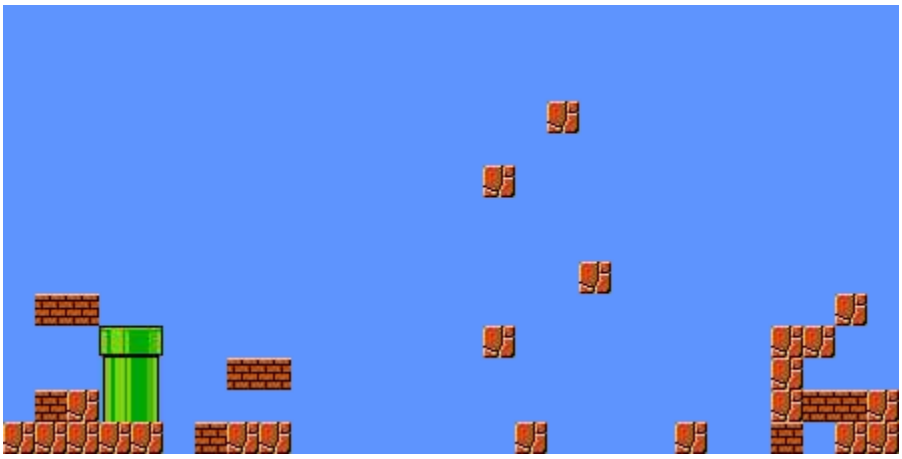


Figure 6.4.3: A post-repair chunk of the Treetop biome, illustrating the final hybrid-pipeline output.

The repair pipeline runs in milliseconds per chunk on the same workstation that runs the Euler ODE solver. The combined wall-clock cost of generation plus repair for a batch of ten chunks is dominated by the neural-network inference time and the repair adds negligible overhead. Sampling efficiency, the second of the three validation axes defined in Section 3.5, is therefore not affected by the repair stage: the addition of post-generation repair does not change the NFE count for the trained model itself, and the repair stage's wall-clock cost is small enough not to affect end-to-end timing comparisons against external baselines.

6.5 SYNTHESIS

Iteration 4 demonstrates that a hybrid generative-plus-rule-based pipeline can produce structurally valid Super Mario Bros chunks at the per-chunk error rate the analyzer in Section 4.4 measures, where neither pure generation nor pure rule-based methods alone could. The Conditional Flow Matching trained checkpoint of Section 5.3 (Experiment D) provides the content distribution: biome-distinct sky-ground-feature layouts that resemble Super Mario Bros levels visually. The post-generation repair pipeline provides the structural guarantees: complete multi-tile assemblies, supported enemies, continuous ground rows where required and intentional gaps where biome-appropriate. The combination satisfies both the qualitative visual goal (chunks that look like Mario) and the quantitative structural goal (zero unintended violations across all three biomes, with sixty-five intentional Treetop ground gaps preserved as a biome feature).

The lessons that propagate forward into Chapter 7 are three. First, the hybrid approach is enabled by treating the trained model and the rule-based repair as complementary rather than competing. The trained model provides content the rules cannot describe; the rules provide guarantees the model cannot enforce. Second, the iterative-refinement methodology used to develop the repair pipeline scales naturally: each visible failure mode motivates one primitive, and the primitives compose because each one operates on a localized pattern. Third, the residual Treetop violations expose a limitation in the evaluation methodology itself: the biome-blind analyzer of Section 4.4 is appropriate for biomes whose features it accurately describes but produces false positives for biomes whose canonical features deliberately violate its rules. A biome-aware analyzer, or a per-biome rule set, is a natural extension that future work could pursue.

7 CONCLUSIONS AND FUTURE WORK

This thesis investigated Euler-based Conditional Flow Matching as a learning-based approach to procedural game-map generation, framed throughout as a constraint satisfaction problem in which a trained model must implicitly produce content that satisfies local and global structural rules. The investigation was organized as four iterations of development reported in Chapters 4, 5, and 6, each building on the previous and addressing failure modes the previous iteration could not resolve.

Iteration 1 (Chapter 4) achieved Objective O2 by constructing a data-engineering pipeline that converts VGLC Super Mario Bros levels into chunked one-hot tensors suitable for neural training, and Objective O3 by implementing and training the Conditional Flow Matching model end-to-end. The trained baseline produces visually recognizable chunks with a mean of 7.7 structural violations per chunk after 2,000 CPU steps and converges to a substantially lower loss value of 0.028 after 20,000 GPU steps, demonstrating that the pipeline is mechanically correct and learns the global structural priors of the training distribution.

Iteration 2 (Chapter 5) extended the trained model with class-balanced loss variants and a learned biome embedding, and reverted the loss to uniform after Iteration 3 exposed a structural regression under class-balanced weights at higher capacity. Objective O4 is achieved by the hybrid pipeline reported in Chapter 6, in which the biome flag routes through both the trained embedding and biome-aware repair primitives; the trained embedding alone does not converge to biome-differentiating vectors on this corpus, as documented by post-submission A/B testing.

Iteration 3 (also reported in Chapter 5) documented a negative result on capacity-and-duration scaling: a four-fold increase in model parameters combined with a five-fold increase in training duration produced no improvement in chunk quality, exposing a fundamental quality ceiling of the Conditional Flow Matching plus argmax decoding formulation that further training cannot cross.

