HIEROS: Hierarchical Imagination on Structured State Space Sequence World Models

HIEROS: Hierarchische Imagination auf strukturierten Zustandsraumsequenz-Weltmodellen

Paul Mattes

Universitätsmasterarbeit
zur Erlangung des akademischen Grades

Master of Science
(M. Sc.)

im Studiengang
IT-Systems Engineering

eingereicht am 18.06.2024 am
Fachgebiet Artificial Intelligence and Sustainability der
Digital-Engineering-Fakultät
der Universität Potsdam

Gutachter	Prof. Dr. Ralf Herbrich
Betreuer	Dr. Rainer Schlosser

Abstract

One of the biggest challenges to modern deep reinforcement learning algorithms is sample efficiency. Many approaches learn a world model in order to train an agent entirely in imagination, eliminating the need for direct environment interaction during training. However, these methods often suffer from either a lack of imagination accuracy, exploration capabilities, or runtime efficiency. This thesis proposes Hieros, a hierarchical policy that learns time abstracted world representations and imagines trajectories at multiple time scales in latent space. Hieros uses an S5 layer-based world model, which shows superior long term dynamic modelling and runtime efficiency capabilities compared to previous work. Due to the special properties of S5 layers, the proposed method can train in parallel and predict next world states iteratively during imagination. This allows for more efficient training than RNN-based world models and more efficient imagination than Transformer-based world models. A time balanced sampling procedure ensures true uniform sampling of the replay dataset in $O(1)$ time. This thesis shows that Hieros outperforms the state of the art in terms of mean and median normalized human score on the Atari 100k benchmark and achieves competitive results on the DMC benchmark. It also shows that the proposed world model is able to predict complex dynamics accurately. Further experiments illustrate how Hieros displays superior exploration capabilities compared to existing approaches.

Zusammenfassung

Eine der größten Herausforderungen für moderne Deep Reinforcement Learning Algorithmen ist die Sampleeffizienz. Einige Ansätze lernen ein Weltmodell, um einen Agenten vollständig in seiner Imagination zu trainieren, was die Notwendigkeit der direkten Interaktion mit der Umgebung während des Trainings eliminiert. Diese Methoden leiden jedoch oft entweder an mangelnder Genauigkeit der Imagination, Erkundungsfähigkeiten oder Laufzeiteffizienz. Diese Arbeit stellt Hieros vor, ein hierarchischer reinforcement learning Algorithmus, der zeitlich abstrahierte Weltrepräsentationen erlernt und Trajektorien auf mehreren Zeitskalen im latenten Raum erzeugt. Hieros verwendet ein auf S5-Layern basierendes Weltmodell, das im Vergleich zu früheren Arbeiten überlegene langfristige dynamische Modellierungs- und Laufzeiteffizienzfähigkeiten zeigt. Aufgrund der speziellen Eigenschaften von S5-Layern kann die vorgeschlagene Methode parallel trainieren und während der Imagination iterativ die nächsten Weltzustände vorhersagen. Dies ermöglicht effizienteres Training als bei RNN-basierten Weltmodellen und effizientere Imagination als bei Transformer-basierten Weltmodellen. Ein zeitlich ausgewogenes Sampling-Verfahren stellt sicher, dass das Replay-Dataset in $O(1)$-Zeit echt gleichverteilt gesampelt wird. Diese Arbeit zeigt, dass Hieros den Stand der Forschung in Bezug auf den mittleren und median-normalisierten Score, den ein Mensch erreicht, auf dem Atari 100k Benchmark übertrifft und wettbewerbsfähige Ergebnisse auf dem DMC-Benchmark erzielt. Sie zeigt auch, dass das vorgeschlagene Weltmodell in der Lage ist, komplexe Dynamiken genau vorherzusagen. Weitere Experimente veranschaulichen, wie Hieros im Vergleich zu bestehenden Ansätzen überlegene Explorationsfähigkeiten zeigt.

Acknowledgments

This thesis would not have been possible without the guidance, support, and encouragement of many individuals. I extend my deepest gratitude to my advisor, Dr. Rainer Schlosser, for his unwavering support, expert guidance, and invaluable feedback throughout the research process and the preparation of this thesis. His encouragement and insightful advice were essential to the completion of this work.

I would also like to thank the professors and mentors at the Hasso Plattner Institute in Potsdam for their exceptional teaching and support, which have greatly contributed to my academic development. Their dedication to the field has provided valuable guidance and motivation throughout my studies.

Furthermore, I acknowledge the Hasso Plattner Institute for providing the necessary resources and facilities to conduct my research. I am also grateful to the German Academic Scholarship Foundation for the financial support that allowed me to focus on my studies during my Bachelor's and Master's program.

Lastly, I wish to express my profound gratitude to my family for their unwavering support and encouragement. Their belief in my abilities has been a constant source of strength and motivation. They kept me motivated at times when I struggled to find the strength to continue. This accomplishment would not have been possible without their love and understanding. Thank you, Elli, Felix, Grit, Laura, Lukas, and Rut.

Contents

1 Introduction

Since the very early days of computer science, researchers have developed ever new ways to encode complex behavior into a computational structure. Early pursuits in this field focused on developing algorithms that could mimic simple learning behaviors, laying the groundwork for more sophisticated approaches (Anyoha, 2017). As technology advanced, the advent of deep learning, which leverages neural networks to process vast amounts of data, revolutionized the landscape. This paradigm shift brought about remarkable advancements in various AI applications, from image recognition (Chai et al., 2021; Krizhevsky, Sutskever, et al., 2012) to natural language processing (Sherstinsky, 2020; Vaswani et al., 2017) to robotic tasks (Pierson and Gashler, 2017). Several different learning schemes emerged for training in different contexts, e.g. supervised learning for tasks where an abundance of labeled data is available, un-, semi- or self-supervised learning when dealing with data without labels or deep reinforcement learning in settings where the collection of and learning from the data is entirely the algorithm's responsibility. Today, deep learning is by far the most popular approach in AI research.

In supervised learning, neural networks learns from an already labeled dataset. E.g., in image classification tasks a network can be trained on a dataset of image-label pairs (e.g. MNIST (LeCun, Bottou, et al., 1998), CIFAR (Krizhevsky, G. Hinton, et al., 2009) or ImageNet (J. Deng et al., 2009)) to predict the class of an image. However, labeling data is a task that requires immense effort (Fredriksson et al., 2020). Unlabeled data on the other hand is available in way larger amounts. This is why research effort in recent years has shifted away from supervised learning (Koçyiğit et al., 2023). Unsupervised learning often focuses on discovering clusters in the data, while semi-supervised learning takes in a small portion of labeled and a larger amount of unlabeled data to identify e.g. anomalies in the training data. Recently self-supervised learning schemes have emerged, where the network is first trained on a so-called pretext task, which is a different from the task the model should be used for in the end. During the pretext training, the model is trained to identify patterns in the data itself (Rani et al., 2023). E.g. in the context of natural language processing, a popular pretext task is next word prediction (Rani et al., 2023). Then the pretrained model is fine-tuned on the actual task at hand. The fine-tuning can be conducted with a very small amount of labeled data, as the

pretext training already made the model understand the inherent structure of the training data.

Reinforcement learning (RL) is a field of machine learning that aims to solve this problem by learning a policy that maximizes the expected cumulative reward in an environment (Sutton and Barto, 2018). The agent interacts with the environment by taking actions and receiving observations and rewards. The goal of the agent is to learn a policy that maximizes the expected cumulative reward. The agent can learn this policy by interacting with the environment and observing rewards, which is called model-free RL. Some agents also learn a model of their environment and learn a policy based on simulated environment states. This is called model-based RL (Moerland et al., 2023). Deep reinforcement learning (DRL) replaces parts of the algorithms with trainable neural networks, e.g. the policy or the environment model. Training these models differs a lot from traditional deep learning schemes, as the RL model collects new data as it is interacting with the environment which poses challenges that models used for e.g. classification do not have: a dynamically increasing dataset, distributions shifts of the collected data over time or the inherent probabilistic dependence of data points on each other in the training data, which is normally not present in deep learning datasets.

DRL can be combined with all training schemes mentioned above in different ways. For example, behavioral cloning (Torabi et al., 2018) focuses on an agent directly copying behavior from a dataset of expert interactions in a specific task, which is often realized by training the policy on these interaction sequences (also called trajectories) in a supervised manner. Predicting next environment states from past observations, as it is done in model based reinforcement learning (MBRL), on the other hand, is often a self supervised training task (Hafner, Lillicrap, J. Ba, et al., 2019).

1.1 Motivation

1.1.1 Sample Efficiency of RL Algorithms

A significant hurdle encountered by RL algorithms is the lack of sample efficiency (Micheli et al., 2022). The demand for extensive interactions with the environment to learn an effective policy can be prohibitive in many real-world applications (Yampolskiy, 2018). In response to this challenge, deep reinforcement learning (DRL) has emerged as a promising solution. DRL leverages neural networks to represent and approximate complex policies and value functions, allowing it to tackle a wide array of problems and environments effectively.

1.1.2 Imagination Instead of Interaction

One such approach to boost sample efficiency is the concept of "world models". World models aim to create a simulated environment within which the agent can generate an infinite amount of training data, thereby reducing the need for expensive interactions with the real environment (D. Ha and Schmidhuber, 2018). However, a key prerequisite for world models is the construction of precise models of the environment, a topic that has garnered substantial research attention (Hafner, Lillicrap, J. Ba, et al., 2019; Hafner, Lillicrap, Fischer, et al., 2019; Kaiser et al., 2019). The idea of learning models of the agent's environment has been around for a long time (Jordan, 1992; Nguyen and Widrow, 1990; Schmidhuber, 1990). After being popularized by D. Ha and Schmidhuber (2018), world models have since evolved and diversified to address the sample efficiency problem more effectively. Most prominently, the DreamerV1-3 models (Hafner, Lillicrap, J. Ba, et al., 2019; Hafner, Lillicrap, Norouzi, et al., 2020; Hafner, Pasukonis, et al., 2023) have achieved state-of-the-art results in multiple benchmarks such as Atari100k (Bellemare et al., 2013) or Minecraft (Kanitscheider et al., 2021). These models use a recurrent state space model (RSSM) (Hafner, Lillicrap, Fischer, et al., 2019) to learn a latent representation of the environment. The agent then uses this latent representation to train in imagination. Recently, Transformers have gained popularity as backbones for world models, due to their ability to capture complex dependencies in data. One of the major drawbacks of those architectures is the inherent lack of runtime efficiency of the attention mechanism used in Transformers. Recently proposed structured state space sequence (S4) models (A. Gu, Goel, et al., 2021) show comparable or superior performance in a wide range of tasks while being more runtime efficient than Transformer-based models, which makes them a promising alternative (F. Deng et al., 2024; Lu et al., 2024).

These works which improve on existing world model architectures enhance overall sample efficiency of the entire algorithm. Exploring further ways to improve the world modelling could have a significant impact on the agent's performance.

1.2 Time Abstract Long Term Memory and Planning

Another avenue for improving RL sample efficiency is hierarchical RL (HRL) (Dayan and G. E. Hinton, 1992; Parr and Russell, 1997; Sutton, Precup, et al., 1999). These approaches often operate at different time scales, allowing the agent to learn and make decisions across multiple levels of abstraction. The idea is that higher

level policies divide the environment task into smaller subtasks or subgoals (also commonly called skills). The lower level policy is then rewarded for fulfilling these subgoals and is thus guided to fulfill the overall environment task. This approach has been shown to be effective in a variety of tasks (Hafner, Lee, et al., 2022; Y. Jiang et al., 2019; Nachum, Ahn, et al., 2019; Nachum, S. S. Gu, et al., 2018). Hafner, Lee, et al. (2022) proposes an HRL approach that builds on DreamerV2 (Hafner, Lillicrap, Norouzi, et al., 2020) and achieves superior results in the Atari100k benchmark.

LeCun (2022) theorizes that an HRL agent that learns a hierarchy of world models and features intrinsic motivation to guide exploration could potentially achieve human-level performance in a wide range of tasks. Nachum, H. Tang, et al. (2019) and Aubret et al. (2023) provide further reasoning that combining HRL with other successful approaches such as world models could lead to a significant improvement in sample efficiency. Also most popular approaches only include two levels of abstraction: a higher and a lower level. However, Hutsebaut-Buysse et al. (2022) highlight several studies that utilize multiple layers of abstraction to address more complex environments, identifying this approach as a promising direction for future research. These studies, such as Haarnoja, Hartikainen, et al. (2018), often face instabilities in their learning progress due to the increased complexity of the learning objectives. Consequently, they can sometimes be less effective than simpler, two-layer solutions. Despite these challenges, multi-layered algorithms excel at decomposing highly complex tasks into smaller subtasks across all levels. This capability makes them potentially more effective for solving complex, long-horizon planning tasks in the long run, even though they may currently face more instability than simpler models.

1.3 Scope of this Thesis

The research question this thesis examines is how multi-layered HRL can be realized with introducing world model based imagination to all layers of the hierarchy. A key requirement of this will be, to make the training of the separate world models stable and to make the layers of the hierarchy collaborate with each other to maximize the accumulated environment reward. As the added layers increase complexity and reduce runtime efficiency, the hierarchical architecture shall incorporate several strategies to alleviate the additional computational burden, such as a more efficient world model architecture or a more streamlined training strategy.

This approach needs to be evaluated on the basis of the accumulated collected reward on a diverse set of tasks in order to test it not only on tasks that it should theoretically excel in (like those that require long term planning and exploration

capabilities) but also in tasks where its specialties should not grant a significant advantage, in order to assess the general usability of the implemented approach in various contexts. The results will be compared to other state-of-the-art approaches.

Furthermore, all introduced novelties of the approach need to be tested separately in an ablation study to assess the influence of each of those additions on the overall accumulated results.

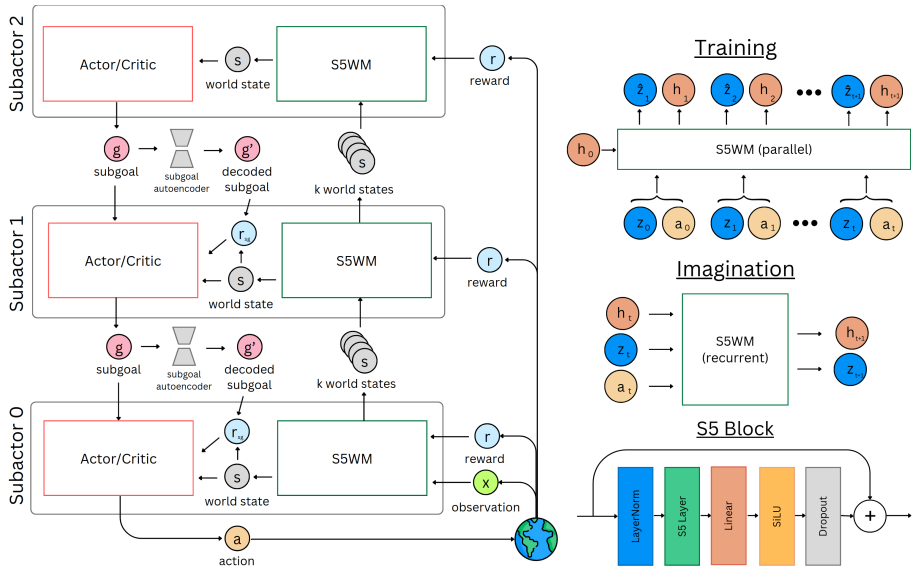

Figure 1.1: On the left: Hierarchical subactor structure of Hieros. Each layer of the hierarchy learns its own latent state world model and interacts with the other layers via subgoal proposal. The action outputs of each actor/critic is the subgoal input of the next lower layer. The output of the lowest level actor/critic is the actual action in the real environment. On the right: Training and imagination procedure of the S5WM. Hieros uses a stack of S5 blocks with their architecture shown above.

1.4 Objectives

Motivated by the findings listed above, this thesis describes the development of HIERarchical imagination On Structured state space sequence world models (Hieros), a multilevel HRL agent that learns a hierarchy of world models. For this, Hieros uses an enhanced version of the world model used in DreamerV3 (Hafner, Pasukonis, et al., 2023) and in S4WM by utilizing S5 layers (J. T. Smith et al., 2022) and implementing several further enhancements to improve the stability of the multi layered world model training.

Finding the right experience replay buffer sampling strategy is key for many RL algorithms, as it has a great influence on the final performance of the agent (D'Oro et al., 2023; Fedus et al., 2020). Robine et al. (2023) introduce a time balanced replay dataset which empirically boosted the performance of their imagination based RL agent. However, this replay procedure relies on recomputing all probabilities in $O(n)$ at each iteration, which reduces its applicability for other approaches. This thesis reformulates the sampling distortion observed during the uniform sampling procedure used in DreamerV3 (Hafner, Pasukonis, et al., 2023) in order to achieve a time balanced sampling strategy with an $O(1)$ runtime.

Specifically, this thesis builds upon prior work and presents the following contributions:

- A Hierarchical Reinforcement Learning (HRL) agent designed to learn a hierarchy of world models, facilitating the acquisition of complex and temporally abstract behaviors. This enables Hieros to project future environment states both far into the future and accurately for near-term interactions. Notably, Hieros' architecture is the first of its kind to employ hierarchical imagination within a multilevel framework, characterized by more than two layers.
- The implementation of S5WM, a world model architecture that leverages S5 layers to forecast subsequent world states. This model exhibits several advantageous properties compared to the RSSM used in DreamerV3 (Hafner, Pasukonis, et al., 2023) and several proposed Transformer-based alternatives (Chen et al., 2022; Micheli et al., 2022; Robine et al., 2023) as well as a recently proposed S4WM (F. Deng et al., 2024). The core novelty of this world model architecture lies in leveraging the efficient design of our S5WM, capitalizing on its accessibility to the internal state, to effectively model environment dynamics in the context of imagination-based reinforcement learning.
- The formulation of an efficient time-balanced sampling (ETBS) for experience dataset sampling from the time-balanced sampling method proposed by Robine et al. (2023) with a sampling time complexity of $O(1)$.
- Showing Hieros achieves a new state-of-the-art mean and median normalized human score in the Atari100k benchmark (Bellemare et al., 2013) and competitive results on the Deep Mind Control (DMC) Suite (Tassa et al., 2018). The Atari100k benchmark contains 26 Atari games that each pose a different learning problem to the RL agent, such as long term planning, sparse rewards, difficult exploration settings and accurate dynamic modeling. The agent interacts with these tasks via discrete button presses. The DMC benchmark contains 20 different robot tasks in a virtual setting where the agent controls different joints via continuous inputs. This suite also contains

different learning problems such as sparse rewards, complex mechanics and difficult exploration tasks.
- A thorough ablation study showing that combining hierarchical imagination based learning and the S5WM yields superior results. Experiments also show the impact of different design choices of HIEROS (e.g., world model choice, hierarchy depth, sampling procedure).

Figure 1.1 shows the hierarchical structure of HIEROS and the internals of the implemented S5WM. The detailed description of how the layers interact with each other and how the S5WM works will be given in Chapter 4.

1.5 Structure of the Thesis

The thesis is structured as follows. In Chapter 2 some key techniques, algorithms and fundamental works for this thesis are explained in detail to give a proper understanding of the tools used for the implementation of HIEROS. In Chapter 3 some works are highlighted that pursue a similar direction as this thesis or inspired some of the implementation details listed in Chapter 4. Chapter 5 contains the experimental evaluations of HIEROS on both the Atari100k and the Deep Mind Control Suite benchmark. This section also contains a wide range of ablations to test out the impact of different design choices for HIEROS. This thesis then ends on a conclusion of the developed method in Chapter 6 and some possible directions for future work in Chapter 7.

2 Background

HIEROS can be classified as an on-policy model based goal conditioned hierarchical deep reinforcement learning agent, which uses a world model that predicts environment dynamics in a compact latent space. In this chapter, some background concepts will be introduced which were used in developing HIEROS. Also, some important works that HIEROS was derived from will be discussed.

First, this chapter gives a basic introduction into reinforcement learning (RL) algorithms in Section 2.1, then into world models in Section 2.2. After this, Section 2.3 gives insight into the concept of hierarchical RL agents and finally the basics of sequence models are explained in Section 2.4.

2.1 Reinforcement Learning

This thesis proposes a novel reinforcement learning algorithm that is based on a model-based reinforcement learning (MBRL) approach. This section first gives a brief overview of how reinforcement learning works and then discusses important distinctions: on- and off policy algorithms, model-free and model-based reinforcement learning, value-based and policy-based reinforcement learning, deep reinforcement learning and the actor-critic algorithm.

A common training scheme when working with Deep Learning (LeCun, Y. Bengio, et al., 2015) is called supervised learning, where a model f is trained on a dataset of N input-output pairs x_i, y_i. The model learns to map inputs to outputs by minimizing the difference between its predictions and the true outputs. This is done by adjusting the model's parameters θ using an optimization algorithm such as gradient descent:

$$\underset{\theta}{\text{minimize}} \quad \mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\, f(\, x_i \,;\, \theta \,), y_i \,)$$

$\mathcal{L}$ is a loss function that measures the difference between the model's predictions and the true outputs. However, supervised learning requires a large amount of labeled data, which can be expensive and time-consuming to obtain.

Reinforcement learning (RL) is a different training scheme that does not require

labeled data. Instead, the model learns from its own experience by interacting with an environment.

An environment is defined by a state space $\mathcal{S}$, an action space $\mathcal{A}$, and a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. An agent interacts with the environment by taking actions $a \in \mathcal{A}$ based on the current state $s \in \mathcal{S}$. The agent receives feedback in the form of rewards $r(s, a)$, which depend on the state and action taken. The agent's goal is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected cumulative reward:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \, \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

γ is a discount factor that determines the importance of future rewards. The agent's policy π is a mapping from states to actions, and the goal is to learn a policy that maximizes the expected cumulative reward. $\mathcal{S}$ and $\mathcal{A}$ can be discrete or continuous. E.g. the states and actions in a board game like chess can only take discrete values, while describing e.g. the control of a robot in a real world setting often requires continuous valued actions when controlling the joints of the robot, which also results in a continuous valued state of the robot in its environment. Depending on the properties of the task, different algorithms can be applied. Figure 2.6 shows the general structure of a RL setting. The environment interactions are often stored in a so-called replay buffer, that just records the taken actions and observed state changes in the environment.

A crucial aspect of reinforcement learning environments is whether they are Markovian or non-Markovian (Hoang et al., 2023). In a Markovian environment, the future state of the environment depends only on the current state and action, not on any previous states or actions. This property simplifies the learning process since the agent only needs to consider the current state and action to predict the next state. Mathematically, it is represented as:

$$P(s_{t+1} \mid s_t, a_t) = P(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0)$$

In contrast, non-Markovian environments do not satisfy the Markov property, meaning the future state can depend on a sequence of past states and actions. This complexity requires the agent to consider a history of states and actions to make accurate predictions. Non-Markovian environments often arise when the state representation is incomplete or when hidden variables affect state transitions. Most experiments conducted in this thesis are non-markovian, as the entire world state is not always observable, so the agent needs to keep track of past interactions in some way.

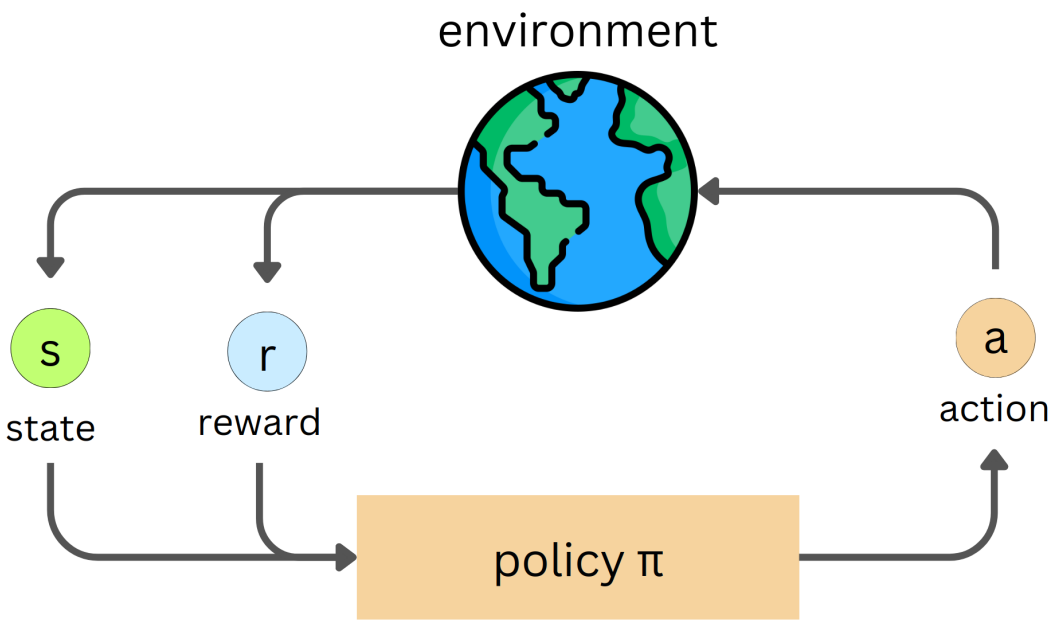

Figure 2.1: The general structure of a reinforcement learning algorithm. The policy π receives observations and rewards and interacts with the environment via actions.

RL has been successfully applied to a wide range of problems, including game playing (Silver et al., 2017; Zheng, 2019), robotics (Polydoros and Nalpantidis, 2017; T. Zhang and Mo, 2021), and natural language processing (Uc-Cetina et al., 2023; Ouyang et al., 2022).

2.1.1 Off-Policy and On-Policy Learning

One of the most popular methods for autonomous decision-making in computing is Q-learning, initially introduced by Watkins and Dayan (1992). In Q-learning, the agent estimates the expected reward for each action taken in the current state and then chooses the action that yields the maximum reward. This estimate is encapsulated in the Q-function, denoted as $Q(s, a)$.

To refine its estimates, the agent uses the Bellman equation (Bellman, 1966). This equation is used to adjust the Q-values based on the immediate rewards and the expected future rewards. The Bellman equation is defined as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big) \tag{2.1}$$

r is the immediate reward obtained upon executing action a in environment state s,

γ represents the discount factor for future rewards, s' denotes the subsequent state after taking action a, and a' represents feasible actions in state s'. The discount factor γ represents how much the agent should prioritize immediate rewards against potential future rewards. This hyperparameter is often encountered in various RL contexts, as the trade-off between immediate and prospective future reward is important in many scenarios. The parameter α denotes the learning rate, governing the extent to which newly acquired information influences existing estimates. Figure 2.2 shows a visualization of the Q-learning approach.

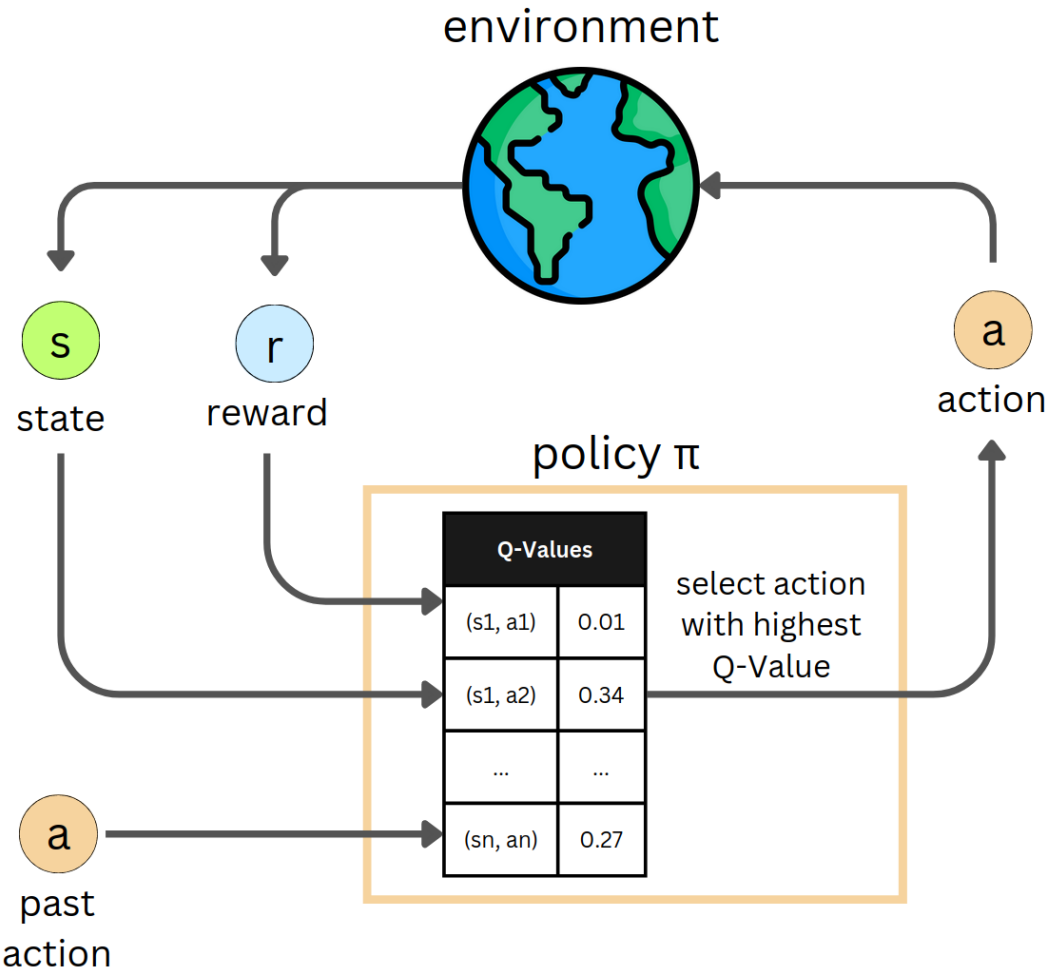

Figure 2.2: Structure of a Q-Learning algorithm. The table with Q-values is updated with the observed states and rewards, using the Bellman equation (Bellman, 1966). This is an example of an off-policy algorithm.

Q-learning operates independently of the agent's behavioral strategy, making

it an "off-policy" algorithm. Despite its conceptual simplicity, Q-learning has demonstrated effectiveness across diverse domains (Jang et al., 2019).

On the other hand, ”on-policy” algorithms, such as SARSA (Sutton and Barto, 2018), update the Q-values based on the agent's current policy. This approach is more computationally intensive than Q-learning, as it requires the agent to explore the environment more thoroughly. However, on-policy algorithms are more stable and less prone to divergence than off-policy algorithms.

SARSA (State-Action-Reward-State-Action) is a classic model-free RL algorithm that falls under the category of on-policy control methods. In SARSA, the agent learns to estimate the action-value function $Q(s, a) \longrightarrow Q(s, a)$, which represents the expected return from taking action a in state s and then following a particular policy π thereafter (as opposed to e.g. Q-Learning, where the Q values for all possible actions from state s are looked at). The algorithm updates its estimates based on observed transitions from one state-action pair to another, using the formula:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[\, r + \gamma\, Q(s', \pi(s')) - Q(s, a)\,]$$

Here, r denotes the immediate reward received upon taking action a in state s, γ represents the discount factor for future rewards, s' denotes the subsequent state after taking action a, and $\pi(s')$ represents the action selected according to the agent's policy π in state s'. The parameter α denotes the learning rate, controlling the extent to which new information overrides old estimates. SARSA is particularly well-suited for environments with a low sample complexity and in scenarios where learning online from interactions with the environment is required.

2.1.2 Model-Free and Model-Based Learning

Besides the categorization into on- and off-policy algorithms, RL approaches are often classified as model-free or model-based based on how they learn and utilize information about the environment. Model-free methods, like Q-learning, learn directly from experience without an explicit model of the environment dynamics (Çalışır and Pehlivanoğlu, 2019). They estimate the value of actions or policies solely based on observed data, updating action-value estimates using the Bellman equation to maximize rewards (see Equation (2.1)). This approach is suitable for environments with complex or unknown dynamics, such as game playing, where trial-and-error learning is possible.

In contrast, model-based methods, such as dynamic programming (Bellman, 1966) or Monte Carlo methods, explicitly build a model of the environment dynamics to make decisions (Sutton and Barto, 2018). They learn the transition probabilities

between states and expected rewards associated with state transitions. For instance, dynamic programming computes the optimal policy by iteratively estimating the value function based on known environment dynamics (Sutton and Barto, 2018). Model-based approaches are beneficial in high-dimensional or continuous state and action spaces when accurate models can be constructed, though they may struggle with high uncertainty or complex dynamics (Sutton and Barto, 2018).

Dyna-Q is an example of a model-based RL algorithm. It was introduced by (Junker and Sijtsma, 2001) as an extension to the Q-learning algorithm. Dyna-Q learns a model of the environment dynamics and uses this model to plan and make decisions.

The Dyna-Q algorithm consists of two main components: a model-learning component and a planning component. The model-learning component learns a model of the environment dynamics, which is represented by the transition probabilities $P(s'|s, a)$ of a state s' following a state s when taking action a and the reward function $R(s, a)$. The planning component uses this model to plan and make decisions. Specifically, Dyna-Q uses this model to simulate environment interactions and update the Q-values on these simulations using the already mentioned Q-learning update rule in Equation (2.1).

The key difference between Dyna-Q and Q-Learning is, that Dyna-Q uses the learned model to simulate experiences and update the Q-values, rather than relying solely on real experiences. This allows Dyna-Q to learn faster and more efficiently than model-free methods, especially in environments with high uncertainty or complex dynamics. Figure 2.3 shows a visualization of the Dyna-Q algorithm.

2.1.3 Value-Based and Policy-Based Reinforcement Learning

Value-based methods rely on estimating solely the value of a state (i.e. the immediate reward plus the prospective discounted reward) and selecting an action that simply maximizes this value. Q-Learning is an example of a value-based RL algorithm.

Policy-based methods instead directly parameterize the policy function without explicitly estimating the value of a state. A very popular policy-based approach is the REINFORCE algorithm, introduced by (Williams, 1992). It optimizes the policy by adjusting its parameters to maximize the expected cumulative reward. The algorithm estimates the gradient of the expected reward with respect to the policy parameters θ:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a \mid s)\, R]$$

where $\pi_\theta(a|s)$ is the policy function parameterized by θ, a is the action, s is the state, and R is the total reward obtained from the state s forward up until the end of the

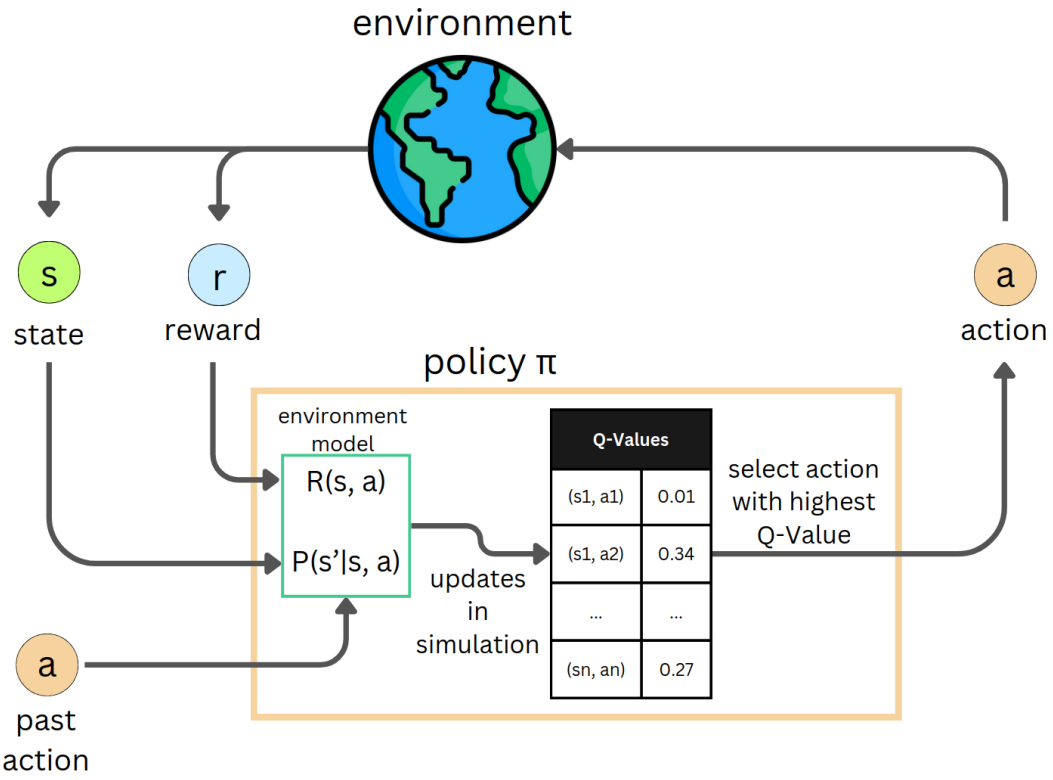

Figure 2.3: Structure of the Dyna-Q algorithm. The agent learns a model of state transition probabilities and observed rewards and uses this model to update its Q-values in simulation instead of by directly interacting with the environment. This is an example of a model-based RL algorithm.

episode. To compute this total reward, REINFORCE samples a complete trajectory from the environment (i.e. interacts with the environment until the episode ends) before computing gradients. The policy parameters θ are then updated the following way for all pairs of (a, s) of the sampled trajectory:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a \mid s)\, R$$

where α is the learning rate. Despite its simplicity, REINFORCE often suffers from high variance in the gradient estimates, which can impede convergence and make learning less stable (J. Zhang et al., 2021). Techniques such as baseline subtraction are used to reduce this variance, improving the algorithm's performance and stability (A. Agarwal et al., 2019). Figure 2.4 shows the structure of the REINFORCE algorithm.

