

Fricker, Nicolai Benjamin; Krüger, Nicolai; Schubart, Constantin

Working Paper

Learning to play Sokoban from videos

IU Discussion Papers - IT & Engineering, No. 3 (Oktober 2024)

Provided in Cooperation with:

IU International University of Applied Sciences

Suggested Citation: Fricker, Nicolai Benjamin; Krüger, Nicolai; Schubart, Constantin (2024) : Learning to play Sokoban from videos, IU Discussion Papers - IT & Engineering, No. 3 (Oktober 2024), IU Internationale Hochschule, Erfurt

This Version is available at:

<https://hdl.handle.net/10419/303523>

Standard-Nutzungsbedingungen:

Die Dokumente auf EconStor dürfen zu eigenen wissenschaftlichen Zwecken und zum Privatgebrauch gespeichert und kopiert werden.

Sie dürfen die Dokumente nicht für öffentliche oder kommerzielle Zwecke vervielfältigen, öffentlich ausstellen, öffentlich zugänglich machen, vertreiben oder anderweitig nutzen.

Sofern die Verfasser die Dokumente unter Open-Content-Lizenzen (insbesondere CC-Lizenzen) zur Verfügung gestellt haben sollten, gelten abweichend von diesen Nutzungsbedingungen die in der dort genannten Lizenz gewährten Nutzungsrechte.

Terms of use:

Documents in EconStor may be saved and copied for your personal and scholarly purposes.

You are not to copy documents for public or commercial purposes, to exhibit the documents publicly, to make them publicly available on the internet, or to distribute or otherwise use the documents in public.

If the documents have been made available under an Open Content Licence (especially Creative Commons Licences), you may exercise further usage rights as specified in the indicated licence.

www.iu.de

IU DISCUSSION

PAPERS

IT & Engineering

Learning to Play Sokoban from Videos

NICOLAI FRICKER

NICOLAI KRÜGER

CONSTANTIN SCHUBART

IU Internationale Hochschule

Main Campus: Erfurt

Juri-Gagarin-Ring 152

99084 Erfurt

Telefon: +49 421.166985.23

Fax: +49 2224.9605.115

Kontakt/Contact: kerstin.janson@iu.org

Autorenkontakt/Contact to the author(s):

Prof. Dr. Nicolai Krüger

ORCID-ID: 0000-0002-0848-6856 (Open Researcher und Contributor ID)

IU Internationale Hochschule - Campus Düsseldorf

Hildebrandtstraße 24c

40215 Düsseldorf

Email: nicolai.krueger@iu.org

Prof. Dr. Constantin Schubart

ORCID-ID: 0009-0008-9259-0533 (Open Researcher und Contributor ID)

IU Internationale Hochschule - Campus Erfurt

Juri-Gagarin-Ring 152

99084 Erfurt Email: constantin.schubart@iu.org

IU Discussion Papers, Reihe: IT & Engineering, Vol. 5, No. 3 (OKT 2024)

ISSN-Nummer: 2750-073X

Website: <https://www.iu.de/forschung/publikationen/>

Learning to Play Sokoban from Videos

Nicolai Fricker**Constantin Schubart****Nicolai Krüger****ABSTRACT:**

In order to learn a task through behavior cloning, a dataset consisting of state-action pairs is needed. However, this kind of data is often not available in sufficient quantity or quality. Consequently, several publications have addressed the issue of extracting actions from a sequence of states to convert them into corresponding state-action pairs (Torabi et al., 2018; Edwards et al., 2019; Baker et al., 2022; Bruce et al., 2024). Using this dataset, an agent can then be trained via behavior cloning. For instance, this approach was applied to games such as Cartpole and Mountain Car (Edwards et al., 2019). Additionally, actions were extracted from videos of Minecraft (Baker et al., 2022) and jump 'n' run games (Edwards et al., 2019; Bruce et al., 2024) to train deep neural network models to play these games.

In this work, videos from YouTube as well as synthetic videos of the game Sokoban were analyzed. Sokoban is a single-player, turn-based game where the player has to push boxes onto target squares (Murase et al., 1996). The actions that a user performs in the videos were extracted using a modified training procedure described by Edwards et al. (2019). The resulting state-action pairs were used to train deep neural network models to play Sokoban. These models were further improved with reinforcement learning in combination with a Monte Carlo tree search as a planning step. The resulting agent demonstrated moderate playing strength.

In addition to learning how to solve a Sokoban puzzle, the rules of Sokoban were learned from videos. This enabled the creation of a Sokoban simulator, which was used to carry out model-based reinforcement learning.

This work serves as a proof of concept, demonstrating that it is possible to extract actions from videos of a strategy game, perform behavior cloning, infer the rules of the game, and perform model-based reinforcement learning – all without direct interaction with the game environment.

Code and models are available at https://github.com/loanMaster/sokoban_learning.

KEYWORDS:

Imitation learning, behavior cloning, deep neural network models, reinforcement learning

AUTHORS



Dr. Nicolai Fricker completed his PhD at the German Cancer Research Center, specializing in Systems Biology. His research focused on the mathematical modeling of biochemical pathways. Additionally, he holds a master's degree in Artificial Intelligence from IU International University of Applied Sciences.



Prof. Dr. Nicolai Krüger is Professor of Information Systems at IU Düsseldorf. His research interests include Digital Innovation, Transformation and Entrepreneurship (DITE) and the application of Generative Artificial Intelligence. He is also the founder of the innovation consultancy pitchnext and a regular jury member of several entrepreneurship programs, especially the Founders@IU program.



Prof. Dr. Constantin Schubart is Professor of General Business Administration at IU Erfurt. His research interests lie in corporate and personnel development in the digital space. He is also the founder of the consulting company Schubart Consulting.

Introduction

SOKOBAN

Sokoban (Japanese for "storage manager") is a single-player game where the player needs to push boxes on a grid board onto target squares. It was designed by Hiroyuki Imabayashi and published by the company Thinking Rabbit in 1982 (Murase et al., 1996). The player has to plan ahead to avoid situations from which they cannot recover, such as pushing a box into a corner. Actions in Sokoban cannot be reversed; boxes can only be pushed, never pulled. The game is won once all boxes have been placed on the targets.

Sokoban is an NP-hard problem (Fryers and Green, 1995), and computers have struggled to reach superhuman performance (Junghanns et al., 1998). Furthermore, it has been shown to be PSPACE-complete (Culberson, 1997). Search trees are often used to solve the game (Junghanns et al., 1998; Shoham and Schaeffer, 2020). A standard breadth-first search tree does not perform well in Sokoban due to the high number of moves required to solve a level and the branching factor of four. Therefore, an A* search with a heuristic to estimate the distance to the goal state can be applied. Using a clever search tree design and hand-crafted features, it was possible to solve all 90 levels of the XSokoban test suite in 2020 for the first time (Shoham and Schaeffer, 2020).