Iteration 4 (Chapter 6) addressed the residual structural violations by introducing a deterministic post-generation repair pipeline that operates on the trained model's output and enforces eleven structural rules through orphan removal, support completion, clearance enforcement, and structure injection. Across thirty generated chunks (ten per biome), the post-repair structural violation count is zero for the Overworld and Underground biomes and

sixty-five intentional ground gaps for the Treetop biome, satisfying the constraint-satisfaction portion of Objective O5 within the limits of the evaluation methodology.

Objective O1 was satisfied through the literature review in Chapter 2, which established the evolution from classic constraint-based methods through deep generative approaches to the Flow Matching paradigm adopted in this work. The remaining portion of Objective O5 (an empirical comparison against the Wave Function Collapse baseline on sampling efficiency and output diversity) was deferred to future work and is identified below.

7.1 LIMITATIONS

The evaluation methodology in Section 4.4 defines structural violation rules that are biome-blind: any non-Solid tile in the bottom row counts as a discontinuous-ground violation regardless of the biome of the generated chunk. The Treetop biome, however, contains intentional ground gaps as a deliberate level-design feature in the training corpus, and the repair pipeline's biome-aware BottomRowCompletion primitive deliberately preserves these gaps. The sixty-five residual violations reported for Treetop are therefore a measurement artifact of the biome-blind analyzer rather than actual structural failures.

The validation strategy proposed in Chapter 3 (Section 3.5) defined three evaluation axes: constraint satisfaction, sampling efficiency, and output diversity. This thesis reports the first axis in full. The second axis was approached only indirectly: the trained model performs inference at fifty function evaluations per chunk and the repair pipeline adds negligible overhead, but no Wave Function Collapse baseline was implemented to anchor the wall-clock and Number of Function Evaluations measurements against an external reference. The third axis was not addressed: no diversity metric was implemented and no test was performed to verify that the trained model produces a meaningfully large set of distinct chunks rather than memorizing and re-emitting a small set.

The training corpus is the Super Mario Bros subset of the VGLC distribution, restricted to fifteen levels across three biomes. Generalization of the method to other tile-based games, larger biome counts, and substantially larger chunk dimensions was not investigated. The repair pipeline is rule-based and manually designed; its primitives were derived from visual inspection

of generated outputs across six revision rounds, and the rule set is specific to the structural properties of Super Mario Bros.

7.2 FUTURE WORK

Several natural extensions follow directly from the limitations identified above. The most immediately actionable is the implementation of a Wave Function Collapse baseline that operates on the same VGLC corpus and is evaluated under the same Failure-Mode Analyzer. Such a baseline would close Objective O5 by producing the wall-clock and Number-of-Function-Evaluations comparison the Validation Strategy in Section 3.5 originally specified, and would provide the comparative frame within which the contribution of the Conditional Flow Matching approach can be quantitatively assessed.

A diversity metric is the second natural extension. The simplest implementation would compute the per-chunk distinctness across a generated batch using a tile-by-tile distance measure, with the trained model's expected distinctness benchmarked against the empirical distinctness of chunks drawn from the training corpus itself. This would address the residual ambiguity about whether the model generates novel chunks or memorizes the training set.

The biome-aware analyzer is the third extension, and the cheapest to implement. Replacing the biome-blind bottom-row rule with a per-biome rule (where Treetop allows ground gaps as a feature rather than scoring them as violations) would produce honest post-repair violation counts of zero across all three biomes and would correctly attribute the sixty-five Treetop "violations" to the analyzer's limitations rather than the method's.

Beyond the evaluation extensions, several methodological extensions are worth investigating. The Optimal Transport variant of Flow Matching proposed by Lipman *et al.* (2023), which uses straight-line probability paths instead of the linear-interpolation paths used throughout this thesis, has been reported to produce both faster training convergence and improved sample quality in continuous domains; its applicability to the discrete tile vocabulary of the present task is open. The recently proposed Discrete Flow Matching variant for categorical data would replace the argmax decoding step that this thesis identified as the quality-ceiling-limiting component, and may eliminate the need for a post-generation repair pipeline entirely. Mario AI Framework integration, which would replace the geometric

Failure-Mode Analyzer with a simulated A-star agent verifying that generated chunks are actually playable, would provide a functionally-grounded evaluation metric that is closer to what level-design quality ultimately measures.

The repair pipeline itself admits several extensions. The eleven rule-based primitives could be replaced by learned primitives trained to perform the same corrective operations from data, eliminating the manual rule-design effort and potentially generalizing to other tile-based games without rule-set redesign. The orchestration could be made adaptive, applying only the primitives needed for a given chunk rather than running the full pass sequence every time. The repair stage could also be integrated into training as an auxiliary signal, encouraging the model to produce chunks that require less repair, potentially closing the quality ceiling that Iteration 3 demonstrated from the training side rather than the post-processing side.

Each of these directions is a candidate for a future iteration of the work. The present thesis establishes that the Conditional Flow Matching plus repair-pipeline approach is feasible, controllable, and produces near-zero-violation output on a well-known tile-based game corpus. The natural next step is to push that approach into the dimensions this thesis did not measure (sampling efficiency, diversity, generalization) and the architectures this thesis did not exercise (Optimal Transport paths, Discrete Flow Matching, learned repair).

