

Distillation of the DreamerV3 World Model for Efficient Deployment

Yichen Ning, Vladimir Bortsov, Ernesto Garcia Cano, Minh Nhat Pham, Niels Saes*

Eindhoven University of Technology
Eindhoven, Netherlands

Abstract

In the context of model-based reinforcement learning, the distillation of world models seeks to transfer the capabilities of large, comprehensive environment models ("teachers") into more compact and efficient alternatives ("students"). While large models can capture complex dynamics across diverse tasks, their deployment often requires significant computational resources and can become infeasible when constrained by limited hardware. In this paper, we distill the world model of the general-purpose algorithm *DreamerV3* and test its performance across the Atari games *Air Raid* and *Ms. Pac-Man*, used as benchmark tasks. The parameter count, performance, retention ratio, and action match of the distilled models are compared to their respective teachers. Students with up to 20× fewer encoder parameters retain at least 95% of the teacher's score in both games. Compressing the RSSM reduces the deployed model by 84.3% and lowers inference latency by 2.91×, while still retaining 94.8% of the teacher's mean return on *Air Raid*. After an adaptation stage, students combining both distilled components outperform an undistilled *DreamerV3* model of comparable size by up to 84%.

1 Introduction

Recent progress and research on deep reinforcement learning (RL) techniques has led to substantial developments in model-based reinforcement learning (MBRL) algorithms [23] – that is, algorithms which learn a *world model* by exploring a given environment, in order to optimize a policy to effectively interact with it. In 2019, Hafner et al. introduced *Dreamer*, an MBRL algorithm designed to learn a model of its environment from visual inputs which then unrolls latent state trajectories to train a policy, a process denoted by the authors as *latent imagination* [8]. The third version of the algorithm, known as *DreamerV3*, is broadly applicable, data-efficient, and stable across several benchmark tasks, such as Atari, DeepMind Control Suite, ProcGen, and even diamond collection in Minecraft [9]. *DreamerV3* was shown to outperform specialized state-of-the-art alternatives across over 150 diverse tasks [9].

Multiple studies have been conducted to apply RL in robotics [17, 30, 33, 38], Internet of things [3], or communication and networking [24]. The common point between these applications is that they often consist of embedded systems with limited hardware resources and a requirement for low energy consumption. This constraint restricts the deployment of the *DreamerV3* algorithm, despite the ample advantages brought forward by its world model. With the default of 200M parameters used for the model by Hafner et al. [9], and the necessity of the encoder and recurrent state-space model to be executed at every control step alongside the actor network, memory and latency costs could exceed what many embedded systems can afford [34].

This difficulty of deployment issue persists among similar large world models, and solutions focused on *sim-to-real transfer* (that is, training an algorithm within a simulated environment with the purpose of deploying it to a real-world machine) have been proposed in related work, such as [20, 35, 37]. Among these, the *TWIST* (Teacher-Student World model Distillation for Sim-to-Real Transfer) method proposed in [37] and benchmarked on *DreamerV2* presents an objective most similar to our own, though we focus our distillation on the encoder and Recurrent State-Space Model (RSSM) components and do not distill imagined rollouts. Therefore, we adapt the *TWIST* methodology to distill the *DreamerV3* world model, and compare against the teacher's performance to justify our work.

Our experimental setup focuses on three cases: (1) distilling only the encoder, (2) only the RSSM, and (3) combining the encoder and RSSM distillations. We obtain favorable results for the student models' performance, with the caveat that these do not always generalize across all training seeds, and identify meaningful limitations of our distillation method.

2 Related Work

Our work lies at the intersection of three lines of research. We first review model-based RL algorithms and compare their inference cost at deployment. We then survey how prior work has deployed RL on constrained hardware, where distillation is a common approach. Finally, we discuss existing world model distillation methods and the gaps they leave for *DreamerV3*.

2.1 Model-Based Reinforcement Learning

Model-based reinforcement learning (MBRL) agents learn a predictive model of their environment and use it to simulate, plan and optimize actions, improving sample efficiency over model-free methods. Notable approaches include Probabilistic Ensembles with Trajectory Sampling (PETS) [5], Model-Based Policy Optimization (MBPO) [15], the *Dreamer* family [8, 9] and the Temporal Difference Learning for Model Predictive Control (TD-MPC) family [10, 11]. TD-MPC combines MBRL with model predictive control and shows particular promise in continuous control. However, like *MuZero* [29], it plans at decision time, querying its world model many times for every action. *Dreamer*, following *Dyna* [31], instead uses its model only during training to imagine trajectories for the actor and critic. At deployment it runs the encoder, one RSSM [7] update and the actor per step, so its inference cost is dominated by the encoder and RSSM, the components we distill. The *Dreamer* algorithm is therefore a good candidate for reinforcement learning in constrained hardware, as its per-step inference cost is a single forward pass through the encoder, the recurrent state update and the actor. In contrast, TD-MPC instead must repeatedly roll out its

*All authors contributed equally to this research.

world model to plan each action, so its compute and latency grow with the number of sampled trajectories.

2.2 RL on constrained hardware

RL, especially Deep RL, demands vast memory and computing power [21], which embedded systems cannot offer. A number of researchers have tried to tackle this obstacle from the hardware aspect [30], while others have attempted to overcome the challenge by means of software implementation. Among them, knowledge distillation appears to be one of the most popular methods to compress a deep reinforcement learning agent [14, 28].

Jang et al. [14] conducted one of the earliest studies on model compression for edge computing systems, where they proposed the idea of training a student model on an edge device by distilling the policies from a pre-trained cloud network. Sharing the same concept of policy distillation, Xu et al. [36] suggested that instead of using a predefined architecture, the student agent can be constructed by continuously pruning the least important filters from the teacher’s policy network. At the beginning of the training process, the students would still be relatively large, comparable to the size of their teacher, resulting in a higher demand for computational power at this stage. Although they did not evaluate whether the students would fit on a system with less hardware resources, their experiments showed that with a high pruning rate, the compressed models were still able to achieve high average reward.

While the listed studies have proven that it is feasible to deploy deep RL agents on embedded hardware via knowledge distillation, their scope has been limited to the compression of Deep Q-Networks. Our research targeted the DreamerV3 agents, which are computationally heavier. We aimed to investigate how much the Dreamer algorithm could be effectively compressed using different distillation techniques. Since our work focused on the distillation approaches rather than finding the optimal size of the student, we made a prior decision on the number of parameters of the student and kept it fixed for all experiments.

2.3 Distillation Approaches

Knowledge distillation [12] is widely used to compress deep RL agents, most commonly for policy distillation [28]. Fewer works distill the world model itself, even though for agents such as DreamerV3, the encoder and the RSSM dominate the inference cost.

The world model distillation methods that do exist are mostly motivated by deployment limitations. The main point of comparison for this paper is the TWIST approach [37], where a trained teacher model supervises a student that sees domain-randomized images by matching latent states, enabling sim-to-real transfer on DreamerV2. With a different focus, Khanzada & Kwon [18] train a DreamerV3-style teacher with a dense, privileged reward, and distill its latent dynamics into a student trained on sparse rewards. Kuzmenko and Shvai propose TD-MPC-Opt [20], which focuses more on the practical limitations of model deployment by distilling a 317M-parameter multi-task TD-MPC2 agent into a 1M-parameter model on the MT30 benchmark. Wang et al. introduce WestWorld [35], another general-purpose world model which they then distill for deployment on a real-world Unitree Go1, a bionic quadruped robot.

And Fan et al. [6] distill the world model of the *dream evaluator* of their DreamAvoid framework to reach interactive latency.

While these approaches show successful examples of world model distillation, they are either: (1) working on world model architectures which are fundamentally different from those in the Dreamer algorithms (WestWorld, DreamAvoid); (2) focused on distilling other components of the model, considering the differing goals and benchmarks; or (3) distilling into student models which are not sufficiently small for tackling hardware limitations for deployment. From these considerations, we motivate the research for our DreamerV3 distillation approach.

