Conditional Diffusion for Specific Cellular Automata Synthesis in Conway's Game of Life

Jin Schofield, Vikram Ramaswamy
Princeton University

Abstract

Generative machine learning can often fail to properly model discrete patterns in logic-based environments. In this work, I present a conditional diffusion framework that makes use of DDPM, classifier-free guidance, and FiLM-based class embeddings. The model is trained on both random and quota-balanced datasets of Conway's Game of Life to ensure representation of rare patterns. Conditional sampling increased the occurrence of targeted life form patterns when compared to unconditional sampling. Motif analysis and t-SNE visualization showcase the frequency of varyingly complex cellular automata as well as the separability of noising embeddings for different forms of life. The findings of this project demonstrate that conditional diffusion can distill and increase the occurrence of fine-grained discrete structures in logic-based environments, furthering the field of applying interpretable generative modeling to discrete data. Diffusion models are better at learning simpler structures as opposed to complex structures, and are capable of learning combinations of these structures.

1 Introduction

Diffusion models are a form of generative machine learning that is especially popular for two-dimensional image generation. By training on a series of images in a dataset, they learn the capability to remove noise from pure Gaussian noise to create images that appear as if they could have been sampled from the original dataset. Compared to other image generation techniques, such as those using generative adversarial networks (GANs) and variational autoencoders (VAEs), diffusion models demonstrate better quality and training stability (Goodfellow 2014; Kingma 2022). However, hallucinations, or logical errors, are often generated in diffusion generations, such as biologically impossible depictions of humans. Diffusion models perform well on continuous image generation tasks, but can perform more poorly when robustly creating images in discrete or logically structured settings.

In particular, diffusion models struggle to identify discrete patterns in data and robustly apply them during image generation (Austin 2023). Diffusion processes, which are ideal for training on datasets of continuous domains, can fail to respect logical, combinatorial rules found in environments such as Conway's Game of Life. If applied naively, diffusion models are more likely to represent outcomes of higher probability in the dataset, such as extinction of a pixel configuration. This leads to the underrepresentation of more rare, delicate structures such as forms of life that persist throughout the game. This is due to how, in Conway's Game of Life, every pixel has the potential to impact other pixels in a sort of butterfly-like effect. Thus, in order to correctly model these forms of life, the diffusion model must be precise with its generations in a discrete setting, learning patterns to adhere to the environment's logical rules. A small mistake in the form of a single pixel has the potential to change the label of an entire sample generation.

In essence, this paper seeks to utilize diffusion to create conditional embeddings that represent logical patterns. In NLP and computer vision, embeddings are numerical representations of tokens that represent abstract ideas, such as the visual manifestation of a ”car”, or the syntactic and semantic manifestations of ”car”. When these embeddings are created conditioned on specific subset of data, they can be referred to as conditional embeddings. The paper trains conditional embeddings in diffusion such that, in order to represent a notion such as a ”still-life” living configuration in Conway’s Game of Life, they must partially emulate the logical pattern of a still-life as well. This attempt to distill logical patterns in embeddings aims to reduce hallucinations caused by logical errors in computer vision, as well as increase interpretability.

Specifically, this work aims to develop a conditional diffusion framework for discrete pattern synthesis in cellular automata.

The contribution of this paper to the fields of interpretability and computer vision are the creation of a pipeline that created embeddings that represent logic-based fine-grained patterns in a discrete setting, and a diffusion model that can synthesize specific cellular automata in the Conway’s Game of Life environment. In particular, the diffusion model will be able to generate still-life and 2-period oscillator forms of life, as well as cellular automata that die off.

In using datasets of only 4000 samples, each of only a 32 by 32 resolution, this technique does not require a large dataset and thus contributes to dataset efficiency as well.

2 Background

2.1 Cellular Automata and Conway’s Game of Life

Cellular automata are dynamical systems that operate in discrete environments, particularly a grid of cells, where each cell can take a value from a finite set. Each cell updates according to local transition rules and all cells update at the same time (Wolfram 1984). One set-up for cellular automata is Conway’s Game of Life (GoL) (Gardner 1970). In GoL, there exists a two-dimensional grid where each cell in the grid can either be dead or alive. For the purposes of this paper, white indicates dead and black indicates alive. Each cell has eight neighbors. If a live cell has two or three live neighbors, it survives. Otherwise, it dies. if a dead cell has exactly three live neighbors, it becomes a live cell. There are many emergent behaviors that occur in GoL. For example, there are various still-life patterns, such as blocks and beehives. These patterns include live cells which always have exactly two or three live neighbors, and dead cells without exactly three live neighbors. Thus, no cells change with each timestep. Period-k oscillators are patterns that repeat every k timesteps, such as blinkers and toads. There are also configurations that move such as gliders and spaceships. In this paper, I will focus on identifying conditional embeddings for still-life and period-2 oscillators.

2.2 Generative Machine Learning and Applications to Discrete Data

Generative machine learning is a form of machine learning that specializes in training models that can generate data that resembles that of a training dataset. Existing techniques are accurate and efficient when dealing with continuous data. One example is the generative adversarial network (GAN), which makes use of competing ”generator” and ”discriminator” models to produce images that resemble a dataset, although the technique suffers from training instability and mode collapse (Goodfellow 2014; Arjovsky 2017). A second technique is the variational autoencoder (VAE) that learns to encode and decode a latent representation of important features of a dataset, and then sample from it using randomization. This technique can suffer from being imprecise in its generations (Kingma 2022). A technique that has been particularly accurate is the denoising diffusion probabilistic model (DDPMs). This technique consists of adding Gaussian noise to data over several time steps and then using a neural network to learn how to denoise an example of pure noise over various timesteps to reverse the noising process and reveal an image similar to the dataset (Ho 2020; Dhariwal 2021). Diffusion models are known for their stable training dynamics and ability to create high fidelity images.

These models leverage decoders that are differentiable, meaning a small change in the output can result from a small change in the input, as well as transformations done by these decoders that are invertible (there is a bijective mapping between input and output). In discrete settings, small changes in the input result in either no change to the output or very large changes to output. Thus, transformations are not bijective. Thus, the previously mentioned generative AI techniques can struggle to learn discrete data. The Gumbel-Softmax reparameterization technique aims to approximate discrete sampling by using continuous relaxations but suffers the pitfalls of poor sample quality and bias (Jang 2017). There exist graph-based generative models that use ”message-passing” networks in order to create discrete graphs (You 2018; Liu 2023), although these graph structures create associations between different tokenized embeddings rather than representing discrete logical rules.

