Using Sparse Transformers as World Models to Improve Generalization

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

EMBEDDED SYSTEMS

by

Ariel Ebersberger

Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Using Sparse Transformers as World Models to Improve Generalization

Author: Ariel Ebersberger
Student id: 4874951

Abstract

This thesis introduces a novel sparsity-regularized transformer to be used as a world model in model-based reinforcement learning, specifically targeting environments with sparse interactions. Sparse-interactive environments are a class of environments where the state can be decomposed into meaningful components, and state transitions depend primarily on a small subset of state components. Traditional neural networks often struggle with generalization in such environments, as they consider all possible interactions between state components, leading to overfitting and poor sample efficiency. We formally define sparse-interactive environments and propose a simple yet effective modification to the standard transformer architecture that promotes sparsity in the attention mechanism through L1 regularization and thresholding. Through extensive experiments on the Minigrid environment, we demonstrate that our sparsity-regularized transformer achieves higher validation transition accuracy and lower variance across random initializations compared to the original transformer, particularly in low-data regimes. Our source code is available on GitHub[1].

Thesis Committee:

Chair:	Wendelin Böhmer, Sequential Decision Making, TU Delft
Committee Member:	Pradeep Murukannaiah, Interactive Intelligence, TU Delft
Daily Supervisor:	Caroline Horsch, Faculty EEMCS, TU Delft

[1] https://github.com/justanotherariel/SparseTransformerWorldModel

Preface

With this thesis, I conclude my Master's degree in Embedded Systems at Delft University of Technology. The past three years have been a period of significant personal and professional growth, during which I have had the opportunity to learn from and collaborate with many talented individuals. During this time, also found my passion for machine learning and joined Team Epoch, both being a pivotal experience in shaping my academic and career aspirations. I am also grateful for the support and guidance I have received from my professors, colleagues, and friends throughout this journey. A special thanks goes to my supervisor, Wendelin Böhmer, for his guidance and support during my thesis project, and Caroline Horsch, who patiently listened to my frustrations and provided encouragement during the inevitable setbacks of research. Finally, I would also like to express my gratitude to my family and friends, for their unwavering support and encouragement throughout my academic journey.

Ariel Ebersberger
Delft, the Netherlands
June 9, 2025

Contents

Chapter 1

Introduction

Reinforcement learning (RL) is hard. Unlike supervised learning, which benefits from labeled datasets and clear feedback signals, RL agents must learn through interaction with an environment, navigating the exploration-exploitation trade-off while relying on delayed and sparse rewards. This fundamental challenge has driven decades of research, from Richard Bellman's mathematical foundations in dynamic programming to modern deep reinforcement learning approaches that have achieved remarkable feats in games, robotics, and resource management.

Despite these advances, reinforcement learning still faces significant obstacles to widespread adoption in real-world applications. Sample efficiency remains a critical challenge, with many current algorithms requiring millions of environment interactions to learn effective policies. This becomes particularly problematic when interactions are costly, dangerous, or time-consuming, as is often the case in physical systems.

Model-based reinforcement learning offers a promising approach to addressing these efficiency concerns. By learning a model of the environment's dynamics, agents can perform planning and simulate experiences, potentially reducing the number of actual interactions needed. Model-based methods thus generally offer better sample efficiency compared to model-free approaches, making them particularly valuable when real interactions are limited or expensive (Sutton 1990; Kaiser et al. 2020). Another advantage is the ability to transfer learned knowledge to new tasks or environments, enabling agents to adapt quickly to new situations. However, model-based methods face their own challenges, including systematic prediction errors within the training distribution, compounding inaccuracies during multi-step rollouts, and degraded performance on out-of-distribution states (Hafner, Lillicrap, I. Fischer, et al. 2019; Hafner, Lillicrap, Ba, et al. 2020).

Sparse-Interactive Environments and their Relevance In this thesis, we introduce and formally define "sparse-interactive environments" - a class of environments where states are not monolithic and individual next-state components require only information from a small subset of the current-state components during a transition. This property creates a form of conditional independence in the transition function and many environments exhibit it. From board games, where moves affect only certain pieces, to real-world physical systems, where interactions are localized. While various forms of structure in environment dynamics have been studied, our specific formalization provides a framework for analyzing and improving generalization. We establish this concept in section 3.1.

Limitations of Traditional Architectures Traditional neural network architectures used for world modeling, such as recurrent neural networks (RNNs) or transformers, are not explicitly designed to leverage this sparse interactive structure within the state to next-state prediction. They tend to consider all possible interactions between state components to predict the

next-state, which can lead to overfitting and thus limited generalization to unseen states. The transformer architecture specifically, with its self-attention mechanism, has shown remarkable success across various domains (Vaswani et al. 2017). And while multiple variations of the original transformer have been proposed, none of them were specifically designed to exploit the sparsity in state transitions that we encounter in sparse-interactive environments.

Our Approach Our research introduces a sparsity-regularized transformer, or sparse transformer for short, specifically designed for modeling sparse-interactive environments. By encouraging the transformer to focus attention only on the fewest possible relevant state components, we aim to improve generalization, sample efficiency, and interpretability. The core innovation lies in a simple yet effective modification to the standard transformer that promotes sparsity in the attention mechanism through regularization and additional attention thresholding.

The central hypothesis of this thesis is that a sparse transformer can better generalize than a classical transformer on sparse-interactive environments, particularly in low-data regimes. By "better generalization," we specifically examine whether the sparse transformer can: (1) achieve higher validation accuracy with limited training data, (2) maintain consistent performance across different random initializations, and (3) perform robustly when the validation distribution differs from the training distribution.

1.1 Contribution

Our contributions include:

1. A formal definition and characterization of sparse-dependent and sparse-interactive environments in the context of reinforcement learning
2. A novel sparsity-regularized transformer, enabled by an added auxiliary loss designed to exploit the sparse interactive structure of these environments with additional attention thresholding
3. A systematic evaluation framework for measuring generalization in sparse-interactive environments
4. Empirical evidence demonstrating the advantages of sparse transformers over classical transformers and other architectures across various data regimes

1.2 Outline

The remainder of this thesis is organized as follows: chapter 2 provides the theoretical background on Markov decision processes, reinforcement learning fundamentals, the transformer architecture, and U-Net architecture. Chapter 3 details our methodology, including the formal definition of sparse-dependent and sparse-interactive environments, our environment selection criteria, and the evaluation framework for assessing generalization. Chapter 4 describes the implementation details of our three architectures: the classical transformer, sparse transformer, and U-Net, including how state/action pairs are processed within them. Chapter 5 presents our experimental results, comparing model performance across in-distribution and out-of-distribution scenarios and analyzing learned attention patterns. Chapter 6 reviews related work in sparse attention mechanisms and structured world models. Finally, chapter 7 concludes the thesis by summarizing our key findings and contributions and chapter 8 discusses promising directions for future research.

Chapter 2

Background

This chapter provides the theoretical foundation for understanding the research presented in this thesis. We begin by introducing Markov decision processes (MDPs) as the mathematical framework for sequential decision-making. We then explore RL fundamentals, including both model-free and model-based approaches and discuss generalization in RL. We continue with an overview of the transformer architecture, before introducing the U-Net architecture as an alternative approach with spatial inductive bias.

2.1 Markov Decision Processes

MDPs provide the mathematical foundation for modeling sequential decision-making problems under uncertainty (Bellman 1957; Puterman 2014). An MDP is formally defined as a tuple (S, A, P, R, γ) where:

- S is the state space, representing all possible configurations of the environment
- A is the action space, representing all possible actions an agent can take
- $P : S \times A \times S \to [0, 1]$ is the transition function, defining the probability $P(s'|s, a)$ of transitioning to state s' when taking action a in state s
- $R : S \times A \to \mathbb{R}$ is the reward function, specifying the immediate reward $R(s, a)$ received when taking action a in state s
- $\gamma \in [0, 1)$ is the discount factor, which determines the importance of future rewards relative to immediate ones

The defining characteristic of MDPs is the Markov property, which states that the future state depends only on the current state and action, not on the history of previous states and actions:

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ..., s_0, a_0) = P(s_{t+1}|s_t, a_t) \tag{2.1}$$

This property significantly simplifies the problem by allowing us to make decisions based solely on the current state without needing to maintain the entire history of interactions. Put into words, at each time step t, an agent within the given environment observes the current state s_t, selects an action a_t according to its policy $\pi : S \to \Delta(A)$ (which maps states to probability distributions over actions), and receives a reward $r_t = R(s_t, a_t)$ and the next state s_{t+1}. The goal in an MDP is to find an optimal policy π^* that maximizes the expected discounted cumulative reward:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right] \tag{2.2}$$

where $a_t \sim \pi(\cdot|s_t)$ and the expectation is taken over trajectories generated by following policy π.

The optimal policy can be derived from the optimal value function $V^*(s)$ or the optimal action-value function $Q^*(s, a)$, which satisfy the Bellman optimality equations:

$$V^*(s) = \max_a \left[R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^*(s') \right] \tag{2.3}$$

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^*(s') \tag{2.4}$$

The Bellman equations provide a recursive relationship that underlies many reinforcement learning algorithms (Sutton and Barto 2018).

In the context of our research, MDPs form the theoretical basis for the environments we study. The transition function P is of particular importance as it captures the dynamics of the environment and is what our world models aim to learn. In sparse-interactive environments, the structure of P has special properties where state transitions depend only on a subset of the state information, which our sparse transformer architecture is designed to exploit.

2.2 Reinforcement Learning Fundamentals

Reinforcement learning (RL) provides a computational approach to solving MDPs through trial-and-error interaction with an environment (Sutton and Barto 2018). Unlike supervised learning, RL does not rely on labeled training data but instead learns from the rewards and punishments received during exploration.

The RL framework consists of an agent interacting with an environment over a sequence of discrete time steps. The agent's objective is to learn a policy that maximizes the expected return, defined as the sum of discounted future rewards. RL algorithms can be broadly categorized into model-free and model-based approaches, with further distinctions within each category.

2.2.1 Model-Free Reinforcement Learning

Model-free RL algorithms learn directly from experience without explicitly modeling the environment dynamics. These approaches can be further classified into three main categories:

1. **Value-based methods** learn a value function that estimates the expected return from each state or state-action pair. Examples include Q-learning (Watkins 1989) and Deep Q-Networks (DQN) (Mnih, Kavukcuoglu, et al. 2013). These methods typically update value estimates using temporal difference learning, which adjusts the current value estimate based on the observed reward and the estimated value of the next state.

2. **Policy-based methods** directly optimize the policy without explicitly maintaining a value function. The REINFORCE algorithm (Williams 1992) is a classic example that utilizes policy gradients to adjust the policy parameters in the direction that increases the expected return. This approach updates the policy parameters based on the collected trajectory information and the corresponding returns.

3. **Actor-critic methods** combine aspects of both approaches by learning both a policy (actor) and a value function (critic) simultaneously, which can reduce variance in policy gradient estimates. Examples include Asynchronous Advantage Actor-Critic (A3C) (Mnih, Badia, et al. 2016) and proximal policy optimization (PPO) (Schulman et al. 2017) algorithms.

Model-free methods are generally simpler to implement and can be effective in many environments. However, they often require large amounts of experience to learn effective policies, as they must infer environmental dynamics implicitly through repeated interaction.

2.2.2 Model-Based Reinforcement Learning

Model-based RL extends traditional RL approaches by incorporating an explicit model of the environment. This model typically consists of two components, which can also be combined into a single world model:

1. A **transition model** $\hat{P}(s_{t+1}|s_t, a_t)$ that predicts the next state given the current state and action

2. A **reward model** $\hat{R}(s_t, a_t, s_{t+1})$ that predicts the immediate reward

These models can be learned from data collected through agent-environment interactions and then be used to:

1. **Generate simulated experience** for offline training, where predicted transitions and rewards augment the dataset used to update the agent's policy or value function (Sutton 1990). The world model can be trained while also training the agent, such that new relevant data is generated and used to improve the model. It is also possible to **perform targeted exploration** in regions with high uncertainty to learn in environments with very sparse rewards (Pathak et al. 2017).

2. **Plan ahead** at decision time by performing look-ahead search to evaluate potential action sequences and their expected outcomes before committing to an action (Silver et al. 2016).

Model-based methods generally offer better sample efficiency compared to model-free approaches, making them particularly valuable in environments where interactions are costly or limited (Sutton 1990; Kaiser et al. 2020). However, they face several challenges:

1. **Model bias**: Inaccuracies in the learned model can lead to suboptimal policies (Kaiser et al. 2020)

2. **Compounding errors**: Small prediction errors can accumulate when the model is used for multi-step planning (Hafner, Lillicrap, I. Fischer, et al. 2019; Hafner, Lillicrap, Ba, et al. 2020)

3. **Distribution shift**: The model may perform poorly on states that differ from those encountered during training. This heavily depends on the model's ability to generalize to unseen states, which is a key focus of our research.

2.2.3 Generalization in Reinforcement Learning

Generalization in reinforcement learning refers to an agent's ability to perform well in situations that differ from those encountered during training. It can be categorized into two main types:

1. **In-Distribution (ID) Generalization**: The ability to perform well on unseen states drawn from the same distribution as the training data.

2. **Out-of-Distribution (OOD) Generalization**: The ability to perform well on states drawn from a different distribution than the training data, which can involve changes in environmental dynamics, visual appearance, or task structure.

Several factors influence generalization in RL:

1. **State Representation**: The way states are encoded can significantly impact generalization. Good representations capture relevant features while ignoring irrelevant details (Bengio, Courville, and Vincent 2013).

2. **Model Architecture**: Different neural network architectures embed different inductive biases that affect generalization. For example, CNNs encode spatial locality, while transformers excel at capturing long-range dependencies (Battaglia et al. 2018). Regularization techniques like dropout and weight decay can also help prevent overfitting (Srivastava et al. 2014).

3. **RL Algorithm**: The choice of RL algorithm can also influence generalization. Some algorithms, like DQN, may overfit to specific training environments, while others, like A3C, may be more robust (Mnih, Badia, et al. 2016). Model-based RL approaches can also improve generalization by explicitly modeling the environment dynamics (Kaiser et al. 2020).

4. **Environment Diversity**: Training across a diverse set of environments can help the agent learn more robust policies (Tobin et al. 2017).

Our research addresses these generalization challenges in model-based RL by examining how architectural innovations—specifically sparse attention mechanisms in transformers—can improve world model generalization in sparse-interactive environments. We propose that these architectural choices can better capture transition dynamics, reducing model bias and improving performance on out-of-distribution states, ultimately leading to improved sample efficiency and more robust planning strategies. We evaluate this hypothesis using controlled variations of the Minigrid environment, testing both in-distribution and out-of-distribution generalization scenarios as detailed in our methodology.

2.3 Transformer Architecture

Transformers, a neural network architecture initially introduced by Vaswani et al. (2017) in their seminal paper "Attention is All You Need", have revolutionized sequence modeling and natural language processing through their self-attention mechanism. Unlike recurrent neural networks, transformers process entire sequences in parallel, enabling them to capture long-range dependencies more efficiently and effectively. First, we will provide a brief overview of the transformer architecture's key components. Then, we will introduce the non-autoregressive variant of transformers, which is particularly relevant to our world modeling task.

2.3.1 Transformer Components

The transformer architecture is typically made up of a few key components, which we will outline below.

Self-Attention and Multi-Head Attention (MHA) The core of the transformer architecture is the scaled dot-product attention mechanism:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.5}$$

where Q (queries), K (keys), and V (values) are linear projections of the input, and d_k is the dimension of the keys. This fundamental operation computes a weighted sum of values, where each weight is determined by the compatibility between the corresponding query and key. It allows the model to pass information from one token to another based on their relevance, effectively enabling the model to focus on different parts of the input sequence when generating each output token. In self-attention, the same input serves as queries, keys, and values.

Multi-head attention further extends this mechanism by allowing the model to learn multiple different attention patterns in the input in one layer. The MHA mechanism splits the input into multiple heads, each with its own set of learned projections, and computes attention independently for each head. The results are then concatenated and projected back to the original dimension:

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2.6}$$

Where W_i^Q, W_i^K, and W_i^V are learned projection matrices for each head, and W^O is the output projection matrix. This allows the model to capture different aspects of the input sequence simultaneously, enhancing its representational power.

Feed Forward Networks In addition to the attention mechanism, each transformer layer contains a position-wise feed-forward network (FFN). This component consists of two linear transformations with a non-linear activation function in between:

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2 \tag{2.7}$$

The FFN typically expands the dimensionality by a factor of 4 in the hidden layer before projecting back to the model dimension. This expansion allows the model to learn complex non-linear transformations and has been shown to be crucial for the transformer's expressive power. Recent work has explored various activation functions beyond ReLU, such as GELU and SwiGLU, which can further improve performance (Shazeer 2020).

Layer Normalization and Residual Connections To stabilize training and improve convergence, transformers use layer normalization and residual connections (Vaswani et al. 2017) after MHA and FFN sub-layers. Layer normalization normalizes the inputs to each sub-layer, ensuring that the mean and variance remain consistent across different layers. Residual connections allow gradients to flow more easily through the network by adding the input of each sub-layer to its output:

$$\text{LayerNorm}(x + \text{Sublayer}(x)) \tag{2.8}$$

This combination helps mitigate the vanishing gradient problem and allows for deeper networks without suffering from degradation in performance.

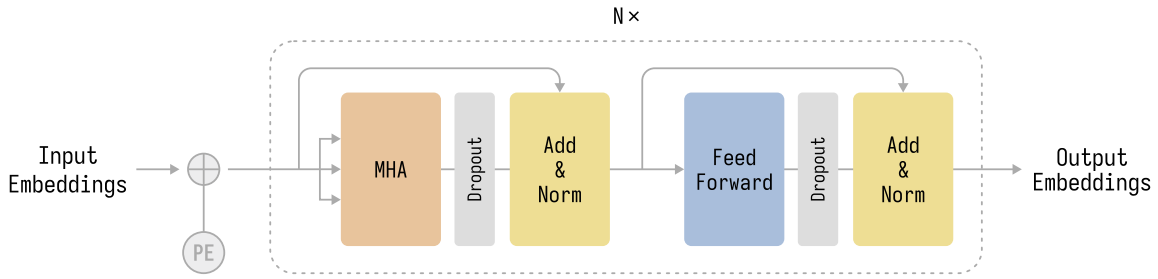


Figure 2.1: Basic non-autoregressive transformer architecture. The model only consists of a decoder stage, which predicts all output tokens in parallel. The input is processed through a series of multi-head self-attention layers, followed by feed-forward networks and layer normalization. "PE" is short for positional encoding.