3 Preliminaries

3.1 Model-Based Reinforcement Learning

3.1.1 Partially Observable Environments. We consider an agent interacting with a partially observable Markov decision process (POMDP) [16], defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{X}, T, R, O, \gamma)$. At each time step t the environment is in a hidden state $s_t \in \mathcal{S}$, the agent receives an observation $x_t \in \mathcal{X}$, $x_t \sim O(x_t | s_t)$, takes an action $a_t \in \mathcal{A}$, and the environment transitions to $s_{t+1} \sim T(s_{t+1} | s_t, a_t)$ while emitting a reward $r_t = R(s_t, a_t)$. Since x_t does not fully determine s_t (a single Atari frame, for instance, does not reveal object velocities), the agent must act on its history of observations and actions. Its goal is to learn a policy that maximizes the expected discounted return $\mathbb{E}[\sum_t \gamma^t r_t]$, with discount factor $\gamma \in [0, 1)$.

3.1.2 World Model Learning. Model-free methods learn a policy or value function directly from environment interaction [32]. Model-based methods instead learn a model of the environment, a *world model*, from a replay buffer $\mathcal{D}$ containing past environment interaction trajectories τ , and use it to improve the policy [26, 32]. For high-dimensional observations such as images, predicting future observations directly is expensive. Latent world models [7] therefore encode the history into a compact latent state and learn the dynamics, reward r_t and episode continuation flag c_t in that latent space. A learned model can be used in two ways. Decision-time planning [32] searches the model at every step to select an action, for example with tree search or model predictive control (MPC). Background planning, as in Dyna [31], uses the model to generate imagined trajectories of latent states, rewards and continuation flags, rolled out from replayed states, on which a policy and value function are trained. Dreamer follows the latter approach. This means that, once trained, the agent acts with a single forward pass of its encoder, recurrent state update and actor, while the remainder of the world model is needed only during training.

3.2 The Dreamer Algorithm

This section focuses on the third generation of the Dreamer algorithm proposed by Hafner et al. [9]. The algorithm operates on three neural networks: the world model predicts the outcome of potential actions, the critic judges the value of each outcome, and the actor chooses actions to reach the most valuable outcomes. The world model will be our distillation target later on. Note that in this paper, we will use a PyTorch implementation of Dreamer [27], from which we extend our distillation pipeline.

At deployment, only the encoder, sequence model and actor are executed at each step (blue in Figure 7, Appendix C); the decoder, reward, continue and dynamics predictors and the critic are used only during training. In the algorithm, the components are trained concurrently from replayed experience of the agent interacting with the environment. The following section describes the inner workings of the three models.

3.2.1 World Model. The world model is implemented as a Recurrent State-Space Model (RSSM) [7] which follows auto-encoding principles [19]. An encoder maps sensory inputs x_t to stochastic representations z_t . Then, a sequence model with recurrent state h_t predicts the sequence of these representations given past actions a_{t-1} . Here, h_t is a deterministic state that summarizes the history of past representations and actions, whereas z_t is a stochastic variable that captures the information in the current input x_t and the uncertainty about it. The concatenation of h_t and z_t forms the model state from which rewards r_t and episode continuation flags $c_t \in \{0, 1\}$ are predicted. Then, a decoder reconstructs the inputs x_t from the model state to ensure informative representations:

$$\text{RSSM} \left\{ \begin{array}{ll} \text{Sequence model:} & h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1}) \\ \text{Encoder:} & z_t \sim q_\phi(z_t | h_t, x_t) \\ \text{Dynamics predictor:} & \hat{z}_t \sim p_\phi(\hat{z}_t | h_t) \\ \text{Reward predictor:} & \hat{r}_t \sim p_\phi(\hat{r}_t | h_t, z_t) \\ \text{Continue predictor:} & \hat{c}_t \sim p_\phi(\hat{c}_t | h_t, z_t) \\ \text{Decoder:} & \hat{x}_t \sim p_\phi(\hat{x}_t | h_t, z_t) \end{array} \right.$$

The dynamics, reward and continue predictors are MLPs. Depending on the input data, the algorithm uses different input-transformers for the model. Given that our input will be image-based (Atari-game screencaptures), the input-transformer to be used is a convolutional neural network (CNN) [9]. The CNN output e_t is concatenated with h_t and mapped by an MLP to the parameters of z_t . Together, the CNN and this MLP form the encoder q_ϕ shown in Figure 7 in Appendix C. The representations are sampled from a vector of softmax distributions. Since this sampling step is not differentiable, straight-through gradients [2] are used: the forward pass uses the discrete sample, while the backward pass treats the sampling as the identity and passes gradients through the softmax probabilities. The world model is trained with reconstruction, reward and continuation losses together with KL terms between the posterior and the prior [9]; for distillation purposes, we will use a pretrained teacher and keep its components frozen throughout.

3.2.2 Actor & Critic. In DreamerV3, the actor and critic are trained on trajectories imagined by the world model [9]. The critic estimates expected future returns, while the actor $\pi_\theta(a_t | s_t)$, with $s_t = (h_t, z_t)$, outputs a categorical distribution over the discrete Atari actions.

3.3 Knowledge Distillation Methods

Knowledge distillation transfers capabilities of a large teacher model to a small student model, by training the student to reproduce the teacher’s outputs. Formally, we aim to minimize the Kullback-Leibler (KL) divergence between the teacher and the student’s temperature softened output distributions [12]:

$$\mathcal{L}_{\text{KL}} = \kappa^2 \text{KL}(\sigma(\ell^T/\kappa) \parallel \sigma(\ell^S/\kappa)), \quad (1)$$

where σ is the softmax function, κ is the temperature, ℓ is the model output logits, and the $(\cdot)^T$ and $(\cdot)^S$ superscript respectively denote any output logits by the teacher and the student model. Throughout this paper, we set the temperature $\kappa = 1$. The student can also regress on the teacher’s internal representations directly with a distance loss [1]:

$$\mathcal{L}_{\text{MSE}} = \|y^S - y^T\|_2^2, \quad (2)$$

where y denotes the matched representation. [12] shows that, at high temperatures, Equation 1 is equivalent to minimizing Equation 2. In RL, same principles are applied to transfer policies from a teacher to a compact student.

In the case of DreamerV3, the CNN output e_t and deterministic recurrent state h_t are representations, and can be matched using equation 2, while the stochastic representation z_t , the actor π_θ , and the dynamics predictor $p_\phi(\hat{z}_t | h_t)$, are probability distributions, and can be matched using equation 1.

4 The DreamerV3 Distillation Process

As established in the introduction, our approach adapts the TWIST methodology [37] to distill the DreamerV3 world model. Figure 7 in Appendix C shows that, at deployment, only the encoder, one recurrent update and the actor are executed, so these are the components whose cost we reduce. We distill the encoder and RSSM separately since they differ in structure and cost, and then combine the independently distilled components. Figure 8 (Appendix C) gives a visual overview of both distillation settings and their loss terms. Various loss functions are chosen with the objective of matching each part of the teacher at a level where the student can reproduce its behavior.

4.1 Distilling for a Smaller Encoder:

The CNN encoder involves repeatedly sweeping a large set of shared weights through an input image, which can be computationally intensive at deployment. Therefore, we want to distill the encoder part of the world model, while keeping the teacher’s frozen RSSM and actor. A top-down view of where the loss signals for distillation could come from is shown in Figure 8 (see Appendix C). The main goal of distillation is to force the students to play just like the teacher over game trajectories, meaning that the agent should take the same actions given the same observations. Then naturally, the two terms we want to minimize would be the difference between policy distributions:

$$\mathcal{L}_\pi = \mathbb{E}_{\tau \sim \mathcal{D}} \left[\sum_t D_{\text{KL}}(\pi_\theta(a_t | s_t^T) \parallel \pi_\theta(a_t | s_t^S)) \right], \quad s_t = (h_t, z_t); \quad (3)$$

and the difference in the history value output of the teacher RSSM, computed respectively, from the teacher and students previous (h_t, z_t, a_t) tuples:

$$\mathcal{L}_h = \mathbb{E}_{\tau \sim \mathcal{D}} \left[\sum_t \|h_{t+1}^T - h_{t+1}^S\|_2^2 \right], \quad h_{t+1} = f_\phi(h_t, z_t, a_t). \quad (4)$$

Note that any network used to compute a representation with the T superscript has their weights frozen.

