Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik

Bachelorarbeit Informatik

GazeControl: Computational modeling of human vision and decision making for visual reasoning tasks

Florentin Doll

17. November 2025

Gutachter

Prof. Dr. Martin Butz
Kognitive Modellierung
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Betreuer

Tomáš Daniš
Kognitive Modellierung
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Doll, Florentin:
GazeControl: Computational modeling of human vision and decision making for visual reasoning tasks
Bachelorarbeit Informatik
Eberhard Karls Universität Tübingen
Bearbeitungszeitraum: 17.07.2025 – 17.11.2025

Abstract

Humans perceive the visual world through a process of selective attention, allowing them to focus high resolution vision on relevant parts of a scene and integrate that information across time. This thesis aims to model this process of visual attention with artificial agents, enabling them to efficiently process visual information in complex environments. We propose a novel architecture that combines a recurrent visual attention mechanism with reinforcement learning to allow agents to learn where to look in order to maximize task performance. An auxiliary decoder helps to stabilize latent memory representations and enables interpretability of the current memory state. We evaluate the proposed architecture on a set of challenging visual reasoning tasks, demonstrating its ability to learn effective attention policies. The aim of this thesis is to lay the groundwork for future research in active vision and reinforcement learning, providing a modular and extensible framework for exploring these ideas further.

Contents

Chapter 1

Introduction

Human vision is inherently selective. Rather than processing the entire visual field uniformly, the human eye relies on a foveated view, where high-resolution vision is concentrated in a small central region (the fovea), while the surrounding peripheral vision is of lower resolution. Humans sequentially direct their gaze towards interesting or task-relevant regions of a scene, integrating information over time to form a coherent understanding of their environment. This mechanism of selective attention allows humans to efficiently process complex visual scenes under limited computational resources.

In contrast, most artificial vision systems process visual inputs in a uniform manner, requiring high computational resources to achieve comparable performance to human vision. They also usually only use a single feedforward pass to process an image, leading to limited interpretability and adaptability to changing environments.

Research in active vision and attention-based reinforcement learning has begun to address these limitations. Models such as the Recurrent Attention Model (RAM) (Mnih *et al.*, 2014) and its variants have demonstrated the potential of foveated vision and sequential attention mechanisms in artificial agents. In addition to these attention mechanisms, auxiliary tasks such as image reconstruction have been shown to improve the stability and performance of reinforcement learning agents (Jaderberg *et al.*, 2016).

In this thesis, we propose a novel architecture that combines a recurrent visual attention mechanism with an auxiliary reconstruction task to enable artificial agents to efficiently process visual information. Our agent will be trained to learn two policies, a decision making policy that classifies the input based on the glimpses seen so far, and a sensory policy that decides where to look next. This sensory policy will move the gaze to points of interest on the input image, integrating information over time to enable the decision making policy to make accurate classifications. We want to combine the benefits of active vision with the stabilizing effects of auxiliary tasks to create a robust and interpretable model for visual reasoning tasks.

This work will be a foundation that can be built upon and improved in the future research and adapted to more difficult tasks, such as dynamic environments or real-world applications.

Chapter 2

Foundations

This chapter provides some foundational background information on the machine learning concepts and techniques that are used throughout this thesis.

2.1 Multi Layer Perceptrons (MLPs)

Those are the most basic types of neural networks, consisting of multiple fully connected layers of neurons. Each neuron applies a linear transformation followed by a non-linear activation function to its inputs. By stacking multiple layers of these neurons, MLPs are able to learn complex non-linear functions, making them able to perform a wide variety of tasks. An equation for an MLP with L layers, so L-1 hidden layers + one output layer, can be written as follows:

$$\begin{aligned} h^{(0)} &= x \\ h^{(l)} &= f^{(l)}(W^{(l)}h^{(l-1)} + b^{(l)}) \quad \text{for } l = 1, \ldots, L-1 \\ y &= W^{(L)}h^{(L-1)} + b^{(L)} \end{aligned}$$

Where:

- x is the input vector.
- $h^{(l)}$ is the output of layer l.
- $W^{(l)}$ and $b^{(l)}$ are the weight matrix and bias vector for layer l.
- $f^{(l)}$ is the activation function for layer l (e.g., ReLU, Sigmoid, Tanh).
- y is the final output vector.

2.2 Convolutional Neural Networks (CNNs)

CNNs (Krizhevsky *et al.*, 2012) are a type of neural network that are particularly well suited for processing grid-like data such as images. CNNs have been widely used in computer vision tasks such as image classification, object detection, and

segmentation. They use convolutional layers that apply a set of learnable filters (kernels) to the input data, allowing them to capture spatial hierarchies and patterns. Each of these filters slides over the input image, performing element-wise multiplications and summing the results to produce feature maps of, for example, edges, textures, or shapes. These feature maps are then passed through non-linear activation functions and pooling layers to reduce dimensionality and retain important information.

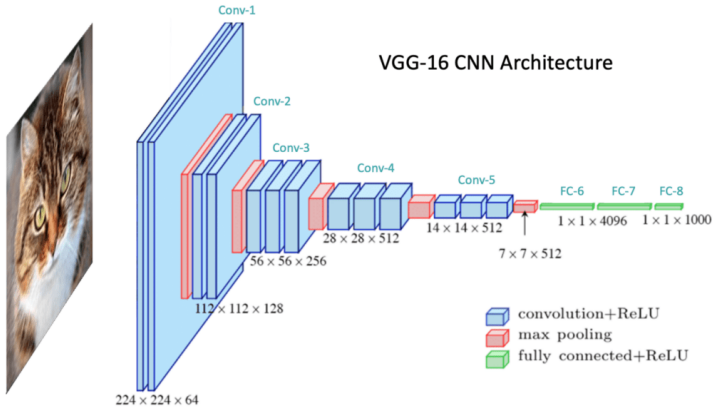


Figure 2.1: Convolutional Neural Network example architecture for VGG16 (reproduced from https://learnopencv.com/understanding-convolutional-neural-networks-cnn)

2.3 Recurrent Neural Networks (RNNs)

RNNs are a type of neural network designed to handle sequential data by maintaining a hidden state that captures information from previous time steps. This hidden state is updated at each time step based on the current input and the previous hidden state. Through that mechanism, RNNs are able to model temporal dependencies in data. LSTMs (Long Short-Term Memory) (Hochreiter and Schmidhuber, 1997) are a popular variant of RNNs that we will use in this thesis. LSTMs consist of a cell state and three gates (input, forget, and output gates) that regulate the flow of information, allowing them to capture long-term dependencies more effectively than standard RNNs. The equations for an LSTM cell can be summarized

as follows:

$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
\tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
h_t &= o_t * \tanh(C_t)
\end{aligned}
$$

Where:

- x_t is the input at time step t.
- h_{t-1} is the hidden state from the previous time step.
- C_{t-1} is the cell state from the previous time step.
- f_t, i_t, and o_t are the forget (later introduced(Gers *et al.*, 2000)), input, and output gates, respectively.
- $\tilde{C}_t$ is the candidate cell state.
- C_t is the updated cell state.
- h_t is the updated hidden state.
- W_f, W_i, W_C, and W_o are weight matrices.
- b_f, b_i, b_C, and b_o are bias vectors.
- σ is the sigmoid activation function.
- $*$ denotes element-wise multiplication.