Positional Encoding Since the self-attention mechanism is permutation-invariant, transformers require explicit positional information to understand the sequential nature of the input. The original work by Vaswani et al. (2017) introduced sinusoidal positional encodings. However, subsequent research has shown that learned positional embeddings often perform equally well or better in practice (Devlin et al. 2019). Many modern transformer variants now use learned positional embeddings as trainable parameters.

2.3.2 Non-Autoregressive Transformers (NATs)

For our world modeling task, we employ a non-autoregressive variant, which differs significantly from the original autoregressive design presented in Vaswani et al. (2017). Instead of predicting each output token sequentially, NATs generate all output tokens in parallel during inference. In addition to removing the final linear projection layer, which is typically used to map the output to a vocabulary in language tasks, NATs also do not use causal masking. This allows all positions in the output sequence to attend to each other simultaneously, enabling the model to capture dependencies across the entire sequence without the constraints of sequential generation.The architecture retains the same core components as the original transformer, including multi-head self-attention, feed-forward networks, and layer normalization, but operates in a non-autoregressive manner. This design choice significantly speeds up inference, making NATs particularly suitable for tasks where rapid generation of outputs is crucial, such as in real-time applications or large-scale sequence modeling tasks. Gu et al. (2018) used NATs for machine translation, where they demonstrated that the model could generate translations in parallel, significantly reducing inference time compared to autoregressive models.

Limitations However, NATs face inherent challenges, for example, capturing complex dependencies within the input sequence, as they do not model the sequential nature of the data explicitly. It also requires a separate mechanism to calculate the number of output tokens, as there is no autoregressive generation step that would naturally determine the length of the output sequence. Gu et al. (2018) addressed this by using 'fertilities'—integers for each input token that indicate how many output tokens it should generate. Lee, Mansimov, and Cho (2018) instead used a separate length prediction network. Token repetition is another challenge in generative tasks, where the model may generate the same token multiple times in a row. This can be mitigated by using post-processing techniques, such as deduplication (Lee, Mansimov, and Cho 2018).

Our Approach In our research, we adopt a basic version of a NAT architecture, as shown in figure 2.1. The model only consists of a decoder stage, with no linear projection layer at

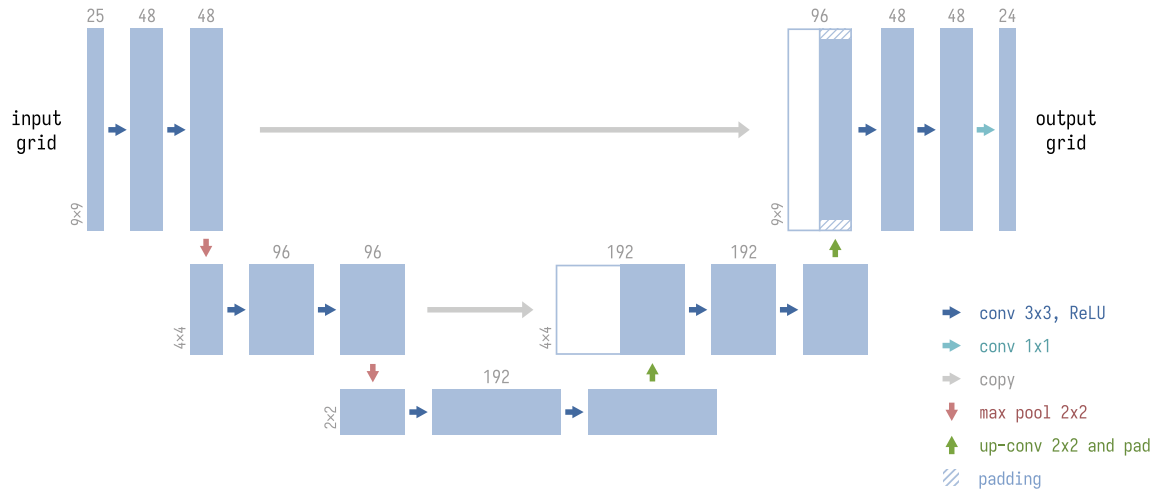


Figure 2.2: U-Net architecture adapted to Minigrid 9x9 environments. The contracting path captures context, while the expanding path enables precise localization through skip connections. Similar to Ronneberger, P. Fischer, and Brox (2015)

the end, which would typically be used to map the output to a vocabulary in language tasks. Instead, the output of the decoder is directly used as the next state prediction in our world modeling task. This allows us to circumvent the challenges of token repetition and length prediction, though this is only possible because the next state is always of the same length as the current state.

2.4 U-Net Architecture

The U-Net architecture, originally introduced by Ronneberger, P. Fischer, and Brox (2015) for biomedical image segmentation, has become a standard approach for tasks requiring precise spatial localization. Its distinctive U-shaped structure consists of a contracting path (encoder) followed by an expanding path (decoder) with skip connections between corresponding levels, as illustrated in figure 2.2. Due to the convolutional nature of the architecture, it is particularly well-suited for tasks involving spatial data, such as images or videos. This bias made it a natural choice for our world modeling task, as our chosen environment, Minigrid (Chevalier-Boisvert et al. 2023), consists of 2D grid-based states. The rest of this section explains the encoder, decoder, and skip connections in more detail.

Contracting Path (Encoder) The contracting path follows the typical architecture of a convolutional network, with repeated application of convolutions, activation functions, and max pooling operations for downsampling. This progressively reduces the spatial dimensions while increasing the number of feature channels, capturing higher-level abstractions of the input.

Each encoder block in the original U-Net consists of two convolution-ReLU operations followed by max pooling:

$$\begin{aligned} h_l^{(1)} &= \text{ReLU}(\text{Conv}_{3\times3}(h_{l-1}, W_l^{(1)}) + b_l^{(1)}) \\ h_l^{(2)} &= \text{ReLU}(\text{Conv}_{3\times3}(h_l^{(1)}, W_l^{(2)}) + b_l^{(2)}) \\ h_l &= \text{MaxPool}_{2\times2}(h_l^{(2)}) \end{aligned} \tag{2.9}$$

where:

- $h_l \in \mathbb{R}^{C_l \times H_l \times W_l}$ represents the feature map at layer l, with C_l channels, height H_l, and width W_l
- $W_l^{(i)} \in \mathbb{R}^{C_l \times C_{l-1} \times 3 \times 3}$ represents the learnable convolutional weights for the i-th convolution
- $b_l^{(i)} \in \mathbb{R}^{C_l}$ is the bias term for each output channel
- $\text{Conv}_{k\times k}(\cdot, W_l)$ denotes a 2D convolution operation with kernel size $k \times k$ (typically 3×3 in U-Net)
- $\text{MaxPool}_{2\times 2}(\cdot)$ is the max pooling operation with stride 2, reducing spatial dimensions by half

Expanding Path (Decoder) The expanding path combines feature information with spatial information through upsampling operations followed by convolutions. The skip connections from the contracting path provide the precise spatial information lost during downsampling, allowing the network to generate high-resolution outputs.

Each decoder block consists of an upsampling step followed by two convolution-ReLU operations:

$$\begin{aligned} h_l^{(up)} &= \text{Up}(h_{l-1}) \\ h_l^{(concat)} &= \text{Concat}(h_l^{(up)}, h_{l'}) \\ h_l^{(1)} &= \text{ReLU}(\text{Conv}_{3\times 3}(h_l^{(concat)}, W_l^{(1)}) + b_l^{(1)}) \\ h_l &= \text{ReLU}(\text{Conv}_{3\times 3}(h_l^{(1)}, W_l^{(2)}) + b_l^{(2)}) \end{aligned} \tag{2.10}$$

where:

- $h_{l-1} \in \mathbb{R}^{C_{l-1} \times H_{l-1} \times W_{l-1}}$ is the feature map from the previous decoder layer
- $\text{Up}(\cdot)$ is the upsampling operation, which can be either:
  - Transposed convolution (deconvolution): $\text{Up}(h) = \text{ConvTranspose}_{2\times 2}(h, W_{up})$ with learnable weights W_{up}
  - Bilinear interpolation followed by a 1×1 convolution for channel adjustment
- $h_{l'} \in \mathbb{R}^{C_{l'} \times H_{l'} \times W_{l'}}$ is the corresponding feature map from the contracting path at the same resolution level
- $\text{Concat}(\cdot, \cdot)$ represents concatenation along the channel dimension, resulting in a tensor with $C_{up} + C_{l'}$ channels, where C_{up} is the number of channels after upsampling
- $W_l^{(i)} \in \mathbb{R}^{C_l \times C_{input} \times 3 \times 3}$ are the convolutional weights, where $C_{input} = C_{up} + C_{l'}$ for the first convolution and $C_{input} = C_l$ for the second
- $b_l^{(i)} \in \mathbb{R}^{C_l}$ is the bias term

Skip Connections The skip connections are a crucial component of U-Net, preserving fine-grained spatial information that would otherwise be lost during the downsampling process. For each decoder layer at resolution level l, the corresponding encoder feature map $h_{l'}$ from the same resolution is concatenated with the upsampled features. This allows the network to combine:

- High-level semantic information from the deeper layers (through h_{l-1})
- Fine-grained spatial details from the earlier layers (through $h_{l'}$)

Our Approach In our research, we adapt the U-Net architecture to function as a world model for predicting state transitions in reinforcement learning environments. While typically, a linear projection layer would be used at the end of the decoder to map the output to the desired number of output classes, we instead directly use the output of the decoder as the next state prediction in our world modeling task. This provides a second baseline in addition to the transformer with an inherent spatial inductive bias, against which we can compare our proposed sparse transformer approach.

Chapter 3

Methodology

This chapter presents our approach to testing whether a sparse transformer can generalize better than a classical transformer in sparse-interactive environments. By improved generalization, we specifically examine if the sparse transformer can (1) achieve higher validation accuracy with limited training data, (2) maintain consistent performance across different random initializations, and (3) perform robustly when the validation distribution differs from the training distribution. We begin by formally defining sparse-dependent and sparse-interactive environments, establishing the mathematical framework for understanding how next-state components depend on limited subsets of information. We then introduce Minigrid, our selected environment that exhibits this property. Finally, we outline our evaluation framework, including sampling methodology and performance metrics.

3.1 Sparse-Dependent and Sparse-Interactive Environments

The state returned by RL environments is often a high-dimensional vector, and is conventionally treated as a monolithic entity by neural networks. However, in many environments, the state can actually be decomposed into smaller components that are meaningful to the task, which in turn often exhibit useful properties. These components can be thought of as elements of the state vector, which can be individual cells in a grid or pieces on a board.

Sparse-dependent and sparse-interactive environments represent a class of MDPs where individual next-state components only depend on a small subset of the available state information. This, in turn, creates a form of conditional independence in the transition function which can be exploited by the model to improve generalization and sample efficiency. We define these environments formally as follows:

Sparse-dependent environments Formally, for a state s with components $s = (s^1, s^2, ..., s^n)$, we define the dependency set for component i as:

$$D(i) \subseteq \{1, 2, ..., n\} \quad \text{where} \quad |D(i)| \ll n \tag{3.1}$$

The transition probability for component s^i_{t+1} given action a_t depends only on the components indexed by $D(i)$:

$$P(s^i_{t+1}|s_t, a_t) = P(s^i_{t+1}|s_t^{D(i)}, a_t) \tag{3.2}$$

where $s_t^{D(i)} = \{s_t^j : j \in D(i)\}$ denotes the subset of state components indexed by $D(i)$. An example of this is depicted in figure 3.2a, where each state component s^i_{t+1} has dependencies that are spatially localized and can be determined a priori from the component's position.

Sparse-interactive environments are a more general class of environments where the dependency structure itself also depends on the current state. In these environments, both which components are relevant and how many are needed can vary dynamically. Formally, we define the state-conditional dependency set as:

$$D(i, s_t) \subseteq \{1, 2, ..., n\} \quad \text{where} \quad |D(i, s_t)| \ll n \tag{3.3}$$

The transition probability becomes:

$$P(s^i_{t+1}|s_t, a_t) = P(s^i_{t+1}|s_t^{D(i,s_t)}, a_t) \tag{3.4}$$

While the dependency structure is no longer fixed, the key property of information sparsity remains: $|D(i, s_t)| \ll n$ for all states s_t. This ensures that only a small fraction of the available information is required for any given next-state component prediction, though the specific information needed may vary.

Examples of sparse-interactive environments appear in many real-world and artificial environments:

1. **Grid-based environments** like Minigrid (Chevalier-Boisvert et al. 2023), where the outcome of an action like "move forward" depends primarily on the agent's position and orientation and the content of the adjacent cell, not on the entire grid state, as illustrated in figure 3.2. This also includes games like chess and go, where the outcome of a move depends only on the positions of relevant pieces.

2. **Cellular automata** like Conway's Game of Life (Gardner 1970), where the next state of a cell depends only on its neighbors, not on the entire grid. This also includes physical systems with localized interactions, such as fluid dynamics or mechanical systems, where the behavior of a component is influenced primarily by its immediate surroundings.

3. **Traffic Simulation** environments, where the behavior of a vehicle depends primarily on its immediate surroundings (e.g., nearby vehicles, traffic lights) rather than the entire road network. This also includes robotics applications, where the state of a robot may depend primarily on its immediate environment (e.g., obstacles, landmarks) rather than the entire scene.

Both sparse-interaction and sparse-dependency can be viewed as a form of structural prior knowledge about the environment dynamics. When properly leveraged, this structure can dramatically simplify the learning problem, as the model knows that it only needs to focus on a small subset of state components to make accurate predictions. For the rest of this thesis, we will focus on sparse-interactive environments, as the solution we propose is designed to handle the more general case of sparse-interactive environments, which also include sparse-dependent environments.

Impact Traditional neural network architectures like fully connected networks or classical dense transformers are not explicitly designed to leverage this sparse-interactive structure. They tend to consider all possible interactions between state components, which can lead to:

- **Overfitting** to spurious correlations in the training data
- **Poor sample efficiency** as the model must learn to ignore irrelevant interactions
- **Limited generalization** to unseen states that differ in irrelevant components

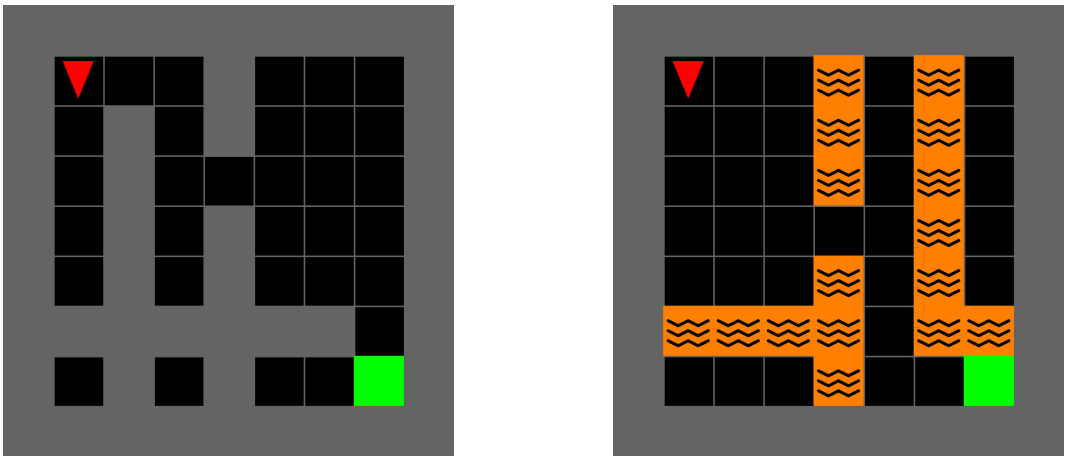
Our research addresses these limitations by introducing a sparse transformer architecture specifically designed to model sparse-interactive environments. By encouraging the transformer to focus attention only on relevant state components, we aim to improve generalization, sample efficiency, and interpretability.

3.2 Environment Selection

To test our hypothesis that a sparse transformer can better generalize on sparse-interactive environments compared to a classical transformer, we needed an environment that fits this criterion. We selected Minigrid (Chevalier-Boisvert et al. 2023), as it is simple to visualize, small in data volume when sampled, and has been extensively used in RL research.

3.2.1 Minigrid

Minigrid is a 2D grid environment where the agent appears as a triangle-like red figure that must navigate toward a green goal field in the most direct way to maximize reward. The environment includes walkable tiles, walls, and lava, and can be extended with objects such as chests, keys, and doors allowing for flexibility in maze creation. Examples of two Minigrid environments are displayed in figure 3.1. This 2D grid structure naturally exhibits spatial locality, making it an ideal testing ground for comparing architectures with different inductive biases.

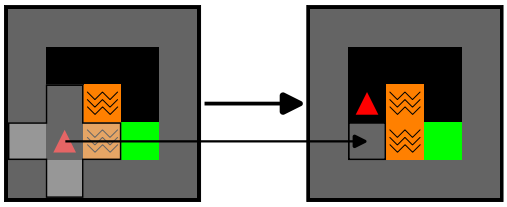


(a) `MiniGrid-SimpleCrossingS9N3-v0` (b) `MiniGrid-LavaCrossingS9N3-v0`

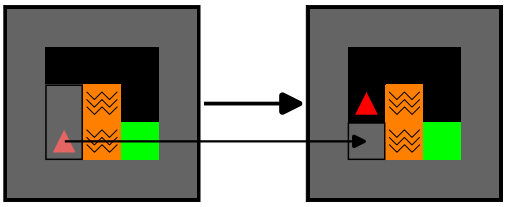
Figure 3.1: Example of two Minigrid Crossing environments. The agent (red triangle) navigates through a 9×9 grid to reach the green goal square while avoiding wall and lava obstacles. The environments presented contain 3 randomly placed wall/lava segments that create a maze-like structure. The agent can perform three actions: rotate left, rotate right, and move forward.

