Probabilistic Sequence Modeling of Cellular Automata

Daniel Israel, Lukas Brockenbrough, Minhao Ren
Advisor: Demetri Terzopoulos

CS275: Artificial Life for Computer Graphics and Vision

Abstract

This project investigates the application of probabilistic sequence modeling using large language models (LLMs) to simulate the behavior of one dimensional elementary cellular automata (ECA). We trained an autoregressive Transformer on sequences representing cellular states across time, aiming to predict the evolution of automaton patterns given specific rules. Despite the model's capacity to capture local structures and short term dependencies, our results suggest that LLMs struggle to fully learn or generalize the underlying deterministic rules governing ECA behavior. This highlights limitations in using standard sequence models for tasks that require precise algorithmic reasoning, pointing to the need for more specialized architectures or training strategies in such settings.

1 Introduction

Cellular automata (CA) are simple, rule-based systems that are capable of producing surprisingly complex behavior. Among them, one dimensional elementary cellular automata (ECA) have been widely studied for their capacity to model emergent phenomena despite their minimalist construction. Given the growing interest in using large language models (LLMs) to perform general-purpose sequence modeling, this project explores the feasibility of applying autoregressive Transformers to predict and generate ECA patterns. While such models have demonstrated remarkable success in natural language and code generation, their ability to internalize deterministic algorithmic processes remains uncertain.

In this work, we train a Transformer based model to learn ECA behavior from data. Each input sequence encodes the rule governing the automaton and a series of observed time steps, with the goal of predicting subsequent states. Ultimately, our results indicate that, although LLMs can capture short range dependencies and some structure in the automaton's evolution, they fall short of fully understanding or reproducing the deterministic rules of ECA. This finding contributes to ongoing discussions about the limitations of LLMs in algorithmic reasoning tasks.

2 Background

2.1 Elementary Cellular Automata

Elementary cellular automata (ECA) are discrete dynamical systems consisting of a one-dimensional array of binary cells updated synchronously in discrete time steps. The next state of each cell is determined by its current value and those of its immediate neighbors. Each possible 3-bit neighborhood maps to a new state, yielding 8 possible input combinations. As each of these can result in either a 0 or 1, there are 256 distinct ECA rules.

Despite their simplicity, ECAs display a wide range of behaviors, from fixed points and simple periodic patterns to chaos and apparent randomness. This makes them valuable for studying complexity, emergence, and the computational capabilities of simple systems.

2.2 Large Language Models for Sequence Modeling

Autoregressive Transformers have achieved state-of-the-art results across a range of sequence modeling tasks, particularly in natural language processing. By predicting each token in a sequence conditioned on the preceding tokens, they learn probabilistic representations of underlying structures in data. However, when applied to algorithmic tasks such as sorting, copying, or simulating deterministic rules, LLMs often fail to generalize, suggesting a gap between statistical learning and algorithmic reasoning.

3 Methodology

3.1 Data Generation

The training dataset consists of one million sequences, each generated from one of the 256 elementary cellular automaton rules. Each sequence represents a CA evolution sampled at 7 timesteps, with each timestep capturing the state of 256 cells. The tokenization scheme is defined as follows:

- Rule numbers 0-255 are mapped to token IDs 5-260.
- Timesteps 0-255 are mapped to token IDs 261-516.
- Cell states 0 and 1 are mapped to token IDs 3 and 4, respectively.
- Special tokens are assigned as: PAD=0, BOS=1, EOS=2.

Each training sequence is tokenized in the following format:

```
[RULE] [TIME_0] [256 bits] [TIME_1] [256 bits] ... [TIME_14]
```

Two sampling methods are employed during data generation:

- **Random sampling:** Timesteps are randomly selected from the complete simulation history.
- **Contiguous sampling:** A continuous block of timesteps is selected starting from a random point.

The ratio between these two methods is controlled by the contiguous sampling ratio parameter, with a default value of 0.7 for contiguous sampling. This structure enables the model to learn temporal dependencies and rule-conditioned evolution patterns from partial trajectories.

3.2 Model Architecture

We use the Pythia-160M model, an autoregressive Transformer-based language model. It consists of:

- 12 decoder layers
- Hidden size (`d_model`): 768

- Feedforward size (`d_ff`): 3072
- 12 attention heads
- Context length: 2048 tokens
- Vocabulary size: 517 (designed to encode rule indices, time steps, and bit states)

The model processes each training sample as a sequence of discrete tokens, without any explicit spatial or temporal inductive bias. This tests the ability of standard sequence models to learn rule-driven deterministic dynamics from data alone.