Following this, we form our primary loss signal using these two loss terms, weighted using GradNorm [4] to ensure both losses facilitate learning at the same rate:

$$\mathcal{L}_{\text{primary}}(\tau) = w_\pi \mathcal{L}_\pi + w_h \mathcal{L}_h. \quad (5)$$

The encoder’s last CNN layer outputs, e_t , and the encoder’s posterior, z_t , do not have to be an exact match between the student and the teacher, since (1) minimizing the loss in equation 5 is the main objective, and (2) we want to allow the inherently smaller student architecture some freedom to learn in flexible ways that may achieve good results with different intermediary representations and distributions. However, it may still help speed up the training if we use the difference in these two terms as a loss signal to guide the student in the right direction. Thus, we have the following loss terms:

$$\mathcal{L}_e(x_t) = \sum_t \text{MSE}(\mathbf{e}_t^T, \mathbf{e}_t^S), \quad \mathbf{e}_t^T = \text{enc}_\phi(x_t), \quad \mathbf{e}_t^S = \text{enc}_\psi(x_t); \quad (6)$$

$$\mathcal{L}_z(x_t) = D_{\text{KL}}\left(q_\phi(z_t^T | h_t^T, \mathbf{e}_t^T) \parallel q_\phi(z_t^S | h_t^T, \mathbf{e}_t^S)\right). \quad (7)$$

Note that in Equation 7, to better guide the student, we use teacher’s history state h_t^T to compute student’s latent z_t^S . Furthermore, since the computation of e_t and z_t do not depend on past observations, we can use any single frame x_t instead of a trajectory τ . Then these two loss terms form our secondary loss signal:

$$\mathcal{L}_{\text{secondary}}(x_t) = w_e \mathcal{L}_e + w_z \mathcal{L}_z. \quad (8)$$

We do not want the secondary loss to disrupt the minimization of the primary loss. Therefore, we first train only using the secondary loss, then switch to training with only the primary loss.

4.2 Distilling for a Smaller RSSM

Unlike encoder distillation, this setting retains the teacher’s frozen CNN encoder and compresses the recurrent state-space model (RSSM). The goal is to preserve the teacher’s latent predictions and action selection while reducing recurrent inference cost.

4.2.1 Student Architecture and Latent Distillation. Both models process the same observations and recorded actions using shared embeddings $e_t = \text{enc}_\phi(x_t)$ (bottom row of Figure 8, Appendix C). Each maintains its own recurrent state $h_t = f(h_{t-1}, z_{t-1}, a_{t-1})$. We write $q_t = q(z_t | h_t, e_t)$ for the observation-conditioned posterior and $p_t = p(z_t | h_t)$ for the prior. We minimize the teacher-to-student KL divergence introduced in Eq. (1), with temperature $\kappa = 1$:

$$\mathcal{L}_{\text{KD}} = \mathbb{E}_{\tau \sim \mathcal{D}} \left[D_{\text{KL}}(q_t^T \parallel q_t^S) + D_{\text{KL}}(p_t^T \parallel p_t^S) \right]. \quad (9)$$

Each KL term sums over the categorical variables. The student follows its own recurrent history; thus, these losses match distributions along recorded trajectories but do not guarantee accurate autonomous rollouts. We do not align the differently sized deterministic states directly.

4.2.2 Policy and Prediction Supervision. Because the compressed state cannot be fed into the teacher’s actor, we train a separate student actor on $s_t^S = (h_t^S, z_t^S)$. Following the policy-distillation principle in Eq. (1), its soft action-imitation loss is

$$\mathcal{L}_{\text{action}} = -\mathbb{E}_{\tau \sim \mathcal{D}} \left[\sum_{a \in \mathcal{A}} \pi_T(a | s_t^T) \log \pi_S(a | s_t^S) \right]. \quad (10)$$

This cross-entropy has the same student gradients as policy KL, since the teacher entropy is constant. The student actor, reward predictor and continuation predictor each have two hidden layers of 256 units. Reward and continuation losses use recorded environment outcomes rather than teacher predictions; their heads support world-model evaluation but are not needed for action selection.

4.2.3 Two-Stage Optimization. We first optimize $\mathcal{L}_{\text{KD}}$ alone, then jointly train the RSSM and compact heads using

$$\mathcal{L}_{\text{joint}} = \mathcal{L}_{\text{KD}} + \mathcal{L}_{\text{reward}} + \mathcal{L}_{\text{cont}} + 10\mathcal{L}_{\text{action}}. \quad (11)$$

The evaluated model uses 5000 KL-pretraining and 3000 joint-training update attempts; mixed-precision overflow can skip optimizer updates. Both stages use existing replay data, with no new environment interaction or actor-critic training. We do not use shared-history supervision or multi-step KL training. At deployment, the student uses the frozen encoder, compressed RSSM, and compact actor.

Additional details on the development of the RSSM distillation procedure, including intermediate configurations and training adjustments, are provided in Appendix A.

4.3 Combining the Distilled Encoder and RSSM

We combine the independently distilled components into one agent. Since separately trained latent representations may be incompatible, we compare three merging strategies: direct merging, encoder-only adaptation and joint fine-tuning. The latter two are each evaluated with fixed loss weights and with GradNorm, giving five configurations in total.

4.3.1 Direct Merge. We connect the distilled encoder to the compressed RSSM and student actor without further training, establishing a baseline.

4.3.2 Encoder-Only Adaptation. We freeze the compressed RSSM and actor and train only the encoder. The reference pathway uses the teacher encoder with the *same* frozen compressed RSSM and actor. The policy KL and the embedding MSE have the forms already introduced in Eqs. (3) and (6), but use teacher’s encoder output as references to adapt the alignment of classes from student’s Encoder to student’s RSSM. For posterior KL, we use one different from that in Equation 7, where the student encoder uses the student history state h_t^S and we compute the KL over full trajectories:

$$\mathcal{L}_z(\tau) = \mathbb{E}_{\tau \sim \mathcal{D}} \left[D_{\text{KL}}\left(q_\phi(z_t | h_t^T, e_t^T) \parallel q_\phi(z_t | h_t^S, e_t^S)\right) \right]. \quad (12)$$

Then our fixed-weight objective is:

$$\mathcal{L}_{\text{adapt}}(\tau) = \mathcal{L}_\pi + 0.1\mathcal{L}_z + 0.05\mathcal{L}_e. \quad (13)$$

Alternatively, we use the GradNorm-weighted policy and hidden-state objective of Eq. (5). Here $\mathcal{L}_h$ compares recurrent states obtained using the two encoder embeddings through the same frozen compressed RSSM, so both states have equal dimension.

4.3.3 Joint Fine-Tuning. Starting from the direct merge of Section 4.3.1, we jointly optimize the encoder, RSSM, and actor against the full teacher. We reuse the embedding, posterior, prior, hidden-state, and policy losses defined above, normalizing each by its fixed

training calibration mean:

$$\mathcal{L}_{\text{FT}}(\tau) = w_e \tilde{\mathcal{L}}_e + w_z \tilde{\mathcal{L}}_z + w_{\text{prior}} \tilde{\mathcal{L}}_{\text{prior}} + w_h \tilde{\mathcal{L}}_h + w_\pi \tilde{\mathcal{L}}_\pi. \quad (14)$$

For hidden-state alignment, a temporary linear projection maps the student’s 2048-dimensional state to the teacher’s 8192-dimensional state. We compare fixed weights (1, 1, 1, 1, 10), ordered as in Eq. (14), against GradNorm, which adapts the weights of all five terms during training; both variants use the same starting point. The projection is discarded after training, so deployment architecture remains unchanged.

5 Experiments

Our experiments investigate the following questions:

- Q1** How much of the teacher’s control performance does a distilled student retain?
- Q2** How much do distilled students reduce parameter count and inference latency?
- Q3** Do distilled students outperform undistilled DreamerV3 models of comparable size?

To study how well the teachers and the students could handle different levels of complexity, we trained 2 Atari games, *Air Raid* and *Ms. Pac-Man*, with 3 seeds for 100k steps each. *Air Raid* only requires the player to move horizontally and evade the enemies’ attacks, while *Ms. Pac-Man* has to travel all over the map to eat all the pellets, running from or chasing after the ghost during the process.

5.1 Experimental Setup

Table 1: Encoder architectures (convolutional channels per layer), parameter counts, and reduction relative to the encoder of the 200M-parameter teacher. *sep* denotes depthwise separable convolutions.