For several years, there has been a shift towards agents using deep neural networks in Sokoban (Weber et al., 2017; Feng et al., 2020; Yang et al., 2022; Shoham and Elidan, 2021). These models implement a policy to decide the next move that should be taken and evaluate the current position. Weber et al. (2017) combined Monte Carlo tree search (MCTS) with reinforcement learning (RL) to train their model, similar to what has been successfully applied to two-player games such as Go and Chess (Silver et al., 2016; Silver et al., 2017).

One challenge in Sokoban is the issue of sparse rewards - rewards are only given when the agent successfully solves a level. To address this, Feng et al. (2020) and Yang et al. (2022) used increasingly difficult levels in a curriculum training approach. Feng et al. removed most boxes from a level before training was initialized. Once the model performed well with just a few boxes, the number of boxes in the level was gradually increased. Shoham and Elidan (2021) suggested using two independent models. One model begins from the final state of a level and attempts to pull all boxes from their target squares. During this process, a state trace is generated. The second model tries to move the boxes close to the trace. With this method, they were able to solve 88 of the 90 levels of XSokoban. In all of these publications, hand-crafted features were used to some extent to guide the models.

BEHAVIOR CLONING FROM OBSERVATION

In machine learning, one distinguishes between supervised and unsupervised learning (Goodfellow et al., 2016, p. 98). Unsupervised learning is done on unlabeled data and is typically used to perform clustering or to find outliers in the data. In supervised learning, a labeled dataset is needed to train the model. Labeled data contains a label, such as a category, in addition to the raw data. Supervised learning is often used to categorize data, for instance in object recognition or image segmentation. Labeled data is also needed for performing behavior cloning as a pretraining step before RL (Edwards et al., 2019). However, labeled data is not available in high quantities in many areas. Creating labeled data from scratch can be costly (Bruce et al., 2024, p. 3). Therefore, several research groups have investigated the possibility of training an agent using unlabeled data (Torabi et al., 2018; Edwards et al., 2019; Zhang et al., 2022; Baker et al., 2022; Bruce et al., 2024). The underlying problem can be formulated as a Markov decision process (MDP).

An MDP consists of a set of states S , actions A , transition probabilities, and a reward function $R: S \times A \rightarrow \mathbb{R}$ mapping a state-action pair to a reward. When dealing with unlabeled data, only sequences of states s_1, \dots, s_n and possibly rewards are given, without any associated actions. In contrast, labeled data - in the context of behavior cloning - consists of sequences of state-action pairs $(s_1, a_1), \dots, (s_n, a_n)$.

Edwards et al. (2019) constructed a machine learning process to extract latent actions from states given as vectors. In their approach, a model G is designed that takes a state s_k and a latent action z_k as arguments and returns the predicted change from s_k to s_{k+1} as output: $G: S \times Z \rightarrow \Delta(S)$. The model receives all possible latent actions Z as input. For each action, the loss is calculated as:

$$L(s_n, z_i, \omega) = \text{MSE}(G_\omega(s_n, z_i), s_{n+1} - s_n)$$

Where MSE is the mean squared error and ω refers to the parameters of the model G . The actual training/gradient descent is performed using only the action yielding the smallest loss. The model learns to predict the change from one state to the next given an action. The actions resulting in minimal loss can be used to construct a labeled dataset $(s_1, z_1), \dots, (s_n, z_n)$. However, for extracted latent actions, there is no immediate correspondence to the actions in the game. For instance, in Sokoban, latent action 0 could sometimes mean *move up* and in other game situations *push a box to the left*. To resolve this ambiguity, Edwards et al. (2019, p. 4) interact with the actual game environment and perform a remapping from latent actions to actual game actions.

Bruce et al. (2024) and Baker et al. (2022) use a similar approach with unlabeled videos instead of vectors as input to their models. To prevent the ambiguity of actions observed by Edwards et al. (2019,

p. 5), Baker et al. (2022, p. 4) perform pretraining with labeled data, whereas Bruce et al. (2024, p. 9) use labeled data after training with unlabeled videos.

The agents that were trained in this manner were used for Pong, CartPole (Edwards et al., 2019), jump 'n' run games (Bruce et al., 2024), and Minecraft (Baker et al., 2022). Although Minecraft is a complex game and can be considered a strategy game, none of these publications has dealt with a classical round-based strategy game that requires planning, such as Sokoban.

This work will evaluate whether the approach presented by Edwards et al. (2019) is applicable to the strategy game Sokoban. Furthermore, it will be evaluated whether a model trained on unlabeled videos of Sokoban can be further improved using RL in both model-based and model-free approaches.

Methodology

ARTIFACTS

The main questions of this study were broken down into several tasks:

- Extract actions from unlabeled Sokoban videos.
- Train a policy model and value model using the extracted actions and video frames.
- Improve the policy and value model with RL in a model-free environment.
- Learn the rules of Sokoban from extracted actions and video frames and generate a Sokoban simulator.
- Conduct model-based RL using the Sokoban simulator.

To accomplish these tasks, several artifacts needed to be generated, as listed in table 1.

Name	Description	Purpose
Frame prediction model	Given video frames and an input action this model predicted the delta to the next video frame.	-Transform a dataset of video frames into a dataset of frame-action pairs. -The model could be used to create a Sokoban simulation.
Policy model	Given input frames from a video, this model decided which action to take	Solving Sokoban levels.
Value model	This model assigned a value to each Sokoban position based on video frames.	Solving Sokoban levels. The value model was needed for RL with MCTS.
Action validation model	Determined if a Sokoban move was valid given video frames as input data.	Create a Sokoban simulation that can validate moves.
Game state model	Determined if a Sokoban level was finished/solved given an input frame.	Create a Sokoban simulation that can determine if the game is finished.

Table 1: An overview over the artifacts produced.

TOOLS USED

To generate synthetic Sokoban videos and facilitate RL, a Sokoban gym environment available on GitHub (<https://github.com/mpSchrader/gym-sokoban>) was utilized. The icons in this environment

were adjusted to match those found in several Sokoban videos on YouTube. This adjustment was helpful for better comparing results obtained from YouTube data and synthetic data.

In addition to the gym environment, a Sokoban solver (<https://github.com/KnightofLuna/sokoban-solver>) was utilized, capable of employing various search methods to solve given levels. For generating synthetic Sokoban videos, an A* search algorithm was used to find solutions to levels generated within the gym environment.

Python, in combination with PyTorch, was used for implementing and training neural networks.

STATISTICS

The results of RL were evaluated using the Two-Proportion Z-Test (Montgomery and Runger, 2010, p. 301).

TRAINING OF NEURAL NETWORKS

Training of networks was conducted using gradient descent, with a batch size of 16 and an Adam optimizer initialized with a learning rate of $1e-5$. An exponential learning rate decay of 0.9998 was applied, unless otherwise specified, with a minimum learning rate set to $1e-6$. In all experiments, training data and validation/test data were drawn from identical distributions.

CREATION OF SYNTHETIC VIDEOS

For behavior cloning, videos from more than 3,300 levels were created using the Sokoban gym environment and Sokoban solver mentioned previously, resulting in over 100,000 video frames in total. For RL, an additional 5,000 training levels and 300 validation levels were generated.