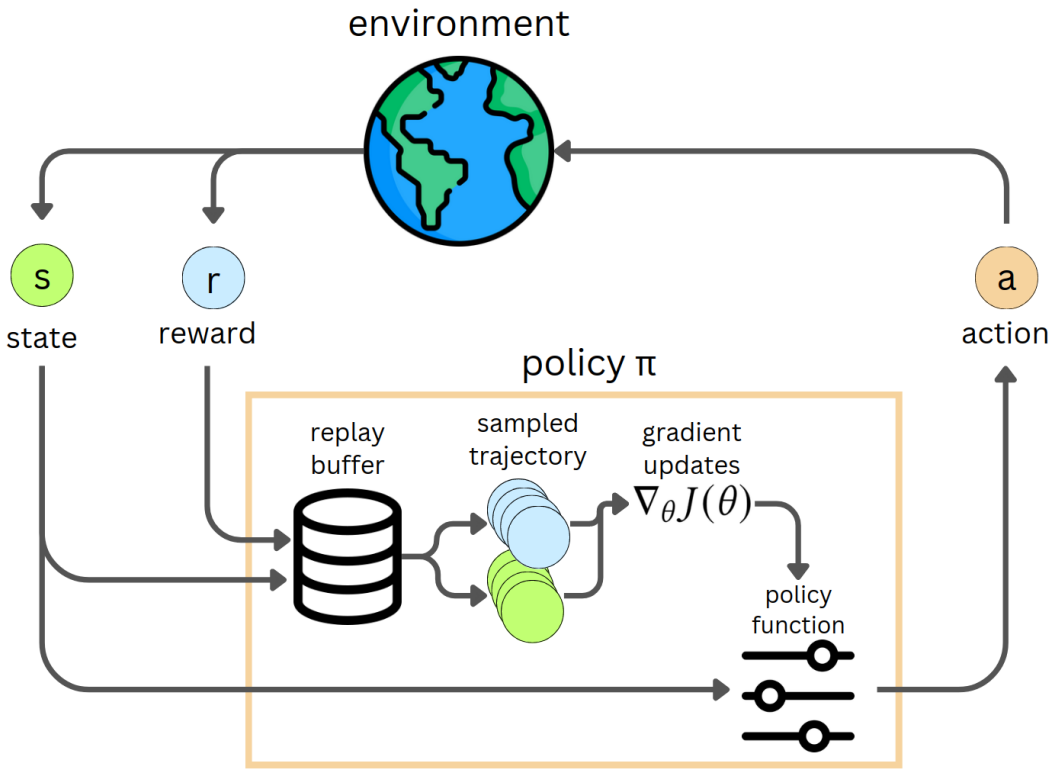

Figure 2.4: Structure of the REINFORCE algorithm. The actor function updates its parameter based on the gradients computed on the sampled trajectories from the replay buffer. This is an example of a policy-based RL algorithm.

2.1.4 Deep Reinforcement Learning

All above-mentioned approaches can be implemented using simple functions, such as lookup tables, for small state and action spaces. However, in many real-world scenarios like e.g. controlling a robot in a real-world environment, the state and action spaces are exceedingly large or continuous, making it impractical to use such simple representations. Deep Reinforcement Learning (DRL) can be used to effectively tackle this problem, by approximating value functions, policies, and models with neural networks. These DRL agents can then be scaled up to solve very large and very complex tasks.

Deep Q-Networks (DQNs) are a prominent example of DRL methods that extend Q-learning to high-dimensional state spaces using a neural network to approximate the Q-function (Mnih et al., 2013). DQNs employ replay buffers and target networks to stabilize training and prevent divergence. Replay buffers store the agent's

interactions, allowing the algorithm to learn from a diverse set of past experiences. Target networks, having the same architecture as the Q-networks, periodically receive the parameters from the Q-network to provide a smoothed and reliable Q-function estimation. Figure 2.5 visualizes the DQN algorithm.

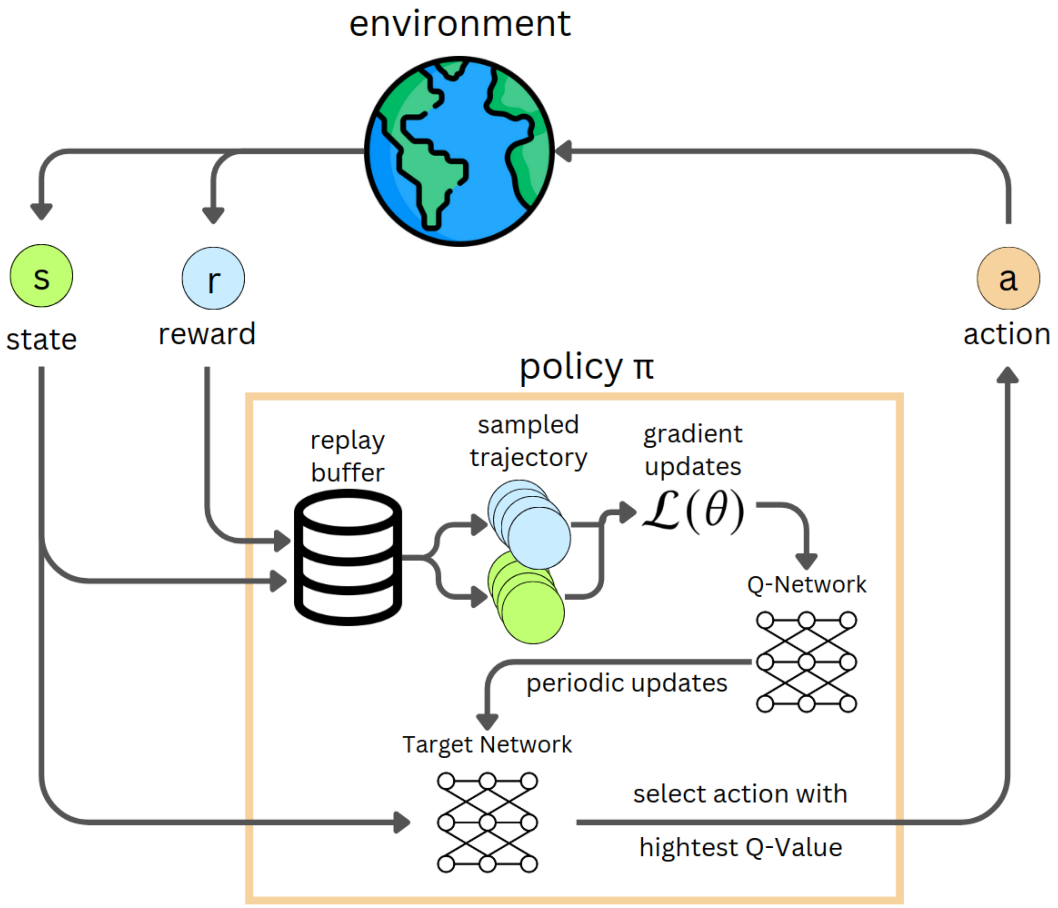

Figure 2.5: Structure of Deep Q-learning. The Q-network is updated based on past interactions stored in the replay buffer. The target network has the same architecture as the Q-network, and its parameters are periodically updated to those of the Q-Network. The target network is used for the actual Q-value estimation. This is an example of a Deep RL algorithm.

Policy-based methods have also benefited from the integration of deep learning. Algorithms such as Trust Region Policy Optimization (TRPO) (Schulman, Levine, et al., 2015) and Proximal Policy Optimization (PPO) (Schulman, Wolski, et al., 2017) use neural networks to parameterize policies and optimize them using gradient

ascent techniques. These methods address some of the stability issues inherent in policy gradient methods by imposing constraints on the policy updates, which produces more reliable and robust learning.

2.1.5 Actor-Critic Reinforcement Learning

HIEROS uses a model of its environment to update its policy. The basis for the policy learning of HIEROS is the soft actor-critic (SAC) RL method.

Actor-Critic RL combines value-based and policy-based methods in a single model (Konda and Tsitsiklis, 1999). The actor learns a policy mapping states $s \in S$ to actions $a \in A$, while the critic evaluates the actor's actions by estimating the value function $V(s)$ or the action-value function $Q(s, a)$. The actor updates its policy based on the critic's feedback, calculated using the temporal difference error δ, which is the difference between the estimated value of the current state and the sum of the immediate reward and the estimated value of the next state.

$$\delta = r + \gamma V(s') - V(s)$$

r is the observed reward, γ is the discount factor, s is the current state, and s' is the next state. By iteratively updating the actor and critic parameters, Actor-Critic algorithms aim to find an optimal policy that maximizes the expected cumulative reward over time. Actor-critic algorithms have been applied in a large variety of different contexts (Grondman et al., 2012). In practice, the Actor- and Critic-functions are often implemented as neural networks and updated via deep learning.

Soft Actor-Critic (SAC) (Haarnoja, Zhou, Abbeel, et al., 2018) is an extension of the actor-critic framework and focuses on maximizing entropy. SAC aims to maximize the collected reward and encourage exploring by maximizing the policy's entropy. This strategy proves helpful especially in uncertain or complex environments. The objective of SAC is to maximize the expected return along with the expected entropy of the policy, defined as:

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \left(r_t + \alpha \mathcal{H}(\pi(\cdot \mid s_t))\right)\right]$$

Here, $\mathcal{H}(\pi(\cdot \mid s_t))$ represents the entropy of the policy at state s_t, and α is a temperature parameter that balances the trade-off between exploration (entropy) and exploitation (reward maximization).

An important part of SAC is slow targets, also called target networks, to make training stable. SAC uses the same target networks as DQN, which helps it to

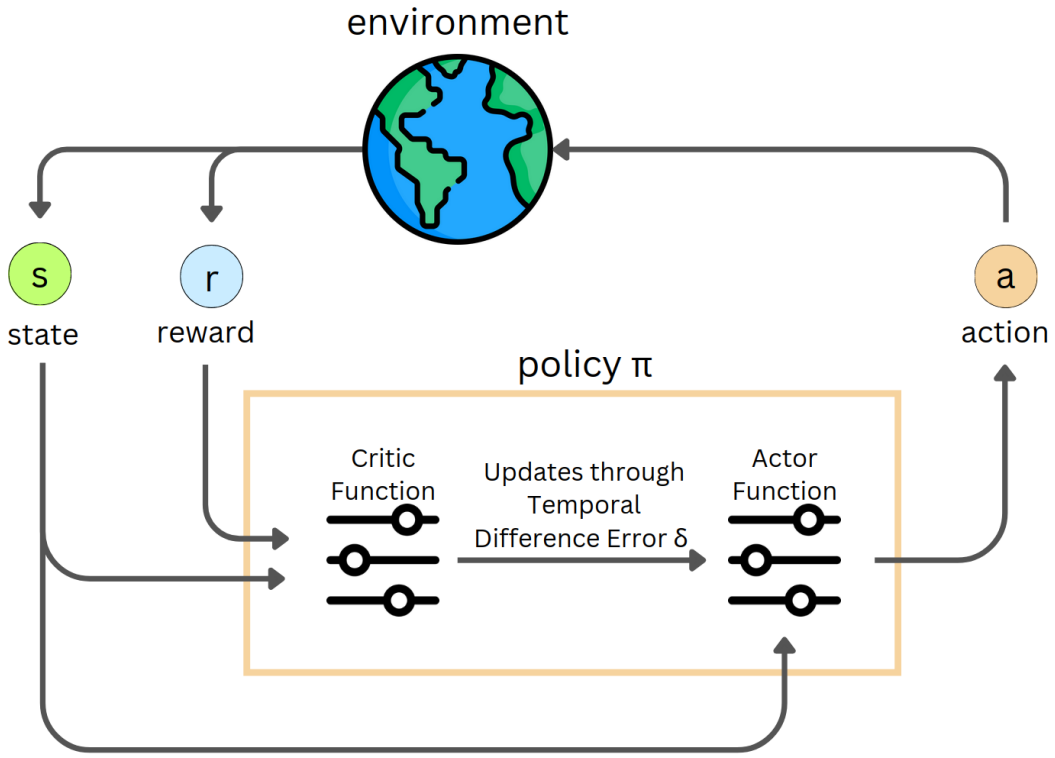

Figure 2.6: Structure of the actor-critic RL algorithm. The critic estimates the value of the current state, and the actor updates its policy based on the temporal difference error δ computed from the output of the critic.

reduce sudden changes in the value estimates, resulting in more stable training. SAC's has been successfully applied to a wide range of problems such as real world and simulated robotic control (Haarnoja, Zhou, Hartikainen, et al., 2018) or video games (Christodoulou, 2019).

2.2 World Models

As explained in Section 2.1, model-based reinforcement learning (MBRL) is a powerful tool to increase sample efficiency in RL. World models are a specific type of model-based reinforcement learning that learns a model of the environment it interacts with. This model can then be used to plan and learn a policy. World models have been shown to be able to generate realistic trajectories which can be used to train a policy (D. Ha and Schmidhuber, 2018).

In their original work, D. Ha and Schmidhuber (2018) proposed a world model that consists of three components: an autoencoder, a sequence model and an actor network. The autoencoder, also called Vision (**V**) module in the paper, is used to compress the environment observations into a compact latent representation z_t. The actor, called controller (**C**) in the paper, takes this latent state z_t together with the last hidden state h_{t-1} of the sequence model (or Memory (**M**)) and outputs an action a_t. **M** uses z_t and a_t to update its internal hidden state to h_t, which will be used in the next iteration. The sequence model gives the controller context information about the environment, as the past interactions are encoded in the hidden state as well as broader information about the current state of the environment, which might only be partially observable by **C**. This way, the **M** acts as a world model for the controller module.

The concept of learning a world model is in itself not really new, as early approaches date back to the 1990s (Sutton, 1991). However, the recent advances in deep learning have enabled the development of more powerful world models that are able to learn more complex environments. The original world model by D. Ha and Schmidhuber (2018) was trained in a self supervised manner (i.e. without any external reward signal and without outside guiding) and was able to learn to play the game *CarRacing* from pixel observations. The world model was able to generate realistic trajectories and which enabled the agent to learn a policy that achieved a high reward in the *CarRacing* game.

2.2.1 Prediction in Observation Space vs. Prediction in Latent Space

There have been many works building on this concept of world models. (Friston et al., 2021; Guan et al., 2024; Hafner, Lillicrap, J. Ba, et al., 2019; Micheli et al., 2022). One important distinction between different approaches is whether the world model predicts the next observation in the original observation space or in a latent space. The original world model by D. Ha and Schmidhuber (2018) predicts the next observation in the original observation space. This has the advantage that the world model can be trained directly with a reconstruction loss, which is a simple and effective way to train the world model. However, predicting the next observation in the original observation space can be difficult, especially for high dimensional observations like images. This is because the model has to learn to predict the exact pixel values of the next observation, which can be difficult. Especially noisy environments pose a significant challenge to prediction models in the observation space, as the image cannot be predicted exactly.

Predicting the next observation in a latent space can be easier, because the model only has to predict a lower dimensional latent state, which ideally abstracts from the pure observation and reduces noise. However, predicting the next observation in a latent space requires a decoder that can map the latent state back to the observation space. How to learn meaningful representations is still an open research question (Y. Bengio et al., 2013; Nozawa and Sato, 2022; D. Zhang et al., 2018).

2.2.2 The Dreamer Architecture

HIEROS, the approach developed in this thesis, is build upon the DreamerV3 architecture. Hafner, Lillicrap, Fischer, et al., 2019 introduce a world model architecture called recurrent state-space model (RSSM). DreamerV2 and DreamerV3 adapt this world model which consists of four different networks:

Deterministic state model	$h_t = f(h_{t-1}, s_{t-1}, a_{t-1})$
Stochastic state model	$s_t \sim p(s_t \mid h_t)$
Observation model	$o_t \sim p(o_t \mid h_t, s_t)$
Reward model	$r_t \sim p(r_t \mid h_t, s_t)$
Encoder model	$s_t \sim p(s_t \mid o_t, a_{t-1})$

with h_t being the deterministic part of the world state at time step t which is predicted by f, s_t is the stochastic part of the world state sampled from a distribution $p(s_t|h_t)$, a_t the action taken, o_t the original observation from the environment, a_{t-1} the action taken leading to observation o_t and r_t the observed reward. The encoder produces latent representations s_t of the current environment state (o_t, a_t). f is the network that predicts the next world state and is implemented as an RNN (Hafner, Lillicrap, Fischer, et al., 2019), and predicts next world states in a latent space consisting of the tuple (h_t, s_t). The split of the world state into a deterministic part and a stochastic part allows the model to access information from multiple time steps in the past (as it is encoded by the RNN in the deterministic latent state h_t) as well as capture world states with partial observability in the stochastic part of the state. If an environment is only partially observable, the agent does not have enough information about all environment dynamics, so some state transactions cannot be reliably predicted. The stochasticity of s_t allows the policy during training in the imagination of the world model to see multiple possible state transitions from the same previous deterministic state (Hafner, Lillicrap, Fischer, et al., 2019).

Based on the RSSM world model, they develop a series of RL agents called Dreamer V1-V3 (Hafner, Lillicrap, J. Ba, et al., 2019; Hafner, Lillicrap, Norouzi,

et al., 2020; Hafner, Pasukonis, et al., 2023). Dreamer models are trained entirely in the imagination of their world model. This is done by simultaneously predicting world states and policy actions. During training, the RSSM receives an initial observation o_1 and an initial deterministic state h_0 and the last action a_0 that lead to the observation o_1. The encoder of RSSM generates the initial $s_0 \sim p(s_0|o_1, a_0)$.

Dreamer then produces a sequence $t \in \{1, \cdots, n\}$ of world states and actions by using the stochastic world states s_t as input for the policy network, which then chooses the next action.

$$\begin{aligned}(h_t, s_t) &= f(h_{t-1}, s_{t-1}, a_{t-1}) \\ r_t &\sim p(r_t \mid h_t, s_t) \\ a_t &= \pi(s_{t-1})\end{aligned}$$

with h_t being the deterministic part of the latent state, s_t being the stochastic part of the latent state and a_t being the action taken at time step t. Dreamer conditions the policy only on the stochastic s_t to enable the agent to deal with uncertainty and stochasticity. r_t is the predicted reward at time step t. This imagination procedure allows Dreamer to train a soft actor-critic (as explained in Section 2.1.5) entirely on the generated trajectories instead of using direct environment interactions.

The world model is trained with a loss function that is a weighted sum of the loss of the dynamic, observation, continue, and reward prediction. The details of this loss function vary between different versions of the Dreamer model. While DreamerV1 samples s_t from a continuous function, DreamerV2 (Hafner, Lillicrap, Norouzi, et al., 2020) introduces the notion of discrete world models, sampling s_t from a discrete categorical distribution of 32 classes. This guides the world model to learn highly expressive world states by introducing sparsity, which often helps the model generalizing. HIEROS is build upon the DreamerV3 (Hafner, Pasukonis, et al., 2023) model, which introduces several small improvements compared to DreamerV2, most notably Symlog normalization of model inputs and regularizing terms for the world model loss based on the concept of free bits (Durk P Kingma et al., 2016).

The world model of DreamerV3 consists of the following networks:

$$\begin{aligned}\text{Sequence model:} \quad & h_t = f_\theta(h_{t-1}, z_{t-1}, a_{t-1}) \\ \text{Encoder:} \quad & z_t \sim q_\theta(z_t \mid h_t, o_t) \\ \text{Dynamics predictor:} \quad & \hat{z}_t \sim p_\theta(\hat{z}_t \mid h_t) \\ \text{Reward predictor:} \quad & \hat{r}_t \sim p_\theta(\hat{r}_t \mid h_t, z_t) \\ \text{Continue predictor:} \quad & \hat{c}_t \sim p_\theta(\hat{c}_t \mid h_t, z_t)\end{aligned}$$

$$\text{Decoder:} \quad \hat{o}_t \sim p_\theta(\hat{o}_t \mid h_t, z_t)$$

with θ being the parameters of the world model, h_t the deterministic state, z_t the discrete stochastic state (which replaces the previous continuous valued s_t, and o_t the observation. $\hat{z}_t$ is the stochastic state predicted solely from h_t, which is used during imagination. As z_t is predicted from the known observation from the environment, it is also often called the posterior and $\hat{z}_t$ is often called the prior. Aligning these two predictions is key in order to have consistent prediction of world states during the imagination phase without explicit observations from the environment. $\hat{r}_t$, $\hat{c}_t$ and $\hat{o}_t$ are the predicted reward, continue signal (indicating if the current step t is the end of the episode or not) and the reconstructed observations.

During training, the networks are all jointly optimized using a single loss term. The world model loss of DreamerV3 is calculated as follows:

$$\begin{aligned}
\mathcal{L}(\theta) &= \mathcal{L}_{pred}(\theta) + \alpha_{dyn} \cdot \mathcal{L}_{dyn}(\theta) + \alpha_{rep} \cdot \mathcal{L}_{rep}(\theta) \\
\mathcal{L}_{pred}(\theta) &= -\ln(\, p_\theta(r_t \mid h_t, z_t)) \; - \ln(p_\theta(c_t \mid h_t, z_t)) \\
&\quad - \ln(p_\theta(o_t \mid h_t, z_t)) \\
\mathcal{L}_{dyn}(\theta) &= \max(\, 1, \mathrm{KL}[\, \mathrm{sg}(\, q_\theta(s_t \mid h_t, o_t)) \,||\, p_\theta(s_t \mid h_t)\,]) \\
\mathcal{L}_{rep}(\theta) &= \max(\, 1, \mathrm{KL}[\, q_\theta(z_t \mid h_t, o_t) \,||\, \mathrm{sg}(\, p_\theta(z_t \mid h_t))\,])
\end{aligned}$$

with $\mathcal{L}_{pred}$ being the loss term for the predictors, $\mathcal{L}_{dyn}$ the loss term for the dynamics prediction and $\mathcal{L}_{rep}$ the loss term for the representation learning. α_{dyn} and α_{rep} are the weights for $\mathcal{L}_{dyn}$ and $\mathcal{L}_{rep}$ and are set as hyperparameters before the training. KL[] denotes the Kullback-Leibler Divergence (Kullback and Leibler, 1951) between two distributions. $\mathrm{sg}(\cdot)$ denotes, that the gradients of $\cdot$ will not be computed for this loss term. So e.g. for $\mathcal{L}_{dyn}$, the dynamics predictor $p_\theta(z_t|h_t)$ is optimized by minimizing this loss while the encoder $q_\theta(z_t|h_t, o_t)$ will not be changed. The clipping of $\mathcal{L}_{dyn}$ and $\mathcal{L}_{rep}$ is to avoid the predictors to produce degenerate solutions where the representations contain no meaningful information and are thus their dynamics is easy to predict.

This concept utilizes the notion of "free bits" (Durk P Kingma et al., 2016). To prevent optimization from being trapped in undesirable equilibria, it ensures that using less information in the network than a certain threshold, typically denoted by a constant (in our case, $\lambda = 1$), does not provide any advantage. In other words, reducing the amount of information in the network beyond this threshold does not lead to further loss reduction.

Figure 2.7 shows the structure of DreamerV3 during environment interaction and during training as it is depicted in Hafner, Pasukonis, et al. (2023).

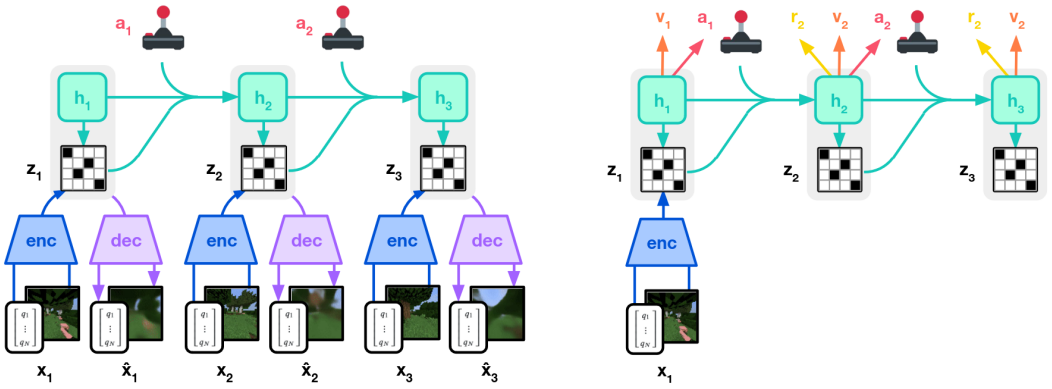

Figure 2.7: Structure of DreamerV3 as depicted in Hafner, Pasukonis, et al. (2023). On the left is the model during environment interaction. The observation from the environment x_t is encoded by *enc* and used together with the latent discrete state h_t to predict z_t. This prediction of z_t from both the observation and h_t is called the posterior. The policy then uses the world state h_t, z_t to produce actions a_t. In the interaction phase, these are directly passed to the outside environment. On the right is DreamerV3 during imagination. The model takes an initial observation x_1 and predicts z_1 from h_t and x_t. After this, the agent outputs an agent and the world model predicts the next deterministic world state h_2. During imagination z_t is only predicted from h_t. As this prediction is without outside knowledge, z_t is predicted a priori. The policy is then trained entirely on the imagined trajectories instead of real environment interactions.

2.3 Hierarchical Reinforcement Learning

Hierarchical reinforcement learning (HRL) is a field of RL that breaks down complex tasks in time and state abstracted subproblems on multiple time scales (Hutsebaut-Buysse et al., 2022; Sutton, Precup, et al., 1999). This subtask definition can be done manually (Tessler et al., 2017) or automatically (Kujanpää et al., 2023; J. Li et al., 2022; Nair and Finn, 2019). This allows the agent to learn subtasks on different time scales and to reuse these subtasks in different contexts. This is especially useful in sparse reward environments, where the agent can learn subtasks that are easier to solve and then combine them to solve the overall task (Nair and Finn, 2019). Hierarchical models have been shown to be a powerful tool in various RL contexts (Dayan and G. E. Hinton, 1992; Parr and Russell, 1997; Sutton, Precup, et al., 1999).

Goal-conditioned HRL (Florensa et al., 2017; Nachum, Ahn, et al., 2019; Nachum, S. Gu, et al., 2018; Nachum, S. S. Gu, et al., 2018) is a subfield of HRL that breaks down complex tasks into subgoals and conditions policies to fulfill these subtasks. The agent learns a policy that takes a goal as input and outputs actions that lead to

the goal. This allows it to learn subtasks on different time scales and to reuse these subtasks in different contexts. A predefined or learned function proposes subgoals in frequent intervals that the HRL policy has to fulfill, which is often incentivized by giving an intrinsic reward for reaching the assigned goals (Hafner, Lee, et al., 2022; Rosete-Beas et al., 2023).

HRL typically involves learning a high level actor that works at larger timescale and proposes subgoals and a low level actor that executes the proposed subgoals (Hafner, Lee, et al., 2022; Y. Jiang et al., 2019; Nachum, Ahn, et al., 2019; Nachum, S. S. Gu, et al., 2018):

$$g_t = \pi_{high}(s_t) \tag{2.2}$$

$$a_t = \pi_{low}(s_t, g_t) \tag{2.3}$$

with s_t being the current environment state, g_t being the proposed subgoal and a_t being the action taken by the low level actor. The low level actor is often encouraged to fulfill the subgoal by adding a subgoal (or intrinsic) reward r_g to the observed environment (or extrinsic) reward r_{extr}. This subgoal reward is often a function of the distance between the current state and the proposed subgoal (Hafner, Lee, et al., 2022; Nachum, Ahn, et al., 2019; Okudo and Yamada, 2021; Paul et al., 2019).

2.3.1 The Director Architecture

Hafner, Lee, et al., 2022 introduce a hierarchical RL agent build on top of the DreamerV1 architecture (Hafner, Lillicrap, Norouzi, et al., 2020) called Director. They use the regular DreamerV1 world model and actor-critic structure but add a so-called manager π_{high} which proposes subgoals for the actor-critic worker policy π_{low} to achieve. For this, a subgoal autoencoder g, which compresses world model states s_t into a smaller discrete subgoal space of 8 categorical one-hot vectors with 8 classes. π_{high} selects a subgoal g_t which is then decoded by the autoencoder back into the model state space s_t^g.

The π_{high} policy is conditioned on the environment rewards while the π_{low} receives subgoal rewards computed using a max-cosine function:

$$r_t^g = (s_t^g/m)^T(s_t/m) \qquad \text{where} \qquad m = \max(\,||\,s_t^g\,||\,,\,||\,s_t||\,)$$

They tested several different distance metrics (e.g. cosine similarity, mean squared error, etc.) and found that this function produced best results. Remarkably, the π_{low} policy is not conditioned on the environment rewards but exclusively on the subgoal reward. This makes the low level policy in principle task independent.

The manager policy updates the proposed subgoal every 8 time steps. However, they show in their ablation study that providing the worker policy also with the extrinsic rewards often improves the agent's performance significantly.

Also, Director rewards the higher level policy with a novelty reward r_{nov} which is defined as the reconstruction error of the subgoal autoencoder g on the compressed world states s_t:

$$r_t^{nov} = || g_{dec}(g_{enc}(s_t)) ||_2$$

with g_{dec} being the decoder and g_{enc} being the encoder of the subgoal autoencoder g. $|| \cdot ||_2$ is the l2-norm function. This novelty reward should incentivize the upper level policy to guide the worker policy towards states that it has not seen before, therefore encouraging exploration in the environment. Figure 2.8 shows the overall structure of director as it is depicted in Hafner, Lee, et al. (2022).

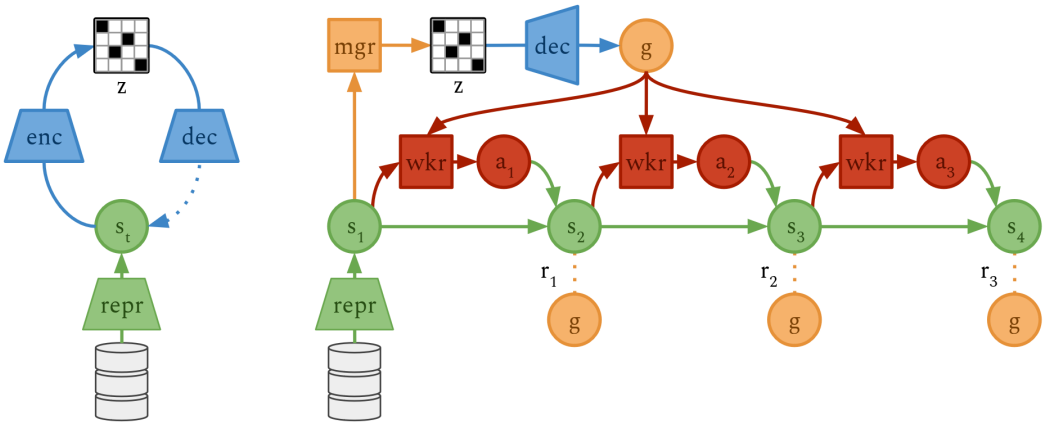

Figure 2.8: The internal structure as depicted in Hafner, Lee, et al. (2022). On the left is the encoding and decoding of latent world states into discrete subgoals by the subgoal encoder *enc* and the subgoal decoder *dec*. On the right is the complete workflow of director during imagination. The initial observation stored in the replay buffer is compressed into its latent representation s_1 by the *repr* module. Then this latent representation is used by the manager policy *mgr* to propose a discrete subgoal z which is decoded into g. The worker policy then trains in imagination and is rewarded by the max-cosine similarity between g and the achieved s_t. The proposed subgoal is only updated every 8 steps of the worker policy.

2.4 Sequence Models

Sequence models are a class of machine learning models designed to handle sequential data, where the order of the data points is significant. These models are particularly effective for tasks where context and temporal dynamics play a crucial role, such as natural language processing (NLP), time series forecasting, speech recognition. In the Dreamer framework, a sequence model is used to predict environment dynamics from past interactions. The essential goal of sequence models is to capture dependencies and patterns across sequences, enabling accurate predictions and generation of sequential data.

A sequence model takes an input sequence $x = (x_1, x_2, \ldots, x_T)$ and processes it to produce an output sequence $y = (y_1, y_2, \ldots, y_T)$. The relationship between the input and output sequences can be formulated as follows:

$$y_t = f(x_1, x_2, \ldots, x_t; \theta)$$

with f being the sequence model function parameterized by θ. Depending on the application, the model can be designed to handle various sequence lengths and structures.

2.4.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are very basic networks and one of the first deep learning approaches for sequential data. They maintain a hidden state h_t that captures information from previous time steps. At each time step t, the hidden state is updated as follows:

$$h_t = \tanh(W_h x_t + U_h h_{t-1})$$

with x_t being the input, W_h and U_h being weight matrices, and tanh is the activation function (Rumelhart et al., 1986). The hidden state h_t then produces the output y_t:

$$y_t = W_y h_t$$

RNNs can struggle with long-term dependencies due to issues like vanishing gradients, which advanced models like LSTMs and GRUs address with specialized gating mechanisms (Cho et al., 2014; Hochreiter and Schmidhuber, 1997). Figure 2.9 shows the structure of an RNN predicting a sequence.

The RSSM used in the Dreamer models (Hafner, Lillicrap, J. Ba, et al., 2019;

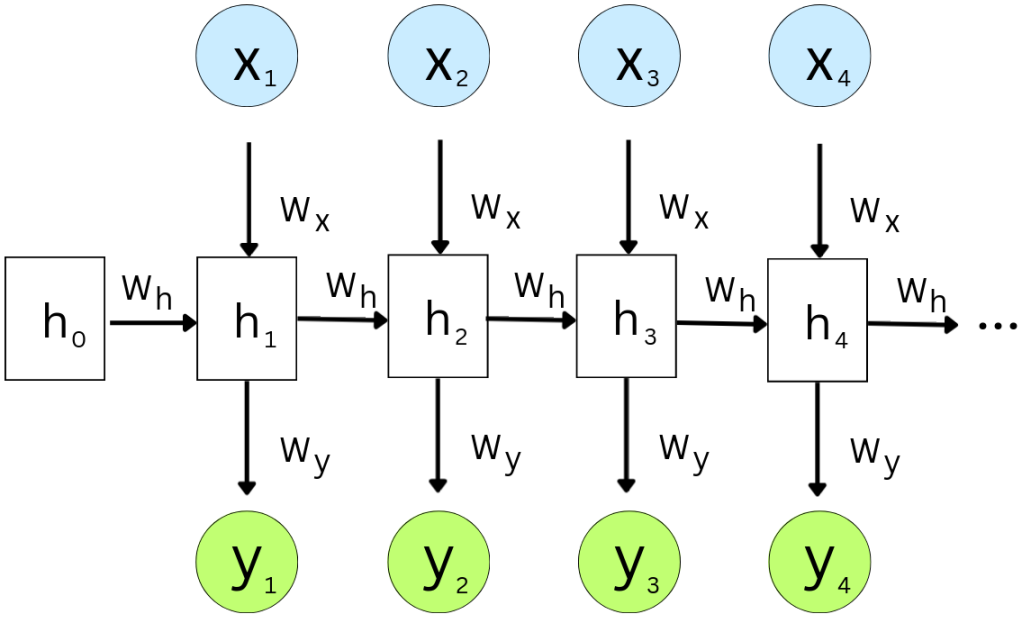

Figure 2.9: Simple depiction of RNNs. The module takes in a sequence of $x_{1:n}$ and outputs a sequence $y_{1:n}$, producing a sequence of hidden states $h_{1:n}$. W_x, W_y, and W_h are learned matrices.

Hafner, Lillicrap, Norouzi, et al., 2020; Hafner, Pasukonis, et al., 2023) utilizes a so-called Gated Recurrent Unit (GRU) introduced by Cho et al. (2014).

A GRU consists of two main gates: the update gate and the reset gate. These gates control the flow of information and enable the model to retain or discard information as needed. The update gate determines how much of the past information needs to be passed along to the future, while the reset gate decides how much of the past information to forget.

A GRU can be defined as follows:

$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1}) \\
r_t &= \sigma(W_r x_t + U_r h_{t-1}) \\
\tilde{h}_t &= \tanh(W_h x_t + U_h(r_t \odot h_{t-1})) \\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t
\end{aligned}
$$

where x_t is the input at time step t, h_{t-1} is the previous hidden state, z_t is the update gate, r_t is the reset gate, and $\tilde{h}_t$ is the candidate hidden state. W_z, W_r, W_h, U_z, U_r, and U_h are weight matrices, and σ represents the sigmoid activation function.

2.4.2 Transformer Models

While RNNs and GRUs are effective for handling sequential data, they can struggle with long-term dependencies and parallelization issues due to their sequential nature. Transformers address these shortcomings by employing a self-attention mechanism that allows for more efficient handling of long-range dependencies and parallel processing of sequences (Reza et al., 2022).

Transformers, introduced by Vaswani et al., 2017, use the so-called self-attention mechanism to weigh the importance of different elements in a sequence when making predictions. A key component of the Transformer architecture is the scaled dot-product attention, which is defined as:

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

where Q (queries), K (keys), and V (values) are matrices derived from the input, and d_k is the dimensionality of the keys. Basically, self-attention allows for data dependent weight matrices to be multiplied with the input sequence as opposed to RNNs, where the weight matrix is fixed and multiplied with all input data points. This allows for capturing and accumulating information from the entire sequence efficiently. The self attention mechanism has a runtime complexity of $O(n^2)$ with n being the length of the input sequence, as the self attention has to be computed for each pair of inputs x_i, x_j with $i, j \in \{1, \cdots, n\}$.

The Transformer models used for environment dynamics modeling in RL often consist of multiple layers of self-attention and multi layer perceptrons (also often called feed forward networks). This architecture allows for efficient parallelization during training, significantly speeding up the process compared to RNNs and GRUs (Chen et al., 2022; S. B. David et al., 2022).

During interaction with the environment, however, they often lack efficiency, as they need a context of multiple past interactions in order to infer the next output, whereas RNNs or GRUs can predict the next step solely from the very last state in $O(1)$. Also, the runtime complexity of the self-attention mechanism is $O(n^2)$ with n being the sequence length. Transformers have achieved state-of-the-art performance in various tasks, including machine translation, text generation, and more (Lin et al., 2022).