2.4 Reinforcement Learning (RL)

Reinforcement Learning is a type of machine learning where an agent learns to make decisions by interacting with an environment. The agent receives observations from the environment and takes actions based on those observations. In contrast to all the other mentioned techniques, RL does not rely on labeled data, but instead learns from feedback in the form of rewards, since for many tasks, there is no clear ground truth available. The agent will get rewarded based on the outcome of its actions, and the goal of the agent is to learn a policy that maximizes the cumulative reward

over time. This reward could, for example, be a +1 for successfully completing a task and a 0 for failing it.

For this thesis, we will use Generalized Advantage Estimation (GAE) (Schulman *et al.*, 2018) to train the agent. In short, GAE estimates how much better an action was than the average by smoothly combining short- and long-term reward signals, reducing noise and making learning more stable. Simplified, the advantage function $A(v, r)$ for one run is computed as follows:

$$\begin{aligned} for t = 0 \dots T-1: & \\ \delta &= r(t) + \gamma V(t+1) - V(s_t) \\ gae &= \delta + \gamma \lambda gae \\ A(t) &= gae \end{aligned}$$

Where:

- $r(t)$ is the reward received at time step t.
- $V(s_t)$ is the estimated value of state s_t.
- γ is the discount factor, determining the importance of future rewards.
- λ is a smoothing parameter that balances bias and variance in the advantage estimates.

We combine this with Advantage Actor-Critic (A2C) (Mnih *et al.*, 2016), which is a policy gradient method that uses two neural networks: an actor network that learns the policy (mapping from states to actions) and a critic network that estimates the value function (expected cumulative reward from a state). For an update we then compute the losses as follows:

$$\begin{aligned} L_{policy} &= -mean(p_{\log} A(t)) \\ L_{value} &= MSE(V(s_t), R_t) \\ L_{entropy} &= -mean(entropies) \\ L_{total} &= L_{policy} + c_1 L_{value} - c_2 L_{entropy} \end{aligned}$$

Where:

- $p_{\log}$ is the log probability of the taken actions.
- $A(t)$ is the advantage estimate from GAE.
- $V(s_t)$ is the estimated value of state s_t.
- R_t is the actual return (cumulative reward) from time step t.
- *entropies* is the entropy of the policy distribution, encouraging exploration.
- c_1 and c_2 are coefficients that balance the contributions of the value loss and entropy bonus.

2.5 Autoencoders

Autoencoders (Hinton and Salakhutdinov, 2006) are a type of neural network used for unsupervised learning of efficient encodings. They consist of two main components: an encoder that compresses the input data into a lower-dimensional representation (latent space), and a decoder that reconstructs the original data from this compressed representation. Autoencoders are commonly used for dimensionality reduction, feature learning, and data denoising.

2.6 Positional Encoding

Positional encoding is a technique used to inject information about the position of something into a model in a way that the model can understand. The general idea is to transform the position into a higher-dimensional space using a set of basis functions. This will allow the model to learn to interpret the position information more effectively. We will use sinusoidal positional encoding (Vaswani *et al.*, 2023) in this thesis, which uses sine and cosine functions of different frequencies to encode the position.

Chapter 3

Methodology

3.1 Architecture

In this chapter, we describe the proposed architecture for modeling visual attention in artificial agents. Since the aim of the architecture is to mimic human visual attention, we need to incorporate several key components:

- **Encoder:** A visual input mechanism that allows the agent to take in information from the image.
- **Memory:** A memory module that enables the agent to maintain a memory of past visual inputs and integrate that information over time.
- **Agent:** A reinforcement learning framework that allows the agent to learn from its interactions with the environment and improve its attention and decision-making strategy over time.
- **Decoder:** As a final component, we introduce an auxiliary decoder that helps to stabilize the learning process and improve the interpretability of the agent's internal representations.

Together, they enable the agent to reason about where to look in a visual scene and how to use that information to perform tasks effectively in a similar way to humans. The overall architecture is illustrated in Figure 3.1.

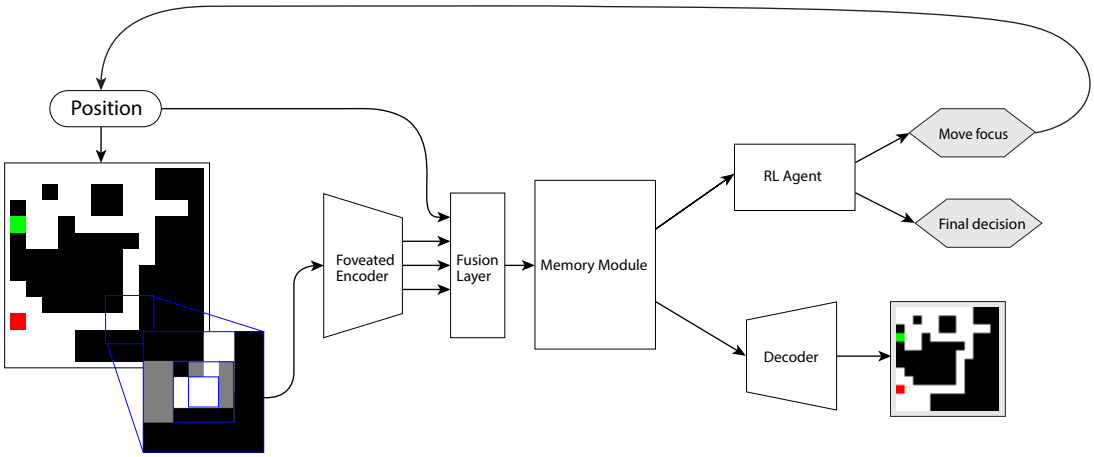


Figure 3.1: Network architecture diagram illustrating the key components of the proposed model.

3.1.1 Encoder

The encoder is responsible for processing the visual input and extracting relevant features. For the purpose of our architecture, we use a convolutional neural network (CNN) 2.2 as the encoder and keep that one relatively small, since it will only process small cutouts of the image. We looked into and tested different ways of implementing foveated vision, such as multi-scale image patches, Gaussian blurring, or specific downsampling techniques. In the end, we decided to go with a simple multi-scale patch extraction approach, similar to the one used in the Recurrent Attention Model (RAM) (Mnih *et al.*, 2014). That means we take three square patches of increasing size (e.g., 4×4, 8×8, 16×16, which are the values used for the maze model) around the current gaze location, downsample them to the smallest size (e.g., 4×4) and pass all of those patches separately through the same CNN encoder. Using this method allows us to simulate foveation while keeping the encoder architecture and compute requirements simple and efficient.

3.1.2 Fusion Layer

After extracting features from the different patches, we need to combine them into a single feature representation. For that, we use a simple MLP 2.1 that takes the concatenated features, as well as gaze coordinates as an input and produces a fixed-size feature vector as output. These gaze coordinates get encoded before being passed into the fusion layer using sinusoidal positional encoding. We use it to preprocess the information into a vector that is easier to interpret and work with for the memory module. This idea originates from the Glimpse Network (Mnih *et al.*, 2014).