Crucially, Minigrid demonstrates the sparse-interactive property we aim to study, as demonstrated in figure 3.2. A single next-state component in Minigrid depends primarily on the components' immediate surroundings rather than the entire grid state. For instance, when an agent attempts to move forward, the outcome of the component with the agent inside depends only on whether the adjacent cell in that direction is a wall or empty space, as well as the component itself—not on the configuration of cells across the grid. This localized dependency structure means that an ideal model would learn to attend only to (or receive information only from) relevant components of the state space when predicting transitions.

Additionally, Minigrid provides a convenient framework for creating controlled experimental variations by adjusting parameters such as grid size, wall density, object placement,



(a) Sparse-dependent: A single component of the next state depends only, and always, on 5 components of the current state. The amount of input components required to predict a specific output component remains constant, regardless of the current state. If we were to make an assumption that the outside walls are always present, the number of components required can be reduced for the outer cells.



(b) Sparse-interactive: A single component of the next state depends on a variable number of components of the current state, depending on the current state. The amount of input components required to predict a single output component can vary, depending on the current state.

Figure 3.2: Example of a transition in Minigrid with action=`forward`. The left grid of each image shows the current state, while the right grid displays the next state after executing the action. These images illustrate why Minigrid is both sparse-dependent and sparse-interactive, as well as how these properties differ from each other.

and goal location. This flexibility allows us to systematically test generalization across increasingly difficult OOD scenarios. The environment's discrete state space also makes it straightforward to define precise accuracy metrics for our predictions.

3.2.2 Minigrid Specification

In Minigrid, the state is represented as a 2D grid where each cell contains a tuple of four elements: [object, color, state, agent]:

- **object**: An index representing the type of object in the cell, such as 'wall', 'lava', or 'empty'
- **color**: The color of the object (e.g., 'red' for lava, 'grey' for walls). Colors can also serve functional purposes, such as matching keys with corresponding doors
- **state**: The state of the object, which is only relevant for interactive objects like chests and doors
- **agent**: A two-part representation indicating (1) the presence of an agent in the cell and (2) the orientation of the agent. To better illustrate, these are the values this field can have: `no_agent`, `agent_east`, `agent_south`, `agent_west`, `agent_north`

The action space in Minigrid is discrete, consisting of seven possible actions: `left`, `right`, `forward`, `pickup`, `drop`, `toggle`, and `done`. For our experiments, we focused on `Crossing` environments, more specifically on the `MiniGrid-SimpleCrossingS9N3-v0` environment, shown in figure 3.1a. These specific crossing environments create sparse maze-like structures where the agent must navigate from one side to the other while avoiding walls. Due to the simplified nature of `MiniGrid-SimpleCrossingS9N3-v0`, we restricted the action space to just three main actions: `left` (rotate left), `right` (rotate right), and `forward` (move one field forward in the direction of the agent). The reward in these environments is continuous, and the next state maintains the same dimensionality as the current state, ensuring consistency in our world model predictions.

For our experiments, we assumed full observability of the environment, as this makes the problem an MDP rather than a partially observable Markov decision process (POMDP), which is easier to handle and analyze. This means that the agent can see the entire grid, including cells behind walls and in areas behind the agent itself, which allows the model to predict the next state based on the current state and action without needing to maintain a memory of previous states. We chose this approach to avoid the complexity of memory mechanisms that would be required for handling partial observability, as maintaining the information of previously observed states would introduce architectural components not directly relevant to our central hypothesis about sparse dependencies.

3.3 Assesing Generalization

To properly evaluate our models and assess generalization, we designed specific training and validation datasets to measure performance on both ID and OOD scenarios. This approach helps determine whether the sparse transformer is superior to other architectures when environments exhibit sparse dependencies.

When sampling a Minigrid environment, it is important to note that not all implementations generate unique environments upon reset. For our methodology, we assume the chosen Minigrid environment can produce multiple unique grids, like `MiniGrid-SimpleCrossingS9N3-v0`, which generates a random grid each time it is reset.

3.3.1 Train/Test Split

In this subsection, we define in- and out-of-distribution contexts for Minigrid environments, which we will refer to when discussing accuracy and generalization.

For clarity, "sampling a grid exhaustively" means placing the agent in all possible locations and orientations and executing all possible actions, while recording the current state, next state, and reward.

In-Distribution (ID) For ID evaluation, we sample n grids exhaustively and split the samples into training and validation sets, thus training and validating on the same grids, but different samples. This simulates a scenario where the model has seen the agent in a few locations in a few unique grids, and now has to extrapolate agent-movement knowledge to these same grids. It is ID because the training and validation sets are sampled from the same distribution.

Out-of-Distribution (OOD) Using this approach, we sample n grids for training and m grids for validation, both exhaustively. We then discard a percentage of the training samples randomly. This simulates a scenario where the model was able to see the agent in a couple of locations in a few unique grids, but now has to extrapolate agent-movement knowledge to new unseen grids, as well as the general behavior of unaffected cells.

3.3.2 Evaluation Metrics

After training the models on the training set, we evaluate them on both the training and validation datasets. Evaluation of the training set allows us to assess the model's ability to learn the samples it has seen, while evaluation of the validation set allows us to assess the model's ability to generalize to unseen data. We will first present environment-agnostic metrics, which give a broad overview of the model's performance, and then present more granular metrics specific to Minigrid environments, which give an indication of where the model is struggling.

Environment-agnostic Metrics State, reward, and transition accuracies are the primary metrics used to evaluate the model's performance. State accuracy measures the percentage of samples where the predicted next state matches the actual next state. Reward accuracy measures the percentage of samples where the predicted reward matches the actual reward. Transition accuracy measures the percentage of state/action pairs that are correctly predicted by the model, meaning that both the next state and reward are predicted correctly.

Minigrid-specific Metrics To gain more insight into the model's performance, we also evaluate it on more granular metrics that give an indication of where the model is struggling. Because each state component in Minigrid represents a cell in the grid, we can evaluate the model's performance on each cell variable separately. This allows us to identify four more metrics: object, color, state, and agent accuracies. Object accuracy measures the percentage of total samples where the object variable of all cells was predicted correctly. The same applies to color, state, and agent accuracies. These metrics help us understand how well the model is able to predict the different components of the state.

Additionally, we can evaluate the model by tracking the agent's behavior in the environment. The most important metric here is 'one agent accuracy', which measures the percentage of samples where only one agent was predicted, regardless of whether the agent was in the correct position or not. This metric is important because the prediction is useless otherwise in the context of model-based RL. Next, we can further break down the agent accuracy by considering the action the agent was required to take. We can measure the percentage of correctly predicted samples where the agent was required to rotate, move, or stay in the same position due to a wall blocking the movement. These metrics help us understand how well the model is able to predict the agent's behavior in the environment.

η Metrics Finally, we can add additional metrics for the sparse transformer to measure the impact of the sparsity regularization. As illustrated in figure 3.2b, we would like to encourage the model to only attend to the relevant components of the state when predicting the next state. To measure this, we track the distribution of attention connections in the η attention maps by counting how many tokens attend to each other across all validation samples. For our 9×9 Minigrid environment with 82 total tokens (81 grid cells + 1 reward token), the ideal sparsity would result in exactly 82 connections for rotation actions (tokens attending only to themselves) and 84 connections for forward actions (self-attention plus bidirectional information exchange between agent and forward cells). We can furthermore group the samples into two categories: rotation and forward actions, and measure the average number of connections for each category. This allows us to see how well the model is able to learn the sparse dependencies in the environment.

Chapter 4

Implementation

In this chapter, we will present the implementation details of our three different architectures: the classical transformer, sparse transformer, and U-Net. We first describe the discretization of the Minigrid state, which is necessary for all models, after which we will go into detail about the architecture and training setup for each model. The classical transformer and sparse transformer share the same hyperparameters and training setup, while the U-Net has its own set of hyperparameters. The hyperparameters used for each model are summarized in table A.1 and table A.2.

4.1 Discretizing the Minigrid State

To be able to use the sampled environment data for training our neural networks, we need to discretize and one-hot encode the state space. The Minigrid environment returns a 2D grid as its state, with `grid_height`×`grid_width` cells, where each cell specifies the object (e.g., wall, door, key), the color (e.g., red, green, blue), the state (e.g. open, close), and the agent (e.g. no-agent, agent looking up/down/...). After discretization, one-hot encoding, and concatenation of all variables, each cell can be represented by a vector with 24 dimensions. The discretization process is illustrated in figure 4.1.

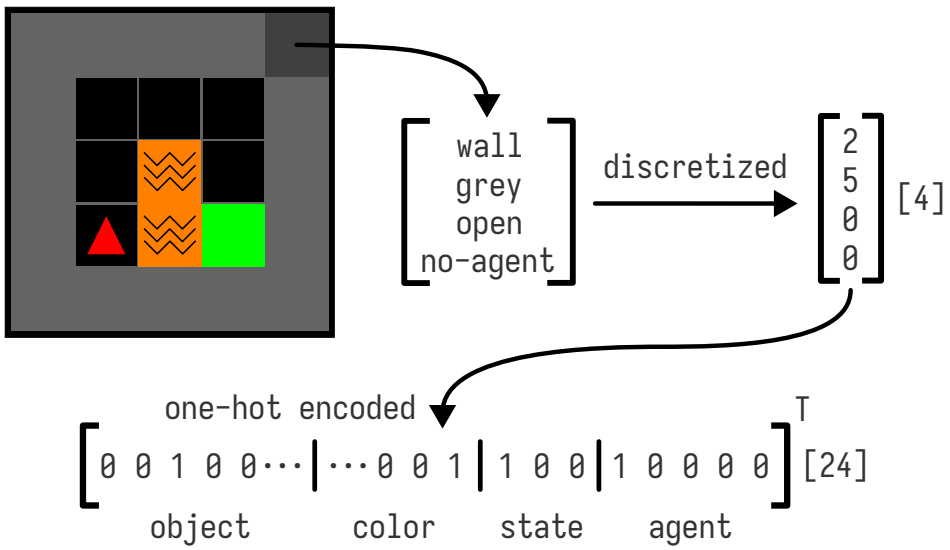


Figure 4.1: Discretization of Minigrid state. Each cell is represented by a vector with 24 dimensions.

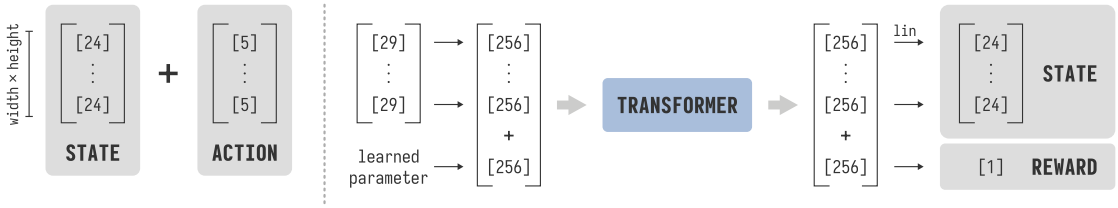


Figure 4.2: Classical transformer architecture. The model reshapes the observation and action into an input sequence of tokens. Afterward, they are processed through multiple layers of self-attention and feedforward networks and reshaped into the next-state and reward.

4.2 Classical Transformer

This section will describe how the classical transformer architecture is implemented and trained. The transformer is a non-autoregressive transformer (NAT), meaning it returns a sequence of tokens directly instead of one token at a time. We implemented a basic decoder-only transformer architecture, which has already been presented in section 2.3, where figure 4.2 further illustrates how the Mingrid state/action pair is fed through the transformer. The transformer is trained to predict the next-state and reward from the current-state and action. The upcoming paragraphs describe how the state/action pair is transformed into a sequence of tokens, and how the output tokens are reshaped into the next-state and reward.

The Hyperparameters used for the transformer are shown in table A.1. The hyperparameters were chosen based on hyperparameter optimization sweeps, which we conducted to find the best-performing configuration. The hyperparameters were kept the same for the sparse transformer, which we will describe in the next subsection.

State/Action to Token Embedding After discretization of the Minigrid state, as explained in section 4.1, the state is flattened into a sequence of tokens, where each token corresponds to a cell in the grid. Each token has a dimension of 24 (for the object, color, state, and agent) and is concatenated with the action (which can take on the state of rotate left, rotate right, forward), resulting in a token dimension of 27 after one-hot encoding.

These tokens are then projected to d_{model} dimensions using a linear layer. Next, a learnable "reward token" of size d_{model} is added at the last position, totaling the sequence length to grid_height $\times$ grid_width $+ 1$ tokens. The reward token is initialized randomly and will be used to predict the reward of the next state, and allows the transformer to pass information to it. Finally, the positional embeddings are added to each token embedding, after which it is passed through the transformer layers.

Non-Autoregressive Transformer (NAT) The resulting sequence was processed by the NAT, which feeds the input through three identical decoder layers. We furthermore employed a dropout rate of 0.15 after both the MHA and feedforward network.

Token Embedding to Next-State/Reward The output of the transformer is a sequence of tokens of size grid_height $\times$ grid_width $+ 1$, where the majority of tokens correspond to the next-state and the last token corresponds to the reward. The next-state tokens are reshaped into a grid of size `grid_height` $\times$ `grid_width`, which is then passed through a linear layer to reduce the dimensionality to 24. This results in a grid of size `grid_height` $\times$ `grid_width` $\times 24$. The reward token is passed through a linear layer to reduce the dimensionality to a scalar value, which represents the predicted reward for the next state.

Loss Function The loss function used for training the classical transformer was a combination of two components: a cross-entropy loss for the next-state prediction and a mean squared error (MSE) loss for the reward prediction. The cross-entropy loss used is actually an adaptation called focal loss, introduced by Lin et al. (2017)—a variant that introduces a modulating factor to focus more on hard-to-classify examples. This is particularly effective for imbalanced datasets and has shown during our hyperparameter optimization sweeps to outperform classical cross-entropy (CE) loss. To apply focal loss, we first split the next-state prediction into its four variables (object, color, state, and agent) and applied the loss function to each variable separately. To predict the reward, we made use of the MSE loss, as it is a continuous value. The final loss was a weighted combination of the two losses, with weights of 0.8 for the next-state loss and 0.2 for the reward loss, again chosen based on the results of the hyperparameter optimization sweeps.

The focal loss implementation can be expressed mathematically as follows:

$$\text{FocalLoss}(predictions, targets) = \frac{1}{N}\sum_{i=1}^{N}(1-p_t)^{\gamma}\cdot \text{CE}_i \tag{4.1}$$

where CE_i is the cross-entropy loss for the i-th example, p_t is the predicted probability for the correct class, $\gamma = 2.0$ is the focusing parameter, and N is the number of examples.

4.3 Sparse Transformer

The sparse transformer is a modified version of the classical transformer, designed to exploit the sparse structure of the state transitions in sparse-interactive environments. It was kept as similar as possible to the classical transformer to ensure a fair comparison while introducing modifications that promote sparsity in the attention mechanism. The only differences are the addition of an auxiliary loss that encourages sparsity in the attention maps and the introduction of a thresholding mechanism to ensure that the information exchange is sparse between tokens. As the sparse transformer is mostly identical to the classical transformer, we will not repeat the details of the architecture here but rather focus on the differences and additional components for the rest of this section.

The sparse transformer uses the same hyperparameters and training setup as the classical transformer with three additional sparsity-related parameters (A_{sum} loss function, A_{sum} loss weight and attention threshold), as shown in table A.1. We did perform hyperparameter sweeps to find the best-performing configuration for the sparse transformer specifically but found that the addition of the auxiliary loss and thresholding did not change the optimal hyperparameter values.

L1 Attention Loss To be able to encourage sparsity, we need to modify the classical transformer to return the attention maps from the MHA module. This is done by modifying the MHA module to return the attention weight matrix $A_{l,h} = \text{Attention}(Q_l, K_l, V_l)$ for each layer l and head h. This matrix has a size of `grid_width`×`grid_width` and contains the attention weight for each token in the sequence. This change does not affect the model's output directly yet. These attention maps are summed across all heads and layers, resulting in a single attention weight matrix A_{sum}.

We want to use L1-Loss on this matrix, which encourages the model to reduce the attention weights. However, due to the softmax operation within the MHA module, the matrix sums to a constant, making L1-Loss ineffective. There are multiple ways to address this, but through experimentation, we found that the most effective way is to exclude token self-attention (the diagonal of the attention matrix) from the loss calculation. This shifts unnecessary attention towards the diagonal, which decreases the overall information exchange between tokens.

This auxiliary loss is added to the main loss with a weight of 0.01 and can be expressed mathematically as:

$$A_{\text{sum}} = \sum_{l=1}^{L} \sum_{h=1}^{H} A_{l,h} \tag{4.2}$$

$$\text{Loss}_{\text{aux}} = A_{sum} \odot (1 - I) \tag{4.3}$$

where $A_{l,h}$ is the attention weight matrix Attention(Q_l, K_l, V_l) in layer l and head h, I is the identity matrix and $\odot$ is the element-wise multiplication.

Attention Thresholding Additionally, to ensure that low attention weights are not amplified in later layers, we add a threshold to the attention weight matrix. This is implemented as an adjustable hyperparameter part of the transformer. Any attention weights below this threshold in an attention head are set to 0 in the ScaledDotProduct module. Mathematically, this can be expressed as:

$$A_{\text{mod } l,h} = A_{l,h} \odot \mathbb{1}_{A_{l,h} > T} \tag{4.4}$$

where $T = 0.1$ is the threshold and $\mathbb{1}$ is the indicator function. To prevent instability during training, we save the original attention matrix $A_{l,h}$ which is used to calculate the auxiliary loss and use the modified attention matrix $A_{\text{mod } l,h}$ for the forward pass (and prediction).

Attention Interpretability For interpretability and analysis in a later chapter, we also track which tokens attend to which other tokens through a binary mask η, which is initialized as an identity matrix and updated across layers:

$$\eta_l = \eta_{l-1} \cdot \sum_{h=1}^{H} A_{l,h} \tag{4.5}$$

where η_l is the attention mask for layer l, and η_0 is initialized as an identity matrix. The final mask $\eta = \eta_L$ (where L is the number of layers) allows us to visualize the flow of information between tokens for specific transitions and analyze the learned attention patterns, providing valuable insights into how the model makes decisions.

4.4 U-Net

Similar to the transformer section, this section will mainly focus on the work done to adapt the U-Net architecture for the Minigrid environment, as the general U-Net architecture has already been presented in section 2.4.