2.4.3 Structured State Space Sequence Models

Structured state space sequence models (S4) were initially introduced by A. Gu, Goel, et al. (2021) as a sequence modeling method that is capable of achieving long-term memory tasks superior to Transformer-based models while having a lower runtime complexity ($O(n^2)$ for the attention mechanism of Transformers and $O(n \log n)$ for S4, n being the sequence length). State Space models are composed of four matrices: A, B, C, and D. They take in an input x_t and output a signal y_t:

$$u_{t+1} = Au_t + Bx_t \tag{2.4}$$

$$y_t = Cu_t + Dx_t \tag{2.5}$$

with u_t being the state of the model at time t. The matrices A, B, C, and D are learned during training. Technically this formulation only holds for continuous time state spaces, so for the S4 layer to operate on discrete time intervals (i.e. $t \in \mathbb{m}$), these matrices need to be discretized using e.g. Euler, bilinear or zero-order hold (ZOH) methods (A. Gu, Goel, et al., 2021). The formulation above shows how the S4 layer operates in the iterative setting, where $y(t)$ only depends on x_t and the last hidden state u_t. In this mode, it basically operates like an RNN.

S4 can also be applied in parallel through the use of the Convolution Theorem, which states that convolution in the time domain is equivalent to pointwise multiplication in the frequency domain (S. Kumar, 2018). The sequence data is first transformed into the frequency domain using the Fast Fourier Transform (FFT) (Brigham and Morrow, 1967), represented as $\mathcal{F}(u(t)) = U(f)$.

In the frequency domain, the state-space dynamics are applied, where the convolution operation becomes a simple element-wise multiplication: $X(f) = \mathcal{F}(A) \cdot X(f) + \mathcal{F}(B) \cdot U(f)$. Finally, the results are transformed back to the time domain using the Inverse Fast Fourier Transform (IFFT), $x(t) = \mathcal{F}^{-1}(X(f))$. This way, the S4 layer can perform parallel computations in $O(n \log n)$ runtime complexity, as the FFT has a complexity of $O(n \log n)$. This is a significant improvement compared to self attention mechanisms.

To recall, the sequence model of the RSSM used in Dreamer computes the next state as follows:

$$h_t = f(h_{t-1}, s_{t-1}, a_{t-1})$$

where h_t is the deterministic world state, s_t is the stochastic world state, and a_t is the action taken at time step t. During environment interaction, only a single tuple $\langle h_t, s_t, a_t \rangle$ needs to be processed by the sequence model, making models with $O(1)$ runtime complexity for single-step prediction (such as RNNs, GRUs, and S4s) highly efficient. Transformer-based models, on the other hand, often lack efficiency

in direct environment interaction due to their parallel interaction nature and high runtime complexity (Chen et al., 2022; F. Deng et al., 2024; Micheli et al., 2022; Narayanan et al., 2023).

During training, however, a complete trajectory of n tuples $\langle h_i, s_i, a_i \rangle$ with $i \in [1 \cdots n]$ is sampled from the replay buffer. The S4 layer's ability to parallelize operations and leverage highly optimized techniques like the FFT allows it to process these sequences much faster and in a single step, while also considering long-term dependencies in the data. This parallel processing capability significantly enhances training efficiency with the S4 layer.

A. Gu, Goel, et al. (2021) propose various techniques to increase the stability, performance, and training speed of S4 in order to model long sequences. They utilize HIPPO initialization matrices (A. Gu, Dao, et al., 2020) for this, which prevent the learned matrices from diverging over time and thus make the training of the S4 layer stable.

J. T. Smith et al. (2022) propose a simplified version of S4 layers (S5) that is able to achieve similar performance while being more stable and easier to train. Their version utilizes parallel scans and different matrix initialization in order to further boost the parallel sequence prediction and runtime performance.

The parallel scan operation works by expressing the processing of a sequence of $[a_0, a_1, ..., a_n] = a_{0:n}$ to produce an output $[b_0, b_1, ..., b_n] = b_{0:n}$ with an associative operator $\bullet$ (i.e. $a \bullet (b \bullet c) = (a \bullet b) \bullet c$). Parallel scans produce the following output sequence:

$$[a_0, (a_0 \bullet a_1), (a_0 \bullet a_1 \bullet a_2), ..., (a_0 \bullet a_1 \bullet ... \bullet a_n)]$$

Since $\bullet$ is associative, the pairs $a_i \bullet a_j$, $i, j \in [1...n]$ can be computed in parallel. So the runtime complexity of applying this parallel scan is $O(T \log n)$ with assuming that the number of processors $L > \frac{n}{2}$ and T being the cost of matrix-matrix multiplication (J. T. Smith et al., 2022).

In the case of the S5 layer, the input sequence to the parallel scan operation is defined as

$$a_k = (a_{k,i}, a_{k,j}) = (\mathbf{A}, \mathbf{B}u_k)$$

with $u_k \in u_{0:n}$ being the input sequence to the entire layer, $a_k \in a_{0:n}$, and $\mathbf{A}, \mathbf{B}$ being discretized learned matrices. This input sequence can be computed before the parallel scans, which takes $O(n)$ time. The associative operation $\bullet$ is then defined as

$$a_k \bullet a_l = (a_{l,i} \odot a_{k,i}, a_{l,i} \otimes a_{k,j} + a_{l,j})$$

with $\odot$ being a matrix-matrix multiplication, $\otimes$ being a matrix-vector multiplication and $+$ being an elementwise addition. J. T. Smith et al. (2022) show the associative

property of this operation. The parallel scan with this operation yields a sequence $x_{0:n}$ which is the internal state of the model. From this, the output sequence $y_{0:n}$ is computed.

J. T. Smith et al. (2022) use a process called diagonalization, in which a square matrix is reduced to its diagonal representation, where all values of the matrix that are not on the diagonal are 0. These diagonal entries are the eigenvalues of the original matrix. Diagonalized matrices allow for extremely efficient computations of matrix powers, as each element of the diagonal just has to be raised to the power of the exponent. Since diagonalized matrices can be easily converted back to the original matrix, using them instead of the original matrix speeds up the parallel scan operation significantly.

The full architecture of a S5 layer is shown in Figure 2.10.

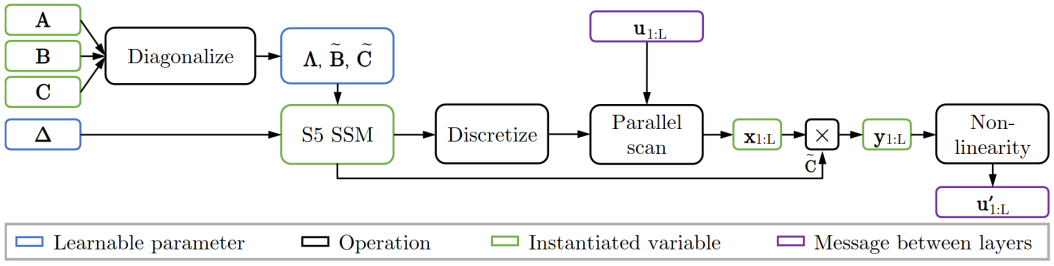

Figure 2.10: Internal structure of the S5 layer as depicted in J. T. Smith et al. (2022). L is the length of the sequences used. $u_{1:L}$ is the input sequence, $u'_{1:L}$ is the output sequence and $x_{1:L}$ is the hidden state. The matrices A, B and C are first diagonalized and then discretized to be used for discrete time intervals. After the parallel scan, the hidden sequence $x_{1:L}$ is multiplied with C and then a non-linear activation is applied to produce the final output sequence $u'_{1:L}$.

Lu et al. (2024) propose a resettable version of S5 layers, which allow resetting the internal state x_t during parallel scans, to apply the S5 layers in a RL setting where the state input sequence could span episode borders. They prove that adding a reset flag c to the $\bullet$ operation does not violate its associative property. So if this flag c_k is set to 1, the internal state x_k becomes simply a_k instead of $(a_0 \bullet a_1 \bullet ... \bullet a_k)$, which effectively resets the hidden state of the S5 layer.

Besides the S5 layer, there have been multiple works proposing improvements on the original S4 layer (A. Gu and Dao, 2023; Hasani et al., 2022; Merrill et al., 2024; Poli et al., 2023; J. Smith et al., 2024). However, they often do not come with further runtime efficiency improvements compared to the S4 layer and also do not offer a resettable solution, which will be crucial for the usage of the state space

model as a component of a world model. This is why this thesis uses S5 layers, as they are more suitable for the use case as a world model backbone and, due to their popularity, there are multiple implementations available, which will be detailed in Section 4.2.

3 Related Work

In this chapter, some related works will be highlighted that pursue similar approaches as Hieros. In Chapter 2 foundational works have been highlighted on which Hieros was build upon. This section contains works that are either similar in their approach or inspired some technical details of the Hieros architecture.

3.1 Hierarchical Reinforcement Learning

As Hierarchical Reinforcement Learning (HRL) has proven to be a powerful solution to many RL problems (Dzhivelikian et al., 2022; Hafner, Lee, et al., 2022; Y. Jiang et al., 2019; Nachum, Ahn, et al., 2019; Nachum, S. S. Gu, et al., 2018), there have been many works exploring different approaches on how to fully capitalize on agents with hierarchic structures.

Nachum, H. Tang, et al. (2019) show that one of the main benefits of HRL is improved exploration, more than inherent hierarchical structures of the problem task itself or larger model sizes. Their study isolates these factors by conducting a series of experiments on tasks such as locomotion, navigation, and manipulation, ultimately attributing most of HRL's empirical advantages to enhanced exploration capabilities. This insight leads them to propose simpler exploration techniques inspired by HRL that achieve competitive performance while using less complex and smaller models.

LeCun (2022) argue that HRL is a promising direction for future research in RL, as complex dependencies often are only discoverable by abstraction, as learned skills are often reusable in different contexts. Central to this is the use of HRL, to leverage multiple levels of abstraction to represent percepts and action plans. They propose an architecture combining a configurable predictive world model, behavior driven by intrinsic motivation, and hierarchical joint embedding architectures trained with self-supervised learning. This approach aims to enable intelligent agents to reason, predict, and plan at multiple time horizons, thus addressing key challenges in AI such as efficient learning, robust reasoning, and effective planning.

As described in Section 2.3, Hafner, Lee, et al. (2022) combine a hierarchical policy with a world model, building on the DreamerV2 architecture (Hafner, Lillicrap, Norouzi, et al., 2020). They show that the combination of a hierarchical policy and

a world model outperforms the original DreamerV2 model on several tasks. The low level policy (also called worker policy) receives only subgoal rewards in this architecture, while the higher level policy (also called manager policy) receives the actual task reward. The proposition of a subgoal which should be fulfilled by the worker policy constitutes an active planning of Director multiple steps into the future.

In a more recent work, Kujanpää et al., 2023 use an autoencoder with a quantized latent space (vector quantized autoencoder, VQAE) to encode key environment states identified from sampled expert replay trajectories. This approach relies on imitation learning, i.e. there already exists a dataset of "experts" (which can be either human or fully trained RL agents) that solve the given problem. Their approach then finds subgoals in these trajectories and trains a VQAE on these subgoals. This generative model for subgoals then proposes subgoals during environment interaction for the current state, which are used by a standard RL algorithm (A* (Hart et al., 1968), PHS (Orseau and Lelis, 2021) or Monte-Carlo Tree Search (Coulom, 2006; Kocsis and Szepesvári, 2006)). The policy searches in the subgoal space for a sequence of subgoals to achieve the given goal. An interesting parallel to Director (Hafner, Lee, et al., 2022) is the use of a discrete subgoal model. The search in subgoal space is also similar to the learned Manager policy used in Director. However, Kujanpää et al., 2023 do not use an explicit notion of a world model for their planning.

The above approaches all only use two levels of hierarchy. There have been some works exploring deeper hierarchies in order to boost the performance on tasks that require long term planning and exploring capabilities (Hutsebaut-Buysse et al., 2022). Haarnoja, Hartikainen, et al., 2018 employ a hierarchy, where the latent space of lower level actors are the action spaces for the higher level actors, which is again is very similar to Director (Hafner, Lee, et al., 2022), where the manager chooses subgoals in a discrete subgoal space that are then decoded into the latent space of the lower level world model. In their framework, each layer is trained after each other, starting from the bottom up. So the lowest level actor is trained first for multiple iterations. Then its layers are frozen, and the next lowest layer is trained with the latent space of the lowest level as its action space. However, they do not use time abstraction and all layers can update their policies at every time step t. They also do not use any kind of environment model for their hierarchical policy.

Levy et al., 2017 also use multiple levels of hierarchy in their Hierarchical Actor Critic (HAC) method. In contrast to Haarnoja, Hartikainen, et al., 2018 their higher level actors operate time abstracted, i.e. they only update every H environment interactions. The higher level actor also proposes subgoals to the next lower level, which is rewarded with a subgoal reward if it reaches the proposed subgoal.

However, they do not use any kind of environment model and are only trained on the recorded environment interactions. They use something called hindsight transitions, where instead of the actions taken by a low level actor to achieve a subgoal g, the system observes the actual in hindsight achieved state g' and trains the higher level policy on the actions taken by the lower level with respect to the actually achieved state g'. This allows the higher level to propose more effective subgoals and guide the lower level policy towards achieving a more complex goal.

Hutsebaut-Buysse et al., 2022 note that multi-level hierarchical policies show promise in solving tasks which need complex long term planning. However, existing approaches often showcase instabilities introduced by complex architectures, non-stationary transition functions and decreased sample efficiency.

3.2 World Model Architectures

Since the initial work of D. Ha and Schmidhuber, 2018 and the release of the first Dreamer model (Hafner, Lillicrap, J. Ba, et al., 2019), there have been many approaches to improve the world model architecture. All works discussed in this subsection will focus on improvements that replace the GRU inside the RSSM (Hafner, Lillicrap, Fischer, et al., 2019), the world model used in Dreamer (Hafner, Lillicrap, J. Ba, et al., 2019), with a different architecture. There are other recent works that implement a different approach to imagination based RL agents, like e.g. IRIS (Micheli et al., 2022), where a Transformer model predicts dynamics on a discrete token latent space of a vector quantized autoencoder (VQAE) and trains an agent in imagination in the decoded observation space. But since Hieros is built upon the Dreamer architecture, this section only includes works that are more closely related to the implemented approach in this thesis.

TransDreamer, a transformer-based MBRL agent introduced by Chen et al. (2022), builds upon the DreamerV2 (Hafner, Lillicrap, Norouzi, et al., 2020) and presents several key distinctions from the traditional Dreamer agent. One significant difference lies in the world model architecture, where TransDreamer replaces the Recurrent Neural Network (RNN) in Dreamer with a Transformer model. This change allows TransDreamer to process state-action pairs $(s_0, a_0), ..., (s_n, a_n)$ more effectively, capturing intricate dependencies between states and actions for improved trajectory predictions. Additionally, TransDreamer adopts a selective approach to generating imagined trajectories by randomly selecting a subset of starting states, unlike Dreamer, which generates trajectories from all states sampled from the replay buffer. They also introduce an important change to the predictive architecture of the world model during training: in DreamerV3's (Hafner, Pasukonis, et al., 2023) RSSM

world model (Hafner, Lillicrap, Fischer, et al., 2019) the sequence model predicts $h_t = f(h_{t-1}, z_{t-1}, a_{t-1})$. So h_t depends on the output of the sequence model h_{t-1} from the previous step. This makes this step impossible to parallelize. Chen et al., 2022 show that this prediction can be replaced by a myopic prediction which only uses z_{t-1}, a_{t-1} to predict the next world state h_t, i.e. $h_t = f(z_{t-1}, a_{t-1})$. In their experiments, this change had no significant impact on the overall agent performance while enabling the parallel prediction during training time. The transformer-based world model is also used during environment interaction, which reduces its runtime efficiency dramatically. TransDreamer demonstrates comparable or superior performance to DreamerV2 in tasks that require long term memory and object permanence.

Robine et al. (2023) propose a similar architecture also build upon DreamerV2 (Hafner, Lillicrap, Norouzi, et al., 2020), but with a key distinction. Unlike TransDreamer (Chen et al., 2022), their Transformer-based World Model (TWM) does not rely on the Transformer model during inference. This design choice significantly enhances the computational efficiency of their model, as the Transformer is only utilized during the training phase. By decoupling the inference process from the Transformer, TWM reduces the computational burden during policy execution, leading to faster decision-making and improved real-time performance. They also provide the obtained reward as world state to its sequence model as well as to the actor, which leads to an improved performance in their experiments. Additionally, they make the observation that in the replay buffers traditionally used in Dreamer models (Hafner, Lee, et al., 2022), early observations are over-sampled compared to newer observations. This is the result of the uniform sampling over the entire replay dataset after each environment interaction. This way, early interactions have more often the opportunity to be selected in a uniform sample than new interactions. Robine et al. (2023) propose to adapt the probability of an interaction to be sampled depending on the times it already has been sampled. Their method takes $O(n)$ time per sample to adjust the probabilities, which reduce the run time efficiency for long training runs. They evaluate their TWM on the Atari100k benchmark (Bellemare et al., 2013) and achieve superior results.

F. Deng et al. (2024) propose S4WM, utilizing S4 layers for the next state prediction. Their model is based on the newer DreamerV3 (Hafner, Pasukonis, et al., 2023) architecture. Since S4 layers can be used both for predicting sequences in parallel and predicting only the next value in an RNN like fashion, their model also proved to be more computationally efficient than the Transformer-based architectures and outperforms them in memorization capabilities. They showcase how S4WM outperform TWM (Robine et al., 2023) and RSSM (Hafner, Lillicrap, Fischer, et al., 2019) in long term memory tasks. Just like DreamerV3, they train their agent

completely on imagined trajectories. However, it is unclear how they handle trajectories that span multiple episodes, as the used S4 layers do not easily allow for a resetting mechanism as it was implemented for S5 layers (Lu et al., 2024; J. T. Smith et al., 2022).

P. Agarwal et al. (2024) use Transformer style networks for both world modelling and actor networks. Their architecture takes a history of past world states and predicts the next action from that sequence of so-called world tokens. On the Atari100K benchmark, their approach achieves competitive results. W. Zhang et al. (2024) implement several changes to the TWM (Robine et al., 2023) architecture, e.g. taking observations, rewards and actions as single tokens for the sequence model prediction or reconstructing the image input without use of the hidden state. Their model STORM achieves state-of-the-art performance on the Atari100k benchmark, which promises that adopting their changes in Hieros might produce even better results. However, in the current thesis these changes are not included, as the improvements implemented in Hieros are primarily focused on the DreamerV3 (Hafner, Pasukonis, et al., 2023) and Director (Hafner, Lee, et al., 2022) architecture.

4 Methodology

This chapter explains the theoretical and practical details of Hieros. To recall, in the context of RL, an agent interacts with an environment at discrete time steps, denoted as t. For the Atari 100K benchmark, for instance, where the environment represents a game like Pong, the agent's interaction involves selecting an action a at time t within the game, similar to making in-game moves or pressing buttons. Subsequently, the agent receives an observation o and a reward r from the environment. In the case of Pong, o typically takes the form of a pixel image capturing the game's visual state, while r represents the points earned as a result of the agent's actions. The agent's primary goal is to learn an optimal policy π, guiding its interactions with the environment, with the overarching objective of maximizing the expected cumulative reward, expressed as $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$, where $\gamma < 1$ represents the discount factor and r_t stands for the reward at time step t.

This chapter shows in depth, how HIERarchical imagination On Structured state space sequence world models (Hieros), a hierarchical model-based RL agent, learns on trajectories generated by a Simplified Structured State Space Sequence (S5) model (J. T. Smith et al., 2022). The approach is mainly based upon DreamerV3 (Hafner, Pasukonis, et al., 2023) and Director (Hafner, Lee, et al., 2022).

First, Section 4.1 discusses the key ideas and innovations that are implemented in Hieros. Then Section 4.2 presents existing implementations that were adapted for the implementation of Hieros and issues encountered during that processs. After this, the three key innovations of Hieros are explained in detail: the hierarchical imagination in Section 4.3, the S5WM in Section 4.4 and the efficient time balanced sampling in Section 4.5. After this, the training and environment interaction procedure is shown step by step in Section 4.6 to fully visualize how the model works.

4.1 Key Ideas of the HIEROS Architecture

Section 1.4 contains an overview over the proposed HIEROS architecture. The key insights from previous work that inspired the proposed solution are explained in this section.

4.1.1 A Modular Hierarchy of Agents

LeCun (2022) outlines a vision of how an agent capable of general intelligence might look like. This paper is more of an opinion piece from a highly reputable researcher who has been on the top of the field for decades than a scientific survey of the current state of autonomous agents. LeCun lists several properties of an agent that will probably be included if this agent is to exhibit some kind of general (or human like) intelligence, as it is depicted in the paper. One key property is a modular implementation, which involves breaking down the agent into smaller, specialized modules that can work together to achieve complex behaviors.

LeCun argues that human like intelligence necessitates a structure where different components or modules handle distinct aspects of cognition and perception. This modular approach allows for specialized processing within each module, enhancing the overall efficiency and adaptability of the system. For instance, modules dedicated to perception, motor control, memory, and decision-making can operate semi-independently yet cooperatively, much like the human brain. A world model would be used to simulate future consequences of actions and be used for planning multiple steps ahead in time. Figure 4.1 shows a visualization of the modular architecture LeCun describes. Remarkably, this closely resembles the Dreamer (Hafner, Lillicrap, J. Ba, et al., 2019) architecture.

The paper also argues that such a modular system should not only consist of distinct components but also be capable of hierarchical organization, where higher-level modules can oversee and coordinate the activities of lower-level ones. This hierarchical structure allows for complex behaviors to emerge from simpler, well-defined tasks managed at different levels, enabling the system to handle a wide range of activities with greater efficiency and adaptability. In this hierarchy, lower-level modules manage basic, reflexive tasks while higher-level modules handle more complex, abstract reasoning. This hierarchical structure not only mirrors the organization seen in biological systems but also allows for more scalable and flexible intelligence.

LeCun (2022) inspired the general research direction in this thesis of RL agents that incorporate a world model for environment simulation as well as a hierarchical structure for long term and abstract planning.

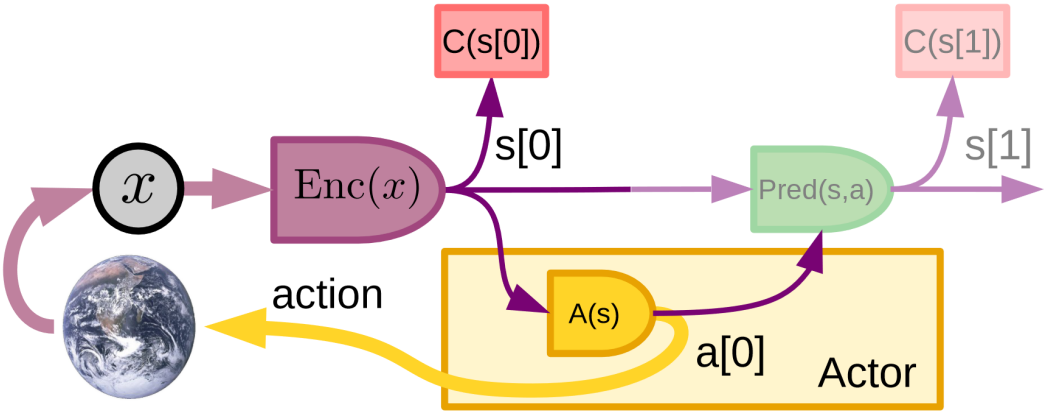

Figure 4.1: A modular architecture for a general autonomous agent as depicted in LeCun (2022). The encoder module *Enc* transforms the observation into a compact latent space. The actor takes the latent world representation and outputs an action. The prediction module (also called world model in the paper) models the dynamics of the environment, and the cost module C determines the value of the current state (also called reward or cost in the paper).

Khetarpal et al. (2022) mention in their survey of RL agents capable of continual learning that skill and goal learning is one essential strategy to tackle the observed problem of agents becoming unflexible in their policy learning over time. After learning a policy that solves a simple task very well, even slight variations of the problem will result in severe degradation of the agents performance (E. Bengio et al., 2020). Skill learning (often realized with a hierarchy of policies) allows transference and reusage of the learned skills across different tasks. In the context of HRL, skill learning can be understood as the communication of subgoals to a lower level policy, which are proposed by a higher level policy. These subgoals are then learned skills for the lower level policy that it can reuse in different contexts for different tasks. An agent capable of continual learning will be able to tackle environments with shifting distributions and dynamics. This once more underlines the importance of HRL as a research topic.

As mentioned in Section 2.3, deeper hierarchies show promise to enhance the ability of an agent to solve complex tasks which require long term planning and contain shifting distributions even better with an increase in hierarchy layers (Hutsebaut-Buysse et al., 2022). However, the key limitation of those algorithms with multiple layers of hierarchy (i.e. more than 2 layers) typically exhibit a lack of stability and efficient communication between layers.

The key idea for Hieros is, that providing each layer of the hierarchy with its own

world model will ultimately allow for each layer to simulate the causes of its own actions in its own time abstracted latent space. Each layer learns how to interact with the next lower level much more sample efficient this way, since the world model can easily generate more training data of layer to layer interactions. The world model will simulate, how the lower level policy executes the proposed subgoals, which will let the higher level policy learn much faster how to communicate with its lower level policy via subgoals to maximize the extrinsic rewards. This hierarchical imagination is a novel procedure that after careful literature review has not been tried yet by other authors.

As Hafner, Lee, et al. (2022) demonstrated, how the Dreamer architecture can be expanded to a hierarchical RL setting, the natural choice for the ground algorithm for Hieros to use in its multi layered architecture is a Dreamer like model, more specifically DreamerV3 (Hafner, Pasukonis, et al., 2023), the newest variant of the architecture.

4.1.2 A More Efficient World Model Architecture

The hierarchical agent with world models for each layer imposes a hefty increase on computational complexity. Effectively, this means that there are multiple subactors that each have to learn a policy and a world model on their own replay datasets. To decrease the additional in resource usage of the agent, a more efficient world model architecture is needed.

As explained in Section 2.4 structured state space models (SSSM) (A. Gu, Goel, et al., 2021) can be used both in parallel mode and in a recurrent iterative mode. The S4WM, described in Section 2.2, poses multiple improvements compared to the RSSM used in the Dreamer models, regarding long term contextualization and memorization as well as runtime improvements. The parallel training of S4WM and the iterative environment interaction would make it a feasible world model backbone for the hierarchical world model architecture.

However, the S5 layer poses multiple improvements on the original S4 layer, most notably the ability to pass a reset signal into the parallel scan, which allows the S5 layer to model sequences spanning multiple episodes. Also, the S5 layer significantly improves runtime efficiency and modeling precision. So implementing a world model architecture with an S5 layer backbone is a natural choice to have a world model that is very runtime efficient while also having superior prediction power compared to other architectures.

Another feature is the availability of an internal state of the S5 layer. In the S4 layer, the internal state is implicitly encoded and would only be available after deconvoluting intermediate results (A. Gu, Goel, et al., 2021) and is not naturally

supported in the original S4 architecture. The S5 layer can directly access its internal state. As shown in Section 5.3.5 using this internal state as part of the world state improves the overall agent performance.

4.1.3 A Fast Sampling Procedure

Storing environment interactions in a replay buffer and drawing trajectories from this buffer to perform policy updates has long been used in RL (Mnih et al., 2013; Mondal and Jamali, 2020). The sampling strategy to select the records to train on has a significant influence of the performance of the RL agent (Fedus et al., 2020; Mondal and Jamali, 2020). In general, sampling from a replay buffer instead directly learning from the environment interactions has the advantage that the data points are more independently and identically distributed (i.i.d.) which is a key prerequisite in deep learning and has been a significant hurdle when training DRL agents (Mnih et al., 2013).

However, Robine et al. (2023) noted, that the traditional way of sampling (uniformly over the collected replay dataset after a certain amount of environment interaction (Hafner, Lillicrap, J. Ba, et al., 2019; Micheli et al., 2022; Mnih et al., 2013)) oversamples the early interactions with the environment. They propose a balancing technique that cancels out this oversampling, but has a $O(n)$ runtime with n being the length of the replay buffer. They show that this balancing technique, which enables a true uniform sampling over the replay buffer, has a positive impact on the performance of the RL agent. For this balancing, the number of times an entry has been sampled is being counted and the probabilities for each entry are then recomputed based on this counter. To use this enhanced replay technique in Hieros, a more efficient implementation is necessary.

The key idea here is that the oversampling of early interactions by uniformly sampling over the replay buffer after each environment can be described with harmonic series. There exists an approximation of $H_x \approx ln(x)$ with H_x being the x-th harmonic number and $ln(x)$ being the natural logarithm of x. A harmonic number is defined as

$$H_x = \sum_{i=1}^{x} \frac{1}{i}$$

Using this approximation of harmonic numbers, the oversampling distribution can be described very precisely. This enables the computation of the cumulative density function (CDF) of this distribution. This function can be used in a process called

probability integral transformation to turn the skewed sample into a true uniform sample over the buffer with respect to the current replay buffer size.

This transformation of the precisely defined probability function can be done in $O(1)$ and produces the same results as the empirical reweighting of probabilities as performed by Robine et al. (2023).

4.2 Existing Implementations, Frameworks and Issues

The basis for the implementation of Hieros was the PyTorch implementation of the DreamerV3 model (NM512, 2023). There exists an original implementation of DreamerV3 (Hafner, Pasukonis, et al., 2023) by the authors in danijar/dreamerv3 (2024). This implementation, however, uses the JAX deep learning framework (*google/jax* 2024), which was developed by Google. The PyTorch framework (Ansel et al., 2024) on the other hand has been under development for much longer, and thus it is way more popular and offers large community support. For this reason, Hieros is implemented in PyTorch.

The PyTorch implementation of DreamerV3 (NM512, 2023), however, has some shortcomings compared to the original implementation in JAX. For example, initially, the implementation did not support parallel execution of multiple environments. So if the agent should be trained on multiple instances of the same environment to boost runtime efficiency and use all the systems resources, the PyTorch implementation would just execute the environments sequentially. Also, the implementation did not support all environments the JAX implementation supported, and the training code was less optimized and took longer.

The JAX implementation of DreamerV3 (danijar/dreamerv3, 2024) contains a library called "embodied" which implements all environments, replay strategies, parallel executions and training procedures for the JAX code. This library is not available as a stand-alone version. So for Hieros, the "embodied" code was extracted and modified to work with a PyTorch based RL agent. This increased runtime efficiency of the training compared to the pure PyTorch version (NM512, 2023), especially when using parallel environments.

There also already exist several implementations of the S5 layer (J. T. Smith et al., 2022). The implementation of the original authors also uses JAX (lindermanlab/S5, 2022), but due to its popularity, another version using PyTorch exists as well (C2D/s5-pytorch, 2023). This implementation was used in Hieros. The reset mechanism of the modified S5 layer from Lu et al. (2024) had to be implemented from scratch to

be used in Hieros, since there is no implementation publicly available. Thorough tests showed the functional correctness of the reset mechanism.

The S5 layer has complex parameters, due to the diagonalization of the weight matrices. This fact posed some issues that had to be solved. For example, there was a bug in the weight decay function of the Adam optimizer in PyTorch which caused it to crash when encountering complex parameters. This bug was fixed in version 2.1.0 of PyTorch but before that, a custom workaround had to be implemented for Hieros. Also, mixed precision training with complex parameters is currently not supported in PyTorch. Mixed precision means that for some operations and parameters, the number of bits representing weights or input values can be reduced from the standard 32 bits to 16 bits or sometimes even 8 bits. This quantization often does not have any influence on the final performance of the model but speeds up the training and inference procedures significantly. Specifically, the gradient scaler used in precision conversion of model gradients lacks a GPU side CUDA code implementation. For Hieros a CPU side workaround was implemented that does the gradient scaling for complex numbers in pure Python code. This enables the use of mixed precision training for Hieros. Since this is not a real fix for the issue, no Pull Request to merge this fix into the PyTorch library has been created.

The full implementation of Hieros is publicly available under https://github.com/Snagnar/Hieros.

4.3 Hierarchical Imagination based RL Agent

Hieros employs a goal conditioned hierarchical policy. Each abstraction layer learns its own S5 world model, an actor-critic and a subgoal autoencoder. Some background on hierarchical reinforcement learning is given in Section 2.3. The overall design of the subgoal proposal and the intrinsic reward computation is similar to the Director architecture (Hafner, Lee, et al., 2022), which successfully implements a hierarchical policy on the basis of DreamerV1. In contrast to Director and many other works, Hieros can easily be scaled to multiple hierarchy levels. Most approaches apply only one lower level and one higher level policy (Hafner, Lee, et al., 2022; Y. Jiang et al., 2019; Nachum, S. S. Gu, et al., 2018; Nair and Finn, 2019).

Figure 4.2 illustrates the hierarchical structure and interaction among subactors. Each subactor i consists of three modules: the world model (W^i_θ), an actor-critic component (Π^i_ϕ), and a subgoal autoencoder (G^i_ψ). θ, ϕ and ψ are the learnable parameters of the specific models. The world model imagines trajectories for the actor-critic training. The subgoal autoencoder compresses and decompresses states

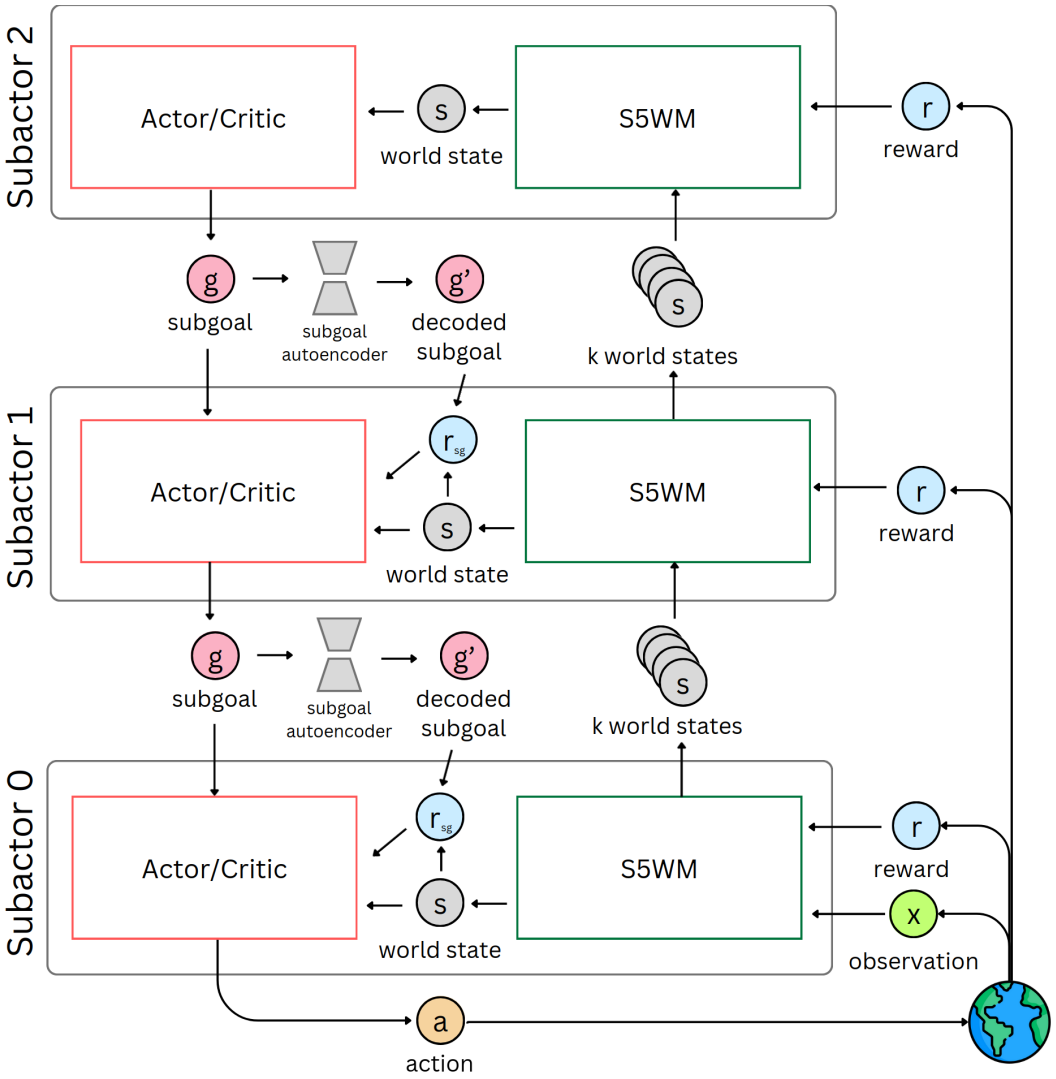

Figure 4.2: Hierarchical subactor structure of HIEROS. Each layer of the hierarchy learns its own latent state world model and interacts with the other layers via subgoal proposal. The action outputs of each actor/critic is the subgoal input of the next lower layer. The output of the lowest level actor/critic is the actual action in the real environment.

within the world model, with the decoder handling subgoal decoding, computing subgoal rewards r_g, and generating novelty rewards r_{nov}. Only the lowest layer (subactor 0) interacts directly with the environment. The higher layers receive k consecutive world model states from the lower level as observations, train their world model on these states and generate subgoals g^i. This is also in contrast to

Director, which only provides the higher level policy with every k-th world state. The effect of providing all intermediate states as input for the higher level policy is shown in a separate ablation in Section 5.3.6.

The subgoals represent world states that the lower layer is tasked to achieve. This follows the general scheme of a goal conditioned hierarchical RL agent. The subgoals proposed by the higher level subactor are kept constant for k steps of the lower layer. So higher level subactors can only update their proposed subgoal every k steps of the next lower level, while the lower level subactor has k steps to maximize the similarity of the current world state to the one represented by the subgoal.

Each subactor i has its own world model and learns its actor in its own latent space, which again is in contrast to Director, where the Manager and Worker policies both operate in the same latent space. The lower level subactor i generates k latent states by interacting with the environment. These k latent space are then concatenated into one feature vector, which is passed to the world model of the upper level subactor $i+1$. As explained in Section 2.2, world models in a Dreamer like architecture contain an encoder (also called representation model), which takes in observations and produces world states in the latent space of the world model. The encoder of the world model produces a latent representation of the k world states received from the lower level subactor i, which is then passed to the actor-critic module Π^{i+1} (together with the subgoal g^{i+1} this subactor received from its next upper level subactor) to produce a subgoal g^i. The subgoal autoencoder G^i decodes these subgoals g_i from a low dimensional subgoal latent space to the latent space of W^i. The lower level actor-critic Π^i then produces actions based on this subgoal and the latent representation of environment inputs.