Encoder	Conv. channels	Params	Reduction
Teacher	[128, 192, 256, 256]	3.49M	–
L	[64, 128, 192, 256]	740k	$\times 4.7$
S	[32, 64, 128, 256]	389k	$\times 9.0$
XS	[16, 32, 64, 256]	172k	$\times 20$
S, sep	[32, 64, 128, 256]	47k	$\times 74$
XS, sep	[16, 32, 64, 256]	21k	$\times 166$

5.1.1 Setup for distillation of a smaller encoder. We set our teacher model as the same 200M parameter model used to train the atari100k from the DreamerV3 paper [9]. For the CNN encoder, the teacher uses a 5×5 kernel with stride 1, then down-samples with a max pooling layer. To save on compute, the student uses a 3×3 kernel with stride 2, skipping the down-sampling. In Table 1, we list the encoder architecture of the teacher and the many students in terms of channel sizes of CNN layers: [channel1, channel2, channel3, channel4]. A student denoted *sep* uses a depthwise separable CNN [13] to further reduce parameter counts. We run two separate experiments where we always train the student for 5000 steps with

the primary loss in Eq. 5, but also train the student beforehand, with 5000 or 0 warm up steps with the secondary loss in Eq. 8. We perform all trainings with a learning rate of 3×10^{-4} and the AdamW optimizer [22]. To answer **Q1** and evaluate students at deployment, we compute the score retention ratio and action match for each encoder size. We quantify these by:

$$\text{retention ratio} = \frac{\text{student game score}}{\text{teacher game score}}, \quad (15)$$

and

$$\text{action match} = \frac{1}{N} \sum_{t=1}^N \mathbb{1}[a_t^S = a_t^T], \quad (16)$$

where N is the number of evaluated time steps, a_t^S and a_t^T are the actions selected by the student and teacher policies given the same observation, and $\mathbb{1}[\cdot]$ is the indicator function. Both metrics are averaged over 100 additional gameplay episodes per Atari game.

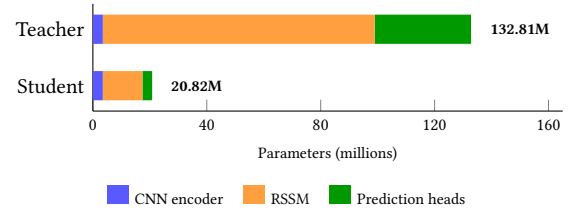


Figure 1: Parameter distribution in the teacher and student models. The CNN encoder remains unchanged. Distillation reduces the parameter count from 132.81M to 20.82M, achieving an 84.3% reduction (6.38 $\times$ compression).

5.1.2 Setup for Distillation of a Smaller RSSM. We use the procedure of Section 4.2, termed Latent Dynamics Decomposition (LDD), which compresses the RSSM by separately matching the teacher’s posterior and prior latent distributions via KL divergence and its action policy via soft imitation. We retain the teacher’s frozen CNN encoder and reduce the deterministic recurrent-state dimension from 8192 to 2048, using a student internal hidden width of 512. The stochastic state retains 32 categorical variables with 64 classes each. The student uses separate actor, reward and continuation heads, each with two hidden layers of 256 units. Figure 1 reports the resulting parameter counts.

Training uses the teacher’s existing *Air Raid* and *Ms. Pac-Man* replay buffer, with a temporal split allocating 90% to training. Each batch contains two sequences with 64 burn-in steps followed by 32 learning steps. The learning rate is 10^{-4} , with zero weight decay, gradient clipping at 100 and mixed-precision training.

We first train the student RSSM using prior and posterior KL losses for 5000 update attempts. We then reset the optimizer and jointly train the RSSM and compact heads for another 3000 attempts, retaining both KL losses and adding reward, continuation, and soft action-imitation losses. The action-imitation loss has weight 10; the remaining terms have unit weight. These counts denote update attempts, since mixed-precision overflow handling may skip optimizer updates. To assess the robustness and generalizability of the approach, we train and evaluate *Air Raid* and *Ms. Pac-Man* on 3 different seeds (Section 5.3.4).

5.2 Results of Encoder Distillation

5.2.1 Summarized results. The results for *Air Raid* and *Ms. Pac-Man* are summarized in Figure 2, where we present the retention ratio and action match for different encoder sizes, averaged over the 3 seeds. We report exact result figures in Appendix B, Table 4. As a measure of how well the students reproduce the teachers’ latent states z_t , we additionally report the mean posterior KL and the mean difference in the history state h_t for both games in Appendix B, Table 5. We also report the inference time of all encoders in Appendix B, Figure 6. We observe that the performance degrades roughly linearly on the given logarithmic scale as the student encoder parameter shrinks.

5.2.2 Results of standard convolution (Q1). Students with standard convolutions retain most of the teacher’s performance with both retention ratio and action match above 90%. We observe that, even though the L student imitates the teacher better by having a higher action match and a lower posterior KL, the S student outperformed the L student on mean retention ratio in 10 out of the 12 runs.

5.2.3 Results of depthwise separable convolution (Q2). Students with depthwise separable convolutions have a further parameter count reduction of four to eight times (Table 1), but the mean student performance also suffers. This also introduced a large variance to retention ratio, action match, posterior KL, and history state differences. We also observe in Figure 6 that the *sep* students, despite having much less parameters, increased the latency. This may be due to the fact that *sep* convolutions are less parallelizable on a GPU than standard convolutions [25].

5.2.4 Effect of pre-training with secondary loss. Pre-training 5000 steps with secondary loss improves the mean in retention ratio for all *Air Raid* students and in 4 out of the 5 students in *Ms. Pac-Man*. It also lowers both the posterior KL and the history state difference in 9 out of the 10 student and game combinations. So we may regard the positive effect of pre-training with the secondary loss a consistent trend. Though do note that it’s effect on action match is minimal.

5.3 Results of RSSM Distillation

We first evaluate the student trained with seed 0 in terms of control performance, compression, inference efficiency, and latent fidelity. We then assess robustness across three training seeds on *Air Raid* and *Ms. Pac-Man* in Section 5.3.4.

5.3.1 Control Performance (Q1). We evaluated the seed-0 student and its teacher on 100 matched *Air Raid* evaluation seeds. As shown in Table 2, the staged student achieved a mean return of 4131.75, retaining 94.8% of the teacher’s 4358.75, measured using raw episode returns.

The approximate 95% bootstrap intervals were [3838.74, 4425.00] for the student and [4075.99, 4637.00] for the teacher. The paired mean difference was -227.00 (95% CI: $[-569.50, 113.01]$), with the student outperforming the teacher on 41 of 100 matched seeds and tying on one seed. The interval includes zero but does not establish equivalence or non-inferiority.

5.3.2 Compression and Inference Efficiency (Q2). Figure 1 shows that RSSM compression reduces parameters from 95.48M to 13.92M

Table 2: Mean *Air Raid* returns over 100 matched validation seeds (6000–6099). Retention is relative to the teacher’s mean return.

Model	Mean Return	Retention
Teacher	4358.75	100.0%
LDD: joint training	4131.75	94.8%
LDD: teacher-width heads	1245.00	28.6%
LDD: compact heads	1243.75	28.5%
LDD: enlarged heads, $\lambda_\pi = 10$	1174.75	27.0%

(6.86 $\times$). The counted model package decreases from 132.81M to 20.82M parameters, an 84.3% reduction (6.38 $\times$). This count includes the encoder, RSSM, actor, reward and continuation heads, excluding training-only components such as the decoder and critic.

As shown in Appendix B, Figure 6, the student achieves a 2.91 $\times$ inference speedup under the evaluated hardware and execution conditions.

5.3.3 Latent Distribution Fidelity. Results (see Table 3, Appendix B) show that joint adaptation preserves comparable latent fidelity: prior KL changes from 2.99 to 2.75, posterior KL from 2.73 to 2.89, and posterior agreement remains at 93.1%. Further details are provided in Appendix A.

5.3.4 Performance Across Training Seeds. To assess robustness, we evaluate teacher–student pairs across three training seeds on *Air Raid* and *Ms. Pac-Man*. Each model is evaluated on 100 matched episodes using previously unseen environment seeds (6000–6099), with no truncated episodes. Figure 3 shows episode returns and paired differences with 95% confidence intervals.