We chose the hyperparameters outlined in table A.2, which were found to work well during hyperparameter optimization sweeps.

State/Action to Input Grid Like the transformer models, the U-Net requires first discretization of the Minigrid state, as explained in section 4.1. This results in a grid of size `grid_height` $\times$ `grid_width` with 24 channels. Afterward, the action is concatenated to every cell in the grid, resulting in an input tensor of size `grid_height` $\times$ `grid_width` $\times$ 27 (24 for the state and 3 for the action).

U-Net This input was processed through a U-Net architecture with three downsampling stages using MaxPool2D and DoubleConv modules, reducing the spatial dimensions to 2×2. The network then used three upsampling stages with skip connections from the corresponding downsampling stages.

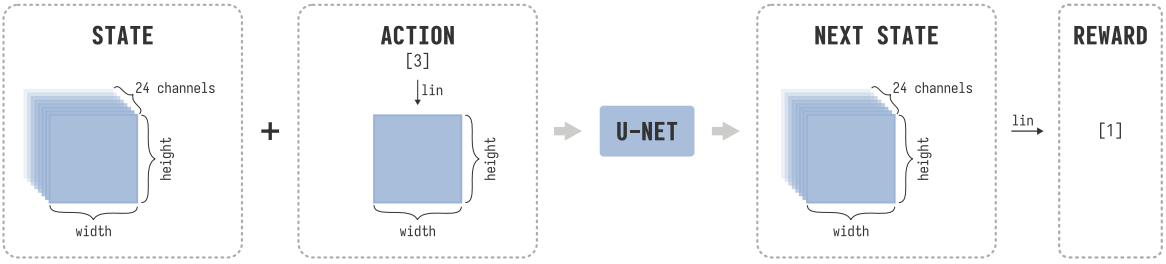


Figure 4.3: U-Net implementation. The model concatenates the action to every cell in the current-state grid and then processes the input through multiple downsampling and upsampling stages with skip connections. The output is a grid of the same size as the input, representing the next state. The reward is predicted separately using a linear layer.

Output Grid to Next-State/Reward The U-Net returns an output grid of the same size as the input but with only 24 channels, representing the next state. The reward is predicted separately using a linear layer by flattening the output grid and passing it through two linear layers with ReLU activation (grid_height $\times$ grid_width $\times$ $24 \to 256 \to 1$).

Loss Function Similar to the transformer models, the loss function was a weighted combination of the next-state loss and reward loss. While the classical and sparse transformers used a combination of cross-entropy loss (focal loss) and MSE loss, the U-Net used a custom loss function called rebalanced focal loss for the state, which is a variant of focal loss that incorporates class weighting to address the class imbalance in the training data. We tried to use the same focal loss as the transformer models but found that it significantly underperformed compared to the rebalanced focal loss. The final loss was a combination of the next-state loss (rebalanced focal loss) and reward loss (MSE Loss), weighted 0.8 and 0.2 respectively.

The rebalanced focal loss extends focal loss by incorporating class weighting:

$$\text{weight}_c = \begin{cases} \frac{\max(class\_counts)}{class\_counts_c} & \text{if } class\_counts_c > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

where $class\_counts$ is the number of samples for each class in the training data. The rebalanced focal loss can then be expressed as:

$$\text{RebalancedFocalLoss}(predictions, targets) = \frac{1}{N}\sum_{i=1}^{N}(1-p_t)^{\gamma} \cdot \text{weight}_{c_i} \cdot \text{CE}_i \tag{4.7}$$

Chapter 5

Experiments and Results

This chapter presents a comprehensive evaluation of our sparse transformer architecture compared to classical transformers and U-Net models. We conducted experiments across multiple training configurations to test our central hypothesis: that sparse transformers can achieve better generalization than classical transformers in sparse-interactive environments and specifically Minigrid. Our evaluation encompasses both In-Distribution (ID) and Out-of-Distribution (OOD) scenarios, with detailed error analysis and ablation studies to understand the mechanisms behind model performance.

5.1 Experimental Setup

Our source code is available on GitHub[1] and the models were implemented using PyTorch. All experiments were conducted on a system with the following specifications:

- Operating System: NixOS 25.04
- CPU: AMD Ryzen 7 2700X (16) @ 3.700GHz
- GPU: AMD Radeon RX 7900 XTX (24GB)
- Memory: 32GB

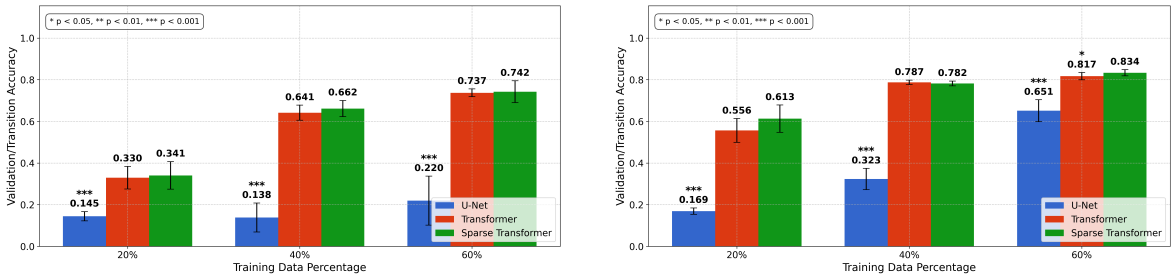
We evaluated the three architectures—classical transformer, sparse transformer, and U-Net—across multiple distinct training datasets, by varying both the number of training environments and the percentage of training samples used. Each configuration was run with 10 different random seeds to ensure statistical reliability. Evaluating the model on multiple configurations allows us to systematically assess how each architecture performs as data availability increases.

We validated the models on two different datasets: one for ID evaluation and one for OOD evaluation. The ID validation set consists of the held-out samples from the training environments, while the OOD validation set consists of 10 new unique, exhaustively sampled, environments. This setup allows us to assess how well the models generalize to unseen environments and how they perform on data that is similar to the training data. The experimental design follows section 3.3.1.

5.2 Training Convergence

All models achieved 100% training transition accuracy across all configurations, which is expected given the deterministic nature of the Minigrid environment and the absence of noise

[1] https://github.com/justanotherariel/SparseTransformerWorldModel



(a) 1 Environment

(b) 2 Environments

Figure 5.1: ID Validation transition accuracy for the U-Net, classical transformer, and sparse transformer models. Each bar represents the average transition accuracy across 10 random seeds, with error bars showing standard deviation. Asterisks denote statistical significance when comparing the performance to the sparse transformer. This data can be found in appendix A.2 in tabular form.

in the data. This perfect training accuracy serves as a prerequisite for evaluating generalization capabilities rather than an indication of overfitting. Interestingly, we observed that validation performance continued to improve significantly after achieving perfect training accuracy. Typically, 100% training accuracy was achieved after approximately 1,000 epochs, while validation accuracy converged after about 3,000 epochs, indicating that the models continued to learn generalization patterns even after perfectly fitting the training data. We thus set the training epochs to 4,000, which was sufficient for all models to stabilize.

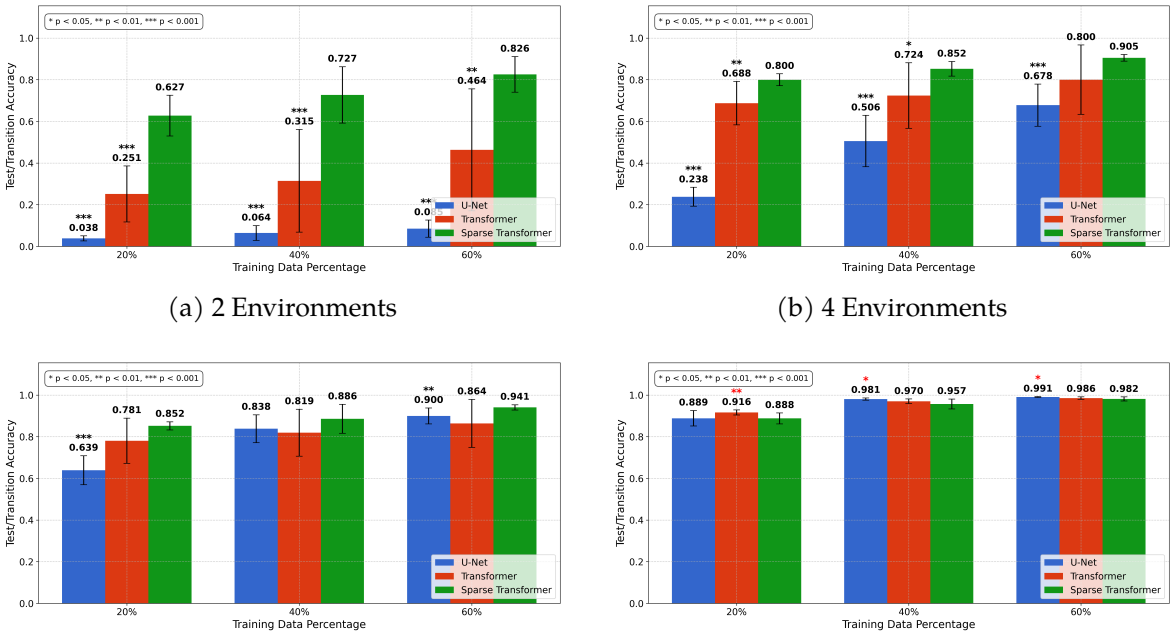
5.3 ID Model Comparison

We first compare the U-Net, classical transformer, and sparse transformer models on ID generalization, where the training and validation data are drawn from the same distribution. For this, we exhaustively sample n environments and split these samples into training and validation sets, where a 'Training Data Percentage' parameter specifies the percentage how many samples are included in the training data set (see section 3.3.1). Training the models on 1 and 2 environments, we observe (in figure 5.1) that the classical and sparse transformer models achieve similar performance, while the U-Net performs significantly worse, catching up only when trained on 4 or more environments. This leads us to conclude, that ID generalization performance does not significantly benefit from sparsity regularization and more challenging environments might be required to see the benefits of the sparse transformer.

5.4 OOD Model Comparison

We now compare the three models on OOD data generalization performance, where the training and validation data are drawn from different distributions. For this, we exhaustively sample n train environments as well as 10 validation environments. From the train environment samples, only a percentage is used for training, which we refer to as 'Training Data Percentage'.

Figure 5.2 shows the OOD validation transition accuracy for the three models across varying training datasets, increasing the available data with each configuration. No model achieved above 1% transition accuracy on the OOD validation set when trained on only 1 environment, which is why we do not show this configuration in figure 5.2. In the OOD scenario, each model exhibits distinct strengths:



(a) 2 Environments

(b) 4 Environments

(c) 8 Environments

(d) 16 Environments

Figure 5.2: Comparison of model performance across different data percentages and environment counts. Each bar (and the number on top) represents the average validation transition accuracy across 10 random seeds, with error bars showing standard deviation. The sparse transformer (green) consistently and significantly outperforms other architectures with limited data, while the U-Net (blue) excels with more abundant data. The classical transformer (red) shows moderate performance across all configurations. Asterisks denote statistical significance when comparing the performance to the sparse transformer. The asterisk is red when the model is better than the sparse transformer. This data can be found in appendix A.2 in tabular form.

- The classical transformer performs reasonably well overall, showing decent performance with limited data and approaching perfect accuracy with more data.
- The U-Net exhibits the best performance with abundant data, achieving near-perfect accuracy. However, it struggles with limited data, particularly in the 4-environment configuration, where it performs poorly.
- The sparse transformer outperforms the classical transformer with limited data but is eventually matched or surpassed by both the classical transformer and U-Net as data volume increases. This supports our hypothesis that the sparsity constraint improves generalization in low-data regimes.

Standard Deviation A notable finding is that the sparse transformer not only achieves higher average validation accuracy with limited data but also demonstrates significantly lower standard deviation across different random seeds. For example, when training on 4 environments with 20% of the data, the classical transformer reaches an average accuracy of 0.6876 ± 0.1042, while the sparse transformer achieves 0.7998 ± 0.0286. The sparse transformer's standard deviation is approximately one-fourth of the classical transformer's variability. This pattern holds across most configurations, indicating that the sparse transformer consistently finds better solutions regardless of initialization parameters.

Train Envs: 4, Train Data: 20% Figure 5.3 illustrates this convergence behavior for a single experimental configuration (4 environments, 20% data). After approximately 2,750 epochs,

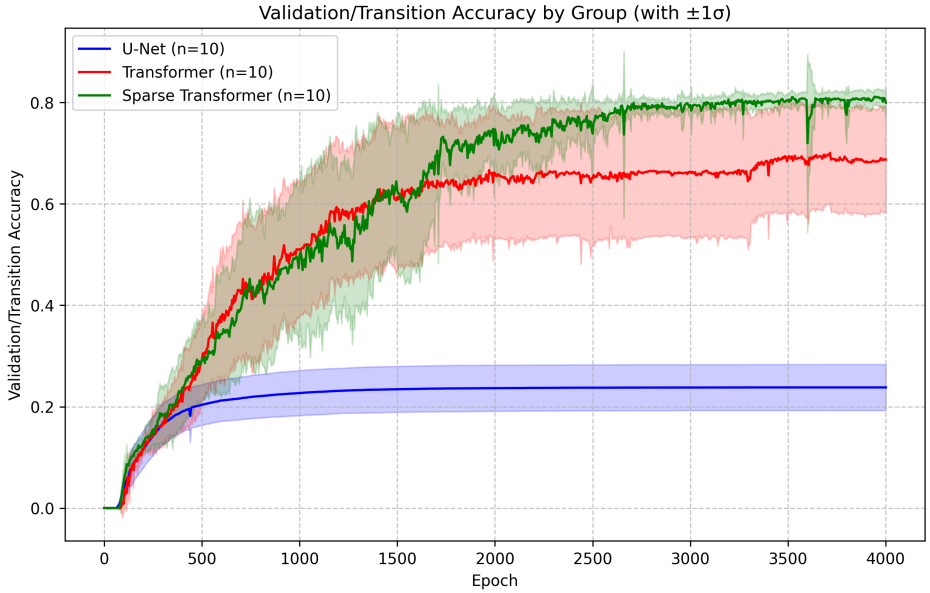


Figure 5.3: Transition accuracy over training epochs for Classical Transformer, Sparse Transformer, and U-Net with 4 Train Environments and 20% Train Data. Shaded areas represent the standard deviation across 10 random seeds. Note how the sparse transformer starts to converge to a consistent solution with minimal variance after approximately 1,750 epochs.

all seeds of the sparse transformer converge to a similar solution, while the classical transformer and U-Net continue to show high variability. This suggests that the sparse transformer effectively reduces the solution space, leading to more consistent and reliable performance.

5.5 OOD Error Analysis

In this section, we analyze the errors made by the models on the OOD validation set. We first qualitatively analyze the common failure patterns observed across the models, followed by a quantitative analysis of these error types.

5.5.1 Qualitative Analysis

Our qualitative analysis revealed several common failure patterns across the models:

1. **Agent Inconsistencies**: The agents' position is not consistently predicted in the next state, leading to incorrect predictions of the agent's location. We observed, for example, that the agent sometimes appears in the wrong cell, is not predicted at all or is predicted multiple times in the next state.
2. **Grid Modification**: The transformer models specifically sometimes modify cells unrelated to the agent's position, indicating overfitting to specific training grids.
3. **Incorrect Rotation**: In some cases, the agent rotates in the wrong direction.

5.5.2 Quantitative Analysis

We now quantitatively analyze the errors made by the models on the OOD validation set and focus on models trained on 2 environments with 20% of the available data, as this configuration highlights the differences in generalization capabilities most clearly. The results for other training configurations can be found in appendix A.2.

- **Action-dependent Performance**: Categorizing the evaluated samples by action type (forward, and rotation) shows that the transformer models struggle most with the forward action, which requires more complex reasoning about the agent's position and the state of the forward cell. Still, the sparse transformer outperforms the classical transformer in both action types, achieving 0.2825 ($\pm$ 0.0470) OOD validation transition accuracy for forward actions and 0.8 ($\pm$ 0.1338) for rotation actions, compared to the classical transformer's 0.0876 ($\pm$ 0.0374) and 0.3334 ($\pm$ 0.1877), respectively.

- **Object Accuracy**: One of the reasons for this significant increase in transition accuracy of the sparse transformer is its ability to predict the object type of all cells in the next state accurately. The sparse transformer achieves an object accuracy of 0.9334 ($\pm$ 0.1163), while the classical transformer achieves only 0.3805 ($\pm$ 0.1937). This difference is statistically significant ($t = 7.73$, $p < 0.001$, Cohen's $d = 3.46$). The U-Net claims the lowest object accuracy of 0.2375 ($\pm$ 0.0672). This indicates that due to the sparsity regularization, the sparse transformer learns that during a transition, most cell variables should remain unchanged, and only the agent's position needs to be updated.

- **Agent Accuracy**: Considering only the agents' positional predictions, disregarding any errors made regarding cell types, we find that the sparse transformer achieves an agent accuracy of 0.6683 ($\pm$ 0.0511), while the classical transformer achieves 0.6173 ($\pm$ 0.0433). This difference is also statistically significant ($t = 4.64$, $p < 0.001$, Cohen's $d = 2.07$). The U-Net again achieves the lowest agent accuracy of 0.1689 ($\pm$ 0.0364). This means that the sparse transformer not only was able to learn that most cells should remain unchanged during a transition, but also was able to better generalize the movement of the agent to new unseen environments.

- **Reward Prediction Difficulty**: All models struggle with correctly predicting positive rewards, which are highly underrepresented in the training data (most rewards are 0), making this task particularly prone to overfitting. Nevertheless, the sparse transformer was able to achieve a 0.5818 ($\pm$ 0.2307) reward prediction accuracy, neither the classical transformer nor the U-Net were able to correctly predict a single positive reward.