REFERENCES

CHENG, D. HAN, H. FEI, G. **Automatic Generation of Game Levels Based on Controllable Wave Function Collapse Algorithm.** In: Entertainment Computing – ICEC 2020. Springer Lecture Notes in Computer Science, vol 12523. 2021. DOI: https://doi.org/10.1007/978-3-030-65736-9_3

GOODFELLOW, I. POUGET-ABADIE, J. MIRZA, M. Xu, B. WARDE-FARLEY, D. OZAIR, S. COURVILLE, A. BENGIO, Y. **Generative Adversarial Nets.** Advances in Neural Information Processing Systems 27. 2014.

GUMIN, M. *WaveFunctionCollapse*. GitHub repository. 2016. Retrieved from https://github.com/mxgmn/WaveFunctionCollapse

HENDRIKX, M. MEIJER, S. VAN DER VELDEN, J. IOSUP, A. Procedural content generation for games: A survey. **ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)**, 9(1), p. 1-22, 2013. DOI: https://doi.org/10.1145/2422956.2422957

HO, J. JAIN, A. ABBEEL, P. **Denoising Diffusion Probabilistic Models**. Advances in Neural Information Processing Systems 33. 2020. DOI: https://doi.org/10.48550/arXiv.2006.11239

LIPMAN, Y. CHEN, R. T. Q. BEN-HAMU, H. GROSSE, R. DUVENAUD, D. **Flow Matching for Generative Modeling**. 2023. DOI: https://doi.org/10.48550/arXiv.2210.02747

LIU, J., SNODGRASS, S., KHALIFA, A. *et al.* **Deep learning for procedural content generation.** ***Neural Comput & Applic*** 33, 19–37 (2021). DOI: https://doi.org/10.1007/s00521-020-05383-8

REN, L. **Application of Denoising Diffusion Probabilistic Models in the Minecraft Environment.** In: Proceedings of the 2024 4th International Conference on Artificial Intelligence, Big Data and Algorithms (CAIBDA 2024). 2024. DOI: https://doi.org/10.1145/3690407.3690499

RUSSELL, S. J. NORVIG, P. **Artificial Intelligence:** A Modern Approach. 4. ed. Pearson. 2022. ISBN 13: 978-1-292-40113-3

SILVA, D. F. TORCHELSEN, R. P. AGUIAR, M. S. **Procedural game level generation with GANs: potential, weaknesses, and unresolved challenges in the literature.** Multimedia Tools and Applications. 2025. DOI: https://doi.org/10.1007/s11042-025-20612-9

SONG, Y. DHARIWAL, P. CHEN, M. SUTSKEVER, I. **Consistency Models.** In: Proceedings of the 40th International Conference on Machine Learning (ICML), v. 202, p. 32211-32252. 2023. DOI: https://doi.org/10.48550/arXiv.2303.01469

SUMMERVILLE, A. SNODGRASS, S. MATEAS, M. ONTAÑÓN, S. **The VGLC: The Video Game Level Corpus.** arXiv:1606.07487. 2016. DOI: https://doi.org/10.48550/arXiv.1606.07487

DECLARATION ON THE USE OF ARTIFICIAL INTELLIGENCE

In compliance with the Federal University of Santa Catarina's guidelines on the use of artificial intelligence in academic work, the author declares the following uses of generative AI tools during the preparation of this Undergraduate Thesis.

The author used Claude (Anthropic) as a writing-assistance tool throughout the preparation of Chapters 4, 5, 6, and 7 of this thesis. The AI's role was confined to prose drafting, prose revision, terminology consistency checking, and editorial cleanup of style elements (em-dash removal, paragraph-indentation normalization, caption shortening). Every paragraph generated by the AI was reviewed by the author for technical accuracy against the source materials (the project's code, training logs, generated chunk images, and the eleven internal development reports written by the author during the project's execution); any AI-generated statement that did not match the source materials was rewritten or removed by the author.

The research direction, the choice of method (Conditional Flow Matching), the architectural choices (the two-level U-Net configuration, the residual block structure, the additive time conditioning, the biome embedding mechanism, and the eleven-primitive repair pipeline), the experimental design (the three sub-experiments of Iteration 2, the scaling test of Iteration 3, the six revision rounds of the post-generation repair pipeline that constitutes Iteration 4), and the interpretation of results were determined by the author. The code implementation of the pipeline was written by the author, with AI assistance limited to code review and debugging during specific incidents. The data engineering pipeline, the failure-mode analyzer, the U-Net architecture, the Conditional Flow Matching loss, the Euler ODE solver, the post-generation repair primitives, and the orchestrator were all designed and implemented by the author.

The literature review in Chapter 2 was conducted by the author through searches in academic databases as documented in Section 2.1. The cited references were selected and read by the author; AI was not used to surface or summarize external publications without the author's verification of the source.

The author takes full responsibility for the content of this thesis, including any errors that may remain despite the verification steps described above.