On *Air Raid*, students retain 94.8%, 58.4%, and 87.5% of their respective teachers’ mean returns for seeds 0, 1, and 2. Seed 1 shows the largest degradation, scoring 41.6% below its teacher. Action traces reveal a strong directional bias: the student selects RIGHTFIRE on 74.9% of teacher-controlled steps, compared with 30.4% for the teacher, and replaces 70.4% of the teacher’s LEFT decisions with RIGHTFIRE. This suggests incomplete transfer of movement behavior.

On *Ms. Pac-Man*, students retain 112.1%, 96.3%, and 108.9% of their teachers’ mean returns for seeds 0, 1, and 2, respectively. Unlike *Air Raid*, students outperform their teachers on two of the three seeds. Seed 0 achieves 80.8% action agreement while exceeding its teacher’s mean return by 12.1%, suggesting that imperfect policy imitation can still produce favorable gameplay outcomes.

Overall, RSSM distillation preserves substantial teacher performance in most evaluated configurations, but its effectiveness varies across training seeds and environments.

5.4 Combining the Distilled Encoder and RSSM

We evaluate five configurations for combining independently distilled encoder and RSSM components on *Air Raid*: direct merging, and encoder-only adaptation and joint fine-tuning, each with fixed weights and with GradNorm. All configurations are evaluated over 100 matched episodes.

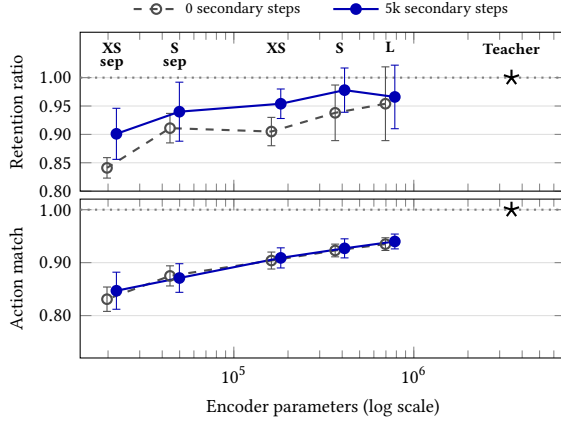
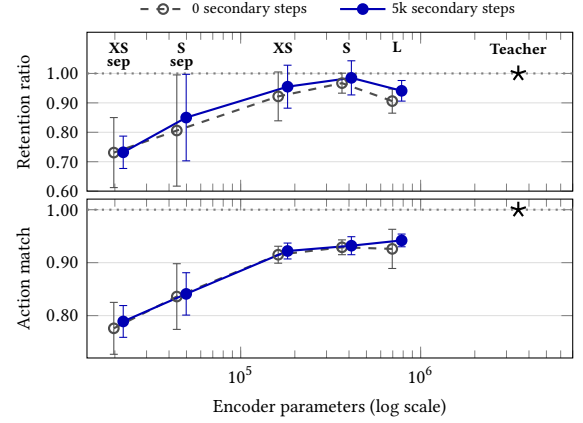

 (a) *Air Raid*

 (b) *Ms. Pac-Man*

Figure 2: Retention ratio (top) and action match with teacher (bottom) against encoder parameter count, with 0 and 5k secondary steps (each followed by 5k primary steps). Points are means over 3 seeds; error bars show ± 1 standard deviation. Points are offset slightly horizontally for readability. The asterisk marks the teacher.

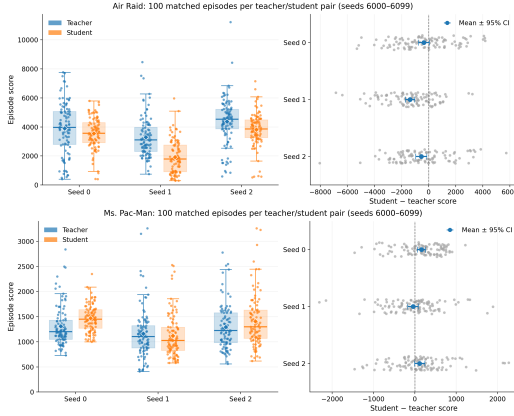


Figure 3: *Air Raid* (top) and *Ms. Pac-Man* (bottom) teacher-student episode returns (left) and paired score differences with 95% confidence intervals (right) across three training seeds, evaluated over 100 matched episodes per model.

Direct merging achieves a mean return of 2085.75. Encoder-only adaptation improves performance to 3032.25 (+45.4%) with fixed weights and 3412.00 (+63.6%) with GradNorm, as illustrated in Figure 4.

Joint fine-tuning achieves mean returns of 2895.75 (+38.8%) with fixed weights and 3224.75 (+54.6%) with GradNorm. Their respective 95% bootstrap confidence intervals are [2548.23, 3254.00] and [2889.74, 3573.51].

Overall, encoder-only adaptation with GradNorm achieves the highest mean return among merged configurations. These results suggest that aligning the independently distilled representations is important, while jointly updating all components does not yield further improvement over encoder-only adaptation.

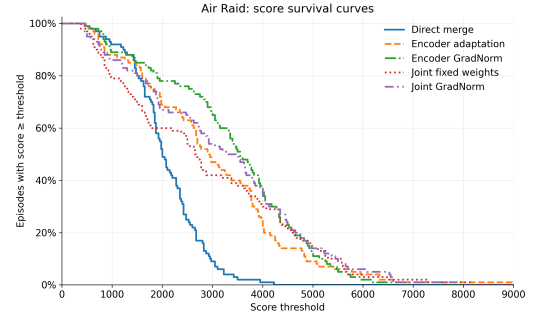


Figure 4: *Air Raid* score survival curves for direct merging and encoder-only adaptation methods over 100 matched episodes. Higher curves indicate better performance.

5.5 Comparison Against a 25M-Parameter Model (Q3)

To assess whether distillation outperforms independent training at a comparable model size, we evaluate our distilled configurations against an undistilled 25M DreamerV3 model on 100 matched *Air Raid* episodes.

As shown in Figure 5, all five merged variants contain 15.42M parameters, compared with 15.71M for the undistilled 25M baseline. The latter count excludes training-only components, such as the decoder, critic, reward and continuation heads, retaining only the parameters included in the deployment package. Despite their smaller size, all merged variants achieve higher mean returns. Encoder adaptation with GradNorm performs best, reaching 3412.00 compared with the baseline’s 1854.50, an improvement of 84.0%.

The student retaining the teacher’s frozen encoder achieves the highest overall return of 4131.75, although its parameter count is larger at 18.53M. Direct merging yields only a modest improvement, whereas subsequent adaptation substantially improves performance. These results suggest that distillation can provide better

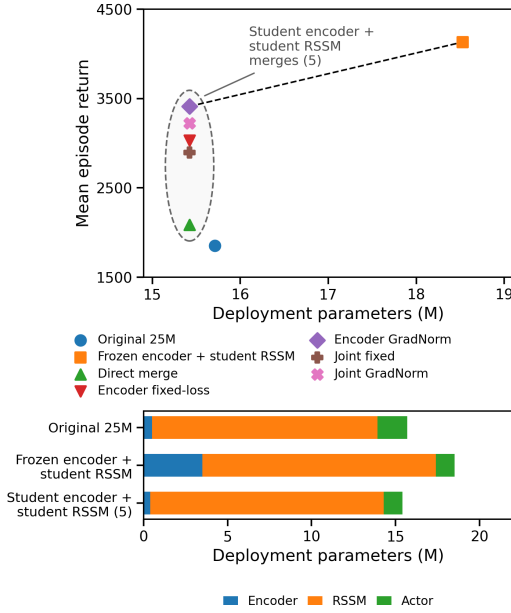


Figure 5: *Air Raid* performance versus deployment parameter count (left) and parameter distribution across model components (right). Returns are averaged over 100 matched evaluation episodes.

control performance than independent training at a comparable parameter budget.

5.6 Discussion

Encoder Distillation: Encoder compression generally preserves teacher performance across training seeds, with standard convolutional students retaining over 90% of the teacher’s return and action agreement. Secondary-loss pretraining further improves retention and latent fidelity in most configurations. However, more aggressive compression using depthwise separable convolutions introduces greater performance variability, indicating a trade-off between model size and behavioral consistency. We have also observed that many S students outperformed L students. We speculate that the S students may have happened to only captured the most useful parts of the teacher and disregarded performance degrading signals as a result of size reduction. Further experiments and more game evaluation would be required to understand this behavior.