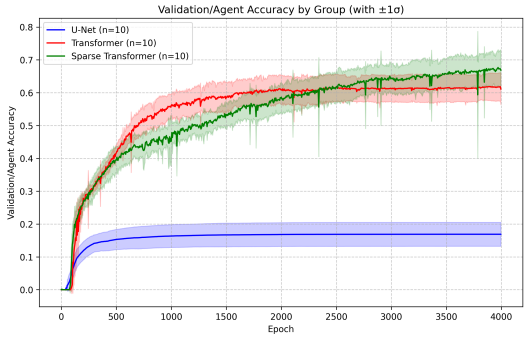
These results indicate that the sparsity regularization term, and the added thresholding, allows the transformer to generalize more effectively on multiple levels, altough only in low-data regimes. As training data becomes more abundant, the classical transformer and U-Net eventually catch up to the sparse transformer's performance.

5.6 Sparse Transformer: η Attention Pattern Analysis

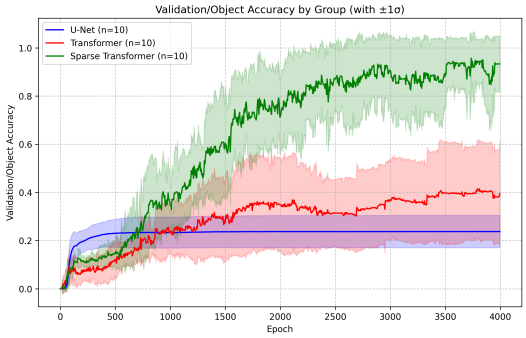
The sparse attention mechanism allows us to analyze and interpret how information flows through the network by examining the η attention map presented in section 4.3. By analyzing the attention patterns, we can gain insights into how the model learns to process information and make predictions. In this section, we will present some illustrative examples of the η attention maps learned by the sparse transformer. For real examples generated by the sparse transformer, see appendix A.4.

5.6.1 Qualitative Analysis

Our qualitative analysis identified several distinct attention patterns. For brevity, we will name the cell containing the agent as the **agent cell**, and the cell in front of the agent as the **forward cell**. The attention patterns are as follows:



(a) OOD validation agent accuracy over training epochs. Agent accuracy is defined as the percentage of samples where the agent's position is correctly predicted.



(b) OOD validation object accuracy over training epochs. Object accuracy is defined as the percentage of samples where the object type of all cells is correctly predicted.

Figure 5.4: Performance comparison of Classical Transformer, Sparse Transformer, and U-Net with 4 Train Environments and 20% Train Data. Shaded areas represent the standard deviation across 10 random seeds.

Rotation Actions Rotation actions (turn left/right) are straightforward, and, considering the rotation OOD validation transition accuracy, easily learned by the model. Thus only the ideal pattern is observed in the η attention maps, where tokens only attend to themselves. This is due to the fact that rotation only affects the agent's orientation within its agent cell, and no other cells change. An example of this pattern is shown in figure 5.5a.

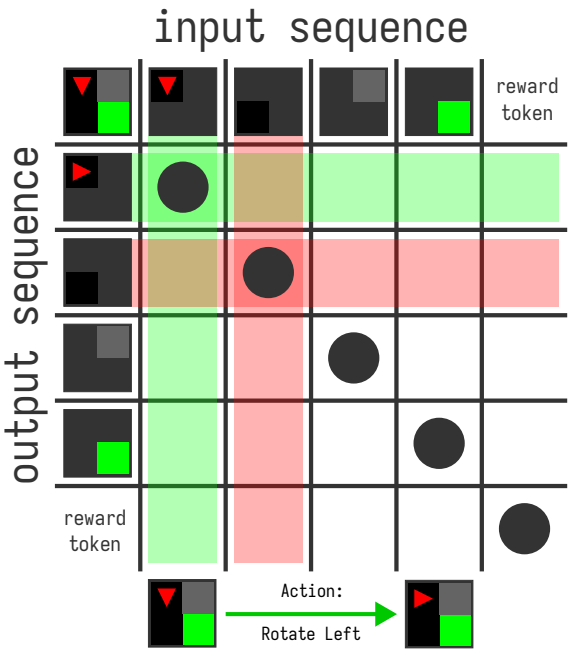
Forward Action The forward action is more complex, as it requires the model to consider the agent's position and the state of the forward cell. We observed three main patterns in the η attention maps for forward actions:

- **Ideal Pattern**: Most tokens attend only to themselves, except for the agent cell and the forward cell, which exchange information with each other. An example of this pattern is shown in figure 5.5b.
- **Agent Overfit Pattern**: The agent cell receives no information from the forward cell. This indicates that the model has learned that the agent can always move forward from this position, neglecting to check whether the forward cell is a wall or empty space. An example of this pattern is shown in figure 5.5c.
- **Multiple Agent Pattern**: A random output token receives information from the agent cell, potentially causing an additional agent to appear in the prediction. An example of this pattern is shown in figure 5.5d.

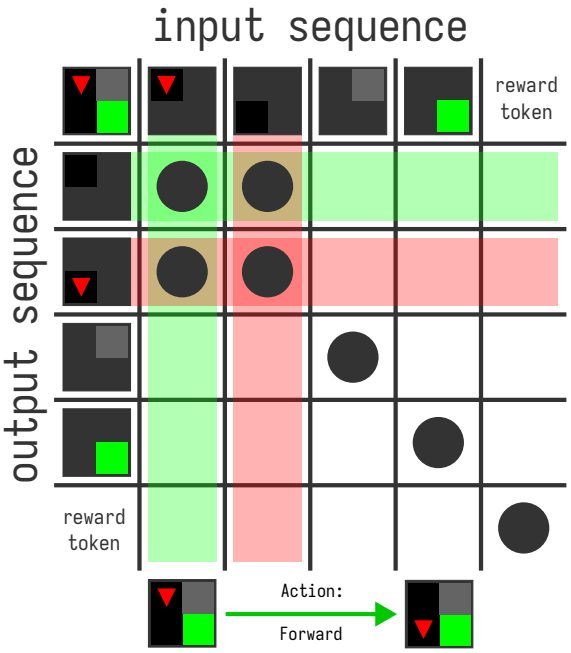
In all scenarios, we occasionally observed random information exchange between tokens unrelated to the agent's position, usually with no effect on the prediction.

5.6.2 Quantitative Analysis

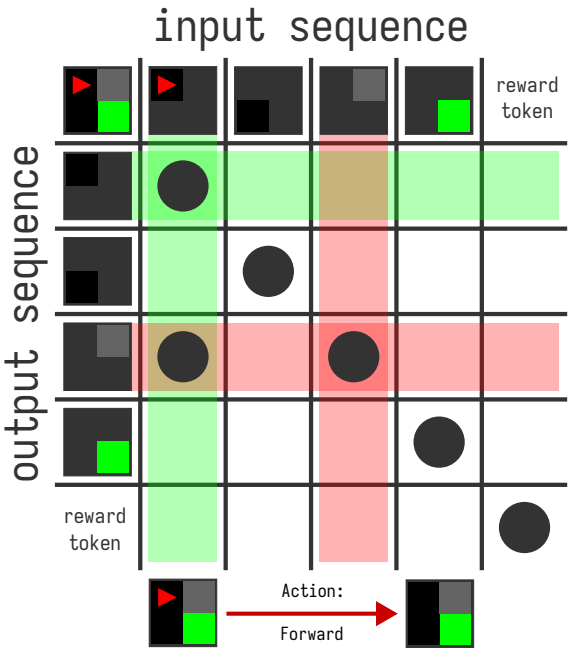
Quantitatively, we analyzed the prevalence of these patterns in our trained models. First, we examined the distribution of the number of tokens attending to each other in the η attention maps by asking the question: "How often do exactly x tokens attend to each other across all validation samples?" This analysis provides insights into how well our sparsity regularization helps the model to focus on the minimal necessary information for making predictions. Figure 5.6 shows the distribution of the number of tokens attending to each other in the η attention maps for the sparse transformer trained on 4 environments with 20% of the data.



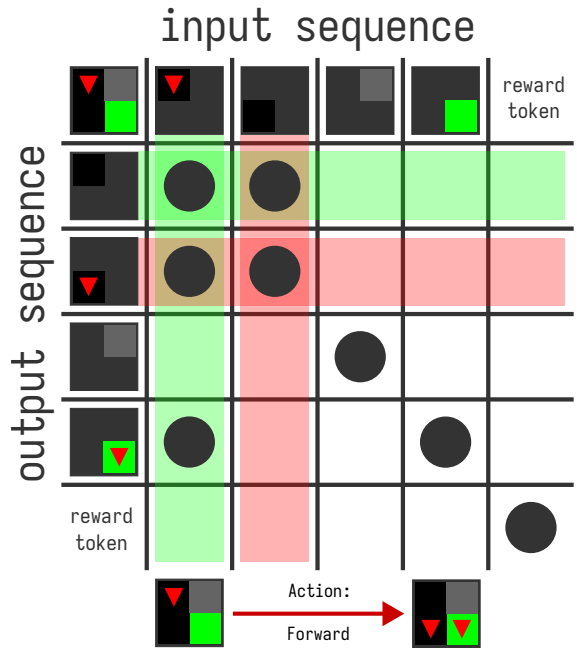
(a) Action: Rotate. Shows the ideal pattern when rotating the agent. All tokens attend to themselves, as rotation only affects the agent cell and it does not require any other information. All other cells remain unchanged.



(b) Action: Forward. Shows the ideal pattern when walking towards an empty field. All tokens attend to themselves, except for the agent cell and the forward cell, which exchange information with each other.



(c) Action: Forward. Shows the agent overfit pattern, where the forward cell did not attend to the agent cell, indicating that the model has learned that the agent can always move forward from this position, neglecting to check whether the forward cell is a wall or empty space.



(d) Action: Forward. Example of a spurious agent. The model predicts two agents in the next state, due to the fact that a random output token receives information from the agent cell.

Figure 5.5: Illustrative examples of η attention patterns learned by the sparse transformer. η attention matrices are binary masks, a black dot indicates that the output token receives information from the input token. The green and red shades indicate the agent cell and forward cell, respectively. For full-size examples, see appendix A.4.

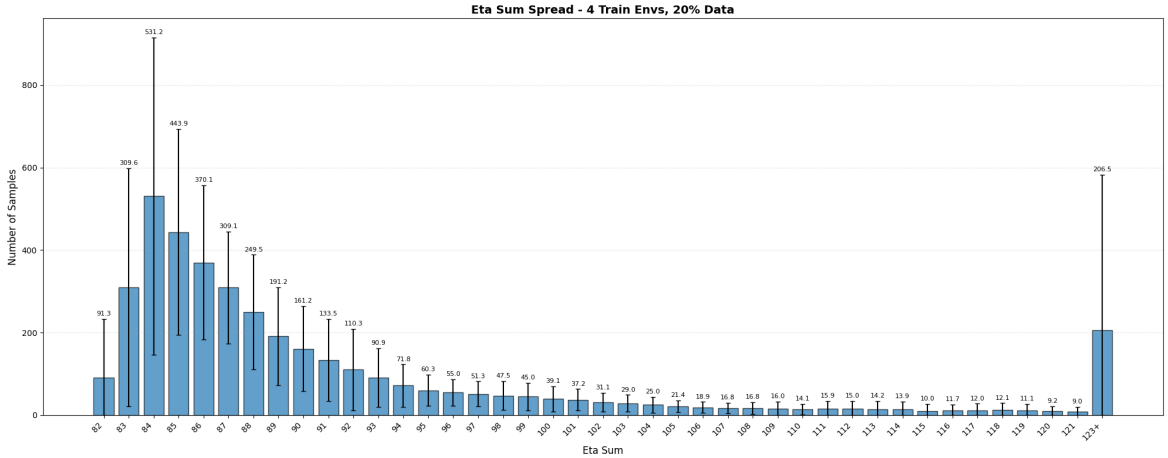


Figure 5.6: η Attention Map Sparsity on OOD validation data when training the model on 4 Mingrid 9×9 environments and keeping 20 % of the data. The plot shows the number of validation samples for which a specific number of tokens attend to each other. The ideal pattern would be a bar at 82 ($9 \times 9 + 1$, only tokens attending to themselves, for rotation action), and 84 (tokens attending to themselves plus information exchange between agent cell and forward cell, for forward action).

The plot shows that the L1 loss encourages sparsity in the η attention maps, but most samples attend to more components than necessary, with a peak at around 84 and a long tail towards $123+$. Considering that the ideal pattern for rotation actions would be a bar at 82 (only tokens attending to themselves, for rotation action), and 84 (tokens attending to themselves plus information exchange between agent cell and forward cell, for forward action), this is suboptimal. We tried to increase the weight of the sparsity regularization term in the loss function to encourage the model to learn even sparser attention, but while this did lead to a stronger peak at 82 and the OOD validation transition accuracy increased when trained on very little data—achieving 0.7096% ($\pm$0.0448) OOD validation transition accuracy—model performance plateaued at around 85% accuracy even when training on 16 environments with 60% of the data.

Categorizing the samples into a group based on the action taken, we found that the rotation action has stronger peaks at 82 and 84, while the forward action has a peak at around 90. This divergence is unsurprising, as the forward action requires more information exchange cells. The sparse transformer thus correctly learns that to predict a single next-state component, less information from other components is needed for rotation actions than for forward actions.

Overall, we can conclude that the sparse transformer does learn to focus on fewer components and thus our method works, but it does not achieve perfect sparsity in the attention maps. Nevertheless, even with the suboptimal attention maps, we can still observe a significant improvement in OOD validation transition accuracy compared to the classical transformer, indicating that the sparsity regularization does help the model to generalize better.

5.7 Additional Experiments

To better understand the robustness and mechanisms underlying our sparse transformer approach, we conducted two additional experiments that examined different aspects of the model's behavior.

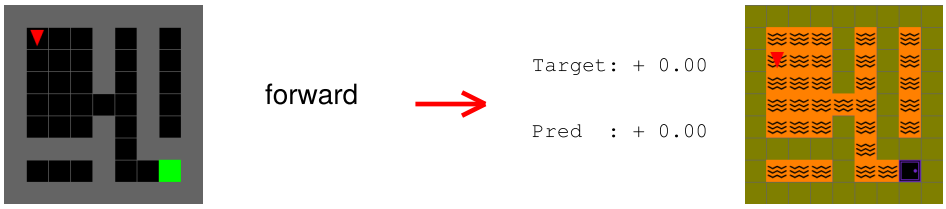


Figure 5.7: Example of mapping next-state components to another representation. The current state is unchanged, but each next state component is mapped to a different deterministic representation.

5.7.1 Next-State Component Mapping

We investigated whether the sparse transformer's improved generalization stems from learning to preserve unchanged state components or from a more fundamental understanding of sparse dependencies. To test this, we trained the sparse transformer on a modified dataset where each next-state component was mapped to a different representation using a deterministic function while keeping the current state unchanged. This transformation ensures that all cells change after a transition, as illustrated in figure 5.7.

The results showed no significant change in validation transition accuracy for both ID and OOD evaluation, indicating that the model's improved generalization stems from learning appropriate sparse dependencies rather than simply preserving unchanged components.

5.7.2 Separate Action Token Architecture

We explored an alternative architecture where the action is represented as a separate token rather than concatenated to every state component. In this "SepAction" variant, the action token attends to the current state independently, which we hypothesized might provide a more natural modeling of the action-state relationship.

However, this modification significantly decreased validation transition accuracy for both ID and OOD evaluation across both sparse and classical transformers. This result suggests that the direct integration of action information with each state component significantly enhances the transformers' ability to generalize and separating them requires the model to learn complex cross-modal relationships from scratch.

Despite the overall performance decrease, the SepAction sparse transformer still outperformed the SepAction classical transformer on OOD validation metrics, particularly object accuracy. This confirms that sparsity regularization provides consistent benefits across different architectural choices. The complete results for the SepAction variants are presented in table A.10.

Chapter 6

Related Work

Our research builds upon several established areas of study in the field of reinforcement learning and transformer architectures. This chapter contextualizes our work on sparse transformers for model-based reinforcement learning by reviewing relevant literature on sparse attention mechanisms and structured world models in reinforcement learning. We highlight existing approaches and identify the gaps our research aims to address, particularly in improving generalization in environments with sparse interactions.

6.1 Sparse Attention Mechanisms

The development of sparse attention mechanisms represents a significant research direction in transformer architectures. Child et al. (2019) introduced the Sparse Transformer, which uses fixed and strided sparse attention patterns to reduce the quadratic complexity of attention, enabling processing of longer sequences. They showed that this approach delivered competitive performance on image classification tasks compared to dense attention, while significantly reducing memory and computation costs.

Building on this idea, Tay et al. (2020) proposed Sparse Sinkhorn Attention, which employs differentiable sorting of internal representations and learns to generate latent permutations over sequences. This approach allows for computing quasi-global attention with only local windows, improving memory efficiency.

Correia, Niculae, and Martins (2019) introduced the adaptively sparse Transformer, which replaces the standard softmax with α-entmax, a differentiable generalization that allows low-scoring words to receive precisely zero weight. Their approach enables attention heads to have flexible, context-dependent sparsity patterns.

Unlike these approaches that focus primarily on computational efficiency, our work explores how sparsity can serve as an inductive bias that improves generalization in reinforcement learning environments with sparse interactions. While prior sparse attention mechanisms often employ fixed patterns or complex sorting networks, our approach uses a simple L1 regularization and thresholding mechanism that encourages the model to learn which connections are most important based on the environment's structure.

6.2 Structured World Models in Reinforcement Learning

Model-based reinforcement learning has seen renewed interest due to its sample efficiency advantages. Hafner, Lillicrap, I. Fischer, et al. (2019) introduced PlaNet, a purely model-based agent that learns the environment dynamics from images and plans through a latent space. Building on this, Hafner, Lillicrap, Ba, et al. (2020) presented Dreamer, which efficiently learns behaviors by propagating analytic gradients of learned state values back through trajectories imagined in the compact state space of a learned world model.

Particularly relevant to our work is research that incorporates structural priors into world models. Our approach differs by focusing specifically on environments with sparse interactions, where state transitions depend primarily on a small subset of the state space. Rather than imposing a fixed relational structure, our sparse transformer learns which state components are relevant for prediction through attention mechanisms, potentially offering greater flexibility across different environment types.

6.3 Distinctiveness of Our Approach

Our work synthesizes ideas from sparse attention mechanisms and structured world models but with a unique focus on leveraging sparsity as an inductive bias for generalization in environments with sparse interactions.