2.3 Diffusion

As discussed earlier, diffusion models use a forward noising process $q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$ to convert samples of a dataset into noise. The technique then trains a network $\epsilon_\theta(x_t, t)$ (Sohl, Dickstein 2015; Ho 2020) to denoise samples of pure noise into samples reminiscent of the original dataset. A key component of the diffusion model is the noise schedule $\{\beta_t\}_{t=1}^{T}$, or the plan for how much noise is added at each timestep. If the noise schedule is linear, per-step noise is added by interpolating β_t. A cosine noise schedule instead aims to gradually increase cumulative noise $\bar{\alpha}_t$ in a smooth manner, which can increase fidelity of the outputs (Nichol 2021). Diffusion models can be made conditional by injecting new information, trained exclusively on certain subsets of the data, into the denoising, such that for certain conditions (i.e. a specific pattern in Conway's Game of Life), specific information faithful to that condition is added. The denoiser takes the form of a U-Net. At each time step, the U-Net takes the current noisy grid, the time step embedding, and any conditional embeddings (to be explained in the next paragraph), and predicts the noise residual, which is then removed from the image.

2.4 Conditional Diffusion and FiLM Layers

Conditional generative modelling can use one of two techniques — classifier guidance and classifier-free techniques. Classifier guidance trains a classifier $p_\phi(c|x_t)$ on the conditions but this causes inference costs to be doubled. Thus, my technique uses classifier-free guidance which randomly adds condition embeddings during training with probability p_{drop} at each timestep. At these timesteps, conditional and unconditional noise predictions are calculated during sampling using $\hat{\epsilon} = (1+w)\epsilon_\theta(x_t, t, c) - w\epsilon_\theta(x_t, t, \emptyset)$ (Ho 2020). My paper uses feature-wise linear modulation (FiLM) which integrates the condition embeddings by learning affine transformations on the intermediate outputs of the layer, which enables precise control over the generations (Perez 2017; Liu 2022).

2.5 Previous Work in Synthesis of Cellular Automata

A significant past approach to automating the discovery of different configurations of cellular automata is the use of genetic algorithms. The use of genetic evolutionary algorithms have been able to evolve initial conditions and even other rule sets that foster various objectives, such as length of life or complexity (Mitchell 1996). Genetic algorithms simulate evolution by evolving a "genetic code" in an environment by using mutation, cross-over, and selection.

In the past, convolutional neural networks have been successful at learning a GoL's dynamics from observation, including on random cellular automata, where probability is used to randomize part of the dynamics (Gilpin 2020). Additionally, neural networks have been used to train rules of cellular automata that allow for a desired configuration to exist (Mordvintsev 2020). However, these works tackle the ability to model cellular automata or create rule sets. The ability to condition for specific types of cellular automata, and thus to distill down logical relationships of specific behaviors within a rule set in a way that is generalizeable, has not been explored.

2.6 Embedding Visualization and Interpretability

Embeddings and conditional embeddings were introduced earlier in my introduction.

Overall, cellular automata and continuous generative modelling works have largely evolved independently. This work aims to connect the two lines of research and integrate generative modelling for this purely discrete environment. The use of diffusion instead of convolutional neural networks in my project allows for the distillation of conditional embeddings that represent and can thus generate specific types of life. A benefit of using diffusion is that it is inherently generative and thus has the potential to utilize conditional embeddings to generate new unseen patterns as well. We keep track of a novelty rate in this project to ensure generations are not clones or rotated or reflected versions of clones of training data.

3 METHODS

3.1 KEY NOVEL IDEA: WHY IT IS NEW AND WHY IT IS EFFECTIVE

The key novel idea of this paper is the framing of Conway's Game of Life cellular automata generation as a conditional diffusion problem. In particular, the novel idea is to leverage the precision of conditional diffusion to capture the patterns and logical backbone of certain froms of GoL cellular life in conditional embeddings. While previous attempts at understanding GoL have been either non-generative (such as with physics-based dynamics papers as well as the work exploring convolutional neural networks to understand the rules) or did not attempt to specify the differences between various GoL life forms, this work uses conditional diffusion to not only generate new GoL life forms but create conditional embeddings that can separate between them (Tapia McClung 2020; Gilpin 2020). This seeks to understand latent representations of structures that survive in the GoL environment rather than purely simulating pre-created patterns or analyzing existing dynamics.

This idea is effective because it leverages the fine-grained conditional control that well-trained conditional diffusion models are able to exhibit. Conditional diffusion not only learns how to differentiate between life forms based on classification, but also how to generate different types of noises at different time steps, thus allowing for finer control at the pixel level in generations, which is crucial for a discrete, logic-based environment such as GoL.

3.2 OVERARCHING EXPLANATION

The overarching goal is, in an environment operating under logical rules, to distill discrete cellular automata patterns using conditional diffusion generation. Each GoL board is represented as a 32 by 32 grid of binary cells, with white coloring representing a dead cell and black coloring representing an alive cell. The binary grid can thus be represented as $x_0 \in \{0,1\}^{32\times 32}$. For example, each training dataset sample takes that form.

The forward diffusion process adds Gaussian noise to x_0 over 200 timesteps. This produces a noisier version of x_0 which we will call x_t at timestep t. The U-Net, augmented with FiLM layers, learns to predict ϵ, the noise, at every timestep. These components are used at each time step to denoise during the reverse process.

There is two types of class conditioning originating in the training datasets by applying class labels to each sample in the dataset, which are then passed into the diffusion model when learning. The model reserves particular layers to learning only when a particular condition is present, thus constructing information to be given to the FiLM layer during the denoising reverse process. One condition explains whether after 200 logical timesteps of the Game of Life, the training sample configuration has at least one living cell left (alive), or all cells are dead (dead). The second condition has four options: dead, still-life, 2-period oscillator, and other form of life. This second condition simply conditions more granularly for a specific type of life. Classifying live samples between still-life, 2-period oscillator, and other occurs by looking at the final frames of the 200 timesteps. If they do not change, it is a still-life. If there is repetition of period 2, it is an oscillator of period 2. Otherwise, it is labelled other.