APPENDIX A - SOURCE CODE

The complete source code for this thesis is publicly available at: https://github.com/luizslongo/procgen-flow-matching. The repository contains, at the time of thesis submission:

- Data engineering layer (c2-pcg.flow-matching-dataloader): VGLC parser, chunker, one-hot tensor converter, tile-frequency computer.
- Model layer (c2-pcg.flow-matching-model): the two-level U-Net architecture (Section 4.3), residual convolutional block, sinusoidal time embedding, biome embedding table.
- Training layer (c2-pcg.flow-matching-training): the Conditional Flow Matching loss (uniform and class-balanced variants), the training loop, checkpoint serialization.
- Inference layer (c2-pcg.flow-matching-inference): the Euler ODE solver at fifty function evaluations, the map generator, argmax decoding.
- Evaluation layer (c2-pcg.flow-matching-evaluation): the six-category failure-mode analyzer (Section 4.4), the eleven-primitive post-generation repair pipeline (Chapter 6), the biome-aware bottom-row-completion primitive, the Python sprite renderer.
- Entrypoints (c4-cmd.pcg-flow-matching-train, c4-cmd.pcg-flow-matching-generate): command-line executables for training and inference.
- Auxiliary scripts (scripts/): Python scripts for loss-curve plotting, ASCII-to-PNG chunk rendering, tile-frequency computation.
- Documentation (docs/): eleven internal development reports written by the author during the project's execution, one per iteration and revision round.

All checkpoints referenced in the thesis (unet-baseline-checkpoint.bin, unet-class-balanced-checkpoint.bin, unet-sqrt-balanced-checkpoint.bin, unet-conditional-checkpoint.bin) are preserved in the repository.

The repository can be built and executed via:

dotnet build procgen-flow-matching.sln

dotnet run --project c4-cmd.pcg-flow-matching-train -- <path-to-VGLC>

dotnet run --project c4-cmd.pcg-flow-matching-generate -- <checkpoint-path>

APPENDIX B - SCIENTIFIC ARTICLE

PROCEDURAL GENERATION OF GAME MAPS AS A CONSTRAINT SATISFACTION PROBLEM: AN EULER-BASED CONDITIONAL FLOW MATCHING APPROACH

GERAÇÃO PROCEDURAL DE MAPAS DE JOGOS COMO UM PROBLEMA DE SATISFAÇÃO DE RESTRIÇÕES: UMA ABORDAGEM DE CONDITIONAL FLOW MATCHING BASEADA EM EULER

Luiz Gabriel Slongo[3]
Elder Rizzon Santos[4]

ABSTRACT

This work investigates the application of Euler-based Conditional Flow Matching (CFM) to the procedural generation of tile-based 2D game maps, framed as a Constraint Satisfaction Problem (CSP) in which generated maps must satisfy implicit local and global structural rules learned from a training corpus. A generative pipeline is implemented from first principles in C# with TorchSharp: a two-level U-Net parameterizes a time-dependent vector field trained via the Conditional Flow Matching objective, and inference proceeds through an Euler ordinary differential equation solver at fifty function evaluations per sample. The Video Game Level Corpus Super Mario Bros distribution provides fifteen levels across three biomes (Overworld, Underground, Treetop) as the training set. Development proceeded across four iterations, each targeting the dominant failure mode observed in the previous iteration's output as measured by a six-category failure-mode analyzer. Iteration 1 established the baseline pipeline; Iteration 2 explored class-balanced loss variants and biome conditioning; Iteration 3 tested capacity-and-duration scaling and produced a negative result identifying a fundamental quality ceiling of the CFM plus argmax decoding formulation; Iteration 4 introduced a deterministic

[3]* Undergraduate student, Computer Science Program, Federal University of Santa Catarina (UFSC). Florianopolis, SC, Brazil. E-mail: luizgabrielslongo@gmail.com

[4]** Advisor. Prof., Ph.D., Department of Informatics and Statistics, Federal University of Santa Catarina (UFSC). Florianopolis, SC, Brazil.

post-generation repair pipeline of eleven rule-based primitives. Across thirty generated chunks, the hybrid pipeline reduced three hundred and nine unintended structural violations to zero, with sixty-five intentional Treetop ground gaps preserved as a biome feature.

Keywords: procedural content generation; conditional flow matching; constraint satisfaction problem; hybrid neural-symbolic systems; Super Mario Bros.

RESUMO

Este trabalho investiga a aplicação de Conditional Flow Matching (CFM) baseado em Euler para a geração procedural de mapas de jogos 2D baseados em tiles, formulada como um Problema de Satisfação de Restrições (CSP) no qual os mapas gerados devem satisfazer regras estruturais locais e globais implícitas, aprendidas a partir de um corpus de treinamento. Uma pipeline generativa e implementada do zero em C# com TorchSharp: uma U-Net de dois níveis parametriza um campo vetorial dependente do tempo, treinando com o objetivo Conditional Flow Matching, e a inferência procede através de um solver de Equações Diferenciais Ordinárias (Euler) com cinquenta avaliações de função por amostra. O corpus Video Game Level Corpus (VGLC), distribuição Super Mario Bros, fornece quinze níveis distribuídos em três biomas (Overworld, Underground, Treetop) como conjunto de treinamento. O desenvolvimento procedeu em quatro iterações, cada uma direcionada a falha dominante observada na saída da iteração anterior, medida por um analisador de modos de falha de seis categorias. A Iteração 1 estabeleceu a pipeline básica; a Iteração 2 explorou variantes de perda class-balanced e condicionamento por bioma; a Iteração 3 testou escalonamento de capacidade e duração, produzindo um resultado negativo que identifica um teto de qualidade fundamental na formulação CFM mais decodificação por argmax; a Iteração 4 introduziu uma pipeline determinística de reparo após-geração composta por onze primitivas baseadas em regras. Em trinta chunks gerados, a pipeline híbrida reduziu trezentas e nove violações estruturais não-intencionais a zero, preservando sessenta e cinco lacunas intencionais no bioma Treetop como característica de bioma.