The key distinctions of our approach include:

1. A focus on generalization rather than computational efficiency as the primary motivation for sparse attention
2. A simple regularization and thresholding approach that encourages the transformer to learn which connections are important, rather than imposing fixed attention patterns
3. A systematic evaluation methodology specifically designed to test generalization in environments with sparse interactions
4. A direct comparison with models that have different inductive biases (standard transformers and U-Net), providing insights into which architectural choices are most effective for different data regimes

By demonstrating that sparse transformers can generalize better than classical transformers in sparse-interactive environments, especially in low-data regimes, our work contributes to the ongoing effort to develop more efficient and robust reinforcement learning algorithms through the incorporation of appropriate inductive biases.

Chapter 7

Conclusion

This thesis has investigated the hypothesis that sparse transformers can better generalize than classical transformers in sparse-interactive environments, particularly in low-data regimes. Through systematic experimentation and analysis, we have demonstrated several key findings that support this hypothesis.

Generalization Improvements Our proposed sparsity-regularized transformer consistently outperformed the classical transformer and U-Net when evaluating them on OOD data and training them on limited data. This advantage was most pronounced when the training dataset consisted of only 2 or 4 unique environments with 20-40% of the available samples. Here, the sparse transformer achieved up to 41% higher average validation transition accuracy, a statistically significant improvement ($p < 0.001$, Cohen's $d = 2.07$, 95% CI: $[0.22, 0.60]$). The performance gain diminished as the amount of training data increased, with all architectures eventually converging to near-perfect accuracy with sufficient data. The U-Net architecture specifically, which has a strong spatial inductive bias well-suited to grid-based environments, outperformed both transformer variants with abundant data, highlighting the trade-offs between different inductive biases.

Perhaps more significantly, our sparse transformer demonstrated remarkably lower variance across random initializations compared to the classical transformer. This consistency—with standard deviations often one-fourth that of the classical transformer—suggests that the sparsity constraint guides the model toward more stable solutions, reducing sensitivity to initialization and leading to more reliable performance across training runs.

Learned Attention Patterns Qualitative analysis of the learned attention patterns revealed that the sparse transformer successfully captured the minimal information exchange between components of the environment. For rotation actions, it learned to keep information local, while for forward actions, it learned to exchange information primarily between the agent's current position and the forward cell. Nevertheless, the sparse transformer also often included some irrelevant cells in its attention, indicating that it did not achieve perfect sparsity. Further increasing the weight of the sparsity regularization term made the model more conservative, but also led to a drop in performance, suggesting a trade-off between sparsity and expressiveness.

Broader Implications We demonstrate that incorporating appropriate inductive biases, in this case, sparsity in attention patterns, can significantly improve model performance without requiring complex architectural changes. This simplicity is a strength, as it suggests the possibility of applying similar principles to a wide range of reinforcement learning problems with sparse dependency structures.

In conclusion, our research provides empirical evidence that sparse transformers can indeed generalize better than classical transformers in sparse-dependent environments, particularly when training data is limited. By encouraging the model to focus only on relevant state components, we effectively reduce overfitting and improve sample efficiency. These findings contribute to the broader goal of developing more efficient and robust reinforcement learning algorithms through the incorporation of appropriate inductive biases, opening avenues for future research in this promising direction.

Chapter 8

Future Work

While our research demonstrates the effectiveness of sparse transformers in improving generalization for sparse-interactive environments, several promising avenues remain for future investigation. These directions could further validate our findings, extend the applicability of our approach, and address current limitations.

Positional Encoding One particularly intriguing direction involves investigating the impact of positional encoding choices on model performance. Our preliminary experiments with sinusoidal positional embeddings showed potential improvements in generalization compared to learned embeddings, especially using 2D sinusoidal embeddings in grid-based environments. However, evaluating this further was beyond the scope of our current work. Future research could systematically explore various positional encoding strategies, including learned embeddings, sinusoidal encodings, and even hybrid approaches that combine both. This could help identify the most effective encoding for different types of sparse-interactive environments and provide deeper insights into how positional information influences model performance.

Handling Partial Observability Another important limitation of our current implementation is its inability to handle partially observable Markov decision processes (POMDPs), as our proposed transformer implementation processes only single state-action pairs without maintaining information from previous observations. Extending our sparse transformer to handle partial observability would significantly broaden its applicability to real-world scenarios where agents rarely have access to complete state information. This could involve incorporating memory mechanisms such as recurrent connections or attention over past states, or alternatively, modifying the architecture to process sequences of state-action pairs simultaneously. Such extensions would need to preserve the sparse attention properties while enabling the model to maintain and selectively access relevant historical information.

Integration into Model-Based RL Frameworks Perhaps the most critical next step involves integrating our sparse transformer into a complete model-based reinforcement learning framework. While we have demonstrated superior generalization in supervised learning settings, the ultimate test lies in whether these improvements translate to better policy learning when the sparse transformer serves as a world model. Integration with established frameworks like PlaNet (Hafner, Lillicrap, I. Fischer, et al. 2019) or Dreamer (Hafner, Lillicrap, Ba, et al. 2020) would allow us to evaluate whether the improved sample efficiency and generalization we observed lead to faster convergence, better asymptotic performance, or improved robustness in online reinforcement learning settings. This integration would also reveal any challenges that arise when using sparse transformers for multi-step planning or when the model must handle the distribution shift that occurs as the policy improves during training.

These future directions collectively aim to transform our promising initial results into a practical tool for improving sample efficiency in reinforcement learning. By addressing current limitations and validating our approach in more challenging settings, we can work toward realizing the full potential of sparse transformers as world models in reinforcement learning applications.

Bibliography

Battaglia, Peter W. et al. (2018). "Relational inductive biases, deep learning, and graph networks". In: *CoRR* abs/1806.01261. arXiv: 1806.01261. URL: http://arxiv.org/abs/1806.01261 (visited on 05/12/2025).

Bellman, Richard (1957). "A Markovian Decision Process". In: *Journal of Mathematics and Mechanics* 6.5. Publisher: Indiana University Mathematics Department, pp. 679–684. ISSN: 0095-9057. URL: https://www.jstor.org/stable/24900506 (visited on 04/27/2025).

Bengio, Yoshua, Aaron Courville, and Pascal Vincent (Aug. 1, 2013). "Representation Learning: A Review and New Perspectives". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 35.8, pp. 1798–1828. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2013.50. URL: https://doi.org/10.1109/TPAMI.2013.50 (visited on 05/12/2025).

Chevalier-Boisvert, Maxime et al. (2023). "Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks". In: *CoRR* abs/2306.13831.

Child, Rewon et al. (Apr. 23, 2019). *Generating Long Sequences with Sparse Transformers*. DOI: 10.48550/arXiv.1904.10509. arXiv: 1904.10509[cs]. URL: http://arxiv.org/abs/1904.10509 (visited on 04/29/2025).

Correia, Gonçalo M., Vlad Niculae, and André F. T. Martins (2019). "Adaptively Sparse Transformers". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). Hong Kong, China: Association for Computational Linguistics, pp. 2174–2184. DOI: 10.18653/v1/D19-1223. URL: https://www.aclweb.org/anthology/D19-1223 (visited on 05/12/2025).

Devlin, Jacob et al. (June 2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423/.

Gardner, Martin (Oct. 1970). "Mathematical Games". In: *Scientific American* 223.4, pp. 120–123. ISSN: 0036-8733. DOI: 10.1038/scientificamerican1070-120. URL: https://www.scientificamerican.com/article/mathematical-games-1970-10 (visited on 05/08/2025).

Gu, Jiatao et al. (Feb. 15, 2018). "Non-Autoregressive Neural Machine Translation". In: International Conference on Learning Representations. URL: https://openreview.net/forum?id=B1l8BtlCb (visited on 05/12/2025).

Hafner, Danijar, Timothy Lillicrap, Jimmy Ba, et al. (Apr. 2020). "Dream to Control: Learning Behaviors by Latent Imagination". In: Eighth International Conference on Learning

Representations. URL: https://iclr.cc/virtual_2020/poster_S1lOTC4tDS.html (visited on 05/12/2025).

Hafner, Danijar, Timothy Lillicrap, Ian Fischer, et al. (June 2019). "Learning Latent Dynamics for Planning from Pixels". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 2555–2565. URL: https://proceedings.mlr.press/v97/hafner19a.html.

Kaiser, Łukasz et al. (2020). "Model Based Reinforcement Learning for Atari". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=S1xCPJHtDB.

Lee, Jason, Elman Mansimov, and Kyunghyun Cho (Oct. 2018). "Deterministic Non-Autoregressive Neural Sequence Modeling by Iterative Refinement". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Ed. by Ellen Riloff et al. Brussels, Belgium: Association for Computational Linguistics, pp. 1173–1182. DOI: 10.18653/v1/D18-1149. URL: https://aclanthology.org/D18-1149/.

Lin, Tsung-Yi et al. (2017). "Focal Loss for Dense Object Detection". In: *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2999–3007. DOI: 10.1109/ICCV.2017.324.

Mnih, Volodymyr, Adrià Puigdomènech Badia, et al. (June 19, 2016). "Asynchronous methods for deep reinforcement learning". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, pp. 1928–1937. (Visited on 05/12/2025).

Mnih, Volodymyr, Koray Kavukcuoglu, et al. (Dec. 19, 2013). *Playing Atari with Deep Reinforcement Learning*. DOI: 10.48550/arXiv.1312.5602. arXiv: 1312.5602[cs]. URL: http://arxiv.org/abs/1312.5602 (visited on 03/06/2025).

Pathak, Deepak et al. (Aug. 2017). "Curiosity-driven Exploration by Self-supervised Prediction". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 2778–2787. URL: https://proceedings.mlr.press/v70/pathak17a.html.

Puterman, Martin (Aug. 2014). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 684 pp. ISBN: 978-1-118-62587-3. URL: https://www.wiley.com/en-us/Markov+Decision+Processes%3A+Discrete+Stochastic+Dynamic+Programming-p-9781118625873 (visited on 04/27/2025).

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab et al. Cham: Springer International Publishing, pp. 234–241. ISBN: 978-3-319-24574-4.

Schulman, John et al. (2017). "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347. arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

Shazeer, Noam (Feb. 12, 2020). *GLU Variants Improve Transformer*. DOI: 10.48550/arXiv.2002.05202. arXiv: 2002.05202[cs]. URL: http://arxiv.org/abs/2002.05202 (visited on 06/09/2025).

Silver, David et al. (Jan. 2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587. Publisher: Nature Publishing Group, pp. 484–489. ISSN: 1476-4687. DOI: 10.1038/nature16961. URL: https://www.nature.com/articles/nature16961 (visited on 05/12/2025).

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56, pp. 1929–1958. ISSN: 1533-7928. URL: http://jmlr.org/papers/v15/srivastava14a.html (visited on 05/08/2025).

Sutton, Richard S. (Jan. 1, 1990). "Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming". In: *Machine Learning Proceedings 1990*. Ed. by Bruce Porter and Raymond Mooney. San Francisco (CA): Morgan Kaufmann, pp. 216–224. ISBN: 978-1-55860-141-3. DOI: 10.1016/B978-1-55860-141-3.50030-4. URL:

https://www.sciencedirect.com/science/article/pii/B9781558601413500304 (visited on 04/28/2025).

Sutton, Richard S. and Andrew G. Barto (Nov. 13, 2018). *Reinforcement Learning, second edition: An Introduction*. Google-Books-ID: uWV0DwAAQBAJ. MIT Press. 549 pp. ISBN: 978-0-262-35270-3.

Tay, Yi et al. (July 13, 2020). "Sparse sinkhorn attention". In: *Proceedings of the 37th International Conference on Machine Learning*. Vol. 119. ICML'20. JMLR.org, pp. 9438–9447. (Visited on 05/12/2025).

Tobin, Josh et al. (Sept. 2017). "Domain randomization for transferring deep neural networks from simulation to the real world". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). ISSN: 2153-0866, pp. 23–30. DOI: 10.1109/IROS.2017.8202133. URL: https://ieeexplore.ieee.org/document/8202133 (visited on 05/12/2025).

Vaswani, Ashish et al. (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Watkins, Christopher J.C.H. (1989). "Learning from Delayed Rewards". PhD thesis. King's College. URL: https://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf (visited on 03/06/2025).

Williams, Ronald J. (May 1, 1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8.3, pp. 229–256. ISSN: 1573-0565. DOI: 10.1007/BF00992696. URL: https://doi.org/10.1007/BF00992696 (visited on 03/06/2025).

List of Figures

List of Tables

Acronyms

CE cross-entropy

FFN feed-forward network

ID In-Distribution

MDP Markov decision process

MHA multi-head attention

MSE mean squared error

NAT non-autoregressive transformer

OOD Out-of-Distribution

POMDP partially observable Markov decision process

PPO proximal policy optimization

RL reinforcement learning

RNN recurrent neural network

Appendix A

Complementary Material

In this appendix, we provide additional data and results that complement the findings presented in the main text.

A.1 Hyperparameters

In this section, we present the hyperparameters used for training the classical transformer, sparse transformer, and U-Net models. These hyperparameters were chosen based on hyperparameter optimization sweeps and are consistent across all runs to ensure comparability unless indicated otherwise.

Table A.1: Transformer and Sparse Transformer Hyperparameters

Parameter	Transformer	Sparse Transformer
Training		
Epochs	4000	4000
Batch size	2048	2048
Optimizer	Adafactor	Adafactor
Learning rate	0.01	0.01
Model		
Number of heads	4	4
Number of layers	3	3
d_{model}	128	128
d_{ff}	128	128
Dropout probability	0.15	0.15
Attention threshold	–	0.1
Loss		
State loss function	FocalLoss	FocalLoss
State loss weight	0.8	0.8
Reward loss function	MSELoss	MSELoss
Reward loss weight	0.2	0.2
A_{sum} loss function	–	L1Loss (excl. diagonal)
A_{sum} loss weight	–	0.01

A.2 OOD Validation Accuracies

This section presents the validation accuracies for the different configurations of the classical transformer, sparse transformer, and U-Net models in tabular form. Each table lists

Table A.2: U-Net Hyperparameters

Parameter	Value
Training	
Epochs	4000
Batch size	2048
Optimizer	AdamW
Learning rate	0.005
Scheduler	StepLRScheduler
Scheduler decay time	150
Scheduler decay rate	0.7
Model	
Hidden channels	[48, 96, 192]
Loss	
State loss function	RebalancedFocalLoss
State loss weight	0.8
Reward loss function	MSELoss
Reward loss weight	0.2

the mean and standard deviation across the 10 runs with different initialization seeds for each configuration. A configuration is defined by the number of environments and the keep probability. For example, the configuration "envs: 4, keep: 0.2" indicates that the training dataset consisted of 4 unique environments that were exhaustively sampled after which 20% of the samples were randomly chosen to be kept. The remaining 80% of the samples were discarded. The OOD validation dataset consists of 10 other unique environments, which were exhaustively sampled and evaluated on in their entirety.

1. **Validation Transition Accuracy:** The accuracy of the model in predicting the complete next state and reward correctly given the current state and action is shown in table A.3.

2. **Validation Transition Accuracy (Forward):** Grouping the samples together by the action taken, the accuracy of the model in predicting the complete next state and reward correctly when the action is 'forward' is shown in table A.4.

3. **Validation Transition Accuracy (Rotate):** Similarly, table A.5 shows the accuracy of the model in predicting the complete next state and reward correctly when the action is 'rotate left/right'.

4. **Validation One Agent Samples Accuracy:** This accuracy represents the percentage of predicted samples where the next-state contains only one agent, irrespective wheter this agent is in the correct position. Other errors, such as predicting the wrong object or color of cells, are also not taken into account. This accuracy is shown in table A.6.

5. **Validation Agent Accuracy:** This accuracy focuses on a single variable of the next-state, the agent and measures the percentage of samples where the agent is predicted in the correct position. This accuracy does not take into account the reward or any prediction about the action/state of any cell. The results are shown in table A.7.

6. **Validation Object Accuracy:** Similarly, this accuracy focuses on the object variable of the next-state and measures the percentage of samples where the object is predicted in the correct position. The results are shown in table A.8.

Configuration	Classical	Sparse	U-Net
envs: 1, keep: 0.2	0.0001 (± 0.0002)	0.0018 (± 0.0053)	0.0000 (± 0.0000)
envs: 1, keep: 0.4	0.0000 (± 0.0000)	0.0194 (± 0.0279)	0.0000 (± 0.0000)
envs: 1, keep: 0.6	0.0000 (± 0.0000)	0.0472 (± 0.0810)	0.0000 (± 0.0001)
envs: 2, keep: 0.2	0.2514 (± 0.1346)	0.6275 (± 0.0978)	0.0384 (± 0.0120)
envs: 2, keep: 0.4	0.3146 (± 0.2468)	0.7271 (± 0.1352)	0.0641 (± 0.0355)
envs: 2, keep: 0.6	0.4635 (± 0.2928)	0.8256 (± 0.0853)	0.0848 (± 0.0412)
envs: 4, keep: 0.2	0.6876 (± 0.1042)	0.7998 (± 0.0286)	0.2380 (± 0.0455)
envs: 4, keep: 0.4	0.7239 (± 0.1573)	0.8522 (± 0.0351)	0.5056 (± 0.1233)
envs: 4, keep: 0.6	0.8004 (± 0.1663)	0.9050 (± 0.0158)	0.6778 (± 0.1014)
envs: 8, keep: 0.2	0.7807 (± 0.1086)	0.8521 (± 0.0194)	0.6392 (± 0.0694)
envs: 8, keep: 0.4	0.8193 (± 0.1126)	0.8861 (± 0.0696)	0.8382 (± 0.0672)
envs: 8, keep: 0.6	0.8640 (± 0.1155)	0.9407 (± 0.0125)	0.8999 (± 0.0383)
envs: 12, keep: 0.2	0.7674 (± 0.1119)	0.8627 (± 0.0240)	0.8003 (± 0.0583)
envs: 12, keep: 0.4	0.8372 (± 0.1313)	0.9213 (± 0.0204)	0.9148 (± 0.0250)
envs: 12, keep: 0.6	0.8977 (± 0.1088)	0.9598 (± 0.0110)	0.9743 (± 0.0108)
envs: 16, keep: 0.2	0.9164 (± 0.0120)	0.8879 (± 0.0261)	0.8886 (± 0.0373)
envs: 16, keep: 0.4	0.9703 (± 0.0109)	0.9570 (± 0.0239)	0.9808 (± 0.0053)
envs: 16, keep: 0.6	0.9855 (± 0.0063)	0.9815 (± 0.0097)	0.9906 (± 0.0022)

Table A.3: OOD Validation/Transition Accuracy - Across Configurations (mean ± standard deviation)

A.3 OOD Validation Accuracy: SepAction Transformer Variant

Table A.10 shows the OOD validation transition accuracy of the SepAction Transformer variant, which does not encode the agent's action in the state representation. Instead, it uses adds a separate token to the input sequence that indicates the action to be executed.