3.1.3 Memory Module

The memory module is a crucial component of the architecture, as it allows the agent to, step by step, build an understanding of the environment. For this, we use a standard long short-term memory (LSTM) 2.3 network as the memory module. The LSTM takes the fused feature vector from the encoder as input at each time step and updates its internal state accordingly. The hidden state of the LSTM serves as the agent's memory representation, which is used to feed into both the agent and the auxiliary decoder.

3.1.4 Agent

The agent is an actor-critic based reinforcement learning framework that consists of three policy heads and one value head:

- The first policy head outputs logits over the 8 possible discrete move actions (Categorical).
- The second policy head outputs a Bernoulli logit for the stop action to terminate the episode.
- The third policy head outputs a binary classification logit for the final decision.
- The value head predicts $V(s_t)$ for advantage estimation.

All of these heads share a common fully connected layer that processes the LSTM hidden state, combined with the current gaze location, before feeding it into the individual heads.

3.1.5 Decoder

The auxiliary decoder, which is also a CNN 2.2 is designed to reconstruct the visual input based on the agent's memory representation. The purpose of the

decoder is twofold: first, it encourages the memory module to learn meaningful and stable representations of the visual input. Training the encoder, fusion layer, and memory together with just the reinforcement learning signal from the agent would lead to slower training and poorer performance, as shown in section 4.3.1. The reconstruction loss from the decoder provides an additional training signal that helps to stabilize the learning process and gives the latent space more structured representations and gradients. Second, the decoder allows us to interpret the agent's internal memory state by visualizing what the agent 'remembers' about the visual input at each time step. Do note that the decoder and agent, while being trained jointly, do not share any weights or parameters or loss signals. Therefore, their interpretation of the latent space might be different. However, both components ultimately use the same memory representations and make decisions based on them, so even though they might have different perspectives, the decoder still gives us valuable insights into the agent's understanding of the visual input.

3.2 Losses

The overall loss function for training the model consists of two main components: The reinforcement learning loss from the agent and the reconstruction loss from the auxiliary decoder. Both of these losses train the encoder, fusion layer, and memory module jointly, while the agent only trains with the reinforcement learning loss and the decoder only trains with the reconstruction loss. The total loss can be expressed as:

$$\mathcal{L}_{total} = \lambda_{RL}\mathcal{L}_{RL} + \mathcal{L}_{rec} \tag{3.1}$$

Where $\mathcal{L}_{RL}$ is the reinforcement learning loss and $\mathcal{L}_{rec}$ is the reconstruction loss. We only scale the reinforcement learning loss since the reconstruction loss is scaled by its individual components.

3.2.1 Reinforcement Learning Loss

For the reinforcement loss, we use GAE (Generalized Advantage Estimation) to compute the policy gradient and update the agent's parameters. In the beginning, we used A2C (Advantage Actor-Critic) as the RL algorithm. We also tried to use PPO (Proximal Policy Optimization) (Schulman *et al.*, 2017), however we couldn't make it work well with our architecture and training setup.

We use standard reinforcement learning techniques for the training, as explained in section 2.4, except for the way we handle the stop action during training. Since we have many components, the early episodes are quite complex to learn, so we have to be careful of early stopping. Therefore, we punish stopping early with a high penalty if the agent is either not confident about its decision or came to the wrong conclusion and stopped early. In practice, that seemed to still not be enough

since even though we couldn't observe early stopping in the early episodes, forcing the model to take at least a certain number of steps for the first half of the training seemed to help a lot with stability and performance.

3.2.2 Reconstruction Loss

The reconstruction loss is primarily based on the L1 loss between the original input image and the reconstructed image produced by the decoder. However, since the agent only sees parts of the image, we only compute the reconstruction loss over the areas that the agent has actually observed through its gaze, since the full reconstruction isn't actually our goal. The goal of this loss is to stabilize the latent representations in the memory module, since we can assume that if we can reconstruct the seen parts of the image well, the latent representation must contain useful information about those parts. We don't care about the unseen parts of the image, since the agent has no information about them anyway. For more complicated to reconstruct datasets we used a combination of multiple losses on top of the L1 loss.

- **Perceptual Loss:** This loss is based on a pretrained VGG19 network (Pihlgren *et al.*, 2024) and helps to capture high-level features in the reconstructed images and therefore helps shape the latent space. It computes the L2 loss between the feature maps of the original and reconstructed images at different layers of the VGG19 network.
- **SSIM Loss:** SSIM (Wang *et al.*, 2004) evaluates similarity by measuring how well the luminance, contrast, and structural relationships between local image patches are preserved. Using SSIM as a loss encourages the model to maintain edges, shapes, and fine structural details that are important for human perception.
- **Gradient Discrepancy Loss:** The GDL loss (Mathieu *et al.*, 2016) focuses on matching the gradients between pixels of the original and reconstructed images, which helps to preserve edges and fine details in the reconstructions.

The final reconstruction loss is a weighted combination of all the mentioned losses at the last step of a reconstruction task, as well as a masked L1 loss at each step to actively encourage step-by-step reconstruction:

$$\mathcal{L}_{rec} = \lambda_{L1}^{steps}\mathcal{L}_{L1}^{steps} + \lambda_{L1}^{final}\mathcal{L}_{L1}^{final} + \lambda_{perc}\mathcal{L}_{perc} + \lambda_{ssim}\mathcal{L}_{ssim} + \lambda_{gdl}\mathcal{L}_{gdl} \tag{3.2}$$

3.3 Pretraining

Since training all of the components at once from scratch is quite difficult and unstable, we use a two-stage training process. In the first stage, we pretrain the

decoder alone, using an autoencoder 2.5 setup. Through this, we give the decoder an expected latent space 'structure' that the memory module and encoder can learn to adjust to. That structure will get overwritten or changed in the second training stage, but having a good starting point helps get nice gradients from the reconstruction loss and therefore helps avoid local minima. This is crucial since local minima are a big problem for complex architectures trained with reinforcement learning (Buşoniu *et al.*, 2018).

3.4 Datasets

Throughout this thesis, we considered three main datasets for evaluating our architecture:

3.4.1 Compositional Visual Reasoning (CVR)

We initially planned to use the Compositional Visual Reasoning (CVR) dataset (Zerroug *et al.*, 2022) for evaluating our architecture. The CVR dataset consists of synthetic images, where there are always four images arranged in a 2×2 grid, containing different objects and shapes. The task is to find the odd one out based on a set of compositional rules. While this dataset is interesting and challenging, we figured other datasets would be more suitable for our specific architecture and research goals.