3.3 Training Procedure

The training objective is an autoregressive next-token prediction log-likelihood objective, where the model is tasked with forecasting the subsequent token within a given partial sequence of a cellular automaton's evolution. The base architecture for this study utilizes pre-trained Pythia language models from EleutherAI. To adapt these models for the cellular automaton (CA) task, their token embeddings are resized to accommodate a custom vocabulary of 1,024 tokens. This vocabulary comprises 256 rule tokens, 256 time tokens, 2 bit tokens, and additional special tokens for padding, beginning-of-sequence, and end-of-sequence.

Training is performed using the Hugging Face Transformers library with the following hyperparameters:

- **Learning rate:** 1×10^4 with a linear warmup phase over 500 steps.
- **Batch size:** Configurable per-device batch size, with gradient accumulation employed to achieve an effective batch size of 32.
- **Optimization:** AdamW optimizer with a weight decay of 0.01.
- **Training epochs:** 3 epochs over the entire dataset.
- **Evaluation:** Conducted every 500 steps on a validation set comprising 1,000 sequences.

Sequences are processed to align with the model's maximum context window, which is determined by the base model's maximum position embeddings. Longer sequences are truncated, while shorter sequences are padded with PAD tokens. An attention mask is utilized to ensure that padded tokens do not contribute to the loss computation. The training procedure incorporates early stopping based on validation loss. At the conclusion of training, the best-performing model checkpoint is loaded, ensuring that the final model selected exhibits optimal generalization performance. Training is performed across all 256 rules to promote generalization and the potential for shared representations. The entire dataset consists of one million such samples, with sequences randomly shuffled and batched during training. No specialized architectural modifications are introduced; the primary goal is to evaluate whether a standard language model can infer and reproduce Elementary Cellular Automata (ECA) dynamics based solely on token patterns.

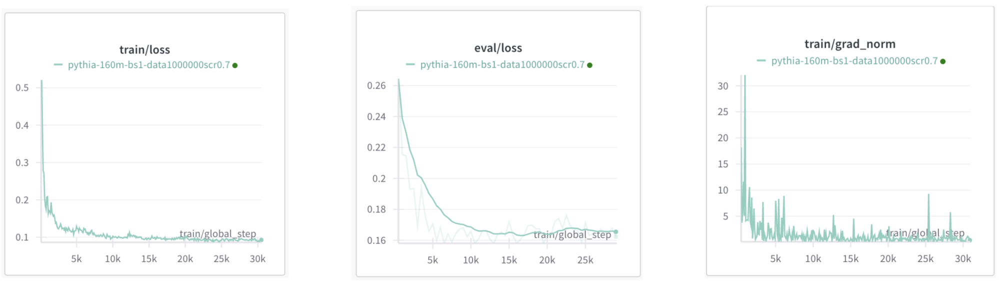

Figure 1: Training metrics: training loss, evaluation loss, and gradient norm over steps for Pythia-160M.

As depicted in Figure 1, both the training and evaluation loss decrease rapidly in the early stages of training and subsequently stabilize. This trend suggests that the model effectively fits the training data. Gradient norms remain well-behaved throughout the training process, with occasional spikes likely corresponding to dynamic changes in the learning process. However, despite these positive trends, the model's generalization ability over longer horizons remains limited, a phenomenon further explored in subsequent sections of this paper.

3.4 Evaluation

We evaluate the model using both quantitative and qualitative methods:

- **Visual inspection** of generated vs. ground truth trajectories to assess structural fidelity.
- **Bitwise evaluation**, where the generated and ground truth are used to find the proportion of correctly predicted cells.
- **Local patch evaluation**, where small spatial-temporal patterns (e.g., 3x3 or 5x5) are compared for matches and hallucinations.
- **Timestep evaluation**, where bitwise accuracy is found at each timestep.

This multi-pronged evaluation approach helps reveal both the surface-level accuracy and deeper structural understanding (or lack thereof) captured by the model.

4 Results

The following results are based off how well the model predicted the cell environment where the starting state has only the middle cell on for each rule.

4.1 Qualitative Evaluation

To assess how well the model replicates the global structure of cellular automata, we compare generated sequences to ground truth simulations across representative ECA rules.