Palavras-chave: geração procedural de conteúdo; conditional flow matching; problema de satisfação de restrições; sistemas híbridos neuro-simbólicos; Super Mario Bros.

1 INTRODUCTION

Procedural Content Generation (PCG) comprises the computational methods used to create game content algorithmically rather than manually, and it is a core enabling technology for the large, complex virtual worlds of the modern games industry (HENDRIKX et al., 2013). Among its most common applications is the generation of maps and levels, present in genres ranging from open-world survival games to science-fiction exploration titles. The generation of a valid map can be formalized as a Constraint Satisfaction Problem (CSP), a member of the NP-hard class in which the cost of finding a guaranteed solution can grow exponentially with map size and constraint count (HENDRIKX et al., 2013; RUSSELL; NORVIG, 2022).

In tile-based 2D maps, the constraints operate on two levels: local constraints, governing the adjacency of individual tiles (for example, the two halves of a pipe cap must sit side by side), and global constraints, governing overall coherence (for example, a continuous ground surface or biome-consistent layout). Classic constraint-based approaches such as Wave Function Collapse (WFC) are effective but rely on explicitly defined rulesets extracted from example maps, which limits their ability to capture higher-level design intent (GUMIN, 2016). Deep generative models offer an alternative: by training on a corpus of valid maps, they learn both local and global constraints implicitly, without manual rule definition (LIU et al., 2021). However, Denoising Diffusion Probabilistic Models (DDPMs) achieve high quality only through a slow sampling process requiring hundreds or thousands of iterative steps (HO; JAIN; ABBEEL, 2020).

This work adopts a more efficient alternative - Euler-based Flow Matching (LIPMAN et al., 2023) - which learns a deterministic vector field defining a direct path from a noise distribution to the valid-map distribution, enabling high-quality generation in far fewer steps. The general objective is to develop, train, and evaluate an Euler-based Flow Matching model for the procedural generation of coherent and diverse maps, framing the task as an implicit solution to a CSP. This objective decomposes into: (O1) a review of PCG from classic constraint-based methods to modern deep generative models; (O2) a data-engineering pipeline converting 2D maps into a training-ready tensor format; (O3) implementation and training of a CFM model that learns the implicit constraints; (O4) a conditional control mechanism guiding generation toward specific attributes; and (O5) an evaluation of constraint satisfaction via a failure-mode analyzer, positioned against WFC, DDPMs, and GANs.

2 THEORETICAL BACKGROUND AND RELATED WORK

This section reviews the landscape that motivates the proposed method, charting the path from the foundational framing of procedural generation as a constraint satisfaction problem to the deep generative techniques that make implicit constraint learning feasible. Subsection 2.1 establishes the CSP framing; Subsection 2.2 surveys the transition from rule-based methods to deep generative models; and Subsection 2.3 introduces the Flow Matching paradigm and the Conditional Flow Matching objective adopted in this work, closing with a comparative synthesis of the alternatives.

2.1 PCG AS A CONSTRAINT SATISFACTION PROBLEM

Hendrikx et al. (2013) provide the seminal survey structuring the PCG-for-games landscape, classifying content into a six-layer pyramid and placing "Constraint Satisfaction and Planning" among the AI generation methods while acknowledging its NP-hard nature. This classification provides the formal grounding for treating map generation as a CSP defined by the tuple (V, D, C): one variable per grid cell, a domain equal to the tile vocabulary, and a constraint set C = C_local U C_global (RUSSELL; NORVIG, 2022).

2.2 FROM RULE-BASED METHODS TO DEEP GENERATIVE MODELS

Liu et al. (2021) document the shift from manually defined rulesets to implicitly learned constraints, emphasizing that game levels are subject to functionality constraints - a level that is not playable has essentially zero utility. Silva et al. (2025) review GAN-based level generation and identify persistent problems of training instability and mode collapse; Ren (2024) demonstrates DDPMs on voxel-based Minecraft structures but re-exposes the diffusion sampling-cost problem; Song et al. (2023) propose Consistency Models for one-step generation at the cost of training instability.

2.3 FLOW MATCHING AND THE CONDITIONAL FLOW MATCHING OBJECTIVE

Lipman et al. (2023) introduce Flow Matching within the Continuous Normalizing Flows framework, learning a deterministic vector field v_theta(x_t, t) that defines an ordinary differential equation dx/dt = v_theta(x_t, t) from noise (t = 0) to data (t = 1). Their simulation-free Conditional Flow Matching (CFM) objective avoids simulating the full trajectory during training by regressing per-sample conditional paths, yielding stable, fast convergence and low-NFE sampling. This work adopts the standard linear-interpolation CFM objective with a U-Net vector-field approximator, integrated with the simple first-order Euler method. Table 1 situates the approach against the alternatives.