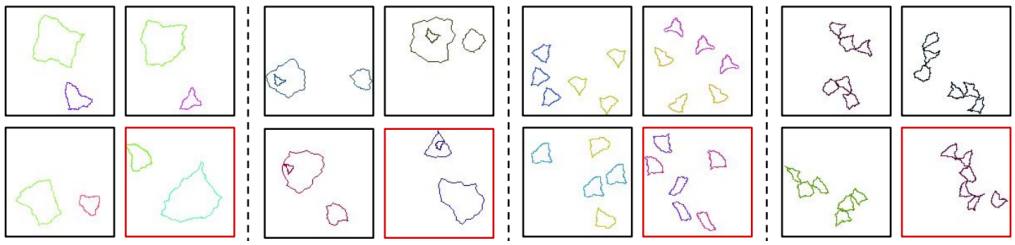
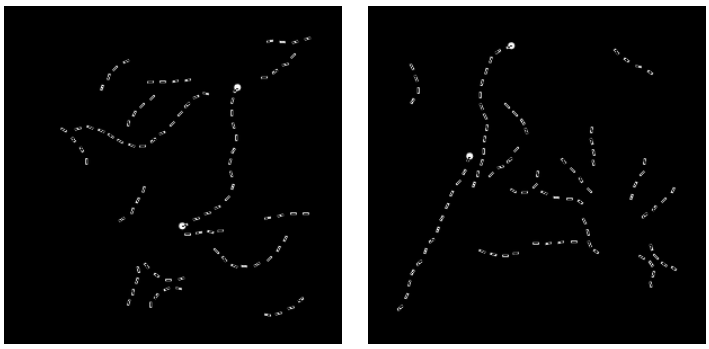


Figure 3.2: Example from the CVR dataset (Reproduced from (Zerroug *et al.*, 2022))

3.4.2 Pathfinder

The Pathfinder dataset (Tay *et al.*, 2020) is another dataset we considered for evaluating our architecture. It consists of images with two dots and a bunch of dashed lines, where the task is to determine whether there is a continuous path connecting the two dots. This dataset is very interesting for our architecture, since it requires long-range visual reasoning and attention to detail. However, it turned out that the very thin lines with most of the image being just background made it very difficult to reconstruct. The model has to train the encoder, learn how

to combine the information in the fusion layer, then figure out how to store that information in the memory module, and finally, the decoder has to reconstruct very thin lines based on that information. Learning all of that is quite complex, so if the training signal from the reconstructions is as weak as it is for this dataset, the model struggles to learn properly and ends up in a fully black local minima.



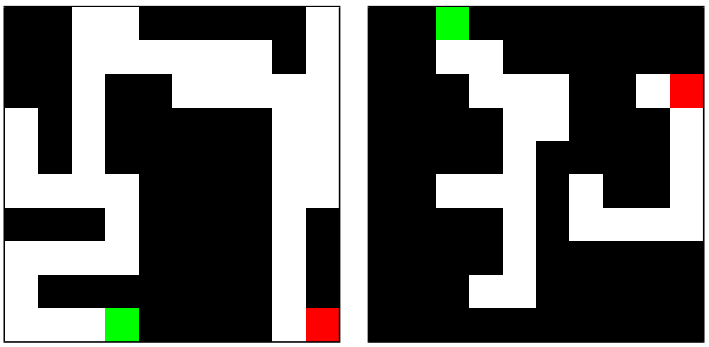
connected disconnected

Figure 3.3: Pathfinder examples (Generated using (drewlinsley, 2019))

3.4.3 Maze Dataset

The Maze dataset is a new synthetic dataset we created, which consists of simple mazes with a clear path from start to end. The task is for the agent to navigate from the start to the end of the maze, while classifying whether such a path exists or not. Connected mazes carve a main path from start to end using depth-first search and then add random branches to increase complexity.

Mazes are built in a grid, where each cell has 4 pixels and is either a wall, a free space, or a start/finish. These grids can be of any size, e.g., 10×10 cells (40×40 pixels) or 15×15 cells (60×60 pixels). We also have options to get specific path lengths, and can therefore control the complexity of the mazes. For our experiments, we mainly used a set of 100k 10×10 mazes with 3k validation and 3k test images.



connected disconnected

Figure 3.4: Maze examples (10×10 Grid)

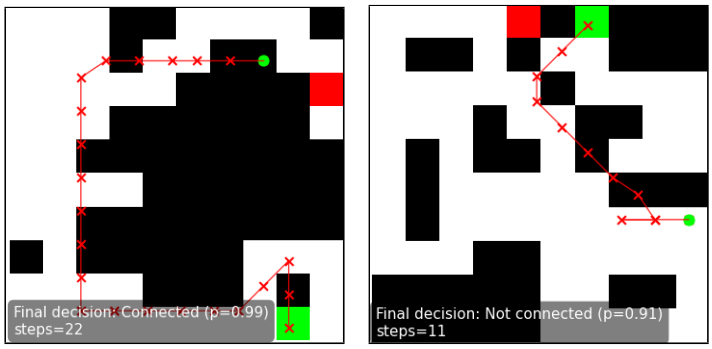
Chapter 4

Results

In this section, we report results for the proposed gaze-control model. On the Maze task, we achieved promising overall performance. We can observe nice exploration behavior of the agent and fairly good generalization to unseen mazes.

4.1 Model Performance

With our current model, we are able to achieve an accuracy of around 96.5% on the validation set, which consists of 3000 randomly generated mazes.



connected disconnected

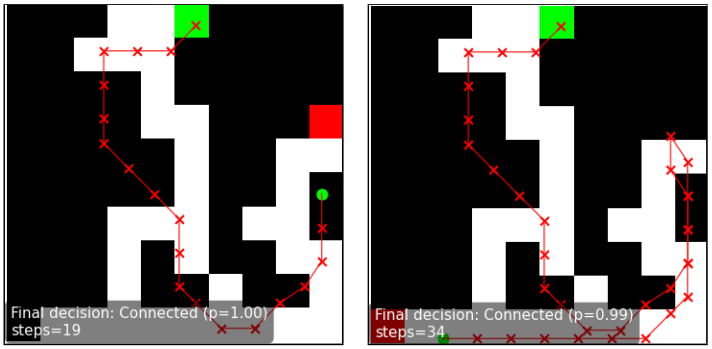
Figure 4.1: Randomly picked Validation Examples

These are some randomly picked validation examples of the maze task in Figure 4.1. As we can see, the agent is able to successfully classify both the connected and disconnected mazes. We can observe quite nice behavior of the agent, especially in the disconnected example, where the agent actually sees the goal in the very first glimpse and then goes on to try and find a path to it. Only after exploring all of the walls surrounding it and not finding a path to the goal, does it correctly classify the maze as disconnected.

Also, just to show how the agent perceives the maze, we can look at Figure 4.2, which shows the last glimpse of the connected example in Figure 4.1. As we can see, the agent doesn't get a lot of information on each step and has to rely on its memory to be able to successfully navigate the maze.

Figure 4.2: Glimpse at the last step of the exploration of the connected example in Figure 4.1

4.1.1 Exploration Behavior



original modified

Figure 4.3: Showcase of exploration behavior of the agent

In Figure 4.3, we took one of the validation mazes, where we saw it skipping over a dead end without looking into it, and moved the goal into that dead end. As we can see, the agent memorizes where it skipped a path and after not finding the goal in the other path, it backtracks and explores the dead end it skipped before, showing that it is able to remember its previous observations and adapt its exploration strategy accordingly.

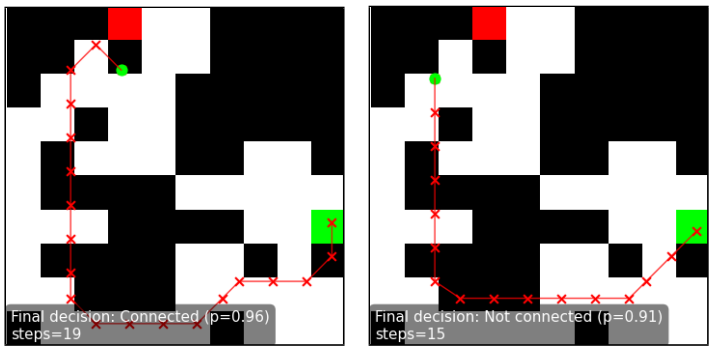
4.1.2 Starting Position

Since the starting position of the agent is chosen randomly on the start patch, we can also analyze how much the starting position matters for the consistency of the agent. As shown in Figure 4.4, there are certain mazes where the starting position can have a big impact on the decision of the agent.