The one other form of conditioning is timestep conditioning, which takes the form of a sinusoidal positional encoding, meaning it is generated by computing a series of sine and cosine functions at varying frequencies. In particular, the vector takes the form,

$$n(w_1 t), \cos(w_1 t), \sin(w_2 t), \cos(w_2 t), \ldots, \sin(w_K t), \cos(w_K t)$$

where each w_k is a different frequency. This technique is used to create a more smooth difference between vectors at each time step, which allows for more stable training.

The timestep and class embeddings are summed together to form vector $e_{t,c}$. During inference, the classifier-guidance thus interpolates between the unconditional and conditional predictions by adjusting w_k. This allows for a balance between diversity of sampling and adherence to the condition.

3.3 DATASETS

We create two different datasets.

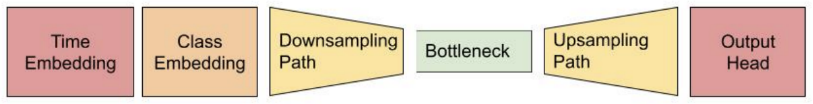


Figure 1: Components of Diffusion Model

The first dataset is the random dataset, meaning that 4000 GoL samples are generated randomly. They are generated by each cell having an independent probability of being alive of 0.05 in a 32 by 32 grid. Each board is then evolved for 200 steps using GoL rules. The outcome is then classified based on the two conditions of whether it is alive or dead, and its specific type of life. A script calculating neighbor counts for each cell is used to simulate the GoL rules. The purpose of this dataset being random is to collect the naturally occurring proportions of dead samples, alive samples, and various types of life. This dataset is created by `generate_random_dataset.py`

The second dataset generates and classified data samples in the same way, except that it has quotas: 1000 dead samples (meaning samples that die at the 200th timestep), 1000 still-life alive samples, 1000 2-period oscillator alive samples, and 1000 other alive samples. This dataset generates until all 4 quotas are full and does not add more than 1000 of each type of sample.

4 Implementation

4.1 Diffusion Model Architecture

The model architecture consists of a standard U-Net backbone with residual FiLM blocks for class embeddings. The components of the network can be seen in Figure 1 and are elaborated upon in the following table:

4.2 Components of Diffusion Model: Descriptions

Table 1: Network Component Descriptions

Component	**Description**
Time Embedding	A sinusoidal position encoder passed through an multi-layer perceptron. It has 256 dimensions, is expanded to 1024 in the perceptron, and then outputs as 256 dimensions. It has SiLU activation.
Class Embedding	It is a table of learnable embeddings for all forms of life, alive, and death classes. Each have 256 dimensions.
Downsampling Path	There are three residual blocks with channel depths 64, 128, and 256. Each consists of GroupNorm, FiLM (parameterized by the $e_{t,c}$ vector), SiLU, and a 3 by 3 convolution. After this convolution, a time embedding is added. There is a 4 by 4 stride-2 convolution to halve resolution.
Bottleneck	A residual FiLM block at 256 channels.
Upsampling Path	This consists of three stages of convolutions and their transposes used to upsample. It utilized skip-connection concatenation from the downsampling path. A skip is the feature tensor saved from the downsampling path during encoding and then given to the decoding upsampling path. There are three FiLM residual blocks at channels 256, 128, and 64.
Output Head	This consists of a FiLM residual block at 64 channels. It has a 1 by 1 convolution that creates a single-channel noise prediction $\hat{\epsilon}(x_t, t, c)$

The weights are initialized using Kaiming normalization and I apply spectral regularization to promote stable training.

4.3 Training Procedure

Training occurs over 250 epochs. One model trains on the random dataset, and the other trains on the dataset of quotas, producing two different models. We will call them the random model and then quota model.

We use the AdamW optimizer with a learning rate 10^{-4}. The batch size is 32.

For every batch, the following steps repeat:

1. Sample $t \sim \text{Uniform}(1, T)$.
2. Calculate $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$ using $\epsilon \sim \mathcal{N}(0, I)$.
3. Send x_t, t, and condition c through the U-Net and predict $\hat{\epsilon}$.
4. Calculate mean squared-error loss, $L_{MSE} = \|\epsilon - \hat{\epsilon}\|^2$.
5. Backpropagate.
6. Clip gradient at norm 1.0.
7. Update optimizer.

4.4 Sampling and Inference

Depending on the experiment, either 300 or 500 boards are sampled at a time. This means 500 32 by 32 boards are initialized using a threshold probability (for example, 0.5) that each cell is alive at the first timestep. They are each evaluated after 200 timesteps using the rules of GoL before being classified as alive or dead and by type of life. We sample random noise with $x_T \sim \mathcal{N}(0, I)$ and iterate the reverse process with classifier-free guidance parameter $w = 1.0$.

4.5 Reproducibility and Environment Setup

4.5.1 Code and Data Repository

All code, data, and other information exist on GitHub:

`https://github.com/jinschofield/Gen-GOL`

The root of the repository possesses:

- README.md: overview, installation, usage.
- requirements.txt: pinned Python dependencies.
- train.py, phase_2_conditional_diffusion/, finished_models/, data/, scripts/.

I designed the codebase to compartmentalize responsibilities and thus allow for easy maintenance.

Upon opening the root of the repository:

- phase_2_conditional_diffusion/: This generates datasets.
- models/: This possesses definitions of the architecture in unet.py. Logic for diffusion is in diffusion.py.
- scripts/: This possesses the code for evaluation as well as the generation of information used in this paper's figures.
- train.py: This file performs training and possesses functions that help with sampling.
- requirements.txt: This contains dependency specifications.
- notebooks/: This contains the notebook used to interface with the rest of the code base for data generation, training, evaluation, and visualization.
- plot_fig8_embedding_similarity.py: This plots the cosine similarity matrix.
- plot_fig2.py: This is used for figures 2 and 4.
- plot_living_normalized.py: This is used for figures 3 and 5.

- plot_same_category_multi.py: This is used to create the conditional versus unconditional comparison figures.
- plot_sample_visuals.py: This is used to print visual examples of each type of life.
- plot_tsne_film_embeds.py: This is used for t-SNE visualizations.

