A MODELS AND HYPERPARAMETERS

A.1 DISCRETE AUTOENCODER

Our discrete autoencoder is based on the implementation of VQGAN (Esser et al., 2021). We removed the discriminator, essentially turning the VQGAN into a vanilla VQVAE (Van Den Oord et al., 2017) with an additional perceptual loss (Johnson et al., 2016; Larsen et al., 2016).

Table 2: Encoder / Decoder hyperparameters. We list the hyperparameters for the encoder, the same ones apply for the decoder.

Hyperparameter	Value
Frame dimensions (h, w)	64×64
Layers	4
Residual blocks per layer	2
Channels in convolutions	64
Self-attention layers at resolution	8 / 16

Table 3: Embedding table hyperparameters.

Hyperparameter	Value
Vocabulary size (N)	512
Tokens per frame (K)	16
Token embedding dimension (d)	512

Note that during experience collection in the real environment, frames still go through the autoencoder to keep the input distribution of the policy unchanged. See Algorithm 1 for details.

A.2 TRANSFORMER

Our autoregressive Transformer is based on the implementation of minGPT (Karpathy, 2020). It takes as input a sequence of $L(K+1)$ tokens and embeds it into a $L(K+1) \times D$ tensor using an $A \times D$ embedding table for actions, and a $N \times D$ embedding table for frames tokens. This tensor is forwarded through M Transformer blocks. We use GPT2-like blocks (Radford et al., 2019), i.e. each block consists of a self-attention module with layer normalization of the input, wrapped with a residual connection, followed by a per-position multi-layer perceptron with layer normalization of the input, wrapped with another residual connection.

Table 4: Transformer hyperparameters

Hyperparameter	Value
Timesteps (L)	20
Embedding dimension (D)	256
Layers (M)	10
Attention heads	4
Weight decay	0.01
Embedding dropout	0.1
Attention dropout	0.1
Residual dropout	0.1

A.3 ACTOR-CRITIC

The weights of the actor and critic are shared except for the last layer. The actor-critic takes as input a $64 \times 64 \times 3$ frame, and forwards it through a convolutional block followed by an LSTM cell (Mnih et al., 2016; Hochreiter & Schmidhuber, 1997; Gers et al., 2000). The convolutional block consists of the same layer repeated four times: a 3x3 convolution with stride 1 and padding 1, a ReLU activation, and 2x2 max-pooling with stride 2. The dimension of the LSTM hidden state is 512. Before starting the imagination procedure from a given frame, we burn-in (Kapturowski et al., 2019) the 20 previous frames to initialize the hidden state.

Table 5: Training loop & Shared hyperparameters

Hyperparameter	Value
Epochs	600
# Collection epochs	500
Environment steps per epoch	200
Collection epsilon-greedy	0.01
Eval sampling temperature	0.5
Start autoencoder after epochs	5
Start transformer after epochs	25
Start actor-critic after epochs	50
Autoencoder batch size	256
Transformer batch size	64
Actor-critic batch size	64
Training steps per epoch	1000
Learning rate	1e-4
Optimizer	Adam
Adam β_1	0.9
Adam β_2	0.999
Max gradient norm	10.0

B ACTOR-CRITIC LEARNING OBJECTIVES

We follow Dreamer (Hafner et al., 2020; 2021) in using the generic λ-return, that balances bias and variance, as the regression target for the value network. Given an imagined trajectory $(\hat{x}_0, a_0, \hat{r}_0, \hat{d}_0, \ldots, \hat{x}_{H-1}, a_{H-1}, \hat{r}_{H-1}, \hat{d}_{H-1}, \hat{x}_H)$, the λ-return can be defined recursively as follows:

$$\Lambda_t = \begin{cases} \hat{r}_t + \gamma(1-\hat{d}_t)\Big[(1-\lambda)V(\hat{x}_{t+1}) + \lambda\Lambda_{t+1}\Big] & \text{if} \quad t < H \\ V(\hat{x}_H) & \text{if} \quad t = H \end{cases} \tag{4}$$

The value network V is trained to minimize $\mathcal{L}_V$, the expected squared difference with λ-returns over imagined trajectories.

$$\mathcal{L}_V = \mathbb{E}_\pi\Big[\sum_{t=0}^{H-1} \big(V(\hat{x}_t) - \mathrm{sg}(\Lambda_t)\big)^2\Big] \tag{5}$$

Here, $\mathrm{sg}(\cdot)$ denotes the gradient stopping operation, meaning that the target is a constant in the gradient-based optimization, as classically established in the literature (Mnih et al., 2015; Hessel et al., 2018; Hafner et al., 2020).