That issue is most likely caused by the combination of us using a discrete action space and the blurring of the outer areas of the glimpse. Because we only work with very few pixels per glimpse, the blurring can either have a big impact on what

the agent actually sees or not, depending on the exact position. If a wall is aligned with the blur, there won't be any blurring done at all, while if the wall is slightly off the blur will make it much harder to see.

In theory, the agent should be able to realize that the information it has is uncertain, and therefore explore more to be able to make a better decision in such cases. However, that doesn't always seem to happen and therefore leads to these inconsistencies.

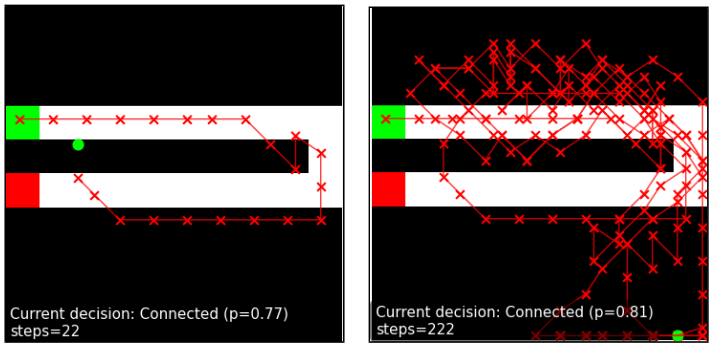


correct incorrect

Figure 4.4: An example of a maze where the initial position matters

4.1.3 Handpicked Mazes

To further analyze the performance of our model, we also created a set of handpicked mazes that are meant to be more challenging for the agent, just to explore its capabilities further. These mazes are built to highlight certain challenges for the agent, like long corridors, lots of dead ends, or special structures that won't be found in randomly generated mazes.



Initial exploration Loop

Figure 4.5: Behavior in one of the handpicked mazes

In Figure 4.5, we can see one of these handpicked mazes, where the agent initially has a pretty decent run. It explores the entire corridor and finds the goal. However, the decision head doesn't become certain enough about the connectivity of the maze and therefore doesn't stop the run. After that, the agent aimlessly wanders around, getting stuck in a loop.

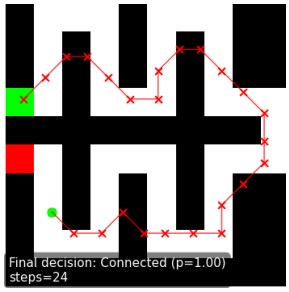
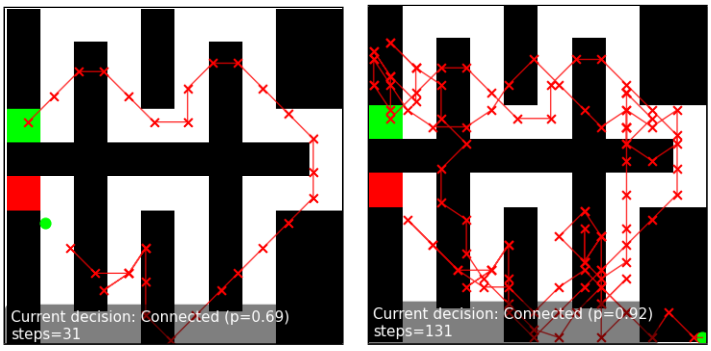


Figure 4.6: Solving a pretty complicated maze

Another interesting example is shown in Figure 4.6, where the agent is able to successfully navigate a pretty complicated maze structure. However, if we slightly modify the maze structure to make the twists a bit larger, the agent fails to properly make a decision and ends up looping again, as shown in Figure 4.7. The exploration itself looks solid, but it doesn't reach a certain level of confidence to make the decision to stop. Then we can observe a quite nice backtrack over the path, back to the start, and see the agent get fairly confident in its (correct) prediction, but even after that, it still doesn't manage to stop the run.



Initial exploration Loop

Figure 4.7: Slightly modified to increase wiggle size

Also, without showing images of all the examples here, we found that the agent generally struggles with simpler structures, for example, a single line of walls with one gap, that separates the start and goal, or a diagonal wall without any gaps. These structures are very different from the random mazes the agent was trained on, and therefore it struggles to properly explore them and make a decision.

We also found that the more complex the mazes get, the more the starting position matters for the final decision of the agent, as mentioned in Subsection 4.1.2.

4.1.4 Maze Size Generalization

We also tested our model, which was trained on a maze with a grid size of 10×10, on differently sized mazes. We ran tests on both smaller and larger mazes to see how well the agent can generalize to unseen maze sizes4.8.

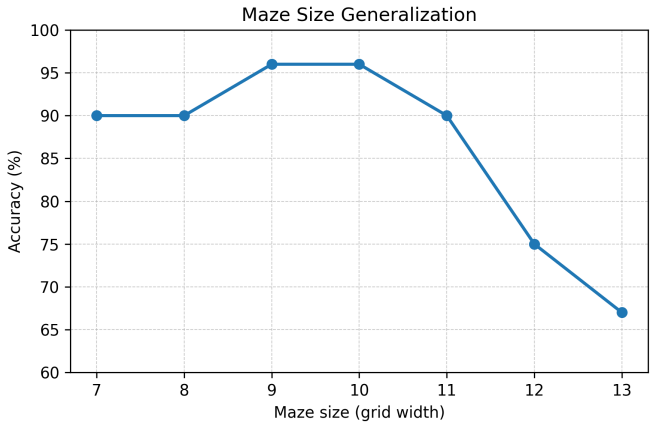


Figure 4.8: Accuracy when evaluating the 10×10 model on different maze sizes.

These results don't look that impressive at first, but this model was never trained on anything other than 10×10 mazes, so being able to generalize at all to different sizes, especially the bigger ones, is already quite impressive, since it never learned to step out of the 10×10 grid. Another point that's worth mentioning is that the steps taken are based on the image size, so that means that it also had to adapt to a different step size when changing the maze size.

4.1.5 Hyperparameters

For this section, we unfortunately could not run all of the interesting comparisons due to time constraints, but we did run a test on different loss weights for the auxiliary loss. Since that one is not weighted directly but rather weighted through weighting the different components, we adjusted these weights to see how they affect the performance. We found that it does have a big impact on the training speed and stability. Our best performing model used a weight of $\lambda_{L1}^{steps} = 0.3$ for the individual steps and a final loss weight of $\lambda_{L1}^{final} = 2$, while only using the L1 loss. For the maze dataset, we decided to only use L1 loss since this one is localized around our current gaze and therefore does not confuse the agent by punishing reconstruction of a part it never saw. Our comparison model was using a step

weight of $\lambda_{L1}^{steps} = 1$ and a final reconstruction weight of $\lambda_{L1}^{final} = 8$. As we can see in Figure 4.9, the increased reconstruction weight leads to a lot more unstable training and significantly slower convergence. Interestingly enough, the model with the higher reconstruction weight even had a higher reconstruction loss throughout the entire training, showing that just increasing the weight of the loss does not necessarily lead to better reconstructions.