RSSM Distillation: RSSM compression is less reliable across training seeds than encoder distillation, suggesting that recurrent dynamics are more sensitive to compression. On *Air Raid*, retention ranges from 94.8% in the seed-0 evaluation to 58.4% for seed 1, indicating that effectiveness depends on the trained model. Moreover, matching latent distributions does not guarantee matching behavior: despite strong prior and posterior matching, the compressed RSSM agrees with the teacher’s actions in only 58.5% of steps while retaining substantial return. This suggests that exact action imitation is not necessary for maintaining task performance. The occasional improvements over the teacher on *Ms. Pac-Man*

may similarly indicate preservation of task-relevant information, although these results may reflect evaluation variance and require further investigation. Finally, a 6.38 $\times$ parameter reduction yields only a 2.91 $\times$ inference speedup, showing that compression and latency improvements are not proportional.

Merged Distillation: Independently distilled encoder and RSSM components require additional adaptation to align their latent representations. GradNorm outperforms fixed loss weights in both merging strategies, suggesting that adaptive loss balancing is beneficial for this alignment. Furthermore, all merged students outperform an independently trained DreamerV3 model of comparable size, indicating that transferring teacher knowledge can be more effective than training a small model from scratch. However, this comparison is limited to a single game and baseline, and further evaluation is needed to establish generalizability.

6 Conclusion

We investigated whether the deployed components of the *DreamerV3* architecture - the encoder and the RSSM - can be compressed through knowledge distillation while preserving the teacher’s behavior. Distilling the CNN encoder alone provided robust results: with a secondary loss warm-up, students with up to 20 $\times$ fewer encoder parameters retain at least 95% of the teacher’s score on both *Air Raid* and *Ms. Pac-Man* (see Appendix B, Table 4). Depthwise separable students reduce the encoder further at a larger and more variable cost in performance. Compressing the RSSM reduces the deployed model from 132.81M to 20.82M parameters (84.3% decrease), which in turn yields a 2.91 $\times$ speed-up, all while still retaining 94.8% of its teacher’s mean return on *Air Raid*. Through different methods, we adapt the independently distilled components to run them in tandem, after which the merged students outperform an undistilled *DreamerV3* model of comparable size by up to 84%. This indicates that distillation is a promising and more effective route to obtaining compact agents than training small models directly.

Future Work: Future research could explore varying encoder and RSSM sizes to identify the optimal trade-off between compression, inference efficiency, and control performance. Explainable AI techniques could help identify redundant and task-relevant latent representations, providing insights into performance degradation and guiding more effective distillation objectives. Evaluating the distilled models on resource-constrained hardware would further validate their practical efficiency in terms of latency, memory, and energy consumption. Additionally, joint encoder-RSSM distillation, on-policy adaptation, multi-step dynamics matching, and ablation studies on different loss signals could improve behavioral consistency and robustness. Finally, the future work could discover possible bottlenecks for the distilled Encoder and RSSM via testing them in different environments with complex spatial and strategic difficulty.

Limitations: Our evaluation covers only two Atari games and three training seeds, with varying RSSM distillation performance. The comparison with a smaller undistilled DreamerV3 is limited to *Air Raid*, and inference efficiency is tested only on a desktop GPU. Furthermore, reliance on offline replay data and separate

encoder–RSSM distillation may limit behavioral fidelity and require additional adaptation.

Acknowledgments

We thank Thiago D. Simão for supervising this project and providing valuable feedback throughout.

References

- [1] Lei Jimmy Ba and Rich Caruana. 2014. Do Deep Nets Really Need to be Deep? arXiv:1312.6184 [cs.LG] <https://arxiv.org/abs/1312.6184>
- [2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint abs/1308.3432* (2013). arXiv:1308.3432 <https://arxiv.org/abs/1308.3432>
- [3] Wuhui Chen, Xiaoyu Qiu, Ting Cai, Hong-Ning Dai, Zibin Zheng, and Yan Zhang. 2021. Deep Reinforcement Learning for Internet of Things: A Comprehensive Survey. *IEEE Communications Surveys Tutorials* 23, 3 (2021), 1659–1692. doi:10.1109/COMST.2021.3073036
- [4] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. 2017. GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks. *CoRR abs/1711.02257* (2017). arXiv:1711.02257 <http://arxiv.org/abs/1711.02257>
- [5] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. 2018. Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. In *Advances in Neural Information Processing Systems*. 4754–4765.
- [6] Xianzhe Fan, Yuxiang Lu, Shenyuan Gao, Xiaoyang Wu, Ruihua Han, Manling Li, and Hengshuang Zhao. 2026. DreamAvoid: Critical-Phase Test-Time Dreaming to Avoid Failures in VLA Policies. arXiv:2605.11750 [cs.RO] <https://arxiv.org/abs/2605.11750>
- [7] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. 2018. Learning latent dynamics for planning from pixels. *arXiv preprint abs/1811.04551* (2018). arXiv:1811.04551 <https://arxiv.org/abs/1811.04551>
- [8] Danijar Hafner, Timothy P. Lillicrap, Jimmy Ba, and Mohammad Norouzi. 2019. Dream to Control: Learning Behaviors by Latent Imagination. *CoRR abs/1912.01603* (2019). arXiv:1912.01603 <http://arxiv.org/abs/1912.01603>
- [9] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. 2024. Mastering Diverse Domains through World Models. arXiv:2301.04104 [cs.AI] <https://arxiv.org/abs/2301.04104>
- [10] Nicklas Hansen, Hao Su, and Xiaolong Wang. 2024. TD-MPC2: Scalable, Robust World Models for Continuous Control. *arXiv preprint arXiv:2310.16828* (2024).
- [11] Nicklas Hansen, Xiaolong Wang, and Hao Su. 2022. Temporal Difference Learning for Model Predictive Control. In *International Conference on Machine Learning*. PMLR, 8385–8406.
- [12] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the Knowledge in a Neural Network. arXiv:1503.02531 [stat.ML] <https://arxiv.org/abs/1503.02531>
- [13] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR abs/1704.04861* (2017). arXiv:1704.04861 <http://arxiv.org/abs/1704.04861>
- [14] Ingoook Jang, Hyunseok Kim, Donghun Lee, Young-Sung Son, and Seonghyun Kim. 2020. Knowledge Transfer for On-Device Deep Reinforcement Learning in Resource Constrained Edge Computing Systems. *IEEE Access* 8 (2020), 146588–146597. doi:10.1109/ACCESS.2020.3014922
- [15] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. 2019. When to Trust Your Model: Model-Based Policy Optimization. In *Advances in Neural Information Processing Systems*. 12519–12530.
- [16] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101, 1–2 (1998), 99–134. doi:10.1016/S0004-3702(98)00023-X
- [17] Md. Al-Masrur Khan, Md Rashed Jaowad Khan, Abul Tooshil, Niloy Sikder, M. A. Parvez Mahmud, Abbas Z. Kouzani, and Abdullah-Al Nahid. 2020. A Systematic Review on Reinforcement Learning-Based Robotics Within the Last Decade. *IEEE Access* 8 (2020), 176598–176623. doi:10.1109/ACCESS.2020.3027152
- [18] Feeza Khan Khanzada and Jaerock Kwon. 2025. Driving Beyond Privilege: Distilling Dense-Reward Knowledge into Sparse-Reward Policies. arXiv:2512.04279 [cs.RO] <https://arxiv.org/abs/2512.04279>
- [19] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint abs/1312.6114* (2013). arXiv:1312.6114 <https://arxiv.org/abs/1312.6114>
- [20] Dmytro Kuzmenko and Nadiya Shvai. 2025. TD-MPC-Opt: Distilling Model-Based Multi-Task Reinforcement Learning Agents. *arXiv preprint arXiv:2507.01823* (2025).
- [21] Hongjia Li, Tianshu Wei, Ao Ren, Qi Zhu, and Yanzhi Wang. 2017. Deep reinforcement learning: Framework, applications, and embedded implementations: Invited paper. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 847–854. doi:10.1109/ICCAD.2017.8203866
- [22] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. arXiv:1711.05101 [cs.LG] <https://arxiv.org/abs/1711.05101>
- [23] Fan-Ming Luo, Tian Xu, Hang Lai, Xiong-Hui Chen, Weinan Zhang, and Yang Yu. 2022. A Survey on Model-based Reinforcement Learning. arXiv:2206.09328 [cs.LG] <https://arxiv.org/abs/2206.09328>
- [24] Nguyen Cong Luong, Dinh Thai Hoang, Shimin Gong, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. 2019. Applications of Deep Reinforcement Learning in Communications and Networking: A Survey. *IEEE Communications Surveys Tutorials* 21, 4 (2019), 3133–3174. doi:10.1109/COMST.2019.2916583
- [25] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. 2018. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. arXiv:1807.11164 [cs.CV] <https://arxiv.org/abs/1807.11164>
- [26] Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. 2023. Model-based Reinforcement Learning: A Survey. *Foundations and Trends in Machine Learning* 16, 1 (2023), 1–118. doi:10.1561/22000000086
- [27] Naoki Morihiro, Amal Nahar, Kartik Bharadwaj, Yasuhiro Kato, Akinobu Hayashi, and Tatsuya Harada. 2026. R2-Dreamer: Redundancy-Reduced World Models without Decoders or Augmentation. In *The Fourteenth International Conference on Learning Representations*. <https://openreview.net/forum?id=Je2QqXrcQq>
- [28] Andrei A. Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. 2016. Policy Distillation. arXiv:1511.06295 [cs.LG] <https://arxiv.org/abs/1511.06295>
- [29] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. 2020. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature* 588, 7839 (2020), 604–609. doi:10.1038/s41586-020-03051-4
- [30] Sergio Spanò, Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Matta, Alberto Nannarelli, and Marco Re. 2019. An Efficient Hardware Implementation of Reinforcement Learning: The Q-Learning Algorithm. *IEEE Access* 7 (2019), 186340–186351. doi:10.1109/ACCESS.2019.2961174
- [31] Richard S. Sutton. 1991. Dyna, an Integrated Architecture for Learning, Planning, and Reacting. *SIGART Bulletin* 2, 4 (1991), 160–163. doi:10.1145/122344.122377
- [32] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press, Cambridge, MA.
- [33] Chen Tang, Ben Abbatematteo, Jiaheng Hu, Rohan Chandra, Roberto Martin-Martin, and Peter Stone. 2025. Deep Reinforcement Learning for Robotics: A Survey of Real-World Successes. *Annual Review of Control, Robotics, and Autonomous Systems* 8, Volume 8, 2025 (2025), 153–188. doi:10.1146/annurev-control-030323-022510
- [34] Dingrui Wang, Zhexiong Sun, Zhouheng Li, Cheng Wang, Youlun Peng, Hongyuan Ye, Baha Zarrouki, Wei Li, Mattia Piccinini, Lei Xie, and Johannes Betz. 2025. Enhancing Physical Consistency in Lightweight World Models. arXiv:2509.12437 [cs.AI] <https://arxiv.org/abs/2509.12437>
- [35] Yuchen Wang, Jiangtao Kong, Sizhe Wei, Xiaochang Li, Haohong Lin, Hongjue Zhao, Tianyi Zhou, Lu Gan, and Huajie Shao. 2026. WestWorld: A Knowledge-Encoded Scalable Trajectory World Model for Diverse Robotic Systems. arXiv:2603.14392 [cs.LG] <https://arxiv.org/abs/2603.14392>
- [36] Rui Xu, Siyu Luan, Zonghua Gu, Qingling Zhao, and Gang Chen. 2022. LRP-based Policy Pruning and Distillation of Reinforcement Learning Agents for Embedded Systems. In *2022 IEEE 25th International Symposium On Real-Time Distributed Computing (ISORC)*. 1–8. doi:10.1109/ISORC52572.2022.9812837
- [37] Jun Yamada, Marc Rigter, Jack Collins, and Ingmar Posner. 2023. TWIST: Teacher-Student World Model Distillation for Efficient Sim-to-Real Transfer. arXiv:2311.03622 [cs.RO] <https://arxiv.org/abs/2311.03622>
- [38] Wei Zhu and Mitsuhiro Hayashibe. 2023. Autonomous Navigation System in Pedestrian Scenarios Using a Dreamer-Based Motion Planner. *IEEE Robotics and Automation Letters* 8, 6 (2023), 3836–3843. doi:10.1109/LRA.2023.3273514