4.5.2 Environment Setup

Running the cells in the notebook will set up relevant dependencies and clone the repository. A GitHub personal access token and GitHub username will have to be entered into the notebook.

4.6 Dataset Generation

The two files responsible for dataset generation are `generate_random_dataset.py` and `generate_quota_dataset.py`. They use argparse in Python to accept parameters such as output directories, sample counts, threshold (or probability for each individual cell to be alive or dead during initialization), and quotas.

Each script does the following:

1. Initializes random seeds for NumPy and PyTorch.
2. Constructs the output directory if it is absent.
3. Samples 32 by 32 binary arrays with numpy.random.binomial.
4. Imports utils.gol_simulator in order to simulate a time step in the game. For example, computing neighbor counts and making the respective change to the cell's live status.
5. Classifies outcomes using the classify function in label_training_data_32x32.py.
6. Calls np.save() to save each board.
7. Writes rows to a CSV using csv.writer

4.7 Labeling Module

The file `label_training_data_32x32.py` simulates GoL for 200 time steps and then classifies the outcome:

- Simulation using `utils/gol_simulator.simulate()`: This pads a 32×32 board. For each timestep, neighbor sums are calculated using np.roll assuming a toroidal structure (the ends wrap):
- Classification using `classify_grid(arr, timesteps)`: Thus runs simulate(), then
  1. If the final frame has all dead cells, it returns ”died_out”.
  2. Otherwise, it searches backward in history for a match p steps back:
     – p = 1: ”still_life”
     – p = 2: ”oscillator_period_2”
     – none: ”others”
- CSV output: This uses Python's built-in csv.writer to write to a CSV.

While I attempted to also classify for larger periods, gliders, and spaceships, I decided to only classify period-2 oscillators and still-life for a few reasons. Firstly, I did not classify space ships because upon manual scanning, my generated datasets seemed to almost never create them. Gliders were more common but I could not find a computationally efficient way to accurately find them all without false negatives or false positives. Finally, checking for oscillators of over period 2 was not common because they did not naturally occur often.

The choice of making the edges toroidal was made in order to avoid having to make special rules for edges and corners. There is no definite set of rules for Conway's Game of Life for cells without eight neighbors, thus I used the convention of making the grid toroidal. This meant that edges and corners would wrap to the opposite side of the grid.

4.8 Model Code

In `models/unet.py`, the UNet class does the following:

- Creates the SinusoidalPosEmb module, which calculates time embeddings via sine/cosine functions.
- It builds a multi-layer perceptron to project the embedding: SinusoidalPosEmb $\rightarrow$ Linear $\rightarrow$ SiLU $\rightarrow$ Linear
- It instantiates nn.Embedding(num_classes, time_emb_dim) for class labels, which is summed with the time embedding during conditioning.
- It defines the ResidualBlock class, which:
  - Applies GroupNorm to the input x.
  - Calculates FiLM parameters γ_1, β_1 using a linear layer on the time embedding t, then applies: $h \leftarrow h \cdot (1 + \gamma_1) + \beta_1$.
  - Activates with SiLU: $h \leftarrow \text{SiLU}(h)$.
  - Applies a 3×3 Conv2d: $h \leftarrow \text{Conv2d}(h)$.
  - Adds the time-MLP skip connection: $h \leftarrow h + \text{MLP}(t)$.
  - Applies GroupNorm to h.
  - Calculates the FiLM parameters γ_2, β_2 from t and applies: $h \leftarrow h \cdot (1 + \gamma_2) + \beta_2$.
  - Activates with SiLU again.
  - Applies a 3×3 Conv2d.
  - Adds the residual connection: output $x + h$.
- It defines UNet.__init__, which builds the following:
  - self.downs: This is meant for down-sampling and is a structure of alternating ResidualBlock $\rightarrow$ Conv2d(stride=2) that halves the spatial dimensions
  - self.bottleneck: This consists of one ResidualBlock at the lowest resolution.
  - self.ups: This is meant for upsampling and is a structure consisting of ConvTranspose2d(stride=2) $\rightarrow$ ResidualBlock with skip-connection concatenation.
- The forward(x, t, c) method:
  1. Calculates t_emb = time_MLP(SinusoidalPosEmb(t)).
  2. If c is provided, adds class_emb(c) for use of the class embedding.
  3. Runs the down-sampling path and stores skip maps.
  4. Applies the bottleneck.
  5. Runs the up-sampling path, concatenating each skip.
  6. Applies a final ResidualBlock and a 1×1 Conv2d to output the noise prediction.

The use of SinusoidalPosEmb allows for more stable training due to the embeddings being smoothly changed across timesteps, rather than having large abrupt changes.

4.9 Diffusion Engine

In `models/diffusion.py`, the Diffusion class is defined. The class contains the forward and reverse processes:

- Initialization (__init__) This precalculates $\{\beta_t\}$, $\{\alpha_t\}$, and $\{\bar{\alpha}_t\}$ for either a linear or cosine noise schedule.
- cosine_beta_schedule(timesteps, device, s=0.008) This implements the Nichol Dhariwal cosine schedule for $\bar{\alpha}_t$ and returns the per-step β_t.
- q_sample(x_start, t, noise) This calculates the forward diffusion: $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$, $\epsilon \sim \mathcal{N}(0, I)$.
- p_losses(model, x_start, t, c=None) This samples noise and computes MSE on the predicted noise. There is code for implementations of MAE, SSIM, and BCE loss, but the weights for these are defaulted to zero.

- p_sample(model, x, t, c=None) This performs one reverse step using classifier-free guidance.
- ddim_sample(model, shape, eta=0.0, c=None) This is a deterministic DDIM sampler ($\eta = 0$), similar to p_sample but not incorporating stochastic noise.
- sample(model, shape, c=None) This causes the full reverse chain to run from $t = T$ down to 1 by calling p_sample at each step.

4.10 Notebook

The notebook organizes set-up such as cloning of the GitHub repository and downloading of dependencies. It proceeds to then generate the relevant datasets, train the two models using these datasets, and generate the data shown in the evaluation section.

4.11 Hyperparameter Ablation

Although a fully controlled grid search was not possible due to compute constraints, ablation studies were performed to assess the best hyperparameters.