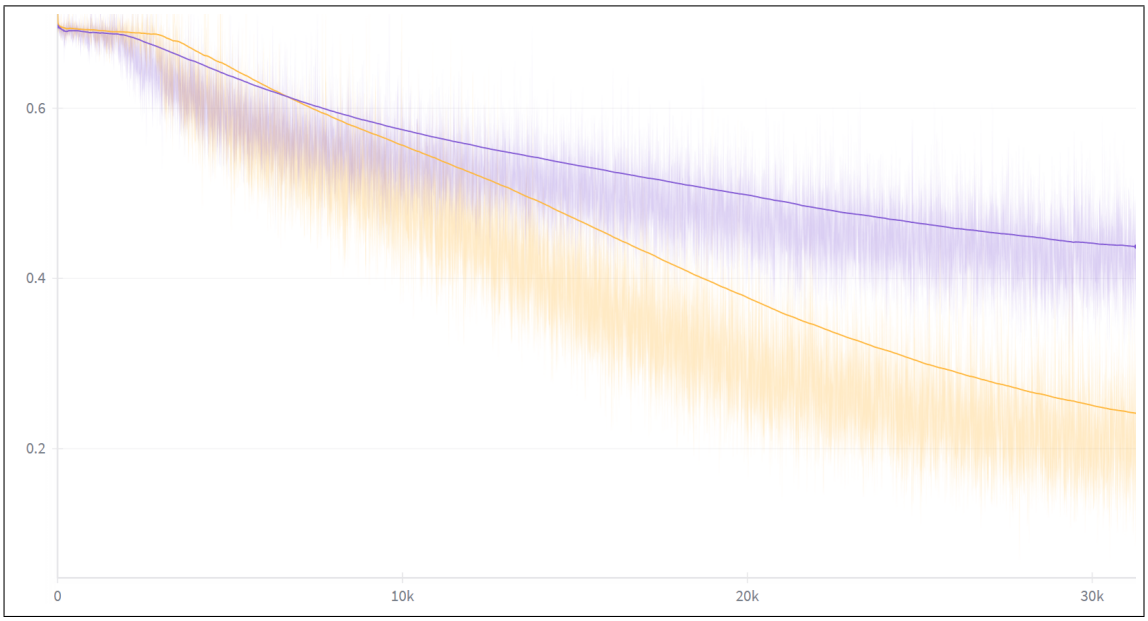
4.2 CNN Classifier comparison

To have a comparison, we also trained a baseline model that was just a simple CNN classifier. For this task, a baseline will get almost perfect results for randomly generated mazes, as it can easily learn to recognize the goal pattern in the input image. If we use similar amounts of parameters for the baseline model as for our gaze-control model, we can observe an accuracy of over 99.50%, while the gaze-control model achieves around 96.5% accuracy.

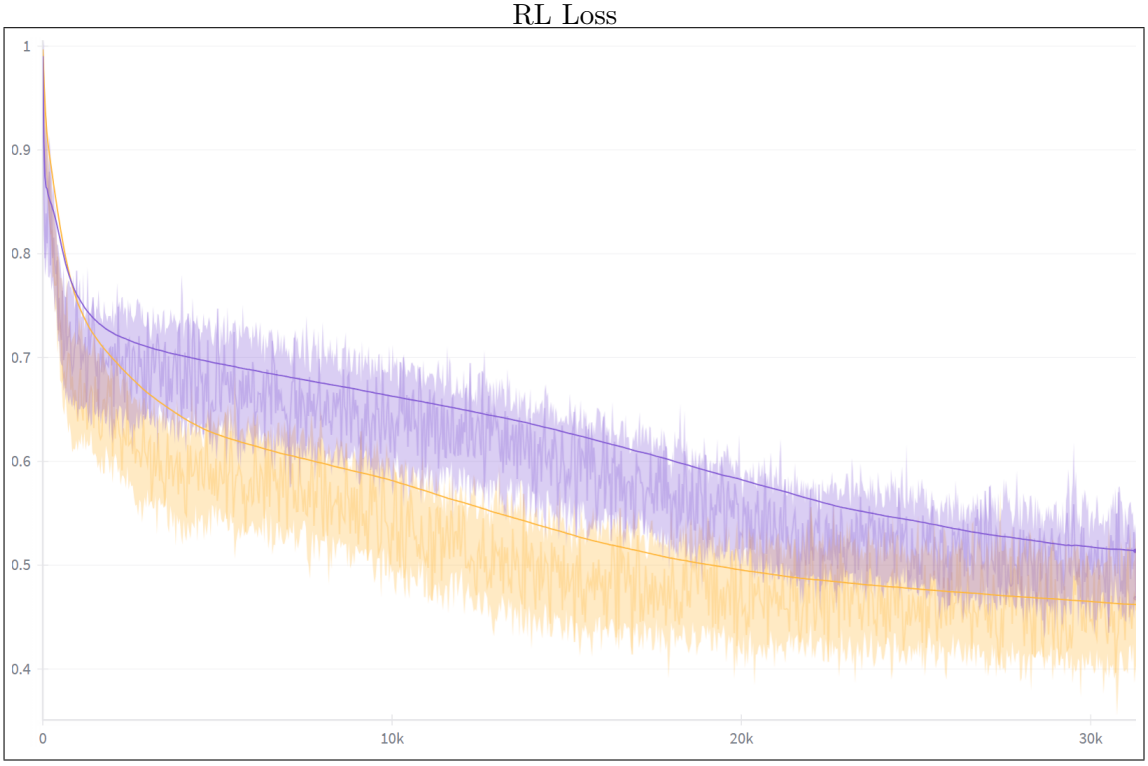
However, this comparison is not entirely fair since the classifier has access to the full image at all times, while our gaze-control model only gets a very limited view of the maze. So while we do run into efficiency problems with our model, since it has a lot of parameters and needs multiple steps to explore the maze, we use way less information to make the decision. The classifier gets the full mazes of 40×40 pixels, so 1600 pixels in total, while our gaze-control model only gets 3 glimpses scaled to 4×4 pixels each, so only 48 pixels in total for each glimpse. So as long as we need fewer than 33 glimpses (for reference, during validation, we averaged below 13 steps per maze), we use less information than the classifier to make a decision.

To further analyze the differences between the two models, we also tested both models on sets of mazes with certain minimal paths from start to goal. On a set with 1000 mazes that have a minimal path length of 35 steps, the classifier accuracy dropped to around 93%, while the gaze-control model accuracy dropped to around 75%. When increasing the minimal path length to 40 steps, the classifier accuracy dropped further to around 85%, while the gaze-control model accuracy only dropped to around 70%. Now, these are not really great results since the gaze-control model is still significantly worse than the classifier; however, we can at least note that the drop in performance for long versus very long mazes affected the gaze-control model less. Also what might be interesting but fairly logical is that our false negative to false positive ratio increased tremendously for these long-path mazes, for the normal set it was around 2:1, while for the 35 step long-path set it increased to around 27:1. Showing that the model struggles more to identify connected mazes when the path is long, which makes sense but is still worth mentioning.