Appendix

A History of RSSM Development

We compared several configurations during the development of the RSSM distillation procedure. Initially, the compressed RSSM used large teacher-width prediction heads or a frozen teacher actor with an adapter. Although these configurations reduced the RSSM size, the additional heads and adapters limited overall compression. We therefore introduced compact trainable heads operating directly on the student state. A subsequent configuration increased head capacity and the action-imitation weight to investigate whether stronger policy supervision could improve control performance.

During development, we also identified and corrected a mixed-precision burn-in issue that prevented gradients from reaching important recurrent and posterior weights. Following this correction, KL-only training achieved strong latent-distribution matching. We then initialized joint training of the RSSM, actor, and prediction heads from the KL-only checkpoint, aiming to preserve latent fidelity while improving action selection. This configuration achieved the highest evaluated student return.

B Extended Results

As shown in Table 3, joint adaptation maintained comparable latent-distribution fidelity while substantially improving teacher-action agreement through student actor training.

Table 3: Replay distribution fidelity before and after joint adaptation for seed 0. Lower KL and higher agreement are preferable. The KL-only checkpoint has an untrained actor.

Metric	KL-only	Joint
Prior KL	2.99	2.75
Posterior KL	2.73	2.89
Posterior agreement	93.1%	93.1%
Teacher-action agreement	19.6%	58.5%

Table 4: Retention ratio and action match with the teacher for each game, with 0 and 5k secondary steps (0 sec., 5k sec.; each followed by 5k primary steps). Values are mean $\pm$ standard deviation over 3 seeds; action match is given in percent. sep: depthwise separable convolutions.

Encoder	Retention ratio		Action match (%)	
	0 sec.	5k sec.	0 sec.	5k sec.
<i>Air Raid</i>				
L	0.95 $\pm$ 0.06	0.97 $\pm$ 0.06	93.5 $\pm$ 1.2	94.0 $\pm$ 1.4
S	0.94 $\pm$ 0.05	0.98 $\pm$ 0.04	92.3 $\pm$ 1.2	92.7 $\pm$ 1.8
XS	0.91 $\pm$ 0.02	0.95 $\pm$ 0.03	90.4 $\pm$ 1.6	90.9 $\pm$ 1.9
S, sep	0.91 $\pm$ 0.03	0.94 $\pm$ 0.05	87.5 $\pm$ 1.9	87.1 $\pm$ 2.7
XS, sep	0.84 $\pm$ 0.02	0.90 $\pm$ 0.05	83.1 $\pm$ 2.3	84.7 $\pm$ 3.5
<i>Ms. Pac-Man</i>				
L	0.91 $\pm$ 0.04	0.94 $\pm$ 0.03	92.6 $\pm$ 3.7	94.2 $\pm$ 1.2
S	0.97 $\pm$ 0.03	0.98 $\pm$ 0.06	92.9 $\pm$ 1.4	93.2 $\pm$ 1.7
XS	0.92 $\pm$ 0.08	0.96 $\pm$ 0.07	91.5 $\pm$ 1.6	92.2 $\pm$ 1.5
S, sep	0.81 $\pm$ 0.19	0.85 $\pm$ 0.15	83.6 $\pm$ 6.2	84.1 $\pm$ 4.0
XS, sep	0.73 $\pm$ 0.12	0.73 $\pm$ 0.05	77.6 $\pm$ 4.9	78.9 $\pm$ 3.0

Table 5: Quality of student latent states for each game, with 0 and 5k secondary steps (0 sec., 5k sec.; each followed by 5k primary steps). The teacher’s frozen RSSM is run on the student embeddings over 64-step windows of the validation split. Post. KL: D_{KL} between teacher and student posteriors over z_t (secondary loss $\mathcal{L}_z$); h L2: L2 distance between teacher and student recurrent states ($\mathcal{L}_h$). Values are mean $\pm$ standard deviation over 3 seeds.