Techniques that were tested but ultimately not used included using a multiplier on the loss for ”live” cells to reinforce live cell occurrence, SSIM (Structural Similarity Index) loss, EMA decay (Exponential moving average), BCE (binary cross-entropy) weight, loss-ramping with SSIM and BCE, classifier-free guidance dropout, a pixel-wise L_1 mean absolute error loss term, and L_2 weight decay.

Hyperparameters that were used are the following:

Table 2: Hyperparameter Settings

Hyperparameter	**Tuned Value**
Noise Schedule	Cosine
Epoch Number	250
Gradient Clipping	The cap was at 1.0.
The use of FiLM layers.	N/A
Learning Rate Scheduler	Cosine
Classifier-free Guidance Drop-out.	0.1

The new, unintroduced concepts above are the learning rate scheduler and classifier-free guidance dropout. The learning rate scheduler being cosine means that the learning rate follows a half-cosine decay over the epochs and ends at nearly zero. This stabilizes convergence. Classifier-free guidance dropout at 0.1 means that during training, the conditioning signal is dropped 10 percent of the time, causing the model to learn conditional and unconditional sampling modes. This trains the value ϵ_{uncond} in $\epsilon_{guided} = \epsilon_{uncond} + w * (\epsilon_{cond} - \epsilon_{uncond})$, which allows the guidance scale value w to be useful during inference.

4.12 Previous Approaches

Prior to attempting to use this standard conditional DDPM approach, I aimed to understand Conway’s Game of Life using Diffusion-of-Thoughts, a framework that can apply diffusion to strings of text, or in this case, notation representing a grid of cells. The aim of this task was still to create abstract representations that could represent logical relationships. The particular task was to predict the grid at the next time step given a grid at a previous time step. Ultimately, my test accuracies were very low due to the inability for the Diffusion-of-Thoughts model to perfectly understand the logical rule as I trained it. I am unsure whether this was because the model is incapable of distilling down the logical rules from unsupervised datasets of text descriptions of rules and sample evolutions as well as supervised datasets, or whether it was an error of my not giving enough information to the model. I decided to find a different way to try to distill down logical relationships in embeddings in a manner that would be able to track incremental, imperfect progress. Thus, I stopped using a model

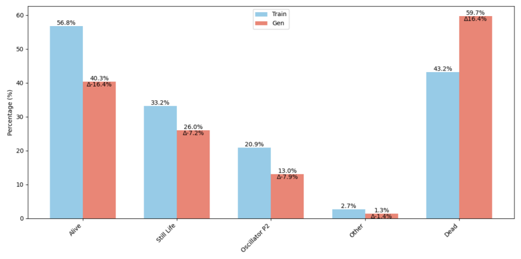


Figure 2: Percentage of Each Type of Generations Between Random Training Dataset and Unconditioned Generations from Diffusion Model Trained on Random Dataset

that also possessed text embeddings and only focused on diffusion on binary data and evaluating on a metric that was more interpretable, outcomes of generativity.

5 Evaluation

5.1 Metrics

There does not exist a standardized benchmark against which I can compare the success of my conditional embeddings at conditioning for specific traits in GoL cellular automata. The evaluation section will be separated into two parts: quantitative and qualitative. Firstly, I will demonstrate the efficacy of trained conditional embeddings quantitatively by showing how they increase the occurrence of their class. In the second qualitative part, I will explore different manifestations of still-life, period-2 oscillator life forms, and other forms of life. I will also present a t-SNE of the conditioned outputs of different GoL outcomes.

5.2 Quantitative Evaluation

We present five sets of figures.

Firstly, we trained a conditional diffusion model on the random dataset constructed with no quotas. The categories, which are also the various conditions, are alive, dead, still-life, period-2 oscillator, and other form of life after 200 timesteps. Samples may possess more than one class label, such as being both alive and a still-life. We compare each class' natural occurrence in the random dataset to their occurrence when sampling from the diffusion model trained on this dataset with unconditioned generations. This seeks to see the ability of the unconditioned sampling to replicate various forms of life.

Using an individual probability of each cell being alive of 0.5 (we refer to this as the threshold), and a sample generation of size 500, Figure 2 and Figure 3 are generated. While the diffusion model produced more dead and less alive configurations in Figure 2, this is subject to the threshold, which was set at 0.5, and thus no real conclusions can be made between the difference between alive and dead in the training dataset and the generations from the model. When we normalize percentages for only the types of life for alive configurations in Figure 3, we can see that generations from the mode increase the proportion of life that are still life, decrease the proportion of period-2 oscillators, and marginally decrease the proportion of the other category. This seems to point towards more complex structures, such as period 2 oscillators, being more difficult for the model to form without conditioning.

In the quota dataset, there are set quotas for proportions of dead, alive, and each category of life. In the quota dataset, as seen in Figure 4, dead samples are once again easier to generate than alive samples from the unconditioned model, although this is likely an effect of the threshold being 0.5.

Secondly, we trained a conditional diffusion model on the quota dataset. We compare each class' occurrence in the quota dataset to their occurrence when sampling from the diffusion model trained on this dataset with unconditioned generations. This seeks to see the ability of the unconditioned

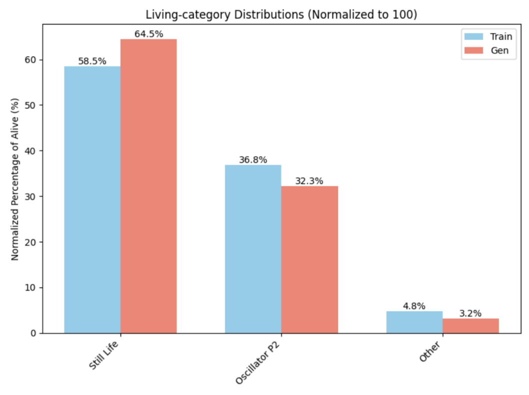


Figure 3: Percentage of Each Type of Living Generations Between Random Training Dataset and Unconditioned Generations from Diffusion Model Trained on Random Dataset

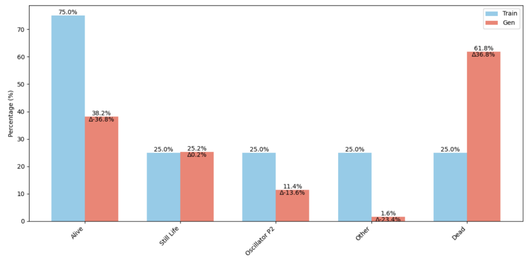


Figure 4: Percentage of Each Type of Generations Between Quota Training Dataset and Unconditioned Generations from Diffusion Model Trained on Quota Dataset

sampling to replicate various forms of life given that there are sufficient examples of rare forms of life in the training dataset. The results can be seen in Figure 4 and Figure 5.

