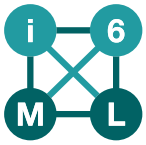


Chair of
Computer Science 6
(Machine Learning
and Reasoning)

RWTH AACHEN UNIVERSITY

The present work was submitted to the Chair of Machine Learning and Reasoning.

Learning Graph-Based Symbolic State Representations for Planning from Images

Master Thesis

Presented by

Jakob Elias Gebler
353797

Supervised by Jonas Reiter, M.Sc.

1st Examiner Prof. Hector Geffner, Ph.D.

2nd Examiner Prof. Dr. rer. nat. Christopher Morris

Aachen, October 12, 2025

With sincere thanks to my parents for their constant encouragement and support throughout my entire studies.

Abstract

Classical planning operates on explicit, symbolic models of an environment. Automatically extracting such models from high-dimensional, unstructured data, such as images, remains a significant challenge. Established methods in state representation learning often fall short, as they either rely on image reconstruction, which can encode task-irrelevant visual details, or they fail to produce representations that generalize to new problem instances within the same domain.

This thesis addresses these limitations by developing an algorithm that learns a lifted, symbolic state representation directly from images, without requiring reconstruction. An inverse action model architecture is proposed that first encodes pairs of state observations into sets of object embeddings. Subsequently, relation predictors construct a graph-based symbolic representation for each state. A graph neural network then processes these relational structures to perform the auxiliary task of jointly classifying the action that caused the transition and predicting the distance between the two states, thereby enabling an implicit learning of a meaningful representation.

The graph-based architecture is designed to be independent of the number of objects, creating a potential pathway for generalization to larger problem instances by fine-tuning only the initial image encoder. The proposed method is evaluated on the classical planning domains N-Puzzle and Sokoban. Experiments demonstrate that the model successfully learns a nearly-injective, structured state representation, and its ability to generalize is assessed by transferring the learned model from the 8-Puzzle to the 15-Puzzle. Furthermore, the utility of the learned representation is validated by employing a learned dynamics model and the predicted distance as a heuristic in a guided search, which successfully solves planning tasks within the latent space.

Contents

1 Introduction

A fundamental task and challenge in the field of artificial intelligence is to create agents that can solve complex problems. An approach to problem-solving is planning, which can be understood as the systematic search for a sequence of actions that transforms an initial state of an environment into a desired goal state [45]. To perform such a search, an agent requires an internal model, or state representation, that describes the environment in a structured manner. Human experts must handcraft such representations, a process that requires domain-specific knowledge [32]. The research area of state representation learning (SRL), however, aims to automate this process by learning meaningful representations directly from raw data, such as images, often in an unsupervised setting [32].

The type of learned state representation depends on its intended purpose. For tasks involving planning and control, the representation must capture the dynamics of the environment, that is, how actions change the environment's states. In fields such as reinforcement learning (RL), various representation learning strategies have demonstrated promising results. For instance, models like DreamerV2 learn a discrete, reconstruction-based latent space to master complex tasks from pixels [22], whereas recent approaches, such as planning with a latent dynamics model (PLDM), utilize a continuous latent space and a forward model without relying on image reconstruction [47].

Despite these advances, a significant challenge remains in learning representations that are explicitly symbolic and structured in a manner similar to the planning domain definition language (PDDL). Learning such representations is desirable because it allows the use of the well-established framework of classical planning, which features powerful, domain-independent planners that provide theoretical guarantees for their solutions.

A key work in this direction is LatPlan, which demonstrates the feasibility of learning a PDDL action schema from a discrete state representation learned from images [4]. However, this approach has notable limitations: its reliance on image reconstruction can force the model to encode visually-relevant features that are unnecessary for a symbolic interpretation. Furthermore, the learned representation is propositional and not designed to generalize to other instances of the same domain, such as transitioning from the classical planning domain 8-Puzzle to a 15-Puzzle.

To address these issues, this thesis develops an algorithm that learns a lifted, symbolic state representation directly from images. The learned representation consists of a set of objects and binary relations between them. This is achieved using an inverse action model (IAM)

that avoids the need for image reconstruction. The model first maps two state-observations (images) to a continuous, lower-dimensional vector space, which is interpreted as a set of object encodings for each state observation, respectively. Subsequently, a series of relation predictors, implemented as multilayer perceptrons (MLPs), determine which binary relations hold for all pairs of objects in each state observation.

The set of objects and their binary relations can be represented as a directed multigraph. This structure allows the use of graph neural networks (GNNs) to perform the primary learning task of the IAM: predicting the action that led from the first given state observation to the second [32]. In addition to classifying the action, the model also tries to predict the number of steps (distance) required to transition between two states. This distance prediction serves as a crucial additional learning signal, particularly for non-consecutive states. A key advantage of this architecture is its potential for generalization. Since the relation predictors and the GNN are independent of the number of objects, the trained model can be applied to larger problem instances by simply varying the number of encoded objects. For such a transfer, only the initial image encoder needs to be retrained on the new instance class.

The proposed algorithm is evaluated on the 8-Puzzle and Sokoban planning domains. Experiments confirm that the model successfully learns a structured state representation and demonstrate promising generalization capabilities. Furthermore, to provide a functional validation that the learned representation captures the environment's dynamics, the predicted distance is employed as a heuristic in a guided search, successfully solving planning problems within the learned latent space.

1.1 Key Contributions

The key contributions of this thesis are as follows:

- The design and implementation of a novel IAM architecture that learns a lifted, symbolic state representation of objects and binary relations directly from pixel data.
- An empirical evaluation of the model's ability to generalize to problem instances larger than those seen during training.
- A demonstration that the method can be transferred to another classical planning domain (Sokoban).
- A validation of the learned dynamics by employing the predicted distance as a heuristic in a guided search.

1.2 Thesis Outline

The remainder of this thesis is structured as follows. Chapter 2 provides the theoretical foundation for this work, covering key concepts in state representation learning, planning,

and GNNs. Chapter 3 discusses related work in representation learning and planning from images and situates this work within the existing literature. Chapter 4 presents the proposed model architecture, the experimental environments, the methodology for data generation and training, and the evaluation protocols. Chapter 5 presents and discusses the empirical results of the experiments, evaluating the quality of the learned state representation, its generalization capabilities, and its practical utility in planning tasks. Finally, Chapter 6 concludes the thesis by summarizing the key findings and outlining potential directions for future research.

2 Background

This chapter provides the theoretical foundations for the methods presented in this thesis. First, Section 2.1 provides an introduction to core concepts of deep learning, covering multilayer perceptrons, convolutional neural networks for processing visual data, and graph neural networks for operating on relational structures. Next, Section 2.2 details the field of state representation learning, surveying key architectures, including autoencoders, variational autoencoders, and inverse action models. Finally, Section 2.3 introduces classical planning, explaining its reliance on symbolic environment models, the planning domain definition language, and the role of heuristic search.

2.1 Deep Learning

The model components presented in this thesis are implemented using deep learning concepts. This field of machine learning utilizes deep neural networks, models composed of multiple processing layers to learn arbitrary mappings from input to output data by optimizing a set of internal parameters [16]. The following covers the fundamental deep learning principles applied in this work.

2.1.1 Multilayer Perceptrons

MLPs, also known as feedforward neural networks, are a foundational class of deep learning models, recognized as universal function approximators [48]. An MLP is composed of a sequence of layers, where each layer applies an affine transformation followed by a non-linear activation [8]. The function for the i-th layer is defined as:

$$\mathbf{x}^{(i)} = h\big(\mathbf{W}^{(i)}\mathbf{x}^{(i-1)} + \mathbf{b}^{(i)}\big), \tag{2.1}$$

with $i > 0$. The layers' input and output are denoted as $\mathbf{x}^{(i-1)}$ and $\mathbf{x}^{(i)}$, respectively. $\mathbf{W}^{(i)}$ and $\mathbf{b}^{(i)}$ are the weight matrix and bias vector, and compose the learnable parameters of the i-th layer. The function $h(\cdot)$ is an element-wise, non-linear activation function, such as the logistic sigmoid or the sigmoid-weighted linear unit (SiLU). An MLP is formed by chaining these layers, where the output of one layer serves as the input to the next. The resulting function can be formalized as $f_\theta : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are the input and output spaces, respectively, and $\theta = (\mathbf{W}, \mathbf{b})$ is the set of all learnable parameters.

The objective of training an MLP is to find the parameters θ that minimize the empirical risk on a given training dataset $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$. This risk, denoted as $J(\theta)$, is the expected value of the loss function $\mathcal{L}$ over the empirical distribution defined by the dataset. This is equivalent to the average loss over all samples:

$$J(\theta) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim D}[\mathcal{L}(f_\theta(\mathbf{x}), \mathbf{y})] = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i). \tag{2.2}$$

The optimization process of finding the parameters θ that minimize $J(\theta)$ is known as empirical risk minimization [16].

Gradient Descent

A standard approach to find such parameters θ is to use gradient-based methods. The foundational algorithm is *gradient descent*, which iteratively updates the parameters in the opposite direction of the gradient of the loss function, $\nabla_\theta J(\theta)$ [8]. The update rule for a single step is given by

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta), \tag{2.3}$$

where η is a small positive scalar known as the learning rate. A drawback of this approach is that it requires computing the gradient over the entire training dataset for a single parameter update. This is computationally expensive and often intractable for large datasets. An alternative is to approximate the gradient by computing it only for a single datapoint. This process is known as *stochastic gradient descent*.

While computing the gradient over a single datapoint is unbiased but has a high variance, in modern practice, the gradient is usually estimated based on a small subset of samples, also called mini-batches. This approach reduces the variance of the gradient estimate compared to per-example-based gradient descent, while remaining computationally efficient. This mini-batch variant is known as stochastic gradient descent (SGD) [8]. While SGD provides the basic optimization principle, more advanced algorithms are used to accelerate convergence and improve stability, such as Adam with decoupled weight decay (AdamW) [34] or root mean square propagation (RMSprop) [44].

Supervised and Self-supervised Learning

The optimization methods described apply to traditional *supervised learning*, which relies on human-annotated labels. This work, however, employs *self-supervised learning*. In this paradigm, a function is learned from inputs to outputs, but the target outputs (or labels) are obtained automatically from the input data itself, without requiring separate human annotation [9]. Although the origin of the labels differs, the training mechanics are identical to

the supervised case: a loss function measures the prediction error, and gradient-based methods like SGD are used to minimize this loss by adjusting the model's parameters.

Regression and Classification

Neural networks can be adapted to solve different types of predictive tasks, where *classification* and *regression* are the most common. Classification is the task of predicting a discrete class label from a finite set of categories. The output layer of a classification network typically uses a softmax function to produce a probability distribution over the possible classes [8]. A common objective for training is to minimize the cross-entropy loss, which for a single example is defined as:

$$\mathcal{L}_{CE}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{c=1}^{C} y_c \log(\hat{y}_c), \tag{2.4}$$

where C is the number of classes, $\mathbf{y}$ is the one-hot encoded true label vector, and $\hat{\mathbf{y}}$ is the predicted probability distribution from the model [16].

Regression is the task of predicting a continuous numerical value. For this task, the output layer typically consists of one or more neurons with a linear activation function. A standard loss function is the mean squared error (MSE), which penalizes the squared difference between the predicted and true values:

$$\mathcal{L}_{MSE}(y, \hat{y}) = (y - \hat{y})^2, \tag{2.5}$$

where y is the true continuous value and $\hat{y}$ is the model's prediction.

The model developed in this thesis employs both tasks jointly, predicting a discrete class label and a continuous value for each training example. In addition to MLPs, this work integrates two other fundamental deep learning architectures for handling specialized data: convolutional neural networks (CNNs) for processing visual inputs, and GNNs for operating on relational data. Their core concepts are described in the following sections.

2.1.2 Convolutional Neural Networks

CNNs are a class of neural networks specifically designed for processing data with a grid-like topology, such as images [16]. They are architecturally distinct from MLPs through the use of the convolution operation, which leverages two key principles: sparse interactions and parameter sharing. Instead of the full matrix multiplication of a dense layer, a convolutional layer uses a set of learnable kernels that are convolved with the input. Following Goodfellow *et al.* [16], the convolution operation, denoted by $*$, for a 2D input $\mathbf{X}$ and kernel $\mathbf{K}$ of size $M \times N$ is defined as:

$$\mathbf{Y}(i, j) = (\mathbf{X} * \mathbf{K})(i, j) = \sum_{m=1}^{M} \sum_{n=1}^{N} \mathbf{X}(i + m, j + n)\mathbf{K}(m, n). \tag{2.6}$$

Here, the summation indices m and n iterate over the kernel elements, which are the actual parameters, or weights, that are optimized during training. The output feature map, $\mathbf{Y}$, is generated by applying this operation across all spatial locations of the input. Sparse interactions arise because each output unit depends only on a local region of the input defined by the kernel size. Parameter sharing refers to using the same kernel at every spatial location, allowing it to detect features regardless of their position [16]. The output of this operation is typically passed element-wise through a non-linear activation function.

Another key component in many CNN architectures is the pooling layer. Instead of convolving the input, the kernel acts as an aggregation function over the respective input region. A common aggregation function is max pooling, which takes the maximum output within the kernel region [16].

Typically, a CNN architecture consists of a series of alternating convolutional and pooling layers. The output from the final stage is then flattened into a vector and often processed by an MLP to produce the final output. CNNs are employed in this work as they are well-suited for the model's image-based inputs. Since this thesis also utilizes graph-based representations, which are used to train GNNs, they are introduced in the following section.

2.1.3 Graph Neural Networks

While conventional deep learning models like MLPs and CNNs are designed for data with a fixed-size, grid-like structure, GNNs are a class of models specifically designed to learn directly from graph-structured data [58]. A graph is formally defined as a tuple $Z = (V, E)$, where V is a set of nodes and $E \subseteq V \times V$ is a set of edges representing relationships between the nodes. For a node $v \in V$, its neighboring nodes are denoted as $\mathcal{N}(v) = \{u \in V \mid (u, v) \in E\}$.

Motivated by the need to capture relational information, GNNs learn a low-dimensional vector representation, or embedding, for each node. Instead of relying on hand-crafted features like node degree or cluster coefficients [57], GNNs generate these embeddings through an iterative message-passing procedure between neighboring nodes [23]. This procedure is defined by two primary operators: AGGREGATE, which collects information from a node's neighbors, and COMBINE, which updates a node's own embedding using the aggregated information. Both operators are typically implemented as functions with learnable parameters, such as neural networks. For an arbitrary node $v \in V$, one layer k of this process can be formalized as:

$$\mathbf{a}_v^k = \text{AGGREGATE}^k\big(\{\mathbf{h}_u^{k-1} : u \in \mathcal{N}(v)\}\big) \tag{2.7}$$

$$\mathbf{h}_v^k = \text{COMBINE}^k\big(\mathbf{h}_v^{k-1}, \mathbf{a}_v^k\big), \tag{2.8}$$

where $\mathbf{h}_v^k$ is the embedding of node v after layer k [57]. By composing multiple such layers, a node's final embedding can capture structural information from other nodes connected over k edges.

While the primary output of a GNN is a set of node-level embeddings, these can be aggregated to produce a single graph-level representation for tasks like graph classification. This is achieved through a pooling or readout function, which might compute the sum or average of all node embeddings.

Many different GNN architectures exist, distinguished by their specific implementations of the AGGREGATE and COMBINE functions and other details, such as supporting mini-batch training. A key strength of this message-passing framework is its inductive learning capability. Because the operations are defined locally on the graph structure, a trained GNN can be applied to graphs of different sizes and topologies than those seen during training, a property crucial for generalization [23].

Relational Graph Convolutional Networks

Within this work, state observations are represented by multi-relational graphs, where different edge types represent different kinds of relationships (cf. Section 4.1.3). Multiple edge types require specific GNN architectures such as the relational graph convolutional networks (R-GCNs) introduced by Schlichtkrull *et al.* [46].

The idea of an R-GCN is to use a different weight matrix for each relation type. The neighborhood aggregation and update steps are combined into a single update rule for the embedding of a node v at layer $k+1$:

$$\mathbf{h}_v^{k+1} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_v^r}\frac{1}{c_{v,r}}\mathbf{W}_r^k\mathbf{h}_j^k + \mathbf{W}_0^k\mathbf{h}_v^k\right), \tag{2.9}$$

where $\mathcal{R}$ is the set of relation types, $\mathcal{N}_v^r$ is the set of neighbors of node v under relation r, and $\mathbf{W}_r^k$ is the learnable weight matrix for that specific relation. The term $c_{v,r}$ is a normalization constant (e.g., set to $|\mathcal{N}_v^r|$) that scales the aggregated messages to prevent unstable behavior of the embeddings during training. Due to the large number of parameters that can arise from having many relation types, Schlichtkrull *et al.* [46] introduce regularization techniques such as basis- and block-diagonal-decomposition, helping to prevent overfitting and improve parameter efficiency.

2.2 State Representation Learning

The ability of an artificial agent to perform complex tasks is highly dependent on the way it represents its environment and the problem at hand. While raw sensor data, such as a stream of pixels from a camera, contains a lot of information, it is often high-dimensional, redundant, and not directly accessible to decision-making or planning algorithms [32]. Historically, human experts have been responsible for feature engineering, a process of manually designing compact and relevant representations from raw data [32]. For instance, an autonomous robot navigating

on Mars might use multiple sensor systems combined with a probabilistic model to derive its position as a set of (x, y, z) coordinates, a more meaningful representation for navigation than raw sensor readings [39]. While it is possible for modern algorithms to learn policies directly from high-dimensional observations [35], it is often beneficial to first map these observations into an explicit, lower-dimensional, and more meaningful representation [32].

SRL is the research area that aims to automate this process. It aims to learn a function that maps high-dimensional observations into a compact, low-dimensional representation without requiring explicit, prior expert knowledge about the state-describing features. The result is a state description in a latent space that encodes the most relevant information from the observation with respect to a specific task [32].

2.2.1 Formalism

The described setting is formalized in the following. SRL considers an agent interacting with an environment $\mathcal{E}$ in discrete time steps. At each time step t, the environment is in a real but unknown state s_t from the state space $\mathcal{S}$. The agent can execute an action a_t from a state-dependent action space $\mathcal{A}(s_t)$, which causes the environment to transition to a successor state s_{t+1} [32]. The agent does not perceive the true state s_t directly. Instead, it receives a high-dimensional observation $x_t \in \mathcal{X}$, such as an image. The goal of SRL is to learn a state representation function, enc : $\mathcal{X} \to \mathcal{Z}$, that maps an observation x_t to a state representation $z_t \in \mathcal{Z}$ in a lower-dimensional latent space [32]. A schematic illustration of this setup is provided in Figure 2.1. The desired properties of the learned state representation z_t depend on

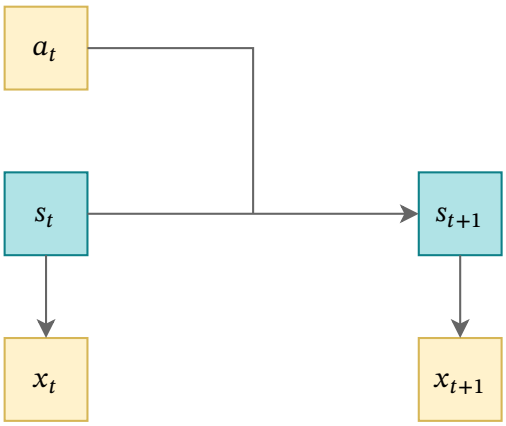


Figure 2.1: Discrete environment schema: An action a_t transitions a state s_t into a successor state s_{t+1}. Yellow boxes indicate known variables. Blue boxes are not directly accessible and cannot be represented.

the downstream task. For control and planning, it is crucial that the representation captures the dynamics of the environment [32]. In the following, different architectures for learning state representations are introduced.

2.2.2 Architectures

A variety of neural network architectures exist for learning state representations. These methods are often guided by reconstruction, where the representation must capture enough information to recreate the original observation, and dynamics modeling, where the representation must encode how the environment evolves under the influence of actions. Different model architectures integrate one or both principles to learn latent space representations [32].

Autoencoders

An autoencoder (AE) is a framework for unsupervised SRL that learns a compressed data representation through reconstruction. While the core components can be any parametric function, a common approach is to implement them as a pair of neural networks [16]:

- the encoder network, $\text{enc}_\phi : \mathcal{X} \to \mathcal{Z}$, which maps an input $x \in \mathcal{X}$ to a compressed latent representation $z \in \mathcal{Z}$, and
- the decoder network, $\text{dec}_\theta : \mathcal{Z} \to \mathcal{X}$, which reconstructs the original input $\hat{x} = \text{dec}_\theta(z)$ from the latent representation.

The training objective is to find the parameters ϕ and θ that minimize the expected reconstruction loss over a dataset D:

$$\min_{\phi,\theta} \mathbb{E}_{x\sim D}\big[\mathcal{L}_{\text{recon}}\big(x, \text{dec}_\theta\big(\text{enc}_\phi(x)\big)\big)\big], \tag{2.10}$$

where $\mathcal{L}_{\text{recon}}$ is a suitable, differentiable loss function such as the sum of squared errors [5]. This allows for optimization via gradient-based methods. The lower dimensionality of the latent space acts as an information bottleneck, forcing the model to learn a representation that preserves the most relevant features required for reconstruction [5].

While AEs are effective for dimensionality reduction, they often produce entangled representations where a single latent dimension may correspond to multiple independent data-generating factors [32]. This lack of explicit structure can make the latent space difficult to interpret semantically, making it suboptimal for downstream tasks that rely on a disentangled and structured understanding of the observed environment.

Variational Autoencoders

Variational autoencoders (VAEs) extend standard AEs by framing representation learning as probabilistic latent-variable modeling [16]. The data x is generated from a latent variable z through the conditional distribution $q_\theta(x|z)$, parameterized by a decoder network $\text{dec}_\theta : \mathcal{Z} \to \mathcal{X}$. The distribution $q_\theta(x|z)$ is typically modeled as a Gaussian, where the decoder, given a latent vector z, outputs the mean μ_θ and diagonal covariance σ_θ. Additionally, a prior

distribution $q(z)$ is imposed over the latent space to encourage a well-structured representation, usually a zero-mean unit-variance Gaussian $q(z) = \mathcal{N}(0, I)$.

The training objective is to maximize the evidence lower bound (ELBO) on the data log-likelihood:

$$\mathcal{L}_{\text{ELBO}} = \mathbb{E}_{z \sim p_\phi(z|x)}[\log q_\theta(x|z)] - D_{\text{KL}}\big(p_\phi(z|x) \,\|\, q(z)\big), \tag{2.11}$$

where $p_\phi(z|x)$, approximates the intractable posterior over the latent space, typically modeled as a Gaussian distribution parameterized by the encoder

$$\text{enc}_\phi : \mathcal{X} \to \mathbb{R}^d \times \mathbb{R}^d, x \mapsto \big(\mu_\phi(x), \sigma_\phi(x)\big). \tag{2.12}$$

The first ELBO term encourages accurate reconstruction, while the Kullback-Leibler (KL) divergence aligns $p_\phi(z|x)$ with the prior $q(z)$, yielding a smooth and continuous latent space [16]. To allow gradient-based optimization despite the stochastic sampling, VAEs employ the *reparameterization trick*, expressing z as

$$z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I), \tag{2.13}$$

where $\odot$ denotes element-wise multiplication. This formulation allows gradients to propagate through μ_ϕ and σ_ϕ during backpropagation [16].

By enforcing distributional structure in this way, VAEs learn continuous, semantically meaningful latent representations that can disentangle generative factors such as object position or shape. However, while effective for static data, they do not explicitly capture the environment dynamics, which are essential for planning or control [27, 32].

Forward Model Learning

To capture environment dynamics, *forward models* (or dynamics models) are trained to predict how latent states evolve under the influence of actions [32]. Given a latent representation z_t and an action a_t, a dynamics model $\text{dyn}_\chi : \mathcal{Z} \times \mathcal{A} \to \mathcal{Z}$ with learnable parameters χ predicts the next latent state:

$$\hat{z}_{t+1} = \text{dyn}_\chi(z_t, a_t). \tag{2.14}$$

The forward model is trained by minimizing a loss $\mathcal{L}_{\text{dyn}}$ that measures the discrepancy between the predicted latent state $\hat{z}_{t+1}$ and the encoded representation of the observed next state $z_{t+1} = \text{enc}_\phi(x_{t+1})$, where $\text{enc}_\phi : \mathcal{X} \to \mathcal{Z}$ is the state representation function with learnable parameters ϕ that maps environment observations to latent representations. A schematic of the forward model learning architecture is shown in Figure 2.2.

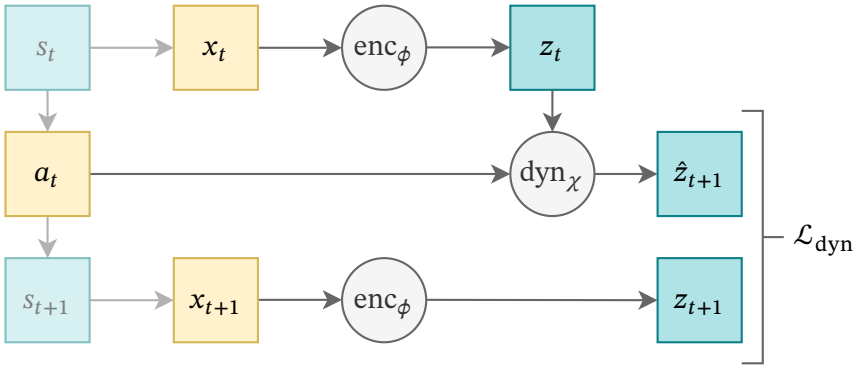


Figure 2.2: Forward Model Learning Schema: The encoder enc_ϕ maps an observation x_t to its latent representation z_t. A dynamics model dyn_χ then predicts the subsequent latent state $\hat{z}_{t+1}$ based on z_t and an action a_t. The model is trained by minimizing a loss $\mathcal{L}_{\text{dyn}}$ between the predicted state $\hat{z}_{t+1}$ and the encoded representation of the true successor observation, z_{t+1}.