EXTRACTION OF YOUTUBE VIDEOS

Sokoban videos found on YouTube encompassed a variety of Sokoban game versions, differing in quality and board size, unlike the synthetic data used in this work (Fig. **Fehler! Verweisquelle konnte nicht gefunden werden.**). Each video was manually selected and downloaded, then adjusted by cropping or padding to achieve a consistent 1:1 ratio of height to width. Frames were extracted using the command-line tool ffmpeg. To eliminate redundant frames - those with identical board positions - a feature vector was derived from each frame using a pretrained ResNet50 model. Frames with feature vectors below a specified threshold were removed. The threshold value was determined manually after conducting several test runs.

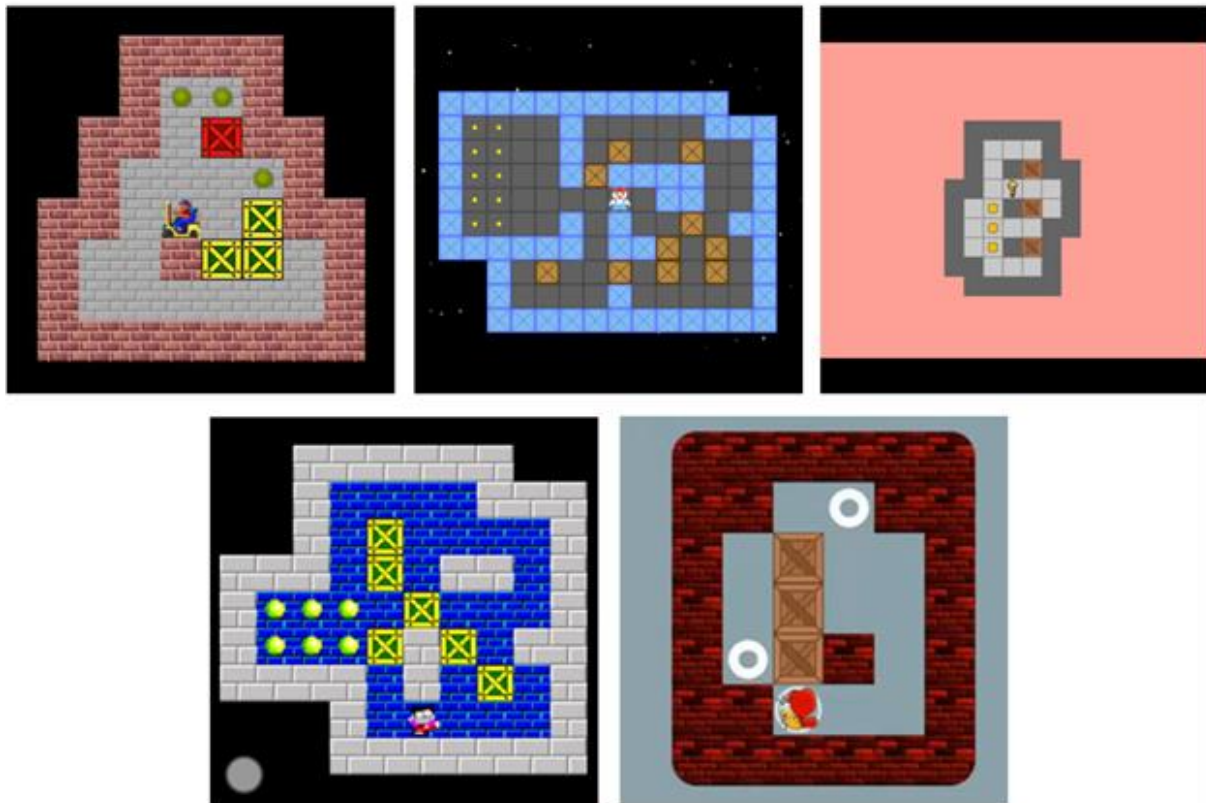


Figure 2: Different Sokoban game versions found on YouTube. (Source: <https://youtube.com>)

EXTRACTION OF ACTIONS FROM VIDEOS

Edwards et al. (2019, p. 2) introduced an approach to extract actions from a sequence of states, which was slightly modified for this study. The problem was formulated as an MDP. To determine actions $a_1 \dots a_n$ that caused the transition from a state s_i to s_{i+1} a frame prediction model G was developed. The model architecture follows the U-Net architecture (Ronneberger et al., 2015). It consists of five convolution blocks. The first four convolution blocks are followed by a pooling layer. Subsequently, there are four upsampling blocks with skip connections from the convolution blocks. The output is an image that predicts the delta from the current frame to the next frame.

The frame prediction model receives as input the first frame of a Sokoban video s_1 , the current frame s_i and an action a_i (represented as a one-hot tensor scaled to match the image size, with values ranging from 0 to 4). The model outputs the expected change between the current frame and the next frame:

$$G: (s_1, s_i, a_i) \rightarrow \Delta(s_{i+1}, s_i)$$

The assumption behind choosing five actions is that there are four directional actions and one "wait" action. The "wait" action was introduced to handle video data extracted from YouTube and could be omitted when working with synthetic data. Leveraging the game's symmetry, a slightly modified

version of the algorithm published by Edwards et al. (2019, p. 2) was implemented (batch size was omitted for clarity):

for n in number_of_training_steps:

$s_1, s_i, s_{i+1} \leftarrow$ fetch training data

set loss := 0

for alpha in [0°, 90°, 180°, 270°]:

$s_1', s_i', s_{i+1}' := s_1, s_i, s_{i+1}$ rotated by alpha

find action a_k such that $k = \arg \min_j \text{MSE}(G(s_1', s_i', a_j), s_{i+1}' - s_i')$

remove a_k from the possible actions if a_k is not "wait"; otherwise remove all actions but a_k

loss += $\text{MSE}(G(s_1', s_i', a_k), s_{i+1}' - s_i')$

perform gradient descent

To directly extract meaningful actions corresponding to each direction (*up, down, left, right*) and to avoid the remapping from latent action to game action used by Edwards et al. (2019), the game's symmetry was leveraged: after selecting an action a_i resulting in minimal loss, the input image was rotated by 90°, 180°, and 270°. For each rotation angle, a different action was chosen for gradient descent. Additionally, action 4 was designated as the "wait action", assumed to keep the game state unchanged regardless of rotation.

Once the frame prediction model had been trained, it could be used to extract actions from unlabeled videos like so:

1. Take a random frame s_i , the start frame s_1 of the level as well as the follow up frame s_{i+1} from the data set.
2. Generate a prediction for the next frame for each action a_1, \dots, a_k using the frame prediction model:

$$\sigma_k := G(s_1, s_i, a_k) + s_i$$

3. Select/extract action a_k where

$$k = \underset{k}{\operatorname{argmin}} \text{MSE}(\sigma_k, s_{i+1})$$

Using the method outlined in this paragraph, a sequence of video frames from a Sokoban level was transformed into a sequence of frame-action pairs that could be used for behavior cloning.