The living-only chart, Figure 5, shows that unconditioned generations seem to favor still life and period-2 oscillators far more heavily than other forms of life, which are by definition more complex, since still life and period-2 oscillators are the simplest. There is a much larger jump in still life production than period-2 production, further supporting that simpler structures are modelled more easily.

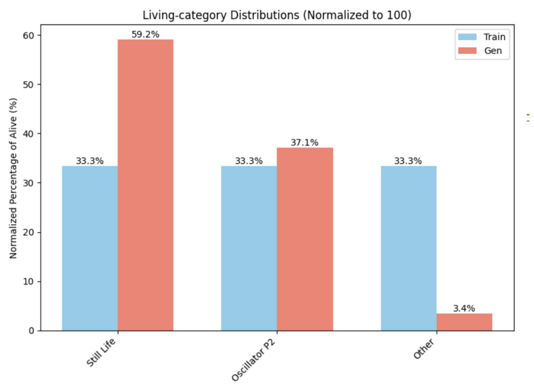


Figure 5: Percentage of Each Type of Living Generations Between Quota Training Dataset and Unconditioned Generations from Diffusion Model Trained on Quota Dataset

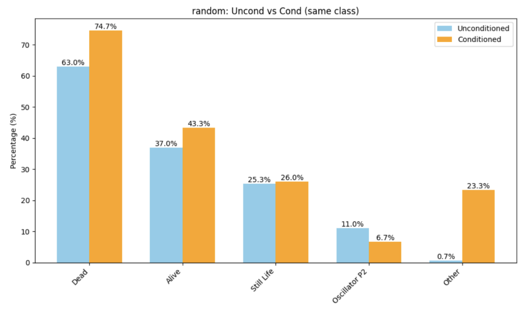


Figure 6: The Occurrence of Each Class Between Unconditioned and Targeted Conditioned Generations When Trained on the Random Dataset

Thirdly, we compare each class' occurrence during unconditioned sampling with their occurrence with each condition applied. This seeks to find the efficacy of each conditional embedding at selecting for cellular automata of that kind given the random dataset. We present the novelty of generated samples. Particularly, we ensure that no generated configurations are direct clones of initial frames in the training data, invariant to reflections and rotations. We do not check for invariance to translation due to the compute associated with translation checks for all 300 samples across all 4000 training steps, as well as that we still consider translations to be valid since we intend for the diffusion model to pick up on patterns and replicate them. We check for novelty to ensure that the model is not memorizing entire 32 by 32 grids. The results can be seen in Figure 6 and Figure 7.

Figure 6 demonstrates that, when trained on a random dataset, and when training conditional embedding on each of the five categories in the figure, the application of the conditional embedding increases the proportion of generations containing that type of life. The threshold was 0.3. For example, the unconditioned bars (excluding alive, which will repeat with the individual life categories) all add to 100 percent because they represent the same run, and the proportions of behavior from that one run. Each of the five conditioned bars represent a different run, thus they do not add to 100 percent. For example, the alive conditioned bar represents the generation from the model when only the alive condition is applied. This results in a larger proportion of the generations being alive, as seen when compared to the no condition run. The highest increases in proportion from unconditioned to conditioned are dead and alive, likely because they represent broad patterns. Other improves slightly, still-life marginally improves, and period-2 oscillator decreases. This decrease in the case of period-2 oscillators can likely be attributed to how the random dataset does not have quotas for specific types of life and thus the model trained on the random dataset did not get enough representative samples to train an embedding that could replicate the type of behavior. The novelty rate of all 5 runs was 100 percent, meaning no frame was a direct rotation or reflection of an image in the training set.

Fourthly, we trained a conditional diffusion model on the quota dataset to generate Figure 7. We compare each class' occurrence during unconditioned sampling with their occurrence with each condition applied. This seeks to find the efficacy of each conditional embedding at selecting for cellular automata of that kind given the quota dataset. We present the novelty of generated samples.

Figure 7 is the same as Figure 6 except the model was trained on the quota dataset, thus providing the model with a large enough dataset for each type of life, regardless of rareness. Both alive and dead conditions increase proportion. Between still life and period-2 oscillators, the increase in proportion is much larger for still life. This supports the idea that more complicated patterns are more difficult for the diffusion model to learn. The other section decreased, likely becuase the category is a catch-all for other forms of life and does not have a consistent meaning.

Finally, we create a 5 by 5 cosine similarity matrix comparing the similarities between the five conditional class embeddings for the following classes: alive, dead, period-2 oscillator, still life, and other life. This can be seen in Figure 8.

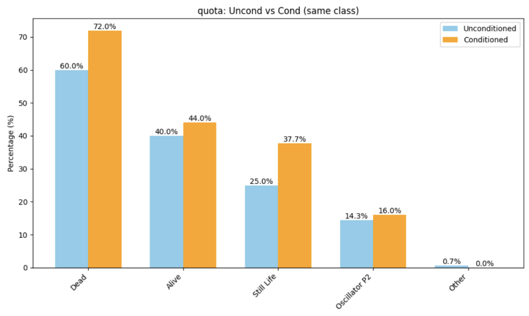
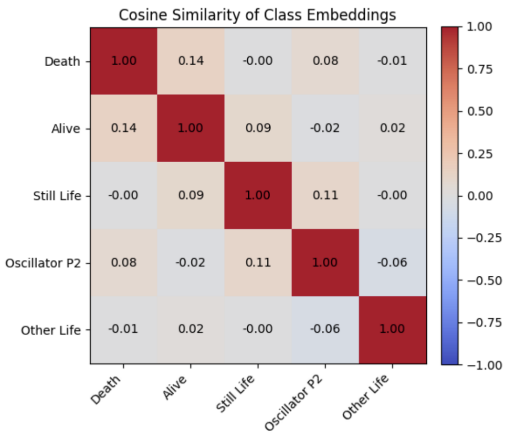