4.3.1 The Subgoal Autoencoder

The subgoal autoencoder G^i_ψ is composed of an encoder and decoder:

$$\text{Encoder: } g_t \sim p_\psi(g_t|h_t)\ , \ \text{Decoder: } h_t \approx f\psi(g_t) \tag{4.1}$$

The deterministic part h_t of the model state is the only input for the subgoal autoencoder. This choice was made because the stochastic part z_t is less controllable by the lower-level actor, making the subgoal less achievable.

Hafner, Lee, et al. (2022) found that the full latent state space of the lower level world model as action space would constitute a high dimensional continuous control problem for the higher level actor, which is hard for a policy to optimize on. Instead, G^i_ψ compresses the latent state into a discrete subgoal space of 8 categorical vectors

of size 8. This compressed action space is much easier to navigate for the subgoal proposing policy than the continuous world state.

While Hafner, Lee, et al. (2022) uses the decoded goal as input for their worker policy, HIEROS demonstrates improved subgoal completion and overall slightly higher rewards when training subactors on the compressed subgoals. The different effects of these two approaches are directly compared in Section 5.3.8. Even though the overall achieved performance of the agent does not differ significantly between using the decoded or encoded subgoals as input for the actor, using encoded subgoals allows for smaller networks with less trainable parameters. The subgoal autoencoder is trained on batches of world model states using a variational loss:

$$\mathcal{L}_\psi = || f_\psi(\hat{g}_t) - h_t ||_2 + \beta \cdot KL\big[\, p_\psi(\, g_t \,|\, h_t) \,||\, q(g_t) \,\big] \tag{4.2}$$

With $||\cdot||_2$ being the L2 norm ($||x||_2 = \frac{1}{n}\sum_{i=0}^{n} x_i^2$). $\hat{g}_t$ is sampled from the encoder distribution $\hat{g}_t \sim p_\psi(g_t|h_t)$ and $q(g_t)$ is a uniform prior. β is a hyperparameter controlling the impact of KL divergence on the overall loss. The KL loss is used to guide the autoencoder towards using the complete latent space and not only a small part of it, by modifying the loss to be smaller if the generated subgoal g_t is close to the uniform distribution (Diederik P Kingma, Welling, et al., 2019).

Regarding the practical implementation of this loss computation: In deep learning, it is not possible to directly compute gradients from probability distributions, as the sampling process is a non-differentiable operation. To circumvent this issue, the reparametrization trick is employed (Diederik P Kingma, Welling, et al., 2019), which re-expresses a random variable $\mathbf{z}$ in terms of a deterministic variable and an auxiliary random variable. Specifically, instead of sampling $\mathbf{z} \sim \mathcal{N}(\mu, \sigma^2)$ directly (in case the prior sampled from is a normal distribution $\mathcal{N}$), the training procedure samples $\epsilon \sim \mathcal{N}(0, I)$ and then computes $\mathbf{z}$ using the transformation $\mathbf{z} = \mu + \sigma \cdot \epsilon$. This reformulation separates the source of randomness (ϵ) from the parameters of the distribution (μ and σ), making the entire process differentiable and allowing gradients to be computed with respect to μ and σ during backpropagation. This reparametrization trick is very commonly used when learning a distribution rather than concrete values. All parts of HIEROS that learn a distribution of values use this trick in order to learn from a distributional loss.

Figure 4.3 shows a visualization of how the subgoal autoencoder training loss is computed.

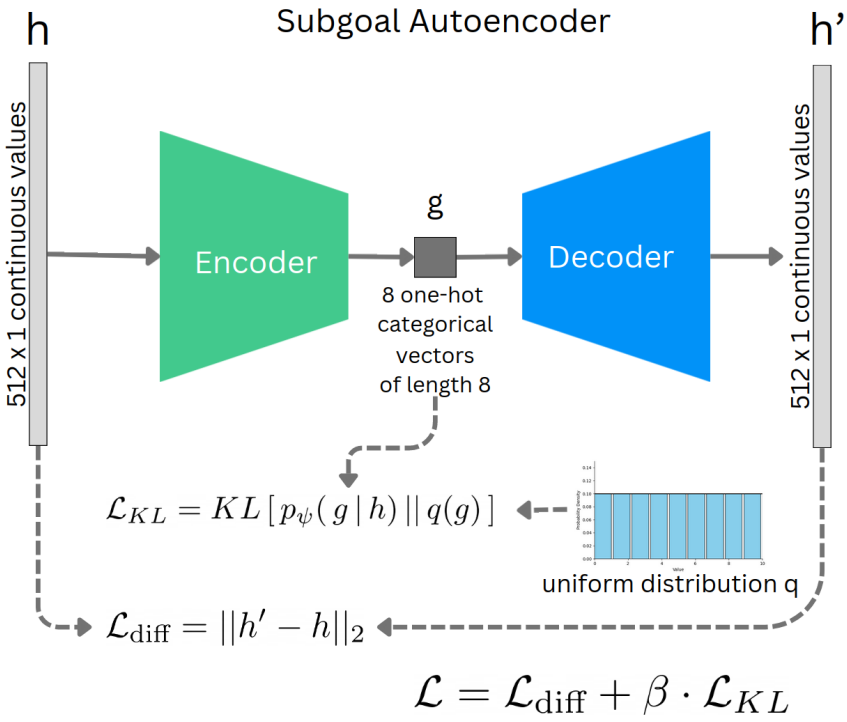

Figure 4.3: The calculation of the training loss $\mathcal{L}$ for the subgoal autoencoder. The autoencoder receives a deterministic world state h as input, compresses it down to 8 one hot categorical vectors of length 8 and trains to reconstruct h' precisely. The KL loss guides it to use the entire latent space, as narrow representations (where only a part of the latent space is used) have a larger distance to the uniform distribution than a representation that uses the entire latent space.

4.3.2 The Actor-Critic component

The actor-critic component Π is trained on imagined rollouts of the word model and consists of a policy network π and several value networks V. The reward r that is maximized by this module is a mixture of extrinsic rewards r^{extr} (i.e. observed rewards directly from the environment), subgoal rewards r^{g}, and novelty rewards r^{nov}:

$$r = r^{extr} + \alpha_g \cdot r^{g} + \alpha_{nov} \cdot r^{nov} \tag{4.3}$$

Here, α_g and α_{nov} are hyperparameters that control the influence of the subgoal and novelty rewards on the overall reward. In contrast to Director, in Hieros all subactor layers receive an extrinsic reward. In their ablation study Hafner, Lee, et al. (2022) show that providing the worker policy also with extrinsic rewards boost

the overall performance of the agent. They do not use this option in Director, as conditioning the lower level policy solely on the subgoal rewards enables learning a task independent policy that in principle can be used for a wide range of different task without retraining (i.e. in a meta learning setting). Since the scope of this thesis does not include meta learning, HIEROS directly conditions all layers also on the extrinsic rewards. The effect of supplying all subactors with the extrinsic rewards is shown in Section 5.3.10.

For higher level subactors i, the extrinsic reward r^{extr} is the summed observed reward of the k^i environment steps that are taken between the subgoal updates of this subactor.

The subgoal reward r_t^g at time step t is defined as

$$r_t^g = \frac{G_{dec}^i(g_t)_t^T \cdot h_t}{\max(||\, G_{dec}^i(g_t)\, ||, ||\, h_t\, ||)} \tag{4.4}$$

With $G_{dec}^i(g_t)$ the decompressed subgoal at time step t The subgoal reward is computed using the cosine max similarity method between the world model state and the decompressed subgoal, which was proposed by Hafner, Lee, et al. (2022). For Director, the authors tested out a wide range of different distance functions, e.g. Euler, Manhattan or cosine distance. From all functions they tested, max cosine similarity as defined above yielded the best result, which is why it is also used in HIEROS.

The novelty reward r_t^{nov} at time step t is computed as follows:

$$r_t^{nov} = ||\, h_t - G_{dec}^i(G_{enc}^i(h_t))\, ||_2 \tag{4.5}$$

with h_t being the deterministic world model state, G_{dec}^i the decoder part of the subgoal autoencoder of subactor i, G_{enc}^i the encoding part of the subgoal autoencoder and $||\cdot||_2$ being the L2 norm. Since the subgoal autoencoder is trained to compress and decompress the world model state, it is able to model the distribution of the already observed model states. World states that have been visited many times and thus provide only very little novelty to the policy, will be provided many times to the subgoal autoencoder as train input, which results in a low reconstruction error on these states. If a world state is very different from past visited state, which would indicate a high novelty of this state, the autoencoder will fail to reconstruct this state very precisely which results in a higher reconstruction error. This connection allows using the reconstruction error of the subgoal autoencoder on the current world model state as a novelty reward r_{nov}. The effect of adding a novelty reward r_{nov} to the overall reward is shown in Section 5.3.12.

The actor learns the policy:

$$a_t \sim \pi_\phi^i(\,(h_t,\, z_t)\,, g_t,\, r_t,\, c_t,\, \mathcal{H}(\,p_\theta(\hat{z}_t \mid m_t)\,)\,) \tag{4.6}$$

with h_t and z_t representing the current model state, g_t being the current subgoal, r_t being the reward, c_t being the continue signal, and $\mathcal{H}(p_\theta(\hat{z}_t|m_t))$ being the entropy of the latent state distribution (this will be explained in depth in Section 4.4). In the case of a higher level subactor, a_t is the subgoal g_t for the next lower layer. For the highest level subactor, there is no subgoal, so this part is just dropped. The effect of supplying the actor-critic with these additional inputs is shown in Section 5.3.11.

Note that the actor learns a distribution of possible actions. During training in imagination, the action is sampled from the distribution, while during environment interaction the mode of the distribution is used. The learning of a distribution of actions ensures that during training the actor sees a diverse set of environment interactions and their respective imagined outcomes, and thus is more robust against noise.

These additional inputs to the actor network are motivated by findings of Robine et al. (2023) which show, that providing the predicted reward as input improves the learned policy. They only pass the reward signal r_t to their policy to inform the actor about states with high reward. By also passing the continue signal c_t the agent can avoid states that might terminate the current episode (or seek these state out, depending on the task). The entropy term is used to encourage exploration and make the actor aware of states with high uncertainty. If the entropy of the latent state distribution is high, this means that the dynamics prediction model cannot reliably predict the next state from the current moment on, which might indicate either that the state has an inherently high stochasticity or that the dynamics model is not sufficiently trained. Depending on the context, the actor might then decide to either avoid or explore states with high entropy. The effect of these additional inputs for the actor-critic module is shown in Section 5.3.11.

For clarity, s_t is defined as $s_t = ((h_t, z_t), g_t, r_t, c_t, \mathcal{H}(p_\theta(\hat{z}_t|m_t)))$ the input for the policy and value networks used in the actor-critic in Hieros.

The actor is trained using the REINFORCE algorithm (Williams, 1992). The actor-critic trains three separate value networks as critics for each reward term to predict their future mean return: V_{extr}, V_{nov}, and V_{sg}. Hieros uses a soft actor-critic method, so for each value network there is also a slow target network that is only updated very few steps to reduce variance of the learned value network outputs.

The value networks train minimizing the squared error between the predicted value and the observed return. For each value network, the target value V^{target} is

computed as:

$$V_t^{\text{target}} = r_t + \gamma\, V_{\phi'}(s_{t+1}) \tag{4.7}$$

where γ is the discount factor, r_t is the observed reward, and $V_{\phi'}(s_{t+1})$ is the predicted value of the next state using the current value network V_ϕ. The loss for each value network is then:

$$\mathcal{L}(\phi) = \frac{1}{2}\left(V_\phi(s_t) - V_t^{\text{target}}\right)^2 \tag{4.8}$$

Each value network (extrinsic, subgoal and novelty) minimizes this loss.

The actor network is trained to maximize the expected return using the REINFORCE algorithm:

$$\Delta\phi = \alpha \nabla_\phi \log \pi_\phi(a_t \mid s_t)\left(R_t - \sum_i V_{\phi_i}^{slow}(s_t)\right) - \eta\mathcal{H}(\pi_\phi(a_t \mid s_t)), \tag{4.9}$$

where $\Delta\phi$ is the parameter change of the policy π_ϕ, α is the learning rate, R_t is the cumulative return from time step t, and $\sum_i V_{\phi_i}(s_t)$ is the combined output of the three value networks (extrinsic, subgoal, and novelty). $\mathcal{H}(\pi_\phi(a_t|s_t))$ is the entropy of the policy and η is a hyperparameter setting the weight of this entropy term. Figure 4.4 shows how the actor and the value networks are trained from a trajectory imagined by the world model.

It should be noted that the original actor-critic used by DreamerV3 (Hafner, Pasukonis, et al., 2023) incorporates several techniques to stabilize learning in various challenging settings, such as tasks with very sparse or dense rewards or tasks where the absolute value of rewards is very large or very small. For example, instead of outputting a single value, the value networks produce a categorical two-hot distribution, and they use a Symlog function to scale the rewards. HIEROS adopts the same actor-critic module and training scheme from DreamerV3 (except for introducing multiple value networks for the different reward terms), thus a fully detailed description of how the actor-critic network functions falls outside the scope of this thesis.

Subgoals from all hierarchy layers can be decoded into the game's original image space, enabling visualization of the agent's objectives and making HIEROS actions explainable. The subgoal proposed by the uppermost subactor g_t^n is decoded by the subgoal autoencoder of the next lower subactor into the latent state of its world model. This decoded world state is then decoded by the world model decoder back into the subactor's observation space, i.e. the world model space of the next lower

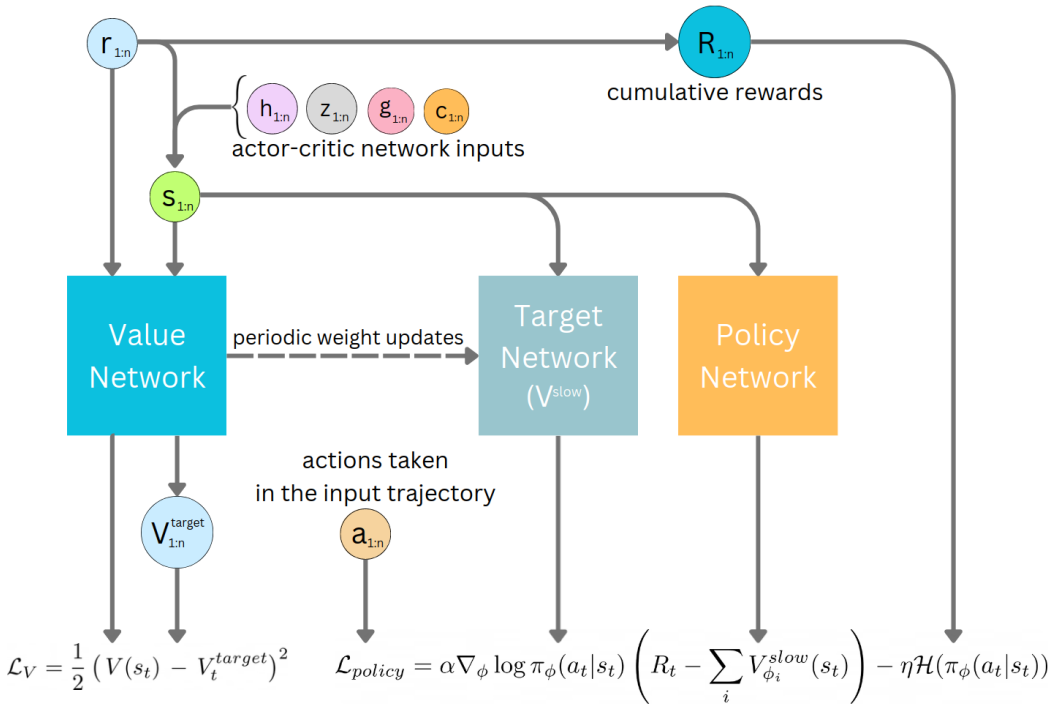

Figure 4.4: Loss computation for the value and policy networks. $s_{1:n}$ contains the input for the value and policy networks: the world state $(h_{1:n}, z_{1:n})$, the current subgoal $g_{1:n}$, the reward $r_{1:n}$ and continue signals $c_{1:n}$. The entropy of the stochastic world state distribution is omitted for clarity. Also, only one value network is shown here. HIEROS uses three value networks for each part of the reward: r_{extr}, r_{nov} and r_{sg}. The losses of the value and policy networks are minimized via gradient descent.

subactor. As the decoded subgoal only contains h_t, the stochastic part of the world state for each subactor is stored in a buffer to be reused for decoding. This way, the initial subgoal can be iteratively decoded into the original observation space of the original environment. Examples are presented in Figure 5.5 and Figure 5.9.

4.4 Enhancing the World Model Architecture

In the previous section, the hierarchical structure of Hieros was explained in detail. However, the inner workings of the world model were not discussed. The world model in Hieros is responsible for imagining a trajectory of future world states and encoding environment observation in an expressive latent state. For this task, Hieros uses a similar architecture as the RSSM used in DreamerV3 (Hafner, Pasukonis, et al., 2023) while swapping out the RNN-based sequence model for an S5-based sequence model. Some background on world models and S4/S5 layers is given in Sections 2.2 and 2.4.

The world model predicts world states in a latent space. These world states consist of the tuple (h_t, z_t) with h_t being the deterministic and z_t being the stochastic part of the world state. z_t is a vector of n one hot categorical vectors of length n. This discrete world state was first used in DreamerV2 (Hafner, Lillicrap, Norouzi, et al., 2020) and proved very successful in modeling stochasticity in complex RL tasks. The world model used in Hieros consists of the following networks:

$$
\begin{aligned}
\text{Sequence model:} \quad & (m_t, h_t) = f_\theta(h_{t-1}, z_{t-1}, a_{t-1}) \\
\text{Encoder:} \quad & z_t \sim q_\theta(z_t \mid o_t) \\
\text{Dynamics predictor:} \quad & \hat{z}_t \sim p_\theta(\hat{z}_t \mid m_t) \\
\text{Reward predictor:} \quad & \hat{r}_t \sim p_\theta(\hat{r}_t \mid h_t, z_t) \\
\text{Continue predictor:} \quad & \hat{c}_t \sim p_\theta(\hat{c}_t \mid h_t, z_t) \\
\text{Decoder:} \quad & \hat{o}_t \sim p_\theta(\hat{o}_t \mid h_t, z_t)
\end{aligned}
$$

With o_t being the observation at time step t, m_t the output of the sequence model, and h_t the internal state of the S5 layers in the sequence model, which is used as the deterministic part of the world state. z_t is the stochastic part of the latent world state, a_t the action taken, $\hat{z}_t$ the predicted latent state, $\hat{r}_t$ the predicted extrinsic reward, $\hat{c}_t$ the predicted continue signal (1 if the current step t is the end of the episode, 0 otherwise) and $\hat{o}_t$ the decoded observation. θ are the learnable parameters of the world model. Using h_t alongside m_t in predicting dynamics would be possible but doesn't add extra information, as m_t is computed from the explicit internal state h_t from the S5 sequence model. The posterior z_t represents the stochastic state with knowledge of the actual observation o_t, while the prior $\hat{z}_t$ is the stochastic state predicted solely from the deterministic sequence model output. During imagination, the actor only has access to the prior $\hat{z}$.

In the actual implementation of Hieros, the sequence model is also supplied with the continue signal c_t during parallel training to reset its internal state when the

episode ends. During iterative environment interaction, the state is reset directly when the episode ends. Since this is only a practical detail, for the rest of the thesis, the sequence model will be denoted as $f_\theta(h_{t-1}, z_{t-1}, a_{t-1})$, without the c_{t-1} input.

Figure 4.5 shows a visualization of how the different networks inside the world model work together.

4.4.1 Training of the World Model

During training of the world model, S5WM processes sequential observations $o_0 \cdots o_n = o_{0:n}$ and actions $a_0 \cdots a_n = a_{0:n}$ from trajectories sampled from the replay buffer. The encoder takes the observations and computes the sequence $z_{0:n}$. The sequence model then predicts output $m_{1:n+1}$ and deterministic state $h_{1:n+1}$. The initial h_0 is just an empty vector filled with 0s. The world states are then used by the decoder and dynamics, reward and continue predictors to generate their output.

All parts of S5WM are optimized jointly using a loss function that follows the loss function of DreamerV3, which was explained in detail in Section 2.2.2. At time step t, the world model loss $\mathcal{L}(\theta)$ is defined as:

$$\mathcal{L}(\theta) = \mathcal{L}_{pred}(\theta) + \alpha_{dyn} \cdot \mathcal{L}_{dyn}(\theta) + \alpha_{rep} \cdot \mathcal{L}_{rep}(\theta) \tag{4.10}$$

$$\mathcal{L}_{pred}(\theta) = -\ln(p_\theta(r_t|h_t, z_t)) - \ln(p_\theta(c_t|h_t, z_t)) - \ln(p_\theta(o_t|h_t, z_t)) \tag{4.11}$$

$$\mathcal{L}_{dyn}(\theta) = \max(1, \mathrm{KL}[\,\mathrm{sg}(q_\theta(z_t \mid o_t)) \,||\, p_\theta(\hat{z}_t \mid m_t)\,]) \tag{4.12}$$

$$\mathcal{L}_{rep}(\theta) = \max(1, \mathrm{KL}[\,q_\theta(z_t \mid o_t) \,||\, \mathrm{sg}(p_\theta(\hat{z}_t \mid m_t))\,]) \tag{4.13}$$

For $\mathcal{L}_{dyn}$ and $\mathcal{L}_{rep}$ the method of free bits introduced by Durk P Kingma et al. (2016) and used in DreamerV3 (Hafner, Pasukonis, et al., 2023) is used to prevent the dynamics and representations from collapsing into easily predictable distributions. How exactly this method works was already explained in Section 2.2. α_{dyn} and α_{rep} are hyperparameters that control the influence of the dynamics and representation loss. For the decoder, reward predictor and continue predictor, the negative log likelihood loss is used.

Figure 4.5 illustrates how the different parts of the world model loss are calculated.

4.4.2 The S5 Sequence Model

The RSSM encoder in DreamerV3 computes the posterior from both o_t and the deterministic model state h_t (i.e. $z_t \sim q_\theta(z_t \mid h_t, o_t)$), which makes the parallel computation of z_t impossible. This is due to the fact that in order for the sequence

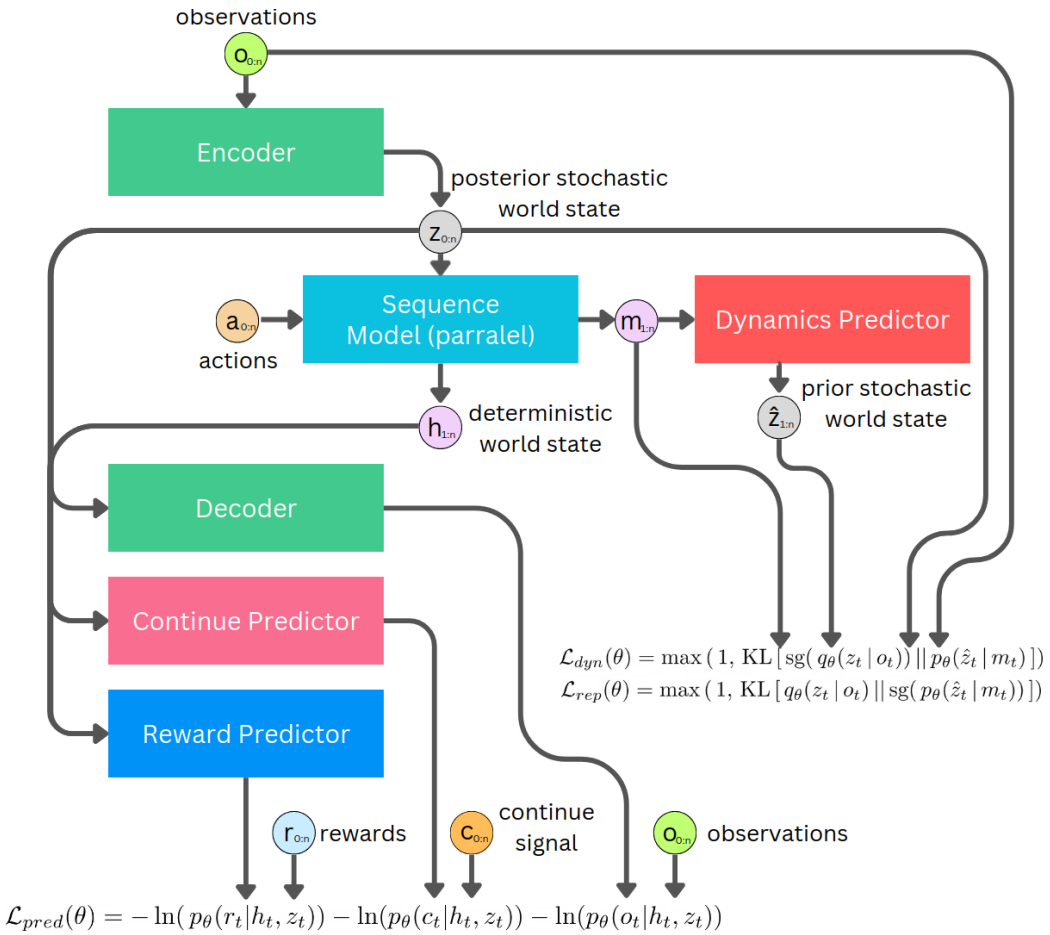

Figure 4.5: The loss calculation during training of the world model. It takes in observations $o_{0:n}$, actions $a_{0:n}$, rewards $r_{0:n}$ and continue signals $c_{0:n}$ and outputs the three losses $\mathcal{L}_{dyn}$, $\mathcal{L}_{rep}$, and $\mathcal{L}_{pred}$ which are then combined as per Equation (4.10).

model to generate $h_{1:n+1}$, it takes in $z_{0:n}$ as inputs. So the sequence $z_{0:n}$ needs to be precomputed independently of h_t to be used in the parallel generation of $h_{1:n+1}$. Chen et al. (2022) show that accurate representations can also be learned by making the posterior z_t independent of h_t and only predict the distribution $p(z_t|o_t)$ without a noticeable loss of precision. So for Hieros the encoder uses this representation model to speed up the training procedure by using parallel prediction.

As detailed in Section 2.4, S5 and S4 layers allow for both parallel sequence modeling and iterative autoregressive next state prediction. This makes them more efficient than RNNs for model training and more efficient than Transformer-based models for imagination. While F. Deng et al. (2024) propose S4WM, an

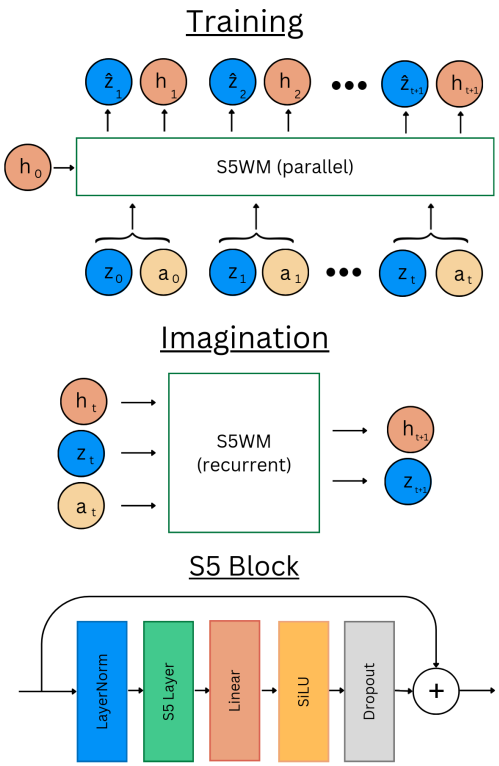

Figure 4.6: The sequence model of the S5WM during training and during imagination. HIEROS uses a stack of S5 blocks with their architecture shown above.

S4-based world model, the use of S5 layers is advantageous for HIEROS. To recall, S5 layers excel in modeling longer-term dependencies and are more memory-efficient. Moreover, the direct accessibility of the latent state x_t in S5 layers enables HIEROS to utilize the recurrent model state as the deterministic component of the world state h_t. This contrasts with other approaches that rely on using the sequence model output as deterministic world model states (Chen et al., 2022; F. Deng et al., 2024; Micheli et al., 2022), which proves less effective in the ablation experiments (Section 5.3.5).

As explained in Section 4.1, HIEROS employs a modified S5 layer proposed by Lu et al. (2024), featuring a reset mechanism that sets the internal state h_t to its initial value h_0 during parallel sequence prediction. This mechanism, which also involves passing the continue signal $c_{0:n}$ to the sequence model, prevents information leakage between episodes, enabling the actor to train on multi-episode trajectories. HIEROS adopts this mechanism for its S5WM, addressing the common occurrence of trajectories spanning episode borders during training. It remains

unclear if the S4WM by F. Deng et al. (2024) encountered and resolved this challenge similarly.

The sequence model of the S5WM uses a stack of multiple S5 blocks, as depicted in Figure 4.6. The design of this block is inspired by the deep sequence model architecture proposed by J. T. Smith et al. (2022) but for Hieros a different norm layer (Layer Normalization (J. L. Ba et al., 2016) in Hieros vs Batch Normalization (Ioffe and Szegedy, 2015) in their model), dropout and activation function (SiLU (Elfwing et al., 2018) vs GeLU (Hendrycks and Gimpel, 2016)) was used. These changes proved empirically beneficiary in a hyperparameter optimization run, comparing the agent's performance with and without these changes. The number of blocks used is listed in Section 4.6.3.

During training, the S5 sequence model in its parallel form predicts $(m_{1:n+1}, h_{1:n+1})$ from the input sequence $(z_{0:n}, a_{0:n})$ and an initial h_0 in a single forward pass through the model. During imagination and environment interaction, the S5 sequence model uses the iterative form, where it predicts (m_{i+1}, h_{i+1}) from a tuple (h_i, z_i, a_i) in one forward pass through the model.

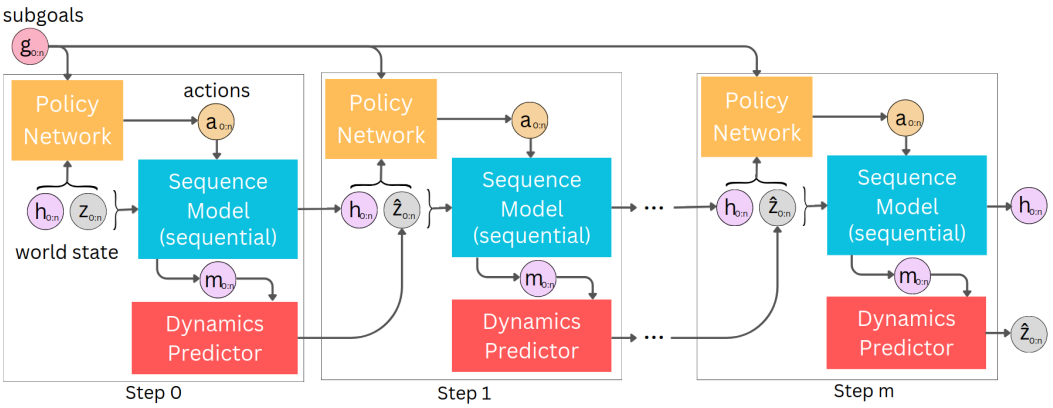

Figure 4.7: The imagination process in Hieros. The sequence model, dynamics predictor and policy network iterate to produce n trajectories $h_{1:n}^{1:m+1}$, $z_{1:n}^{1:m+1}$, $a_{1:n}^{1:m+1}$ of length m. The subgoal $g_{1:n}$ is kept constant for all imagination steps. For clarity, the additional inputs $r_{1:n}^{1:m+1}$, $c_{1:n}^{1:m+1}$, and $\mathcal{H}(p_\theta(\hat{z}_t|m_t))$ for the policy network are omitted.

4.4.3 Imagination on the World Model

As detailed in Section 2.2, the purpose of using a world model for imagining trajectories is to generate more data for the actor-critic to learn a policy on. The training of the world model described earlier ensures that the model learns to predict the next world state precisely and realistically.

Figure 4.7 shows a visualization of the imagination procedure in HIEROS. Similarly to DreamerV3, HIEROS imagines next world state entirely inside its own latent space. The imagination starts with a list of world states tuples $(h_{0:n}, z_{0:n})^0$. The actor produces actions for all states. This is done in parallel, as the input sequence is passed as a batch of data to the actor network, which uses batch parallelism commonly utilized in deep learning.

The sequence model in its iterative form uses $((h_{0:n}, z_{0:n})^0, a_{0:n})$ to produce the next state sequence $(m_{0:n}, h_{0:n})^1$, which is used by the dynamics predictor module to generate $(h_{0:n}, z_{0:n})^1$. Note that the sequence model does not use its parallel form here, as it should only predict one single next state from each state action pair in the sequence. The parallel form during training is used to predict multiple future steps in *one* pass through the model. Batch parallel processing, however, is used here as well.

The actor and world model then produce a total of n trajectories of length m from the initial states $(h_{0:n}, z_{0:n})^0$ to the final $(h_{0:n}, z_{0:n})^m$. m is the horizon depth hyperparameter and has to be selected beforehand. This way, the world model can be used to increase the training data for the actor-critic module m-fold by simulating environment interaction in imagination.

4.5 An Efficient Time-Balanced Replay Sampling Strategy

When interacting with the environment, HIEROS collects observations, actions, and rewards in an experience dataset. After a fixed number of interactions, trajectories are sampled from the dataset in order to train the world model, actor, and subgoal autoencoder. DreamerV3 (Hafner, Pasukonis, et al., 2023) uses a uniform sampling. This, however, leads to an oversampling of the older entries of the dataset, as the iterative uniform sampling can select these entries more often than newer ones. As explained in Section 4.1, Robine et al. (2023) solve this using a time-balanced replay buffer with a $O(n)$ sampling runtime complexity with n being the size of the replay buffer.

HIEROS uses an Efficient Time-Balanced Sampling method (ETBS), which produces similar results with $O(1)$ time complexity. When iteratively adding elements to the experience replay dataset and afterward sampling uniformly, the expected number of times N_{x_i} an element x_i has been drawn after n iterations is

$$\mathbb{E}(N_{x_i}) = \frac{1}{i} + \frac{1}{i+1} + \cdots + \frac{1}{n} = H_n - H_i \tag{4.14}$$

with H_i being the i-th harmonic number. Figure 4.8 visualizes how this expected number of draws of an element x_i is derived.

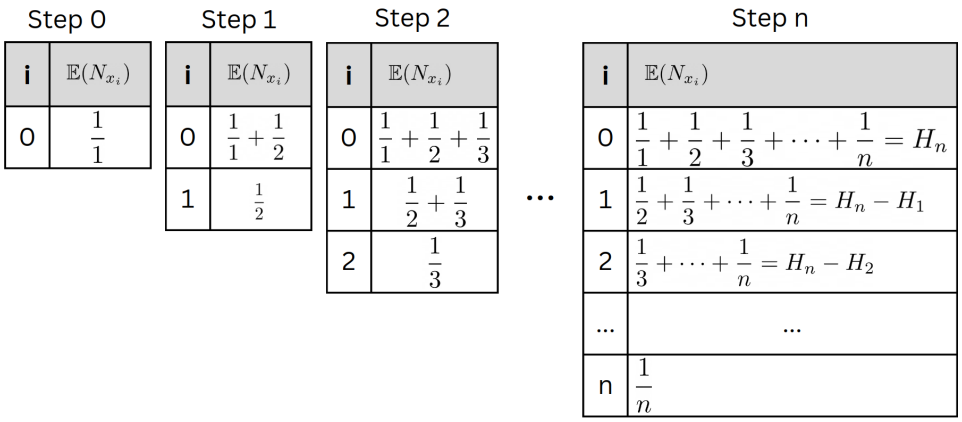

Figure 4.8: The expected number of times an element x_i is drawn when iteratively adding one element and sampling uniformly over the entire dataset n times.

The probability of sampling x_i, $i = 1, ..., n$, is

$$p_i = \frac{1}{n} \cdot \mathbb{E}(N_{x_i}) = \frac{H_n - H_i}{n} \approx \frac{\ln(n) - \ln(i)}{n} \tag{4.15}$$

H_x can be approximated using $H_x \approx \ln(x)$ which removes the need to compute harmonic values. The idea is, to compute the cumulative distribution function (CDF) of this probability distribution between 0 and n to transform samples from the imbalanced distribution into true uniform samples via probability integral transformation F. David and Johnson, 1948.

The skewed distribution, which arises when adding elements to a buffer and sampling over the entire buffer after each addition, is defined as follows:

$$p(x) = \frac{H_n - H_x}{n} \approx \frac{\ln(n) - \ln(x)}{n} = \tilde{p}(x) \tag{4.16}$$

with n being the size of the dataset. Since the inequality $\ln(x) < H_x < 1 + \ln(x)$ holds for all $x \geq 2$, assume without loss of generality that $2 \leq x \leq n$. However, because $\tilde{p}(x)$ does not sum to 1 over the interval $[2, n]$, it must be divided by its integral:

$$\begin{aligned} p_s(x) &= \frac{\tilde{p}(x)}{\int_2^n \tilde{p}(x)dx} = \frac{\frac{\ln(n)-\ln(x)}{n}}{\frac{-2\ln(n)+n+2\ln(2)-2}{n}} \\ &= \frac{\ln(n) - \ln(x)}{-2\ln(n) + n + 2\ln(2) - 2} \end{aligned} \tag{4.17}$$

with p_s being the approximate probability density function of the skewed sampling distribution. The CDF of this distribution is defined as follows:

$$CDF_s(x) = \int_2^x p_s(t)\, dt = P_s(x) - P_s(2) \tag{4.18}$$

with $P_s(x)$ being the antiderivative of $p_s(x)$:

$$P_s(x) = \int_{-\infty}^x p_s(t)\, dt = \frac{x \cdot (\ln(x) - \ln(n) - 1)}{2\ln(n) - n - 2\ln(2) + 2} \tag{4.19}$$

With this, the $CDF_s(x)$ can be derived in closed form:

$$CDF_s(x) = \frac{x \cdot (\ln(x) - \ln(n) - 1) + 2(\ln(n) - \ln(2) + 1)}{2\ln(n) - n - 2\ln(2) + 2} \tag{4.20}$$

This can be computed in $O(1)$ time and is therefore very efficient. With the $CDF_s(x)$, the efficient time-balanced sampling distribution $p_{etbs}(x)$ can be calculated as follows:

$$p_{etbs}(x) = CDF(\, p(x)\,) \cdot \tau + p(x) \cdot (1 - \tau) \tag{4.21}$$

with p being the original sampling distribution, CDF being the CDF of the original distribution and τ being a temperature hyperparameter that controls the influence of the original distribution on the overall distribution. In the experiments, τ is set to 0.3. Empirically, a slight oversampling of earlier experiences seems to have a positive influence on the actor performance. The time complexity of this sampling method is $O(1)$, as the CDF can be precomputed and the sampling is done in constant time.