BEHAVIOR CLONING

Behavior cloning was implemented using categorical cross-entropy loss:

$$L(s_1, s_i, \omega) := -\sum_k a_k \log(\pi_\omega(s_1, s_i))$$

Where $a_k = 1$ if the action a_k was the action performed in the training data, otherwise $a_k = 0$. Here, ω refers to the parameters of the policy model π .

A convolutional network was chosen as the policy model, consisting of five convolution blocks with pooling layers, followed by ten residual blocks with two convolutions per residual block. Following the residual blocks there was a pooling layer and fully-connected network (FCN) layer resulting in an output vector with a length corresponding to the number of possible actions. This design resembles the architecture presented in the Go publication by Silver et al. (2016, p. 27).

In addition to the policy model, a value model was trained with the intention of using it later in RL. The value model estimates the expected discounted reward given a state. The discounted reward can be calculated easily from a sequence of video frames $s_1 \dots s_n$ from a level:

$$\text{value}(s_i) = \text{reward} * \gamma^{n-i}$$

Here, the reward was set to 1 once a level was finished, and a discount factor γ of 0.95 was used. The value model had the same architecture as the policy model, except it had a single output variable with a sigmoid activation function. The loss function for training the value model was defined as MSE between the predicted and actual value.

REINFORCEMENT LEARNING

RL was performed using planning steps implemented with MCTS. This approach has been successfully applied to learn games such as Go and Chess (Silver et al., 2016; Silver et al., 2017) and has also been utilized in several publications dealing with Sokoban (Feng et al., 2020; Efroni et al., 2019). Unlike depth-first or breadth-first search, MCTS applies a heuristic that balances exploration and exploitation.

Typically, during the simulation phase of the MCTS search, a rollout is performed. However, in this work, rollouts were skipped, and instead, the value of a node was estimated using a value model. This was necessary due to the low performance of rollouts and hardware constraints.

An MCTS with a depth of 75 was used, meaning that at most 75 new nodes were expanded during each planning step.

A modified version of an implementation available on GitHub was used to train the policy and value models (<https://github.com/foersterrobert/AlphaZeroFromScratch>). A modification was added to prevent the agent from revisiting a state: if a state had already been visited during MCTS, any new node leading to the same state would be ignored and not added to the search tree. This was achieved by storing a hash code for each visited state in memory.

Results

EXTRACTION OF ACTIONS FROM VIDEOS

Approximately 100,000 images from 3,000 synthetically generated Sokoban videos were used to train a frame prediction model to obtain actions corresponding to each transition from one frame to another. The actions 0 to 3 returned by the training procedure aligned with the game actions *up*, *down*, *left*, and *right*. No additional mapping from latent action to real action was required, in contrast to the findings by Edwards et al. (2019). This lack of necessity for mapping can be attributed to two main factors:

- Initially, during the training procedure, an additional loss term was introduced to encourage the model to select each action with equal probability. This loss term was progressively reduced with each training step.
- The symmetry of the game was used to ensure each action had a distinct meaning, as explained in the methodology section.

When the training was performed exactly as described in Edwards et al. (2019, p. 2), it was indeed observed that, at times, an action (e.g., action 2) could mean either moving in one direction or pushing a box in another direction.

Regarding the frames extracted from YouTube videos, several issues with data quality were identified:

1. Not all redundant frames could be automatically removed, resulting in some frames where there was no change in the board position between subsequent frames. This issue was mitigated by adding a *wait* action in addition to the four movement actions, each corresponding to a direction.
2. In some instances, the player moved quickly, making some game steps not visible in the video. For example, diagonal movements, which are not possible in Sokoban, were observed.
3. Some videos showed players using the undo function, which appeared as if the player started pulling a box, an illegal move in Sokoban.

Videos where issues (2) or (3) occurred frequently were omitted. If such issues occurred infrequently (less than once in 100 moves), the data was used as-is. In total, more than 100,000 moves/frames remained after removing redundant frames.

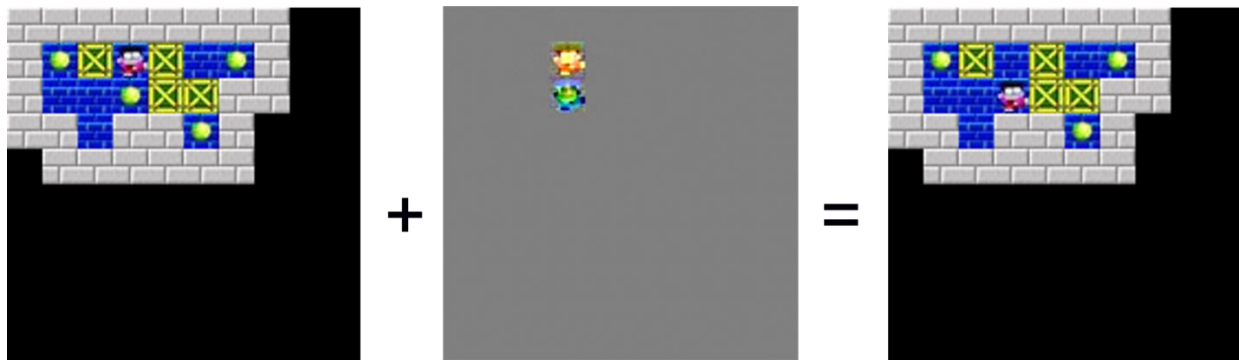


Figure 2: Prediction of frames. From left to right: screenshot of a Sokoban level; predicted frame change after taking the action "move down"; predicted next frame as a result of adding the current screenshot with the predicted frame change.

When manually verifying the correctness of the action extraction in several hundred frames, an accuracy of 100% was observed both for synthetic and YouTube data. Therefore, the labeled datasets of frame-action pairs obtained from videos were accurate and suitable for behavior cloning. Fig. 2 shows the prediction of the frame change upon taking an action.

BEHAVIOR CLONING

Behavior cloning was performed according to the procedure explained in the methodology section, with datasets obtained from synthetic data or YouTube data. More than 100,000 images were used for training for each of the two datasets. During training with synthetic data, no overfitting was observed. In the final model, the accuracy in predicting actions was approximately 85% for images from both the training set and the validation set. The training loss decreased considerably slower when using YouTube compared to synthetic data, and the accuracy was much lower. After ~150,000 training steps, the accuracy of a policy model trained on YouTube data was about 50%, compared to ~85% after just 50,000 training steps using synthetic data.

The ability of these policy models to solve new, unknown levels was evaluated. 200 new levels of the same 10x10 board size were generated. When the policy models were used to solve these levels, a low rate of solved levels was observed. Investigation of the issue revealed that actions chosen by the policies often led to repeated states or cycles. To address this issue, whenever the environment ended up in the same state twice, a different action was chosen. With this adjustment, 40 out of 200 levels were solved by a policy model trained on synthetic data, corresponding to a success rate of 20%. In contrast, an untrained model or a model trained on YouTube data was unable to solve any of the levels (Fig. 3).