Configuration	Classic Comb	Sparse Comb	U-Net
envs: 1, keep: 0.2	0.0000 ($\pm$ 0.0000)	0.0000 ($\pm$ 0.0000)	0.0000 ($\pm$ 0.0000)
envs: 1, keep: 0.4	0.0000 ($\pm$ 0.0000)	0.0049 ($\pm$ 0.0146)	0.0000 ($\pm$ 0.0000)
envs: 1, keep: 0.6	0.0000 ($\pm$ 0.0000)	0.0393 ($\pm$ 0.0947)	0.0000 ($\pm$ 0.0000)
envs: 2, keep: 0.2	0.0876 ($\pm$ 0.0374)	0.2825 ($\pm$ 0.0470)	0.0268 ($\pm$ 0.0140)
envs: 2, keep: 0.4	0.1694 ($\pm$ 0.1608)	0.4170 ($\pm$ 0.0786)	0.0502 ($\pm$ 0.0250)
envs: 2, keep: 0.6	0.3184 ($\pm$ 0.2361)	0.5919 ($\pm$ 0.0739)	0.0808 ($\pm$ 0.0329)
envs: 4, keep: 0.2	N/A	0.4269 ($\pm$ 0.0874)	N/A
envs: 4, keep: 0.4	N/A	0.6034 ($\pm$ 0.0741)	N/A
envs: 4, keep: 0.6	N/A	0.7268 ($\pm$ 0.0457)	N/A
envs: 8, keep: 0.2	N/A	0.5623 ($\pm$ 0.0544)	N/A
envs: 8, keep: 0.4	N/A	0.7197 ($\pm$ 0.0836)	N/A
envs: 8, keep: 0.6	N/A	0.8223 ($\pm$ 0.0372)	N/A
envs: 12, keep: 0.2	N/A	0.5891 ($\pm$ 0.0722)	N/A
envs: 12, keep: 0.4	N/A	0.7642 ($\pm$ 0.0610)	N/A
envs: 12, keep: 0.6	N/A	0.8793 ($\pm$ 0.0330)	N/A
envs: 16, keep: 0.2	N/A	0.6652 ($\pm$ 0.0754)	N/A
envs: 16, keep: 0.4	N/A	0.8709 ($\pm$ 0.0716)	N/A
envs: 16, keep: 0.6	N/A	0.9445 ($\pm$ 0.0291)	N/A

Table A.4: OOD Validation/Transition Accuracy (Forward) - Across Configurations (mean $\pm$ standard deviation)

Configuration	Classic Comb	Sparse Comb	U-Net
envs: 1, keep: 0.2	0.0001 ($\pm$ 0.0003)	0.0027 ($\pm$ 0.0080)	0.0000 ($\pm$ 0.0000)
envs: 1, keep: 0.4	0.0000 ($\pm$ 0.0000)	0.0267 ($\pm$ 0.0369)	0.0000 ($\pm$ 0.0000)
envs: 1, keep: 0.6	0.0000 ($\pm$ 0.0000)	0.0511 ($\pm$ 0.0768)	0.0000 ($\pm$ 0.0001)
envs: 2, keep: 0.2	0.3334 ($\pm$ 0.1877)	0.8000 ($\pm$ 0.1338)	0.0442 ($\pm$ 0.0151)
envs: 2, keep: 0.4	0.3872 ($\pm$ 0.2958)	0.8822 ($\pm$ 0.1836)	0.0710 ($\pm$ 0.0435)
envs: 2, keep: 0.6	0.5360 ($\pm$ 0.3341)	0.9425 ($\pm$ 0.1029)	0.0867 ($\pm$ 0.0573)
envs: 4, keep: 0.2	N/A	0.9862 ($\pm$ 0.0121)	N/A
envs: 4, keep: 0.4	N/A	0.9766 ($\pm$ 0.0375)	N/A
envs: 4, keep: 0.6	N/A	0.9941 ($\pm$ 0.0146)	N/A
envs: 8, keep: 0.2	N/A	0.9971 ($\pm$ 0.0029)	N/A
envs: 8, keep: 0.4	N/A	0.9693 ($\pm$ 0.0854)	N/A
envs: 8, keep: 0.6	N/A	0.9999 ($\pm$ 0.0002)	N/A
envs: 12, keep: 0.2	N/A	0.9995 ($\pm$ 0.0011)	N/A
envs: 12, keep: 0.4	N/A	0.9998 ($\pm$ 0.0004)	N/A
envs: 12, keep: 0.6	N/A	1.0000 ($\pm$ 0.0000)	N/A
envs: 16, keep: 0.2	N/A	0.9993 ($\pm$ 0.0016)	N/A
envs: 16, keep: 0.4	N/A	1.0000 ($\pm$ 0.0000)	N/A
envs: 16, keep: 0.6	N/A	1.0000 ($\pm$ 0.0000)	N/A

Table A.5: OOD Validation/Transition Accuracy (Rotate) - Across Configurations (mean $\pm$ standard deviation)

Configuration	Classic Comb	Sparse Comb	U-Net
envs: 1, keep: 0.2	0.7793 (± 0.1178)	0.7924 (± 0.0727)	0.3636 (± 0.1147)
envs: 1, keep: 0.4	0.8151 (± 0.0377)	0.8294 (± 0.0517)	0.2789 (± 0.1012)
envs: 1, keep: 0.6	0.8304 (± 0.0679)	0.8855 (± 0.0374)	0.3125 (± 0.1215)
envs: 2, keep: 0.2	0.8608 (± 0.0171)	0.8620 (± 0.0266)	0.3907 (± 0.0838)
envs: 2, keep: 0.4	0.8869 (± 0.0195)	0.8964 (± 0.0154)	0.4868 (± 0.1173)
envs: 2, keep: 0.6	0.9149 (± 0.0185)	0.9164 (± 0.0159)	0.6966 (± 0.0566)
envs: 4, keep: 0.2	0.8912 (± 0.0253)	0.8769 (± 0.0256)	0.5612 (± 0.0893)
envs: 4, keep: 0.4	0.9343 (± 0.0097)	0.9073 (± 0.0243)	0.7579 (± 0.1279)
envs: 4, keep: 0.6	0.9643 (± 0.0101)	0.9430 (± 0.0135)	0.9024 (± 0.0173)
envs: 8, keep: 0.2	0.9295 (± 0.0136)	0.9137 (± 0.0122)	0.7419 (± 0.0601)
envs: 8, keep: 0.4	0.9581 (± 0.0165)	0.9415 (± 0.0261)	0.8959 (± 0.0444)
envs: 8, keep: 0.6	0.9739 (± 0.0103)	0.9606 (± 0.0113)	0.9401 (± 0.0251)
envs: 12, keep: 0.2	0.9404 (± 0.0115)	0.9139 (± 0.0199)	0.8467 (± 0.0474)
envs: 12, keep: 0.4	0.9697 (± 0.0101)	0.9406 (± 0.0172)	0.9481 (± 0.0142)
envs: 12, keep: 0.6	0.9797 (± 0.0122)	0.9700 (± 0.0106)	0.9878 (± 0.0068)
envs: 16, keep: 0.2	0.9527 (± 0.0165)	0.9245 (± 0.0210)	0.8993 (± 0.0333)
envs: 16, keep: 0.4	0.9831 (± 0.0116)	0.9680 (± 0.0213)	0.9833 (± 0.0050)
envs: 16, keep: 0.6	0.9924 (± 0.0047)	0.9867 (± 0.0079)	0.9922 (± 0.0018)

Table A.6: OOD Validation/One Agent Samples Accuracy - Across Configurations (mean ± standard deviation)

Configuration	Classic Comb	Sparse Comb	U-Net
envs: 1, keep: 0.2	0.3923 (± 0.0667)	0.4184 (± 0.0734)	0.1366 (± 0.0421)
envs: 1, keep: 0.4	0.6819 (± 0.0281)	0.7079 (± 0.0568)	0.1627 (± 0.0767)
envs: 1, keep: 0.6	0.7382 (± 0.0696)	0.7932 (± 0.0431)	0.2396 (± 0.1017)
envs: 2, keep: 0.2	0.6173 (± 0.0433)	0.6683 (± 0.0511)	0.1689 (± 0.0364)
envs: 2, keep: 0.4	0.8139 (± 0.0130)	0.8264 (± 0.0115)	0.3434 (± 0.0906)
envs: 2, keep: 0.6	0.8587 (± 0.0145)	0.8698 (± 0.0196)	0.6638 (± 0.0629)
envs: 4, keep: 0.2	0.8023 (± 0.0198)	0.8019 (± 0.0286)	0.3525 (± 0.0822)
envs: 4, keep: 0.4	0.8805 (± 0.0041)	0.8595 (± 0.0278)	0.7324 (± 0.1522)
envs: 4, keep: 0.6	0.9268 (± 0.0074)	0.9093 (± 0.0154)	0.9003 (± 0.0182)
envs: 8, keep: 0.2	0.8627 (± 0.0083)	0.8528 (± 0.0191)	0.7015 (± 0.0747)
envs: 8, keep: 0.4	0.9217 (± 0.0132)	0.9078 (± 0.0269)	0.8950 (± 0.0453)
envs: 8, keep: 0.6	0.9536 (± 0.0085)	0.9409 (± 0.0126)	0.9398 (± 0.0254)
envs: 12, keep: 0.2	0.8827 (± 0.0054)	0.8633 (± 0.0242)	0.8402 (± 0.0523)
envs: 12, keep: 0.4	0.9480 (± 0.0082)	0.9216 (± 0.0199)	0.9479 (± 0.0143)
envs: 12, keep: 0.6	0.9680 (± 0.0110)	0.9602 (± 0.0111)	0.9876 (± 0.0068)
envs: 16, keep: 0.2	0.9165 (± 0.0121)	0.8887 (± 0.0262)	0.8968 (± 0.0366)
envs: 16, keep: 0.4	0.9703 (± 0.0109)	0.9571 (± 0.0238)	0.9829 (± 0.0050)
envs: 16, keep: 0.6	0.9855 (± 0.0063)	0.9815 (± 0.0097)	0.9919 (± 0.0018)

Table A.7: OOD Validation/Agent Accuracy - Across Configurations (mean ± standard deviation)

Configuration	Classic Comb	Sparse Comb	U-Net
envs: 1, keep: 0.2	0.0001 (± 0.0003)	0.0036 (± 0.0106)	0.0001 (± 0.0004)
envs: 1, keep: 0.4	0.0000 (± 0.0000)	0.0230 (± 0.0322)	0.0000 (± 0.0001)
envs: 1, keep: 0.6	0.0000 (± 0.0000)	0.0579 (± 0.0960)	0.0001 (± 0.0001)
envs: 2, keep: 0.2	0.3805 (± 0.1937)	0.9334 (± 0.1163)	0.2375 (± 0.0672)
envs: 2, keep: 0.4	0.3788 (± 0.3091)	0.8836 (± 0.1542)	0.1987 (± 0.1063)
envs: 2, keep: 0.6	0.5299 (± 0.3377)	0.9478 (± 0.0989)	0.1576 (± 0.0701)
envs: 4, keep: 0.2	0.8358 (± 0.1399)	0.9989 (± 0.0020)	0.6718 (± 0.1055)
envs: 4, keep: 0.4	0.8134 (± 0.1860)	0.9957 (± 0.0110)	0.7099 (± 0.1102)
envs: 4, keep: 0.6	0.8592 (± 0.1829)	0.9960 (± 0.0106)	0.7860 (± 0.1131)
envs: 8, keep: 0.2	0.9021 (± 0.1315)	0.9995 (± 0.0010)	0.9190 (± 0.0449)
envs: 8, keep: 0.4	0.8846 (± 0.1251)	0.9773 (± 0.0676)	0.9398 (± 0.0403)
envs: 8, keep: 0.6	0.9045 (± 0.1237)	0.9999 (± 0.0002)	0.9661 (± 0.0287)
envs: 12, keep: 0.2	0.8711 (± 0.1249)	1.0000 (± 0.0000)	0.9653 (± 0.0300)
envs: 12, keep: 0.4	0.8823 (± 0.1430)	1.0000 (± 0.0000)	0.9753 (± 0.0223)
envs: 12, keep: 0.6	0.9263 (± 0.1140)	1.0000 (± 0.0000)	0.9903 (± 0.0087)
envs: 16, keep: 0.2	1.0000 (± 0.0000)	1.0000 (± 0.0000)	0.9951 (± 0.0039)
envs: 16, keep: 0.4	1.0000 (± 0.0000)	1.0000 (± 0.0000)	0.9995 (± 0.0005)
envs: 16, keep: 0.6	1.0000 (± 0.0000)	1.0000 (± 0.0000)	0.9997 (± 0.0004)

Table A.8: OOD Validation/Object Accuracy - Across Configurations (mean ± standard deviation)

Configuration	Classic Comb	Sparse Comb	U-Net
envs: 1, keep: 0.2	0.0000 (± 0.0000)	0.0000 (± 0.0000)	0.0000 (± 0.0000)
envs: 1, keep: 0.4	0.0000 (± 0.0000)	0.0000 (± 0.0000)	0.0000 (± 0.0000)
envs: 1, keep: 0.6	0.0000 (± 0.0000)	0.0000 (± 0.0000)	0.0000 (± 0.0000)
envs: 2, keep: 0.2	0.0000 (± 0.0000)	0.5818 (± 0.2307)	0.0000 (± 0.0000)
envs: 2, keep: 0.4	0.0000 (± 0.0000)	0.5727 (± 0.2508)	0.0000 (± 0.0000)
envs: 2, keep: 0.6	0.2727 (± 0.0000)	0.6545 (± 0.2182)	0.1818 (± 0.0704)
envs: 4, keep: 0.2	0.5750 (± 0.0250)	0.4833 (± 0.2034)	0.0250 (± 0.0382)
envs: 4, keep: 0.4	0.5833 (± 0.0000)	0.4333 (± 0.2380)	0.0333 (± 0.0408)
envs: 4, keep: 0.6	0.5833 (± 0.0000)	0.5833 (± 0.0000)	0.0167 (± 0.0333)
envs: 8, keep: 0.2	0.5000 (± 0.0000)	0.4667 (± 0.1000)	0.2250 (± 0.1493)
envs: 8, keep: 0.4	0.9833 (± 0.0500)	0.9833 (± 0.0333)	0.4583 (± 0.0932)
envs: 8, keep: 0.6	0.9833 (± 0.0500)	0.9417 (± 0.1493)	0.3750 (± 0.1193)
envs: 12, keep: 0.2	0.8769 (± 0.1584)	0.8846 (± 0.1796)	0.3385 (± 0.1727)
envs: 12, keep: 0.4	1.0000 (± 0.0000)	0.8769 (± 0.3003)	0.4385 (± 0.2177)
envs: 12, keep: 0.6	1.0000 (± 0.0000)	0.8385 (± 0.2077)	0.6000 (± 0.3187)
envs: 16, keep: 0.2	0.9500 (± 0.1247)	0.7833 (± 0.2640)	0.2417 (± 0.1417)
envs: 16, keep: 0.4	1.0000 (± 0.0000)	0.9667 (± 0.1000)	0.6417 (± 0.2765)
envs: 16, keep: 0.6	1.0000 (± 0.0000)	1.0000 (± 0.0000)	0.8333 (± 0.1394)

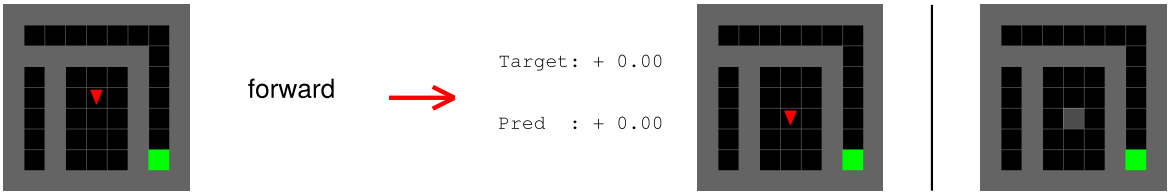
Table A.9: OOD Validation/Reward Positive Accuracy - Across Configurations (mean ± standard deviation)

Configuration	Classic Sep	Sparse Sep
envs: 4, keep: 0.2	0.6053 ($\pm$ 0.1717)	0.7500 ($\pm$ 0.0654)
envs: 4, keep: 0.4	0.7307 ($\pm$ 0.1385)	0.8596 ($\pm$ 0.0262)
envs: 4, keep: 0.6	0.7493 ($\pm$ 0.1372)	0.9144 ($\pm$ 0.0078)
envs: 8, keep: 0.2	0.8273 ($\pm$ 0.0856)	0.8363 ($\pm$ 0.0306)
envs: 8, keep: 0.4	0.8967 ($\pm$ 0.0696)	0.9042 ($\pm$ 0.0096)
envs: 8, keep: 0.6	0.9188 ($\pm$ 0.0826)	0.9268 ($\pm$ 0.0251)
envs: 12, keep: 0.2	0.8876 ($\pm$ 0.0204)	0.8781 ($\pm$ 0.0135)
envs: 12, keep: 0.4	0.9513 ($\pm$ 0.0082)	0.9149 ($\pm$ 0.0389)
envs: 12, keep: 0.6	0.9161 ($\pm$ 0.1108)	0.9614 ($\pm$ 0.0116)
envs: 16, keep: 0.2	0.9276 ($\pm$ 0.0115)	0.9001 ($\pm$ 0.0178)
envs: 16, keep: 0.4	0.9718 ($\pm$ 0.0086)	0.9499 ($\pm$ 0.0299)
envs: 16, keep: 0.6	0.9872 ($\pm$ 0.0038)	0.9810 ($\pm$ 0.0086)

Table A.10: Comparison of OOD Validation Accuracy of the SepAction Transformer Variant. (mean $\pm$ standard deviation)

A.4 Eta Attention Maps

This appendix section provides real examples of the eta attention maps created by the sparse transformer. This is only qualitative evidence, but we tried to choose examples that are representative of the behavior of the model. Each figure has two parts, the top which gives information about the input/output of the model, and the bottom which shows the η attention map. The top part has, from left to right, the input state, the input action, an arrow (green indicating that the prediction was correct, red indicating that the prediction was incorrect), information about the reward (target, prediction), the expected next-state, and the predicted next-state. The attention maps follow the same structure as the illustrative examples in figure 5.5 and are explained in section 5.6.1. We only show samples where the action was to move forward, as incorrect rotations were very infrequent and thus not representative of the model's behavior.