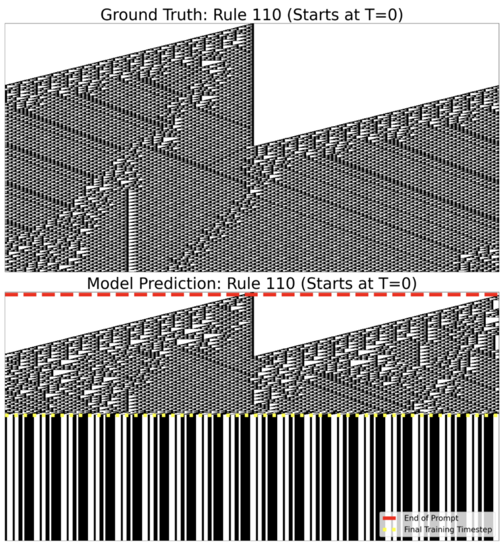

Figure 2: Rule 110: The model initially matches the true structure but collapses to noise past the prompt region.

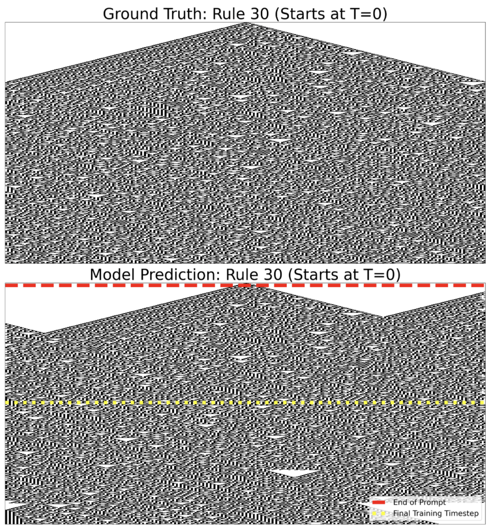

Figure 3: Rule 30: The model captures short-term local patterns but fails to sustain chaotic dynamics beyond training.

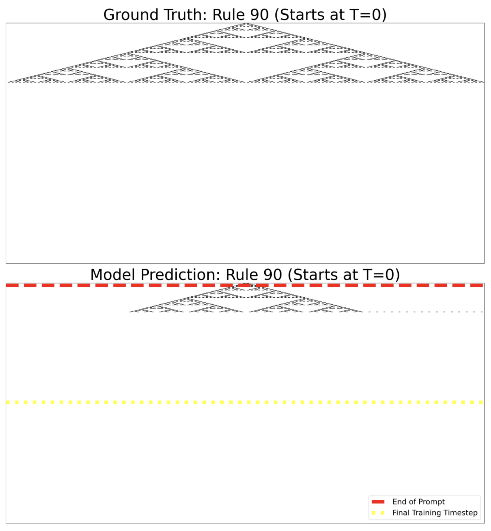

Figure 4: Rule 90: Partial structural fidelity is preserved, but repetition and fading begin shortly after the prompt.

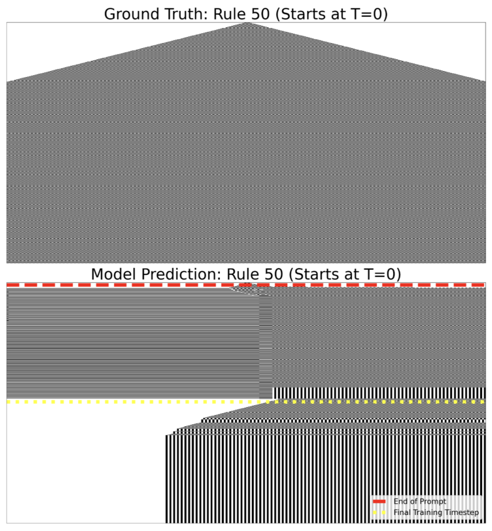

Figure 5: Rule 50: The model prediction becomes periodic and degenerates into vertical striping.

In Figures 2, 3, 4, 5, and **??**, the red dashed line indicates the boundary between the input prompt and the model-generated region, and the yellow dotted line in each prediction marks the final training timestep. While the model demonstrates some success in copying short-term structure, it struggles to extrapolate or maintain coherence across long horizons, especially for complex rules like 30 or 110. The model struggles even before the final training timestep and has worse performance beyond this timestep. These failures highlight the limitations of standard LLMs in capturing deterministic, rule based dynamics over time.