Since both the state representation function enc_ϕ and the forward model dyn_χ are typically implemented as neural networks, the dynamics loss is fully differentiable, allowing the use of gradient-based methods to jointly learn the representation and forward model parameters.

While this predictive objective enforces information of the dynamics in the latent space, it may lead to a *latent collapse*, where representations degenerate into trivial solutions. Regularization strategies such as additional reconstruction losses or variance-covariance regularization have been proposed to counteract this effect and preserve meaningful state representations [6, 18].

Inverse Action Model Learning

An alternative approach to learn a state representation that captures environment dynamics is the *inverse action model* (IAM). Instead of predicting the successor state, an IAM predicts the action a_t that caused a transition between two consecutive states [32]. Given two observations x_t and x_{t+1}, the encoder $\text{enc}_\phi : \mathcal{X} \to \mathcal{Z}$ maps them to latent representations $z_t = \text{enc}_\phi(x_t)$ and $z_{t+1} = \text{enc}_\phi(x_{t+1})$. The classification model $\text{cls}_\psi : \mathcal{Z} \times \mathcal{Z} \to \mathcal{A}$ with learnable parameters ψ then predicts the action:

$$\hat{a}_t = \text{cls}_\psi(z_t, z_{t+1}). \tag{2.15}$$

Both the encoder enc_ϕ and the classification model cls_ψ are trained end-to-end by minimizing a classification loss $\mathcal{L}_{\text{cls}}$, such as cross-entropy, which measures the deviation between the predicted action $\hat{a}_t$ and the true action a_t. A schematic of this architecture is shown in Figure 2.3. As with the forward model, differentiable encoders and classifiers (e.g., neural networks) allow IAMs to be trained with gradient-based methods [32].

Unlike reconstruction-based methods, such as AE, which encode static information from individual observations, forward- and IAMs abstract the representation further and learn to

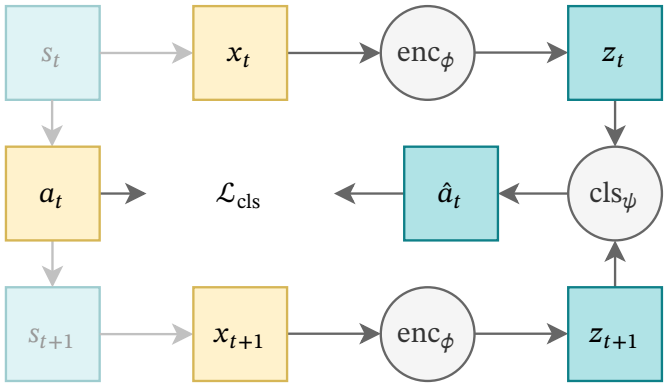


Figure 2.3: Inverse Action Model Learning Schema: An encoder enc_ϕ maps two consecutive observations, x_t and x_{t+1}, to their respective latent representations, z_t and z_{t+1}. The inverse model cls_ψ then predicts the action $\hat{a}_t$ that likely caused the transition from z_t to z_{t+1}. The encoder and the inverse model are trained jointly by minimizing a classification loss $\mathcal{L}_{cls}$ between the predicted action $\hat{a}_t$ and the ground-truth action a_t.

explicitly incorporate information about the environment dynamics in the state representation [32]. Learning explicit dynamics models from such representations potentially allows for planning within the latent space. Latent space predictions, however, are not necessarily sufficiently accurate, which means that over a sequence of actions, the deviation of the predicted state can differ significantly from the actual state that would have been reached if the steps had been executed, due to error accumulation.

Traditional planning approaches, in contrast, assume that the state change can be described precisely and that action sequences of any length can therefore be simulated [45]. An example of this is classical planning, which represents states and actions symbolically. Since this work aims for such a symbolic state representation, the classical planning domain is explained below, followed by an introduction of model predictive control, which can be used for planning with inaccurate latent models [20].

2.3 Classical Planning

Classical planning is a long-established part of AI research. It uses mathematical logic to represent environments as planning problems symbolically, allowing for planning through search and reasoning. Therefore, classical planning assumes a finite and discrete state space $\mathcal{S}$ and action space $\mathcal{A}$, and to know which actions a from the action space $\mathcal{A}$ are applicable in each state. Furthermore, a deterministic dynamics function $s' = \text{dyn}(s, a)$ is assumed. Together with a known initial state $s_0 \in \mathcal{S}$, a set of goal states $S_G \subseteq \mathcal{S}$, and a positive action cost $c(a, s)$, this represents a state model $M = (\mathcal{S}, s_0, S_G, \mathcal{A}, \text{dyn}, c)$. A plan is a sequence of actions that transitions s_0 into a state $s \in S_G$. A plan is considered optimal if it has a minimal sum of action

costs. Other planning models can be derived by relaxing these assumptions, for example, by allowing non-deterministic dynamics or partial observability.

It is noted that, different from the dynamics models mentioned in the last section, a state transition, whether the transition function is deterministic or not, always results in a well-defined symbolic state that corresponds to a true state of the environment. This is a crucial property that enables classical planning models to conduct a systematic search for goal states.

The introduced state model can be expressed using planning languages, such as the Stanford Research Institute Problem Solver (STRIPS) language or PDDL.

2.3.1 Planning Languages

Planning languages define a general standard for formulating state models, which planners use to solve planning problems. Examples of such languages are STRIPS and PDDL, which represent a formal syntax for describing states, actions, and goals [14].

STRIPS

STRIPS is a simple but expressive planning language based on propositional logic [14]. A STRIPS problem is formally a tuple $P = \langle F, O, I, G \rangle$, where F is a finite set of grounded, functionless atoms (also called fluents or propositions) that describe the properties of the environment, O is the set of all operators, representing the actions, and $I \subseteq F$ is the initial state, defined as the set of atoms that are true. Atoms not in I are assumed to be false (closed-world assumption) [45]. Furthermore, $G \subseteq F$ is the goal situation, a set of atoms that must be true in a goal state.

Each operator $o \in O$ is defined by three sets of atoms:

- $\text{Pre}(o) \subseteq F$: The preconditions of the operator.
- $\text{Add}(o) \subseteq F$: Atoms that become true after the operator is executed.
- $\text{Del}(o) \subseteq F$: Atoms that become false after the operator is executed.

With respect to a state model, a state s is any subset of F. An action o is applicable in state s if $\text{Pre}(o) \subseteq s$. The next state s' is computed by the transition function $s' = (s \setminus \text{Del}(o)) \cup \text{Add}(o)$. A STRIPS problem P therefore implicitly defines the state model $\mathcal{S}(P)$ [14].

PDDL

While the STRIPS formalism described above, also called ground STRIPS, uses a set of propositions, it can be generalized to a lifted representation with typed variables and action schemas depending on them. PDDL is the de facto standard syntax for describing such lifted planning problems. Additionally, the language separates the domain (environment description) from a specific problem instance [24].

A *domain D* defines the type of objects, the predicates that describe the relationships between the objects, and the action schema. The action schema works the same way as with ground STRIPS, except that the actions can now depend on typed variables. Predicates for which objects have substituted the variables are referred to as grounded, otherwise as lifted. A *problem instance I* defines all objects, the initial state (grounded predicates), and the target specification [24]. Together, the domain and instance represent a PDDL description $P = (D, I)$.

When a planner processes these descriptions, the action schemas are grounded by substituting the variables with all possible object combinations of the correct type, resulting in a set of operators equivalent to those in ground STRIPS. This schema-based representation avoids the redundancy of explicitly listing every possible action, making it scalable for defining complex domains [14, 24].

2.3.2 Guided Search and Heuristics

For efficient search, PDDL planners typically employ heuristic-based search methods, such as A*. The state-dependent heuristic function, which measures the distance to the goal, is of crucial importance and is intended to prevent the entire state space from being considered in the worst case. A key strength of classical planning is the ability to derive domain-independent heuristics automatically from the symbolic PDDL description [14]. This entire search paradigm is enabled by the discrete, symbolic, and deterministic nature of state transitions defined in the model.

As previously discussed, learned latent dynamics models do not inherently share these properties, as they often operate on continuous vector representations and may exhibit stochasticity. Nevertheless, heuristics can also be defined or learned for the predicted states. These can then be used to execute rollouts of fixed length and compare them based on the heuristics. Subsequently, one or more promising actions are executed, the resulting state is observed, and a new rollout is started. Research fields that pursue this search concept include model predictive control and model-based reinforcement learning [10, 36].

3 Related Work

SRL has emerged as an important component across various domains that use high-dimensional and unstructured data. In particular, when agents perceive image data from the environment, it can be challenging to extract low-dimensional representations that capture relevant environmental information.

3.1 Reconstruction-based Representation Learning

AEs are a common and straightforward method of SRL. In a self-supervised manner, they learn a lower-dimensional representation by first encoding the input into a latent space and then reconstructing the original input. The objective of minimizing the reconstruction error encourages the model to capture and compress relevant environmental information within its latent space representation [5]. While AEs facilitate more accessible representations for downstream tasks and have proven effective due to their non-linear mapping capabilities, they do not necessarily yield disentangled representations of the underlying generative factors [25]. The dimensions of AE-learned representations are often entangled and highly correlated, making semantic interpretation difficult. This limitation has been explicitly addressed by models like the β-VAE [25]. AEs and their variants are employed in a wide range of applications, including denoising, feature extraction, synthetic data generation, anomaly detection, and planning [2, 19, 51, 52].

3.2 Scene Graphs

While approaches like β-VAEs learn disentangled properties such as object position and shape, these representations remain in an abstract, compressed form and lack an explicit, symbolic description of the scene. The rapidly evolving field of scene graphs (SGs) tries to address this by explicitly modeling objects and their pairwise relationships in a graph [61]. Scene graphs can be used in a variety of ways. Some models use them to generate images from given graphs, like Johnson *et al.* [26]. Khandelwal and Sigal [28] extract SGs from images. Scene graphs are also used in combination with language models in multi-modal settings to use information outside of the image for interpretation and reasoning [11, 29, 33, 63]. This also allows the use of scene graphs for question-answering problems [53, 62], which were introduced by Antol *et al.* [3]. The model GRID, proposed by Ni *et al.* [38], uses a given scene graph of the agent and the environment as input to a large language model (LLM) to fulfill a human-language instructed task. Similarly, Ekpo *et al.* [12] presented the VeriGraph model, which creates a

scene graph from an image of the initial and goal state. Then, using an LLM to generate a sequence of actions to transition from the initial to the goal scene graph.

Other approaches extend SGs along a temporal dimension, resulting in spatio-temporal scene graph generation (ST-SGG), like Zhu *et al.* [62]. This abstraction is directly related to our experiment, as it also models the dynamics of the environment through binary relations between objects. Unlike in our case, scene graphs typically represent objects and their relations using human-readable text labels. In contrast, this thesis encodes the environment and its dynamics as a graph in an abstract latent space.

3.3 Latent Dynamics Model

Models like GRID or VeriGraph use scene graphs without a temporal dimension as an intermediate representation. Instead, the contextual understanding of language models is used to derive a plan. In contrast, other approaches use a given or learned explicit model of the environment to plan with, as in the field of reinforcement learning or optimal control [47]. In the area of classical planning, explicit dynamics models, such as PDDL, are also used [15].

These models and their features are often created by humans and are therefore easily interpretable. Images, on the other hand, are high-dimensional, and features cannot be defined easily. Developing a dynamics model directly on the images is therefore difficult. Projecting the images first into a lower-dimensional latent space is one way to learn features (e.g., through autoencoders). With these learned features, a dynamics model can then be created.

A well-known example of this type of model is Dreamer [19]. Here, a VAE was used to learn a continuous latent space in which the images of the environment are reconstructed from the latent space. In subsequent further developments, Dreamer v2 [21] and Dreamer v3 [22], the latent space was discretized, which empirically led to better results without a precise understanding of the reason. Simultaneously, a transition model and a reward model are learned. The transition model computes the next latent state given a latent state and a discrete action. The reward model predicts the expected reward after an action. The trained model now seeks to find actions that maximize expected reward. With this architecture, model-based agents could be trained that achieved human-level performance in Atari games [21]. Furthermore, it could be shown that model-based RL can thus surpass well-known model-free RL methods [21].

As an extension of the Dreamer model, FOCUS [13] was developed. Instead of learning a single latent state for the entire image, FOCUS uses an object-centric world model and learns a separate latent state for each object. This enables object-centric exploration by deliberately exploring the interaction between objects in order to better learn the state and dynamics of an object [13]. Such an object-separated representation is also pursued in this work.

Dreamer and Focus are both reward-driven and train a transition model and a reward model in an online RL framework. From reward-free and random policy trajectories, Sobal *et al.* [47] learn a PLDM. Their model is based on a joint embedding predictive architecture (JEPA) introduced by LeCun [31]. Just like Dreamer and Focus, it uses a latent space representation of the environment for planning with a dynamics model. However, a reconstruction of the input image is omitted, as this can lead to suboptimal features. Instead, a variance-invariance-covariance regularization (VIGReg) is used to discriminate the latent states and thus prevent the latent space from collapsing. To plan the latent states, the model predictive path integral (MPPI) control algorithm by Williams *et al.* [56] is used.

3.4 Symbolic State Representations

Advanced models like DreamerV3 operate within the framework of a Partially Observable Markov Decision Process (POMDP), characterized by probabilistic transitions and incomplete state information. By assuming full observability and deterministic action outcomes, the general POMDP framework simplifies to a classical planning problem (CPP). CPPs are typically formalized using logic-based languages such as the PDDL. This language provides a symbolic and discrete representation of the world through predicates and action schemas. This symbolic abstraction allows for the use of specialized, domain-independent planners for CPPs that make use of heuristics and search algorithms. To make use of CPPs in a planning framework, there are several approaches to derive a PDDL-like representation from raw, high-dimensional sensor data, such as images, directly.

One work that is closely related to our work is LatPlan, proposed by Asai and Fukunaga [4]. It uses images of classical planning domains to learn a discrete state representation with an IAM. They introduce a state autoencoder (SAE) that learns to map images to discrete state representations. For the discretization of the latent space, they use a Gumbel-Softmax activation function. After training the SAE, they learn two action model acquisition (AMA) systems. The first one (AMA_1) directly derives a PDDL instance from the discrete state representation. Therefore, they used the whole state space of the environment, which is, in many cases, not feasible and lacks generalization capabilities. Besides this, the work shows that VAEs are able to learn propositional features [4], which were used with an off-the-shelf PDDL-based planner to compute action sequences to transition from a given state to a goal state. The second one (AMA_2) uses an action autoencoder to predict the next state from the current state and action by combining an inverse action model and a forward model. The resulting forward model is used for planning by doing A* search in the latent space. We follow the idea of learning a discrete state and use graphs to take advantage of the generalization capabilities of GNNs. Among others, Asai and Fukunaga [4] use the 8-puzzle, 15-puzzle, and Sokoban as test environments, like we do.

Another approach that focuses more on learning an action model from images instead of the state representation is ROSAME-I, proposed by Xi *et al.* [59]. The paper introduces an end-to-end neuro-symbolic framework designed to learn lifted action models directly from visual traces, which are sequences of images and corresponding actions. The framework's goal is to jointly learn to predict states and infer the symbolic action models that govern the environment. Therefore, it combines a deep learning computer vision model with a symbolic reasoning component called ROSAME. ROSAME works by relaxing the symbolic action model into a probabilistic one, making the entire system differentiable and trainable with standard back-propagation techniques. The framework uses a vision model to calculate a symbolic state representation. Together with an action, it learns by predicting the next state from the current state along the whole trace. Only the final symbolic state of each trace is labeled and serves as a learning signal. This makes the data collection less expensive. The system represents the learning task as a classification problem over four possible cases for how a predicate relates to an action. Experiments conducted in various planning domains, such as Blocks World and Towers of Hanoi, show that the learned action models are highly accurate, often matching or even improving upon human-created models.

3.5 Research Gap

Existing reconstruction-based methods, such as AEs and VAEs, can capture visual structure but often encode task-irrelevant details and lack symbolic interpretability [5, 25]. Scene graph approaches introduce relational structure yet rely on predefined object categories and labels, preventing autonomous learning from raw images [28, 38, 61]. Latent dynamics models like Dreamer or PLDM learn predictive latent spaces but remain continuous and non-symbolic, limiting transfer and interpretability [19, 47]. Prior symbolic representation works, such as LatPlan, demonstrate the feasibility of discrete, PDDL-like encodings but depend on reconstruction and do not generalize to new instance sizes [4]. This thesis addresses these limitations by learning a reconstruction-free, graph-based symbolic state representation directly from images, combining object encodings in a relational structure to enable planning within a learned latent space.

4 Methods

Inverse action models learn a latent representation of an environment, thereby capturing its dynamics. Consequently, the learned representations can be used to model such dynamics. In this thesis, an IAM architecture is designed and employed to learn a symbolic representation from images of the classical planning domains N-Puzzle and Sokoban. The planning problems serve as prototypical examples, and the proposed method is largely independent of the environment. The proposed IAM architecture learns a symbolic, PDDL-like representation, which potentially allows the generation of action schemas. The learned representation and action schemas can be utilized to solve planning problems. Since the focus of this thesis is on learning the representation, the extraction of action schemas is omitted. Instead, a dynamics model is learned to demonstrate that the learned representation captures the dynamics of the environment and can be used for planning.

A state in a PDDL description is defined by a set of true atoms (also referred to as facts or grounded predicates). In this thesis, the formalism of a τ-structure is used to represent a state, which is part of the underlying logical foundation (first-order logic) for a state in the PDDL [24]. This more general formalism is chosen for its accessibility to potential future work. Only τ-structures with binary relations are considered, as they offer the advantage of being directly representable as a directed multigraph. This allows a straightforward usage of GNNs and leverages their generalization capabilities to larger problem instances, as they merely lead to an extension of the graph with additional nodes and edges. For the selected planning problems (Sokoban, N-Puzzle), such τ-structures do not impose a limitation, as their states can be fully described by binary predicates.

Subsequently, the architecture of the inverse action model is described. Its components and design decisions are discussed, and it is explained why the proposed architecture is suitable for addressing the research questions outlined in Chapter 1.

4.1 Inverse Action Model Architecture

The inverse action model presented here is a combination of an inverse model and a forward model, as described by Lesort *et al.* [32]. Figure 4.1 provides a schematic representation of the model's architecture. The state, observation, and action space of an environment (here N-puzzle and Sokoban) are referred to below as $\mathcal{S}$, $\mathcal{X}$, and $\mathcal{A}$, respectively. In addition to the action $a \in \mathcal{A}$, the number of steps $v \in \mathbb{R}$ (hereafter referred to as distance) between two states is used as a training target for the model. Together with the visual observations $\mathbf{X}_a$ and $\mathbf{X}_b \in \mathcal{X}$,

the tuple $(\mathbf{X}_a, a, \mathbf{X}_b, v)$ represents a data sample for training the model. The state observations $\mathbf{X}_a$ and $\mathbf{X}_b$ are image observations corresponding to true environment states $\mathbf{s}_a, \mathbf{s}_b \in \mathcal{S}$. They are provided as gray-scale images, so that $\mathcal{X} = [0,1]^{w \times h}$, where w and h are the width and height of the image, respectively. The parameters of the entire model are trained end-to-end using gradient descent with mini-batches.

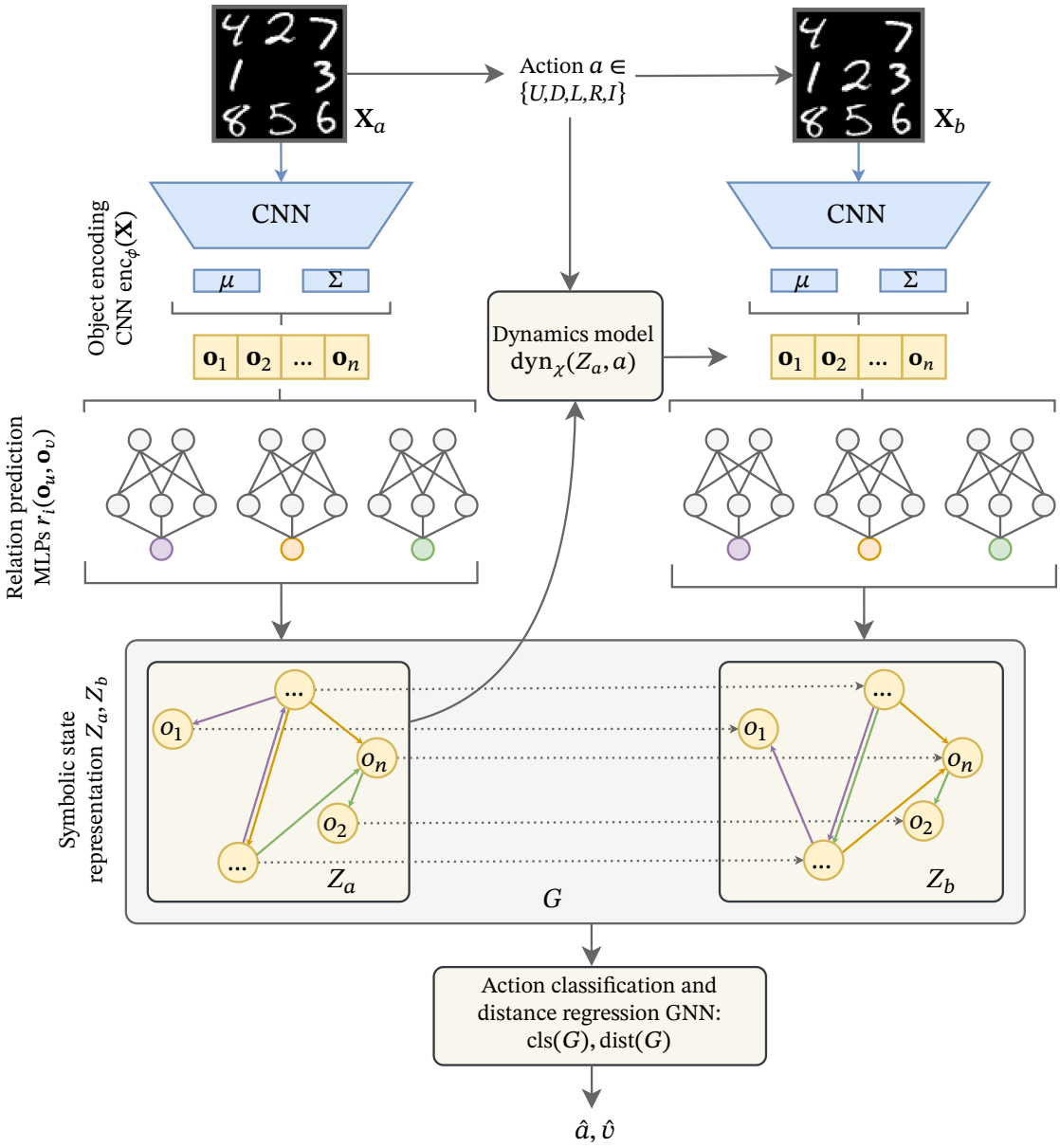


Figure 4.1: Model overview: The images of two states are mapped to the graph Z_a and Z_b. For this purpose, first an embedding is computed using the probabilistic encoding model enc_ϕ, then edges of different types are computed using a set of m relation predictors rel_i, for $i \in \{1, ..., m\}$. The graph G, merged from Z_a and Z_b, is used to predict the action $\hat{a}$ and the distance $\hat{v}$ between the two states. Additionally, the dynamics model dyn_χ predicts the successor state object encoding.

Therefore, both images $\mathbf{X}_a$ and $\mathbf{X}_b$ are first processed by a CNN and mapped into two matrices in the lower-dimensional vector space $\mathbb{R}^{n \times d}$. These matrices are interpreted as n objects in $\mathbb{R}^d$ each and are hereafter referred to as object encodings $\mathbf{O}_a$ and $\mathbf{O}_b$. The dimensions of these object encodings n and d are model hyperparameters.

The model contains m relation predictors in the form of MLPs. Each relation predictor predicts whether the respective relation between the objects holds for each pair of objects, and for both object encodings, respectively. The number m of relation predictors is a hyperparameter of the model. From the result of the relation predictors, the directed multigraphs Z_a and Z_b are constructed with n nodes and m edge types, respectively. These graphs are the intended symbolic latent representations of the states s_a and s_b.

The IAM's training objective is now to predict the action $\hat{a}$ and the distance $\hat{v}$ between the two corresponding states s_a and s_b, from their latent state representations Z_a and Z_b. Therefore, the graphs are first merged into a single one. Since the same encoder and relation predictors processed both $\mathbf{X}_a$ and $\mathbf{X}_b$, the nodes of Z_a and Z_b correspond to each other. The graphs are therefore combined into the graph G by adding a directed edge of a different type between the corresponding nodes of Z_a and Z_b. This graph, with initial object encodings $\mathbf{O}_a$ and $\mathbf{O}_b$, serves as an input to a GNN that predicts the action $\hat{a}$ and the distance $\hat{v}$. Additionally, a second GNN model is used to predict the object encoding $\hat{\mathbf{O}}_b$ from the state graph Z_a and action a. This GNN is hereafter referred to as the dynamics model.

A combined loss formulation, encompassing action prediction, distance regression, a regularization term for the object encodings, and the dynamics model, is used for training the model. A layer-by-layer description of each implemented component can be found in Appendix A.2.

The action space $\mathcal{A}$ used in this work for the N-puzzle and Sokoban environments is introduced in the following. In any given state of the environment, a subset of actions from $\mathcal{A}$ can be applied. All possible actions in the N-Puzzle can be understood as the movement of the blank position (cf. Section 4.2.1). In Sokoban, all possible actions are described by the movement of the player character (cf. Section 4.2.2). Consequently, in both environments, there are four valid actions: *up* (U), *down* (D), *left* (L), and *right* (R).