Table 1 - Comparative analysis of generative approaches for procedural map generation

Approach	**Representative work**	**Constraint handling**	**Sampling efficiency**	**Data requirement**	**Key limitation**
Standard WFC	Gumin (2016)	Excellent local / poor global	Very high	Single example	Rigid output; weak global structure
Controllable WFC	Cheng et al. (2021)	Improved global (heuristics)	Medium (backtracking)	Single example + heuristics	Manual rule complexity
GANs	Silva et al. (2025)	Learns patterns; weak validity	High (single pass)	Large dataset	Mode collapse; instability
DDPMs	Ren (2024)	Excellent global & local	Low (high NFE)	Large dataset	Very slow (1000s of steps)
Flow Matching	Lipman et al. (2023)	Excellent global & local	High (low NFE)	Large dataset	Emerging; requires ODE solver
This work (Hybrid CFM+repair)	Slongo (2026)	Excellent global & local	High (low NFE)	Large dataset + rule set	Manual rule design; corpus-specific

Source: prepared by the authors.

3 MATERIALS AND METHODS

This section describes the method as a sequence of connected stages, each addressing one of the specific objectives stated in Section 1. The data-engineering pipeline (3.1) converts heterogeneous source levels into a uniform tensor representation; the generative model (3.2) learns the constraint distribution implicitly through a U-Net trained with the Conditional Flow Matching objective; and the inference procedure (3.3) traverses the learned vector field with an Euler ODE solver. The evaluation methodology (3.4) and the iterative development process (3.5) that organizes how the method is built and measured close the section.

3.1 DATA ENGINEERING PIPELINE

To address Objective O2, the training corpus is drawn from the Video Game Level Corpus (VGLC) Super Mario Bros distribution (SUMMERVILLE et al., 2016), which encodes levels as ASCII text where each character maps to one 16x16-pixel NES tile. Fifteen levels are used (mario-1-1 through mario-8-1), each fourteen tile-rows tall with column counts from 149 to 373. Levels are sliced into fixed 14x28 chunks and one-hot encoded over a 14-type tile vocabulary - the thirteen VGLC visual types plus an Error sentinel reserved at index 0 as a defensive uninitialized-state marker - producing tensors of shape (C, H, W) with C = 14. Each level is labeled with one of four biome values (Error, Overworld, Underground, Treetop) propagated to every derived chunk.

3.2 GENERATIVE MODEL: U-NET AND CONDITIONAL FLOW MATCHING

To address Objective O3, the vector field is parameterized by a two-level U-Net (~3 million parameters) - the maximum depth permitted by the 14-row chunk height - with channel widths of 64 -> 128 -> 256 and mirrored decoder, skip connections concatenated along the channel dimension, and 1x1 channel adapters translating between the 14-channel one-hot space and the 64-channel feature space. Each residual block applies two 3x3 convolutions with GroupNorm and SiLU, injects a 128-dimensional sinusoidal time embedding additively between the convolutions, and carries an identity (or 1x1) shortcut. A learnable biome embedding,

summed with the time embedding, provides the conditional signal that later supports Objective O4.

The CFM objective uses the linear interpolant x_t = (1-t)*x0 + t*x1 with noise x0 ~ N(0, I) and data x1, and the constant target velocity u_t(x_t | x1) = x1 - x0. Training minimizes the mean squared error L(theta) = E[|| v_theta(x_t, t) - u_t(x_t | x1) ||^2]; the constancy of the target along each path is what makes the objective simulation-free. The pipeline is implemented from first principles in C# on .NET 8 with TorchSharp 0.107.0 (a managed libtorch wrapper), trained on a single NVIDIA RTX 5060 GPU. Fixed hyperparameters: Adam optimizer, learning rate 5x10^-4, batch size 32, manual seed 42, and global gradient-norm clipping at 0.5 (a stability guard adopted after early NaN divergence).

3.3 INFERENCE: EULER ODE SOLVER

Generation integrates the learned ODE from noise to data using the Euler method with 50 uniform steps (dt = 0.02), applying x_{t+dt} = x_t + v_theta(x_t, t, c)*dt, followed by an argmax over the 14 output channels to decode a discrete tile map. Euler is chosen over higher-order integrators to minimize the Number of Function Evaluations (NFE), the primary sampling-efficiency metric of Flow Matching.

3.4 EVALUATION: FAILURE-MODE ANALYZER

To address Objective O5, each generated chunk is scored by a six-category analyzer, each category an independent linear-time local scan: (1) broken horizontal pipe segment, (2) broken pipe-top connection (left), (3) broken pipe-top connection (right), (4) broken bullet-bill launcher, (5) floating enemy, and (6) discontinuous ground. The analyzer yields per-category and total violation counts, serving as the study's primary quantitative metric.

3.5 ITERATIVE DEVELOPMENT METHODOLOGY

Development followed four iterations, each producing a complete working pipeline, measured against the analyzer, and informing the next. Each iteration targeted the dominant failure mode observed in the previous one.

4 RESULTS

This section reports the four development iterations, each measured against the failure-mode analyzer of Subsection 3.4 and each targeting the dominant failure mode observed in the previous one. Subsection 4.1 presents the baseline pipeline; Subsection 4.2 covers the class-balanced loss, biome-conditioning, and scaling experiments of Iterations 2 and 3, including the negative result on scaling; and Subsection 4.3 reports the hybrid repair pipeline of Iteration 4 that drives the unintended structural violations to zero.