Figure 7: The Occurrence of Each Class Between Unconditioned and Targeted Conditioned Generations When Trained on the Quota Dataset



	Death	Alive	Still Life	Oscillator P2	Other Life
Death	1.00	0.14	-0.00	0.08	-0.01
Alive	0.14	1.00	0.09	-0.02	0.02
Still Life	-0.00	0.09	1.00	0.11	-0.00
Oscillator P2	0.08	-0.02	0.11	1.00	-0.06
Other Life	-0.01	0.02	-0.00	-0.06	1.00

Figure 8: Cosine Similarity Between Class Embeddings

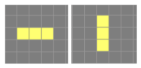
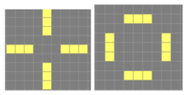
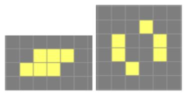
Structure	Depiction: Two images represent different configurations during the period.	Count
F: A rotating 1 by 3 line		85
G: 4 Fs formed in a specific configuration		24
H: Parallel Bars		1
1 F and 1 G in One Sample	Any possible configuration.	3
2 F's in One Sample	Any possible configuration.	1
3 F's in One Sample	Any possible configuration.	2
1 F and 1 g in One Sample	Any possible configuration.	1

Figure 9: Samples of Still-Life Generated From Model Trained on Quota Dataset

5.3 Qualitative Evaluation

In the following section, yellow cells indicate alive cells and grey cells indicate dead cells.

Interestingly, the alive and death conditions are the most similar, followed by the relationship between still-life and period-2 oscillator, the relationship between death and period-2 oscillator, and the relationship between alive and still-life. The least similar are other life and period-2 oscillators. It seems that similarity seems to reflect the reliance on structure rather than living or dying. For example, death and period-2 oscillation might be very different because period-2 oscillation requires on the existence of structure where death does not. This is not entirely consistent throughout the matrix, so it is unclear whether the cosine similarity of embeddings given there are only five can yield meaningful results.

We provide four figures. Firstly, we show common forms of still-life found in the generations from the model trained on the quota dataset. Samples were generated until 100 still-lifes were collected. In the chart below, the counts for individual structures will be the count including if more than one structure occurred in a single generation (i.e. if there are two of type A in a single structure, the count is incremented by 2). I also add counts of combinations of structures. This chart reinforces the idea that simpler patterns are more easily generated by the model, although there is room for diversity. This is demonstrated by the dominance of Pattern A but the interesting presence of pattern E. The frequency of combinations of structures suggests that the diffusion model is learning to create these structures independently.

Secondly, we show common forms of period-2 oscillators found in the generations from the model trained on the quota dataset. Samples were generated until 100 two-period oscillators were collected.

This chart once again reinforces that simpler patterns are easier for the diffusion model to learn due to dominance of pattern F. Pattern G was also very interesting to observe as it is just a combination of 4 F's. This suggests that the diffusion model also learns combinations of structures. It is unclear whether this means that the diffusion model understands modularity or if it is memorizing these large-scale combinations as well.

Thirdly, we present t-SNE visualizations of the embeddings that denoise the intermediate outputs at each time step. At each time step, a time embedding (pre-noising) and time embedding and class embedding combination (post-noising) are plotted. t-SNE visualizations of intermediate outputs were created but there were no discernible clusters or differences between inclusion of a class, thus the figure was not included in this report.

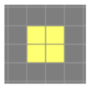
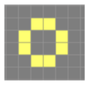
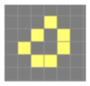
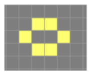
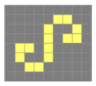
Structure	Depiction	Count
A: 2 by 2 grid		71
B: Cornerless 4 by 4 square		3
C: Half square		8
D: Cornerless rectangle		31
E: S shape		1
2 As in One Sample	Any possible configuration.	7
1 A and 1 D in One Sample	Any possible configuration.	3
2 Ds in One Sample	Any possible configuration.	1

Figure 10: Samples of Period 2 Oscillators Generated from Model Trained on Quota Dataset

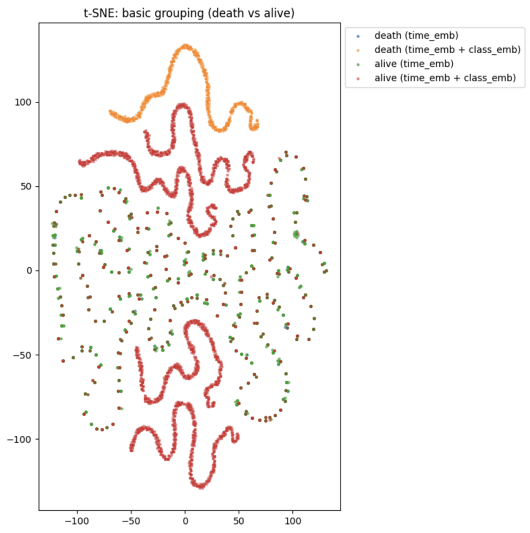


Figure 11: t-SNE: Basic Grouping (Death vs. Alive) for Pre-Noise (Time Embedding Only) and Post-Noise (Class and Time Embedding)

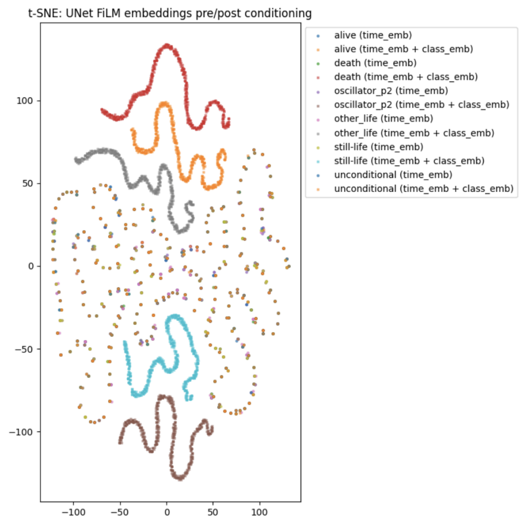


Figure 12: t-SNE: All 5 Class Embeddings for Pre-Noise (Time Embedding Only) and Post-Noise (Class and Time Embedding)

Figure 11 is a visualization of the pre-noise and post-noise embeddings for the dead and alive conditions. Each point represents an occurrence of the embedding at a timestep.