However, as explained in more detail in Section 4.3.1, this means that the blank or player position is sufficient to classify the corresponding actions correctly. To enable the IAM to represent more in the latent space than just this, the additional action, *invalid (I)*, is introduced. An invalid transition is defined as a transition where no action possible in $\mathbf{X}_a$ can be applied to obtain $\mathbf{X}_b$. The action space $\mathcal{A}$ is thus defined as

$$\mathcal{A} = \{U, D, L, R, I\}, \tag{4.1}$$

where U, D, L, R, and I are canonical unit vectors in $\mathbb{R}^5$. The set of valid actions is denoted by $\hat{\mathcal{A}} = \{U, D, L, R\}$. The inclusion of invalid transitions also enables the use of distances $v \neq 1$ as a regression target, thereby enriching the overall learning signal for the model. Due to these potentially non-causal transitions, temporal indexing is avoided, and instead a and b are used to denote the corresponding input states for the model.

The following provides a detailed description of the model components, starting with object encoding.

4.1.1 Object Encoding

Let $\mathbf{X} \in [0,1]^{w \times h}$ be a grayscale image observation with width w and height h, and let $\mathbf{u} \in \mathbb{R}^{n \cdot d}$ be a one-dimensional representation of an element of the object-encoding space. Inspired by other work that uses VAEs for SRL [17, 19], a CNN is used to model a conditional distribution $p(\mathbf{u}|\mathbf{X})$ over the object-encoding space, given an image observation $\mathbf{X}$. This allows us to introduce the prior distribution $p(\mathbf{u}) = \mathcal{N}(0, I)$ over the object-encodings. The prior distribution $p(\mathbf{u})$ helps to smooth and regularize the object-encoding space and could lead to better generalization capabilities over the input data [40]. By following Kingma and Welling [30], the conditional distribution $p(\mathbf{u}|\mathbf{X})$ over the object-encoding space is parameterized by a neural network that outputs the mean μ and a diagonal covariance matrix Σ for a normal distribution $\mathcal{N}(\mu, \Sigma)$. To guarantee a consistent computation graph for the gradient of mean and covariance, the reparameterization trick is used. Samples are drawn by calculating $\mathbf{u} = \mu + \Sigma^{1/2}\varepsilon$, where $\varepsilon \sim \mathcal{N}(0, I)$.

The VAE introduced by Kingma and Welling [30] uses a probabilistic neural network $p(\mathbf{X}|\mathbf{u})$ for reconstruction, which is used to derive the reconstruction loss in the ELBO. Since image reconstruction from an object encoding is not intended, the object encoding is not required to preserve visual details. Instead, two samples of two states are used to perform a classification and regression task. This leads us to replace the reconstruction loss of the ELBO by the classification and regression loss and keep the Kullback–Leibler divergence D_{KL} on the conditional and prior of the object-encoding space:

$$\mathcal{L}_{\mathrm{KL}}(\mu, \Sigma) = D_{\mathrm{KL}}\big(p_{\mu,\Sigma}(\mathbf{u}|\mathbf{X}) \| p(\mathbf{u})\big). \tag{4.2}$$

The samples $\mathbf{u}$ from the object-encoding space get reshaped to $\mathbf{O} \in \mathbb{R}^{n \times d}$ and get interpreted as n objects with each object encoding $\mathbf{o}_i \in \mathbb{R}^d$. The encoding network, sampling step, and reshaping operation are formalized by the function

$$\mathrm{enc}_\phi : [0,1]^{w \times h} \to \mathbb{R}^{n \times d}, \tag{4.3}$$

such that $\mathbf{O} = \mathrm{enc}_\phi(\mathbf{X})$, as schematically shown in Figure 4.2, where ϕ is the set of all learnable parameters of the encoding network. With $\mathbf{O}_a$ and $\mathbf{O}_b$, the object encodings of the two state

observations $\mathbf{X}_a$ and $\mathbf{X}_b$ are denoted, respectively. For larger problem instances, the number of objects required for modelling may vary. To apply the overall trained model to larger instances, the object encoder is retrained on new images and a larger number of objects n without retraining the rest of the model. This variation in the number of objects poses no problem for the remaining model components. The following describes the procedure for estimating whether a relation between two objects exists or not.

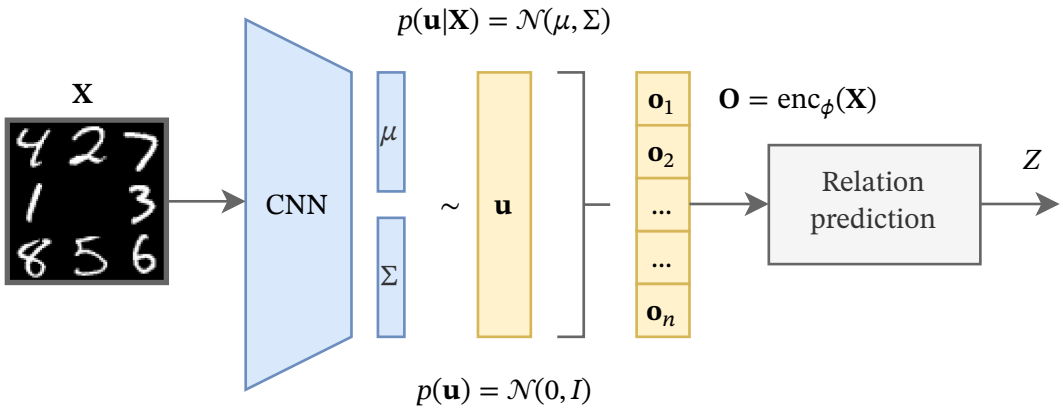


Figure 4.2: Schematic of the variational latent-space encoder. The diagram shows a convolutional neural network that maps the input image $\mathbf{X} \in [0,1]^{w \times h}$ to the distribution parameters (μ, Σ) of $p(\mathbf{u} \mid \mathbf{X})$, stochastic sampling of $\mathbf{u}$, and the reshaping into the object matrix $\mathbf{O} \in \mathbb{R}^{n \times d}$ with objects $\mathbf{o}_i \in \mathbb{R}^d$. The overall mapping from the input image to the object encoding matrix $\mathbf{O}$ is indicated by the function $\text{enc}_\phi(\mathbf{X}) = \mathbf{O}$.

4.1.2 Relation Prediction

The relations between objects that are introduced in the following correspond to the predicates of a PDDL description. Formally, this is equivalent to a finite relational structure, known in mathematical logic as a τ-structure, where τ is the signature of the predicate symbols [45]. In this thesis, this more general formalism is used to represent a state, as it provides a more flexible representation not strictly tied to the semantics of PDDL.

Let the set of m binary predicates $\tau = \{P_1, ..., P_m\}$ denote the signature of the τ-structure $\mathfrak{O} = \left(O, P_1^{\mathfrak{O}}, ..., P_m^{\mathfrak{O}}\right)$ with $O = \{o_1, ..., o_n\}$. The goal is to find the relations $P_1^{\mathfrak{O}}, ..., P_m^{\mathfrak{O}} \subseteq O \times O$ for a given state of the environment. Only binary predicates are considered. This restriction is necessary to enable the graph construction described below, and is only a limitation if the environment to be learned cannot be modelled with binary predicates. Since no meaning of the relations based on the environment should be enforced, an arbitrary but finite set of m learnable functions $\text{rel}_i : \mathbb{R}^d \times \mathbb{R}^d \to \{0,1\}$ that estimate whether the corresponding relation $P_i^{\mathfrak{O}}$ contains the tuple (o_v, o_w) for $v, w \in \{1, ..., n\}$ and $i \in \{1, ..., m\}$ is considered. The objects

$\mathbf{o}_i$ from a sample of the latent space distribution correspond to the objects in O and can be seen as an encoding of these objects. They are used as an input to the relation predictors rel_i.

As described in the next chapter, the predicted relations are used for graph construction. Therefore, the relation prediction is interpreted as a discrete classification task (edge or no edge between two objects in the graph), and the sigmoid output of the relation predictor rel_i is discretized in a probabilistic manner:

$$\text{rel}_i(a_i) = \begin{cases} 1 & \text{if } w_i < \sigma(a_i) \\ 0 & \text{otherwise,} \end{cases} \tag{4.4}$$

where $w_i \sim U[0,1]$ is uniform, $\sigma(\cdot)$ is the sigmoid function and $a_i = \text{MLP}_{\text{rel},i}(\mathbf{o}_v, \mathbf{o}_w)$ is the scalar output of an MLP. Instead of using a hard threshold here, using a sampling step lets the output of the relation predictor rel_i converge to either one or zero as training progresses. Let $\mathbf{R}_i \in \{0,1\}^{n \times n}$ be the binary output matrix of the i-th relation predictor for all object combinations. When needed, a and b are used as subscripts (e.g., $\mathbf{R}_{a,i}$) to refer to the output matrices of the two state observations $\mathbf{X}_a$ and $\mathbf{X}_b$, respectively. Furthermore, the sets of all respective output matrices per state observation are denoted R_a and R_b.

To allow backpropagation through this sampling step, we use a straight-through gradient estimator as proposed by Bengio *et al.* [7]. Therefore, the gradient of some loss $\mathcal{L}(\text{rel}_i)$ with respect to $\sigma(a_i)$ can then be estimated by:

$$\frac{\partial \mathcal{L}(\text{rel}_i)}{\partial \sigma(a_i)} \approx \frac{\partial \mathcal{L}(\text{rel}_i)}{\partial\, \text{rel}_i(a_i)}. \tag{4.5}$$

4.1.3 Graph Construction

Graphs are formally represented as tuples, where the first element is the set of nodes, and each subsequent entry is a set of edges for each relation type. The set of relations R_a and R_b are now represented as two directed multigraphs

$$Z_a = (V_a, E_{a1}, ..., E_{am}) \text{ and} \tag{4.6}$$

$$Z_b = (V_b, E_{b1}, ..., E_{bm}), \tag{4.7}$$

with $V_a = \{o_{1a}, ..., o_{na}\}$, $V_b = \{o_{1b}, ..., o_{nb}\}$, and

$$E_{ai} = \{(o_v, o_w) \mid \text{rel}_i(o_v, o_w) = 1 \text{ and } o_v, o_w \in V_a\}, \tag{4.8}$$

$$E_{bi} = \{(o_v, o_w) \mid \text{rel}_i(o_v, o_w) = 1 \text{ and } o_v, o_w \in V_b\}. \tag{4.9}$$

The combination of relation prediction and graph construction for each state observation is formalized by the function $\text{rel}_\theta : \mathbb{R}^{n \times d} \to \mathcal{G}_n$, which maps an object encoding to a directed multigraph with n nodes and m edge types. The set of all learnable parameters of all specified relation predictors is denoted by θ. The graphs Z_a and Z_b are graph representations of the intended latent state representation.

The same encoder and relation predictor are used for both state observations $\mathbf{X}_a$ and $\mathbf{X}_b$. Therefore, the rows and columns of the adjacency matrices of both latent state representations correspond to each other. This correspondence is used to combine Z_a and Z_b by adding a directed edge from every node in V_a to the corresponding node in V_b, as done by Yan *et al.* [60]. The resulting overall graph can be defined as follows:

$$G = (V_a \cup V_b, E_{a1} \cup E_{b1}, ..., E_{an} \cup E_{bn}, \{(o_{a1}, o_{b1}), ..., (o_{ak}, o_{bk})\}). \tag{4.10}$$

On this graph, a classification and regression task is performed using a message-passing R-GCN architecture as proposed by Schlichtkrull *et al.* [46], which supports different edge types.

4.1.4 Action Classification and Distance Regression

The architecture of Schlichtkrull *et al.* [46] does not accommodate edge weights and therefore lacks a corresponding gradient with respect to the edges. To enable backpropagation to the relation predictors, the two graphs, Z_a and Z_b, are considered fully connected, and the output of the relation predictor is used as edge weights. The node embedding update rule, including the edge weights of the R-GCN, is then given by

$$\mathbf{o}'_v = \Theta \cdot \mathbf{o}_v + \sum_{i=1}^{m} \sum_{w \in \mathcal{N}_i(v)} \frac{\text{rel}_i(\mathbf{o}_v, \mathbf{o}_w)}{|\mathcal{N}_i(v)|} \Theta_{\text{rel}_i} \cdot \mathbf{o}_w, \tag{4.11}$$

where $\mathcal{N}_i(v)$ is the set of neighbors of node $\mathbf{o}_v$ for relation rel_i and Θ_{rel_i} is the parameter matrix for relation rel_i. The weighting matrix Θ for self-edges is shared across all nodes and relations [46].

Several R-GCN layers with this update rule are now used to process G. Therefore, the missing edges that fully connect Z_a and Z_b are added to G, together with all corresponding edge weights (R_a, R_b). The latent-space encodings, which are also used for relation prediction, serve as initial node embeddings to make the nodes distinguishable from each other. To prevent a gradient update of the object encodings based on the message passing procedure, the stop-gradient operator $\text{sg}(\cdot)$ is used on the object encodings in the first R-GCN layer, thus stopping a gradient flow that bypasses the relation predictors (cf. Section 4.4). All node embeddings of the final R-GCN layer output are summed up to a graph-level output $\mathbf{g} \in \mathbb{R}^{d'}$, where the aggregation dimension d' is a model hyperparameter.

The construction of G as described in Equation (4.10), the insertion of the edge weights and missing edges, and the calculation of the graph-level output are formalized by the function

$$\text{comb}_\omega : \mathcal{G}_n \times \mathcal{G}_n \times \mathbb{R}^{n \times d} \times \mathbb{R}^{n \times d} \to \mathbb{R}^{d'}, \tag{4.12}$$

which combines the two directed multigraphs Z_a and Z_b to G and uses the initial node embeddings $\mathbf{O}_a$ and $\mathbf{O}_b$ to calculate the graph-level output $\mathbf{g}$. The set of all learnable parameters of this function is denoted by ω.

This graph-level output is then used to calculate a probability distribution over the action space and an estimate of the distance between $\mathbf{X}_a$ and $\mathbf{X}_b$. Two different MLPs are used as function approximators for the action classification and the distance regression:

$$\text{cls}_\psi : \mathbb{R}^{d'} \to \Delta^{|\mathcal{A}|}, \tag{4.13}$$

$$\text{dist}_\rho : \mathbb{R}^{d'} \to \mathbb{R}, \tag{4.14}$$

where $\Delta^{|\mathcal{A}|}$ is the $|\mathcal{A}|$-dimensional standard simplex. The cls-MLP is called action head and the dist-MLP is called distance head. The set of all learnable parameters of the action classification and the distance regression function is denoted by ψ and ρ, respectively.

The cross entropy and the mean squared error are used as loss functions for the action classification and the distance regression, respectively, and defined as

$$\mathcal{L}_{\text{cls}} = -\sum_{i=1}^{5} a_i \cdot \log \hat{a}_i, \tag{4.15}$$

$$\mathcal{L}_{\text{dist}} = \left(\frac{\lambda}{\lambda - 1} (\lambda^v - 1) - \hat{v} \right)^2, \tag{4.16}$$

with $\hat{a} = \text{cls}(\mathbf{g})$, $\hat{v} = \text{dist}(\mathbf{g})$, and $\lambda \in (0, 1)$ being a hyperparameter. The chosen formalization of distance loss as a geometric series is explained in Section 4.5.1. The real action and distance between $\mathbf{X}_a$ and $\mathbf{X}_b$ are denoted as $a \in \mathcal{A}$ and $v \in \mathbb{R}$, respectively. Both the action classification and the distance regression use a mean aggregation over the batch while training.

4.1.5 Dynamics Model

Together with the action classification model, a dynamics GNN model

$$\text{dyn}_\chi : \mathcal{G}_n \times \mathbb{R}^{n \times d} \times \hat{\mathcal{A}} \to \mathbb{R}^{n \times d}, \tag{4.17}$$

as shown in Figure 4.1, is trained. This model could be used to solve arbitrary planning problems with the learned state representation. For example, it is possible to use breadth-first

search with the dynamics model to find an action trajectory that transitions a state to a goal state.

The graph Z_a, the initial node embeddings $\mathbf{O}_a$, and a valid action $a \in \hat{\mathcal{A}}$ are used to predict the object encoding of the successor state $\hat{\mathbf{O}}_b = \text{dyn}_\chi(Z_a, \mathbf{O}_a, a)$. As with the action and distance prediction function comb_ω, the graph Z_a is treated as fully connected and uses the set of relations R_a as edge weights to apply the R-GCN and its update rule from Equation (4.11). Subsequently, the relation prediction model can be used to calculate an estimate for Z_b. The action a is one-hot encoded and concatenated to each node embedding. After multiple R-GCN layers, the node embeddings are concatenated and passed through an MLP to predict $\hat{\mathbf{O}}_b$. The set of all learnable parameters of the dynamics model is denoted by χ. This architecture is shown schematically in Figure 4.3.

The mean squared error between the object encoding $\hat{\mathbf{O}}_b$ predicted by the dynamics model and the object encoding $\mathbf{O}_b$ is used as a loss function to train the dynamics model:

$$\mathcal{L}_{\text{dyn}} = \| \text{sg}(\mathbf{O}_b) - \hat{\mathbf{O}}_b \|_F^2, \tag{4.18}$$

where $\text{sg}(\cdot)$ is the stop-gradient operator and $\| \cdot \|_F$ denotes the Frobenius norm. Since the dynamics model is trained simultaneously with the other model components, the loss is only calculated for valid actions, because successor states of invalid actions are not predictable. During training, the loss is averaged over the batch.

The loss is only used to calculate the gradient with respect to $\hat{\mathbf{O}}_b$. Propagating the gradients into the object encoding model with $\mathbf{O}_b$ from this loss is not done to mitigate a collapse of the representation. In Section 5.4, it is also demonstrated that it is possible to isolate the dynamics model from the other components completely and train it independently after an IAM without a dynamics model has been trained.

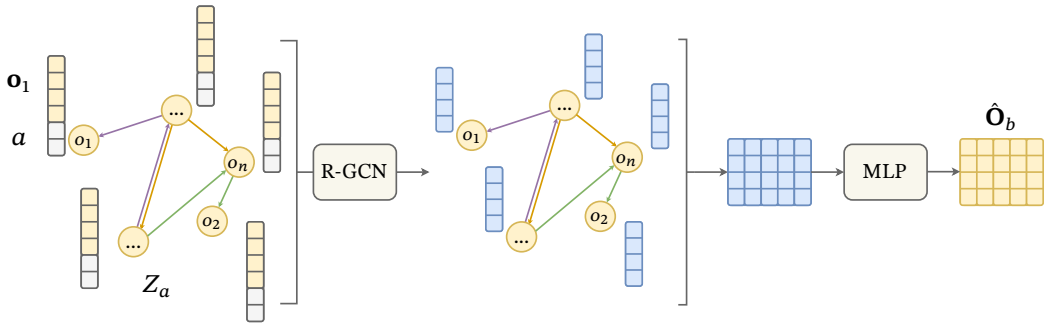


Figure 4.3: Dynamics model: The input is the multigraph Z_a with node embeddings $\mathbf{O}_a$, where the action a is concatenated to each node embedding. Multiple R-GCN layers produce updated node embeddings, which are concatenated and passed through an MLP to predict the successor object encoding $\hat{\mathbf{O}}_b$.

4.2 Environments

The proposed model is applied to two different environments, the *N*-puzzle and Sokoban. In general, the model is applicable to arbitrary classical planning domains whose actions can be modeled in an object-independent manner. The *N*-puzzle and Sokoban serve as prototypical examples and are described in the following section.

4.2.1 *N*-Puzzle

The *N*-puzzle is a sliding-tile game. It consists of a square board with $N + 1$ grid positions and N distinct tiles. One position is empty and is referred to as the blank position. The *N*-puzzle is commonly played on a 3×3 board, so $N = 8$. Larger variants include the 15-puzzle and the 24-puzzle. The objective is to arrange the tiles in the correct order by sliding tiles into the blank position, thereby swapping the position of the blank with the moved tile. Consequently, the possible actions can be described by moving the blank position to a neighboring cell, yielding the four actions *up*, *down*, *left*, and *right* ($\hat{A}$). In this work, the 8-puzzle is used for training, and the learned model is evaluated on the 15-puzzle to assess the generalization capability of the IAM. In total, the tiles can be arranged in $(N + 1)!$ different configurations. The state space of the *N*-puzzle can be represented as a graph whose nodes are the reachable puzzle states and whose edges correspond to the possible actions in each state. This graph consists of two disconnected components with the same number of nodes and edges. Depending on the defined goal state, the game can only be solved if the initial state lies in the same component as the goal state. This restriction is therefore relevant for planning. However, since the state representation is to be learned independently of any specific planning task, samples from the entire state space are considered in this work.

The grid structure of the game is exploited to encode the state space

$$\mathcal{S}_N = \left\{ s \mid s \in \mathbb{N}_0^{M \times M}, M = \sqrt{N+1}, s_{ij} \neq s_{vw} \text{ for } i \neq v \text{ or } j \neq w, s_{ij} \leq N \right\} \tag{4.19}$$

of the *N*-puzzle by square matrices. The blank position is represented by zero. Each tile is assigned an integer from 1 to N. For example, three consecutive states $s_1, s_2, s_3 \in \mathcal{S}_8$ with associated actions *up* and *left* can be represented by the three matrices

$$s_1 = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}, \quad s_2 = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 0 \\ 7 & 8 & 6 \end{bmatrix}, \quad s_3 = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 0 & 5 \\ 7 & 8 & 6 \end{bmatrix}.$$

For training, image observations are generated from these states, as shown in Figure 4.4.

4.2.2 Sokoban

Sokoban is also a grid-based game. In a given level layout, the player must push one or more boxes onto specified goal positions. The player controls a character that can move within the grid world. Boxes can only be pushed and not pulled. Thus, the action set is the same as for the N-puzzle ($\hat{\mathcal{A}}$). Different levels define different layouts with varying numbers of boxes. The objective is to push all boxes from their starting positions onto the goal positions in as few steps as possible. The number of possible Sokoban states depends on the level size, the number of boxes, and the number of wall elements. A combinatorial upper bound on the number of possible Sokoban states obtained by randomly placing the game elements is given by

$$\binom{X \cdot Y}{W} \cdot \binom{X \cdot Y - W}{b} \cdot \binom{X \cdot Y - W}{b+1}, \tag{4.20}$$

where X and Y denote the level width and height, W the number of wall elements, and b the number of boxes. Not all states generated in this way are solvable, irrespective of how the goal state is defined. In this work, levels with width and height $X = Y = 6$ are considered. For $b = 1$ and $b = 2$, there are approximately 1.15×10^{16} possible arrangements, which makes it computationally infeasible to provide an exact count of solvable states.

To describe the state space, a character encoding is used, allowing the state space to be represented as

$$\mathcal{S}_S = \left\{ s \mid s \in \{\#, @, +, ., *, \$, \text{' '}\}^{6\times 6} \right\}. \tag{4.21}$$

Here, # denotes a wall element, @ the player character, + the player on a goal tile, . an empty goal tile, \$ a box, $*$ a box on a goal tile, and the space character ' ' an empty cell. For example, three consecutive states $s_1, s_2, s_3 \in \mathcal{S}_S$ with associated actions *up* and *left* can be represented by the three matrices

$$s_1 = \begin{bmatrix} \# & \# & \# & \# & \# & \# \\ \# & & \# & & & \# \\ \# & . & \$ & & & \# \\ \# & & & @ & & \# \\ \# & & & \# & \# & \# \\ \# & \# & \# & \# & \# & \# \end{bmatrix}, \quad s_2 = \begin{bmatrix} \# & \# & \# & \# & \# & \# \\ \# & & \# & & & \# \\ \# & . & \$ & @ & & \# \\ \# & & & & & \# \\ \# & & & \# & \# & \# \\ \# & \# & \# & \# & \# & \# \end{bmatrix}, \quad s_3 = \begin{bmatrix} \# & \# & \# & \# & \# & \# \\ \# & & \# & & & \# \\ \# & * & @ & & & \# \\ \# & & & & & \# \\ \# & & & \# & \# & \# \\ \# & \# & \# & \# & \# & \# \end{bmatrix},$$

accordingly. As for the N-puzzle, image observations are generated from these states, as shown in Figure 4.4.

4.3 Training Data

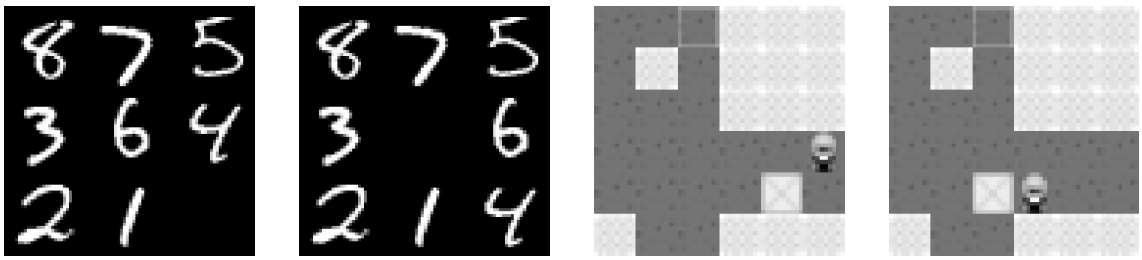
For training and evaluating the models, different datasets are used. For the N-puzzle, two dataset types are employed, each used to train different models. For Sokoban, datasets of only a single type are used, which consist of a predefined set of Sokoban levels together with several

suboptimal solutions for each. The dataset types differ in the degree and manner in which domain-specific knowledge is employed during the sample generation process.