4.1 ITERATION 1: BASELINE PIPELINE

A 2,000-step CPU verification run reduced the CFM loss from 1.07 to 0.21 (minimum 0.13). Ten generated chunks accumulated 77 violations (mean 7.7 per chunk, 1.96% per tile position), dominated by discontinuous ground (46; 59.7%) and floating enemies (12; 15.6%). A subsequent 20,000-step GPU run drove the loss to 0.028 (a 97% reduction) in ~10 minutes and produced visually recognizable Mario segments with well-formed pipe assemblies - yet the violation profile did not fall proportionally (discontinuous ground remained ~62% of violations). This established the key diagnosis: the residual structural errors are not driven by under-training.

4.2 ITERATIONS 2 AND 3: CLASS-BALANCED LOSS, BIOME CONDITIONING, AND SCALING

The tile distribution is severely imbalanced (Empty 86.2%, Solid 8.4%, remaining twelve types 5.4%), so a class-balanced loss was tested (Table 2). Raw inverse-frequency weighting (v1, dynamic range 2487:1) fully resolved discontinuous ground but induced absurd Breakable-brick

saturation (65-75% of tiles). Square-root inverse-frequency weighting (v2, range 50:1) recovered a balanced distribution and reached the best loss minimum (0.055), but exposed a biome-averaging artifact: an Underground cave-ceiling appearing even when unrequested - the unconditional posterior's marginal over the unobserved biome variable. Experiment D added a learned biome embedding and reverted to the uniform loss, producing three visually distinct biomes. A post-submission ablation, however, showed the trained embedding alone does not converge to biome-differentiating vectors on this small imbalanced corpus; biome distinction is ultimately contributed by the biome-aware repair primitives of Iteration 4.

Table 2 - Tile frequency and the two class-balanced weight schedules (selected rows)

Tile type	**Count**	**Frequency (%)**	**v1 weight (raw inv)**	**v2 weight (sqrt inv)**
Empty	850570	86.17	0.089	0.575
Solid	83322	8.44	0.911	1.836
Breakable	23529	2.38	3.227	3.455
PipeTopLeft	2310	0.23	32.869	11.025
QuestionFull	768	0.08	98.864	19.121
BulletBillBody	343	0.03	221.363	28.612
Total	987056	100	—	—

Source: prepared by the authors.

Iteration 3 tested scaling jointly - 64 -> 128 channels (~4x parameters) and 20,000 -> 100,000 steps - and produced a negative result. The loss plateaued at 0.18 for the final 55,500 steps and output quality regressed relative to Experiment D. Three mechanisms explain this: the loss floor is set by the CFM-plus-argmax formulation rather than by capacity or duration; wider channels spread the same signal without improving per-pixel argmax accuracy; and argmax decoding has an irreducible error rate the loss never penalizes directly. The conclusion - that zero-violation generation is a decoder problem, not a training problem - motivated Iteration 4. Table 3 summarizes the training configurations.

Table 3 - Training configuration and reported loss across iterations

Iteration / experiment	Steps	Loss (start)	Loss (end)	Notes
Iter 1 Phase 1 (CPU baseline)	2,000	1.07	0.21	Mechanical verification
Iter 1 Phase 2 (GPU production)	20,000	1.07	0.028	Lower loss bound, uniform formulation
Iter 2 Exp B v1 (raw inv-freq)	20,000	1.034	0.174	Continuous ground; brick saturation
Iter 2 Exp B v2 (sqrt inv-freq)	20,000	1.033	0.088	Best minimum; biome averaging exposed
Iter 2 Exp D (biome conditional)	20,000	1.067	0.239	Closes Iter 2; loss reverted to uniform
Iter 3 (BC=128, 100k steps)	100,000	1.048	0.282	Negative result; plateau at 0.18

Source: prepared by the authors.

4.3 ITERATION 4: HYBRID REPAIR PIPELINE

A deterministic, side-effect-free post-generation repair pipeline of eleven rule-based primitives operates on the trained model's output within a convergence = loop (2-4 sweeps until no tile changes). The primitives perform four kinds of correction: orphan removal (PipeRepair, BulletBillRepair, QuestionBlockRepair), support completion (PipeSupportRepair, BulletBillSupportRepair, EnemyRepair, BottomRowCompletion), ceiling and clearance enforcement (CeilingCompletion, PipeTopClearance, BulletBillLauncherClearance), and structure injection (PipeInjection). The pipeline was developed empirically across six revision rounds, each targeting the dominant failure mode visible in the previous round's output, converging when no further failures were observable.

On a 30-chunk audit (ten per biome), the pipeline reduced Overworld violations from 155 to 0 and Underground from 154 to 0. In Treetop, the count fell from 123 to 65, where every remaining violation is an intentional ground gap that the biome-aware BottomRowCompletion primitive deliberately preserves - Treetop levels use such gaps as a design feature (Table 4). The

residual 65 are thus a measurement artifact of the biome-blind analyzer, not structural failures. Repair adds only milliseconds per chunk, leaving sampling efficiency unaffected.

Table 4 - Structural violation counts across thirty generated chunks, before and after repair

Biome	Pre-repair	Post-repair (unintended)	Post-repair (intentional)
Overworld	155	0	0
Underground	154	0	0
Treetop	123	0	65 (preserved as biome feature)
Total	432	0	65

Source: prepared by the authors.