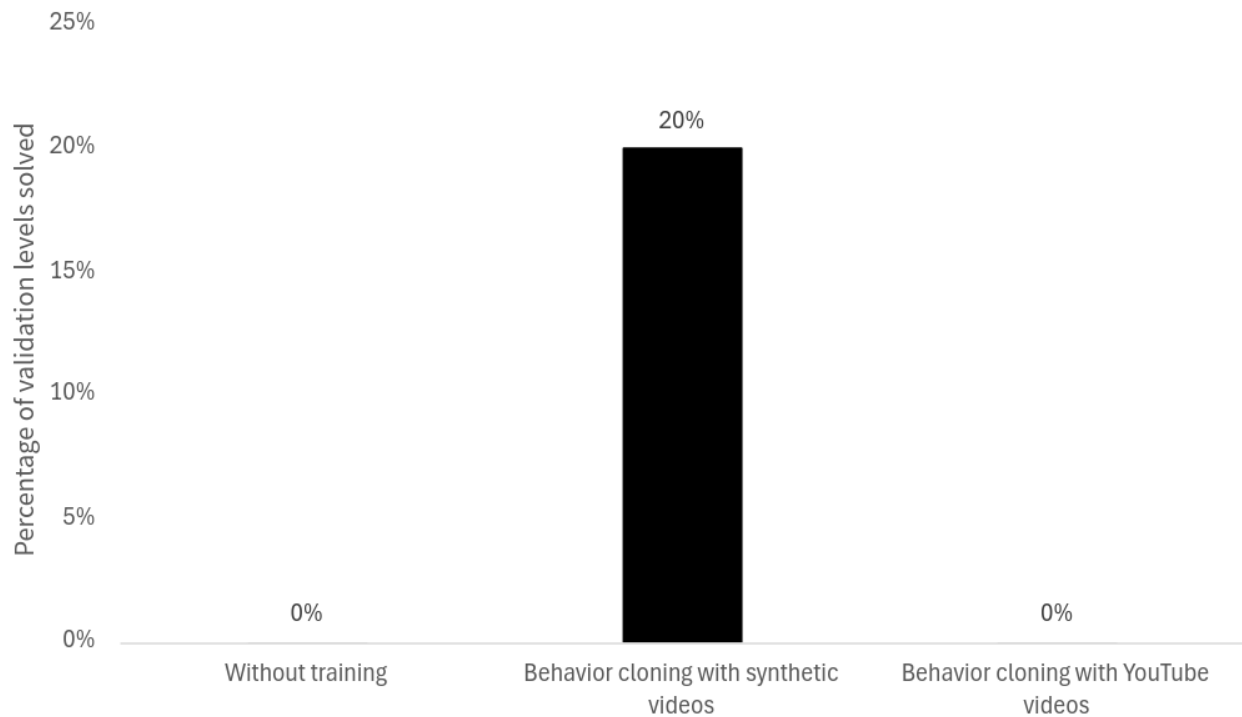


Figure 3: Performance of policy models after behavior cloning with synthetic or YouTube data.

In addition to the policy model, a value model was trained. This model was tasked with estimating the expected discounted reward given a state. The value error was around 0.001 for a value model trained on either synthetic or YouTube data.

MODEL-FREE REINFORCEMENT LEARNING

The policy and value models obtained after training on state-actions pairs extracted from synthetic or YouTube videos were used as a starting point for RL. RL utilized MCTS as a planning step. After each MCTS, an action was executed in the Sokoban gym environment. The planning step was applied both during training and evaluation.

Using the models obtained after behavior cloning with synthetic data but before RL, 156 out of 300 validation levels could be solved, corresponding to a success rate of ~52% (Fig. 4). This success rate is higher than the 20% obtained using the policy model alone, due to the incorporation of MCTS at each step. Silver et al. (2016) utilized a similar setup to train agents for playing the board game Go. They observed that training the policy model could lead to a decrease in performance, whereas training the value model consistently had a positive effect (Silver et al., 2016, p. 8). In this thesis, this effect was confirmed. None of the conditions tested showed that training the policy model described by Silver et al. (2016) improved performance. Silver et al. (2016) speculate that training the policy model might narrow down the search tree, thereby reducing overall performance. Therefore, instead of using the

relative node visit count as the target for the policy model, the policy model was trained differently: it was trained using the state-action pairs of the solution trajectory only when a level was solved. Using these adjusted policy model targets, training of both the policy and value models proceeded effectively. The value model was trained as proposed by Silver et al. (2016), using the values obtained from state traces during RL.

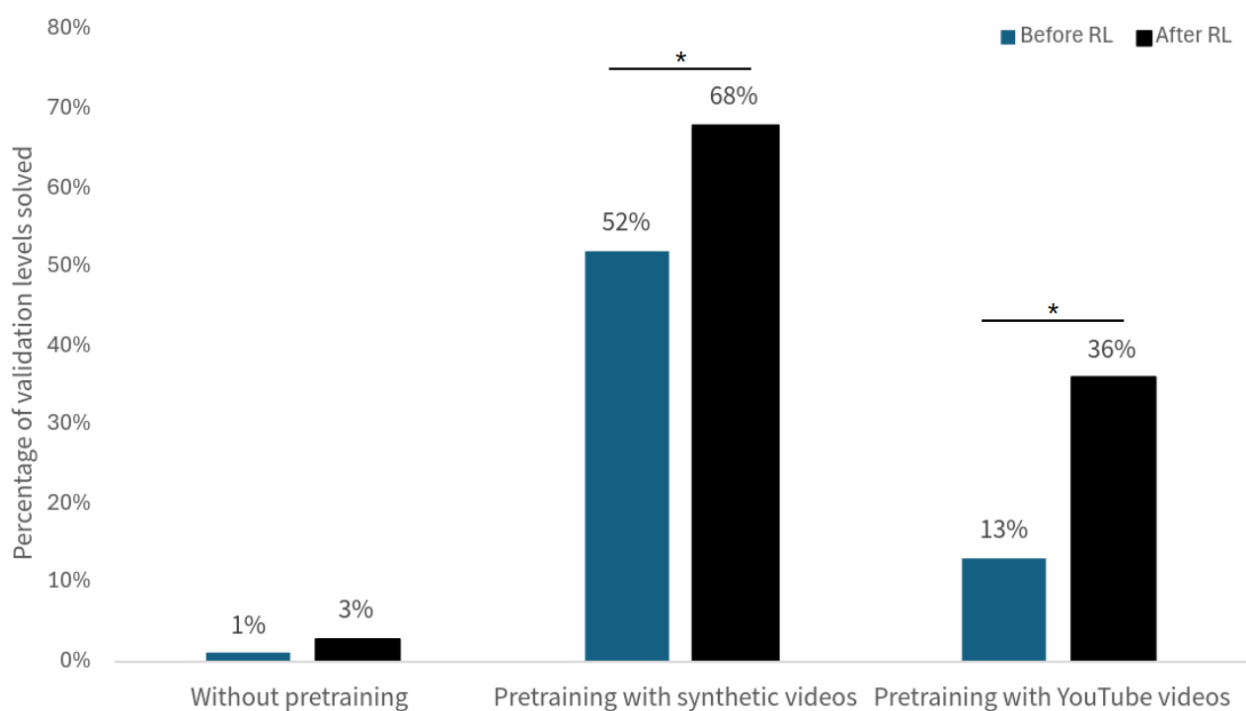


Figure 4: Performance of differently trained agents. From left to right: (1) Results before and after RL of randomly initialized models. (2) Performance of an agent pretrained with synthetic data (3) Performance of an agent pretrained with YouTube data. Asterisks indicate a significant difference of an agent before and after RL, as determined by the Two-Proportion z-Test.