For both environments, the dataset consists of a collection of grayscale images, denoted by $\mathbf{X}_a$, $\mathbf{X}_b \in [0,1]^{w \times h}$, together with corresponding action and distance labels $a \in \mathcal{A}$ and $v \in \mathbb{R}$. The images $\mathbf{X}_a$ and $\mathbf{X}_b$ serve as inputs to the model and constitute visual observations of the states s_a and s_b of the respective environment. They are generated by rendering each grid element with a fixed texture, ensuring that all entities, such as the numbered tiles in the N-Puzzle or the player and boxes in Sokoban, maintain a consistent appearance throughout the dataset. Observational noise, such as variations in lighting, texture, or viewpoint, is intentionally omitted. This simplification allows the investigation to focus on the fundamental feasibility of learning a symbolic state representation from image data with the proposed model architecture. Furthermore, this allows the assumption that the observation space has the same cardinality as the corresponding state space.

For the N-puzzle, $w = h = 80$ is used, and for Sokoban, $w = h = 278$. A sample is referred to as the tuple $(\mathbf{X}_a, a, \mathbf{X}_b, v)$. Since the previously described encodings of the state spaces are used to generate the data, a transition is referred to as the tuple (s_a, a, s_b, v) corresponding to a sample. A transition or sample is called invalid if $a = I$. Figure 4.4 shows an invalid sample for each environment.

For the four valid actions, $v = 1$ holds trivially. For invalid actions, the distance is greater than one and specifies the number of actions required to transform state s_a into state s_b. The number of actions is not necessarily minimal. This depends on the chosen sampling strategy (see Section 4.3.2). Invalid transitions offer the advantage of enriching the loss function with



$a = I, v = 2$ $\qquad$ $a = I, v = 3$

Figure 4.4: Examples of samples in the 8-puzzle (left) and Sokoban (right). The corresponding minimal action sequence is U, L for the 8-puzzle sample and D, L, L for the Sokoban sample. The target field for the box in Sokoban can be identified by the light border around the bottom element.

contextual information through distance labels, which is necessary for learning unique state representations. This is because not all distances can be uniquely determined solely by the change of the blank position. Furthermore, the distance between two states, as estimated by the model, can be used as a heuristic for planning and guided search (cf. Section 4.4.2). In

addition to distance estimation, modeling invalid transitions provides the advantage that some invalid transitions can only be distinguished from valid ones if the model incorporates more information than just the blank position. The reason for this will be explained below using the example of the 8-puzzle.

4.3.1 Invalid Transitions

When showing only valid samples to the model, the model might learn to represent only the blank position in the relations, without information about the position of the tiles, since this is sufficient to predict the four valid actions correctly. *Invalid* actions address this problem by forcing the model to capture information about tiles other than the blank position in the state representation. Let $T_{inv} = (s_a, I, s_b, v)$ with $v > 1$ be an invalid transition and $T_{suc} = (s_a, a, \hat{s}_b, 1)$ with $a \in \hat{\mathcal{A}}$ be a valid transition, both with the same initial state s_a. Then T_{inv} can be of two types:

1. The blank position in s_b is unequal to the blank position in $\hat{s}_b$.
2. The blank position in s_b is equal to the blank position in $\hat{s}_b$.

In the first case, the blank position in T_{inv} does not change or moves a Manhattan distance greater than one (e.g., Figure 4.5 invalid state left side). In the second case, other tiles may change the position (e.g., circular movements) while the blank ends up in a position equal to the blank position in $\hat{s}_b$. The invalid state on the right side of Figure 4.5 is of this type and can therefore only be distinguished from the valid transition on the left side in Figure 4.5 if at least one of the tiles marked red is identified. For training the IAM, this means that using invalid transitions of this second type, the model is forced to encode information about other tiles than the blank position in the state representation to be able to distinguish between valid and invalid transitions. It is thereby necessary to have *conflicting* valid and invalid transition

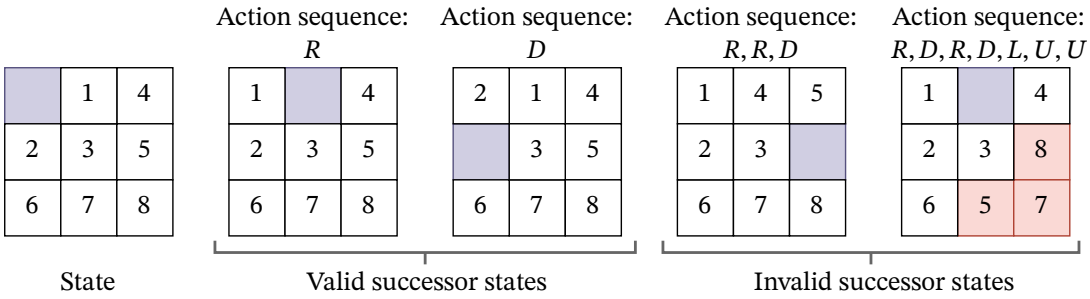


Figure 4.5: 8-puzzle transition: Possible transition from a state to a valid or invalid successor state. The blue box indicates the blank position. The two valid successor states can each be reached with one action. The blank position of the invalid transition on the left moved a Manhattan distance of three. For the invalid successor state on the right, the positions of tiles 5, 7, and 8 have changed, but the blank is at a valid position. This transition conflicts with the first valid successor state.

pairs in the dataset to learn a unique state representation for each state of the environment.

Definition 4.3.1 (N-Puzzle Conflict)**.** Two transitions $(\mathbf{X}_a, a, \mathbf{X}_b, v)$ and $\left(\mathbf{X}'_a, a', \mathbf{X}'_b, v'\right)$ are in conflict if the blank position of $\mathbf{X}_a$ and $\mathbf{X}_b$ is the same as the blank position of $\mathbf{X}'_a$ and $\mathbf{X}'_b$, respectively, and $a \in \hat{\mathcal{A}}$ and $a' = I$.

When the model learns to capture more information about the environment (e.g., one more tile) in the state representation, it becomes less likely that a conflicting transition pair is indistinguishable. This sampling issue results in a diminishing learning signal as more information is represented in the state representation. Without further investigation of the probability of indistinguishable conflicts when more information is represented, adding conflicts to the data set will enrich the state representations. This motivates choosing the relative proportion of invalid transitions of the second type such that the frequency of conflicts in the sampled data set is maximized. The relative proportion of invalid transitions of the second type is denoted by $p_{inv2} \in [0, 1]$.

Due to the grid world nature of the selected environment, invalid transitions of the second type always have an odd distance label. To use the distance estimate as a heuristic for planning, it is also necessary to predict even distances. This is achieved by using invalid transitions of the first type with even and odd distance labels. The ratio of invalid transitions of the first type in a dataset is denoted by $p_{inv1} \in [0, 1]$. The total ratio of invalid transitions is then given by $p_{inv} = p_{inv1} + p_{inv2} \leq 1$, and $p_{suc} = 1 - p_{inv}$ is the ratio of valid transitions.

Analogous to the N-puzzle, conflicts in the Sokoban dataset are also necessary to learn a state representation, since representing only the player character is sufficient for the IAM to predict the four valid actions. As the previously introduced definition of conflicts is not directly applicable to Sokoban, we extend it to Sokoban. In contrast to the N-puzzle, Sokoban has an explicit spatial structure that must be represented. It is therefore necessary for the IAM that a transition encode information about the spatial structure. Therefore, wall transitions are introduced in Sokoban. A wall transition is a valid action that does not change the player's position because the movement is blocked by a wall or a box element.

Definition 4.3.2 (Sokoban Conflict)**.** Let $P(\mathbf{X})$ and $B(\mathbf{X})$ denote the player and box positions in state $\mathbf{X}$. Two transitions $(\mathbf{X}_a, a, \mathbf{X}_b, v)$ and $\left(\mathbf{X}'_a, a', \mathbf{X}'_b, v'\right)$ can have a conflict of type

1. $P(\mathbf{X}_a) = P(\mathbf{X}'_a), P(\mathbf{X}_b) = P\left(\mathbf{X}'_b\right), B(\mathbf{X}_a) = B(\mathbf{X}'_a), B(\mathbf{X}_b) \neq B\left(\mathbf{X}'_b\right), a \in \mathcal{A}$, and $a' = I$ or
2. $P(\mathbf{X}_a) = P(\mathbf{X}'_a), P(\mathbf{X}_b) \neq P(\mathbf{X}_a), P\left(\mathbf{X}'_b\right) = P(\mathbf{X}'_a), a, a' \in \mathcal{A}$, and $a = a'$.

The first case is necessary to represent the positions of the boxes, and the second is necessary to represent the positions of the wall elements. A drawback of this definition is that conflicts of the second type can only occur for wall elements in the immediate surroundings of the player. If wall transitions over longer distances were also considered, the distance label would no

longer be uniquely predictable: the wall transition could occur at any point along the trajectory, or not at all (e.g., in cyclic movements).

In the N-puzzle, every action has an inverse, which could in principle also be used for training. Since this is not the case for Sokoban and, more generally, invertibility cannot be assumed for other environments, inverse actions are not considered. In the following, the specific sampling strategies for the training data for both environments are discussed.

4.3.2 Sampling N-Puzzle Data

Transitions in the N-puzzle are sampled in two different ways, which are denoted by partial-graph sampling (PGS) and breadth-first search sampling (BFS). They differ primarily in that BFS incorporates knowledge about the blank position in order to maximize the number of conflicts. Since PGS is intended to operate with a minimum of domain knowledge, the blank position is ignored. Nevertheless, domain knowledge is used to recognise whether two states are direct successors or not, in order to obtain either valid or invalid samples uniquely.

Partial-Graph Sampling

Starting from a state sampled uniformly over the entire state space $\mathcal{S}_N$, a random walk with a maximum of 100 steps is performed. The states and corresponding actions are stored as nodes and directed edges of a graph, respectively. States that have already been visited are not added to the graph again. The edges are labeled with the action and a step count of one. After k steps, a new state is sampled and the random walk restarts. The process continues until the specified number of valid transitions (number of edges in the graph) is reached. Invalid transitions of types one and two are then generated by adding edges to this graph. To this end, a random node is sampled uniformly from the graph and d random steps are taken along existing edges. The start and end nodes are then connected by an edge labeled with the invalid action and a step count d. The step count d is sampled from an exponential distribution and increased by two, i.e., $d = u + 2$ with $u \sim \text{Exp}(1/3)$. This yields a minimum of two steps and an average of five. It may occur that a valid transition exists between two nodes sampled in this way. Such cases are detected, and the transition is not added. For this sampling strategy, no distinction is made between invalid transitions of type one and type two, and only the ratio of valid (p_{suc}) to invalid transitions ($p_{inv} = 1 - p_{suc}$) is considered. The value of p_{suc} is chosen empirically to maximize the number of conflicts, as shown in Figure 5.1.

Breadth-First Search Sampling

As in PGS, a state is sampled uniformly at random from the state space. A breadth-first search of the state space with maximum depth 1000 is then performed, and the explored region is stored as a directed graph (states as nodes, actions as edges). The edges are labeled with the action and a step count of one, as in PGS. States that have already been visited are not added to

the graph. When the search depth is reached, a new state is sampled uniformly, and the search restarts. As with PGS, the process continues until the specified number of valid transitions is reached, and invalid transitions are generated by adding additional edges to the graph.

Invalid transitions of type two are added by sampling a random node s uniformly from the graph, and selecting a random valid successor s'. From s, a breadth-first search with maximum depth ten is started. As soon as a state $s'' \neq s'$ is found that has the blank in the same position as s', the search is terminated. The edge (s, I, s'', d) is then added to the graph, where d is the minimum number of steps from s to s'' resulting from the breadth-first search. Edges added in this way always have an odd distance label, and between s and s'' the blank has always moved by a Manhattan distance of one. Because the learned model is intended for planning, it is also necessary that the blank moves by a Manhattan distance not equal to one, so that even distance labels can be predicted. Therefore, additional invalid transitions of type 1 are generated, analogous to the invalid transitions in PGS. For both PGS and BFS, the resulting dataset consists of all edges of the graph, with the model inputs being the corresponding states (nodes) and the labels being the edge annotations. The ratio p_{suc}, p_{inv1}, and p_{inv2} that maximizes the number of conflicts is investigated in Section 5.1.1.

4.3.3 Sampling Sokoban Data

For the Sokoban domain, a sampling strategy distinct from those used for the N-Puzzle was required. If random traces were generated in Sokoban as in PGS, a box would only be moved very rarely. At the same time, optimal solution traces, such as those used in BFS, are not intended for use. The approach for Sokoban is therefore to utilize solution traces that emulate realistic, potentially suboptimal, solution trajectories, similar to those that might be observed from a human player.

Therefore, datasets with one and two boxes on a 6×6 grid were created. In total, 181.059 levels were generated. The levels were then solved using an A* search with probabilistic state transitions. More precisely, in random states during the search for a solution, the player was only provided with a random selection of all possible actions (those in which the player changes position). Additionally, in another random number of states, the player was provided with all actions, including those in which the player's position does not change (cf. Definition 4.3.2). In this way, five suboptimal solution traces were generated for each level. Because the search used a maximum depth of 20, not every level could be solved. From the solution traces, a set of valid and a set of invalid transitions were subsequently sampled. For invalid transitions, as in the N-puzzle, the distance was sampled as $v = u + 2$ with $u \sim \text{Exp}(1/3)$. An overview of the resulting base dataset is shown in Table 4.1. For training, valid and invalid transitions were then sampled independently and uniformly at random from this base dataset. We refer to this sampling strategy as trace sampling (TS). Because no environment knowledge was used for the Sokoban datasets, as with PGS, only the ratio of p_{suc} to p_{inv} can be adjusted to vary the

Number of boxes	Number of levels	Valid transitions	Invalid transitions	Total transitions
1	97.398	2.844.002	1.841.823	4.685.825
2	83.661	1.689.979	1.131.089	2.821.068

Table 4.1: Number of transitions in the Sokoban base datasets.

number of conflicts. The dependence of the number of conflicts on p_{suc} and $p_{inv} = 1 - p_{suc}$ was investigated empirically in Section 5.1.1.

In addition to the dependence on p_{suc}, p_{inv1}, and p_{inv2}, it was also investigated how the number of conflicts scales with the number of transitions for all three sampling strategies. It is assumed that the number of conflicts is proportional to the number of distinct transition pairs. The corresponding proportionality factor is determined in Section 5.1.1 for all three strategies and can be used as an estimate for larger datasets. This is useful because computing conflicts for large numbers of transitions is computationally expensive. The datasets and parameters used in the experiments are summarized in Table 5.2.

4.4 Model Training

The components explained in the preceding sections are assembled into an overall model that is trained end-to-end using backpropagation. To update the model parameters, the AdamW optimization algorithm is used, as introduced by Loshchilov and Hutter [34]. The end-to-end training scheme requires an overall loss function $\mathcal{L}$, which is utilized for the backpropagation step. It is composed of four weighted terms:

$$\mathcal{L} = \alpha\mathcal{L}_{\text{cls}} + \beta\mathcal{L}_{\text{dist}} + \gamma\mathcal{L}_{\text{dyn}} + \delta(\mathcal{L}_{\text{KL}}(\mu_a, \Sigma_a) + \mathcal{L}_{\text{KL}}(\mu_b, \Sigma_b)). \tag{4.22}$$

Here, the factors α, β, γ, and δ serve as hyperparameters to weight the individual loss terms with the constraint $\alpha + \beta + \gamma + \delta = 1$. During backpropagation, the gradient of the overall loss function $\mathcal{L}$ is computed with respect to the parameters of each model component:

$$\begin{aligned} &\text{Object encoding model: } \nabla_\phi\mathcal{L} && \text{Relation predictor: } \nabla_\theta\mathcal{L} \\ &\text{Dynamics model: } \nabla_\chi\mathcal{L} && \text{Action- \& Distance GNN: } \nabla_\omega\mathcal{L} \\ &\text{Action classification head: } \nabla_\psi\mathcal{L} && \text{Distance regression head: } \nabla_\rho\mathcal{L} \end{aligned} \tag{4.23}$$

A detailed formal derivation of the individual gradients is omitted here for the sake of clarity. Instead, Figure 4.6 illustrates the complete computation graph of the overall model, which shows the dependencies for the gradient calculation.

The dashed lines in Figure 4.6 illustrate that no gradient is backpropagated to the object encodings from the action and distance model comb_ω, and the dynamics model dyn_χ. A gradient

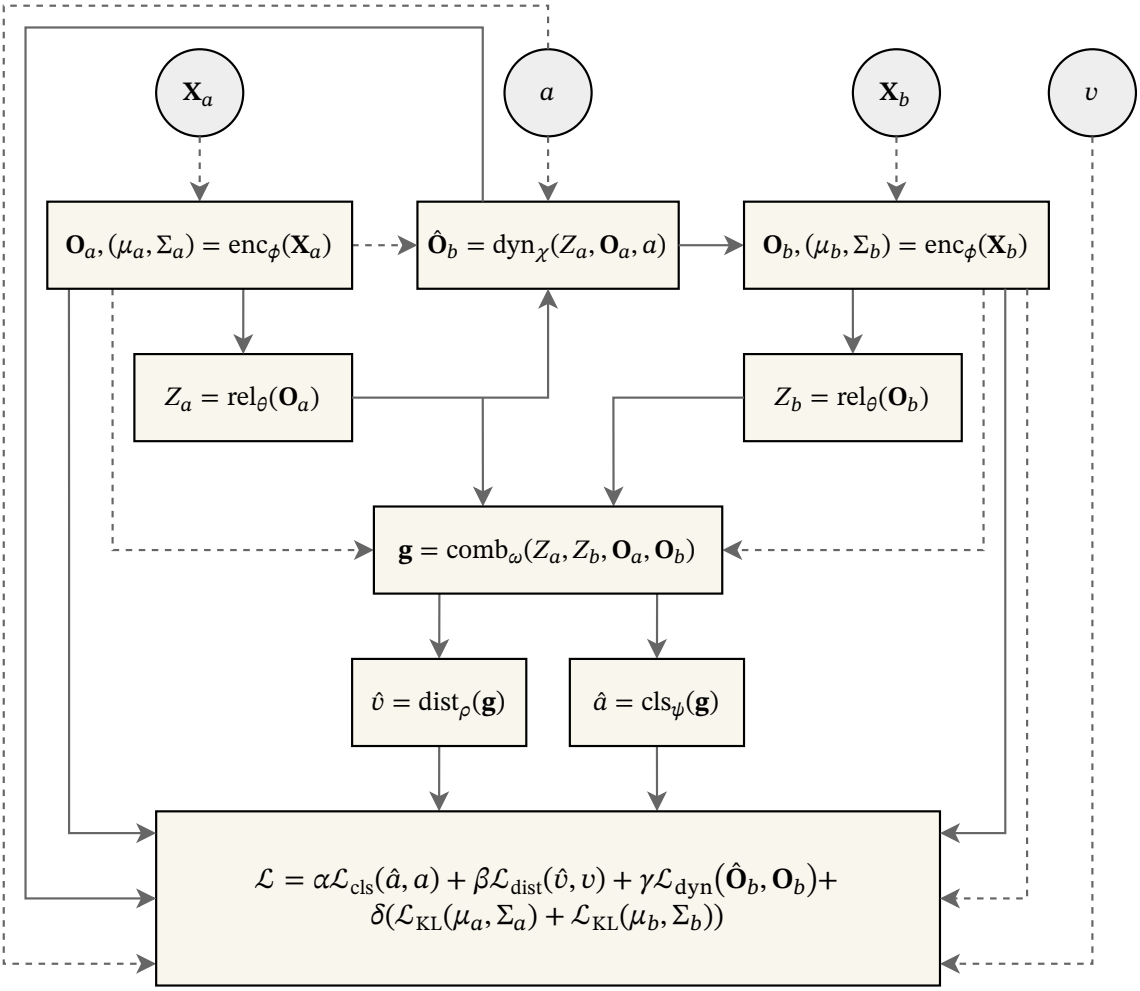


Figure 4.6: Computation graph of the overall model and its gradient flow. Solid edges indicate paths used for backpropagation, while dashed edges denote stop-gradient (detach) operations that block gradient flow. The solid line from the object encodings represents the gradient flow of the mean and variance through the KL-loss term. The dynamics loss propagates gradients only into the predicted object encoding $\hat{\mathbf{O}}_b$.

flow from the dynamics model back into the relation predictor was allowed to influence and structure the latent representation, to potentially better align with the environment dynamics. However, within the experiment in Section 5.4, it is shown that this gradient flow can be omitted without any particular disadvantages being found. The gradient is propagated via the assumed edge weights of the graph Z_a. To prevent a latent collapse (where, for instance, all edge weights have the same value), a careful balancing of the weighting factor γ relative to the other terms in the loss function is necessary. Similarly, the choice of δ has a significant impact on the structure of the object encodings. A relatively high value for δ would cause the encodings to approximate the prior distribution $p(\mathbf{u}) = \mathcal{N}(0, I)$ too closely. This could result in the encodings no longer containing any specific information from the input $\mathbf{X}_a$.

Algorithm 1 General training loop

Input:
Training dataset D_{train},
Untrained model components enc_ϕ, rel_θ, dyn_χ, comb_ω, cls_ψ, dist_ρ,
Weighting factors $\alpha, \beta, \gamma, \delta$,
Number of epochs E, batch size B, learning rate χ

Output:
Trained model components enc_ϕ, rel_θ, dyn_χ, comb_ω, cls_ψ, dist_ρ

1 **for** Epoche $e = 1$ **to** E **do**
2 **for** each batch in D_{train} **do**
3 $\mathcal{L} \leftarrow 0$ *Initialize batch loss*
4 **for** each element $(\mathbf{X}_a, a, \mathbf{X}_b, v)$ in batch **do**
5 $\mathbf{O}_a, (\mu_a, \Sigma_a) \leftarrow \text{enc}_\phi(\mathbf{X}_a)$ *Encode state*
6 $\mathbf{O}_b, (\mu_b, \Sigma_b) \leftarrow \text{enc}_\phi(\mathbf{X}_b)$ *Encode successor state*
7 $Z_a \leftarrow \text{rel}_\theta(\mathbf{O}_a)$ *Predict relations*
8 $Z_b \leftarrow \text{rel}_\theta(\mathbf{O}_b)$
9 $\mathbf{g} \leftarrow \text{comb}_\omega(Z_a, Z_b, \text{sg}(\mathbf{O}_a), \text{sg}(\mathbf{O}_b))$ *Combined graph encoding*
10 $\hat{a} \leftarrow \text{cls}_\psi(\mathbf{g})$ *Predict action*
11 $\hat{v} \leftarrow \text{dist}_\rho(\mathbf{g})$ *Predict distance*
12 $\mathcal{L} \leftarrow \alpha\mathcal{L}_{\text{cls}}(\hat{a}, a) +$ *Compute losses*
$\beta\mathcal{L}_{\text{dist}}(\hat{v}, v) +$
$\delta(\mathcal{L}_{\text{KL}}(\mu_a, \Sigma_a) + \mathcal{L}_{\text{KL}}(\mu_b, \Sigma_b))$
13 **if** a is valid **then** *Only for valid actions*
14 $\hat{\boldsymbol{\mu}}_b \leftarrow \text{dyn}_\chi(Z_a, \text{sg}(\mathbf{O}_a), a)$ *Predict dynamics*
15 $\mathcal{L} \leftarrow \mathcal{L} + \gamma\mathcal{L}_{\text{dyn}}(\hat{\boldsymbol{\mu}}_b, \mathbf{O}_b)$ *Add dynamics loss*
16 $\mathcal{L} \leftarrow \frac{\mathcal{L}}{B}$ *Average batch loss*
17 Compute gradients: $\nabla_\phi\mathcal{L}, \nabla_\theta\mathcal{L}, \nabla_\chi\mathcal{L}, \nabla_\omega\mathcal{L}, \nabla_\psi\mathcal{L}, \nabla_\rho\mathcal{L}$ *Backpropagation*
18 Update parameters: *AdamW optimization*
$\{\phi, \theta, \chi, \omega, \psi, \rho\} \leftarrow \text{AdamW}\big(\{\phi, \theta, \chi, \omega, \psi, \rho\}, \{\nabla_\phi, \nabla_\theta, \nabla_\chi, \nabla_\omega, \nabla_\psi, \nabla_\rho\}, \chi\big)$
19 **return** enc_ϕ, cls_ψ, dist_ρ, comb_ω, dyn_χ, rel_θ

Algorithm 1 outlines the training procedure used for the model. It serves as the foundational loop for all experimental setups, with only minor deviations for specific experiments. The model is trained using mini-batches. For each batch, the algorithm initializes a total loss $\mathcal{L}$ to zero (line 3) and accumulates the loss for each sample in the batch. This accumulated batch loss is then averaged (line 16) and used to compute the gradients with respect to the parameters of each model component via backpropagation (line 17). The parameters are subsequently updated using the AdamW optimizer (line 18).

Two key details are worth noting. First, the stop-gradient operator ($\text{sg}(\cdot)$) is applied to the object encodings in lines 9 and 14. This prevents gradients of the action and distance GNN model comb_ω and the dynamics model dyn_χ from flowing directly back into the encoder enc_ϕ.

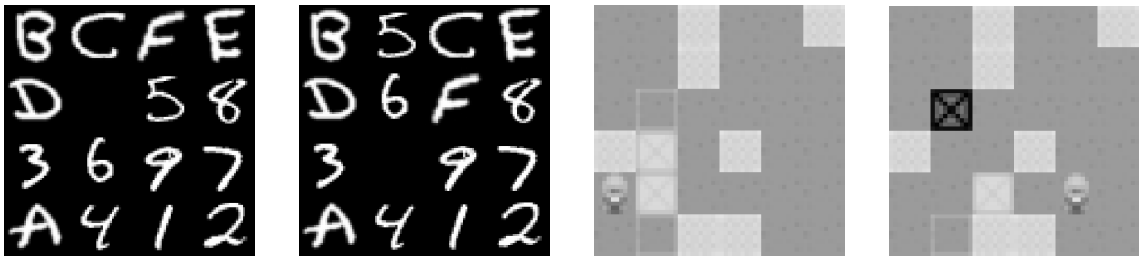
Second, the dynamics model dyn_χ and its corresponding loss term $\mathcal{L}_{\text{dyn}}$ are only computed for valid actions (lines 13-15). This is a crucial step, as invalid actions do not have a deterministic or unambiguous successor state, making a meaningful dynamics prediction impossible. The comprehensive training procedure detailed in Algorithm 1 is applied to all datasets identically.