It is also important to mention that the calculations above assume, that there is only one item added to the dataset and then sampled one time after each step. However, if k items are added before sampling s times, the expected number of draws for one element becomes $\mathbb{E}(N_{x_i}) = \frac{k \cdot (H_n - H_i)}{s \cdot n}$. This additional factor $\frac{k}{s}$ is canceled out when computing the probabilities for one element from the expected value, which is why it can be ignored in the computations.

Figure 4.9 shows the sampling counts for different temperatures τ for ETBS.

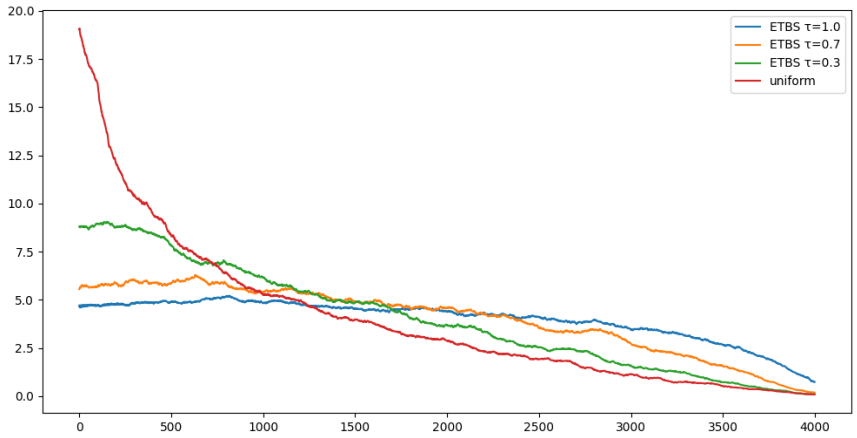

Figure 4.9: The number of times an entry i is sampled in a dataset of final size 4000 with iteratively adding one element to the dataset and sampling over the dataset afterward. Compared are the original uniform sampling method implemented in DreamerV3 (Hafner, Pasukonis, et al., 2023) and the proposed ETBS with different temperatures τ. During the experiments, a temperature of $\tau = 0.3$ is used.

4.6 Environment Interaction and Training Procedure

This section ties the processes mentioned above together and tries to give a complete overview over how specifically HIEROS interacts with the environment and how it trains on collected trajectories.

4.6.1 How HIEROS Interacts with its Environment

The interaction with the environment follows the general scheme of reinforcement learning agents: each time step t HIEROS receives an observation o_t and a reward r_t^{extr} from the environment and outputs an action a_t.

Figure 4.10 illustrates how HIEROS processes inputs, focusing on the simultaneous subgoal updates of all subactors, which typically occurs every k^i steps for subactor i. Initially, the lowest level subactor encodes the observation into z_t^0. The sequence model then updates the last world state (h_{t-1}^0, z_{t-1}^0) using the previous action a_{t-1}^0 to generate h_t^0. This new world state (h_t^0, z_t^0), along with the last k world states, is passed to the upper-level subactor. Each subactor encodes its incoming observation similarly.

After all subactors have encoded their observations and updated their current world states (h_t^i, z_t^i), the highest level subactor generates additional inputs for its actor (r_t^i, c_t^i, and $\mathcal{H}(\hat{z}_t^i | m_t^i)$) and updates its proposed subgoal g_t^{i-1}. Lower levels then evaluate their policies, incorporating the subgoal from the next upper level into their actor inputs. The lowest level actor's action is then passed back to the environment.

After the subactors have updated their outputs, each subactor stores its observation, the action, the observed reward, the continue signal (1 if the current step is the end of the episode, 0 otherwise) and its current subgoal in its own replay buffer.

As each subactor i is only updated every k^i steps, the overall overhead of employing a hierarchy of n subactors is in $O(\log n)$. However, HIEROS uses smaller networks than the original DreamerV3 architecture, so the overall execution speed of environment interactions of HIEROS is comparable to DreamerV3 (see Section 5.1.1).

Algorithm 1 shows the environment interaction as a high level algorithm to give a detailed description of the execution order of the different modules.

4.6.2 How each Subactor Trains

Each subactor maintains its own replay buffer. For a training iteration, a batch of training trajectories is sampled from this buffer using the efficient time balanced sampling strategy.

Figure 4.11 visualizes the training procedure. First, the world model is trained on the sampled trajectories $(o_0, a_0, g_0, r_0, c_0) \cdots (o_n, a_n, g_n, r_n) = (o_{0:n}, a_{0:n}, g_{0:n}, r_{0:n}, c_{0:n})$, o being an observation, a the action, r the observed reward and c the continue signal. For this, the observations $o_{0:n}$ are encoded by the representation model into $z_0 \cdots z_n$, from which the sequence model and dynamics predictor generate $z_{1:n+1}, h_{1:n+1}$. The decoder, reward and continue predictor then produce their outputs and the combined loss is computed and the model parameter updated. For convenience and because there is no subgoal g_{n+1} available, the state z_{n+1}, h_{n+1} is dropped and only $z_{1:n}, h_{1:n}$ is being used.

Next, the world model is used to imagine trajectories starting from the initial sequences $z^0_{1:n}, h^0_{1:n}, g_{1:n}$. The policy produces actions $a^0_{1:n}$, from which the world model predicts $h^1_{1:n}, z^1_{1:n}$. This is repeated m times, m being the imagination horizon hyperparameter. Note that the subgoals $g_{1:n}$ are being kept constant for the entire imagination, so the actor has enough time to explore how to fulfill the proposed subgoal. Also, keeping the subgoal constant cuts down on total runtime, as for the new subgoal proposal, the higher level subactors need to be invoked during the imagination procedure. Experiments in Section 5.3.9 show, that keeping the subgoal constant as opposed to updating it during imagination has a negligible impact on the overall performance of Hieros on the benchmark tasks. The world model then predicts the rewards $r^{0:m}_{1:n}$ for all imagined world states $(h_{1:n}, z_{1:n})^{0:m}$ and the actor-critic uses the imagined data to update its policy and value networks as described in Section 4.3.2.

Finally, the subgoal autoencoder G is trained on the entire batch of sampled and imagined world states $(h_{1:n}, z_{1:n})^{0:m}$ via the variational autoencoder loss.

The training procedure is also shown in Algorithm 2.

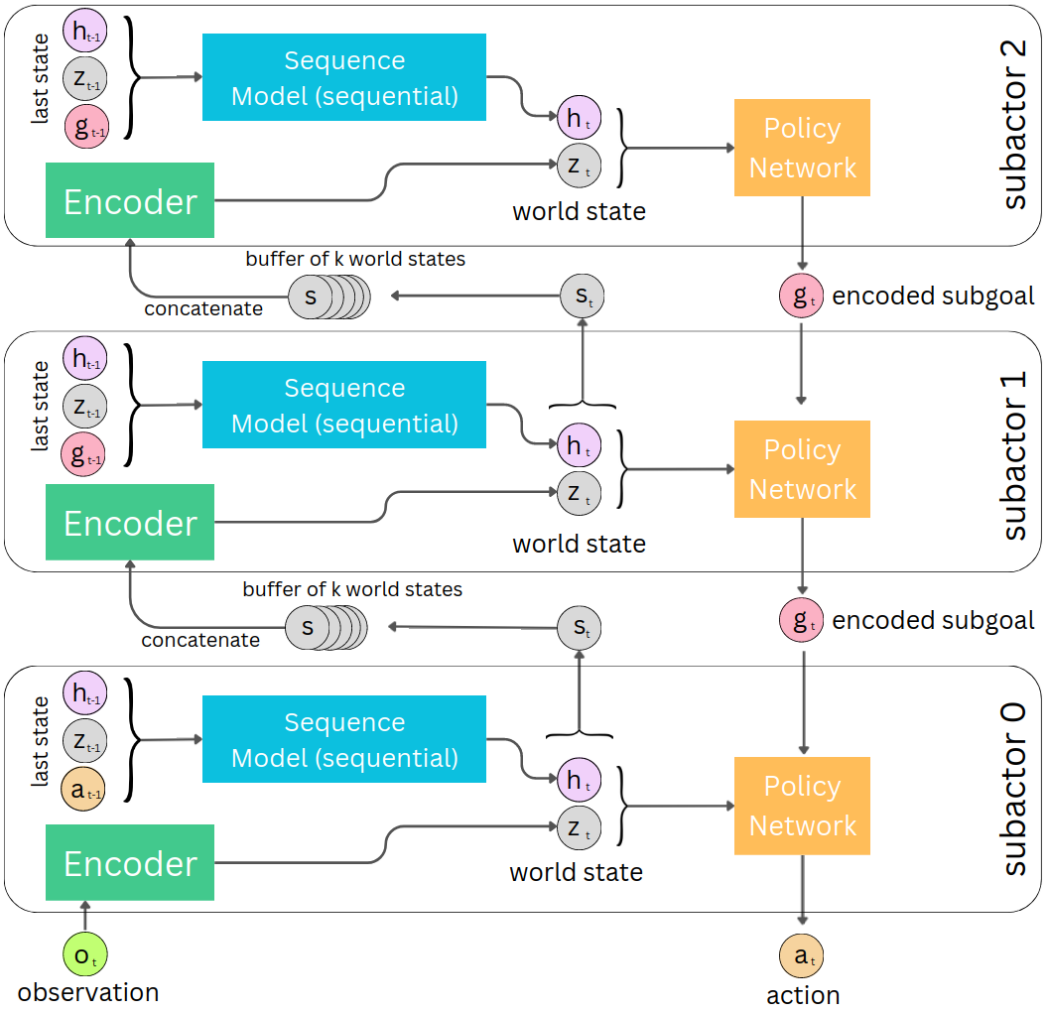

Figure 4.10: Interaction of HIEROS with an environment. For clarity, the additional inputs for the policy networks (r_t, c_t and $\mathcal{H}(p_\theta(\hat{z}_t|m_t))$ are omitted. The observed reward is also not shown in the graphic. The agent takes in an observation o_t and outputs an action a_t. After the interaction, each subactor stores $(o_t, a_t, r_t, c_t, g_t)$ in its replay buffer.

Algorithm 1: Environment interaction of HIEROS. Note that additional inputs for the actor networks are omitted for clarity. More readable names were used for the subactor modules, e.g. encoder(o_t) instead of $p_\theta(z_t|o_t)$

Input: o_t, r_t, k `// observation and reward at time step` t, `subactor update interval` k
Output: a_t `// action`

Function `Encode(`*subactor*, o_t`)`**:**
  $z_t \longleftarrow$ subactor.encoder(o_t)
  $h_{t-1}, z_{t-1} \longleftarrow$ subactor.last_world_state
  $a_{t-1} \longleftarrow$ subactor.last_action
  $m_t, h_t \longleftarrow$ subactor.sequence_model($h_{t-1}, z_{t-1}, a_{t-1}$)
  subactor.last_world_state $\longleftarrow (h_t, z_t)$
  return h_t, z_t
  End Function
Function `Interact(`o_t, r_t, t, k`)`**:**
  `//` s_t^i `contains the world state tuple` (h_t^i, z_t^i)
  $s_t^0 \longleftarrow$ *Encode*(subactor$_0$, o_t, a_t)
  foreach *subactor*$_i$ $\in$ *subactor*$_1 \cdots$ *subactor*$_n$ **do**
    subactor$_i$.add_observation_to_buffer(s_t^{i-1})
    if $t \mod k^i == 0$ **then**
      $s_t^i \longleftarrow$ *encode*(subactor$_i$, subactor$_i$.observation_buffer)
    else
      break
    end
  end
  `// additional actor inputs are omitted for clarity`
  $g_t^n \longleftarrow$ subactor$_n$.actor(s_t^n) **if** $t \mod k^n == 0$ **else** subactor$_n$.*last_action*
  subactor$_n$.store_interaction_in_replay_buffer()
  foreach *subactor*$_i$ $\in$ *subactor*$_{n-1} \cdots$ *subactor*$_0$ **do**
    $g_t^i \longleftarrow$ subactor$_i$.actor(s_t^i, g_t^{i+1}) **if** $t \mod k^i == 0$ **else** subactor$_i$.last_action
    subactor$_i$.last_action $\longleftarrow g_t^i$
    subactor$_i$.store_interaction_in_replay_buffer()
  end
  return g_t^0 as a_t
  End Function

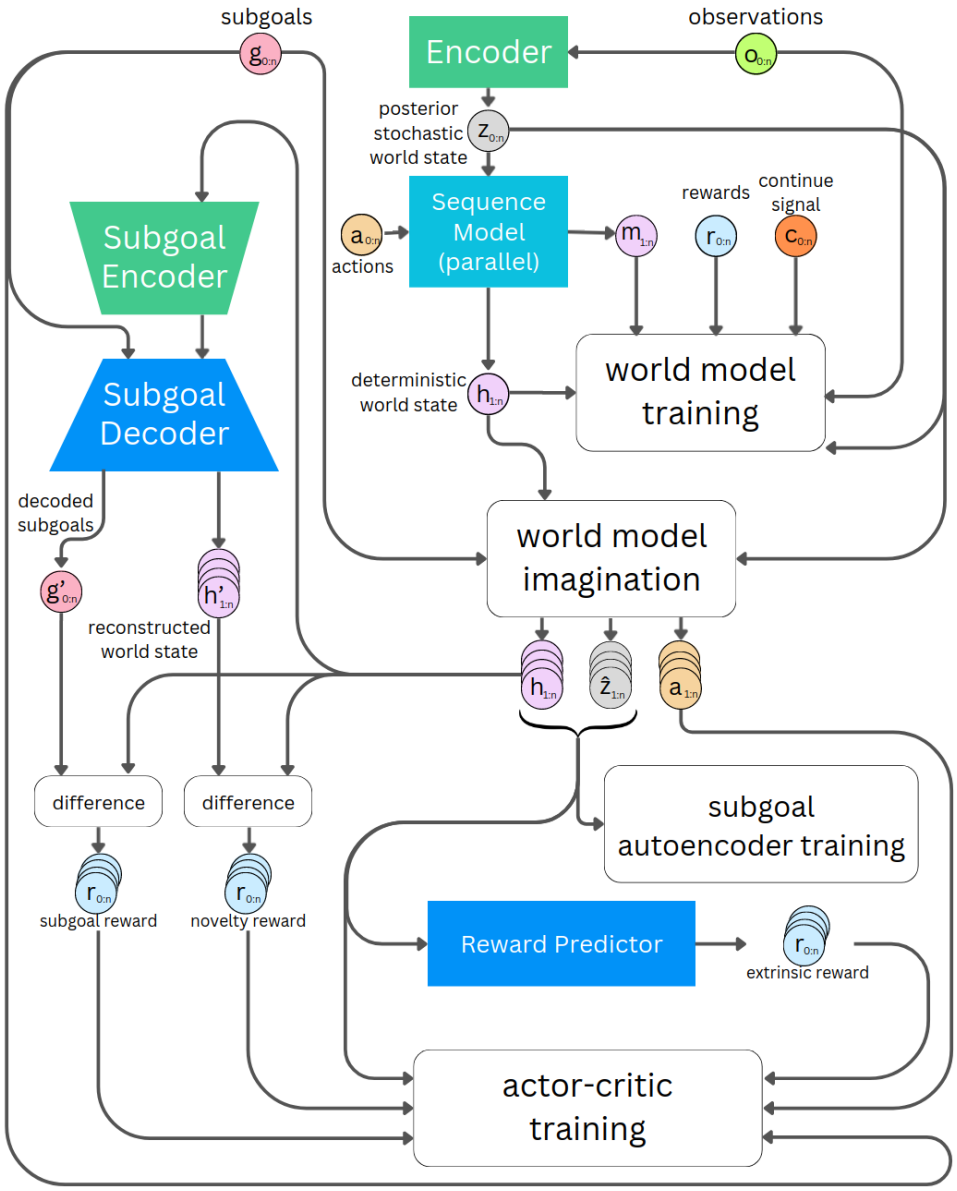

Figure 4.11: The training of a single subactor on sampled trajectories $(o_{0:n}, a_{0:n}, g_{0:n}, r_{0:n}, c_{0:n})$. The world model training is shown in detail in Figure 4.5, the world model imagination in Figure 4.7, the subgoal autoencoder training in Figure 4.3, and the actor-critic training is shown in detail in Figure 4.4. For clarity, the continue signal input $c_{1:n}$ and the entropy term $\mathcal{H}(p_\theta(\hat{z}_t|m_t))$ inputs for the actor-critic learning are omitted.

Algorithm 2: Training of a single subactor of HIEROS. For better readability, the names of modules have been used instead of their symbols (e.g. subgoal_autoencoder instead of G).

Input: $(o_{0:n}, a_{0:n}, g_{0:n}, r_{0:n}, c_{0:n}), h$ `// sampled trajectory from replay buffer and imagination horizon`

Function `Train`$(o_{0:n}, a_{0:n}, g_{0:n}, r_{0:n}, c_{0:n})$**:**

$z_{0:n} \longleftarrow$ encoder$(o_{0:n})$

$m_{1:n}, h_{1:n} \longleftarrow$ sequence_model$(z_{0:n}, a_{0:n})$ `// the sequence model produces` $m_{1:n+1}, h_{1:n+1}$ `but for convenience the last state is dropped.`

`// compute the loss as per Equation (4.10)`

$\mathcal{L}_\theta \longleftarrow$ compute_world_model_loss$(h_{1:n}, z_{1:n}, a_{1:n}, r_{1:n}, c_{1:n}, o_{1:n})$

update_params(world_model, $\mathcal{L}_\theta$)

$h^{0:m}_{1:n}, \hat{z}^{0:m}_{1:n}, a^{0:m}_{1:n} \longleftarrow$ imagine$(h_{1:n}, z_{1:n}, g_{1:n})$

$(r_{extr})^{0:m}_{1:n} \longleftarrow$ reward_predictor$(h^{0:m}_{1:n}, \hat{z}^{0:m}_{1:n})$

`//` $\hat{h}$ `is the reconstruction of the subgoal autoencoder of the original deterministic world state` h

$\hat{h}^{0:m}_{1:n} \longleftarrow$ subgoal_autoencoder$(h^{0:m}_{1:n})$

$(r_{nov})^{0:m}_{1:n} \longleftarrow || h^{0:m}_{1:n} - \hat{h}^{0:m}_{1:n} ||_2$

$\hat{g}_{1:n} \longleftarrow$ subgoal_autoencoder.decode$(g_{1:n})$

`//` $\max$ `cosine distance between` $\hat{g}_{1:n}$ `and` $h^{0:m}_{1:n}$

$(r_{sg})^{0:m}_{1:n} \longleftarrow \frac{(\hat{g}_{1:n})^T h^{0:m}_{1:n}}{\max(||\hat{g}_{1:n}||, ||h^{0:m}_{1:n}||)}$

`// training as described in Section 4.3.2`

train_actor_critic$(h^{0:m}_{1:n}, a^{0:m}_{1:n}, g_{1:n}, (r_{extr})^{0:m}_{1:n}, (r_{nov})^{0:m}_{1:n}, (r_{sg})^{0:m}_{1:n})$

`// compute autoencoder loss as per Equation (4.2)`

$\mathcal{L}_G \longleftarrow$ compute_autoencoder_loss$(\hat{h}^{0:m}_{1:n}, h^{0:m}_{1:n})$

update_params(subgoal_autoencoder, $\mathcal{L}_G$)

End Function

4.6.3 Hyperparameters

All hyperparameters of HIEROS are listed in Section 4.6.3. Following DreamerV3 (Hafner, Pasukonis, et al., 2023), HIEROS' hyperparameters remain fixed unless otherwise specified (e.g., in ablation experiments).

Training parameters:	
learning rate	10^{-4}
weight decay	0
optimizer	AdamW
learning rate scheduler	None
warmup episodes	1000
ETBS temperature τ	0.3
batch size	16
trajectory length for training	64
imagination horizon	16

Hierarchy parameters:	
hierarchy layers	3
subgoal proposal intervals k	4
extrinsic reward weight	1
subgoal reward weight	0.3
novelty reward weight	0.1
subgoal shape	8x8

World Model parameters:	
S5 model dimension	256
S5 state dimension	128
Number of HIPPO-N initialization blocks J	4
Number of S5 blocks	8
dynamic loss weight a_{dyn}	0.5
representation loss weight a_{rep}	0.1
h_t dimension	256
z_t dimension	32x32
MLP units	256
G_ψ KL loss weight β	0.5

Total parameters	37.1 M

5 Evaluation

The performance of Hieros is evaluated on two benchmarks: the Atari100k test suite (Bellemare et al., 2013), which contains a wide range of games with different dynamics and objectives, and on the DeepMind Control (DMC) suite (Tassa et al., 2018), which contains a variety of robot control tasks of different degrees of complexity.

In each of the Atari environments, the agent is limited to 100k interactions with the environment, equivalent to approximately 2 hours of total gameplay. Hieros is evaluated on a subset of 25 of these games. Within these environments, the agent performs discrete actions (i.e., button presses), resulting in a finite number of possible actions. This number varies across games, as the agent's action space is constrained to the button presses utilized in the game. The maximum action space size is 18, allowing the actor to perform up to 18 different button presses within an Atari game.

The DMC suite consists of 20 different tasks, in which a robot has to be controlled to maximize the reward. In each of these tasks, the agent controls the joints of the robot using continuous actions, setting the position of those joints. Stability of the output is oftentimes important for the robot to move smoothly and get the maximum reward. The agent has a budget of 1 million steps on each of those tasks.

Both those benchmarks are very popular when evaluating the capabilities of RL agents (Hafner, Pasukonis, et al., 2023; Hui et al., 2024; Micheli et al., 2022; Robine et al., 2023). In both benchmarks, the agent receives pixel inputs from frames of the game or from a camera pointing towards the robot as input. There also exists a variation of the DMC suite, where the agent receives direct representations of the robot state in the environment (how the joints are aligned or the position of the robot) as vector inputs (Tassa et al., 2018). But as the visual version of the benchmark poses a more challenging task (the agent first has to understand how to interpret the input image in order to move the robot correctly), Hieros is only evaluated on the visual DMC suite.

All experiments were conducted on a machine with an NVIDIA A100 graphics card with 40 GB of VRAM, 8 CPU cores and 32 GB RAM. Training Hieros on one Atari game for 100k steps took roughly 14 hours in this setup. A training run on one task of the DMC suite took approximately 2 days to complete.

The used hyperparameters as specified in Section 4.6.3 and are kept constant for

all experiments. Hieros follows the example of DreamerV3 (Hafner, Pasukonis, et al., 2023), where the hyperparameters for all suites and tasks are the same, which allows for an easier application of the agent to different environments. The code for Hieros is publicly accessible at https://github.com/Snagnar/Hieros, so the experiments shown in this chapter are fully reproducible.

5.1 Atari100k Benchmark

Hieros is compared to the following baselines, on the Atari100k benchmark: DreamerV3 (Hafner, Pasukonis, et al., 2023), Director (Hafner, Lee, et al., 2022), IRIS (Micheli et al., 2022), TWM (Robine et al., 2023), and SimPLe (Kaiser et al., 2019). SimPLe trains a policy on the direct pixel input of the environment using PPO (Schulman, Wolski, et al., 2017), while TWM, IRIS, Director and DreamerV3 train a policy on imagined trajectories of a world model. TWM and IRIS use a Transformer-based world model. IRIS, however, trains the agent not in the latent space of the world model but in the decoded state space of the environment, which is one of the main differences to other approaches that leverage a Transformer-based world model (e.g. TWM). TWM, IRIS, Director and DreamerV3 were already portrayed in detail in Chapter 2.

Director (Hafner, Lillicrap, J. Ba, et al., 2019) does not report results on the Atari100k benchmark in their paper, so the reported scores are from self conducted experiments. Also, in their original version, Director does not condition the lower level policy on the extrinsic environment rewards. In their ablation, they show this approach often leads to worse performance compared to rewarding the lower level policy also with the extrinsic rewards. Since all subactors in Hieros receive the extrinsic reward, the experiments on Director were executed with the lower level policy also conditioned on the extrinsic reward in order to provide a better comparability between Director and Hieros.

Hafner, Pasukonis, et al. (2023) recently published an updated version of DreamerV3 and updated results on the Atari100k benchmark. However, they use a larger model with 200 million parameters to achieve these results. In order to provide better comparability, Hieros is compared with the results of the original 18 million parameter version of DreamerV3.

Ye et al. (2021) propose EfficientZero, which holds the current absolute state of the art in the Atari100k test suite. However, this method relies on look-ahead search during policy inference and is thus not comparable to the other methods.

Comparing the S5WM of Hieros against the S4WM proposed by F. Deng et al. (2024) would be interesting, as both models are based on structured state spaces.

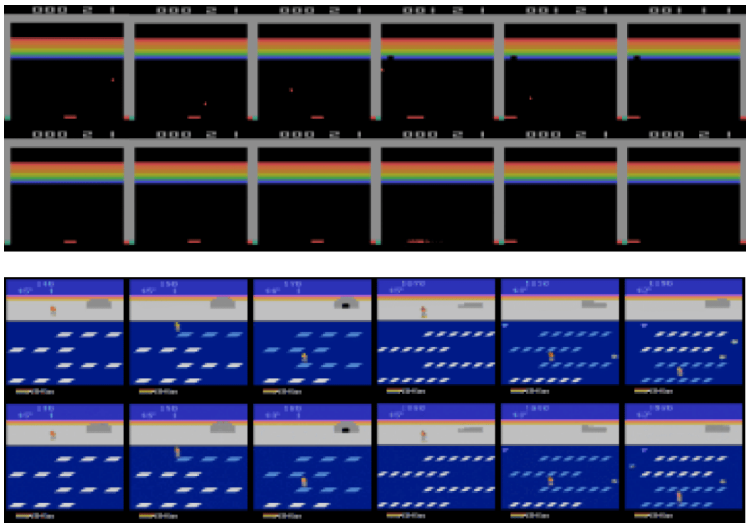

Figure 5.1: Trajectories for Breakout (top) and Frostbite (bottom). For each, the upper frame is the image observed in the environment and the lower frames are the imagined trajectories of the S5WM of the lowest level subactor.

However, F. Deng et al. (2024) only report results on their proposed set of memory testing environments and their code base is not public yet and was not made available after contacting the original authors. So this thesis is not able to compare the results of Hieros directly to S4WM. There is, however, an ablation experiment to compare S5WM to the RSSM used in DreamerV3 in Section 5.1.2.

The discussion of findings in the evaluation of Hieros on this benchmark will be done on a selection of games, namely Frostbite, Breakout, Krull, Battle Zone and Freeway. In Frostbite, the agent has to hop onto ice floes. Once it has hopped on all ice floes in the level, an ice iglu appears in the top right corner, which the agent has to approach and enter. Then a new level starts with differently shaped ice floes and other obstacles. In the game Breakout, the agent can control a small platform at the bottom of the frame and can use it to bounce a ball against bricks. When the ball touches a brick, the agent receives a reward of 1 and the ball bounces back. The goal of the game is to keep the ball from falling out of the frame and break as many bricks as possible. In Krull, the agent faces multiple levels with different dynamics and reward mechanisms. First the agent is in front of a castle and being attacked by soldiers. Then it has to travel through a desert and fight a spider in a large web. Multiple levels follow. In Battle Zone, the agent controls a tank which drives in a rudimentary 3D environment filled with enemy tanks that shoot at the agent. The tank has to approach, target and shoot the enemy tank in order to receive rewards. This game is a difficult control task, as the tank steering is rather complex.

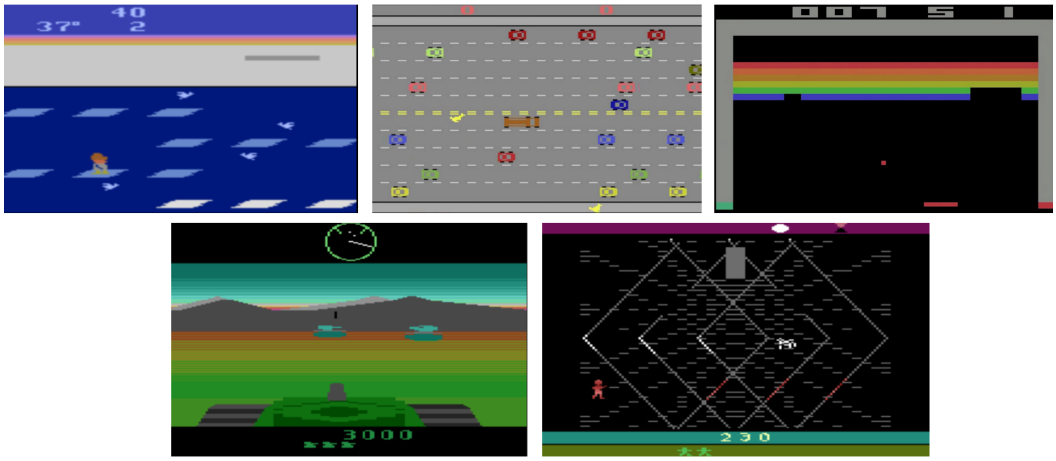

Figure 5.2: Screenshots of the games Frostbite, Freeway, Breakout (upper row) and Battle Zone and Krull (lower row) from the Atari100k benchmark.

In Freeway, the agent has to cross a street with vehicles on it driving from one site to the other. A reward is only received if the agent crosses the entire street without touching a single vehicle, which makes this game a difficult exploration problem (Micheli et al., 2022). Figure 5.2 shows example frames from each game.

5.1.1 Results

All scores are the average of three runs with different seeds. At the end of each run, the final score is computed as the accumulated reward over an episode, averaged across 100 evaluation episodes. To aggregate the results, the normalized human score (Bellemare et al., 2013) is commonly used, which is defined as $(score_{agent} - score_{random})/(score_{human} - score_{random})$. The achieved aggregated mean and median normalized human scores are shown in Table 5.1.

Hieros achieves a new state of the art in regard to the mean and median normalized human score. The agent also achieves a new state of the art in regard to the achieved reward on 9 of the 25 games. Adhering to R. Agarwal et al. (2021), the table also lists the optimality gap and the interquartile mean (IQM) of the human normalized scores and achieve state-of-the-art results in both of those metrics. R. Agarwal et al. (2021) argue, that only reporting point estimates of aggregate performance such as mean and median scores across tasks is insufficient, as it ignores the statistical uncertainty implied by the variability in performance across different tasks. This variability is better captured by metrics like the interquartile mean, which provides a more robust measure of central tendency that is less sensitive to

Task	Random	Human	SimPLe	TWM	IRIS	Director	DreamerV3	Hieros (ours)
Alien	228	7 128	617	675	420	365	**959**	828
Amidar	6	1 720	74	123	143	26	139	127
Assault	222	742	527	683	1 524	451	706	**1 764**
Asterix	210	8 503	**1 128**	1 117	854	350	932	899
BankHeist	14	753	34	467	53	12	**649**	177
Battle Zone	2 360	37 188	4 031	5 068	13 074	11 500	12 250	**15 140**
Boxing	0	12	8	78	70	13	**78**	65
Breakout	2	30	16	20	**84**	7	31	10
Chop.Command	811	7 388	979	**1 697**	1 565	801	420	1 475
CrazyClimber	10 780	35 829	62 584	71 820	59 324	68 050	**97 190**	50 857
DemonAttack	152	1 971	208	350	**2 034**	322	303	1 480
Freeway	0	30	17	24	**31**	22	0	**31**
Frostbite	65	4 335	237	1 476	259	920	909	**2 901**
Gopher	258	2 412	597	1 675	2 236	835	**3 730**	1 473
Hero	1 027	30 826	2 657	7 254	7 037	2 117	**11 161**	7 890
JamesBond	29	303	100	362	463	144	445	**939**
Kangaroo	52	3 035	51	1 240	838	402	4 098	**6 590**
Krull	1 598	2 666	2 205	6 349	6 616	7 308	7 782	**8 130**
KungFuMaster	258	22 736	14 862	**24 555**	21 760	15 663	21 420	18 793
Ms.Packman	307	6 952	1 480	1 588	999	658	1 327	**1 771**
Pong	-21	15	13	**18**	15	-18	**18**	5
PrivateEye	25	69 571	35	86	100	761	882	**1 507**
Qbert	164	13 455	1 289	3 331	746	305	**3 405**	770
RoadRunner	12	7 845	5 641	9 109	9 615	4 556	15 565	**16 950**
Seaquest	68	42 055	683	**774**	661	492	618	560
Mean	0	100	34	96	105	52	112	**120**
Median	0	100	11	50	29	18	49	**56**
IQM	0	100	13	46	50	14	N/A	**53**
Optimality Gap	100	0	73	52	51	71	N/A	**49**

Table 5.1: Scores of HIEROS and baselines on the Atari100k test suite. Higher scores for Mean, Median and IQM are better. For Optimality Gap, lower scores are better. DreamerV3 does not report the scores for IQM or Optimality Gap. We show the best results for each row in **bold** font.

outliers, and the optimality gap, which quantifies how far the performance is from the optimal policy in a more interpretable manner.

HIEROS outperforms the other approaches while having significant advantages with regard to runtime efficiency during training and inference, as well as resource demand. For training on a single Atari game for 100k steps, TWM takes roughly 0.8 days on a A100 GPU, while DreamerV3 takes 0.5 days and IRIS takes 7 days. HIEROS takes roughly 14 hours $\approx$ 0.6 days and is thus significantly faster than IRIS while being on par with DreamerV3 and TWM.

Significant improvements were achieved in Frostbite, JamesBond, and PrivateEye. These games feature multiple levels with changing dynamics and reward distributions. In order for the S5WM to learn to simulate these levels, the actor needs

to employ a sufficient exploration strategy to discover these levels. This shift in the state distribution poses a challenge for many imagination-based approaches (Micheli et al., 2022). Hieros is able to overcome this challenge by using the proposed subgoals on different time scales to guide the agent towards the next level. Section 5.1.3 shows different proposed subgoals for e.g. Frostbite, which guide the lower level actor to finding the way to the next level by building the igloo in the upper right part of the image.

Hieros seems to perform significantly worse than other approaches in Breakout or Pong, which feature no significant shift in states or dynamics. It seems as if the hierarchical structure makes it harder to grasp the relatively simple dynamics of these games, as the dynamics remain the same across all time abstractions. This is backed by the findings in Section 5.3.4 that Hieros with only one subactor performs significantly better on Breakout. Section 5.1.2 shows empirically that using the S5WM also seems to deteriorate the performance of Hieros in those games compared to the RSSM used for DreamerV3. Figure 5.5 shows some proposed subgoals for Breakout. The subgoals seem only to propose to increase the level score, which is does not provide the lower level agent any indication on how to do so. So the lower level actor is not able to benefit from the hierarchical structure in these games.

Interestingly, Director seems to show similar weaknesses and strengths, as its performance is significantly worse than the non-hierarchical approaches in tasks showcasing simpler dynamics like Pong or Breakout, while showing comparable results on tasks which contain more complex dynamics and distributions shifts like Frostbite or Krull. This indicates that, indeed, the hierarchical structure has a significant influence on the difference in performances on these groups of tasks. The deeper hierarchy and improved world model architecture help Hieros to both perform even better in complex environments with shifting distributions and not lose too much of performance in simple environments that seem to be naturally difficult for the tested hierarchical algorithms.

One possible explanation why hierarchical models might perform worse on tasks with simple dynamics and without distributional shifts could be, that in those games the optimial policy is very narrow, meaning the agent has to be precise in its action in order to achieve a high reward. In the case of Breakout this means, that the platform needs to be moved very precisely in order to bounce the ball accurately in the desired direction. Subgoals that are not learned properly or slightly outdated (which is the case when sampling subgoals from the replay buffer as it is done in Hieros) might disturb the actor-critic enough to not be able to be as precise in navigating the environment as would be necessary to achieve a high reward. This theory is partly supported by the ablation study in Section 5.3.9.

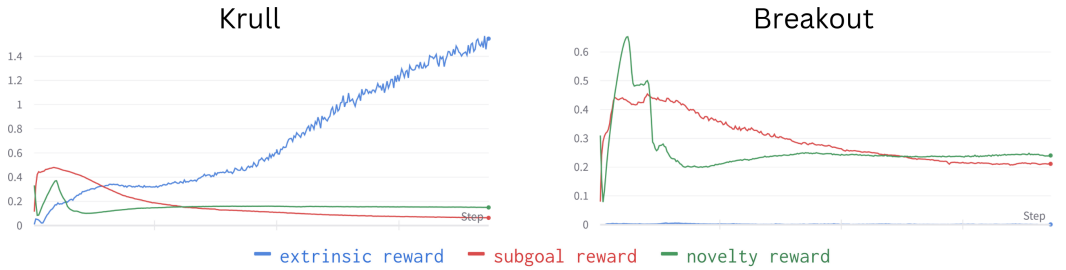

Figure 5.3: Extrinsic, subgoal, and novelty rewards per step for Krull (top) and Breakout (bottom) for the lowest level subactor.