As large amounts of trajectories are generated in the imagination MDP, we can use a straightforward reinforcement learning objective for the policy, such as REINFORCE (Sutton & Barto, 2018). To reduce the variance of REINFORCE gradients, we use the value $V(\hat{x}_t)$ as a baseline (Sutton & Barto, 2018). We also add a weighted entropy maximization objective to maintain a sufficient exploration. The actor is trained to minimize the following REINFORCE objective over imagined trajectories:

$$\mathcal{L}_\pi = -\mathbb{E}_\pi\Big[\sum_{t=0}^{H-1} \log(\pi(a_t|\hat{x}_{\leq t}))\,\mathrm{sg}(\Lambda_t - V(\hat{x}_t)) + \eta\,\mathcal{H}(\pi(a_t|\hat{x}_{\leq t}))\Big] \tag{6}$$

Table 6: RL training hyperparameters

Hyperparameter	Value
Imagination horizon (H)	20
γ	0.995
λ	0.95
η	0.001

C OPTIMALITY GAP

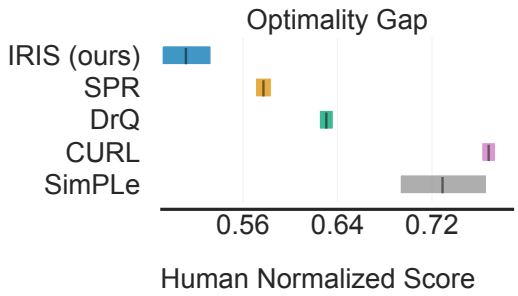

Figure 8: Optimality gap. The amount by which the algorithm fails to reach a human-level score (Agarwal et al., 2021), lower is better.

D IRIS ALGORITHM

Algorithm 1: IRIS

Procedure `training_loop()`:
- **for** *epochs* **do**
  - `collect_experience(`*steps_collect*`)`
  - **for** *steps_world_model* **do**
    - `update_world_model()`
  - **for** *steps_behavior* **do**
    - `update_behavior()`

Procedure `collect_experience(`n`)`:
- $x_0 \leftarrow$ `env.reset()`
- **for** $t = 0$ **to** $n - 1$ **do**
  - $\hat{x}_t \leftarrow D(E(x_t))$ `// forward frame through discrete autoencoder`
  - Sample $a_t \sim \pi(a_t|\hat{x}_t)$
  - $x_{t+1}, r_t, d_t \leftarrow$ `env.step(`a_t`)`
  - **if** $d_t = 1$ **then**
    - $x_{t+1} \leftarrow$ `env.reset()`
- $\mathcal{D} \leftarrow \mathcal{D} \cup \{x_t, a_t, r_t, d_t\}_{t=0}^{n-1}$

Procedure `update_world_model()`:
- Sample $\{x_t, a_t, r_t, d_t\}_{t=\tau}^{\tau+L-1} \sim \mathcal{D}$
- Compute $z_t \coloneqq E(x_t)$ and $\hat{x}_t \coloneqq D(z_t)$ for $t = \tau, \ldots, \tau + L - 1$
- Update E and D
- Compute $p_G(\hat{z}_{t+1}, \hat{r}_t, \hat{d}_t \mid z_\tau, a_\tau, \ldots, z_t, a_t)$ for $t = \tau, \ldots, \tau + L - 1$
- Update G

Procedure `update_behavior()`:
- Sample $x_0 \sim \mathcal{D}$
- $z_0 \leftarrow E(x_0)$
- $\hat{x}_0 \leftarrow D(z_0)$
- **for** $t = 0$ **to** $H - 1$ **do**
  - Sample $a_t \sim \pi(a_t|\hat{x}_t)$
  - Sample $\hat{z}_{t+1}, \hat{r}_t, \hat{d}_t \sim p_G(\hat{z}_{t+1}, \hat{r}_t, \hat{d}_t \mid z_0, a_0, \ldots, \hat{z}_t, a_t)$
  - $\hat{x}_{t+1} \leftarrow D(\hat{z}_{t+1})$
- Compute $V(\hat{x}_t)$ for $t = 0, \ldots, H$
- Update π and V

E AUTOENCONDING FRAMES WITH VARYING AMOUNTS OF TOKENS

The length of the input sequence of G is determined by the number of tokens K used to encode a single frame and the number of timesteps L in memory. Increasing the number of tokens per frame results in better reconstructions, although it requires more compute and memory.

This tradeoff is particularly important in Atari games where enemies and players are moving in mazes with rewards to collect. Due to the high number of possible configurations, the discrete autoencoder struggles to properly encode frames with only $K = 16$ tokens. Indeed, sometimes the player, its enemies, or rewards are not correctly reconstructed, which severely hinders agent performance.

In Figure 9, we show that when increasing the number of tokens per frame to 64, the discrete autoencoder is perfectly capable of dealing with detailed environments such as *Alien*. However, this increases the sequence length of G from 340 to 1300. Therefore, with more computational resources, IRIS would most likely improve in these settings.