4.4.1 Generalization

Both the *N*-Puzzle and Sokoban are classical planning domains. A key characteristic of such domains is that larger problem instances, like the 15-Puzzle or Sokoban with an additional box (SokobanB2), can be described by a larger set of objects while utilizing the same set of relations. This structural property motivates an investigation into whether the model components, particularly the relation predictors, can generalize from smaller instances to these larger, more complex problems. Examples of such larger instances are shown in Figure 4.7.

The central hypothesis is that the relation predictor, rel_θ, learns a set of rules for object relations that are independent of the total number of objects in the scene. Consequently, a model trained on the 8-Puzzle or SokobanB1 datasets should be adaptable to the 15-Puzzle and SokobanB2 domains, respectively.

To test this hypothesis, an adaptation strategy is employed. Since the object encoder enc_ϕ is architecturally bound to output a fixed number of objects n, it cannot be directly applied to instances requiring more objects. Therefore, the original encoder is replaced by a new instance, $\text{enc}_{\phi'}$, configured to output a larger number of required objects. The dimension of each object encoding, d, remains constant. This ensures that the interfaces to all other model components remain unchanged, allowing the parameters for the relation predictors rel_θ, the GNN comb_ω, and the action and distance head cls_ψ and dist_ρ to be frozen and reused without modification.



$a = I, v = 5$ $a = I, v = 8$

Figure 4.7: Examples of samples in the 15-Puzzle (left) and Sokoban with two boxes (right). The corresponding minimal action sequence is *R, U, L, D, D* for the 15-Puzzle sample and *R, U, R, U, R, R, D, D* for the Sokoban sample. The target field for the box in Sokoban can be identified by the light border around the bottom element. The right Sokoban state visualizes one box on a goal cell.

The adaptation process involves training only the new object encoder $\text{enc}_{\phi'}$ on the larger dataset (e.g., 15-Puzzle). This is achieved by following the general training procedure in Algorithm 1, but exclusively updating the new encoder's parameters ϕ' during the optimization step (line 18). Success in this transfer task would demonstrate that the model has successfully learned a generalizable, symbolic understanding of the environment's dynamics.

4.4.2 Guided Search

To evaluate the practical utility of the learned state representation Z and dynamics model dyn_χ for planning, a guided search algorithm is employed. Given an initial state observation $\mathbf{X}_t$ and a goal observation $\mathbf{X}_G$, the objective is to find a sequence of actions that transitions the environment from the start to the goal state. The learned distance predictor, dist_ρ, serves as a heuristic to guide this search process.

First, both the start and goal observations are mapped to their respective object encodings ($\mathbf{O}_t$, $\mathbf{O}_G$) and symbolic graph representations (Z_t, Z_G) using the trained encoder enc_ϕ and relation predictor rel_θ. The search then proceeds iteratively. At each step, beginning from the current state represented by ($\mathbf{O}_t$, Z_t), the algorithm performs rollouts of length $T = 3$ to evaluate potential action sequences. For each possible action sequence, the dynamics model dyn_χ and relation predictor rel_θ are applied recursively to predict the resulting state. The one-step update rules are therefore given by:

$$\hat{\mathbf{O}}_{t+1} = \text{dyn}_\chi(Z_t, \mathbf{O}_t, a), \quad a \in \hat{\mathcal{A}} \tag{4.24}$$

$$\hat{Z}_{t+1} = \text{rel}_\theta\big(\hat{\mathbf{O}}_{t+1}\big). \tag{4.25}$$

The distances between the predicted states for $t = T$ and the goal state, Z_G, are then computed using the GNN comb_ω and the distance regression head dist_ρ. The action sequence that results in the minimum predicted distance is identified as the locally optimal path. All actions of this optimal sequence are then executed, leading to a new state that becomes the current state for the next iteration of the search. When the goal is reached within these actions, the execution is terminated. This process is repeated until the goal is reached or an iteration limit is exceeded. A schematic of a single search step is depicted in Figure 4.8. For clarity, the figure illustrates a two-step rollout, though all experiments are conducted with the three-step horizon described.

Two key simplifications are adopted, reflecting the nature of the training data:

1. Goal Condition: The search terminates successfully if the search reaches a state that is one step away from the goal state. This is a necessary simplification for reaching the goal, as the model was not trained on samples with a distance of zero, which would correspond to identical states.

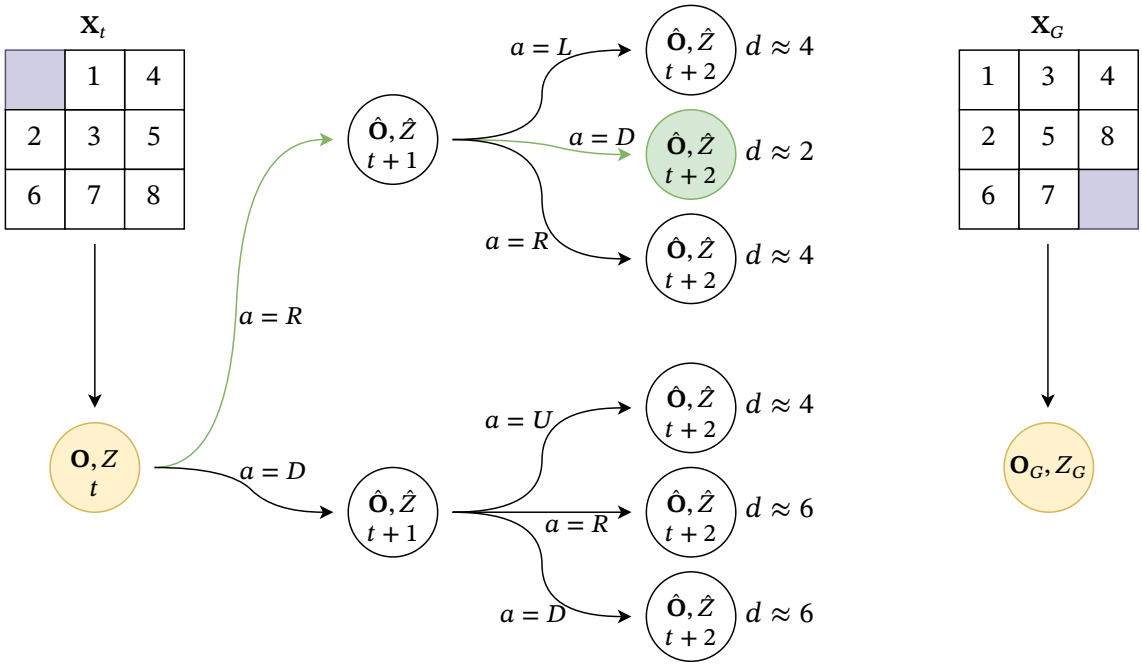


Figure 4.8: Schematic of a single guided search step, illustrating a two-step rollout. The initial state observation $\mathbf{X}_t$ (left) and the goal observation $\mathbf{X}_G$ (right) have a minimal distance of four steps. Both observations are mapped to their respective object encodings and graph representations. Using the initial state representation $(\mathbf{O}_t, Z_t)$, all possible two-step action sequences are then simulated with the dynamics model dyn_χ and the relation predictor rel_ϑ. The distance $d = \text{dist}_\rho(\text{comb}_\omega(\hat{Z}, Z_G, \hat{\mathbf{O}}, \mathbf{O}_G))$ to the goal is estimated for each final state of the rollouts. The first action of the trajectory that leads to the minimum predicted distance is selected for execution.

2. Action Applicability: During the rollouts, only actions that are applicable and result in a state change are considered. The model was not trained on transitions where a valid action does not alter the state (e.g., attempting to move the blank out of the grid in the 8-Puzzle), so such actions are excluded from the search space at each step.

It is important to note that these are no general limitations of the proposed architecture. Both scenarios could be addressed by augmenting the training data with zero-distance samples and self-looping transitions, respectively, which can be generated automatically without domain-specific knowledge. Due to the goal condition, it is sensible to execute all three actions of each rollout per iteration, as explained above. Executing only the first action can cause infinite loops, whereas executing all three increases the number of solved instances, albeit more suboptimally.

4.4.3 Tools and Frameworks

The presented approach and all related experiments are implemented in *Python 3.13* [49]. The *PyTorch* library [42] serves as the foundation for building individual model components and handling datasets. In order to work efficiently with graph structures, the *NetworkX* library

is used [37]. The GNN components are implemented with the PyTorch extension *PyTorch Geometric* [41]. The training procedure and data management are organized with the *PyTorch Lightning* framework [55]. For experiment tracking, monitoring, and hyperparameter optimization, the *Weights & Biases (W&B)* platform is utilized [54].

4.5 Evaluation

In the following subsections, the evaluation metrics used to assess all trained models throughout this thesis are defined. During the evaluation phase of the model, neither the object encodings nor the threshold in the discretisation step of the relation predictors are sampled, resulting in a deterministic model. Thus, $\mathbf{u} = \mu$ and $\omega_i = 0.5$ for all $i \in \{1, ..., m\}$, while evaluation.

4.5.1 General Metrics

To evaluate the performance of the IAM, two primary metrics are employed, corresponding to its main learning targets: action classification and distance regression.

For the action classification task, performance is measured using classification accuracy. While the model is trained using a cross-entropy loss function, accuracy provides an interpretable measure of predictive performance. It is defined as:

$$\mathrm{ACC}_a = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(a_i = \tilde{a}_i), \tag{4.26}$$

where N is the total number of samples, a_i is the ground-truth action, and $\tilde{a}_i$ is the predicted action. The predicted action is determined by the one-hot encoded argmax of the model's output vector $\hat{a}_i$.

For the distance function prediction, which is a regression task, the root mean squared error (RMSE) is used. This metric is a common choice for regression problems and is particularly suitable here because it corresponds directly to the error in the predicted number of actions between two states. The RMSE is calculated as:

$$\mathrm{RMSE} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(v_{i,\mathrm{target}} - \hat{v}_i\right)^2}, \tag{4.27}$$

where N is the number of samples, $v_{i,\mathrm{target}}$ is the target distance, and $\hat{v}_i$ is the distance predicted by the model.

Let $T = (s_a, I, s_b, v)$ denote an invalid transition and $v_{\min}$ the minimal number of actions required to transition from s_a to s_b. If $v = v_{\min}$, as is the case for all transitions sampled with BFS, then the target distance can be uniquely determined by the two states, and $v_{i,\mathrm{target}} = v = v_{\min}$. With PGS and TS, however, the distance labels can be suboptimal, i.e., $v \geq v_{\min}$, and the

distance between the two states can no longer be uniquely determined by the IAM, allowing for the interpretation of v as a random variable. The lowest RMSE can therefore only be achieved when the model learns to predict the expected value of v of all transitions that share the same v_{min} [50].

Since PGS uses a random walk to generate the transitions, the variance of v increases with higher v_{min}. Therefore, for transitions with greater v_{min}, the loss penalties are, on average, higher. The target distances with TS exhibit a similar behaviour. To account for this effect, the target distance is weighted for sampling strategies that produce suboptimal distance labels:

$$v_{i,\text{target}} = \sum_{k=1}^{v} \lambda^k = \frac{\lambda}{\lambda - 1}(\lambda^v - 1), \tag{4.28}$$

where $\lambda \in (0, 1)$ is a hyperparameter. This geometric series effectively discounts predictions for larger distances.

Similar to the value function prediction, the MSE is used for the loss of the dynamics model. The interpretation of the MSE in the object encoding space is not straightforward due to the stochasticity in the encoding space. Therefore, the dynamics model is evaluated by comparing the set of relation matrices R_b of the successor state with that of the predicted successor state $\hat{R}_b$, which is calculated by evaluating the relation predictor rel_θ on the predicted object encoding $\hat{\mathbf{O}}_b$ of the dynamics model dyn_χ. The accuracy of the correctly predicted relations is used as an evaluation metric:

$$\text{ACC}_d = \frac{1}{N} \sum_{N} \frac{1}{m \times n \times n} \sum_{k=1}^{m} \sum_{i,j=1}^{n} \mathbb{1}\big(R_{b,k,ij} = \hat{R}_{b,k,ij}\big), \tag{4.29}$$

where m is the number of relation predictors, n is the number of objects, and N is the number of samples.

4.5.2 Injectivity Metric

The tasks of action classification and distance prediction of the IAM serve to learn a function that maps environment observations to a latent, symbolic state representation. Ideally, this function represents an injective mapping, where each environment observation $x \in \mathcal{X}$ that corresponds to a unique state $s \in \mathcal{S}$ is mapped to a unique latent state representation $z \in \mathcal{Z}$. Within this work, each state observation corresponds exactly to one state, which is why the injectivity of the mapping $h : \mathcal{X} \to \mathcal{Z}, \mathbf{X} \mapsto \text{rel}_\theta\big(\text{enc}_\phi(\mathbf{X})\big)$, can be examined directly (cf. Section 4.3). Consequently, the cardinality of the observation space, denoted as $N = |\mathcal{X}|$, is considered equal to that of the state space $|\mathcal{S}|$. The cardinality of the latent space is denoted as $M = |\mathcal{Z}|$. To simplify the analysis, we further assume that the latent space is at least half the size of the observation space, i.e., $M \geq N/2$. This is not a general restriction, but it simplifies the explanation and allows for a simple mathematical description. Mappings that do not satisfy

this condition are considered insufficiently injective within the context of this work, making a more detailed consideration of such cases unnecessary.

As explained in Chapter 4.3.1, conflicts in the training dataset are crucial for learning unique state representations. However, since the number of these conflicts decreases as the state representation becomes more expressive, the learned function is not guaranteed to be strictly injective. To quantify the extent to which the learned mapping deviates from the property of injectivity, we introduce, similar to the definition of collision resistance in cryptography [43], the collision probability (CP).

Let $\mathbf{X}_1, \mathbf{X}_2 \in \mathcal{X}$ be two independent and identically distributed state observations with $\mathbf{X}_1 \neq \mathbf{X}_2$. Then $Z_1 = h(\mathbf{X}_1)$ and $Z_2 = h(\mathbf{X}_2)$ are the resulting latent state representations. The CP is then defined as

$$\mathrm{CP} = \mathbb{P}(Z_1 = Z_2), \tag{4.30}$$

and gives the probability that $Z_1 = Z_2$ for two randomly drawn samples. Since Z_1 and Z_2 are multigraphs, we say that they are equal if and only if their adjacency matrices are equal for all edge types, i.e., $\mathbf{R}_{1,i} = \mathbf{R}_{2,i}$ for all $i \in \{1, ..., m\}$ (cf. Section 4.1.2). An injective function has a CP of 0, whereas a function that maps all elements to a single latent element has a CP of 1.

For large state spaces, computing the mapping for all unique state observations to evaluate the CP is infeasible. Therefore, considering only a random uniformly distributed sampled subset $\hat{\mathcal{X}} \subset \mathcal{X}$ of the observation space, the CP can be estimated by

$$\mathrm{CP}_{\mathrm{est}} = \frac{|\{(\mathbf{X}_i, \mathbf{X}_j) \in \hat{\mathcal{X}} \times \hat{\mathcal{X}} | i \neq j, h(\mathbf{X}_i) = h(\mathbf{X}_j)\}|}{|\{(\mathbf{X}_i, \mathbf{X}_j) \in \hat{\mathcal{X}} \times \hat{\mathcal{X}} | i \neq j\}|}. \tag{4.31}$$

This approximation of the CP using $\hat{\mathcal{X}}$ is unbiased, i.e., $\mathrm{CP} = \mathbb{E}_{\hat{\mathcal{X}} \subset \mathcal{X}}[\mathrm{CP}_{\mathrm{est}}]$ (see proof in Appendix A.1). However, the estimated CP does not allow for an intuitive interpretation of the actual size of the latent space, as the exact distribution of multiple mapped elements in $\mathcal{Z}$ (how often which element is mapped in the latent space) is unknown. For a given size M of the latent space, however, two edge cases of the distribution can be considered, for which the CP becomes maximal and minimal:

1. **Maximum CP:** This case occurs when i elements from the state space are mapped to a single element in the latent space, while the remaining $N - i$ elements are mapped injectively and produce no collisions, thus $M = N - i + 1$. Formally:

$$\mathrm{CP}_{\max} = \frac{\binom{i}{2}}{\binom{N}{2}} = \frac{i(i-1)}{N(N-1)}. \tag{4.32}$$

2. **Minimum CP:** This case occurs when the mapping is as uniform as possible. Here, j elements in the latent space are each targeted by exactly two elements from the state space,

while the remaining elements are targeted only once, and thus $M = N - j$. Formally:

$$\mathrm{CP}_{\mathrm{min}} = \frac{j}{\binom{N}{2}} = \frac{2j}{N(N-1)}. \tag{4.33}$$

Conversely, for an empirically measured $\mathrm{CP}_{\mathrm{est}}$, we can estimate the bounds on the latent space size. By setting $\mathrm{CP}_{\mathrm{est}}$ equal to $\mathrm{CP}_{\mathrm{max}}$ and $\mathrm{CP}_{\mathrm{min}}$, we can solve for i and j, respectively. This yields a lower and upper bound on the cardinality of the latent space: $M_{\mathrm{max}} = N - j$ and $M_{\mathrm{min}} = N - i + 1$.

To provide an intuitive measure, we report the latent space ratio interval (LSRI) as the interval of these bounds relative to the observation space size:

$$\mathrm{LSRI} = [M_{\mathrm{min}}/N, M_{\mathrm{max}}/N]. \tag{4.34}$$

This interval quantifies the mapping's deviation from perfect injectivity. A higher CP yields a wider interval and a lower upper bound, indicating a more significant loss of unique state representations. On the other hand, as the CP approaches zero, the interval narrows towards $[1, 1]$, signifying a nearly injective mapping where $M_{\mathrm{min}} \approx M_{\mathrm{max}} \approx N$.

5 Results

This chapter presents the empirical evaluation of the proposed model. Initially, the experimental setup and dataset sampling parameters are outlined. Results on learning a useful state representation, assessing generalization to larger problem instances, and evaluating planning performance with the learned components are reported.

5.1 Experiment Setup

To ensure a robust and reliable evaluation, the methodology of Agarwal *et al.* [1] is adopted. Specifically, to account for stochasticity, each experimental setup involves multiple training runs with unique random seeds. This seed affects dataset sampling, model weight initialization, and all probabilistic processes within the IAM. The performance metrics from these runs are aggregated using the interquartile mean (IQM), with a 95% bootstrapped confidence interval to quantify statistical uncertainty. The IQM is employed as a robust aggregator as it mitigates the influence of outliers, to which the arithmetic mean is susceptible, while remaining more sensitive to the data distribution than the median [1].

An exception is the CP, which is aggregated using the arithmetic mean due to the potential sparsity of non-zero values for nearly injective functions. The LSRI is subsequently calculated from this aggregated mean CP. The dataset sampling parameters, summarized in Table 5.2, were chosen to maximize the number of conflicts as described in Section 5.1.1. Unless otherwise specified, the hyperparameters listed in Table 5.1 are used for all experiments. These were determined through a Bayesian hyperparameter search across 200 runs on the W&B platform, using PGS datasets only.

5.1.1 Dataset Sampling Parameters

As discussed in Section 4.3.1, invalid transitions are essential for learning the state representation, with the number of conflicts being a crucial factor in this process. The sampling strategies for the N-puzzle (BFS and PGS) and Sokoban (TS), as introduced in Section 4.3, generate different numbers of such conflicting transitions. The frequency of them is largely determined by the ratio between valid and invalid transitions. Therefore, an empirical investigation is conducted in the following for each sampling strategy to determine the number of conflicts as a function of the sampling parameters (p_{suc}, p_{inv1}, and p_{inv2}). For this investigation, 50 datasets, each containing $10,000$ transitions, were generated for various parameter configurations, and the number of conflicts was calculated.

Parameter	Symbol	Value
Optimizer	–	AdamW
Learning rate	η	1.38×10^{-4}
Batch size	B	200
Number of epochs	E	100
Classification loss weight	α	0.411
Distance loss weight	β	0.391
Dynamics loss weight	γ	0.198
KL loss weight	δ	5.18×10^{-5}
Distance discount	λ	0.95
Number of objects	n	9 (16)
Object dimension	d	25
Aggregation dimension	d'	128
Number of relations	m	4

Table 5.1: Model and training hyperparameters used in all experiments, unless otherwise noted. Values were selected via Bayesian optimization over 200 trials utilizing the W&B platform. Dataset sampling parameters are detailed separately in Table 5.2. The use of 16 objects is specific to the generalization experiments.

Furthermore, for a fixed parameter configuration, the number of conflicts was analyzed as a function of the total number of transitions in a dataset. It is assumed that the number of conflicts, N_c, is proportional to the number of distinct transition pairs ($N_c \propto n_t(n_t - 1)$ for n_t transitions). The corresponding proportionality factor was determined by applying a least squares fit to the IQM of the conflict counts. The proportionality factor enables the estimation of the number of conflicts in larger datasets, where an exact calculation would be computationally expensive.

8-Puzzle Partial Graph Sampling

In the PGS strategy, the two invalid transition types are not distinguished, as detailed in Section 4.3.1. Consequently, the number of conflicts depends solely on the proportion of valid transitions, $p_{suc} = 1 - p_{inv}$. The left plot in Figure 5.1 illustrates this relationship for p_{suc} values ranging from 0.0 to 1.0. The IQM of the conflict count reaches its maximum at $p_{suc} = 0.4$. Although the IQM for $p_{suc} = 0.5$ is nearly identical and their confidence intervals exhibit substantial overlap, a ratio of $p_{suc} = 0.4$ was selected for all subsequent experiments to maximize the expected number of conflicts.

For this selected ratio, the right plot in Figure 5.1 shows how the number of conflicts scales with dataset size. The resulting estimation for the number of conflicts, N_c, at $p_{suc} = 0.4$ is:

$$N_c \approx 1.6 \times 10^{-3} \cdot n_t \cdot (n_t - 1). \tag{5.1}$$

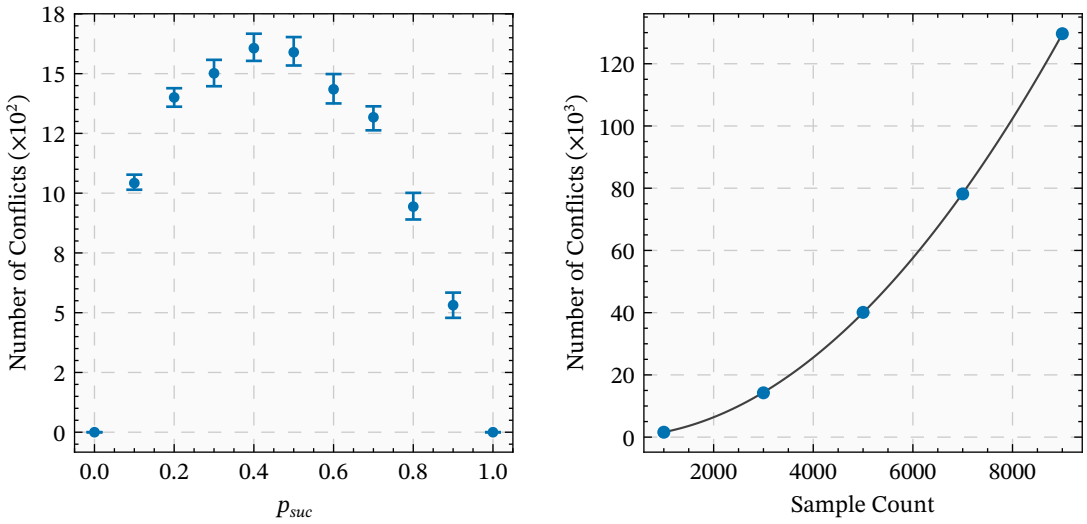


Figure 5.1: The IQM for the number of conflicts in the 8-Puzzle with PGS, based on 50 datasets for each parameter configuration with 10,000 transitions each. Left: Dependency of the conflict count on p_{suc}. The plot shows the IQM and its 95% bootstrapped confidence interval. Right: Scaling of the conflict count with dataset size for $p_{suc} = 0.4$. The plot shows the IQM and a quadratic fit, $N_c \approx 1.6 \times 10^{-3} \cdot n_t \cdot (n_t - 1)$.

8-Puzzle Breadth-First Search Sampling

In contrast to the PGS strategy, the BFS approach allows for the explicit generation of both invalid transition types, as detailed in Section 4.3.1. Consequently, the number of conflicts depends on the proportions of p_{suc}, p_{inv1}, and p_{inv2}. These parameters are constrained by $p_{suc} + p_{inv1} + p_{inv2} = 1$, defining a triangular region in the parameter space. The IQM of the conflict counts was evaluated over a grid of parameter values, indicated by the marked points in the left plot of Figure 5.2, and interpolated to generate the contour plot.

The analysis reveals that the maximum number of conflicts occurs when $p_{inv1} = 0.0$ and $p_{inv2} \in [0.4, 0.5]$. Increasing the proportion of type-one invalid transitions consistently decreases the total number of conflicts.