After training the agent on approximately 1,500 levels with a learning rate of $1e-6$, it was able to solve 204 out of 300 validation levels, corresponding to an increase in solved levels from 52% to 68% (Fig. 4). However, training appeared to plateau after approximately 1,500 games.

The agent that had been pretrained using data from YouTube performed considerably worse. Before RL it could solve just 13% of levels. After RL with 1,400 games with a learning rate of $1e-6$ and subsequently with 2,200 games with a learning rate of $1e-7$ this number increased to 36%.

SIMULATING SOKOBAN

The initial steps in training a model presented in this thesis were all based on unlabeled video data. However, when starting model-free RL, this approach was abandoned, because the Sokoban environment was introduced as external component, making unlabeled videos no longer the only source of information (Fig. 5). Therefore, there was a motivation to test if not only playing Sokoban

could be learned from videos but also if the rules of the game could be inferred, thereby enabling model-based RL without relying on additional data sources or components. This idea was further motivated by the findings of Bruce et al. (2024), who simulated a jump 'n' run game.

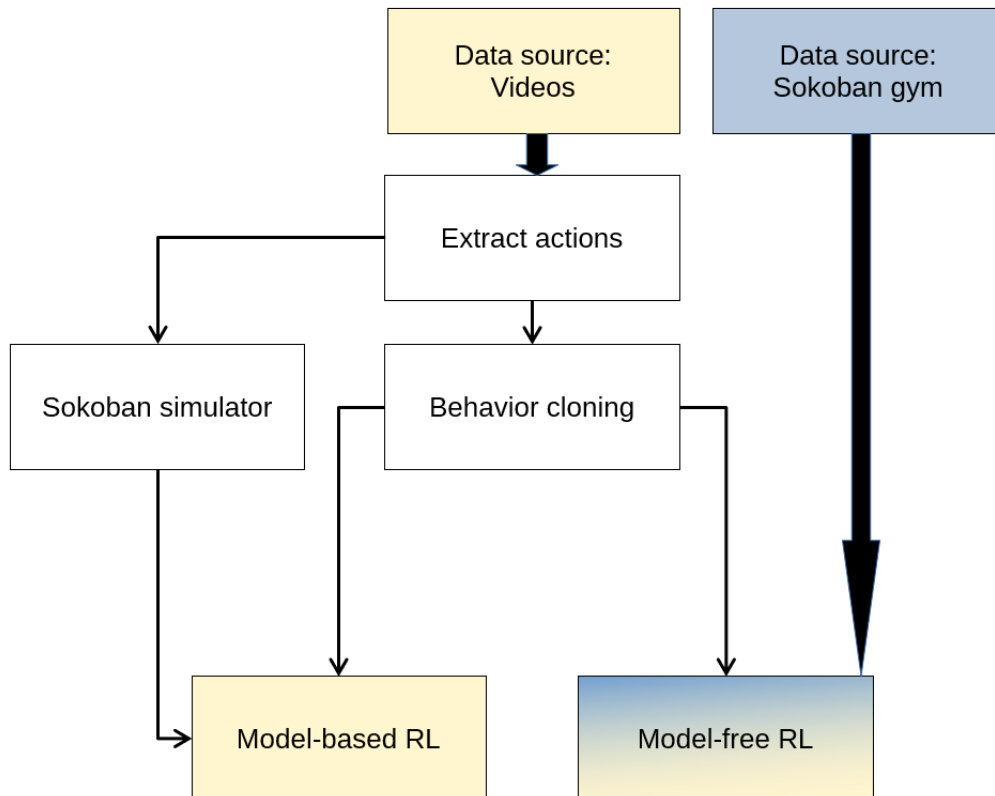


Figure 5: Data sources during model-free and model-based RL.

To construct a simulator, three models were needed:

- a frame prediction model
- an action validation model
- a game state model

When extracting the actions from videos a frame prediction model was trained as a by-product. However, a degradation in image quality was observed after performing several moves, each time feeding the next predicted frame back into the model. To prevent this degradation, the learning procedure was slightly adjusted. Instead of minimizing the loss between the predicted next frame and the actual next frame for each move, frames were skipped. For instance, if actions a_1, \dots, a_4 led from frame s_1 to frame s_5 , the frame prediction model G was called sequentially with actions a_1, \dots, a_4 , each time reusing its intermediate result. Then the resulting image was compared to the actual frame s_5 . A pseudocode of the procedure is shown here:

repeat for n training steps:

select a random number t between 1 and 5

fetch t frame-actions pairs $(s_i, a_i) \dots (s_{i+t}, a_{i+t})$ from the training set

current_frame := s_i

for k in 0..(t-1):

current_frame := $G(\text{current_frame}, a_{i+k}) + \text{current_frame}$

loss = $MSE(\text{current_frame}, s_{i+t})$

perform gradient descent

Using this adjusted learning process, image degradation was no longer observed, even after dozens of time steps. The model could accurately predict the next frame, but it could not determine if a move was legal or whether the game was won.

Given a dataset of a sequence of frames and actions, it is not immediately apparent which moves are valid and which are not. From the data, we know which moves were played. However, when examining the moves/actions that were not taken, it becomes necessary to distinguish between moves that were not played because they are bad or unusual, and those that are illegal or impossible.

An action validation model, consisting of five convolution blocks with pooling layers, followed by an FCN layer and a sigmoid function, was trained to address this issue. It received two frames as input: the current frame and a possible next frame. The model was supposed to determine whether the next frame originated from a legal move or not. It was trained on frames generated by the frame prediction model, which were produced by either performing a move found in a Sokoban video or a random move that was not part of the training set (Table 3). An issue was that the desired output of the frame prediction model did not always correspond to the label in the dataset. The dataset labeled moves as "true" if they were observed in a Sokoban video and otherwise "false". However, the action validation model should return "true" if a move was valid, even if it was not found in a Sokoban video.

Initially, standard binary cross-entropy loss was used to train the model. However, the model performed poorly. Therefore, the loss function was adjusted such that false negative results incurred a loss that was 100 times larger than false positive results:

$$\text{loss} = 100 * y * \log(\hat{y}) + (1 - y) * \log(1 - \hat{y})$$

Where y are the true labels and \hat{y} is the model output.