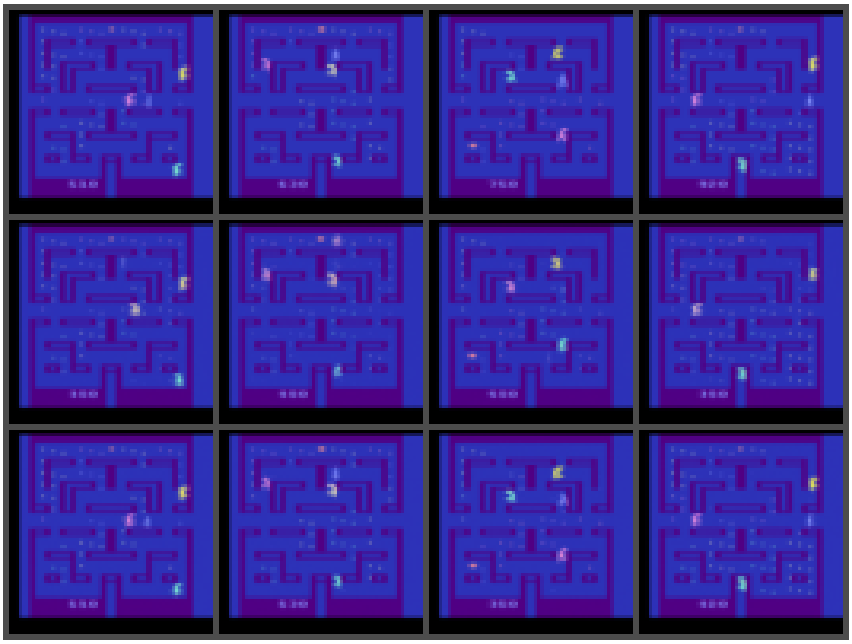

Figure 9: Tradeoff between the number of tokens per frame and reconstructions quality in *Alien*. Each column displays a 64×64 frame from the real environment (top), its reconstruction with a discrete encoding of 16 tokens (center), and its reconstruction with a discrete encoding of 64 tokens (bottom). In *Alien*, the player is the dark blue character, and the enemies are the large colored sprites. With 16 tokens per frame, the autoencoder often erases the player, switches colors, and misplaces rewards. When increasing the amount of tokens, it properly reconstructs the frame.

F COMPUTATIONAL RESOURCES

For each Atari environment, we repeatedly trained IRIS with 5 different random seeds. We ran our experiments with 8 Nvidia A100 40GB GPUs. With two Atari environments running on the same GPU, training takes around 7 days, resulting in an average of 3.5 days per environment.

SimPLe (Kaiser et al., 2020), the only baseline that involves learning in imagination, trains for 3 weeks with a P100 GPU on a single environment. As for SPR (Schwarzer et al., 2021), the strongest baseline without lookahead search, it trains notably fast in 4.6 hours with a P100 GPU.

Regarding baselines with lookahead search, MuZero (Schrittwieser et al., 2020) originally used 40 TPUs for 12 hours to train in a single Atari environment. Ye et al. (2021) train both EfficientZero and their reimplementation of MuZero in 7 hours with 4 RTX 3090 GPUs. EfficientZero's implementation relies on a distributed infrastructure with CPU and GPU threads running in parallel, and a C++/Cython implementation of MCTS. By contrast, IRIS and the baselines without lookahead search rely on straightforward single GPU / single CPU implementations.

G EXPLORATION IN FREEWAY

The reward function in Freeway is sparse since the agent is only rewarded when it completely crosses the road. In addition, bumping into cars will drag it down, preventing it from smoothly ascending the highway. This poses an exploration problem for newly initialized agents because a random policy will almost surely never obtain a non-zero reward with a 100k frames budget.

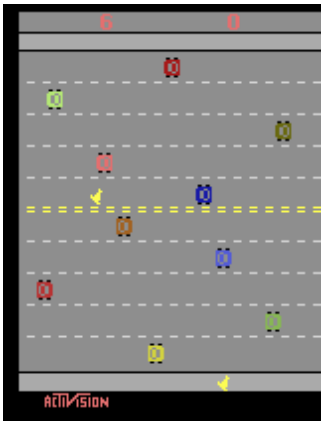

Figure 10: A game of *Freeway*. Cars will bump the player down, making it very unlikely to cross the road and be rewarded for random policies.

The solution to this problem is actually straightforward and simply requires stretches of time when the UP action is oversampled. Most Atari 100k baselines fix the issue with epsilon-greedy schedules and argmax action selection, where at some point the network configuration will be such that the UP action is heavily favored. In this work, we opted for the simpler strategy of having a fixed epsilon-greedy parameter and sampling from the policy. However, we lowered the sampling temperature from 1 to 0.01 for Freeway, in order to avoid random walks that would not be conducive to learning in the early stages of training. As a consequence, once it received its first few rewards through exploration, IRIS was able to internalize the sparse reward function in its world model.