However, as discussed in Section 4.3.1, type-one invalid transitions are necessary for the model to learn to predict even-valued distances. A trade-off was therefore made to ensure a sufficient proportion of these transitions without significantly diminishing the conflict count. A final configuration of $p_{inv1} = 0.2$ and $p_{inv2} = 0.4$ was chosen, which implicitly sets $p_{suc} = 0.4$. For this configuration, the scaling of the conflict count with dataset size is shown in the right plot of Figure 5.2. Following the same methodology as for PGS, the proportionality factor was determined, allowing the number of conflicts to be estimated with

$$N_c \approx 7.25 \times 10^{-3} \cdot n_t \cdot (n_t - 1). \tag{5.2}$$

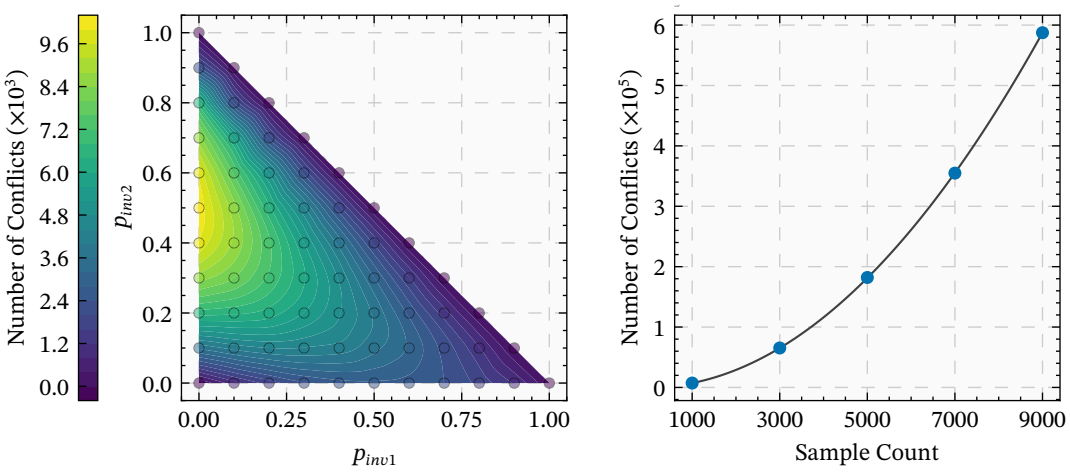


Figure 5.2: The IQM for the number of conflicts in the 8-Puzzle with BFS, based on 50 datasets for each parameter configuration with 10,000 transitions each. Left: Contour plot showing the number of conflicts in the 8-Puzzle depending on p_{inv1} and p_{inv2}. p_{suc} is given implicitly by $p_{suc} = 1 - p_{inv1} - p_{inv2}$. The number of conflicts was evaluated at the marked points, and a cubic 2D polynomial fit was used to compute the contour lines. Right: Scaling of the conflict count with dataset size for $p_{inv1} = 0.2$ and $p_{inv2} = 0.4$. The plot shows the IQM and a quadratic fit, $N_c \approx 7.25 \times 10^{-3} \cdot n_t \cdot (n_t - 1)$.

Sokoban Traces Sampling

The Sokoban environment features two distinct conflict types (cf. Definition 4.3.2), which are analyzed separately. With the TS strategy, the invalid transition types are not distinguished, similar to PGS, and the number of conflicts depends on $p_{suc} = 1 - p_{inv}$ only. The left and middle plots in Figure 5.3 show the number of type-one and type-two conflicts for p_{suc} values ranging from 0.0 to 1.0.

The number of type-one conflicts is maximized in the range of $p_{suc} \in [0.4, 0.5]$, whereas the frequency of type-two conflicts, which arise only from pairs of valid transitions, increases quadratically with p_{suc}. A ratio of $p_{suc} = 0.5$ is selected for subsequent experiments, as this value maximizes type-one conflicts while yielding a sufficiently large number of type-two conflicts.

The dependency of the conflict count on dataset size is also analyzed and visualised in Figure 5.3 (right) to determine the proportionality factor. For Sokoban with one box and the selected $p_{suc} = 0.5$ proportion of valid transitions, the number of conflicts can be estimated with:

$$N_c \approx 2.65 \times 10^{-5} \cdot n_t \cdot (n_t - 1). \tag{5.3}$$

All described results are summarized in Table 5.2.

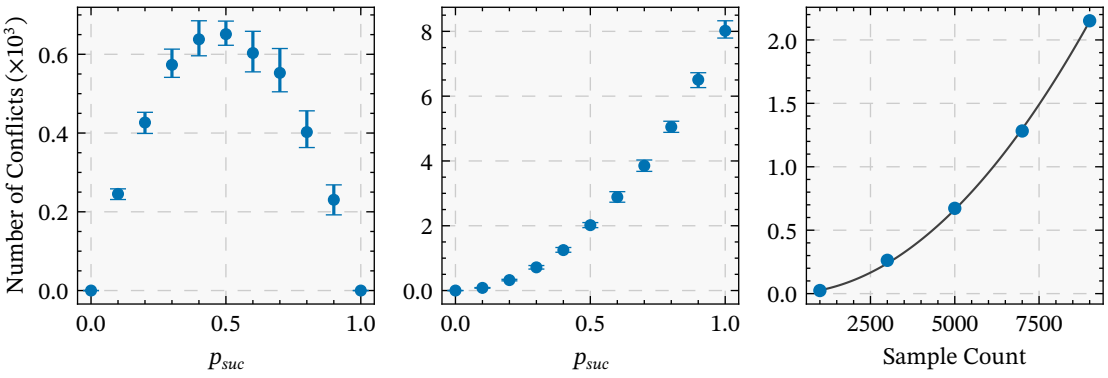


Figure 5.3: The IQM and 95% bootstrapped confidence interval for the number of conflicts for Sokoban with one box, based on 50 datasets with 10,000 transitions each. The left and middle plots show the dependency of type-one and type-two conflicts, respectively, on the proportion of valid transitions (p_{suc}). The right plot illustrates the scaling of the total conflict count as a function of the dataset size for a fixed ratio of $p_{suc} = 0.5$, together with a quadratic fit, $N_c \approx 2.65 \times 10^{-5} \cdot n_t(n_t - 1)$.

Environment	**Sampling**	$\mathbf{p}_{\text{suc}}$	$\mathbf{p}_{\text{inv(1)}}$	$\mathbf{p}_{\text{inv2}}$
8-Puzzle	PGS	0.4	0.6	-
8-Puzzle	BFS	0.4	0.2	0.4
15-Puzzle	PGS	0.4	0.6	–
15-Puzzle	BFS	0.4	0.2	0.4
SokobanB1	TS	0.5	0.5	–
SokobanB2	TS	0.5	0.5	–

Table 5.2: Overview of the sampling parameters for different dataset types. SokobanB1 and SokobanB2 refer to datasets with one and two boxes, respectively.

5.2 Analysis of the Learned State Representation

This section analyzes the IAM's performance in learning state representations for the 8-Puzzle and Sokoban domains. For each sampling strategy (PGS, BFS, and TS), 60 models were trained and evaluated to ensure robust results. The key parameters for the datasets, including training size, validation size, and the estimated number of conflicts, are detailed in Table 5.3. The models resulting from these strategies, hereafter referred to as PGS, BFS, and TS models, were evaluated using the performance metrics also presented in the table. This analysis extends beyond the quantitative metrics to qualitatively assess how the learned representations address the key contributions of this thesis formulated in Section 1.1.

Hereafter, the term "learned state representation" refers to the multigraph Z representing the latent state, as described in Section 4.1. A direct visual interpretation of these learned graphs proved challenging. Since no a priori meaning was assigned to the relations during training, the model developed an abstract encoding of the environment that does not directly

map to human-interpretable symbols. The state representation is therefore assessed using the performance metrics introduced in Section 4.5.1, supplemented by an action classification analysis using a row-normalized confusion matrix in Figure 5.4, where the diagonal elements directly correspond to the recall for each action label. The analysis begins by comparing the two sampling strategies for the 8-Puzzle before extending this comparison to the Sokoban domain.

Environment	8-Puzzle	8-Puzzle	Sokoban (one box)
Sampling Method	PGS	BFS	TS
Performance Metrics			
ACC_a [%]	98.6 [98.5, 98.7]	**99.94** [**99.93**, **99.96**]	96.6 [96.5, 96.7]
RMSE_d	2.3 [2.29, 2.31]	**0.47** [**0.44**, **0.5**]	0.79 [0.77, 0.81]
ACC_d [%]	75.6 [75.2, 75.9]	75.4 [74.7, 76.1]	75.6 [75.3, 75.9]
LSRI [%]	[100.0, 100.0]	[80.0, 99.9]	[100.0, 100.0]
Dataset Parameters			
Training Set Size	120,000	120,000	120,000
N_c	$\approx 23 \times 10^6$	$\approx 104 \times 10^6$	$\approx 382 \times 10^3$
Validation Set Size	20,000	20,000	20,000

Table 5.3: Performance metrics and dataset parameters for the IAM on the 8-Puzzle ($N = 9!$) and Sokoban with one box ($N \approx 2.06 \times 10^{14}$). Performance metrics are reported as the Interquartile Mean (IQM) over 60 independent training runs, accompanied by a 95% confidence interval derived from 5000 bootstrap repetitions. The reported LSRI is calculated from the arithmetic mean of the CP over all runs, as described in Section 4.5.2. The estimated number of conflicts N_c was calculated with the estimation formulas defined in Section 5.1.1.

Partial-Graph and Breadth-First Search Sampling

Action Classification Accuracy (ACC_a) serves as a primary indicator of the learned representation's quality. The BFS strategy achieves a nearly perfect IQM of 99.94%, suggesting its representations capture the necessary features (e.g., the positions of several tiles) to distinguish between actions. As shown in the confusion matrix (Figure 5.4), both the BFS- and the PGS models can reliably differentiate between valid actions. The primary source of error is the misclassification between valid and invalid transitions.

The superior performance of the BFS models is particularly evident in their handling of invalid actions, misclassifying less than 0.06% of them as valid. This high recall is likely due to the dataset's composition, where at least 66.6% of all invalid transitions are of type two. These challenging, ambiguous cases force the model to encode a substantial amount of environmental information to resolve them correctly.

In contrast, the PGS models misclassify a higher proportion of invalid actions, nearly 2%. Although the proportion of type two invalid transitions to all invalid transitions was not defined for dataset generation, it was calculated for the validation set to be 15.3%. The model's

ability to achieve a 98.02% recall on invalid actions indicates that it learns to encode more than just the blank's position. However, the higher error rate compared to BFS clearly demonstrates that it encodes less environmental information overall.

Theoretically, every invalid transition in the PGS dataset is distinguishable from a valid one since the PGS strategy (cf. Section 4.3.2) prevents any invalid transition from having a true minimal distance of one. The performance gap between the strategies, therefore, highlights that the proportion of conflicts is a critical factor for learning a meaningful representation, which is around four times higher in the BFS training dataset compared to the PGS training dataset. It is important to note, however, that higher accuracy does not necessarily imply a proportionally larger amount of encoded information, as the number of available conflicts for learning decreases exponentially as the representation captures more information.

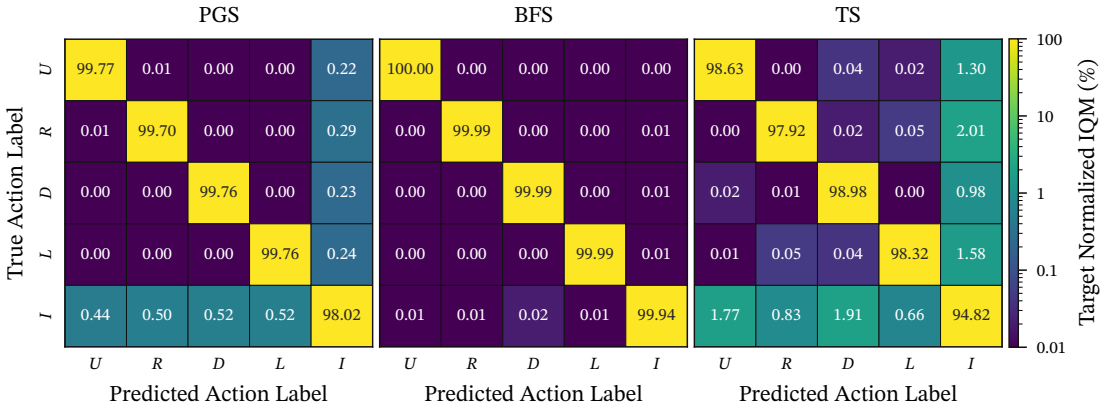


Figure 5.4: Confusion matrices illustrating the action classification performance for models trained with different sampling strategies (PGS and BFS for the 8-Puzzle, TS for Sokoban). Each matrix shows the row-normalized IQM of predictions for a given true label. Consequently, the diagonal elements represent the recall for each action class. The results demonstrate that all strategies distinguish well between valid actions (U, D, L, R), with misclassifications occurring primarily between valid and invalid actions.

Also, for distance prediction, the BFS strategy is the most accurate, achieving an RMSE of 0.47. This not only shows that models trained with the BFS strategy represent symbolic states distinctly but also that the model's distance head is capable of predicting the number of steps between two states. In contrast, the PGS strategy yields a considerably higher RMSE of 2.3. This difference can partly be attributed to the sampling strategy itself. While BFS models are trained on optimal, shortest-path distances, PGS models learn from non-unique, potentially suboptimal distance labels (cf. Section 4.3.2), making it necessary to compare the RMSE with the lowest possible RMSE, as described in Section 4.5.1. This best-case RMSE for PGS was empirically calculated from the validation dataset to have a mean of 1.39. The RMSE of the model's distance predictions must therefore be considered in comparison to this minimal error.

While the BFS model's RMSE of 0.47 directly represents its deviation from the optimal ground truth, the PGS model's deviation from its best-case RMSE is 0.91. This result indicates that the distance estimates from both PGS and BFS models are effective enough to be viable as heuristics in a guided search (cf. Section 5.4).

As previously argued, accurate action classification and distance regression require distinguishable state representations. It would therefore be expected that the mapping learned by the BFS model is also closer to the property of injectivity. However, the results show the opposite. The mappings learned by the PGS models are effectively injective, with an LSRI of [100.0%, 100.0%]. In contrast, the BFS models yield a less injective mapping, with an LSRI of [80.0%, 99.9%]. This means that for the worst-case mapping behaviour, the learned representation space collapses to only 80% of the observation space, and, in the best case, the learned mapping is only close to being injective. This counterintuitive result suggests two possibilities. First, strict injectivity of the symbolic graph structure may not be essential for high task performance. For example, the object encodings themselves, used as node features in the GNN comb_ω, could provide sufficient distinguishing information even when the graph structures collide. Second, the hyperparameter search was conducted exclusively on PGS datasets, which may have disadvantaged the BFS strategy in optimizing for injectivity and RMSE_d.

The model's strong performance on its primary training targets (ACC_a and RMSE_d) suggests that the IAM architecture is fundamentally capable of learning a complete, lifted symbolic state representation. The quality and structure of the training data are clearly a catalyst for the quality of this representation.

Interestingly, the accuracy of the dynamics model (ACC_d) remains consistently low at around 76% across all experiments. This suggests that predicting the exact successor object encoding is a fundamentally challenging task when the dynamics model is trained simultaneously with the other components. However, as supported by later findings, the model's performance can be significantly improved by fine-tuning it after the representation has converged (cf. Section 5.4).

Traces Sampling

The results for Sokoban are now discussed separately to evaluate the model's ability to generalize to a structurally different classical planning domain (cf. Section 1.1). Most performance metrics, such as accuracy and RMSE, are comparable to those from the 8-Puzzle, as both domains share the same grid-based action space and were trained with similar model parameters and distance label distributions. However, a direct comparison of the LSRI is infeasible due to Sokoban's intractably large state space. For its size, the combinatorial upper bound was assumed.

Sokoban exhibits the lowest action classification accuracy, with an IQM of 96.6%. This is attributable to two main factors. First, as seen in the confusion matrix in Figure 5.4, the model has more difficulty distinguishing between valid actions, which may be caused by ambiguous

”wall transitions” where an action results in no movement (cf. Section 4.3.3). Second, the dataset itself contains inherent ambiguity: 4.95% of its invalid transitions are indistinguishable from valid ones because they have a true minimal distance of one. This issue arises because the TS strategy allows for suboptimal traces.

Despite these challenges, the model performs well. It misclassifies only a small fraction (0.22%) of the inherently ambiguous invalid transitions, suggesting that the learned representation successfully encodes essential environmental information, such as player and box positions. The model's distance prediction performance further supports this conclusion, with an RMSE of 0.79. As with the PGS models, a zero RMSE is not possible because the distance targets are suboptimal, but they are potentially closer to zero, since the transitions are derived from goal-directed solution traces rather than pure random walks. Achieving this level of accuracy in distance estimation likely requires the representation to include information about static spatial elements, such as walls. Consistent with the findings for the 8-Puzzle, the Sokoban analysis strongly indicates that the sampling strategy, and thus the structure of the training data, is the most significant factor in the quality of the learned state representation. Crucially, these results demonstrate that the model architecture is transferable across domains. This is underscored by the fact that the Sokoban experiments required no domain-specific adjustments to the model's core components, such as the number of objects or relations.

The dynamics model for Sokoban performs similarly to its 8-Puzzle counterparts, achieving a low accuracy of 75.6%, which means a large majority of the edges in the predicted successor graph are incorrect. This result appears to be an artifact of the simultaneous training methodology rather than a fundamental flaw in the learned representation (cf. Section 5.4).

Finally, while the calculated LSRI of $[100.0\%, 100.0\%]$ for the TS models suggests their latent space mapping is injective, this result offers no meaningful insight into this property. The calculation relies on a combinatorial upper bound for the Sokoban state space, which is only a rough approximation of its true size, thereby artificially suppressing the underlying CP. A precise calculation of the true state space size is computationally expensive and was not undertaken as part of this thesis.

5.3 Generalizability Assessment

This section evaluates whether the relation predictors learned in the previous section have captured general domain properties. In this context, general domain properties refer to a PDDL-like structure, where predicates are retained across problem instances with a different number of objects. Therefore, the models previously trained on the 8-Puzzle and Sokoban with one box are now evaluated on the 15-Puzzle and Sokoban with two boxes.

Following the methodology from Section 4.4.1, the trained model components are reused and frozen. Since the new instances' images are outside the training distribution and contain a different number of objects, a new image encoder is trained. The dynamics model is omitted from this assessment due to its poor performance. All sampling parameters and hyperparameters remain unchanged, except for the number of objects, which is set to 16. The key performance metrics are summarized in Table 5.4, with a detailed analysis of the action classification performance provided by the confusion matrices in Figure 5.5.

Environment	15-Puzzle	15-Puzzle	Sokoban (two boxes)
Sampling Method	PGS	BFS	TS
ACC_a[%]	73 [69, 76]	48 [45, 51]	87 [86, 88]
$RMSE_d$	1.95 [1.93, 1.96]	1.49 [1.44, 1.56]	0.91 [0.90, 0.92]
LSRI [%]	[< 50.0, 99.9]	[< 50.0, 99.8]	[100.0, 100.0]

Table 5.4: Performance metrics for the IAM on the 15-Puzzle ($N = 16!$) and Sokoban with two boxes. Performance metrics are reported as the IQM over 60 independent training runs, accompanied by a 95% confidence interval derived from 5000 bootstrap repetitions. The reported LSRI is calculated from the arithmetic mean of the CP over all runs, as described in Section 4.5.2.

Partial-Graph Sampling

The PGS-trained models exhibit a notable decline in performance on the 15-Puzzle, achieving an overall action classification accuracy of only 73% for PGS. The confusion matrix reveals that while valid actions can be distinguished from one another with sufficient robustness, the primary source of error is the misclassification of valid transitions as invalid. Approximately half of all valid transitions are incorrectly classified as invalid. Conversely, invalid transitions are identified with a high recall of 92.6%, with only about 7.3% being misclassified as valid. The validation set contains 11.5% type-two invalid transitions, which is higher than the 7.3% error rate, but this does not suggest that the model encodes more than just the blank position. This is because the models appear to have developed a bias: since 60% of the training data consists of invalid transitions, misclassifying valid transitions randomly as invalid likely incurs a smaller penalty to the overall loss than the alternative.

Regarding distance prediction, the model achieves an $RMSE_d$ of 1.95, which is better than its performance on the 8-Puzzle. This can be attributed to the structure of the 15-Puzzle, since there are, on average, more actions possible per state than in the 8-Puzzle. Consequently, a random walk, as employed by the PGS strategy, tends to generate traces that are closer to the minimal distance. The empirically calculated best-case $RMSE_d$ on the validation set is 1.28 (15-Puzzle), confirming this trend. For comparison, predicting the Manhattan distance of the blank position between the two states in a transition yields a much higher $RMSE_d$ of 4.28. This indicates that the learned structure generalizes better for distance prediction than for action classification.

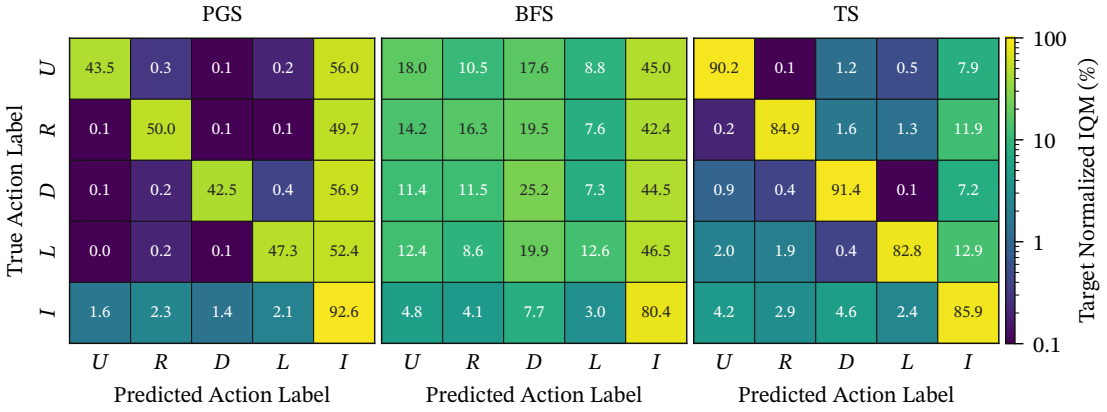


Figure 5.5: Confusion matrices for the generalization assessment, evaluating models on larger problem instances: the 15-Puzzle (for PGS and BFS strategies) and Sokoban with two boxes (for the TS strategy). Each matrix displays the row-normalized IQM. Therefore, the diagonal elements represent the recall for each action class. The performance of the BFS models degrades significantly, with a high rate of misclassification. In contrast, the TS model demonstrates more robust generalization to the Sokoban environment with two boxes.

Despite this partial success, the learned state representation fails the test of injectivity. The reported LSRI of $[< 50.0\%, 99.9\%]$ indicates that, in the worst-case mapping scenario, the latent space collapses to less than half the size of the observation space. This means a significant number of distinct environment states could be mapped to the same latent representation. Therefore, the learned state representations do not meet the requirement for a suitable representation due to their lack of injectivity. Nevertheless, it cannot be ruled out that the learned relation predictors generalize, as they clearly encode part of the relevant domain information (action and distance labels), and it may be a methodological problem to learn the object encodings correctly.

Breadth-First Search Sampling

With an action classification accuracy of 48%, the newly trained BFS models perform the worst. The confusion matrix clearly shows that the models are unable to distinguish between different valid actions. The highest recall is achieved for invalid transitions at 80.4%. The fact that approximately 45% of valid transitions are classified as invalid suggests the model has learned a simple rule to distinguish valid from invalid transitions but has failed to capture the structure of the valid transitions. This strongly indicates that the learned relations are too specific to the 8-Puzzle and do not generalize to larger instances.

This conclusion is supported by the RMSE_d of 1.49 for distance prediction. The model's predictions are only marginally better than the mean of the distance labels in the validation set (1.64), implying that it has largely memorized the label distribution rather than learning a

meaningful state representation. This is further reflected in the test of injectivity, where the reported LSRI of $[< 50.0\%, 99.8\%]$ is slightly worse than that of the PGS models, indicating a more significant collapse of the latent space, which aligns with the overall poor performance.

Traces Sampling for Sokoban

In contrast, the TS-trained model demonstrates the best generalization performance, achieving an action classification accuracy of 87% on Sokoban with two boxes. As shown in the confusion matrix, the model is able to distinguish between valid actions to some degree (misclassification $\leq 4.3\%$), despite the inherent ambiguity introduced by "wall transitions" (cf. Section 4.3.3). This indicates that the state representation successfully captures the information necessary to differentiate between actions even in a more complex environment. Similarly, the RMSE_d for distance prediction is the lowest among the generalization experiments, although it is higher than in the one-box case.

The superior performance on Sokoban is expected. The input grid size remains constant (6×6), and only one additional game element (a second box) is introduced. This is a smaller change in the input distribution compared to the transition from the 8-Puzzle to the 15-Puzzle, which involves a larger board and seven new, visually distinct objects. Therefore, the Sokoban results are not directly comparable to those from the N-Puzzle, and they alone do not allow for a conclusion on the generalizability of the relation predictors. As in the one-box scenario, the LSRI is not meaningfully interpretable due to the use of a combinatorial upper bound for the state space size.

In summary, the analysis suggests that the proposed model architecture is capable, in principle, of learning generalizable, lifted representations. However, the significant performance disparity between the PGS and BFS strategies indicates that either the methodology for adapting the object encodings needs refinement or the properties of the initially learned representation are critical for successful generalization.

5.4 Planning Assessment

This section validates the learned state representation by assessing its practical utility for planning, thereby addressing the key contribution of whether the representation effectively encodes the environment's dynamics.

5.4.1 Experiment Setup

The following experiment and evaluation are conducted solely for the 8-puzzle environment using the PGS and BFS sampling strategies. Due to computational cost, only a single model is trained for each strategy. This approach is justified as the primary objective is to demonstrate the general feasibility of a guided search with the learned components, rather than performance

optimization. Therefore, the reported metrics are direct measurements from the two models, without aggregation or confidence interval specification unless otherwise specified.