4.2 Quantitative Evaluation

4.2.1 Bitwise Accuracy

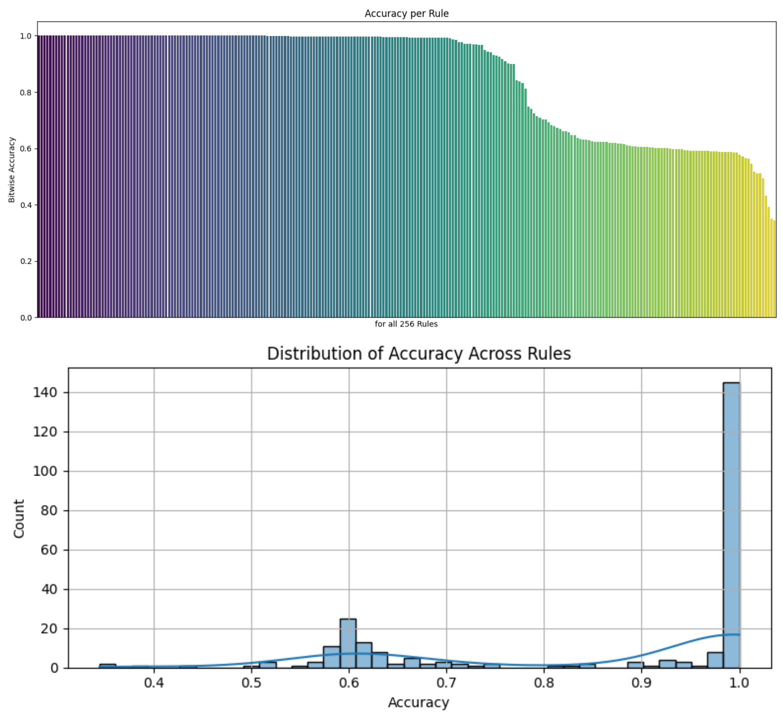

Figure 6: Bitwise accuracy across rules

One way we evaluate the success of the model is its bitwise accuracy, or the number of correctly predicted cells divided by the total number of cells, for each rule is shown. As seen in Figure 6, many rules are almost fully correctly predicted. Specifically, 78 rules ($\sim$ 30.5%) have a perfect bitwise accuracy, and 154 rules ($\sim$ 60.2%) have a bitwise accuracy of at least 95%. However, many rules also have an accuracy of around 60%. Specifically, 62 rules ($\sim$ 24.2%) have an accuracy between 55% and 65%. While the majority of rules are well-predicted by model, there are many rules with an accuracy that is closer to 50%, indicating that at a bitwise scope, many rules are not predicted much more successfully than a random model.