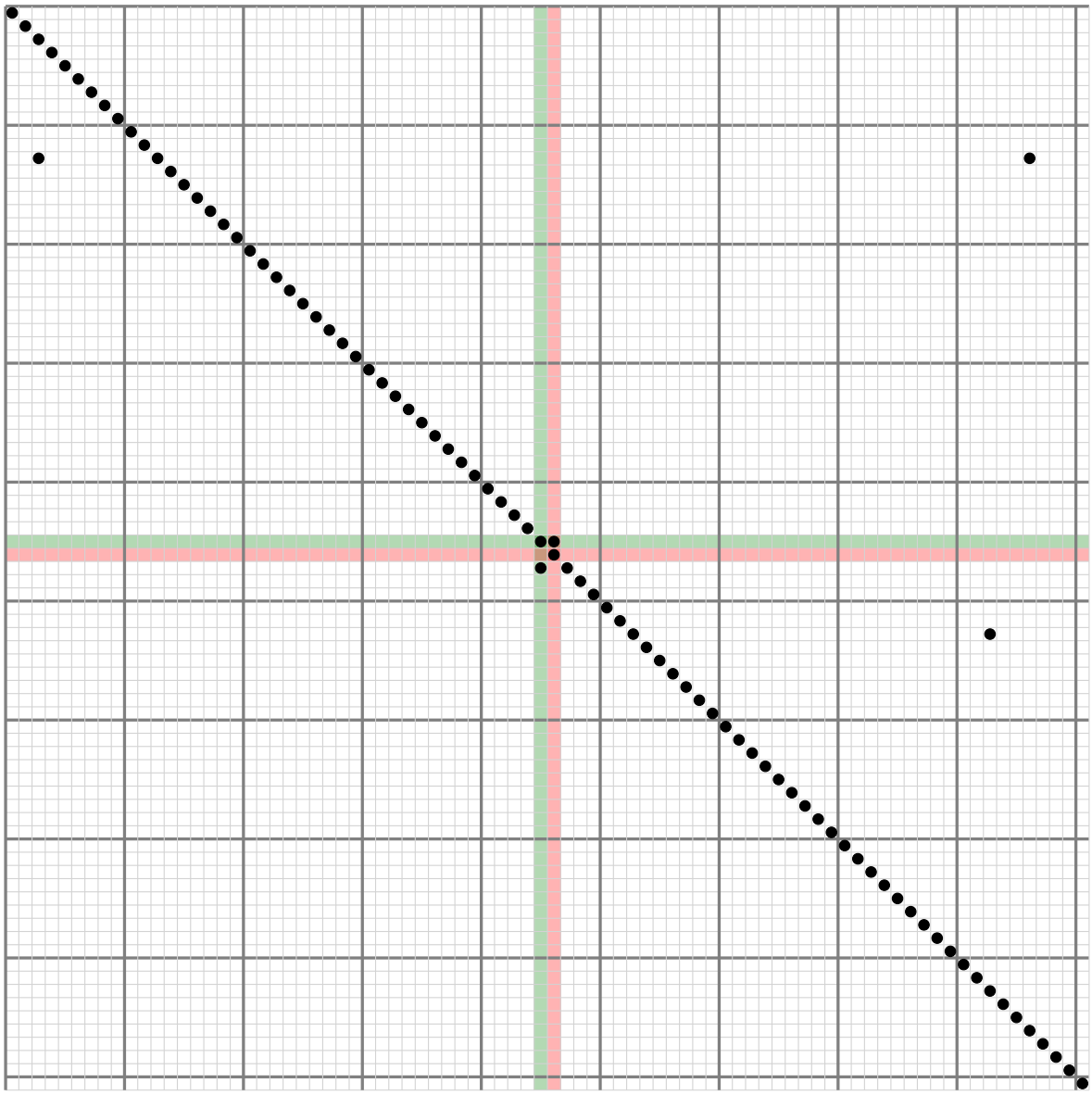
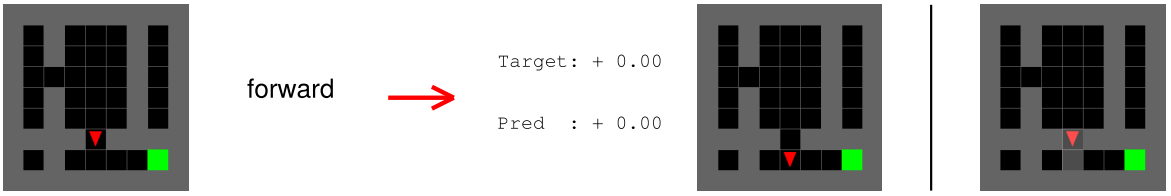


Figure A.11: η attention map example of a sample where the agent disappeared. The cell with the agent correctly received information about the cell in front of the agent (empty), thus removing the agent from that cell. The cell in front of the agent did not, however, receive information about the agent, which is why this cell was not updated to add an agent to it.



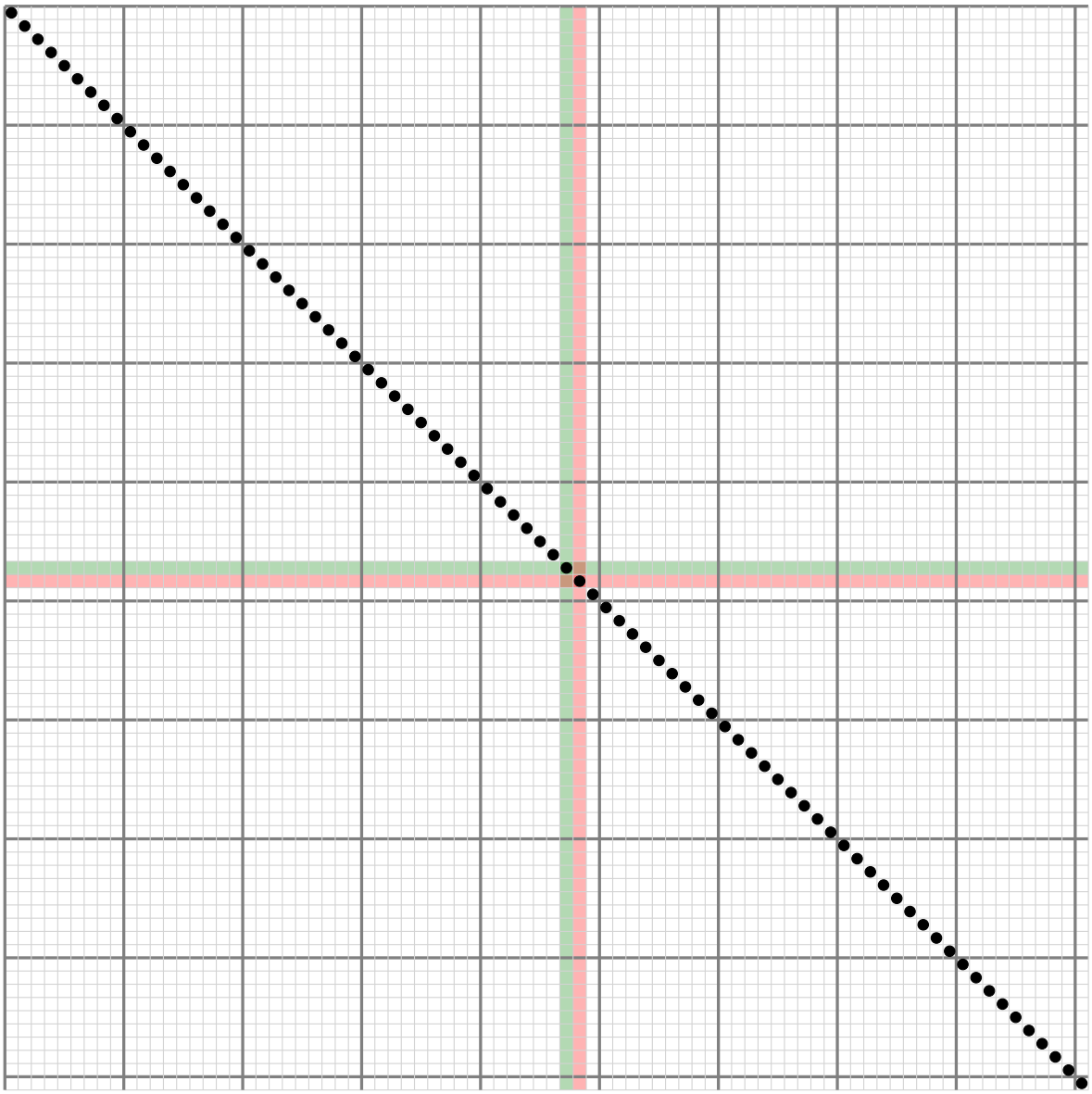
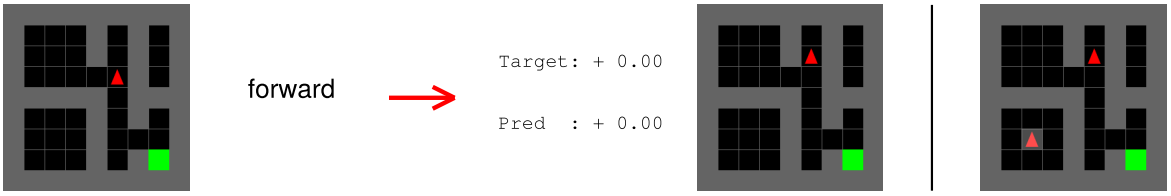


Figure A.12: η attention map example of a sample where no information was passed to any cell. This would have been the correct behavior if the action was to rotate, but the action was to move forward. Thus, no cells were updated, and the agent did not move. Most likely, the model overfit to the training data and learned on that position it can never move forward, thus not 'requesting' any additional information.



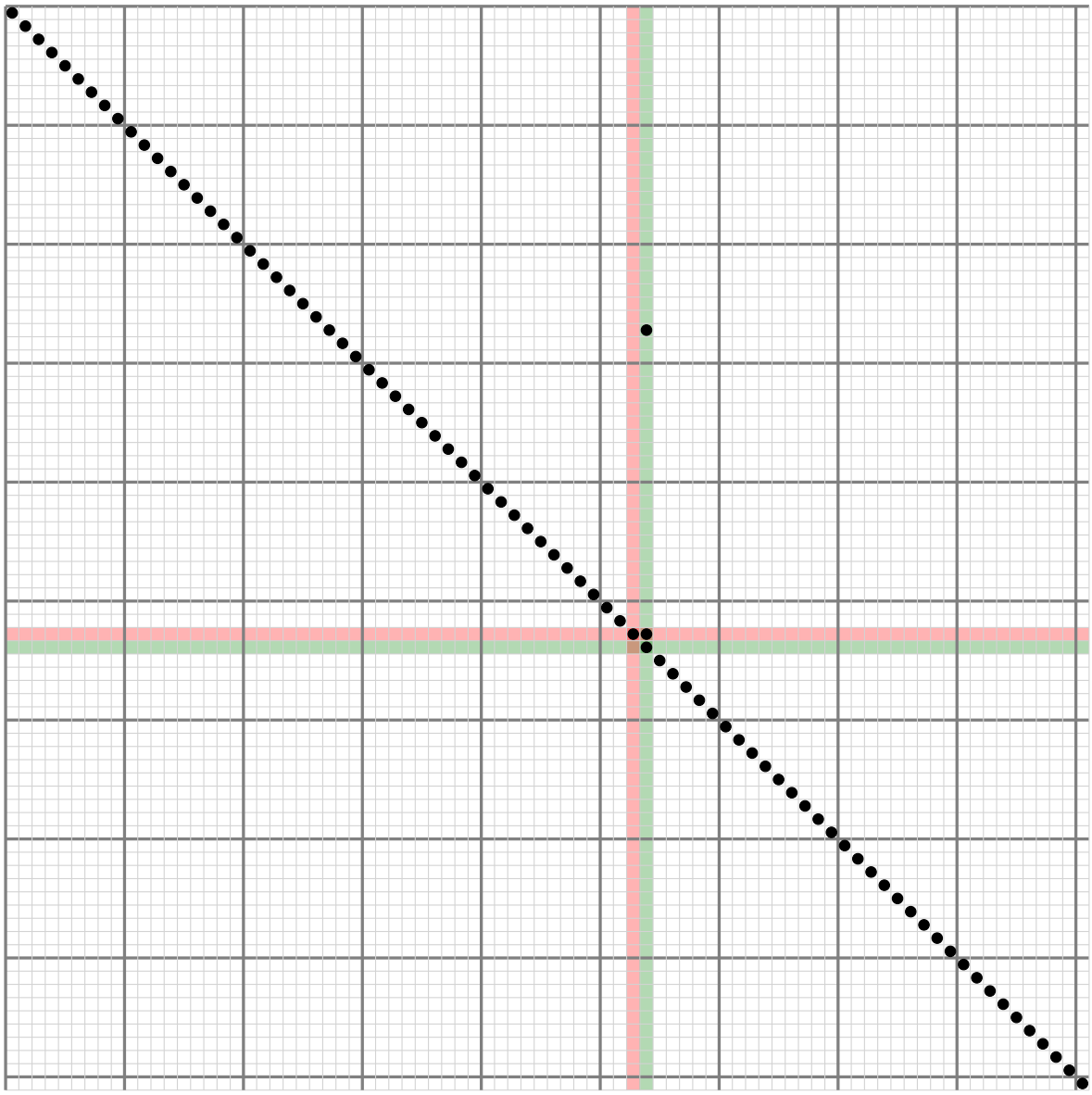


Figure A.13: η attention map example of a sample where a random agent was added to the grid. Even though the movement of the original agent was correct, the model passed information about the agents' location to another cell, thus inducing a spurious agent in the grid.

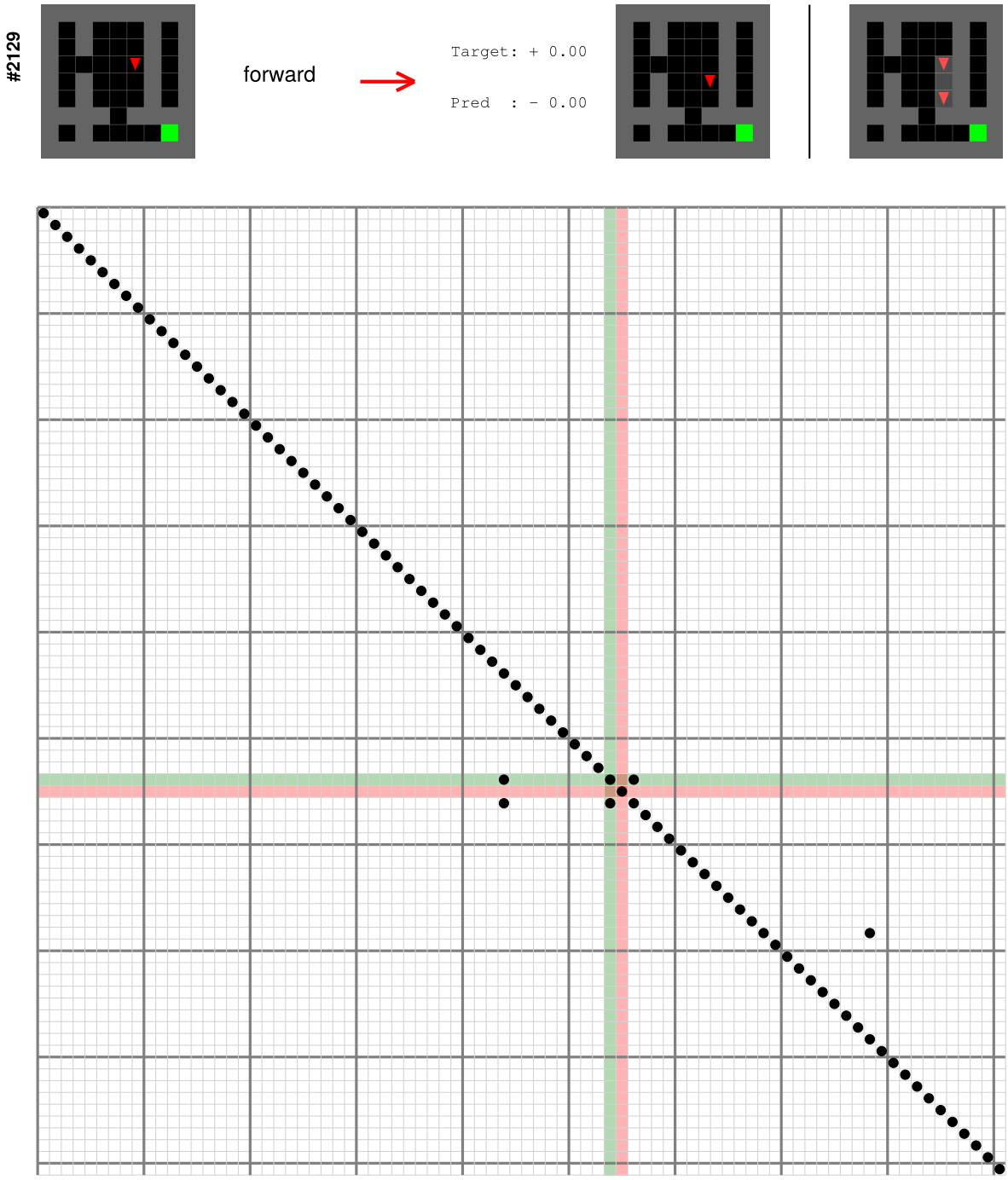


Figure A.14: η attention map example of a sample where the agent moved forward, but the cell in front of the agent was skipped, not only in the final prediction but also in the attention maps.