The evaluation in Section 5.2 shows that simultaneously training the dynamics model with all other components yields a suboptimal dynamics model. To address this, a two-stage training procedure is done for this assessment. First, two new IAM instances, one for each sampling strategy, are trained from scratch on a dataset of 500,000 samples. The larger dataset is chosen to ensure that the distance prediction generalizes sufficiently well across the entire observation space. During this stage, the dynamics model is excluded from training by setting its corresponding loss weight, γ, to zero, and rescaling the remaining weights to ensure they sum to one.

Subsequently, the parameters of the trained IAM components are frozen, and only the dynamics model is then trained, also using 500,000 transitions. Therefore, only the dynamics model's parameters χ get updated, using a loss function where only the dynamics term is active ($\gamma = 1$ and $\alpha = \beta = \delta = 0$). This sequential training approach ensures that the dynamics model is trained on a stable latent state representation.

The fully trained models are then employed in a guided search, as described in Section 4.4.2. The evaluation is performed on 100 unique start-goal pairs for each optimal distance between two and ten. The search algorithm tries to find a solution path for each instance within 20 rollouts. The following evaluation examines the overall planning performance and discusses the roles of the contributing components: the dynamics model and the distance prediction heuristic. The evaluations of the PGS model's distance predictions are done with respect to the discounted target labels on which it was trained (cf. Section 4.5.1).

5.4.2 Results

The overall planning performance is depicted in Figure 5.6, which shows the guided search's success rate as a function of the problem's optimal solution length. Since the search terminates successfully when it transitions to a state one step from the goal (cf. Section 4.4.2), the optimal distance in the figure is the minimum number of actions required to reach such a state.

The plot indicates a performance distinction between the two strategies. The BFS-trained model is effective on short-horizon problems, solving nearly 100% of all instances with an optimal distance of up to four. Beyond this, its performance steadily declines, reaching a success rate of approximately 18% for problem instances requiring at least nine actions. In contrast, the PGS model performs consistently worse. A performance gap of around 2 to 3% is already noticeable at a distance of two to three. For higher optimal distances, the performance further drops and reaches a success rate of 9% at a distance of nine. This decay in performance for both models can partially be attributed to the training data distribution. As the optimal distance

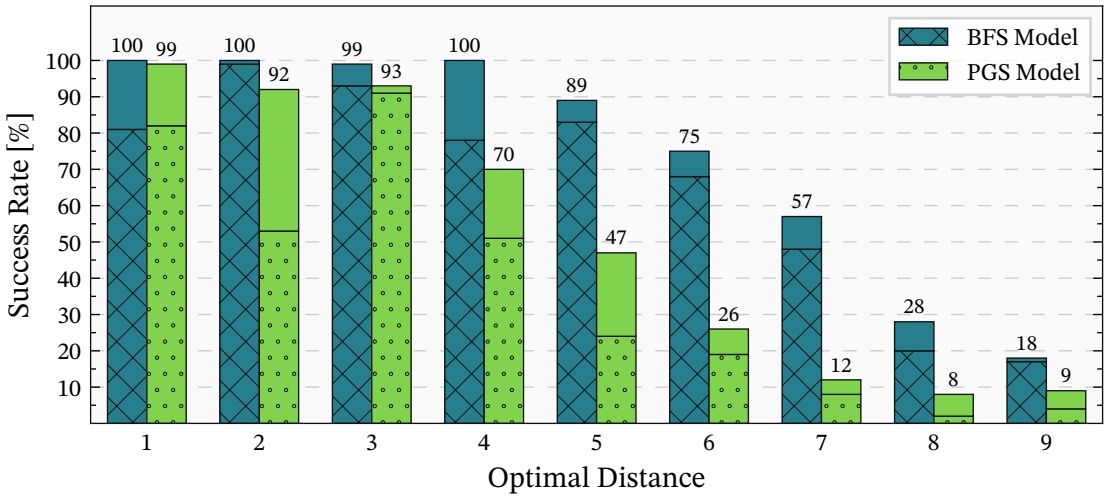


Figure 5.6: Success rate of the guided search for the BFS and PGS models on the 8-Puzzle, plotted against the optimal solution length. The bars represent the percentage of 100 problem instances solved within 20 rollouts. The hatched area indicates the proportion of optimally solved instances.

increases, the number of corresponding samples decreases, providing a weaker learning signal for long-horizon predictions.

To further investigate the causes of the performance decline at longer distances and the overall poorer performance of the PGS model, the following analysis will examine the accuracy of the dynamics model and the distance-prediction model's errors.

Dynamics Model Accuracy

During a planning rollout, the dynamics model predicts a sequence of latent states, with each prediction based on the previous one. Therefore, inaccurate predictions accumulate and can degrade the quality of the rollout, potentially misleading even a perfect distance-prediction model.

The multi-step prediction accuracy (ACC_d) of the dynamics model is shown in Figure 5.7. The results are plotted against the prediction horizon, representing the number of sequential actions taken from an initial state. As the figure illustrates, the accuracy of both models decreases approximately linearly with increasing prediction horizon. While the BFS model has a higher accuracy after a one-step prediction, its performance declines more steeply than that of the PGS model.

For the three-step rollouts used in the guided search, however, the BFS model's accuracy is higher. After three steps, it achieves an ACC_d of 98%, whereas the PGS model's accuracy is slightly lower at nearly 96%. This performance gap partially explains the PGS model's weaker overall planning results, as its rollouts are based on less accurate state predictions.

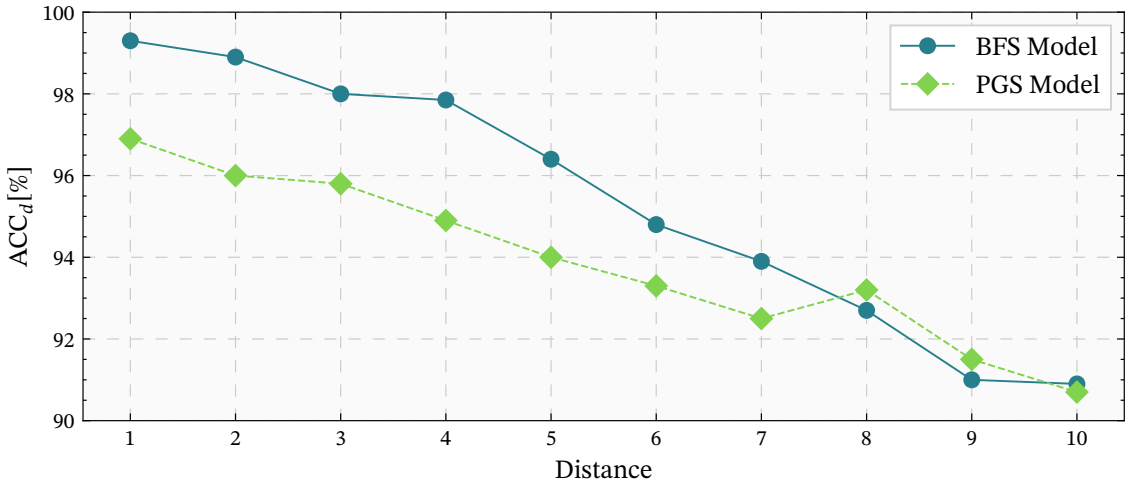


Figure 5.7: Multi-step prediction accuracy of the dynamics model (ACC$_d$), comparing the BFS and PGS models on the 8-Puzzle. Accuracy is plotted against the prediction horizon (number of sequential steps) and averaged over 100 rollouts.

Nevertheless, the general high accuracy of both models within the three-step horizon suggests that dynamics error alone does not account for the sharp drop in planning success at longer distances, particularly for the BFS model.

Distance Prediction Error

The distance prediction error of the IAM is evaluated by its RMSE$_d$ under two conditions, as shown in Figure 5.8. The left plot shows the inherent error in distance prediction, directly calculated from environmental observations for different optimal distances between them. The right plot shows the RMSE$_d$ when the distance is predicted based on the latent states, predicted by the dynamics model, over the respective optimal distance horizon.

The observation-based results (left) indicate a difference between the models. The BFS model's RMSE$_d$ is precise and stable for distances up to seven steps. Beyond this point, its error increases significantly, making it an unreliable heuristic for problems with a high optimal distance. In contrast, the PGS model's heuristic exhibits a considerably higher baseline error that grows continuously. At an optimal distance of just 4, its RMSE$_d$ is approximately 1.0, meaning its estimates can be off by a full step.

In comparison, for the states predicted by the dynamics model (right), the BFS model's prediction errors increase. This is attributable to the inaccuracy of the dynamics model. For the PGS model, however, the additional error falls within its already high baseline confidence interval.

This error analysis provides a possible explanation for the planning performance observed in Figure 5.6. The PGS model's consistently lower success rate is a direct consequence of its inherently less accurate distance and dynamics model predictions. The BFS model performs

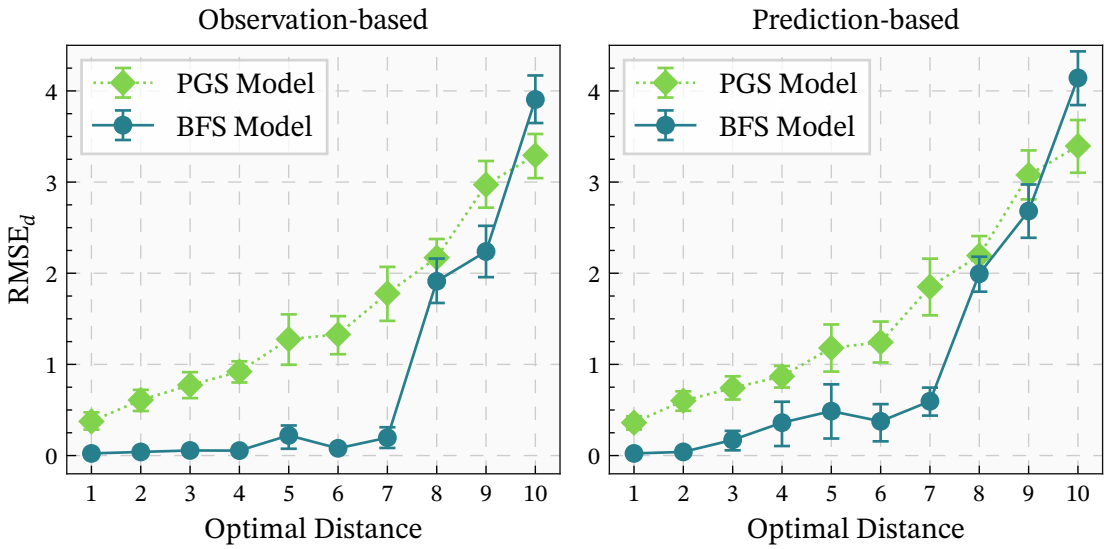


Figure 5.8: IQM and 95% bootstrapped confidence interval (5000 repetitions) of the RMSE$_d$ of the distance predictions for both models based on 50 datasets, each containing 2000 transitions per optimal distance. Left: Error based on environment observations with the respective optimal distance. Right: Error based on states predicted by the dynamics model for the respective optimal distance, illustrating the impact of the accumulated prediction error of the dynamics model on the distance prediction.

well at shorter distances due to its precise distance predictions. Its performance drops when the optimal distance exceeds five because the three-step rollouts require distance estimates for unreliable distances. For instance, if the current search state is six steps from the goal, a three-step rollout will lead to states that are potentially eight or more steps away, where the high error of the distance predictor makes the guided search ineffective.

Further Analysis

This section presents the general performance metrics of the initially retrained IAMs on the larger 500,000-sample dataset. As expected and shown in Table 5.5, both models achieve higher action classification accuracy and lower distance prediction error than their counterparts trained in Section 5.2. A notable result is the PGS model's aggregate RMSE$_d$ of 1.47, which is remarkably close to its empirically calculated optimal value of 1.39.

Sampling Method	**PGS**	**BFS**
ACC$_a$ [%]	99.4	100
RMSE$_d$	1.47	0.24

Table 5.5: Performance metrics for the PGS and BFS models trained on 500,000 samples and evaluated on a validation set of 2,000 samples. Results are direct results from a single model instance per sampling strategy.

This near-optimal performance appears to contradict the previous finding that the PGS distance prediction is consistently less accurate than the BFS distance prediction at specific distances (cf. Figure 5.8). The discrepancy is explained by examining the PGS model's predictions for different optimal distances, as shown in Figure 5.9. The figure plots the mean and variance of the model's distance predictions for each optimal distance, along with the optimal prediction targets that minimize the error.

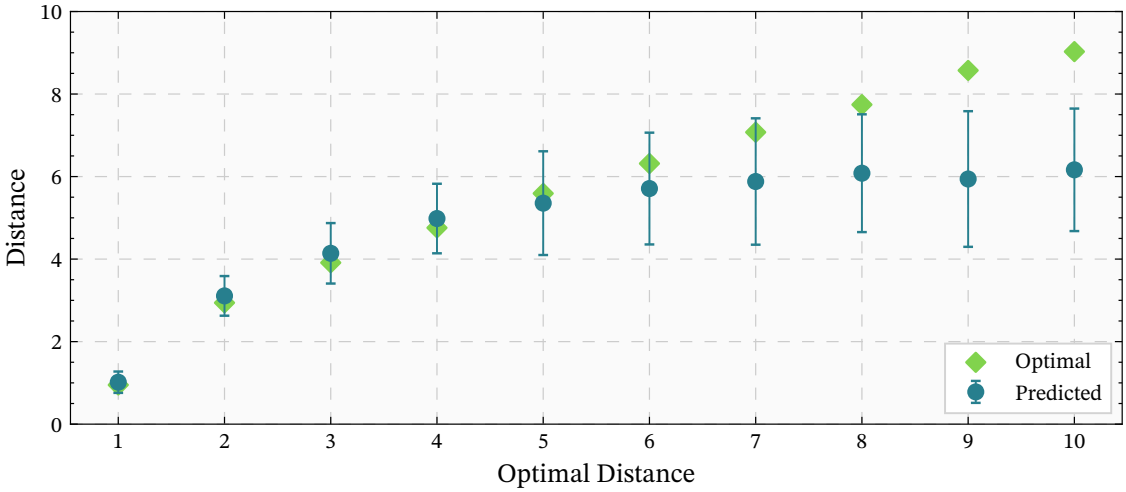


Figure 5.9: Mean and variance of the PGS-model's distance predictions for each optimal distance. The diamond markers show the optimal prediction targets that minimize the RMSE_d on the training distribution, using suboptimal distance labels with a discount factor of $\lambda = 0.95$. The mean and variance are calculated from the 500,000-sample dataset used to train the PGS model.

The analysis reveals that for optimal distances up to four, the distance predictions are unbiased on average but suffer from high variance. For distances greater than 4, the model shows significant bias, systematically underestimating the optimal prediction target. Although not visualised, this problem is not as drastic for the BFS model predictions, whose prediction variance remains below 0.1 for all optimal distances up to 7. However, while the PGS model is accurate on average, its high variance at each step makes it unreliable as a heuristic for guided search.

Finally, Figure 5.6 visualizes the proportion of optimally solved instances. The BFS model finds optimal paths for the majority of problems across all distances, a trend that also holds for the PGS model, except at horizons greater than eight. The majority of suboptimal solutions can be attributed to the three-step execution strategy (cf. Section 4.4.2). Executing all actions of each rollout per iteration increases the number of solved instances but results in more suboptimal solutions. While a high rate of optimally solved instances is a positive outcome, this investigation focuses on the learned state representation and the dynamics model, not on optimizing the planner itself. The overall planning performance could likely be improved

with standard enhancements to the search algorithm, such as implementing cycle detection to avoid revisiting states.

5.4.3 Summary

This assessment successfully validates the key contribution that the proposed IAM architecture learns a lifted state representation that effectively captures the environment's dynamics, enabling functional guided search. The learned dynamics model and distance predictor are jointly used to solve multiple 8-Puzzle problem instances, with the BFS-trained model demonstrating particularly strong performance on short to medium-horizon tasks.

The analysis shows that a main limitation is the imprecise distance prediction for greater optimal distances. The accuracy is lower at ranges where training data is sparse, a weakness reinforced by the accumulation of prediction error in the dynamics model. These results demonstrate the architecture's ability to plan from images, while suggesting that its effectiveness on more complex tasks could be improved by enriching the training data with more transitions corresponding to higher optimal distances.

6 Conclusion

This thesis addressed the challenge of learning symbolic state representations for planning directly from raw image data. Traditional planning approaches rely on hand-crafted, symbolic models of the environment, and constructing such models automatically from high-dimensional data, such as images, remains an open research problem. The presented IAM introduces a reconstruction-free approach for learning lifted, graph-based symbolic representations that capture the structure of the objects and their relational dependencies from images of a classical planning domain.

The presented IAM architecture combines convolutional, feed-forward, and graph neural networks. Given two environment observations as images, these are first encoded into sets of object encodings. The object encodings are then used to infer multiple binary relations between them. The objects and their relations are then used to construct a graph-based state representation for each environment observation. These graphs are then merged and processed by a GNN. It classifies the action that lies between the corresponding states of the observed environment, and predicts the number of actions needed to transition between them. This auxiliary task encourages the model to learn a meaningful symbolic latent-state representation implicitly.

The conducted experiments on the *N*-Puzzle and Sokoban domains demonstrate that the IAM successfully learns symbolic structures that capture the dynamics of the environment. The model achieved high action-classification accuracy and low distance-prediction error, demonstrating its ability to represent relational dependencies required for these training objectives. Furthermore, the predicted distances served as heuristics in a guided search, allowing the model to solve planning tasks within its latent space, which underscores that the learned structure captures the environment's dynamics.

The evaluation also revealed that the IAM's performance depends on the structure and composition of the training data. Sampling strategies that incorporate more environmental knowledge yielded more consistent and discriminative representations than strategies that incorporate less environment knowledge. This observation indicates that the distribution of the training data plays an important role in learning meaningful symbolic abstractions.

Finally, the architecture showed limited but encouraging generalization capabilities, successfully transferring learned relations from the 8-Puzzle to the 15-Puzzle and from single-box to two-box Sokoban instances, without requiring domain-specific architectural modifications. Together, these results validate the key contributions of this thesis by presenting a reconstruction-

free, relational inverse-action model architecture that learns symbolic representations from pixels, supports planning, and indicates possible transfer to other classical planning domains.

6.1 Future Work

Despite these promising results, several limitations highlight directions for future research. While the learned representations are symbolic in structure, the learned relations are not yet interpretable in terms of explicit predicates, such as PDDL-like ones. Future work could investigate methods to interpret the learned representations and make them more accessible.

Furthermore, this work assumes noise-free image observations. In realistic scenarios, perception is often affected by partial observability, occlusion, and noise. Future research could assess the model's reliability across different observation noise scenarios, thereby improving its applicability to real-world environments.

This work also shows that the observed generalization capabilities remain limited when training on a single domain instance. To enhance generalization capabilities, multiple instances of a domain could be used simultaneously for training, such as combining the 8- and 15-Puzzles, to encourage the discovery of higher-level, instance-invariant relations that generalize more robustly to larger or more complex problem instances.

Additionally, the model's reliance on sampling strategies that incorporate domain knowledge highlights a dependency that could limit scalability. Developing domain-independent sampling or training strategies could help overcome this limitation and make the approach more generally applicable across different domains.

Finally, the planning capabilities of the current model are constrained by the distance function's prediction horizon, which is bounded by the maximum distance in the training data. Future work could therefore explore datasets with transitions over longer distances and alternative training strategies to enable planning over longer horizons.

In summary, this work represents a meaningful step toward bridging perception and symbolic reasoning by learning structured, relational representations directly from raw images. It extends the development of autonomous systems capable of building and exploiting their own symbolic abstractions.

A Appendix

A.1 Proof of Unbiasedness of Collision Probability Estimator

Proof. Let $\mathcal{X}$ be the total observation space of size $N = |\mathcal{X}|$, and let $h : \mathcal{X} \mapsto \mathcal{Z}$ be the learned state representation function. The true collision probability is the probability that two distinct states, chosen uniformly at random from $\mathcal{X}$, map to the same latent representation:

$$\mathrm{CP}_{\mathrm{true}} = \mathbb{P}\big(h(x_i) = h\big(x_j\big) \mid x_i, x_j \in \mathcal{X}, x_i \neq x_j\big).$$

Let $\hat{\mathcal{X}} \subset \mathcal{X}$ be a set of k distinct observations sampled uniformly at random. The sample-based estimate of the collision probability is calculated as the fraction of colliding pairs within this sample:

$$\mathrm{CP}_{\mathrm{est}} = \frac{|\{\big(x_i, x_j\big) \in \hat{\mathcal{X}} \times \hat{\mathcal{X}} \mid i \neq j, h(x_i) = h\big(x_j\big)\}|}{|\{\big(x_i, x_j\big) \in \hat{\mathcal{X}} \times \hat{\mathcal{X}} \mid i \neq j\}|} = \frac{\text{Number of colliding pairs in } \hat{\mathcal{X}}}{\binom{k}{2}}.$$

We aim to show that this estimator is unbiased, i.e., $\mathbb{E}[\mathrm{CP}_{\mathrm{est}}] = \mathrm{CP}_{true}$, where the expectation is taken over all possible samples $\hat{\mathcal{X}}$ of size k.

Let's define an indicator random variable I_{uv} for any pair of distinct states $\{u, v\}$ from the sample $\hat{\mathcal{X}}$. Let $I_{uv} = 1$ if $h(u) = h(v)$ and $I_{uv} = 0$ otherwise. The number of colliding pairs in the sample is the sum of these indicators over all unique pairs $\mathcal{Y} = \{\{u, v\} \mid u, v \in \hat{\mathcal{X}} \times \hat{\mathcal{X}}, u \neq v\}$: $\sum_{\{u,v\}\in\mathcal{Y}} I_{uv}$. The expected value of the estimator is:

$$\mathbb{E}[\mathrm{CP}_{\mathrm{est}}] = \mathbb{E}\left[\frac{\sum_{\{u,v\}\in\mathcal{Y}} I_{uv}}{\binom{k}{2}}\right].$$

By the linearity of expectation, we can move the expectation inside the summation:

$$\mathbb{E}[\mathrm{CP}_{\mathrm{est}}] = \frac{1}{\binom{k}{2}} \sum_{\{u,v\}\in\mathcal{Y}} \mathbb{E}[I_{uv}].$$

Since the sample $\hat{\mathcal{X}}$ is drawn uniformly at random from $\mathcal{X}$, any pair of states $\{u, v\}$ from the sample constitutes a uniformly random pair of distinct states from the total space $\mathcal{X}$. Therefore, the probability that this pair collides is, by definition, the true collision probability:

$$\mathbb{E}[I_{uv}] = P(h(u) = h(v)) = \mathrm{CP}_{true}.$$

This holds for every one of the $\binom{k}{2}$ pairs in the sample. Substituting this result back into our equation for the expected value of the estimator yields:

$$\mathbb{E}[\mathrm{CP}_{\mathrm{est}}] = \frac{1}{\binom{k}{2}} \sum_{\{u,v\}\in\mathcal{Y}} \mathrm{CP}_{true} = \frac{1}{\binom{k}{2}} \left(\binom{k}{2} \cdot \mathrm{CP}_{true} \right) = \mathrm{CP}_{true}.$$

Thus, the sample-based collision probability is an unbiased estimator of the true collision probability over the entire state space. □

A.2 Model Architecture

The subsequent tables provide a detailed, layer-by-layer specification of the proposed inverse action model architecture. All parameters and layer definitions are consistent with the conventions of the libraries used for implementation (cf. Section 4.4.3).

Symbol	Description
B	Batch size
C	Number of image channels (e.g., 1 for grayscale)
n	Number of objects in latent representation
d	Dimensionality of each object feature vector
m	Number of relation types
RGCN	Relational Graph Convolutional Network
SiLU	Sigmoid Linear Unit activation ($x \cdot \sigma(x)$)

Table A.1: Model architecture notation.

Layer	Type	Output Shape	Parameters
Conv1	Conv2d	$(B, 32, 42, 42)$	kernel=5, stride=2, padding=2
BatchNorm1	BatchNorm2d	$(B, 32, 42, 42)$	–
Activation	SiLU	$(B, 32, 42, 42)$	–
Conv2	Conv2d	$(B, 64, 21, 21)$	kernel=5, stride=2, padding=2
BatchNorm2	BatchNorm2d	$(B, 64, 21, 21)$	–
Activation	SiLU	$(B, 64, 21, 21)$	–
Conv3	Conv2d	$(B, 128, 11, 11)$	kernel=5, stride=2, padding=2
BatchNorm3	BatchNorm2d	$(B, 128, 11, 11)$	–
Activation	SiLU	$(B, 128, 11, 11)$	–
Conv4	Conv2d	$(B, 256, 6, 6)$	kernel=5, stride=2, padding=2
BatchNorm4	BatchNorm2d	$(B, 256, 6, 6)$	–
Activation	SiLU	$(B, 256, 6, 6)$	–
Global Pool	AdaptiveAvgPool2d	$(B, 256)$	output_size=1
Mean	Linear	$(B, n \times d)$	–
Variance	Linear + Softplus	$(B, n \times d)$	$\epsilon = 10^{-6}$

Table A.2: Architecture of the object encoding model enc_ϕ.