There is a clear structure separating plotted points from one another if they do not represent the same combination of embeddings except in the case of the alive time only embedding, the alive time and class embedding, and the death time only embedding. It is interesting to see how the death time and class embedding exists at the edge of the structure. This is very different from the death time only embedding which exists exclusively intermingled with the alive time only embedding and the alive time and class embedding. The clustering of the alive time and class embedding in two different areas is also interesting to observe. The formation of curved lines is especially interesting.

I wonder whether the mixing of the embeddings in the center would be separated if one dimension is changed. For example, if we imagine this 2-D visualization as a projection of a 3-D structure, if we were to observe it from its top, we might see the red and orange clusters mixed. Thus, the separation of clusters is not immediately interpretable. Figure 12 is similarly structured to the last except all embeddings exist in one cluster. The still life time and class embedding and the oscillator time embedding exist at the bottom. The time and class embedding, the alive time and class embedding, and the death time and class embedding are distinct and all at the top. Every other embedding is mixed in the center. The same question of rotation as in the previous figure remains.

I cannot discern a pattern in the structure of the visualization that might link back to the meaning of the embedding placements in either of the figures above.

6 Summary

Through my evaluation, I demonstrate that conditional diffusion in the form of a denoising diffusion probabilistic model using classifier-free guidance and FiLM-based class embeddings can effectively learn and increase the occurrence discrete cellular automata patterns in the Conway's Game of Life environment. In particular, it is capable of distilling embeddings that yield conditional sampling that increases the prevalence of targeted conditioned forms of life and behavior in the cellular automata. It can be observed that patterns of simpler structure are easier to generate using diffusion. Additionally, diffusion can easily learn combinations of smaller structures. Novelty metrics confirm diversity of the generations as not being clones of the training set 32 by 32 grids. Visualizations of motifs and t-SNE visualizations of the embeddings demonstrate the various types of patterns that the diffusion model can emulate as well as the distinct difference between embeddings of different conditions.

This establishes conditional diffusion as a technique for generative modelling of logical rule-based discrete environments.

One limitation is the size of the training sample, which was 4000 samples for either dataset. This was limited to 4000 to ease compute use due to limited resources and time, but could hae potentially led to underrepresentation of rare forms of life. This would result in biased outcomes.

A second limitation is that only two forms of life were properly categorized, still-life and period-2 oscillators. Oscillators of larger than period-3 as well as gliders and spaceships were not categorized due to the increased time complexity of such operations. This limitation means that the outcomes of this technique on many lfie forms in Conway's Game of Life are unknown.

A third limitation is the use of the 32 by 32 grid as the only grid upon which these embeddings act. The effects of this are twofold. Firstly, this limits the size of configurations that could be explored in this project. Second, the generalizability of conditional embeddings to smaller or larger grids is unknown.

Immediate next steps would be to scale to larger grid resolutions to explore the generalizability of the conditional embeddings. Additionally, conducting the same analysis on oscillators of larger periods as well as gliders and spaceships would be interesting. Applying this framework to various cellular automata rule sets such as Brian's Brain in order to evaluate the generalization of the technique as a whole on increasingly complex environments would also be very insightful.

REFERENCES

Arjovsky, M., Chintala, S., & Bottou, L. (2017). *Wasserstein GAN*. arXiv:1701.07875.

Austin, J., Johnson, D. D., Ho, J., Tarlow, D., & van den Berg, R. (2023). *Structured denoising diffusion models in discrete state-spaces*. arXiv:2107.03006.

Dhariwal, P., & Nichol, A. (2021). *Diffusion models beat GANs on image synthesis*. arXiv:2105.05233.

Gardner, M. (1970). The fantastic combinations of John Conway's new solitaire game 'life'. *Scientific American*, 223(4), 120-123.

Gilpin, W. (2020). *Cellular automata as convolutional neural networks*. arXiv:1809.02942.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). *Generative adversarial networks*. arXiv:1406.2661.

Ho, J., Jain, A., & Abbeel, P. (2020). *Denoising diffusion probabilistic models*. arXiv:2006.11239.

Jang, E., Gu, S., & Poole, B. (2017). *Categorical reparameterization with Gumbel-Softmax*. arXiv:1611.01144.

Kingma, D. P., & Welling, M. (2022). *Auto-encoding variational Bayes*. arXiv:1312.6114.

Liu, W., Ren, G., Yu, R., Guo, S., Zhu, J., & Zhang, L. (2022). *Image-adaptive YOLO for object detection in adverse weather conditions*. arXiv:2112.08088.

Liu, X., Zhang, L., & Guan, H. (2023). *Uplifting message passing neural network with graph original information*. arXiv:2210.05382.

Mitchell, M. (1996). *Evolving cellular automata with genetic algorithms*. In Proceedings of the First International Conference on Genetic Algorithms (pp. 125-131).

Mordvintsev, A., Randazzo, E., Niklasson, E., & Levin, M. (2020). Growing neural cellular automata. *Distill*. https://distill.pub/2020/growing-ca

Nichol, A., & Dhariwal, P. (2021). *Improved denoising diffusion probabilistic models*. arXiv:2102.09672.

Perez, E., Strub, F., de Vries, H., Dumoulin, V., & Courville, A. (2017). *FiLM: Visual reasoning with a general conditioning layer*. arXiv:1709.07871.

Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., & Ganguli, S. (2015). *Deep unsupervised learning using nonequilibrium thermodynamics*. arXiv:1503.03585.

Tapia-McClung, R., & Hernandez-Montoya, V. (2020). A diffusion approach to the dynamics of Conway's Game of Life: Emergence of multiple power law fluctuation regimes. *Chaos, Solitons & Fractals*, 140, 110213.

Wolfram, S. (1984). Cellular automata as models of complexity. *Nature*, 311(5985), 419-424.

You, J., Ying, R., Ren, X., Hamilton, W., & Leskovec, J. (2018). GraphRNN: Generating realistic graphs with deep auto-regressive models. In *Proceedings of the 35th International Conference on Machine Learning* (pp. 5708-5717).