4.2.2 Local Patch Evaluation

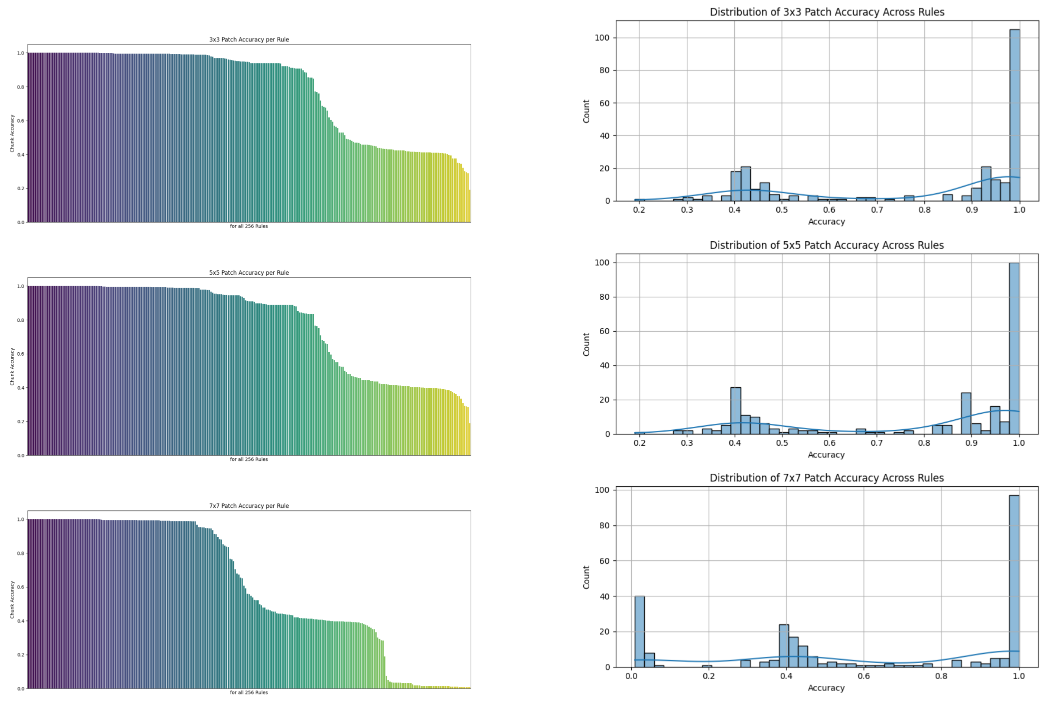

Figure 7: Local patch accuracy across rules

Another way we evaluate the success of the model is by testing its success in local patches of varying sizes (3, 5, and 7). For each patch, we find its squared bitwise accuracy and average across all chunks. We square bitwise accuracy to more harshly punish incorrect patches and to better reward correct patches. The local patch accuracies and their distributions for each patch size are shown in Figure 7. As can be seen, similar to the distribution of bitwise accuracies across rules, the 3x3 and 5x5 patch accuracies have their main peak at around 1 and another lower peak, though this peak is now around 0.4. However, the distribution of 7x7 patch accuracies has another peak at around 0, showing that there are many rules that the model cannot accurately predict over a larger scope. Fittingly, the rules with the worst bitwise accuracy, 3x3 patch accuracy, and 5x5 patch accuracy are the same, while the rules with the worst 7x7 patch accuracy (153, 187, and 247) are different than the rules that are worst at the other metrics.

4.2.3 Timestep Accuracies

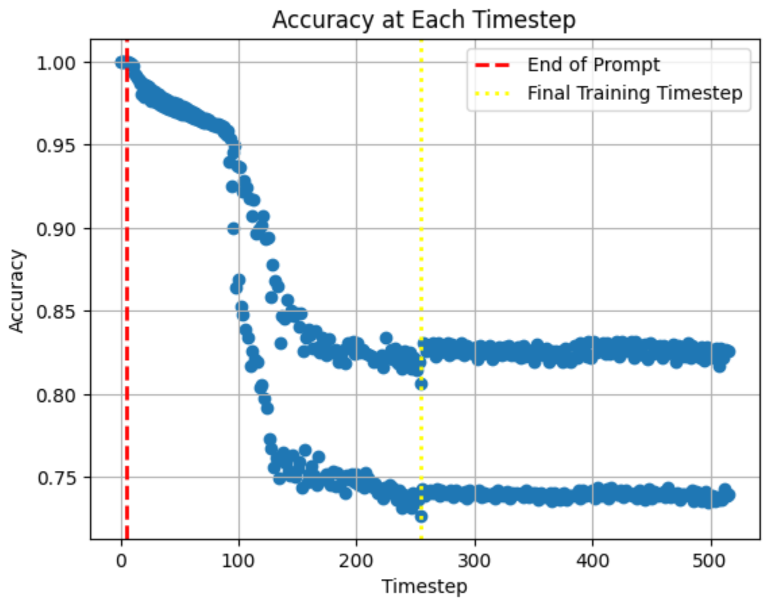

Figure 8: Timestep accuracy across rules

The last way we evaluate the success of the model is by finding its bitwise accuracy at each timestep. In Figure 8, the red dashed line indicates the boundary between the input prompt and the model-generated region, and the yellow dotted line in each prediction marks the final training timestep. As seen in this graph, the timestep accuracy is 1 until the end of the prompt, then decreases linearly until the timestep is about 90. Then, timestep accuracy sharply decreases until the timestep is about 170. Then, the timestep accuracy stays about the same for the rest of the predicted timesteps. Interestingly, at around timestep 100, the accuracy at even timesteps starts to diverge from the accuracy at odd timesteps. Specifically, the accuracy of the odd timesteps levels out at about 83%, while the accuracy of even timesteps levels out at about 74%. This is likely because the model fails to replicate periodic behavior. Many of these rules result in a cell environment that stays relatively similar at all even or odd timesteps but changes more dramatically between even and odd timesteps. This graph indicates that the model cannot predict this behavior.

5 Conclusion

Our model was able to accurately predict simulations under the majority of rules. However, for more complex, harder-to-predict rules, the model failed, showing performance that was only somewhat better than random. On visual inspection, the model is able to replicate smaller patterns found in the ground truths. However, when it comes to replicating patterns in a larger scale, the model fails. This visual observation is reinforced by the 7x7 patch accuracy degrading from 3x3 and 5x5 patch accuracy for many rules. Additionally, the model's accuracy per timesteps fluctuates, showing a lack of ability to predict periodic simulation behavior. Seemingly, our model fails to recreate patterns over many time steps and often randomly diverges or freezes after few time steps. It replicates simple patterns, but fails to replicate these patterns at a larger scale.

Through our results, we find that our Transformer model is largely unable to learn to simulate the evolution of artificial life purely from raw patterns. As such, the dynamics of life-like systems cannot be fully learned and predicted by large language models, at least with out current setup.