Layer	**Type**	**Output Shape**	**Parameters**
Input	Pairwise Concat	$(B, 2, n^2, 2d)$	All object pairs
FC1	Linear	$(B, 2, n^2, 32)$	–
LayerNorm1	LayerNorm	$(B, 2, n^2, 32)$	–
Activation	SiLU	$(B, 2, n^2, 32)$	–
FC2	Linear	$(B, 2, n^2, 64)$	–
LayerNorm2	LayerNorm	$(B, 2, n^2, 64)$	–
Activation	SiLU	$(B, 2, n^2, 64)$	–
FC3	Linear	$(B, 2, n^2, 128)$	–
LayerNorm3	LayerNorm	$(B, 2, n^2, 128)$	–
Activation	SiLU	$(B, 2, n^2, 128)$	–
FC4	Linear	$(B, 2, n^2, 128)$	–
LayerNorm4	LayerNorm	$(B, 2, n^2, 128)$	–
Activation	SiLU	$(B, 2, n^2, 128)$	–
FC5	Linear	$(B, 2, n^2, 128)$	–
LayerNorm5	LayerNorm	$(B, 2, n^2, 128)$	–
Activation	SiLU	$(B, 2, n^2, 128)$	–
FC6	Linear	$(B, 2, n^2, 128)$	–
LayerNorm6	LayerNorm	$(B, 2, n^2, 128)$	–
Activation	SiLU	$(B, 2, n^2, 128)$	–
FC7	Linear	$(B, 2, n^2, 64)$	–
LayerNorm7	LayerNorm	$(B, 2, n^2, 64)$	–
Activation	SiLU	$(B, 2, n^2, 64)$	–
FC8	Linear	$(B, 2, n^2, 32)$	–
LayerNorm8	LayerNorm	$(B, 2, n^2, 32)$	–
Activation	SiLU	$(B, 2, n^2, 32)$	–
FC9	Linear	$(B, 2, n^2, 16)$	–
Activation	SiLU	$(B, 2, n^2, 16)$	–
FC10	Linear	$(B, 2, n, n)$	–
Output	Sigmoid + Sampling	$(B, 2, n, n)$	–

Table A.3: Architecture of the relation predictor rel_i for two simultaneous inputs (per relation type, m instances).

Layer	Type	Output Shape	Parameters
Graph Convolutional Backbone:			
RGCN Conv1	WeightedRGCNConv	$(2Bn, 64)$	$m + 1$ relation types
BatchNorm1	BatchNorm1d	$(2Bn, 64)$	–
Activation	SiLU	$(2Bn, 64)$	–
RGCN Conv2	WeightedRGCNConv	$(2Bn, 128)$	$m + 1$ relation types
BatchNorm2	BatchNorm1d	$(2Bn, 128)$	–
Activation	SiLU	$(2Bn, 128)$	–
RGCN Conv3	WeightedRGCNConv	$(2Bn, 256)$	$m + 1$ relation types
BatchNorm3	BatchNorm1d	$(2Bn, 256)$	–
Activation	SiLU	$(2Bn, 256)$	–
Global Pool	Global Add Pool	$(B, 256)$	Graph-level aggregation
Action Classification Head:			
FC1 Action	Linear	$(B, 64)$	–
Activation	SiLU	$(B, 64)$	–
FC2 Action	Linear	$(B, 5)$	Output action logits
Value Regression Head:			
FC1 Value	Linear	$(B, 64)$	–
Activation	SiLU	$(B, 64)$	–
FC2 Value	Linear	$(B, 1)$	State value estimate

Table A.4: Architecture of the R-GCN comb_ω, the action classification head cls_ψ, and the value regression head dist_ρ.

Layer	Type	Output Shape	Parameters
Input	State + Action Concat	$(Bn, d+5)$	One-hot action
RGCN Conv1	WeightedRGCNConv	$(Bn, 32)$	m relation types
LayerNorm1	LayerNorm	$(Bn, 32)$	–
Activation	SiLU	$(Bn, 32)$	–
RGCN Conv2	WeightedRGCNConv	$(Bn, 64)$	m relation types
LayerNorm2	LayerNorm	$(Bn, 64)$	–
Activation	SiLU	$(Bn, 64)$	–
RGCN Conv3	WeightedRGCNConv	$(Bn, 128)$	m relation types
LayerNorm3	LayerNorm	$(Bn, 128)$	–
Activation	SiLU	$(Bn, 128)$	–
RGCN Conv4	WeightedRGCNConv	$(Bn, 128)$	m relation types
LayerNorm4	LayerNorm	$(Bn, 128)$	–
Activation	SiLU	$(Bn, 128)$	–
RGCN Conv5	WeightedRGCNConv	$(Bn, 128)$	m relation types
LayerNorm5	LayerNorm	$(Bn, 128)$	–
Activation	SiLU	$(Bn, 128)$	–
RGCN Conv6	WeightedRGCNConv	$(Bn, 128)$	m relation types
LayerNorm6	LayerNorm	$(Bn, 128)$	–
Activation	SiLU	$(Bn, 128)$	–
RGCN Conv7	WeightedRGCNConv	$(Bn, 128)$	m relation types
Activation	SiLU	$(Bn, 128)$	–
Reshape	View	$(B, n \times 128)$	Flatten per batch
FC1	Linear	$(B, 128)$	–
Activation	SiLU	$(B, 128)$	–
FC2	Linear	$(B, 128)$	–
Activation	SiLU	$(B, 128)$	–
FC Mean	Linear	(B, n, d)	Predicted next state

Table A.5: Architecture of the dynamics model dyn_χ.

List of Acronyms

AdamW	Adam with Decoupled Weight Decay
AE	Autoencoder
AMA	Action Model Acquisition
BFS	Breadth-First Search Sampling
CNN	Convolutional Neural Network
CP	Collision Probability
CPP	Classical Planning Problem
ELBO	Evidence Lower Bound
GNN	Graph Neural Network
IAM	Inverse Action Model
IQM	Interquartile Mean
JEPA	Joint Embedding Predictive Architecture
KL	Kullback-Leibler
LLM	Large Language Model
LSRI	Latent Space Ratio Interval
MLP	Multilayer perceptron
MPPI	Model Predictive Path Integral
MSE	Mean Squared Error
PDDL	Planning Domain Definition Language
PGS	Partial-Graph Sampling
PLDM	Planning with a Latent Dynamics Model
R-GCN	Relational Graph Convolutional Network
RL	Reinforcement Learning
RMSE	Root Mean Squared Error
RMSprop	Root Mean Square Propagation
SAE	State Autoencoder

SG	Scene Graph
SGD	Stochastic Gradient Descent
SiLU	Sigmoid-Weighted Linear Unit
SRL	State Representation Learning
ST-SGG	Spatio-Temporal Scene Graph Generation
STRIPS	Stanford Research Institute Problem Solver
TS	Traces Sampling
VAE	Variational Autoencoder
VIGReg	Variance-Invariance-Covariance Regularization
W&B	Weights & Biases

List of Symbols

General Mathematical and Machine Learning Notation

$x, \mathbf{X}$	input observation, typically an image
y	target label for supervised learning
$\hat{y}$	predicted output from a model
D	dataset of input-output pairs $\{(x_i, y_i)\}$
$\mathcal{L}$	general loss function
$J(\theta)$	empirical risk or cost function parameterized by θ
$\theta, \phi, \psi, \omega, \rho, \chi$	sets of learnable parameters for different model components
$\nabla_\theta J(\theta)$	gradient of the cost function J with respect to parameters θ
η	learning rate in gradient-based optimization
$\mathbf{W}, \mathbf{b}$	weight matrix and bias vector of a neural network layer
$h(\cdot), \sigma(\cdot)$	non-linear activation function, e.g., Sigmoid or SiLU
$\mathbb{R}^{n \times d}$	space of $n \times d$ matrices with real-valued entries
$\mathbb{E}[\cdot]$	expected value
$\mathcal{N}(\mu, \Sigma)$	normal (Gaussian) distribution with mean μ and covariance Σ
$D_{\mathrm{KL}}(p\|\|q)$	KL divergence between distributions p and q
I	identity matrix
$\mathrm{sg}(\cdot)$	stop-gradient operator
$*$	convolution operation
$\odot$	element-wise multiplication
$\alpha, \beta, \gamma, \delta$	hyperparameters weighting the terms of the combined loss function
λ	hyperparameter for discounting the distance loss

State Representation Learning

s_t	true, underlying state of the environment at time t
a_t	action taken at time t
x_t	high-dimensional observation (e.g., an image) of state s_t
z_t	low-dimensional latent representation of state s_t
$\mathcal{S}, \mathcal{A}, \mathcal{X}, \mathcal{Z}$	state, action, observation, and latent spaces, respectively
$\mathrm{enc}_\phi(\cdot)$	encoder network mapping observations to latent representations
$\mathrm{dec}_\theta(\cdot)$	decoder network mapping latent representations back to observations
$\mathrm{dyn}_\chi(\cdot)$	forward dynamics model predicting the next latent state
$\mathrm{cls}_\psi(\cdot)$	inverse action model for classifying the action from two states
$\mathrm{dist}_\rho(\cdot)$	model for regressing the distance between two states

$\mathcal{L}_{recon}$	reconstruction loss used in autoencoders
$\mathcal{L}_{cls}$	classification loss (e.g., cross-entropy) for the inverse action model
$\mathcal{L}_{dist}$	regression loss (e.g., MSE) for the distance prediction
$\mathcal{L}_{dyn}$	prediction loss for the forward dynamics model
$\mathcal{L}_{KL}$	KL-divergence loss term used in variational methods

Graph Theory and Graph Neural Networks

$Z = (V, E)$	graph with a set of nodes V and a set of edges E
Z_a, Z_b	symbolic state representations of states a and b as directed multigraphs
h_v^k	embedding (feature vector) of node v at the k-th GNN layer
$\mathcal{N}(v)$	set of neighboring nodes of node v
$\mathbf{W}_r^k$	learnable weight matrix for relation type r at layer k in an R-GCN
$\text{comb}_\omega(\cdot)$	GNN module that processes the combined state graphs
$\mathbf{g}$	graph-level embedding produced by the GNN

Proposed Model Architecture

$\mathbf{X}_a, \mathbf{X}_b$	two input image observations
a, v	ground-truth action and distance labels for a transition
$\hat{a}, \hat{v}$	predicted action and distance from the model
n	number of objects in the symbolic representation
d	dimensionality of a single object embedding
$\mathbf{O} \in \mathbb{R}^{n \times d}$	object encoding matrix, containing n object embeddings
$\mathbf{o}_i \in \mathbb{R}^d$	embedding of the i-th object
τ	signature, a set of predicate symbols $\{P_1, ..., P_m\}$
$\mathfrak{D}$	τ-structure representing a state, consisting of objects and relations
$\text{rel}_\theta(\cdot)$	relation prediction module with parameters θ
$R_{a,i}, R_{b,i}$	binary adjacency matrices for relation i in states a and b

Classical Planning

M	classical planning state model (S, s_0, S_G, A, dyn, c)
s_0, S_G	initial state and the set of goal states
$Pre(o)$	preconditions of a STRIPS operator o
$Add(o)$	add effects of a STRIPS operator o
$Del(o)$	delete effects of a STRIPS operator o

Evaluation Metrics

ACC_a	action classification accuracy
RMSE_d	root mean squared error of the distance prediction
ACC_d	prediction accuracy of the dynamics model's output graph
CP	collision probability, a measure of injectivity
LSRI	latent space ratio interval, an interval as measure of injectivity

M_{min}, M_{max}	estimated minimum and maximum cardinality of the latent space
N_c	number of conflicts in a dataset
p_{suc}, p_{inv}	proportion of valid (successor) and invalid transitions in a dataset

List of Figures

List of Tables

List of Algorithms

List of References

[1] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare, "Deep Reinforcement Learning at the Edge of the Statistical Precipice,"

[2] J. An and S. Cho, *Variational Autoencoder based Anomaly Detection using Reconstruction Probability*, http://dm.snu.ac.kr/static/docs/TR/SNUDM-TR-2015-03.pdf, Dec. 2015. (visited on 07/20/2025).

[3] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh, "VQA: Visual Question Answering," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile: IEEE, Dec. 2015, pp. 2425–2433. DOI: 10.1109/iccv.2015.279. (visited on 07/22/2025).

[4] M. Asai and A. Fukunaga, *Classical Planning in Deep Latent Space: Bridging the Subsymbolic-Symbolic Boundary*, Dec. 2017. DOI: 10.48550/arXiv.1705.00154. arXiv: 1705.00154 [cs]. (visited on 01/20/2025).

[5] D. Bank, N. Koenigstein, and R. Giryes, *Autoencoders*, Apr. 2021. DOI: 10.48550/arXiv.2003.05991. arXiv: 2003.05991 [cs]. (visited on 07/20/2025).

[6] A. Bardes, J. Ponce, and Y. LeCun, *VICReg: Variance-Invariance-Covariance Regularization for Self-Supervised Learning*, Jan. 2022. DOI: 10.48550/arXiv.2105.04906. arXiv: 2105.04906 [cs]. (visited on 07/12/2025).

[7] Y. Bengio, N. Léonard, and A. Courville, *Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation*, Aug. 2013. DOI: 10.48550/arXiv.1308.3432. arXiv: 1308.3432 [cs]. (visited on 02/17/2025).

[8] C. M. Bishop, *Pattern Recognition and Machine Learning* (Information Science and Statistics). New York: Springer, 2006, ISBN: 978-0-387-31073-2.

[9] C. M. Bishop and H. Bishop, "Deep Neural Networks," in *Deep Learning: Foundations and Concepts*, C. M. Bishop and H. Bishop, Eds., Cham: Springer International Publishing, 2024, pp. 171–207, ISBN: 978-3-031-45468-4. DOI: 10.1007/978-3-031-45468-4_6. (visited on 10/01/2025).

[10] E. F. Camacho, C. Bordons, and J. M. Maestre, *Model Predictive Control* (Advanced Textbooks in Control and Signal Processing), 3rd ed. Cham, Switzerland: Springer Cham, 2025, p. 383, ISBN: 978-3-031-87595-3.

[11] S. Chandhok, *SceneGPT: A Language Model for 3D Scene Understanding*, Aug. 2024. DOI: 10.48550/arXiv.2408.06926. arXiv: 2408.06926 [cs]. (visited on 07/22/2025).

[12] D. Ekpo, M. Levy, S. Suri, C. Huynh, and A. Shrivastava, *VeriGraph: Scene Graphs for Execution Verifiable Robot Planning*, Nov. 2024. DOI: 10.48550/arXiv.2411.10446. arXiv: 2411.10446 [cs]. (visited on 07/22/2025).

[13] S. Ferraro, P. Mazzaglia, T. Verbelen, and B. Dhoedt, *FOCUS: Object-Centric World Models for Robotics Manipulation*, Jul. 2023. DOI: 10.48550/arXiv.2307.02427. arXiv: 2307.02427 [cs]. (visited on 07/14/2025).

[14] H. Geffner and B. Bonet, *A Concise Introduction to Models and Methods for Automated Planning*. Springer International Publishing, 2013. DOI: 10.1007/978-3-031-01564-9.

[15] M. Ghallab, D. Nau, and P. Traverso, "ACTING, PLANNING, AND LEARNING,"

[16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[17] D. Ha and J. Schmidhuber, "World Models," Mar. 2018. DOI: 10.5281/zenodo.1207631. arXiv: 1803.10122 [cs]. (visited on 02/08/2025).

[18] D. Hafner, K.-H. Lee, I. Fischer, and P. Abbeel, "Deep Hierarchical Planning from Pixels,"

[19] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, *Dream to Control: Learning Behaviors by Latent Imagination*, Mar. 2020. DOI: 10.48550/arXiv.1912.01603. arXiv: 1912.01603 [cs]. (visited on 01/12/2025).

[20] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, *Learning Latent Dynamics for Planning from Pixels*, Jun. 2019. DOI: 10.48550/arXiv.1811.04551. arXiv: 1811.04551 [cs]. (visited on 01/21/2025).

[21] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba, *Mastering Atari with Discrete World Models*, Feb. 2022. DOI: 10.48550/arXiv.2010.02193. arXiv: 2010.02193 [cs]. (visited on 07/24/2025).

[22] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, *Mastering Diverse Domains through World Models*, Apr. 2024. DOI: 10.48550/arXiv.2301.04104. arXiv: 2301.04104 [cs]. (visited on 07/24/2025).

[23] W. L. Hamilton, R. Ying, and J. Leskovec, *Inductive Representation Learning on Large Graphs*, Sep. 2018. DOI: 10.48550/arXiv.1706.02216. arXiv: 1706.02216 [cs]. (visited on 02/01/2025).

[24] P. Haslum, N. Lipovetzky, D. Magazzeni, and C. Muise, *An Introduction to the Planning Domain Definition Language* (Synthesis Lectures on Artificial Intelligence and Machine Learning). Cham: Springer International Publishing, 2019, ISBN: 978-3-031-00456-8 978-3-031-01584-7. DOI: 10.1007/978-3-031-01584-7. (visited on 10/04/2025).

[25] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "β-VAE: LEARNING BASIC VISUAL CONCEPTS WITH A CONSTRAINED VARIATIONAL FRAMEWORK," 2017.

[26] J. Johnson, A. Gupta, and L. Fei-Fei, *Image Generation from Scene Graphs*, Apr. 2018. DOI: 10.48550/arXiv.1804.01622. arXiv: 1804.01622 [cs]. (visited on 07/20/2025).

[27] R. Jonschkowski and O. Brock, "Learning state representations with robotic priors," *Autonomous Robots*, vol. 39, no. 3, pp. 407–428, Oct. 2015, ISSN: 0929-5593, 1573-7527. DOI: 10.1007/s10514-015-9459-7. (visited on 01/10/2025).

[28] S. Khandelwal and L. Sigal, *Iterative Scene Graph Generation*, Jul. 2022. DOI: 10.48550/arXiv.2207.13440. arXiv: 2207.13440 [cs]. (visited on 07/21/2025).

[29] K. Kim, K. Yoon, J. Jeon, Y. In, J. Moon, D. Kim, and C. Park, *LLM4SGG: Large Language Models for Weakly Supervised Scene Graph Generation*, Jul. 2024. DOI: 10.48550/arXiv.2310.10404. arXiv: 2310.10404 [cs]. (visited on 07/21/2025).

[30] D. P. Kingma and M. Welling, *Auto-Encoding Variational Bayes*, Dec. 2022. DOI: 10.48550/arXiv.1312.6114. arXiv: 1312.6114 [stat]. (visited on 10/05/2025).

[31] Y. LeCun, "A Path Towards Autonomous Machine Intelligence Version 0.9.2, 2022-06-27,"

[32] T. Lesort, N. Díaz-Rodríguez, J.-F. Goudou, and D. Filliat, "State Representation Learning for Control: An Overview," *Neural Networks*, vol. 108, pp. 379–392, Dec. 2018, ISSN: 08936080. DOI: 10.1016/j.neunet.2018.07.006. arXiv: 1802.04181 [cs]. (visited on 01/10/2025).

[33] X. Linghu, J. Huang, X. Niu, X. Ma, B. Jia, and S. Huang, *Multi-modal Situated Reasoning in 3D Scenes*, Nov. 2024. DOI: 10.48550/arXiv.2409.02389. arXiv: 2409.02389 [cs]. (visited on 07/22/2025).

[34] I. Loshchilov and F. Hutter, *Decoupled Weight Decay Regularization*, Jan. 2019. DOI: 10.48550/arXiv.1711.05101. arXiv: 1711.05101 [cs]. (visited on 09/03/2025).

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, ISSN: 1476-4687. DOI: 10.1038/nature14236. (visited on 01/24/2025).

[36] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker, *Model-based Reinforcement Learning: A Survey*, Mar. 2022. DOI: 10.48550/arXiv.2006.16712. arXiv: 2006.16712 [cs]. (visited on 10/04/2025).

[37] *NetworkX — NetworkX documentation*, https://networkx.org/. (visited on 10/07/2025).

[38] Z. Ni, X. Deng, C. Tai, X. Zhu, Q. Xie, W. Huang, X. Wu, and L. Zeng, *GRID: Scene-Graph-based Instruction-driven Robotic Task Planning*, Mar. 2024. DOI: 10.48550/arXiv.2309.07726. arXiv: 2309.07726 [cs]. (visited on 07/15/2025).

[39] C. Olson, "Probabilistic self-localization for mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 1, pp. 55–66, Feb. 2000, ISSN: 1042296X. DOI: 10.1109/70.833191. (visited on 01/24/2025).

[40] L. Pinheiro Cinelli, M. Araújo Marins, E. A. Barros da Silva, and S. Lima Netto, "Variational Autoencoder," in *Variational Methods for Machine Learning with Applications to Deep Networks*, L. P. Cinelli, M. A. Marins, E. A. Barros da Silva, and S. L. Netto, Eds., Cham: Springer International Publishing, 2021, pp. 111–149, ISBN: 978-3-030-70679-1. DOI: 10.1007/978-3-030-70679-1_5. (visited on 10/12/2025).

[41] *PyG Documentation — pytorch_geometric documentation*.

[42] *PyTorch documentation — PyTorch 2.8 documentation*, https://docs.pytorch.org/docs/stable/index.html, /index.html. (visited on 09/04/2025).

[43] P. Rogaway and T. Shrimpton, "Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance," in *Fast Software Encryption*, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, B. Roy, and W. Meier, Eds., vol. 3017, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 371–388, ISBN: 978-3-540-22171-5 978-3-540-25937-4. DOI: 10.1007/978-3-540-25937-4_24. (visited on 09/02/2025).

[44] S. Ruder, *An overview of gradient descent optimization algorithms*, Jun. 2017. DOI: 10.48550/arXiv.1609.04747. arXiv: 1609.04747 [cs]. (visited on 10/02/2025).

[45] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach* (Prentice Hall Series in Artificial Intelligence), Third edition, Global edition. Boston Columbus Indianapolis: Pearson, 2016, ISBN: 978-0-13-604259-4 978-1-292-15397-1.

[46] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, *Modeling Relational Data with Graph Convolutional Networks*, Oct. 2017. DOI: 10.48550/arXiv.1703.06103. arXiv: 1703.06103 [stat]. (visited on 03/04/2025).

[47] V. Sobal, W. Zhang, K. Cho, R. Balestriero, T. G. J. Rudner, and Y. LeCun, *Learning from Reward-Free Offline Data: A Case for Planning with Latent Dynamics Models*, Feb. 2025. DOI: 10.48550/arXiv.2502.14819. arXiv: 2502.14819 [cs]. (visited on 07/12/2025).

[48] Stinchcombe and White, "Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions," in *International 1989 Joint Conference on Neural Networks*, Jun. 1989, 613–617 vol.1. DOI: 10.1109/IJCNN.1989.118640. (visited on 10/01/2025).

[49] *The Python Language Reference*, https://docs.python.org/3/reference/index.html. (visited on 09/04/2025).

[50] S. Tibshirani and H. Friedman, “Valerie and Patrick Hastie,”

[51] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” in *Proceedings of the 25th International Conference on Machine Learning - ICML ’08*, Helsinki, Finland: ACM Press, 2008, pp. 1096–1103. DOI: 10.1145/1390156.1390294. (visited on 07/20/2025).

[52] Z. Wan, Y. Zhang, and H. He, “Variational autoencoder based synthetic data generation for imbalanced learning,” in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, Nov. 2017, pp. 1–7. DOI: 10.1109/SSCI.2017.8285168. (visited on 07/20/2025).

[53] Y. Wang, M. Yasunaga, H. Ren, S. Wada, and J. Leskovec, *VQA-GNN: Reasoning with Multimodal Knowledge via Graph Neural Networks for Visual Question Answering*, Sep. 2023. DOI: 10.48550/arXiv.2205.11501. arXiv: 2205.11501 [cs]. (visited on 07/22/2025).

[54] *Weights & Biases: The AI Developer Platform*.

[55] *Welcome to ⚡ PyTorch Lightning — PyTorch Lightning 2.5.4 documentation*, https://lightning.ai/docs/pytorch/stable/?referrer=platform-docs. (visited on 09/04/2025).

[56] G. Williams, A. Aldrich, and E. Theodorou, *Model Predictive Path Integral Control using Covariance Variable Importance Sampling*, Oct. 2015. DOI: 10.48550/arXiv.1509.01149. arXiv: 1509.01149 [cs]. (visited on 07/25/2025).

[57] L. Wu, P. Cui, J. Pei, L. Zhao, and L. Song, “Graph Neural Networks,” in *Graph Neural Networks: Foundations, Frontiers, and Applications*, L. Wu, P. Cui, J. Pei, and L. Zhao, Eds., Singapore: Springer Nature, 2022, pp. 27–37, ISBN: 978-981-16-6054-2. DOI: 10.1007/978-981-16-6054-2_3. (visited on 01/31/2025).

[58] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A Comprehensive Survey on Graph Neural Networks,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, Jan. 2021, ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2020.2978386. arXiv: 1901.00596 [cs]. (visited on 01/31/2025).

[59] K. Xi, S. Gould, and S. Thiébaux, “Neuro-Symbolic Learning of Lifted Action Models from Visual Traces,” *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 34, pp. 653–662, May 2024, ISSN: 2334-0843. DOI: 10.1609/icaps.v34i1.31528. (visited on 07/29/2025).

[60] S. Yan, Y. Xiong, and D. Lin, *Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition*, Jan. 2018. DOI: 10.48550/arXiv.1801.07455. arXiv: 1801.07455 [cs]. (visited on 03/16/2025).

[61] G. Zhu, L. Zhang, Y. Jiang, Y. Dang, H. Hou, P. Shen, M. Feng, X. Zhao, Q. Miao, S. A. A. Shah, and M. Bennamoun, *Scene Graph Generation: A Comprehensive Survey*, Jun. 2022. DOI: 10.48550/arXiv.2201.00443. arXiv: 2201.00443 [cs]. (visited on 02/03/2025).

[62] G. Zhu, L. Zhang, Y. Jiang, Y. Dang, H. Hou, P. Shen, M. Feng, X. Zhao, Q. Miao, S. A. A. Shah, and M. Bennamoun, *Scene Graph Generation: A Comprehensive Survey*, Jun. 2022. DOI: 10.48550/arXiv.2201.00443. arXiv: 2201.00443 [cs]. (visited on 01/16/2025).

[63] Y. Zhu, J. Tremblay, S. Birchfield, and Y. Zhu, *Hierarchical Planning for Long-Horizon Manipulation with Geometric and Symbolic Scene Graphs*, Mar. 2021. DOI: 10.48550/arXiv.2012.07277. arXiv: 2012.07277 [cs]. (visited on 07/22/2025).