Figure 5.3 shows the partial rewards r_{extr}, r_{nov}, and r_{sg} for the lowest level subactor for Breakout and Krull, another game featuring multiple levels similar to Frostbite. As can be seen, the extrinsic rewards for Breakout are very sparse and do not provide any indication on how to increase the level score. So the subactor learns to follow the subgoals from the higher levels more closely instead, which in the case of Breakout or Pong does not lead to a better performance. In Krull however, the extrinsic rewards are more frequent and provide a better indication on how to increase the level score. So the subactor is able to learn to follow the subgoals from the higher levels more loosely, treating them more like a hint, and is able to achieve a better performance.

5.1.2 Imagined Trajectories

Figure 5.1 shows an observed trajectory for the games Frostbite and Breakout alongside the imagined trajectories of the S5WM of the lowest level subactor. The imagination starts from the first observation. Then the actions that created the observed trajectory are passed to the S5WM to create the imagined trajectory. A well-trained world model should have no or only slight differences to the observed trajectory.

As can be seen, the world model is not able to predict the movement of the ball in Breakout. This indicates that the model is not able to model the very small impact of the ball movement to the change in the image. Deeper investigations into the replay buffer showed that during random exploration at the beginning of the training, the events where the ball bounces back from the moving plateau are very rare and most of the time the ball is lost before it can bounce back. This makes it difficult for the world model to learn the dynamics of the ball movement and their impact on the reward distribution. A similar performance of the world model can be observed for the game Pong.

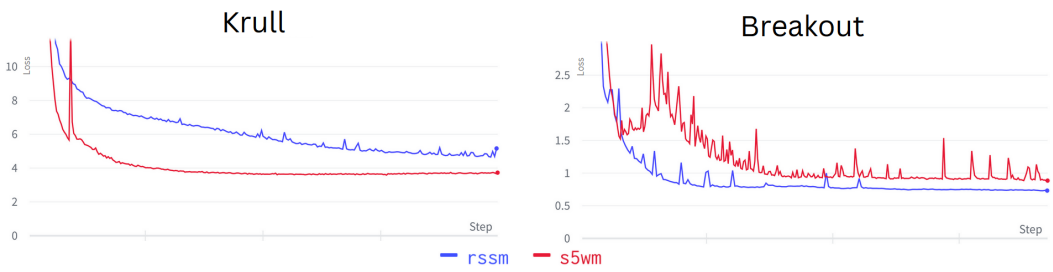

Figure 5.4: World model losses for the S5WM and RSSM for Krull and Breakout. The S5WM is able to achieve an overall lower world model loss compared to the RSSM for Krull, while those roles are reversed for Breakout.

S4-based models are shown to perform worse than Transformer models on short term sequence modeling tasks (Dao et al., 2022; Mehta et al., 2022; Zuo et al., 2022) while excelling on long term modeling tasks. This could also give some reasoning why our S5WM seems to perform worse on Breakout or Pong compared to Frostbite or Krull. Freeway poses a similar challenge, with sparse rewards that are only achieved after a complex series of environment interaction. However, unlike with Breakout and Pong, in Freeway Hieros is able to profit from its hierarchical structure in order to guide exploration. An example of this can be seen in Figure 5.5, where the subgoals are able to guide the agent across the road. S5WM is able to accurately capture the multiple levels of Frostbite, despite only having access to train on these levels after discovering them, which usually happens after roughly 50k interactions. As the reaching of the next level is directly connected to a large increase in rewards, the S5WM is able to correctly predict the next level after only a few interactions. This is also reflected in the significantly higher reward achieved by Hieros.

To directly compare the influence of our S5WM architecture to the RSSM used in DreamerV3, the S5WM was replaced with an RSSM. With this change, Hieros was trained on four different games. The results are shown in Section 5.3.2. The RSSM is not able to achieve the same performance as the S5WM for Krull, Battle Zone and Freeway, but is slightly better compared to using the S5WM for Breakout. Figure 5.4 shows the world model losses for S5WM and RSSM for Krull and Breakout. The S5WM consistently achieves lower overall loss than the RSSM in Krull, whereas this trend reverses for Breakout. Given the substantially higher absolute loss values for the more complex Krull compared to Breakout, it seems that the larger S5WM excels in environments with complex dynamics and substantial input distribution shifts. In contrast, the smaller RSSM is better suited for environments with simple dynamics and a stable input distribution.

As both models are trained with the same number of gradient updates, it makes

sense that the larger model has difficulties matching the performance of the smaller model in non-shifting environments with simple dynamics. So a smaller S5WM might achieve better results for Breakout. This hypothesis is tesed in Section 5.3.7. Lu et al. (2024) and F. Deng et al. (2024) find that structured state space models generally surpass RNNs in terms of memory recollection and resilience to distribution shifts, which can be confirmed with the results shown above.

5.1.3 Visualization of Proposed Subgoals

Figure 5.5 shows the proposed subgoals for one observation in Frostbite, Breakout, and for Freeway. As can be seen, for Breakout, the subgoals are only to increase the level score and the ball is not simulated at all, while for Frostbite the subgoals guide the actor towards building up the igloo in the upper right part of the image in order to advance to the next level. For Freeway, which also features a single level and sparse rewards, the subgoals are much more meaningful than for Breakout and guide the actor to move across the road.

To give more insight into how Hieros formulates subgoals in different contexts, Figure 5.6 shows more proposed subgoals from a larger variety of games.

5.2 DeepMind Visual Control Suite

This section shows the results of Hieros on the DeepMind Control (DMC) suite. Each task of the suite contains one robot consisting of multiple joints, which the agent can control using continuous valued actions. The agent has a budget of 1 million environment interactions. Hieros is compared against four other models: DreamerV3 (Hafner, Pasukonis, et al., 2023), DrQ-v2 (Yarats et al., 2021), CURL (Laskin et al., 2020) and a Soft-Actor-Critic baseline (Haarnoja, Zhou, Abbeel, et al., 2018). DrQ-v2 is a model free RL agent, which employs special data augmentation techniques to the raw pixel input and achieves a state-of-the-art in the DMC suite compared to other model free techniques. CURL uses unsupervised contrastive learning to extract features from the raw pixels, on which a model free RL policy is trained. Soft-Actor-Critic was already explained in detail in Section 2.1.5. DrQ-v2 and CURL are two recently proposed model free approaches that outperformed other model based and model free agents when they were introduced, which is why Hieros and DreamerV3 are compared against those models.

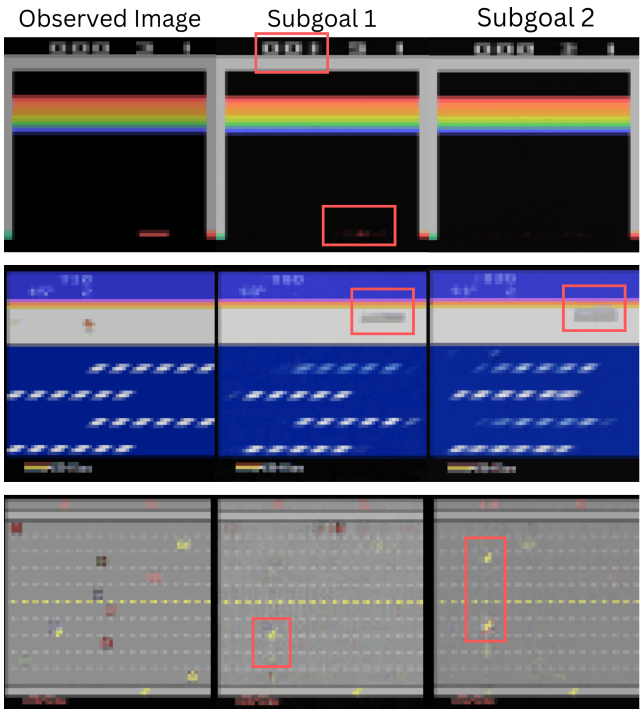

Figure 5.5: Proposed subgoals for Breakout (top row), Frostbite (middle row), and Freeway (bottom row). The left most frame is the original observation from the environment, and the following frames are the proposed subgoals from the higher level actor. For Breakout, the subgoals are only to increase the level score (marked with the red rectangles) and the ball is not simulated at all, while for Frostbite the subgoals guide the actor towards building up the igloo in the upper right part of the image in order to advance to the next level (red rectangles). For Freeway, which also features a single level and sparse rewards, the subgoals are much more meaningful than for Breakout and guide the actor to move across the road (red rectangles).

5.2.1 Results

For comparison, the final reward collected over one episode averaged across 100 evaluation episodes is reported in Section 5.2.1.

As can be seen, Hieros is not able to achieve a new state of the art in the DMC suite. However, the mean and median performance is close to DreamerV3, which indicates that the improvements that helped Hieros to achieve a new state of the art in the Atari100k benchmark do not bring a significant advantage in the continuous control setting. As the development of Hieros was mainly guided by experiments on the Atari100k benchmark, it is possible, that slight variations in the architecture might balance out the shortcomings of the agent on the DMC benchmark.

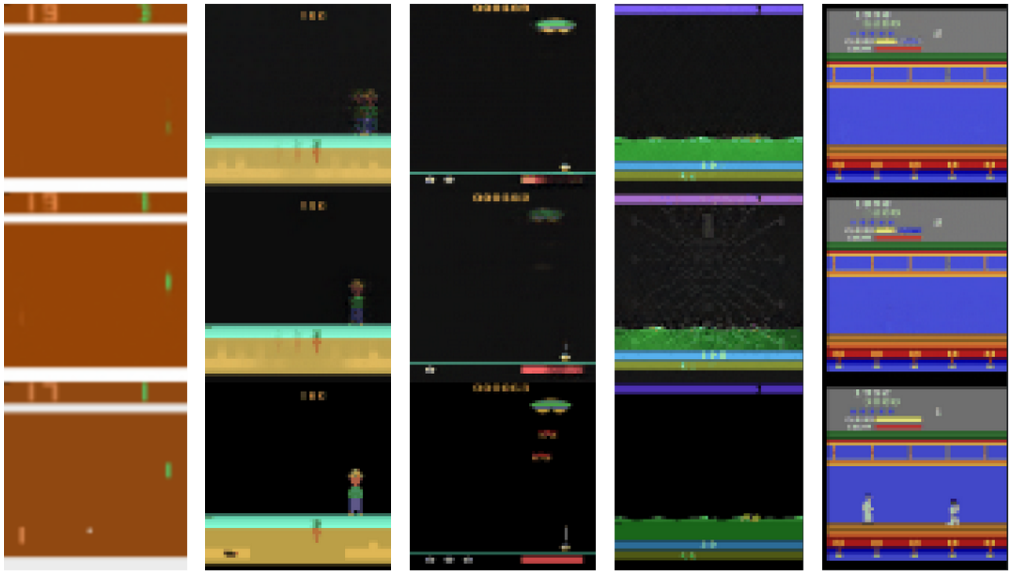

Figure 5.6: Proposed subgoals for (from left to right) Pong, Gopher, Assault, Krull and Kung Fu Master. The bottom row is the actual observation, the middle row the proposed subgoal from subactor 1 and the top row the proposed subgoal from subactor 2.

One possible explanation for the poor performance of Hieros on this benchmark could be, that the proposed subgoals of 8 categorical vectors of length 8 contain too little information in order for the higher level subactors to effectively guide the lower level subactors towards actions that yield a higher reward. The subgoal space allows for only $8^8 = 2048$ different subgoals, which might not be sufficient for a continuous valued action space and a high dimensional visual observation space to form effective strategies that maximize the reward. Another possible explanation could be, that the latent space of the world model itself is not expressive enough. For Hieros smaller network sizes for each subactor were used, with a deterministic state dimension of 256 and a stochastic state size of 32 categorical vectors of length 32. The original DreamerV3 architecture uses a twice as large deterministic state dimension of 512 for the DMC benchmark. The limited capacity of the world state latent space could lead to imprecise representations and incorrect dynamic predictions.

This can be shown in the high dynamic and representation losses recorded for the training of Hieros. The loss curves are shown in Figure 5.7. To rule out the S5WM to be responsible for the high losses, the figure also shows the loss curves with Hieros with a RSSM world model. For comparison, the dynamics and representation losses for some Atari100k games are shown below. The losses

Task	SAC	CURL	DrQ-v2	DreamerV3	Hieros
Acrobot Swingup	5.1	5.1	128.4	**210**	90.4
Cartpole Balance	963.1	979	991.5	996.4	**997.2**
Cartpole Balance Sparse	950.8	981	996.2	**1000**	**1000**
Cartpole Swingup	692.1	762.7	**858.9**	819.1	813.7
Cartpole Swingup Sparse	154.6	236.2	706.9	**792.9**	374.1
Cheetah Run	27.2	474.3	691	**728.7**	380
Cup Catch	163.9	**965.5**	931.8	957.1	942.8
Finger Spin	312.2	877.1	846.7	818.5	**890.2**
Finger Turn Easy	176.7	338	448.4	787.7	**795**
Finger Turn Hard	70.5	215.6	220	**810.8**	641.6
Hopper Hop	3.1	152.5	189.9	369.6	**376.5**
Hopper Stand	5.2	786.8	893	**900.6**	877.1
Pendulum Swingup	560.1	376.4	**839.7**	806.3	797.3
Quadruped Run	50.5	141.5	**407**	352.3	201.4
Quadruped Walk	49.7	123.7	**660.3**	352.6	213.7
Reacher Easy	86.5	609.3	**910.2**	898.9	905.3
Reacher Hard	9.1	400.2	**572.9**	499.2	534.7
Walker Run	26.9	376.2	517.1	**757.8**	648.6
Walker Stand	159.3	463.5	974.1	976.7	**780.3**
Walker Walk	38.9	828.8	762.9	**955.8**	791.3
Median	78.5	431.8	734.9	**808.5**	785.8
Mean	225.3	504.7	677.4	**739.6**	652.56

Table 5.2: The score of Hieros on the DeepMind Control (DMC) suite compared against baselines. In the two bottom rows the mean and the median score of all tasks is shown. The best scores in each row are shown in **bold** font.

are significantly higher for the complex tasks like Walker Walk and Quadruped Walk, regardless of the used world model architecture. This indicates that the model indeed is not able to learn informative representations and to predict the dynamics of the more complex DMC environments accurately. The limited size of the latent space is one possible explanation for this. As can be seen, the high dynamics and representation losses are not encountered in the Atari100k games, which indicates, that the differences in loss might be one of the main causes of Hieros' poor performance on the DMC benchmark.

A lack of exploration capability is especially noticeable in more complex environments like Quadruped Walk or Run, where the agent controls a robot with four

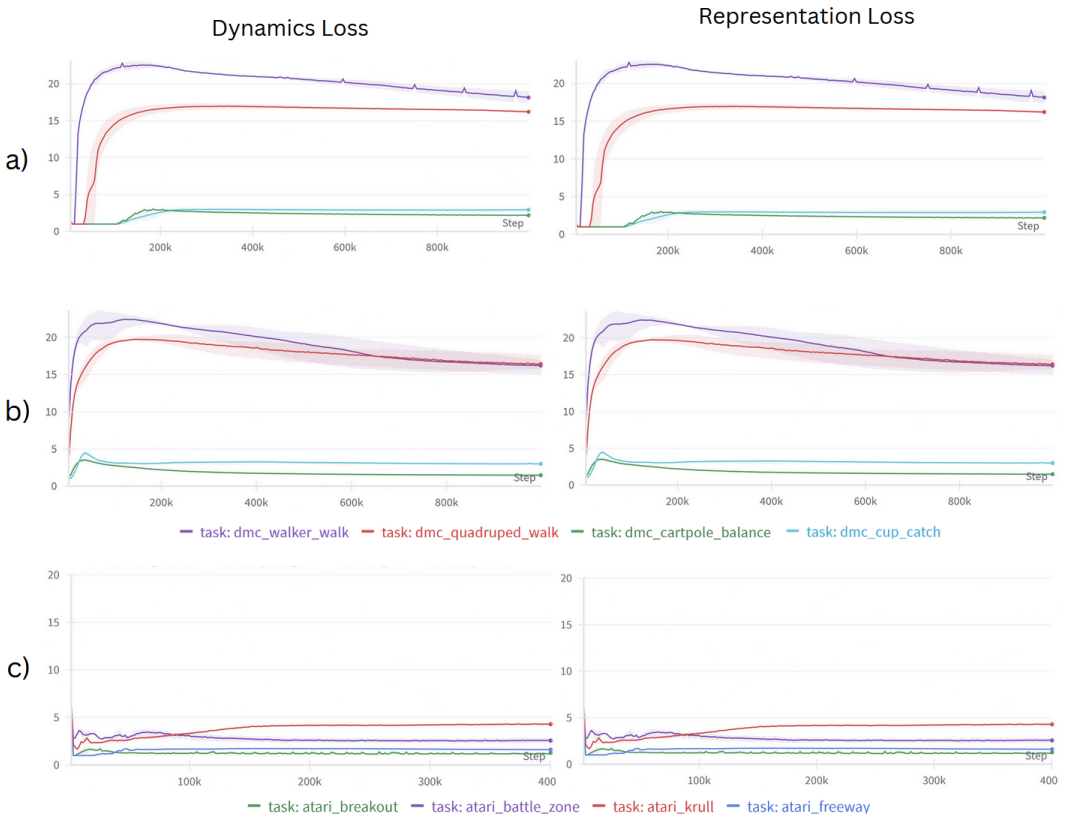

Figure 5.7: Loss graphs containing the dynamics losses (left column) and representation losses. a) contains loss curves for HIEROS training on DMC tasks Cartpole Balance, Cup Catch, Quadruped Walk and Walker Walk. The more complex tasks Quadruped Walk and Walker Walk show a significantly higher loss in both columns than the simpler environments. b) contains loss curves for HIEROS training on the same tasks using an RSSM world model. These graphs show that the lack of dynamic prediction quality and representation learning is not caused by the use of the S5WM, but also present when using the same world model as DreamerV3. c) contains loss curves of HIEROS training on four Atari100k games: Krull, Battle Zone, Freeway and Breakout. This is just for comparison to illustrate, that the high dynamics and representation losses are not encountered in the Atari100k games, which indicates, that the differences in loss might be one of the main causes of HIEROS' poor performance on the DMC benchmark.

legs and is rewarded by the forward velocity achieved by moving the legs of the robot. The robot has a total of 12 joints (3 per leg), which makes controlling the robot an especially difficult task. The robot starts from a random position at the start of the episode, which often results in it falling on its back first. During the

experiments, Hieros seldom worked out, how to turn the robot from its back onto its feet to walk faster and collect higher rewards. From the results on the Atari100k benchmark it would have been reasonable to assume that the hierarchical structure and superior dynamic modeling capabilities of the S5WM would help Hieros to achieve superior performance in difficult control tasks, as these problems have a clear intrinsic hierarchical structure (move single joints would be the lowest level of interaction, moving a leg and finally moving the entire robot would mark higher levels of abstraction). The limited capacity of the subgoal and world model space might explain the poor performance on tasks that require good exploration skills.

As already stated in Section 5.1.1, another possible explanation could be, that due to the hierarchic structure of Hieros, the precise control of the robot joints is made more difficult, as slightly outdated subgoals (as the ones sampled from the replay buffer would be) can distract the actor-critic into following these imprecise subgoals rather than the actual environment task.

Hieros achieves competitive results on some of the easier tasks, like Cartpole Balance, where the agent has to move a platform with a pole attached and is rewarded if it can balance the pole so that it points upwards, Finger Spin, where the agent controls a finger with two joints to spin a rotating element as fast as possible, or Finger Turn, where the agent is rewarded if it rotates an object with a finger by a fixed degree. This indicates, that on tasks that are simple enough, the world state has enough capacity to fully express the current state of the environment, which leads to Hieros achieving new state-of-the-art results on a few of those simpler tasks.

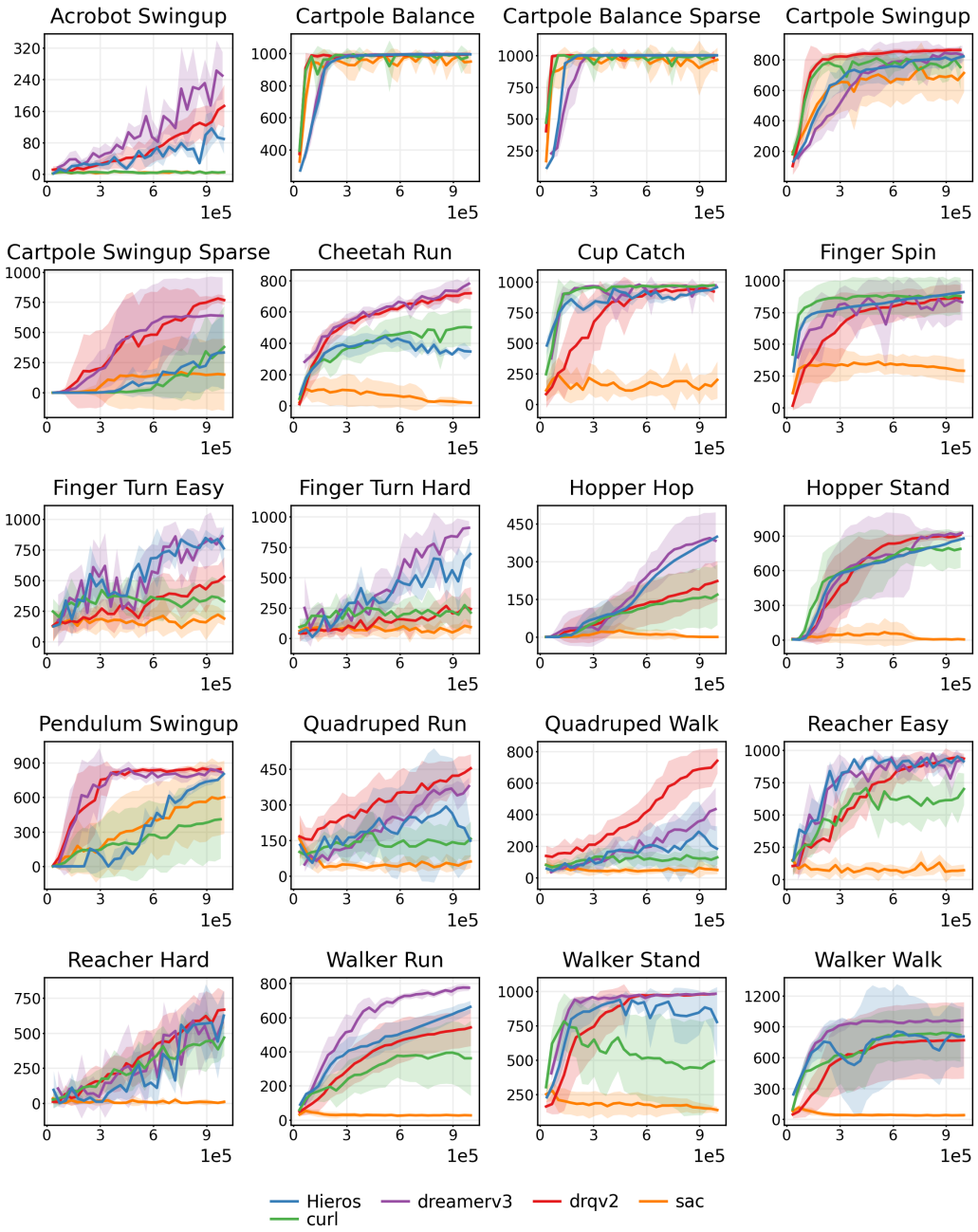

Figure 5.8: Graphs showing the collected rewards per episode over the entire training process for Hieros and the four compared baselines. The curves are aggregated from multiple runs with different seeds. The lines indicate the average achieved score, while the shaded areas around the line indicate the range between the minimal and maximal values.

5.2.2 Training Curves

To compare, how quickly the agents optimize their policies, Figure 5.8 shows the collected rewards per episode over the entire training process for all five compared models. As can be seen, HIEROS shows no significant difference in how quickly it optimizes its policy for most tasks. In the case of Pendulum Swingup, where the agent can exhibit force on a pendulum that is attached to a fixed point and receives rewards if the pendulum is pointing upwards, HIEROS takes considerably longer than DreamerV3 and DrQ-v2 to achieve a high reward, which indicates, that in this case, HIEROS needed more environment interactions to explore reward yielding strategies. As stated in Tassa et al. (2018), in the Pendulum Swingup task, the agent can exhibit $\frac{1}{6}$ of the force needed to push the pendulum to the upwards position, so multiple swings are needed to reach this state. This poses a more difficult exploration problem for the agent and due to the limited capabilities of HIEROS to effectively find exploration strategies, it needs more interactions to train its policy than DreamerV3 or DrQ-v2. In some of the more complex environments, like Quadruped Run or Walker Walk, HIEROS shows a very high variance of achieved scores. This is caused by the high complexity of the tasks, as the policy is not able to converge to a policy that yields similar results in every run.

5.2.3 Visualization of the Proposed Subgoals

Figure 5.9 shows proposed subgoals for Cup Catch, Cartpole Balance, Walker Walk and Quadruped Walk. As can be seen, the proposed subgoals are often very blurred, indicating, that they represent a multitude of different states of the robot. This partially confirms the theory, that the subgoal space is not expressive enough, as one subgoal can represent a wide range of different states, which makes it difficult for the lower level subactor to precisely follow the proposed subgoal.

Overall, the results of HIEROS on the DMC suite show, that the implementation of the architecture and training details need to be examined to find the precise cause of the comparatively poor performance on the DMC benchmark. Also, a separate experiment is needed to confirm, that the small subgoal space indeed has a negative impact on the agent's performance and if a larger subgoal space (either by making the subgoal space continuous valued or by increasing the subgoal dimensions) can increase the achieved scores of HIEROS on complex tasks.

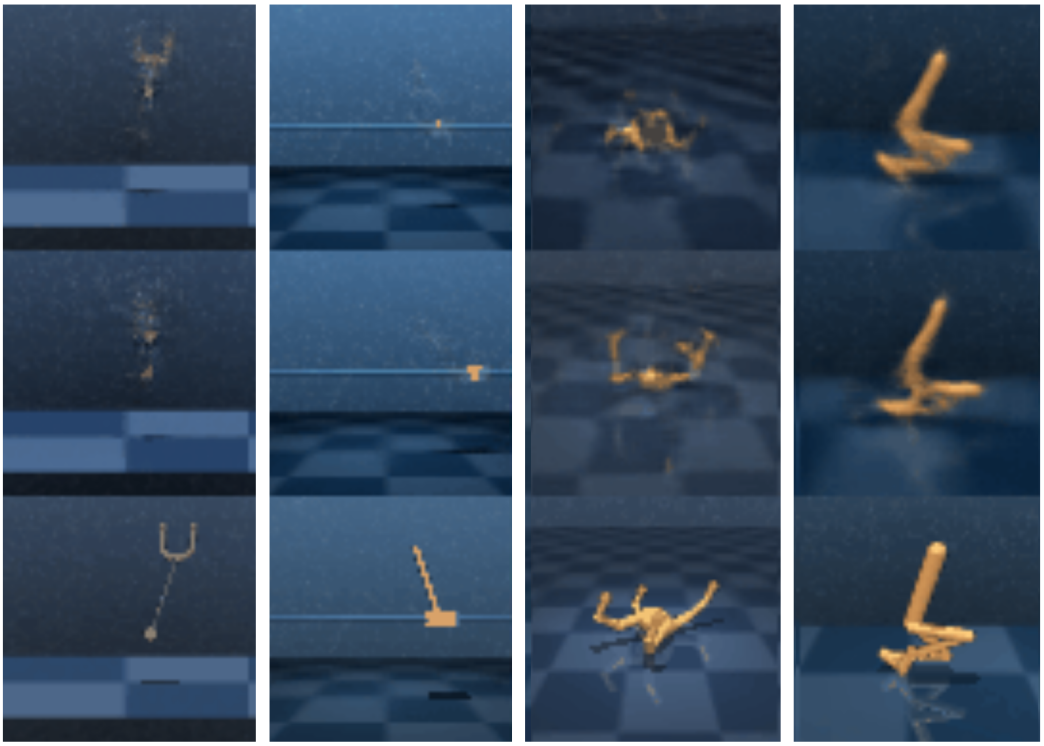

Figure 5.9: Proposed subgoals for (from left to right) Cup Catch, Cartpole Balance, Quadruped Walk and Walker Walk. The lowest row frames are the original observation from the environment, and the middle and upper row frames are the proposed subgoals from the second and third higher level subactors. For all four tasks, the subgoals are very blurry, indicating that they represent a multitude of different possible world states, which make them less usable to guide the lowest level actor towards strategies of high reward.

5.3 Ablations

The following section provides additional ablation studies exploring the effect of different components of Hieros. The ablations are conducted on four Atari100k games: (i) Krull, a game that features multiple levels, (ii) Breakout, a game with a single level and simple dynamics, (iii) Battle Zone, a game with a single level and complex hierarchical dynamics and (iv) Freeway, a game with difficult exploration properties (Micheli et al., 2022). Since training Hieros on a DMC task takes much more time than training on a game of the Atari100k benchmark (14 hours vs 2 days), the ablations are limited to the Atari100k benchmark.

Hieros uses the same hyperparameters as described in Section 4.6.3 for all ablations. The figures show the collected reward of the lowest level subactor of

HIEROS in red and the collected reward of HIEROS with the ablation in blue. This color scheme is used for all ablation studies, except those where the graphs contain more than two lines.

The experiments use three training runs for each ablation, and the graphs illustrate the average performance across those runs as a line. The shaded area around the line, in the same color, represents the range of values observed during the three runs. In every interaction with the environment, the actor's action is repeated for four frames, aligning with the methodology adopted in other papers such as Hafner, Lillicrap, Norouzi, et al. (2020), Hafner, Pasukonis, et al. (2023), Micheli et al. (2022), and Robine et al. (2023). Consequently, the plots display results up to 400k steps, but it's important to note that only 100k interactions were actually conducted in the specified environments due to the frame repetition.

The first three ablations test out the key claims of this thesis: 1) hierarchical imagination makes training of multilayered HRL agents stable and significantly improves performance, 2) the S5WM outperforms the RSSM when used in HIEROS and 3) using ETBS for sampling improves the overall performance of the agent compared to uniform sampling without having a significant impact on the runtime efficiency. These follow nine more ablation experiments, which test out several different variations of the HIEROS architecture.

5.3.1 HIEROS Without Hierarchical Imagination

The main novelty that HIEROS introduces, is the use of world models on every level of the hierarchy. This ablation experiment compares HIEROS to a hierarchy of subactors that uses only one world model. This architecture works similarly as Director (Hafner, Lee, et al., 2022), which also uses only one world model. Each higher level subactor i receives k^i consecutive world model states (k being the update interval of the subactors) of the world model as observation, instead of a concatenation of k states of its lower level subactors world model. This ablation tests one of the key claims of this thesis, that providing world models to all abstraction levels stabilizes and greatly benefits the performance of the RL agent.

Figure 5.10 shows the performance of HIEROS with and without hierarchical imagination. As can be seen, using only one world model for all subactor trainings significantly reduces the performance of HIEROS in Krull, Battle Zone and Freeway. In the case of Freeway, a game that requires complex exploration in order to encounter the sparse rewards of this environment, the version without hierarchical imagination is not able to find a policy that pursues a winning strategy at all, resulting in an obtained score of close to zero. A large spike of the collected rewards per step can be seen in three of the four graphs, after which the performance

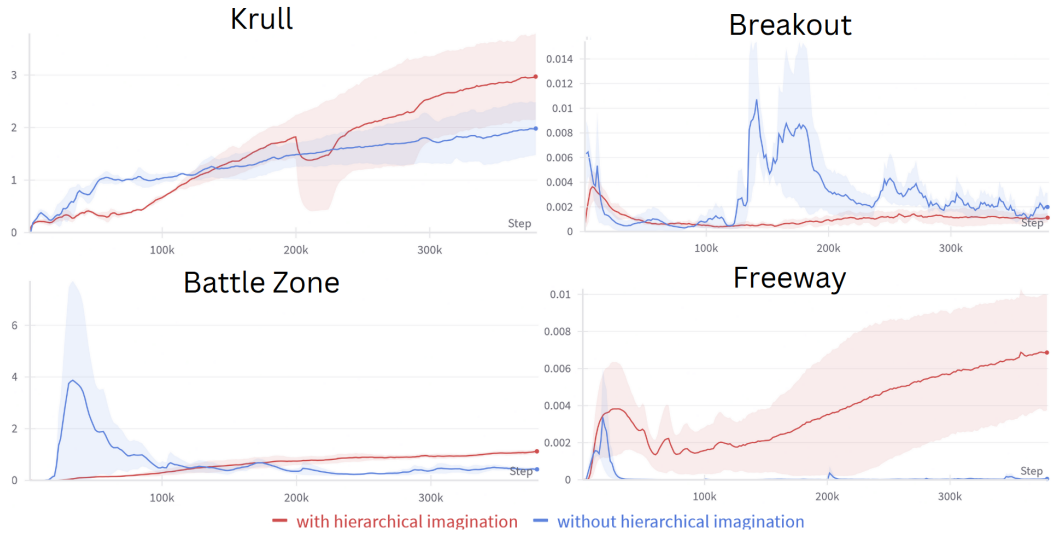

Figure 5.10: Comparison of the collected reward per step of HIEROS when supplying each subactor with its own world model (i.e. using hierarchical imagination) (red) and when using only a single world model for all subactors (blue) for Krull, Breakout, Battle Zone, and Freeway. The higher the collected reward, the better the agent performed.

degrades and finally is either on par or significantly lower than HIEROS with hierarchical imagination. This shows, that providing each subactor with its own world model indeed stabilizes the learning process and the communication between subactors. The hierarchical imagination makes multilayered HRL agents trainable without large instabilities.

5.3.2 S5WM vs. RSSM

Section 5.1.2 compares the model losses and partial rewards of HIEROS in the game of Krull and Breakout using either RSSMs or S5WMs as world models. This ablation tests another key claim of this thesis, that the S5WM indeed is able to model complex environments with long term memory requirements better than the RSSM used in DreamerV3.

Figure 5.11 compares the collected rewards of HIEROS using either RSSMs (blue) or S5WMs (red) for Krull, Breakout, Battle Zone, and Freeway against each other. The RSSM is not able to achieve the same performance as the S5WM for Krull, Battle Zone and Freeway. However, for Breakout, the RSSM exhibits a slight improvement compared to the S5WM. This aligns with the observation made in Section 5.1.2, where it was noted that the world model losses for Breakout were lower when

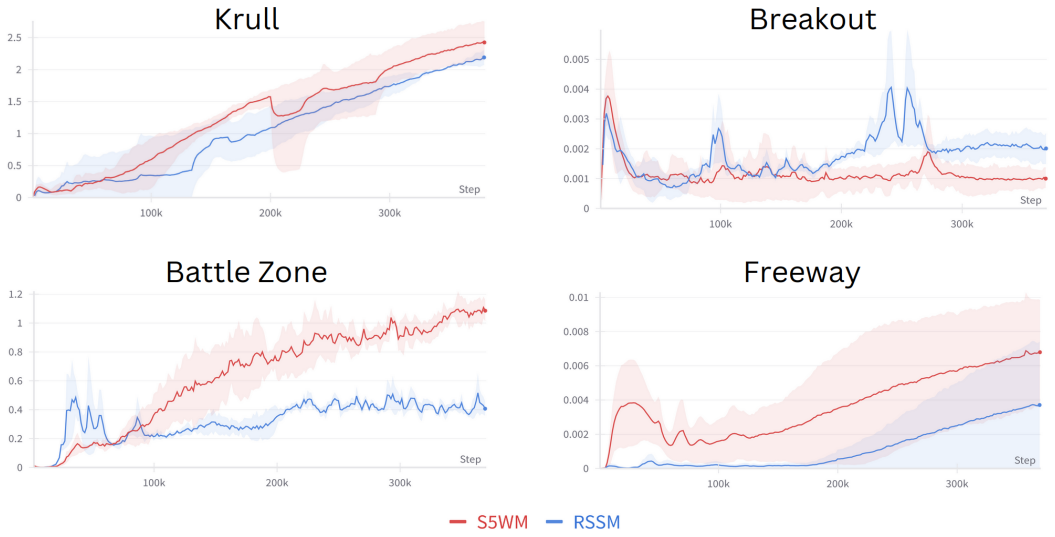

Figure 5.11: Collected rewards per step for Krull, Breakout, Battle Zone, and Freeway of the lowest level subactor for HIEROS with an RSSM (blue) and HIEROS with an S5WM (red). The higher the collected reward, the better the agent performed.

employing an RSSM as the dynamics model compared to using the S5WM. Overall, this experiment shows, that the S5WM indeed has a net positive influence on the performance of HIEROS.

Another thing to note is the impact of the used world model architecture on the overall runtime performance. With the computational resources used for the experiments, HIEROS with the S5WM world model runs with approximately **7.73** frames per second on average. Every four frames, the agent interacts with the environment and trains its networks. With the RSSM, the frames per second drop to **4.85** frames per second on average. This shows, that the parallel training of the sequence model in fact has a significant impact on the overall runtime efficiency of the agent.

5.3.3 Uniform vs. Time-Balanced Replay Sampling

Section 4.5 describes the efficient time-balanced sampling (ETBS) method. This subsection compares the effect of using uniform sampling vs. the proposed efficient time-balanced sampling for the experience dataset. Figure 5.12 shows the collected reward of HIEROS with S5WM using uniform sampling (red) and time-balanced sampling (blue) for Krull, Breakout, Battle Zone, and Freeway. As can be seen, in the case of Freeway and Breakout, the time balanced sampling did not provide a

significant advantage, while it boosted the performance considerably for Krull and Battle Zone, two games that both display hierarchical challenges. This indicates, that in cases where the model hierarchy can contribute a lot to the overall performance, the actor is more sensitive to overfitting on older training data, containing subgoals produced by less trained higher level subactors.