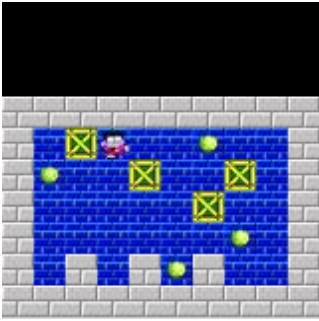
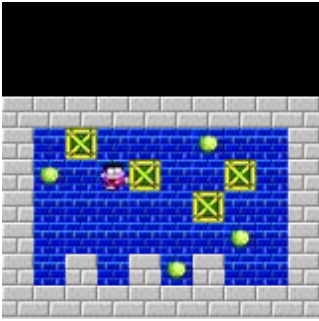
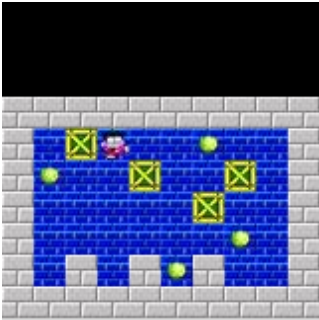
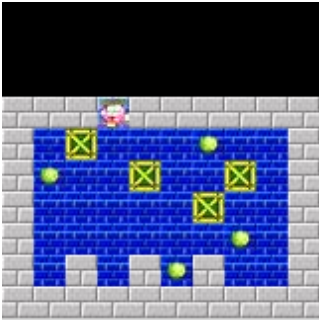
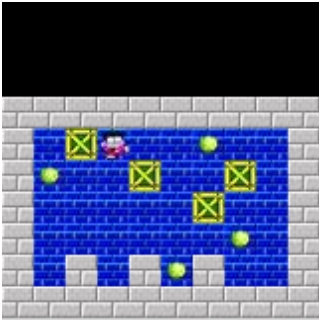
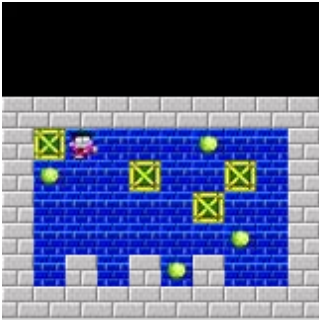
Current frame	Frame candidate	Label in training set	Desired model output / move valid?
		True	True
		False	False
		False	True

Table 3: Training data used to train the action validation model. Top row: a frame candidate extracted from a Sokoban video. Both desired model output and training label are "true". Middle row: An illegal move. Both desired model output and training label are false. Bottom row: the frame candidate was generated by the frame prediction model as a consequence of performing a random action. The desired output for this frame is "true" because the move is valid. The training label is "false", however, because this move is not observed in the Sokoban video.

This adjusted loss encouraged the model to classify moves as "false" only when the model was certain and the risk of misclassification was low. Training the model for 3,000 steps was sufficient to obtain good results. When validating the model using the Sokoban gym environment, only 15 out of 22,500 random valid moves were classified as invalid. This corresponded to a false negative rate of 0.06%. The false positive rate was 0.04%. Therefore, the model could determine with high accuracy whether a move was legal or not.

Combining the frame prediction model, the action validation model and a game state model a Sokoban simulator was created.

MODEL-BASED REINFORCEMENT LEARNING

RL with MTCS was performed as before. As a starting point, the policy and value models obtained after behavior cloning with synthetic data were used. Training using the Sokoban simulator was about 10 times slower than using a Sokoban gym environment. For each MCTS step, three DNNs had to be called:

- the frame prediction model
- the action validation model
- the game state model

Another difference was that it was no longer possible to easily detect states already visited. For instance, if the agent was in state s , moved left, and subsequently moved right, the resulting frame/state s' was not completely identical to s . There were small differences in the pixels between these states. When working with a model-free environment, a hash value for each visited state was stored in memory to remember which states had already been visited. Using the Sokoban simulator, this approach no longer worked.

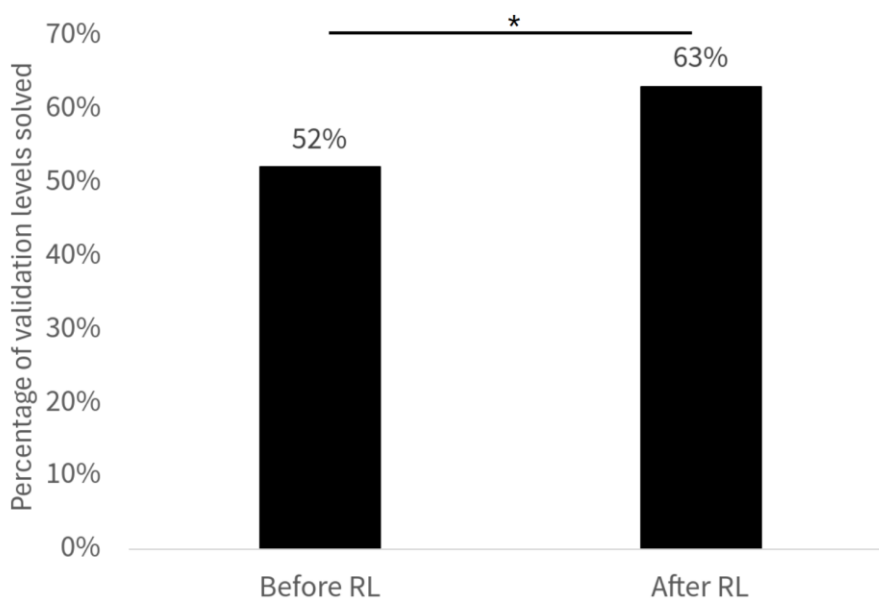


Figure 6: Model-based RL. The diagram shows the percentage of levels solved before and after RL of an agent pretrained with synthetic data via behavior cloning. The asterisk indicates a significant difference.

Nevertheless, after playing 600 games with a learning rate of $3e-7$, the playing strength of the agent had increased. The playing strength was evaluated in a model-free setting, allowing direct comparison to the results from model-based RL. The agent could solve approximately 63% of levels after RL compared

to 52% of levels before RL (Fig. 6). This difference was significant with a z-score of -2.73 corresponding to a p-value of 0.3%.

Discussion

In this work, agents learned to play Sokoban with the help of unlabeled videos. Similar work has been done with games such as Cartpole (Edwards et al., 2019), jump 'n' run games (Bruce et al., 2024), and Minecraft (Baker et al., 2022). In this thesis, the approach outlined by Edwards et al. (2019, p. 2) was applied to a turn-based strategy game.

It was possible to extract the actions taken by a player from both synthetic videos and videos downloaded from YouTube. This was done without any interaction with the Sokoban environment or the use of any labeled data. A mapping from latent actions to real actions, as done by Edwards et al. (2019, p. 2), was not required. The reason that neither remapping nor any environment interaction was needed is that some implicit knowledge was used for training. The knowledge was that there are five actions in total: four actions for each direction and one "wait" action. This knowledge went into the design of the action extraction process and made further mapping/environment interactions obsolete.

The policy and value models could be trained using the state-action pairs obtained after action extraction. The policy model could solve around one-fifth of the puzzles. This performance could be boosted to 68% with RL. Notably, RL without prior behavior cloning performed very poorly, highlighting the importance of behavior cloning. However, the Sokoban solver used for preparing synthetic data performed much better than agents trained in this thesis. This solver uses an A* search with heuristic. The models trained in this thesis are unable to compete with other Sokoban agents either (Shoham and Schaeffer, 2020; Efroni et al., 2019), some of which use deep neural networks as well (Feng et al., 2020; Shoham and Elidan, 2021).