Encoder	Post. KL		h L2	
	0 sec.	5k sec.	0 sec.	5k sec.
<i>Air Raid</i>				
L	2.64 $\pm$ 0.23	2.33 $\pm$ 0.17	7.6 $\pm$ 1.6	7.0 $\pm$ 2.7
S	3.68 $\pm$ 0.60	3.20 $\pm$ 0.02	11.9 $\pm$ 1.5	10.1 $\pm$ 2.8
XS	5.08 $\pm$ 0.31	4.79 $\pm$ 0.17	19.5 $\pm$ 3.7	18.7 $\pm$ 4.4
S, sep	7.53 $\pm$ 0.44	7.73 $\pm$ 0.43	33.2 $\pm$ 8.3	36.8 $\pm$ 12.4
XS, sep	12.65 $\pm$ 2.47	10.30 $\pm$ 1.36	60.7 $\pm$ 14.4	49.6 $\pm$ 21.5
<i>Ms. Pac-Man</i>				
L	4.50 $\pm$ 2.26	3.18 $\pm$ 0.05	20.0 $\pm$ 14.3	13.1 $\pm$ 2.0
S	4.62 $\pm$ 0.70	4.45 $\pm$ 0.51	18.5 $\pm$ 4.2	18.3 $\pm$ 5.0
XS	5.88 $\pm$ 0.14	5.35 $\pm$ 0.51	23.9 $\pm$ 3.1	21.6 $\pm$ 2.0
S, sep	13.99 $\pm$ 3.71	13.43 $\pm$ 3.24	86.8 $\pm$ 38.2	87.1 $\pm$ 33.2
XS, sep	20.90 $\pm$ 2.30	18.81 $\pm$ 2.47	159.5 $\pm$ 22.9	147.8 $\pm$ 20.2

Tables 4 and 5 report the students’ retention ratio plus action match, and mean posterior KL plus mean difference in history states respectively. Figure 6 summarizes the inference time of all encoders on an RTX 4080 Laptop GPU.

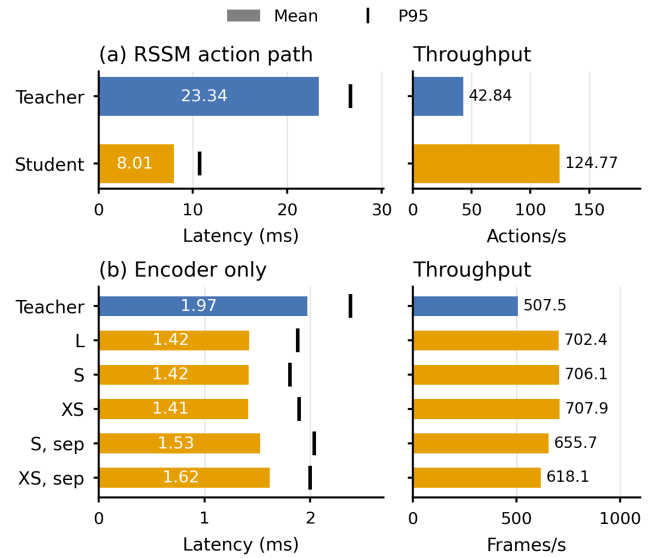


Figure 6: Inference efficiency on an RTX 4080 Laptop GPU using float32 inference. (a) RSSM action-selection latency and throughput. (b) Encoder latency and throughput across model sizes. Bars show mean latency, black markers indicate P95.

C Algorithm Architecture Diagrams

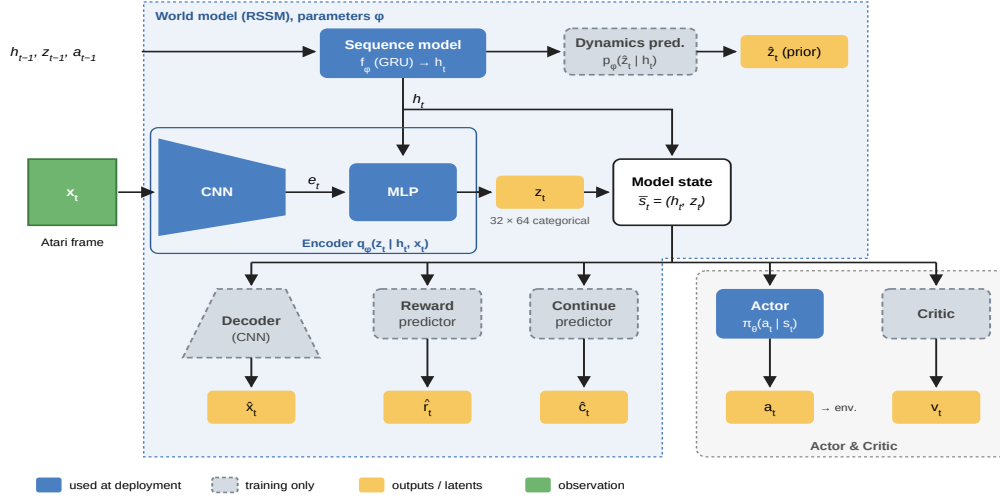


Figure 7: DreamerV3 at a single timestep. The world model (left) encodes the frame x_t into z_t , which together with the recurrent state h_t forms the model state s_t ; the actor and critic (right) operate on s_t . Blue components are executed at deployment, grey components are used only during training.

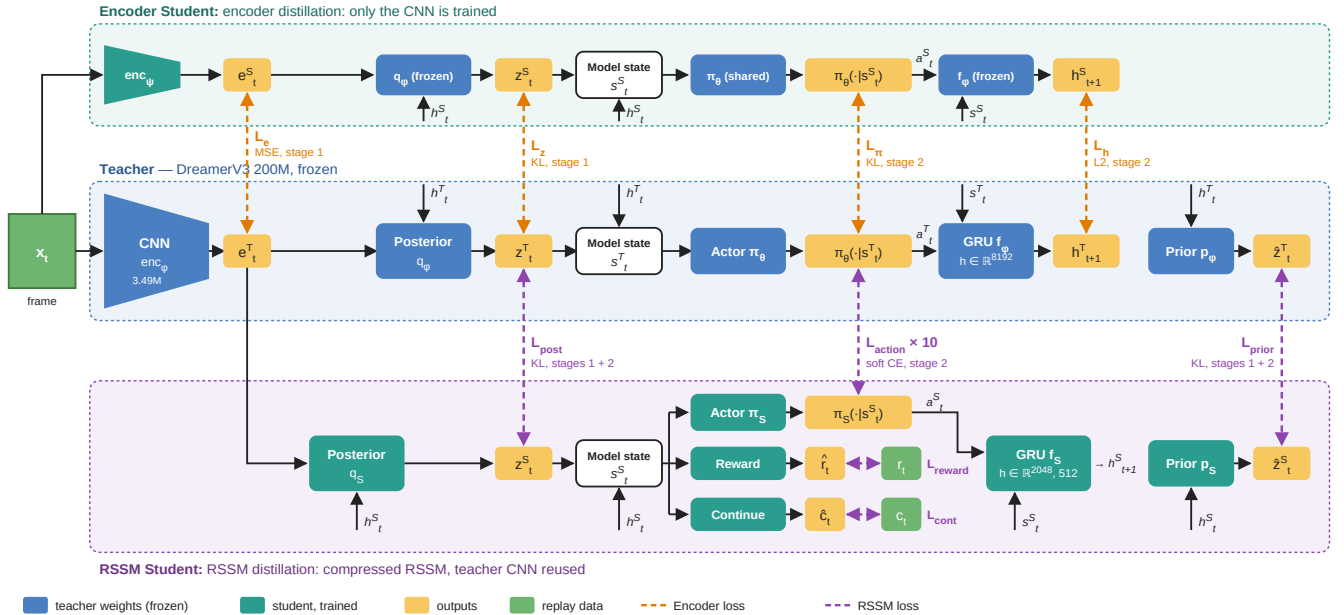


Figure 8: Overview of both distillation settings around the frozen DreamerV3 teacher (middle). Top: encoder distillation (Section 4.1), where only the student CNN enc_ψ is trained and the posterior, actor and GRU are frozen teacher modules; orange losses L_e , L_z form the secondary (stage 1) loss and L_π , L_h the primary (stage 2) loss. Bottom: RSSM distillation (Section 4.2), where the student reuses the teacher's CNN embedding e_t and trains a compressed RSSM ($h_t \in \mathbb{R}^{2048}$) with its own actor, reward and continuation heads; purple losses are the prior and posterior KL (both stages), soft action imitation (stage 2), and reward and continuation losses on replay data (stage 2).