Lastly, we also created a set of mazes that only consists of mazes that the classifier model classifies wrongly, while excluding trivial cases like the start being right next to the goal since the classifier for some reason often failed on those. These mazes are



RL Loss



Rec Loss

Figure 4.9: Both rec and rl loss for different weighting schemes (Yellow is the run with lower reconstruction weights)

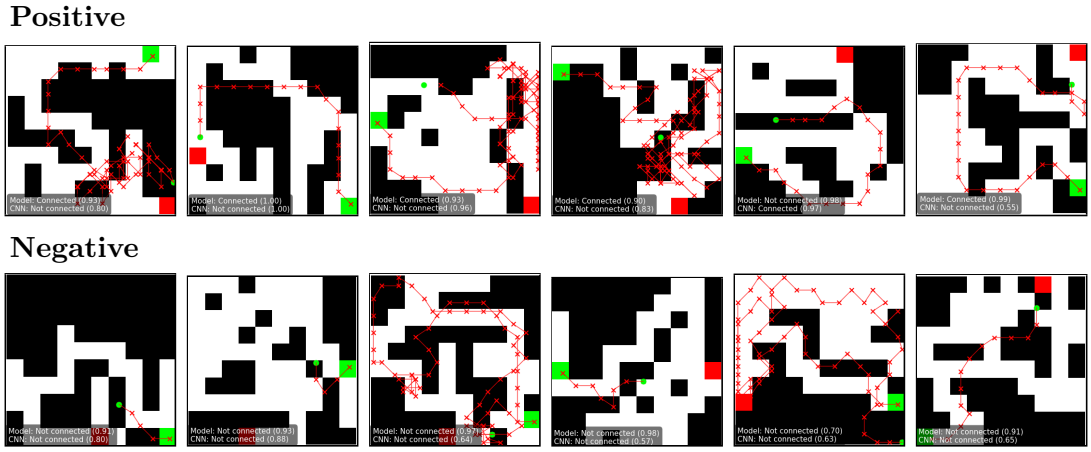


Figure 4.10: Positive vs. Negative Examples on misclassified mazes

mostly composed of long paths to the finish with lots of twists, making it hard for the classifier to properly identify the connectivity. On that dataset the gaze-control model was able to achieve a success rate of around 56%, showing that it definitely can have an advantage over a simple classifier in certain scenarios, even in fairly simple environments like these mazes. In general we observed that the model is often even more starting position dependent on these mazes than on normal ones, as mentioned in Subsection 4.1.2, and that the model often doesn't manage to stop on its own, even if it is fairly certain about its prediction. In Figure 4.10, we just took the first six mazes of that set that the gaze-control model managed to solve, excluding simple disconnected ones that the classifier failed on since they are not that interesting, and the first six mazes that it failed on, to give an impression of what these mazes look like.

4.3 Ablation Studies

To better understand the contributions of different components of a model its common to test the model without certain components. For our gaze-control model we will always need a certain core structure to be able to perform the task, which includes the encoder, the LSTM and the agent, since removing these would defeat the point of the architecture. However we can ablate some of the additional components that we added to the model to see how much they contribute to the overall performance. We would have liked to also do an ablation study on the fusion layer, however due to time constraints we were not able to get that done.

4.3.1 Decoder

In particular we tested the model without the decoder branch and therefore without the auxiliary loss.

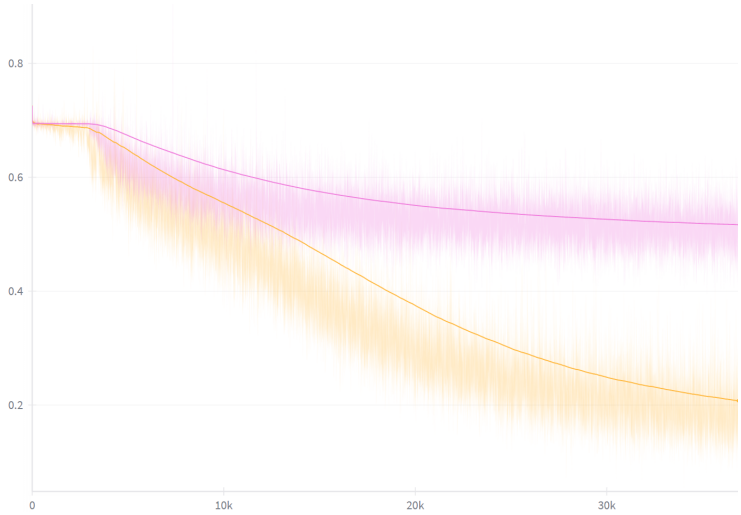


Figure 4.11: Loss Curve comparison Auxiliary Loss (Yellow is with auxiliary loss)

Without the auxiliary loss the training progress was significantly slower but if trained way longer it could potentially reach similar performance4.11. We ran the training for the ablated model twice on the same model, just to see how good we could get it to perform and it turns out that after that much training it was able to reach around 92.5% accuracy on the validation set, which is worse than the full model but still not completely terrible, showing that the reinforcement learning signal can be enough to train the model to a certain degree. However this is only tested on this fairly simple maze task and that actually makes sense. The auxiliary loss helps to stabilize training by helping the lstm memorize what was seen. If the task is simple enough the model can get away with just using the reinforcement learning signal to learn a decent exploration strategy.

We expect that on more complex datasets the auxiliary loss will be even more crucial for training since the sparse reinforcement learning signal will most likely not be enough to train the entirety of the model.

Chapter 5

Limitations

Along the way of developing and testing our model, we ran into some limitations that we want to discuss in this chapter.

5.1 Reconstruction of Datasets

As mentioned in Section 3.4, we originally wanted to use the Pathfinder and CVR datasets to evaluate our model on more complex data. However, due to the way we implemented our auxiliary loss, using a decoder to reconstruct the input image, we ran into a lot of problems. These datasets consist of thin lines on a mostly blank (or white for the CVR dataset) background. Since most of the image is just blank space, the model quickly learns to just reconstruct a blank image to minimize the loss. The minimal loss of the thin lines is not enough encouragement for the model to actually learn to reconstruct the lines, since the overall loss is already very low when reconstructing a blank image.

We tried various loss functions, but they were either not enforcing a strong enough learning signal to avoid the local minima, or we couldn't use them for local losses around the glimpse, and could only run them on the full image, which also harms training since the model never saw some of the regions that it gets penalized on.

We are certain that these problems can be fixed. Be it either with a better way of training the decoder or with a different auxiliary task that is better suited for these datasets. One of these different auxiliary losses is, for example, the Sensorimotor Reward from SUGARL (Shang and Ryoo, 2023), where they punish gazes that do not contribute useful information, which is measured in the accuracy of the decision-making policy. However, that was not within the scope of this thesis, and therefore, we had to limit ourselves to the simpler maze dataset.

5.2 Parameter and Training Efficiency

On simpler tasks like the maze dataset, our model is pretty inefficient in terms of parameters and training time compared to the classifier model. Due to the necessity of using an LSTM that keeps track of the seen glimpses, our model has a lot

of parameters compared to other models that just use feedforward architectures. Additionally, the training time is significantly higher since we have a lot of components that need to be trained together and the reinforcement learning through time steps is a lot noisier than a simple classification loss.