How can the mediocre performance be explained? Naturally, none of the other agents use relatively large images as input for training. These images must first be processed or analyzed by a model. However, that is not the only or the most important difference. Successful Sokoban agents use additional hand-crafted features, rewards, or a training process specifically designed for Sokoban. For instance, Shoham and Elidan (2021) propose adding additional information such as the distance of the boxes to their respective target squares. Feng et al. (2020) propose curriculum learning to solve particularly difficult levels by breaking them down into simpler tasks.

The approach presented here to learn Sokoban from unlabeled videos aimed to be more generic. Having a more generic approach and forgoing hand-crafted features or specific training methods contributed to low performance.

In addition to learning to solve Sokoban puzzles, it was possible to infer the rules of the game from unlabeled videos. The "frame prediction model" could generate an image of the next frame, and the "action validation model" could determine if a move was valid with an accuracy of >99.9%. Using these models, a Sokoban simulation could be created. Bruce et al. (2024) provided a game simulation for jump 'n' run games based on unlabeled videos. Their publication focused on the graphical aspects of the game rather than the rules. Therefore, it might be valuable to investigate whether the rules of more complicated games or phenomena can be inferred from unlabeled videos as well.

In this work, it could be shown that using only videos as data source it is possible to perform behavior cloning, learn the rules of a strategy game and finally carry out model-based RL. Using simulated environments based on unlabeled data could be intriguing: employing simulations for RL appears particularly beneficial and may outperform using the actual environment in certain cases.

Tapping into unused data on social media platforms might help in learning other, more meaningful tasks in the future, such as:

- driving
- tasks in robotics
- learning games where no API is available
- learning to simulate games or environments without implementing them

In addition to exploring other tasks/games, efforts could focus on improving the process of utilizing unlabeled data. The current methods for extracting actions from videos are evidently imperfect, as most publications require some labeled data or interaction with the environment to perform this task.

In summary, there are numerous tasks and open questions to address when attempting to leverage unlabeled videos for machine learning.

References

- Baker, B., Akkaya, I., Zhokov, P., Huizinga, J., Tang, J., Ecoffet, A., Houghton, B., Sampedro, R., & Clune, J. (2022). Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35, 24639-24654.
<https://doi.org/10.48550/arXiv.2206.11795>
- Bruce, J., Dennis, M., Edwards, A., Parker-Holder, J., Shi, Y., Hughes, E., Lai, M., Mavalankar, A., Steigerwald, R., Apps, C., Aytar, Y., Behtle, S., Behbahani, F., Chan, S., Heess, N., Gonzalez, L., Osindero, S., Ozair, S., Reed ... & Rocktäschel, T. (2024). Genie: Generative Interactive Environments. arXiv preprint arXiv:2402.15391. <https://doi.org/10.48550/arXiv.2402.15391>
- Culberson, J. (1997). Sokoban is PSPACE-complete. Technical Report TR 97-02, Dept. of Computing Science, University of Alberta. <https://doi.org/10.7939/R3JM23K33>
- Edwards, A., Sahni, H., Schroecker, Y., & Isbell, C. (2019). Imitating latent policies from observation. In *International conference on machine learning* (pp. 1755-1763). PMLR.
<https://doi.org/10.48550/arXiv.1805.07914>
- Efroni, Y., Dalal, G., Scherrer, B., & Mannor, S. (2019). How to combine tree-search methods in reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 33, No. 01, pp. 3494-3501). <https://doi.org/10.48550/arXiv.1809.01843>
- Feng, D., Gomes, C. P., & Selman, B. (2020). Solving hard AI planning instances using curriculum-driven deep reinforcement learning. arXiv preprint arXiv:2006.02689.
<https://doi.org/10.48550/arXiv.2006.02689>
- Fryers, M., & Greene, M. (1995). Sokoban. *Eureka* (54): 25–32.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.
- Junghanns, A., & Schaeffer, J. (1998). Sokoban: Evaluating standard single-agent search techniques in the presence of deadlock. In *Advances in Artificial Intelligence: 12th Biennial Conference of the Canadian Society for Computational Studies of Intelligence, AI'98 Vancouver, BC, Canada, June 18–20, 1998 Proceedings 12* (pp. 1-15). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-64575-6_36
- Montgomery, D. C., & Runger, G. C. (2010). *Applied statistics and probability for engineers*. John Wiley & sons.

- Murase, Y., Matsubara, H., & Hiraga, Y. (1996). Automatic making of sokoban problems. In PRICAI'96: Topics in Artificial Intelligence: 4th Pacific Rim International Conference on Artificial Intelligence Cairns, Australia, August 26–30, 1996 Proceedings 4 (pp. 592-600). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-61532-6_50
- Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18 (pp. 234-241). Springer International Publishing. https://doi.org/10.1007/978-3-319-24574-4_28
- Shoham, Y., & Schaeffer, J. (2020). The FESS algorithm: A feature based approach to single-agent search. In 2020 IEEE Conference on Games (CoG) (pp. 96-103). IEEE. <https://doi.org/10.1109/CoG47356.2020.9231929>
- Shoham, Y., & Elidan, G. (2021). Solving Sokoban with forward-backward reinforcement learning. In Proceedings of the International Symposium on Combinatorial Search (Vol. 12, No. 1, pp. 191-193). <https://doi.org/10.1609/socs.v12i1.18580>
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), 484-489. <https://doi.org/10.1038/nature16961>
- Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K. & Hassabis, D. (2017). Mastering Chess and shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815. <https://doi.org/10.48550/arXiv.1712.01815>
- Torabi, F., Warnell, G., & Stone, P. (2018). Behavioral cloning from observation. arXiv preprint arXiv:1805.01954. <https://doi.org/10.48550/arXiv.1805.01954>
- Weber, T., Racaniere, S., Reichert, D. P., Buesing, L., Guez, A., Rezende, D. J., Badia, A. P., Vinyals, O., Heess, N., Li, Y., Pascanu, R., Battaglia, P., Hassabis, D., Silver, D. & Wierstra, D. (2017). Imagination-augmented agents for deep reinforcement learning. arXiv preprint arXiv:1707.06203. <https://doi.org/10.48550/arXiv.1707.06203>
- Yang, Z., Preuss, M., & Plaat, A. (2022). Transfer learning and curriculum learning in Sokoban. In Artificial Intelligence and Machine Learning: 33rd Benelux Conference on Artificial Intelligence,

BNAIC/Benelearn 2021, Esch-sur-Alzette, Luxembourg, November 10–12, 2021, Revised Selected Papers 33 (pp. 187-200). Springer International Publishing. https://doi.org/10.1007/978-3-030-93842-0_11

Zhang, Q., Peng, Z., & Zhou, B. (2022). Learning to drive by watching youtube videos: Action-conditioned contrastive policy pretraining. In European Conference on Computer Vision (pp. 111-128). Cham: Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-19809-0_7