5 DISCUSSION

The central finding is that the CFM-plus-argmax formulation has a fundamental quality ceiling that neither capacity nor training duration can cross: across two orders of magnitude of training steps and a fourfold capacity variation, the model produces visually plausible chunks but cannot guarantee local structural rules. This makes the case for a hybrid neural-symbolic design in which the trained model and the rule-based repair stage are complementary rather than competing - the model supplies the biome-distinct content distribution that rules cannot describe, and the rules supply the hard structural guarantees the model cannot enforce. The result is zero unintended violations across all three biomes at low NFE.

Three secondary findings emerged. First, loss-formulation changes can shift the failure distribution across categories but cannot eliminate failures, because argmax error is not penalized by the CFM loss. Second, biome conditioning is essential for correct multi-biome generation, since the unconditional posterior necessarily averages biome-specific features; on this small corpus, however, that conditioning is realized through biome-aware repair rather than through the trained embedding. Third, the iterative-refinement methodology scales naturally: each visible failure mode motivates one localized primitive, and the primitives compose.

The study's limitations are explicit. The analyzer is biome-blind, so it misreports intentional Treetop gaps as violations. Of the three validation axes proposed (constraint satisfaction, sampling efficiency, diversity), only constraint satisfaction is reported in full; no

WFC baseline was implemented to anchor efficiency comparisons, and no diversity metric was implemented to rule out memorization. The corpus is restricted to fifteen Super Mario Bros levels across three biomes, and the repair rules are hand-designed and corpus-specific.

6 FINAL CONSIDERATIONS

This work established that an Euler-based Conditional Flow Matching model, combined with a deterministic post-generation repair pipeline, is a feasible, controllable approach to procedural tile-map generation that produces near-zero-violation output on a well-known corpus while retaining the low-NFE sampling efficiency of Flow Matching. Objectives O1-O3 were fully met, O4 was met by the hybrid conditional interface, and the constraint-satisfaction portion of O5 was met within the limits of the evaluation methodology.

The most direct extensions follow from the limitations: a WFC baseline and a diversity metric to close the remaining validation axes, and a biome-aware analyzer to report honest Treetop counts. Methodologically, the Optimal Transport variant of Flow Matching and, most promisingly, Discrete Flow Matching - which would replace the argmax decoding step identified here as the quality-ceiling-limiting component - could reduce or eliminate the need for post-generation repair. Finally, the eleven hand-designed primitives could be replaced by learned repair operators, and the repair signal could be folded back into training so the model learns to require less correction.

REFERENCES

CHENG, D.; HAN, H.; FEI, G. **Automatic Generation of Game Levels Based on Controllable Wave Function Collapse Algorithm.** In: Entertainment Computing - ICEC 2020. Springer Lecture Notes in Computer Science, vol. 12523, 2021. DOI: https://doi.org/10.1007/978-3-030-65736-9_3.

GOODFELLOW, I. et al. **Generative Adversarial Nets.** Advances in Neural Information Processing Systems 27, 2014.

GUMIN, M. **WaveFunctionCollapse.** GitHub repository, 2016. Available at: https://github.com/mxgmn/WaveFunctionCollapse.

HENDRIKX, M.; MEIJER, S.; VAN DER VELDEN, J.; IOSUP, A. **Procedural content generation for games: A survey.** ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), v. 9, n. 1, p. 1-22, 2013. DOI: https://doi.org/10.1145/2422956.2422957.

HO, J.; JAIN, A.; ABBEEL, P. **Denoising Diffusion Probabilistic Models.** Advances in Neural Information Processing Systems 33, 2020. DOI: https://doi.org/10.48550/arXiv.2006.11239.

LIPMAN, Y.; CHEN, R. T. Q.; BEN-HAMU, H.; GROSSE, R.; DUVENAUD, D. **Flow Matching for Generative Modeling.** 2023. DOI: https://doi.org/10.48550/arXiv.2210.02747.

LIU, J.; SNODGRASS, S.; KHALIFA, A. et al. **Deep learning for procedural content generation.** Neural Computing & Applications, v. 33, p. 19-37, 2021. DOI: https://doi.org/10.1007/s00521-020-05383-8.

REN, L. **Application of Denoising Diffusion Probabilistic Models in the Minecraft Environment.** In: Proceedings of the 2024 4th International Conference on Artificial Intelligence, Big Data and Algorithms (CAIBDA 2024), 2024. DOI: https://doi.org/10.1145/3690407.3690499.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 4. Ed. Pearson, 2022. ISBN 978-1-292-40113-3.

SILVA, D. F.; TORCHELSEN, R. P.; AGUIAR, M. S. **Procedural game level generation with GANs: potential, weaknesses, and unresolved challenges in the literature.** Multimedia Tools and Applications, 2025. DOI: https://doi.org/10.1007/s11042-025-20612-9.

SONG, Y.; DHARIWAL, P.; CHEN, M.; SUTSKEVER, I. **Consistency Models.** In: Proceedings of the 40th International Conference on Machine Learning (ICML), v. 202, p. 32211-32252, 2023. DOI: https://doi.org/10.48550/arXiv.2303.01469.

SUMMERVILLE, A.; SNODGRASS, S.; MATEAS, M.; ONTANON, S. **The VGLC: The Video GameLevel Corpus.** arXiv:1606.07487, 2016. DOI: https://doi.org/10.48550/arXiv.1606.07487.