However, what we lack in efficiency we gain in flexibility, since our model could, for example, easily adapt to differently sized mazes without retraining or changing the architecture at all (except the decoder, which is not needed for classification). Another major advantage is that our model is already set-up to work in a dynamic environment, since we already integrate information over time steps and can therefore directly be used in a setting where the input is changing over time, like a video stream from a robot navigating in an environment or a game.

Chapter 6

Conclusion and Future Work

In this thesis, we presented an architecture for using active vision reinforcement learning models to solve visual reasoning tasks.

6.1 Summary of Findings

- We proposed a gaze-control architecture that combines an encoder–policy module with a decoder branch for auxiliary reconstruction loss.
- We evaluated the proposed gaze-control architecture on our Maze benchmark and were able to show that our model is able to successfully learn to classify mazes by actively exploring them with a limited field of view.
- Though on average worse than a feedforward classifier with full image access, our model showed promising results and was able to solve mazes that the classifier failed on.
- By using ablation studies, we were able to show that the decoder branch did improve the training stability and convergence speed of the model.
- We put a lot of effort into trying to make our model work on more complex datasets like the Pathfinder and CVR datasets; however, we ran into limitations regarding the reconstruction of these datasets.

The main contribution of this thesis is not necessarily the performance of the model on the Maze dataset, but rather the groundwork that we laid for future research in this area. We provided a reliable and modular implementation of the gaze-control architecture, along with a reproducible data pipeline and evaluation scripts. This foundation can be used to easily extend the model to more complex datasets and objectives in future work.

6.2 Future Work

There are a lot of interesting directions for future work that build upon the groundwork laid in this thesis.

- **Different Datasets:** The most obvious next step is to extend the model to more complex datasets like the Pathfinder and CVR datasets. This would require addressing the limitations regarding the reconstruction of these datasets that we discussed in Section 5.1. But we are confident that this can be achieved with some modifications to the auxiliary loss or the decoder architecture. It might also be interesting to see what happens if we just accept the reconstruction to be fully black, since we are mainly interested in the loss signal and not in the actual reconstruction.

- **Alternative Auxiliary Tasks:** Another interesting direction is to explore alternative auxiliary tasks that are better suited for complex datasets. For example, the mentioned Sensorimotor Reward from SUGARL (Shang and Ryoo, 2023) or predictive coding could be used as auxiliary tasks instead of reconstruction.

- **Transformer-based visual sequencing:** Explore replacing the recurrent state with a transformer encoder that ingests sequences of foveated patches with explicit positional/temporal embeddings, possibly enabling improved modeling of spatial/temporal dependencies.

- **Transfer Learning:** The modular nature of the model makes it well-suited for transfer learning scenarios. Future work could investigate how well the model can transfer knowledge from one dataset to another, or how well it can adapt to new tasks with limited data.

- **Dynamic Environments:** As mentioned earlier, our model is already set-up to work in dynamic environments. This would be an exciting direction for future work, where the model could be tested in scenarios where the input is changing over time, like a video stream from a robot navigating in an environment or a game. Robots are a really hot topic right now, and combining active vision with robotics could lead to some really interesting applications.

Appendix A

Used Hyperparameters

Hyperparameters used for training the models on the maze dataset are listed in Table A.1.

Table A.1: Maze model configuration hyperparameters from ‘config_maze.py‘.

Parameter	Value	Description
IMG_SIZE	(40, 40)	Input image resolution (H,W)
FOVEA_OUTPUT_SIZE	(4, 4)	Glimpse patch size after resizing per scale
FOVEA_CROP_SIZE	(4, 4)	Base crop size; larger scales use multiples
K_SCALES	3	Number of multi-scale glimpses per step
HIDDEN_SIZE	256	LSTM hidden/state dimension
POS_ENCODING_DIM	64	Positional encoding dimension for gaze
LSTM_LAYERS	1	Number of stacked LSTM layers
ENCODER_C1	12	Channels of first encoder conv block
ENCODER_C2	16	Channels of second encoder conv block
ENCODER_OUTPUT_SIZE	64	Flattened encoder feature size (2x2 pooled * C2)
DECODER_LATENT_CH	24	Latent channels in decoder fusion
FUSION_TO_DIM	128	Dimension after glimpse fusion MLP
FUSION_HIDDEN_MUL	1.5	Hidden layer multiplier in fusion MLP
BATCH_SIZE	128	Training batch size
LEARNING_RATE	3e-4	Adam learning rate
WEIGHT_DECAY	1e-5	L2 weight decay
GRAD_CLIP_NORM	1.0	Global gradient norm clipping threshold
MAX_STEPS	45	Max gaze steps per sample rollout
MAX_MOVE	0.1	Max normalized movement per step (one tile)
MIN_STEPS_BEFORE_STOP	0	Minimum steps before STOP allowed
STOP_CONF_THRESH	0.9	Required class confidence to permit STOP
START_JITTER	0.02	Uniform jitter radius around start gaze
RL_STOP_INIT_BIAS	-8.0	Initial bias discouraging premature STOP
RL_STEP_PENALTY	0.0025	Per-step reward penalty encouraging efficiency
RL_GAMMA	0.95	Discount factor for RL returns
RL_LAMBDA	0.95	GAE lambda parameter
RL_ENTROPY_COEF	0.02	Entropy bonus coefficient
RL_REWARD_SCALE	20.0	Scale factor for classification reward
RL_INIT_STD	0.2	Initial action distribution std (if stochastic)
RL_ONLY_EPOCHS	0	Epochs of RL-only training phase
EPOCHS	80	Total supervised training epochs scheduled
DEVICE	cuda/auto	Selected computation device (runtime dependent)

Bibliography

Buşoniu, L., de Bruin, T., Tolić, D., Kober, J., and Palunko, I. (2018). Reinforcement learning for control: Performance, stability, and deep approximators. *Annual Reviews in Control*, **46**, 8–28.

drewlinsley (2019). Pathfinder. `https://github.com/drewlinsley/pathfinder`. Accessed: 2025-11-07.

Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural Computation*, **12**(10), 2451–2471.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, **313**(5786), 504–507.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, **9**(8), 1735–1780.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS) 25*.

Mathieu, M., Couprie, C., and LeCun, Y. (2016). Deep multi-scale video prediction beyond mean square error.

Mnih, V., Heess, N., Graves, A., and Kavukcuoglu, K. (2014). Recurrent models of visual attention.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning.

Pihlgren, G. G., Nikolaidou, K., Chhipa, P. C., Abid, N., Saini, R., Sandin, F., and Liwicki, M. (2024). A systematic performance analysis of deep perceptual loss networks: Breaking transfer learning conventions.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2018). High-dimensional continuous control using generalized advantage estimation.

Shang, J. and Ryoo, M. S. (2023). Active vision reinforcement learning under limited visual observability.

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., and Metzler, D. (2020). Long range arena: A benchmark for efficient transformers.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2023). Attention is all you need.

Wang, Z., Bovik, A., Sheikh, H., and Simoncelli, E. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, **13**(4), 600–612.

Zerroug, A., Vaishnav, M., Colin, J., Musslick, S., and Serre, T. (2022). A benchmark for compositional visual reasoning.

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Bachelorarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Ort, Datum	Unterschrift