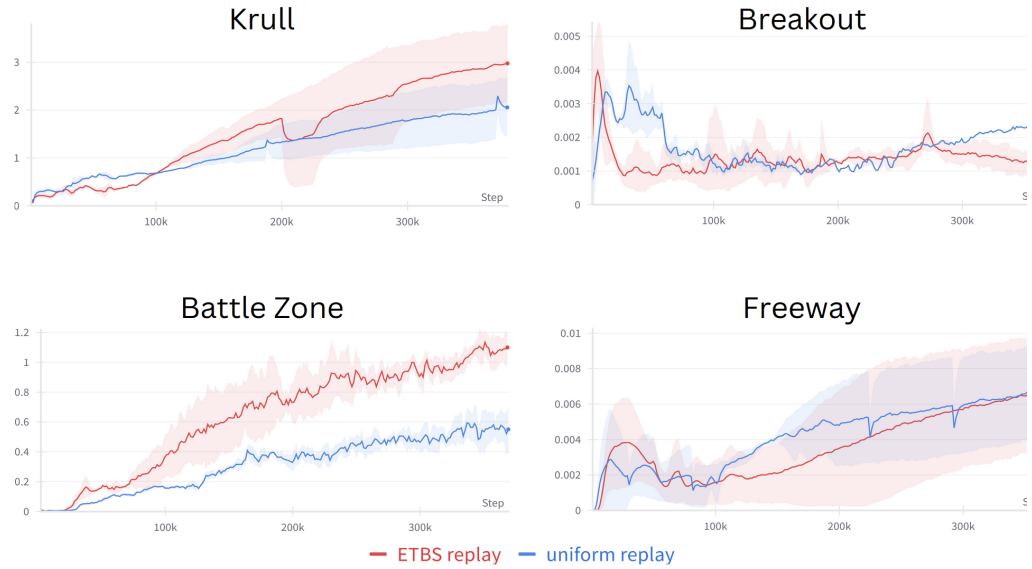

Figure 5.12: Comparison of the collected reward per step of Hieros with S5WM using uniform sampling (red) and time-balanced sampling (blue) for Krull, Breakout, Battle Zone, and Freeway. The higher the collected reward, the better the agent performed.

Figure 5.13 compares the frames per second of Hieros using ETBS against the sampling method proposed by Robine et al. (2023), which produces the same results but has a runtime efficiency of $O(n)$, n being the size of the replay buffer. The difference in runtime efficiency is clearly visible and shows that the speedup achieved by ETBS has a significant impact on the overall runtime of the algorithm.

5.3.4 Hierarchy Depth

This subsection shows the effect of using different numbers of subactors on the overall performance of the agent. The experiments evaluate Hieros with S5WM using a model hierarchy depth of 1, 2, 3, and 4. Figure 5.14 shows the collected reward of Hieros with S5WM using a model hierarchy depth of 1 (purple), 2 (red), 3 (green), and 4 (light blue) for Krull, Breakout, Battle Zone, and Freeway. Most remarkably, the graphic shows that Hieros with just one layer achieves significantly

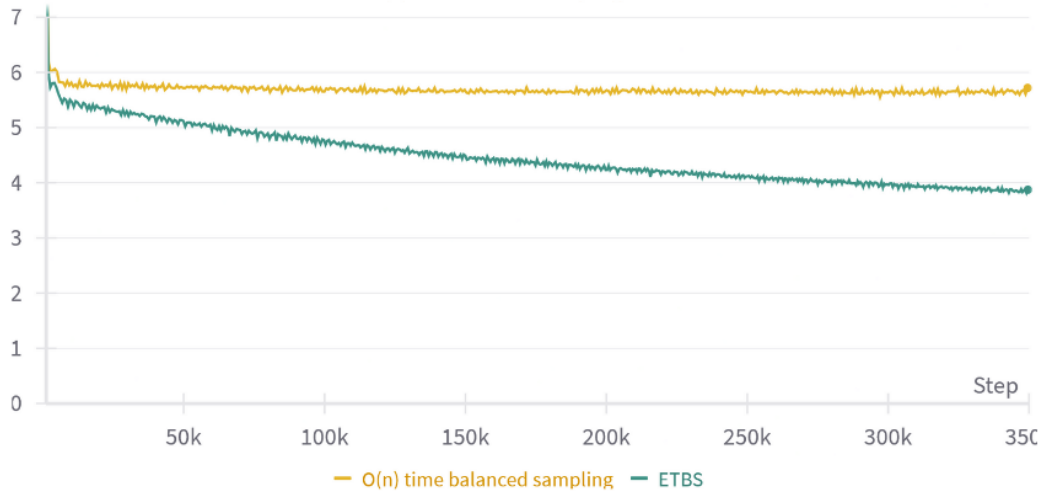

Figure 5.13: Comparison of the frames per second of HIEROS using ETBS (green) and the time balanced sampling strategy in O(n) (yellow) proposed by Robine et al. (2023). With the O(n) sampling strategy, the frames per second decrease through the course of the training. With ETBS they stay constant.

better results in Breakout than with two or more subactors. This indicates that a single layer algorithms, like DreamerV3, Iris, or TWM have a significant advantage over multi-layer algorithms, like HIEROS in games with no distribution shifts and easy to predict dynamics like Pong or Breakout while deeper hierarchies showcase a better performance for games with an intrinsic hierarchical structure like Krull.

As noted in Section 4.6.3, HIEROS uses three subactors per default. This choice is confirmed by this ablation study, as three subactors achieve the highest overall performance. Also, this experiment shows that the hierarchy of HIEROS does not seem to scale all too well, as using 4 subactors already seems to deteriorate the overall performance of the agent. Adding more than 4 hierarchy layers does not seem a promising approach to further boost the performance of HIEROS. The main reason for this could be that higher levels collect data in their replay buffers much slower than lower levels, which leads to them being undertrained. It is possible, that a different training ratio could prove beneficial for very deep hierarchies (i.e. more than four layers). D'Oro et al. (2023) show, that using a very large training ratio (training steps per environment interaction) can indeed improve the scaling behavior of many RL algorithms. Testing this with HIEROS, however, remains for future work.

However, this ablation study shows that indeed the deeper hierarchy used in HIEROS is one of the main causes for its poor performance on games with simple

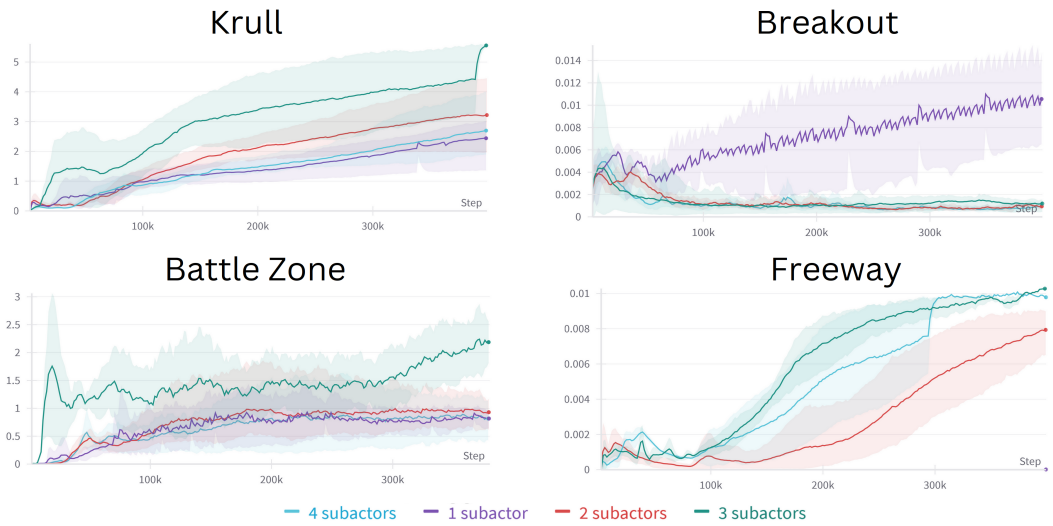

Figure 5.14: Comparison of the collected reward per step of HIEROS with S5WM using a model hierarchy depth of 1 (purple), 2 (red), 3 (green), and 4 (light blue) for Krull, Breakout, Battle Zone, and Freeway. The higher the collected reward, the better the agent performed.

dynamics like Breakout and Pong. This was already indicated by Director's poor performance on the same set of problems. Since HIEROS with only 1 layer performs significantly better at Breakout, there might be an underlying limitation inherent to the way hierarchical policies interact and train with each other in HIEROS, that causes it to perform significantly worse in environments with simple dynamics.

5.3.5 Internal S5 Layer State as Deterministic World State

Section 4.4 describes the S5WM, which uses the S5 layer to predict the next world state. This ablation experiment explores the effect of using the internal state x_t of the stacked S5 layers of S5WM as the deterministic part of the latent state h_t instead of the output m_t of the S5 layers. Other comparable approaches that replace the GRU in the RSSM with a sequence model usually use the output of the sequence model as h_t (Chen et al., 2022; F. Deng et al., 2024; Micheli et al., 2022; Robine et al., 2023). So, testing this ablation provides valuable insight into how the learned world states can be enhanced in order to boost prediction performance. Figure 5.15 shows the influence of using the internal state as h_t vs. using the output of the S5 layers as h_t for HIEROS. Using the internal S5 layer state as deterministic world model state seems to improve the performance for Krull and Battle Zone while having no clear benefit for Breakout and Freeway.

This confirms that using the hidden state of the sequence model indeed is a better representation of the deterministic world state than the output of the world model, which is only used in HIEROS to parameterize the prior stochastic state distribution.

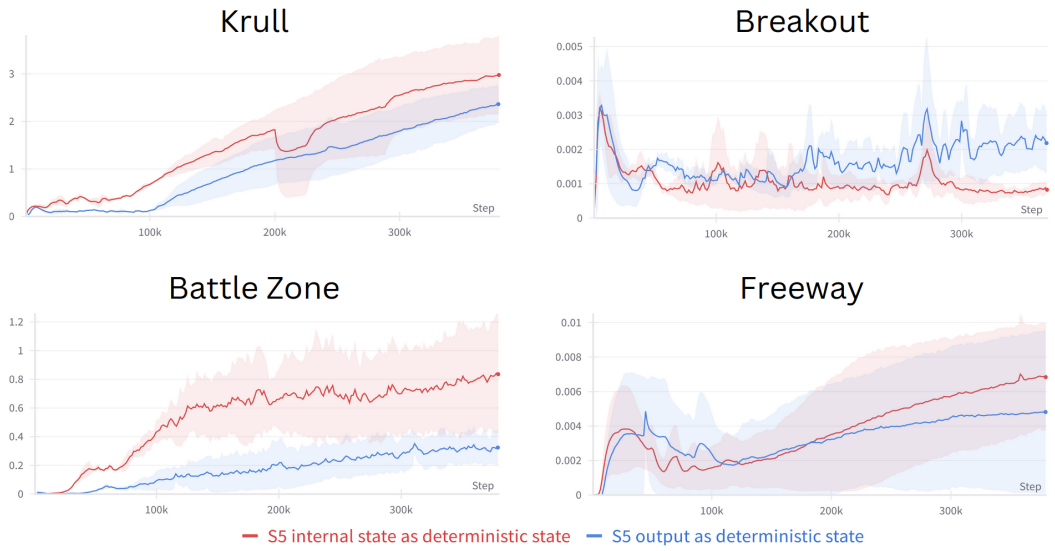

Figure 5.15: Comparison of the collected reward per step of HIEROS with S5WM using the internal state x_t of the stacked S5 layers as the deterministic part of the latent state h_t (red) and using the output of the stacked S5 layers as h_t (blue) for Krull, Breakout, Battle Zone, and Freeway. The higher the collected reward, the better the agent performed.

5.3.6 Providing k World States vs. Only the k-th World State as Input for the Higher Level World Model

Section 4.3 describes how HIEROS provides k consecutive world states of the lower level world model as input for the higher level subactor. However, many approaches such as Director (Hafner, Lee, et al., 2022) only provide the k-th world state as input for the higher level world model. This section explores the effect of providing only the k-th world state as input for the higher level world model. Figure 5.16 shows the collected reward of HIEROS with S5WM using k consecutive world states as input for the higher level world model (red) and only the k-th world state as input for the higher level world model (blue) for Krull, Breakout, Battle Zone, and Freeway. Providing only the k-th world state as input seems to deteriorate the performance for Battle Zone and Freeway, while being beneficial for Krull. Overall, providing k consecutive observations to the higher level subactor yields higher rewards on average, which is why this is used in HIEROS.

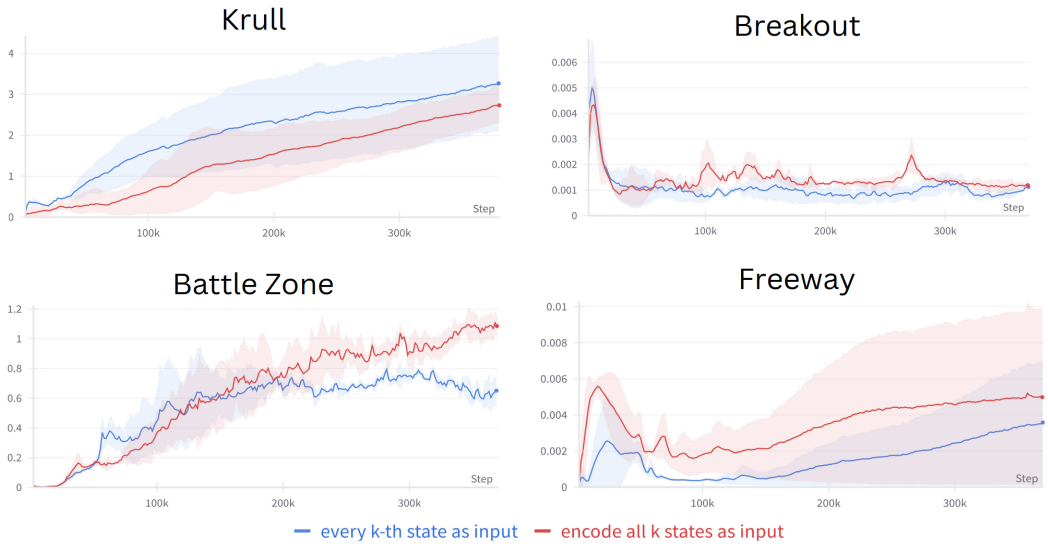

Figure 5.16: Comparison of the collected reward per step of HIEROS with S5WM using k consecutive world states as input for the higher level world model (red) and only the k-th world state as input for the higher level world model (blue) for Krull, Breakout, Battle Zone, and Freeway. The higher the collected reward, the better the agent performed.

5.3.7 Using a Smaller Version of S5WM

Section 5.1.2 contains a theory that suggests larger S5WM models yield superior results in complex environments, whereas smaller dynamic models may be more advantageous in simpler dynamic environments such as Breakout. Figure 5.17 compares HIEROS using the standard-sized S5WM (in red) with HIEROS using a smaller S5WM, which consists of only 4 S5 blocks and is thus half the size (in blue). In the case of Battle Zone, a game characterized by complex dynamics, the larger S5WM significantly enhances performance. Conversely, for Breakout, the smaller S5WM performs better during certain stages of training but exhibits a decline towards the end. This suggests that the smaller S5WM could potentially outperform the larger counterpart in this environment, provided additional measures are implemented to prevent actor degeneration during later training stages.

Overall, this ablation confirms the explanation given in Section 5.1.2 that the poor performance of HIEROS is partly caused by the overparameterization of the used world model.

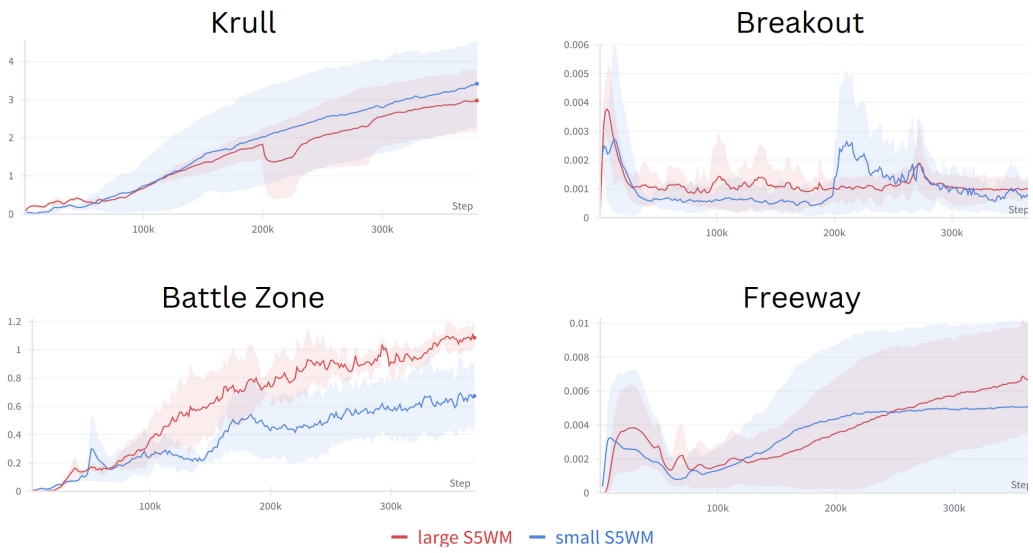

Figure 5.17: Comparison of the collected reward per step of HIEROS with the standard S5WM (red) and HIEROS using a smaller version of S5WM (blue) for Krull, Breakout, Battle Zone, and Freeway. The higher the collected reward, the better the agent performed.

5.3.8 Providing Decoded Subgoals as Actor Input

Section 4.3.1 describes that in HIEROS, the encoded subgoal g^i from the higher-level policy is passed as input to the lower-level input policy. This differs from Director Hafner, Lee, et al. (2022), where decoded subgoals from the subgoal autoencoder are passed to the worker policy in their hierarchical model. Figure 5.18 directly compares the impact of using encoded (in red) and decoded (in blue) subgoals as input for the lower-level subactor. Notably, utilizing encoded subgoals appears to enhance performance in Krull and Breakout but leads to a significant degradation in performance for Freeway. In practical terms, a tradeoff must be considered based on the specific environment. It's worth noting that passing decompressed subgoals increases the overall parameter count of the model, but in cases like Freeway, the additional model size contributes to further improvements in rewards.

Figure 5.19 shows the achieved subgoal rewards for the lowest level subactor in HIEROS when passing encoded subgoals as input to the actor-critic versus passing decoded subgoals to the actor-critic. As can be seen, there is no significant difference between the two approaches. This indicates, that using the decoded subgoals does not lead to a better ability for the subactors to complete the proposed subgoals compared to using encoded subgoals. Because using encoded subgoals leads to smaller model sizes, HIEROS uses this approach.

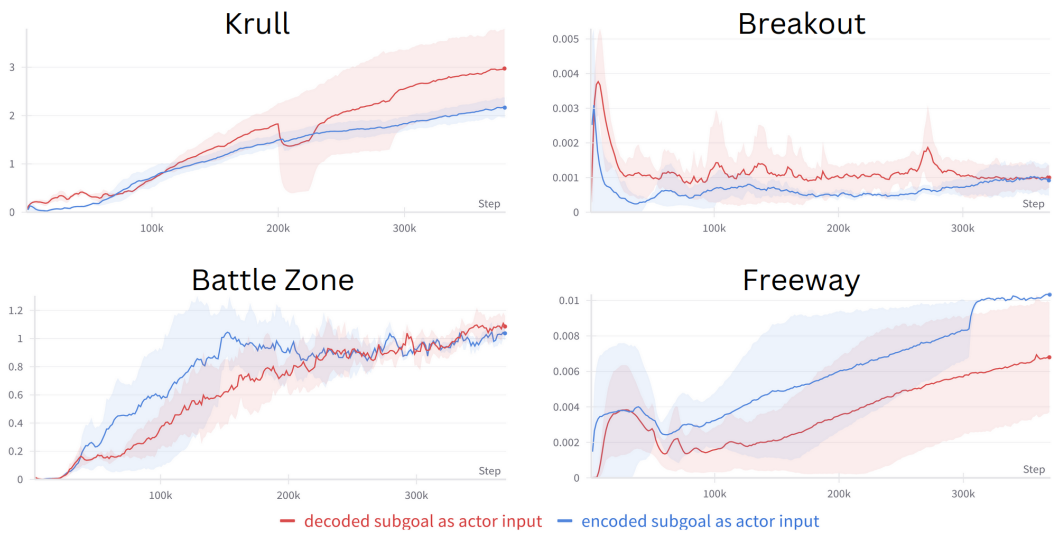

Figure 5.18: Comparison of the collected reward per step of HIEROS when passing encoded (red) and when passing subgoals decoded by the subgoal autoencoder (blue) to the lower level subactors for Krull, Breakout, Battle Zone, and Freeway.

5.3.9 Updating Subgoals During Imagination

As described in Section 4.4.3, during the imagination of trajectories, the subgoal is kept constant. This ablation experiment tests the impact of updating the subgoals during imagination. This, however, entails invoking the upper level subactors every k imagination steps (k being the update interval of the subactors) which decreases runtime efficiency significantly. On the other hand, updating the subgoals should provide a better learning signal to the subactor, as the subgoals sampled from the replay buffer were generated by an older version of the higher level subactor and might not be useful for learning the task at hand.

Figure 5.20 compares the two approaches of keeping the subgoal constant during imagination vs updating it. As can be seen, updating subgoals during imagination indeed improves the performance of HIEROS in Krull and Breakout, while being worse on Battle Zone and Freeway. Remarkably, for Breakout the improvement is significant, indicating that outdated subgoals sampled from the replay buffer might have a greater influence on the overall performance of the agent in environments with simple dynamics. This might be due to the optimal policy in Breakout is rather narrow, meaning that the agent has to be very precise when moving the platform to shoot the ball in the intended direction. An outdated subgoal might distract the agent and reduce precision, which leads to poorer overall performance.

It is also important to note, that updating the subgoals during imagination

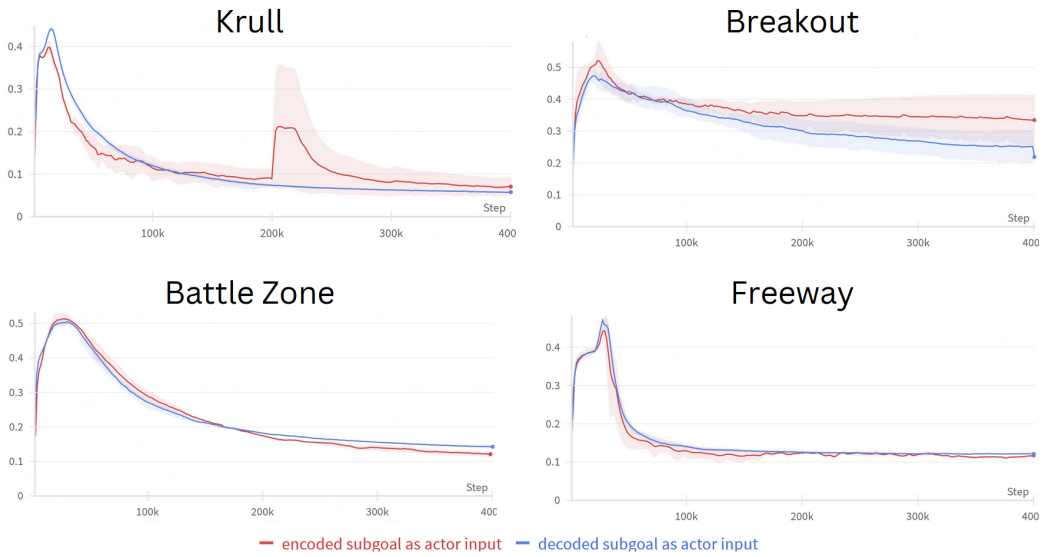

Figure 5.19: Comparison of the collected subgoal reward per step of HIEROS when passing encoded (red) and when passing subgoals decoded by the subgoal autoencoder (blue) to the lower level subactors for Krull, Breakout, Battle Zone, and Freeway. The higher the collected reward, the better the agent fulfilled the subgoals.

reduces runtime efficiency significantly, from roughly **7.73** frames per second without updating subgoals to **3.47** frames per second with the updating of subgoals. Since updating the subgoals only has a positive influence on Breakout and affected the performance in the three other games, HIEROS does not update its subgoals during imagination.

5.3.10 Conditioning only on Subgoal Rewards

As described in Section 4.3.2, HIEROS conditions all subactors on the extrinsic reward r_{extr}, a novelty reward r_{nov} and a subgoal reward r_{sg} (r_{sg} is not available to the highest level subactor). Director (Hafner, Lee, et al., 2022), however, conditions its worker policy only on the subgoal reward. This experiment compares HIEROS with all layers conditioned on all rewards to a version, where the extrinsic reward is only available to the highest level subactor, while the other subactors are only conditioned on r_{nov} and r_{sg}.

Figure 5.21 directly compares the two rewarding approaches. As can be seen, not supplying the lower level subactors with the extrinsic reward deteriorates the performance on Battle Zone and Freeway (for Freeway the achieved rewards are

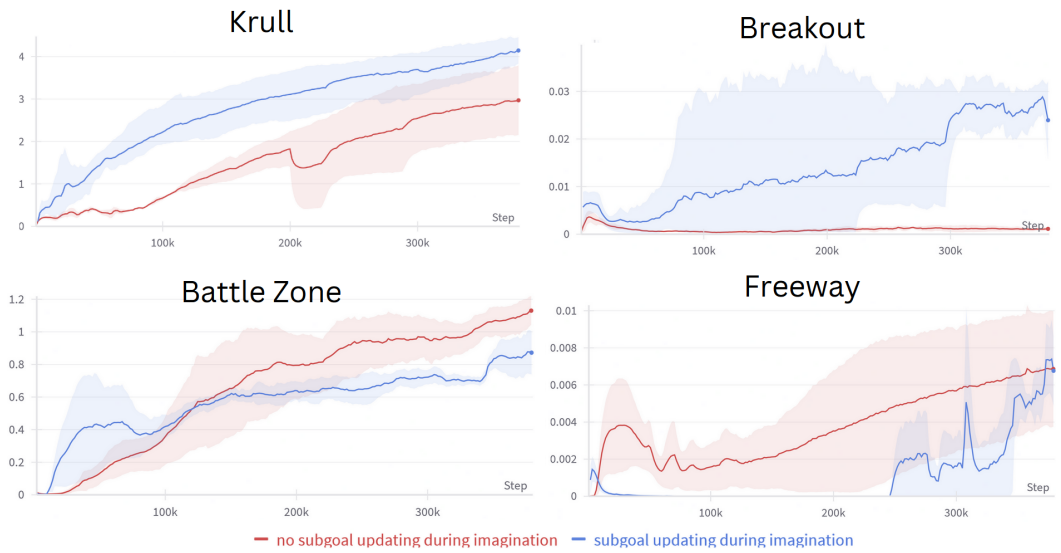

Figure 5.20: Comparison of the collected reward per step of HIEROS with updating the subgoals during imagination (blue) and HIEROS using the subgoals stored in the replay buffer (red) for Krull, Breakout, Battle Zone, and Freeway. The higher the collected reward, the better the agent performed.

close to zero). For Breakout there is no significant influence, while on Krull the performance is better with this modification. This indicates that in environments with complex dynamics, like Battle Zone and Freeway, the extrinsic reward at lower levels is crucial for guiding the subactors to make meaningful progress towards the overall goal. In contrast, in environments with simpler dynamics, like Breakout, the absence of extrinsic rewards for lower-level subactors does not significantly impact performance because as explained in Section 5.1.2, the events where extrinsic rewards are encountered are sparse and do not often occur. So the agent relies more on following subgoals and exploring novel states than on the extrinsic rewards.

In Krull, the improved performance without extrinsic rewards suggests that the environment's complexity benefits from subactors focusing on intrinsic motivation, possibly due to the diverse strategies available in different levels that do not strictly depend on immediate extrinsic rewards. This adaptability in Krull allows the agent to explore and exploit more varied and potentially more rewarding strategies. Overall, supplying each subactor with an extrinsic reward achieves higher rewards than without this change, which is why HIEROS gives each subactor access to the extrinsic rewards.

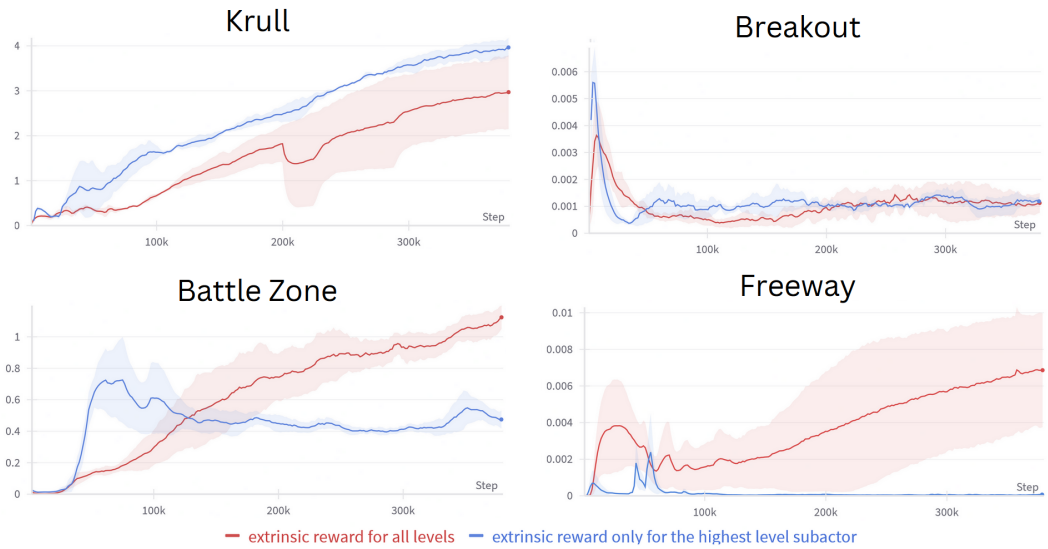

Figure 5.21: Comparison of the collected reward per step of HIEROS with conditioning only the highest level subactor on the extrinsic reward r_{extr} (blue) and HIEROS conditioning all subactors on the extrinsic reward r_{extr} (red) for Krull, Breakout, Battle Zone, and Freeway. The higher the collected reward, the better the agent performed.

5.3.11 Providing no Additional Inputs to the Actor-Critic

Section 4.3.2 shows how the actor-critic is supplied with additional inputs, such as the current reward, the continue signal and the entropy of the dynamics predictor. This was inspired by Robine et al. (2023) who found that supplying the actor with the predicted rewards has a positive influence on the overall performance of the agent. This ablation study examines the influence of these additional inputs by comparing HIEROS to a version, where the actor-critic only receives the current world state (h, z) as input.

Figure 5.22 compares the two approaches. As can be seen, without the additional inputs, the rewards are lower for all tasks except Breakout. This indicates, that the reward, continue signal and entropy of the dynamics predictor indeed can provide the actor-critic crucial information about how to learn a reward maximizing policy in the given environment. The difference in Breakout is not significant enough to rule out variance of the environment interactions as a factor for the better results.

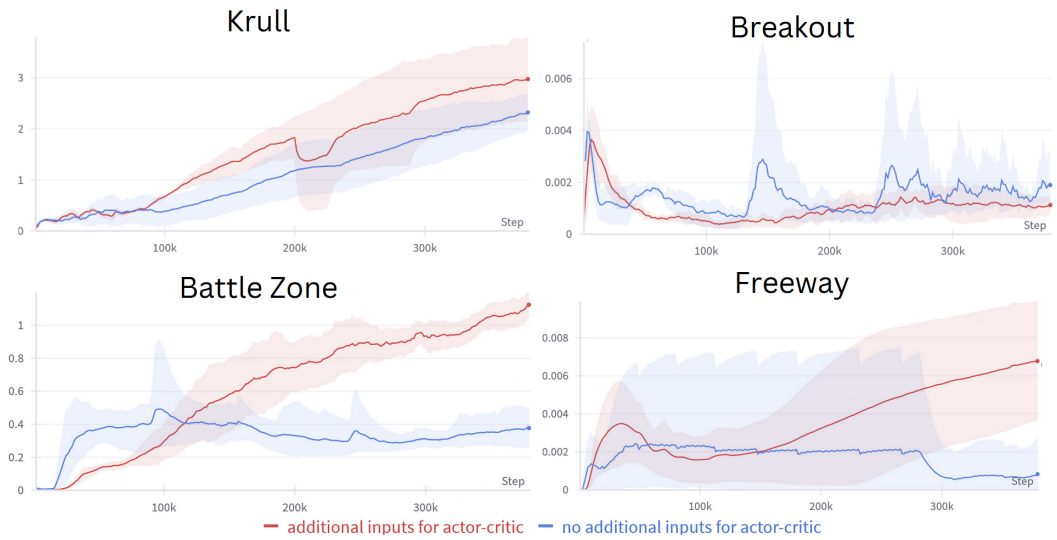

Figure 5.22: Comparison of the collected reward per step of HIEROS with passing additional inputs (reward, continue signal and entropy of the dynamics predictor) to the actor-critic (red) and HIEROS without these additional inputs (blue) for Krull, Breakout, Battle Zone, and Freeway. The higher the collected reward, the better the agent performed.

5.3.12 No Novelty Reward

Section 5.3.10 shows that conditioning all subactor layers on the extrinsic reward has a positive impact on the agent's performance. This ablation experiment tests the impact of the novelty reward on the overall agent's performance, by excluding r_{nov} from the reward term of the actor-critics of all subactors.

Figure 5.23 compares the performance of HIEROS with and without the novelty reward. For Krull and Breakout, the missing novelty reward does not have a significant impact on the overall performance. For Freeway, the missing novelty reward has a significant impact, as Freeway poses a complex exploration problem. Without guiding the agent towards novel states, it is not able to explore a winning strategy. This proves, that adding a novelty reward is an overall beneficial factor for the exploration capabilities of HIEROS.

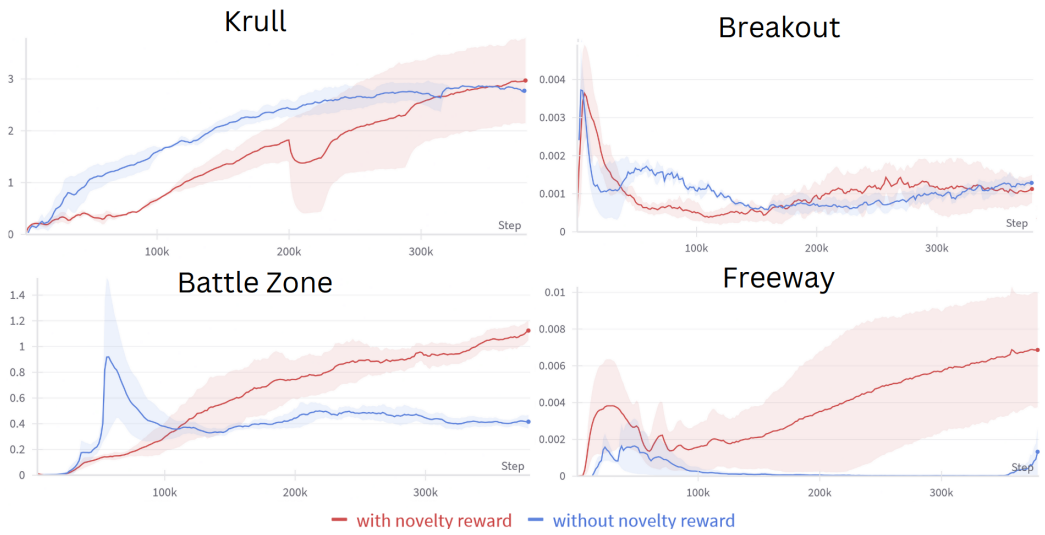

Figure 5.23: Comparison of the collected reward per step of HIEROS with using a novelty reward r_{nov} (red) and HIEROS without the novelty reward r_{nov} (blue) for Krull, Breakout, Battle Zone, and Freeway. The higher the collected reward, the better the agent performed.

6 Conclusion

This thesis presents and analyzes the HIEROS architecture, a multilayered goal conditioned hierarchical reinforcement learning (HRL) algorithm with hierarchical world models, an S5 layer-based world model (S5WM) and an efficient time-balanced sampling (ETBS) method which allows for a true uniform sampling over the experience dataset. The approach developed in this thesis represents the first HRL agent in the literature capable of training multiple layers of abstraction in a stable manner at state-of-the-art capabilities. HIEROS is evaluated on the Atari100k test suite (Bellemare et al., 2013) and achieves a new state of the art mean human normalized score for model-based RL agents without look-ahead search. Significant improvements have been achieved on tasks that require complex exploration and long term planning capabilities, while HIEROS achieves significantly worse results on tasks with simple dynamics and no distributional shifts compared to other approaches. HIEROS achieves competitive results on the DMC benchmark, showing that it currently cannot leverage its hierarchical structure effectively to solve the more complex control tasks of this benchmark. The results hint at some prospective changes that might boost the performance of HIEROS on this benchmark and make the agent usable in a wider range of RL environments. The option to decode proposed subgoals gives some explainability to the actions taken by HIEROS. A deeper evaluation on which subgoals are proposed and how the lower level workers are able to achieve these subgoals is left for future research.

The core findings of this thesis can be summarized as follows:

- Supplying each layer of a multilayered hierarchy of agents with a world model stabilizes training and makes efficient collaboration between subactors possible (see Section 4.3). This leads to an overall performance increase of the hierarchical actor (see Section 5.3.1) on tasks which require complex exploration or long term planning capabilities, while showing poorer performance on tasks with simpler dynamics.

- A world model which uses S5 layers for sequence prediction increases the runtime efficiency of training and interaction compared to other world model architectures that use GRU or Transformers as their sequence model (see Section 4.4 and Section 5.3.2). In direct comparison, the S5 world model outperforms the GRU world model (e.g. the RSSM used in DreamerV3) in

environments with multiple levels and shifting dynamics, while it performs slightly worse in environments with simple dynamics and no shift (see Section 5.1.1 and Section 5.2.1).

- Modifying the sampling probabilities of replay buffer entries so that the sampling follows a true uniform distribution over the entire training of the agent, increases the overall performance of the agent compared to the naive uniform sampling after each step over the entire buffer (see Section 5.3.3). This true uniform sampling distribution can be computed in closed form with a runtime complexity in $O(1)$ (see Section 4.5).

Besides those main takeaways, the ablation study shows further insights into what boosts and what weakens the performance of an HRL agent such as Hieros:

- A hierarchy of agents seems to be inherently worse at tasks with simple dynamics than agents that have only one layer. This is indicated by the similar profile of weaknesses and strengths when comparing Hieros with Director (Hafner, Lee, et al., 2022) in Section 5.1.1 and by the ablation study in Section 5.3.4 evaluating Hieros with different numbers of subactors.
- The internal state of the S5 sequence model contains more information on the current world state simulated by the world model than its direct output (see Section 5.3.5) which helps the agent achieve an overall higher performance.
- Providing the higher level policy with all steps of the lower level policy taken during the update interval k of the higher level is beneficial in most tasks for the performance of the HRL agent, compared to only providing every k-th step. This comes at the cost of increased model size (see Section 5.3.6).
- Using a compressed discrete subgoal as input for a lower level policy produces similar results for the overall task compared to providing a decompressed subgoal in the world model state space. The compressed subgoal also allows for smaller model sizes (see Section 5.3.8).
- Providing both a novelty and the extrinsic rewards to all subactors increases the performance of Hieros significantly in environments which require complex explorations (see Section 5.3.12 and Section 5.3.10).
- Giving the actor-critic access to the current reward, continue signal and entropy of the dynamics predictor increases the overall performance of the agent (see Section 5.3.11). Note that these inputs can only be made accessible

to an actor-critic that is trained with a world model, as this model can produce these inputs during environment interaction.

Regarding reproducibility of the results, all important architectural and training details are described in Chapter 4 and the used hyperparameters are shown in Section 4.6.3. The source code of Hieros is publicly available under the following link: https://github.com/Snagnar/Hieros. The material also gives an explanation on how to install and use the Atari100k benchmark for reproducing our results. The computational resources that were used for the experiments are described in Chapter 5.

Autonomous agents pose many ethical concerns, as they are able to act in the real world and can cause harm to humans. In this work, only simulated environments were used, so there should not be any potential for misuse of the contents of this. The proposed new world model architecture featuring S5 layers which could be used to train agents in imagination rather than in the real world. This could be leveraged to train agents for real-world applications, such as autonomous driving, without the need to train them in the real world, which could reduce the risk of harm to humans and the environment.

This thesis aims to provide a new perspective on the field of hierarchical reinforcement learning (HRL) and to inspire future research in this area.

7 Future Work

This section focuses on different avenues future research could take, in order to further understand, evaluate and improve Hieros. First, some directions to tackle the specific weaknesses observed during the evaluation in Chapter 5 are given in Section 7.1, then Section 7.2 lists several ideas inspired by current literature or the conducted experiments on how to generally improve the architecture, while it is not entirely clear, how exactly these changes might benefit Hieros. Finally, Section 7.3 lists different benchmarks and experiments that might be useful to further understand the strengths and weaknesses of the Hieros RL agent.

7.1 Possible Improvements for the Conducted Experiments

In Chapter 5 and Chapter 6 several weaknesses of Hieros in environments with simple dynamics and in continous control problems have been identified. The following paragraphs name different approaches on how to further increase the performance of Hieros in order to tackle the observed shortcomings of the approach.

As theorized in Section 5.1.1 and Section 5.2.1, possibly the outdated subgoals used to train the actor-critic during imagination might be one of the factors that deteriorate the performance on tasks with simpler dynamics but high requirement of precision of the actor network. This was supported by findings in Section 5.3.9. So testing the performance of Hieros with updating subgoals during imagination again on the DMC benchmark might yield higher results. Future work should also be directed towards finding a way to provide the actor-critic with up-to-date subgoals without the significant impact on the runtime efficiency, that the current mode of subgoal updating during imagination has. This could be achieved e.g. through model parallelism. The different subactors can be moved to different GPUs and run in parallel, which should reduce the execution overhead encountered in the experiments.

Another possibility to improve the performance, especially on the DMC benchmark, could be to increase the deterministic and stochastic state dimensions and the capacity of the subgoal state space. The high representation and dynamics losses shown in Figure 5.7 point towards a lack of expressiveness of the learned rep-

resentations, which should be improved by giving the world model more capacity to encode the state of the robot in its environment precisely.

7.2 General Improvements to Hieros

The following section describes some ways in which the current Hieros architecture might be improved in general. It is not clear, how exactly these improvements might affect the overal performance of the agent.

A possible way to improve performance could be to use the S5-based actor network proposed by Lu et al. (2024) for Hieros. Right now, the imagination procedure still relies on a single step prediction of the world model due to the single step architecture of the actor network. Using the S5-based actor network would allow for a multistep imagination procedure, which could further improve the performance of Hieros. This would also open the door to more efficient look-ahead search methods.

Since S4/S5-based models and Transformer-based models show complementary strengths in regard to short and long-term memory recall, many hybrid models have been proposed (Dao et al., 2022; Ankit Gupta et al., 2022; Mehta et al., 2022; Zuo et al., 2022). Exploring these more universal models might also be a promising direction for future research. Recently, a new sequence model architecture called Mamba (A. Gu and Dao, 2023) has been proposed, which builds upon S4 layers and achieves a new state of the art in a wide range of benchmarks. Testing Hieros with a Mamba world model might also be a promising approach.

In Section 3.2, multiple works have been presented that try to increase the performance of Dreamer like agents, such as e.g. STORM (W. Zhang et al., 2024) or the works of D'Oro et al. (2023). They introduce a wide range of minor changes to the initial DreamerV3 architecture. Testing out how these changes can be implemented for Hieros might also result in an agent with superior capabilities compared to the architecture presented in this thesis. E.g. increasing the training ratio (the number of trainings per environment interaction) and introducing regular parameter resets of the world model, actor-critic or subgoal autoencoder might help tackling challenges that are currently very challenging for Hieros to solve (e.g. the Breakout or Pong games of the Atari100k benchmark or the more complex control tasks of the DMC benchmark) (D'Oro et al., 2023).

LeCun (2022) describes a modular RL architecture, which combines HRL, world models, intrinsic motivation, and look-ahead planning in imagination as a potential candidate architecture for a true general intelligent agent. They propose learning a reconstruction free latent space to prevent a collapse of the learned representations,

which is already explored in several works for RL (Okada and Taniguchi, 2021; Schwarzer et al., 2021). They also describe two modes of environment interaction: reactive (Mode 1) and using look-ahead search (Mode 2). HIEROS implements several parts of this architecture, namely the hierarchical structure, hierarchical world models and the intrinsic motivation. HIEROS, like most RL approaches, uses the reactive mode, while approaches like EfficientZero (Ye et al., 2021) could be interpreted as Mode 2 actors. Koul et al. (2020) implement a Monte Carlo Tree Search (MCTS) planning method in the imagination of a Dreamer world model. Implementing similar methods for the hierarchical structure of HIEROS, combined with the more efficient S5-based world model (potentially also an S5 based actor network) could yield a highly efficient planning agent capable of learning complex behavior in very dynamic and stochastic environments. Different works try to combine the Dreamer architecture with look ahead search (Goodall and Belardinelli, 2023; Koul, 2022; Wang et al., 2023).

Since Figure 5.14 demonstrates that for some environments a deeper hierarchy can deteriorate performance, a possible future research could include an automatic scaling of the hierarchy depending on the current environment. E.g. if the accumulated reward stagnates and the agent cannot find a policy that further improves performance, the agent might automatically add another hierarchy layer, perform a model parameter reset and retrain on the collected experience dataset.

7.3 Future Experiments

All approaches listed above might increase the performance of HIEROS, but a thorough ablation study would be needed to fully examine the effect of each of these improvements. Also, the ablation study conducted in this thesis only looked at the effects of different single changes to the HIEROS architecture. Further experiments could focus on how different changes to the architecture interact with each other, e.g. what happens to the performance of the agent if two or more changes are removed at once. The deeper investigation of the effects of the different changes, however, remains for future research.

For now, HIEROS has been only evaluated on the Atari100k benchmark (Ye et al., 2021) and the DeepMind visual Control (DMC) suite (Tassa et al., 2018), but experiments on further benchmarks are necessary. BSuite (Osband et al., 2020) is a collection of low dimensional experiments directed at creating a detailed profile of an RL algorithm in several categories, e.g. long term memory, exploration or robustness against stochasticity. Also, Crafter (Hafner, 2021), a complex 2D environment with difficult exploration tasks, would give deeper insights in how

HIEROS' exploration capabilities compare with other state-of-the-art approaches. Hafner, Pasukonis, et al. (2023) evaluate their DreamerV3 model also on the game Minecraft (Kanitscheider et al., 2021) and show for the first time, that an agent can solve the challenge of finding a Diamond in the game completely without prior pretraining. As HIEROS surpasses DreamerV3 in exploration and long term planning capabilities, testing its performance on this benchmark might show the advantages of the HIEROS architecture in greater detail. For now, these further evaluations are left to future work. A thorough comparison between S5WM and the S4WM proposed by F. Deng et al. (2024) remains for future research as well.

Bibliography

Agarwal, Alekh, Nan Jiang, Sham M Kakade, and Wen Sun (2019). **Reinforcement learning: Theory and algorithms**. *CS Dept., UW Seattle, Seattle, WA, USA, Tech. Rep* 32, 96 (see page 15).

Agarwal, Pranav, Sheldon Andrews, and Samira Ebrahimi Kahou (2024). **Learning to Play Atari in a World of Tokens**. *arXiv preprint arXiv:2406.01361* (see page 39).

Agarwal, Rishabh, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare (2021). **Deep reinforcement learning at the edge of the statistical precipice**. *Advances in Neural Information Processing Systems* 34, 29304–29320 (see page 76).

Ansel, Jason, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala (2024). **PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation**. In: *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM. DOI: 10.1145/3620665.3640366 (see page 46).

Anyoha, Rockwell (2017). *The History of Artificial Intelligence*. https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/. [Accessed 21-05-2024] (see page 1).

Aubret, Arthur, Laetitia Matignon, and Salima Hassas (2023). **An Information-Theoretic Perspective on Intrinsic Motivation in Reinforcement Learning: A Survey**. *Entropy* 25:2, 327. ISSN: 1099-4300. DOI: 10.3390/e25020327 (see page 4).

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E Hinton (2016). **Layer normalization**. *arXiv preprint arXiv:1607.06450* (see page 60).

Bellemare, Marc G, Yavar Naddaf, Joel Veness, and Michael Bowling (2013). **The arcade learning environment: An evaluation platform for general agents**. *Journal of Artificial Intelligence Research* 47, 253–279 (see pages 3, 6, 38, 73, 76, 105).

Bellman, Richard (1966). **Dynamic programming**. *science* 153:3731, 34–37 (see pages 11–13).

Bengio, Emmanuel, Joelle Pineau, and Doina Precup (2020). **Interference and generalization in temporal difference learning**. In: *International Conference on Machine Learning*. PMLR, 767–777 (see page 43).

Bengio, Yoshua, Aaron Courville, and Pascal Vincent (2013). **Representation learning: A review and new perspectives**. *IEEE transactions on pattern analysis and machine intelligence* 35:8, 1798–1828 (see page 21).

Brigham, E. O. and R. E. Morrow (1967). **The fast Fourier transform**. *IEEE Spectrum* 4:12, 63–70 (see page 30).

C2D/s5-pytorch (2023). *Pytorch implementation of Simplified Structured State-Spaces for Sequence Modeling (S5)*. URL: https://github.com/i404788/s5-pytorch (visited on 09/28/2023) (see page 46).

Çalışır, Sinan and Meltem Kurt Pehlivanoğlu (2019). **Model-free reinforcement learning algorithms: A survey**. In: *2019 27th signal processing and communications applications conference (SIU)*. IEEE, 1–4 (see page 13).

Uc-Cetina, Victor, Nicolás Navarro-Guerrero, Anabel Martin-Gonzalez, Cornelius Weber, and Stefan Wermter (2023). **Survey on reinforcement learning for language processing**. *Artificial Intelligence Review* 56:2, 1543–1575 (see page 11).

Chai, Junyi, Hao Zeng, Anming Li, and Eric WT Ngai (2021). **Deep learning in computer vision: A critical review of emerging techniques and application scenarios**. *Machine Learning with Applications* 6, 100134 (see page 1).

Chen, Chang, Yi-Fu Wu, Jaesik Yoon, and Sungjin Ahn (2022). **Transdreamer: Reinforcement learning with transformer world models**. *arXiv preprint arXiv:2202.09481* (see pages 6, 29, 31, 37, 38, 58, 59, 95).

Cho, Kyunghyun, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio (2014). **On the Properties of Neural Machine Translation: Encoder–Decoder Approaches**. In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Association for Computational Linguistics, 103–111. DOI: 10.3115/v1/W14-4012 (see pages 27, 28).

Christodoulou, Petros (2019). **Soft actor-critic for discrete action settings**. *arXiv preprint arXiv:1910.07207* (see page 19).

Coulom, Rémi (2006). **Efficient selectivity and backup operators in Monte-Carlo tree search**. In: *International conference on computers and games*. Springer, 72–83 (see page 36).

D'Oro, Pierluca, Max Schwarzer, Evgenii Nikishin, Pierre-Luc Bacon, Marc G Bellemare, and Aaron Courville (2023). **Sample-Efficient Reinforcement Learning by Breaking the Replay Ratio Barrier**. In: *The Eleventh International Conference on Learning Representations* (see pages 6, 94, 110).

danijar/dreamerv3 (2024). *danijar/dreamerv3*. URL: https://github.com/danijar/dreamerv3 (visited on 06/02/2024) (see page 46).

Dao, Tri, Daniel Y Fu, Khaled K Saab, Armin W Thomas, Atri Rudra, and Christopher Ré (2022). **Hungry hungry hippos: Towards language modeling with state space models**. *arXiv preprint arXiv:2212.14052* (see pages 80, 110).
David, FN and NL Johnson (1948). **The probability integral transformation when parameters are estimated from the sample**. *Biometrika* 35:1/2, 182–190 (see page 63).
David, Shmuel Bar, Itamar Zimerman, Eliya Nachmani, and Lior Wolf (2022). **Decision s4: Efficient sequence-based rl via state spaces layers**. In: *The Eleventh International Conference on Learning Representations* (see page 29).
Dayan, Peter and Geoffrey E Hinton (1992). **Feudal reinforcement learning**. *Advances in Neural Information Processing Systems* 5 (see pages 3, 24).
Deng, Fei, Junyeong Park, and Sungjin Ahn (2024). **Facing off world model backbones: Rnns, transformers, and S4**. *Advances in Neural Information Processing Systems* 36 (see pages 3, 6, 31, 38, 58–60, 74, 75, 81, 95, 112).
Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei (2009). **Imagenet: A large-scale hierarchical image database**. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255 (see page 1).
Dzhivelikian, Evgenii, Artem Latyshev, Petr Kuderov, and Aleksandr I Panov (2022). **Hierarchical intrinsically motivated agent planning behavior with dreaming in grid environments**. *Brain Informatics* 9:1, 8 (see page 35).
Elfwing, Stefan, Eiji Uchibe, and Kenji Doya (2018). **Sigmoid-weighted linear units for neural network function approximation in reinforcement learning**. *Neural networks* 107, 3–11 (see page 60).
Fedus, William, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney (2020). **Revisiting fundamentals of experience replay**. In: *International Conference on Machine Learning*. PMLR, 3061–3071 (see pages 6, 45).
Florensa, Carlos, Yan Duan, and Pieter Abbeel (2017). **Stochastic neural networks for hierarchical reinforcement learning**. *arXiv preprint arXiv:1704.03012* (see page 24).
Fredriksson, Teodor, David Issa Mattos, Jan Bosch, and Helena Holmström Olsson (2020). **Data labeling: An empirical investigation into industrial challenges and mitigation strategies**. In: *International Conference on Product-Focused Software Process Improvement*. Springer, 202–216 (see page 1).
Friston, Karl, Rosalyn J Moran, Yukie Nagai, Tadahiro Taniguchi, Hiroaki Gomi, and Josh Tenenbaum (2021). **World model learning and inference**. *Neural Networks* 144, 573–590 (see page 20).
Goodall, Alexander W and Francesco Belardinelli (2023). **Approximate shielding of atari agents for safe exploration**. *arXiv preprint arXiv:2304.11104* (see page 111).
google/jax (2024). URL: https://github.com/google/jax (visited on 06/02/2024) (see page 46).
Grondman, Ivo, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska (2012). **A survey of actor-critic reinforcement learning: Standard and natural policy gradients**.

IEEE Transactions on Systems, Man, and Cybernetics, part C (applications and reviews) 42:6, 1291–1307 (see page 18).

Gu, Albert and Tri Dao (2023). **Mamba: Linear-time sequence modeling with selective state spaces**. *arXiv preprint arXiv:2312.00752* (see pages 32, 110).

Gu, Albert, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré (2020). **Hippo: Recurrent memory with optimal polynomial projections**. *Advances in neural information processing systems* 33, 1474–1487 (see page 31).

Gu, Albert, Karan Goel, and Christopher Ré (2021). **Efficiently modeling long sequences with structured state spaces**. *arXiv preprint arXiv:2111.00396* (see pages 3, 30, 31, 44).

Guan, Yanchen, Haicheng Liao, Zhenning Li, Jia Hu, Runze Yuan, Yunjian Li, Guohui Zhang, and Chengzhong Xu (2024). **World models for autonomous driving: An initial survey**. *IEEE Transactions on Intelligent Vehicles* (see page 20).

Gupta, Ankit, Harsh Mehta, and Jonathan Berant (2022). **Simplifying and Understanding State Space Models with Diagonal Linear RNNs**. *arXiv preprint arXiv:2212.00768* (see page 110).

Ha, David and Jürgen Schmidhuber (2018). **World models**. *arXiv preprint arXiv:1803.10122* (see pages 3, 19, 20, 37).

Haarnoja, Tuomas, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine (2018). **Latent space policies for hierarchical reinforcement learning**. In: *International Conference on Machine Learning*. PMLR, 1851–1860 (see pages 4, 36).

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine (2018). **Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor**. In: *International conference on machine learning*. PMLR, 1861–1870 (see pages 18, 81).

Haarnoja, Tuomas, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. (2018). **Soft actor-critic algorithms and applications**. *arXiv preprint arXiv:1812.05905* (see page 19).

Hafner, Danijar (2021). **Benchmarking the Spectrum of Agent Capabilities**. *arXiv preprint arXiv:2109.06780* (see page 111).

Hafner, Danijar, Kuang-Huei Lee, Ian Fischer, and Pieter Abbeel (2022). **Deep hierarchical planning from pixels**. *Advances in Neural Information Processing Systems* 35, 26091–26104 (see pages 4, 25, 26, 35, 36, 38, 39, 41, 44, 47, 49–52, 74, 90, 96, 98, 100, 106).

Hafner, Danijar, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi (2019). **Dream to control: Learning behaviors by latent imagination**. *arXiv preprint arXiv:1912.01603* (see pages 2, 3, 20, 21, 27, 37, 42, 45, 74).

Hafner, Danijar, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson (2019). **Learning latent dynamics for planning from pixels**. In: *International conference on machine learning*. PMLR, 2555–2565 (see pages 3, 21, 37, 38).

Hafner, Danijar, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba (2020). **Mastering atari with discrete world models**. *arXiv preprint arXiv:2010.02193* (see pages 3, 4, 21, 22, 25, 28, 35, 37, 38, 56, 90).

Hafner, Danijar, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap (2023). **Mastering diverse domains through world models**. *arXiv preprint arXiv:2301.04104* (see pages 3, 5, 6, 22–24, 28, 37–39, 41, 44, 46, 54, 56, 57, 62, 64, 71, 73, 74, 81, 90, 112).

Hart, Peter E, Nils J Nilsson, and Bertram Raphael (1968). **A formal basis for the heuristic determination of minimum cost paths**. *IEEE transactions on Systems Science and Cybernetics* 4:2, 100–107 (see page 36).

Hasani, Ramin, Mathias Lechner, Tsun-Hsuan Wang, Makram Chahine, Alexander Amini, and Daniela Rus (2022). **Liquid structural state-space models**. *arXiv preprint arXiv:2209.12951* (see page 32).

Hendrycks, Dan and Kevin Gimpel (2016). **Gaussian error linear units (gelus)**. *arXiv preprint arXiv:1606.08415* (see page 60).

Hoang, Dinh Thai, Nguyen Van Huynh, Diep N Nguyen, Ekram Hossain, and Dusit Niyato (2023). **Markov Decision Process and Reinforcement Learning** (see page 10).

Hochreiter, Sepp and Jürgen Schmidhuber (1997). **Long short-term memory**. *Neural computation* 9:8, 1735–1780 (see page 27).

Hui, David Yu-Tung, Aaron C Courville, and Pierre-Luc Bacon (2024). **Double gumbel q-learning**. *Advances in Neural Information Processing Systems* 36 (see page 73).

Hutsebaut-Buysse, Matthias, Kevin Mets, and Steven Latré (2022). **Hierarchical reinforcement learning: A survey and open research challenges**. *Machine Learning and Knowledge Extraction* 4:1, 172–221 (see pages 4, 24, 36, 37, 43).

Ioffe, Sergey and Christian Szegedy (2015). **Batch normalization: Accelerating deep network training by reducing internal covariate shift**. In: *International conference on machine learning*. pmlr, 448–456 (see page 60).

Jang, Beakcheol, Myeonghwi Kim, Gaspard Harerimana, and Jong Wook Kim (2019). **Q-learning algorithms: A comprehensive classification and applications**. *IEEE access* 7, 133653–133667 (see page 13).

Jiang, Yiding, Shixiang Shane Gu, Kevin P Murphy, and Chelsea Finn (2019). **Language as an abstraction for hierarchical deep reinforcement learning**. *Advances in Neural Information Processing Systems* 32 (see pages 4, 25, 35, 47).

Jordan, IM (1992). **Rumelhart,"Supervised learning with a distal teacher,"** *Cognitive Science* 16, 307–354 (see page 3).

Junker, Brian W and Klaas Sijtsma (2001). **Cognitive assessment models with few assumptions, and connections with nonparametric item response theory**. *Applied Psychological Measurement* 25:3, 258–272 (see page 14).

Kaiser, Lukasz, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. (2019). **Model-based reinforcement learning for atari**. *arXiv preprint arXiv:1903.00374* (see pages 3, 74).

Kanitscheider, Ingmar, Joost Huizinga, David Farhi, William Hebgen Guss, Brandon Houghton, Raul Sampedro, Peter Zhokhov, Bowen Baker, Adrien Ecoffet, Jie Tang, et al. (2021).

Multi-task curriculum learning in a complex, visual, hard-exploration domain: Minecraft. *arXiv preprint arXiv:2106.14876* (see pages 3, 112).
Khetarpal, Khimya, Matthew Riemer, Irina Rish, and Doina Precup (2022). **Towards continual reinforcement learning: A review and perspectives**. *Journal of Artificial Intelligence Research* 75, 1401–1476 (see page 43).
Kingma, Diederik P, Max Welling, et al. (2019). **An introduction to variational autoencoders**. *Foundations and Trends® in Machine Learning* 12:4, 307–392 (see page 50).
Kingma, Durk P, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling (2016). **Improved variational inference with inverse autoregressive flow**. *Advances in Neural Information Processing Systems* 29 (see pages 22, 23, 57).
Kocsis, Levente and Csaba Szepesvári (2006). **Bandit based monte-carlo planning**. In: *European conference on machine learning*. Springer, 282–293 (see page 36).
Koçyiğit, Mustafa Taha, Timothy M Hospedales, and Hakan Bilen (2023). **Accelerating self-supervised learning via efficient training strategies**. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 5654–5664 (see page 1).
Konda, Vijay and John Tsitsiklis (1999). **Actor-critic algorithms**. *Advances in neural information processing systems* 12 (see page 18).
Koul, Anurag (2022). **Investigating Latent State and Uncertainty Representations in Reinforcement Learning** (see page 111).
Koul, Anurag, Varun V Kumar, Alan Fern, and Somdeb Majumdar (2020). **Dream and search to control: Latent space planning for continuous control**. *arXiv preprint arXiv:2010.09832* (see page 111).
Krizhevsky, Alex, Geoffrey Hinton, et al. (2009). **Learning multiple layers of features from tiny images** (see page 1).
Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). **Imagenet classification with deep convolutional neural networks**. *Advances in neural information processing systems* 25 (see page 1).
Kujanpää, Kalle, Joni Pajarinen, and Alexander Ilin (2023). **Hierarchical imitation learning with vector quantized models**. In: *International Conference on Machine Learning*. PMLR, 17896–17919 (see pages 24, 36).
Kullback, Solomon and Richard A Leibler (1951). **On information and sufficiency**. *The annals of mathematical statistics* 22:1, 79–86 (see page 23).
Kumar, Sanjay (2018). **Convolution, Product and Correlation Theorems for Simplified Fractional Fourier Transform: A Mathematical Investigation**. *arXiv preprint arXiv:1803.07381* (see page 30).
Laskin, Michael, Aravind Srinivas, and Pieter Abbeel (2020). **Curl: Contrastive unsupervised representations for reinforcement learning**. In: *International conference on machine learning*. PMLR, 5639–5650 (see page 81).
LeCun, Yann (2022). **A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27**. *Open Review* 62 (see pages 4, 35, 42, 43, 110).

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). **Deep learning**. *nature* 521:7553, 436–444 (see page 9).

LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). **Gradient-based learning applied to document recognition**. *Proceedings of the IEEE* 86:11, 2278–2324 (see page 1).

Levy, Andrew, George Konidaris, Robert Platt, and Kate Saenko (2017). **Learning multi-level hierarchies with hindsight**. *arXiv preprint arXiv:1712.00948* (see page 36).

Li, Jinning, Chen Tang, Masayoshi Tomizuka, and Wei Zhan (2022). **Hierarchical planning through goal-conditioned offline reinforcement learning**. *IEEE Robotics and Automation Letters* 7:4, 10216–10223 (see page 24).

Lin, Tianyang, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu (2022). **A survey of transformers**. *AI open* 3, 111–132 (see page 29).

lindermanlab/S5 (2022). *Implementation of Simplified State Space Layers for Sequence Modeling*. URL: https://github.com/lindermanlab/S5 (visited on 06/02/2024) (see page 46).

Lu, Chris, Yannick Schroecker, Albert Gu, Emilio Parisotto, Jakob Foerster, Satinder Singh, and Feryal Behbahani (2024). **Structured state space models for in-context reinforcement learning**. *Advances in Neural Information Processing Systems* 36 (see pages 3, 32, 39, 46, 59, 81, 110).

Mehta, Harsh, Ankit Gupta, Ashok Cutkosky, and Behnam Neyshabur (2022). **Long range language modeling via gated state spaces**. *arXiv preprint arXiv:2206.13947* (see pages 80, 110).

Merrill, William, Jackson Petty, and Ashish Sabharwal (2024). **The illusion of state in state-space models**. *arXiv preprint arXiv:2404.08819* (see page 32).

Micheli, Vincent, Eloi Alonso, and François Fleuret (2022). **Transformers are sample-efficient world models**. *arXiv preprint arXiv:2209.00588* (see pages 2, 6, 20, 31, 37, 45, 59, 73, 74, 76, 78, 89, 90, 95).

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller (2013). **Playing atari with deep reinforcement learning**. *arXiv preprint arXiv:1312.5602* (see pages 16, 45).

Moerland, Thomas M, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. (2023). **Model-based reinforcement learning: A survey**. *Foundations and Trends® in Machine Learning* 16:1, 1–118 (see page 2).

Mondal, Amit Kumar and N Jamali (2020). **A survey of reinforcement learning techniques: strategies, recent development, and future directions**. *arXiv preprint arXiv:2001.06921* (see page 45).

Nachum, Ofir, Michael Ahn, Hugo Ponte, Shixiang Gu, and Vikash Kumar (2019). **Multi-agent manipulation via locomotion using hierarchical sim2real**. *arXiv preprint arXiv:1908.05224* (see pages 4, 24, 25, 35).

Nachum, Ofir, Shixiang Gu, Honglak Lee, and Sergey Levine (2018). **Near-optimal representation learning for hierarchical reinforcement learning**. *arXiv preprint arXiv:1810.01257* (see page 24).

Nachum, Ofir, Shixiang Shane Gu, Honglak Lee, and Sergey Levine (2018). **Data-efficient hierarchical reinforcement learning**. *Advances in neural information processing systems* 31 (see pages 4, 24, 25, 35, 47).

Nachum, Ofir, Haoran Tang, Xingyu Lu, Shixiang Gu, Honglak Lee, and Sergey Levine (2019). **Why does hierarchy (sometimes) work so well in reinforcement learning?** *arXiv preprint arXiv:1909.10618* (see pages 4, 35).

Nair, Suraj and Chelsea Finn (2019). **Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation**. *arXiv preprint arXiv:1909.05829* (see pages 24, 47).

Narayanan, Deepak, Keshav Santhanam, Peter Henderson, Rishi Bommasani, Tony Lee, and Percy S Liang (2023). **Cheaply estimating inference efficiency metrics for autoregressive transformer models**. *Advances in Neural Information Processing Systems* 36, 66518–66538 (see page 31).

Nguyen, Derrick and Bernard Widrow (1990). **The truck backer-upper: An example of self-learning in neural networks**, 11–19 (see page 3).

NM512 (2023). *dreamerv3-torch*. URL: https://github.com/NM512/dreamerv3-torch (visited on 09/28/2023) (see page 46).

Nozawa, Kento and Issei Sato (2022). **Empirical evaluation and theoretical analysis for representation learning: A survey**. *arXiv preprint arXiv:2204.08226* (see page 21).

Okada, Masashi and Tadahiro Taniguchi (2021). **Dreaming: Model-based reinforcement learning by latent imagination without reconstruction**. In: *2021 ieee international conference on robotics and automation (icra)*. IEEE, 4209–4215 (see page 111).

Okudo, Takato and Seiji Yamada (2021). **Subgoal-based reward shaping to improve efficiency in reinforcement learning**. *IEEE Access* 9, 97557–97568 (see page 25).

Orseau, Laurent and Levi HS Lelis (2021). **Policy-guided heuristic search with guarantees**. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 14, 12382–12390 (see page 36).

Osband, Ian, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepesvári, Satinder Singh, Benjamin Van Roy, Richard Sutton, David Silver, and Hado van Hasselt (2020). **Behaviour Suite for Reinforcement Learning**. In: *International Conference on Learning Representations* (see page 111).

Ouyang, Long, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. (2022). **Training language models to follow instructions with human feedback**. *Advances in neural information processing systems* 35, 27730–27744 (see page 11).

Parr, Ronald and Stuart Russell (1997). **Reinforcement learning with hierarchies of machines**. *Advances in Neural Information Processing Systems* 10 (see pages 3, 24).

Paul, Sujoy, Jeroen Vanbaar, and Amit Roy-Chowdhury (2019). **Learning from trajectories via subgoal discovery**. *Advances in Neural Information Processing Systems* 32 (see page 25).

Pierson, Harry A and Michael S Gashler (2017). **Deep learning in robotics: a review of recent research**. *Advanced Robotics* 31:16, 821–835 (see page 1).

Poli, Michael, Stefano Massaroli, Eric Nguyen, Daniel Y Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré (2023). **Hyena hierarchy: Towards larger convolutional language models**. In: *International Conference on Machine Learning*. PMLR, 28043–28078 (see page 32).

Polydoros, Athanasios S and Lazaros Nalpantidis (2017). **Survey of model-based reinforcement learning: Applications on robotics**. *Journal of Intelligent & Robotic Systems* 86:2, 153–173 (see page 11).

Rani, Veenu, Syed Tufael Nabi, Munish Kumar, Ajay Mittal, and Krishan Kumar (2023). **Self-supervised learning: A succinct review**. *Archives of Computational Methods in Engineering* 30:4, 2761–2775 (see page 1).

Reza, Selim, Marta Campos Ferreira, José JM Machado, and João Manuel RS Tavares (2022). **A multi-head attention-based transformer model for traffic flow forecasting with a comparative analysis to recurrent neural networks**. *Expert Systems with Applications* 202, 117275 (see page 29).

Robine, Jan, Marc Höftmann, Tobias Uelwer, and Stefan Harmeling (2023). **Transformer-based world models are happy with 100k interactions**. *arXiv preprint arXiv:2303.07109* (see pages 6, 38, 39, 45, 46, 53, 62, 73, 74, 90, 93–95, 102).

Rosete-Beas, Erick, Oier Mees, Gabriel Kalweit, Joschka Boedecker, and Wolfram Burgard (2023). **Latent Plans for Task-Agnostic Offb02ine Reinforcement Learning**. en. In: *Conference on Robot Learning*. PMLR, 1838–1849 (see page 25).

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). **Learning Internal Representations by Error Propagation, Parallel Distributed Processing, Explorations in the Microstructure of Cognition, ed. DE Rumelhart and J. McClelland. Vol. 1. 1986**. *Biometrika* 71, 599–607 (see page 27).

Schmidhuber, Jürgen (1990). **An on-line algorithm for dynamic reinforcement learning and planning in reactive environments**. In: *1990 IJCNN international joint conference on neural networks*. IEEE, 253–258 (see page 3).

Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz (2015). **Trust region policy optimization**. In: *International conference on machine learning*. PMLR, 1889–1897 (see page 17).

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). **Proximal policy optimization algorithms**. *arXiv preprint arXiv:1707.06347* (see pages 17, 74).

Schwarzer, Max, Nitarshan Rajkumar, Michael Noukhovitch, Ankesh Anand, Laurent Charlin, R Devon Hjelm, Philip Bachman, and Aaron C Courville (2021). **Pretraining representations for data-efficient reinforcement learning**. *Advances in Neural Information Processing Systems* 34, 12686–12699 (see page 111).

Sherstinsky, Alex (2020). **Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network**. *Physica D: Nonlinear Phenomena* 404, 132306 (see page 1).

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. (2017). **Mastering chess and shogi by self-play with a general reinforcement learning algorithm**. *arXiv preprint arXiv:1712.01815* (see page 11).

Smith, Jimmy, Shalini De Mello, Jan Kautz, Scott Linderman, and Wonmin Byeon (2024). **Convolutional state space models for long-range spatiotemporal modeling**. *Advances in Neural Information Processing Systems* 36 (see page 32).

Smith, Jimmy TH, Andrew Warrington, and Scott W Linderman (2022). **Simplified state space layers for sequence modeling**. *arXiv preprint arXiv:2208.04933* (see pages 5, 31, 32, 39, 41, 46, 60).

Sutton, Richard S (1991). **Dyna, an integrated architecture for learning, planning, and reacting**. *ACM Sigart Bulletin* 2:4, 160–163 (see page 20).

Sutton, Richard S and Andrew G Barto (2018). **Reinforcement learning: An introduction**. MIT press (see pages 2, 13, 14).

Sutton, Richard S, Doina Precup, and Satinder Singh (1999). **Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning**. *Artificial Intelligence* 112:1-2, 181–211 (see pages 3, 24).

Tassa, Yuval, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. (2018). **Deepmind control suite**. *arXiv preprint arXiv:1801.00690* (see pages 6, 73, 88, 111).

Tessler, Chen, Shahar Givony, Tom Zahavy, Daniel Mankowitz, and Shie Mannor (2017). **A deep hierarchical approach to lifelong learning in minecraft**. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 1 (see page 24).

Torabi, Faraz, Garrett Warnell, and Peter Stone (2018). **Behavioral cloning from observation**. *arXiv preprint arXiv:1805.01954* (see page 2).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). **Attention is all you need**. *Advances in neural information processing systems* 30 (see pages 1, 29).

Wang, Hanqing, Wei Liang, Luc Van Gool, and Wenguan Wang (2023). **Dreamwalker: Mental planning for continuous vision-language navigation**. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 10873–10883 (see page 111).

Watkins, Christopher JCH and Peter Dayan (1992). **Q-learning**. *Machine learning* 8, 279–292 (see page 11).

Williams, Ronald J (1992). **Simple statistical gradient-following algorithms for connectionist reinforcement learning**. *Machine learning* 8, 229–256 (see pages 14, 53).

Yampolskiy, Roman V (2018). **Artificial intelligence safety and security**. CRC Press (see page 2).

Yarats, Denis, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto (2021). **Mastering visual continuous control: Improved data-augmented reinforcement learning**. *arXiv preprint arXiv:2107.09645* (see page 81).

Ye, Weirui, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao (2021). **Mastering atari games with limited data**. *Advances in Neural Information Processing Systems* 34, 25476–25488 (see pages 74, 111).

Zhang, Daokun, Jie Yin, Xingquan Zhu, and Chengqi Zhang (2018). **Network representation learning: A survey**. *IEEE transactions on Big Data* 6:1, 3–28 (see page 21).

Zhang, Junzi, Jongho Kim, Brendan O'Donoghue, and Stephen Boyd (2021). **Sample efficient reinforcement learning with REINFORCE**. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 35. 12, 10887–10895 (see page 15).

Zhang, Tengteng and Hongwei Mo (2021). **Reinforcement learning for robot research: A comprehensive review and open issues**. *International Journal of Advanced Robotic Systems* 18:3, 17298814211007305 (see page 11).

Zhang, Weipu, Gang Wang, Jian Sun, Yetian Yuan, and Gao Huang (2024). **STORM: Efficient stochastic transformer based world models for reinforcement learning**. *Advances in Neural Information Processing Systems* 36 (see pages 39, 110).

Zheng, Yue (2019). **Reinforcement learning and video games**. *arXiv preprint arXiv:1909.04751* (see page 11).

Zuo, Simiao, Xiaodong Liu, Jian Jiao, Denis Charles, Eren Manavoglu, Tuo Zhao, and Jianfeng Gao (2022). **Efficient long sequence modeling via state space augmented transformer**. *arXiv preprint arXiv:2212.08136* (see pages 80, 110).

Declaration of Authorship

I hereby declare that this thesis is my own unaided work. All direct or indirect sources used are acknowledged as references.

Potsdam, July 14, 2024

Paul Mattes