Improving Token-Based World Models with Parallel Observation Prediction

Lior Cohen [1] **Kaixin Wang** [1] **Bingyi Kang** [2] **Shie Mannor** [1]

Abstract

Motivated by the success of Transformers when applied to sequences of discrete symbols, token-based world models (TBWMs) were recently proposed as sample-efficient methods. In TBWMs, the world model consumes agent experience as a language-like sequence of tokens, where each observation constitutes a sub-sequence. However, during imagination, the sequential token-by-token generation of next observations results in a severe bottleneck, leading to long training times, poor GPU utilization, and limited representations. To resolve this bottleneck, we devise a novel Parallel Observation Prediction (POP) mechanism. POP augments a Retentive Network (RetNet) with a novel forward mode tailored to our reinforcement learning setting. We incorporate POP in a novel TBWM agent named REM (Retentive Environment Model), showcasing a 15.4x faster imagination compared to prior TBWMs. REM attains superhuman performance on 12 out of 26 games of the Atari 100K benchmark, while training in less than 12 hours. Our code is available at https://github.com/leor-c/REM.

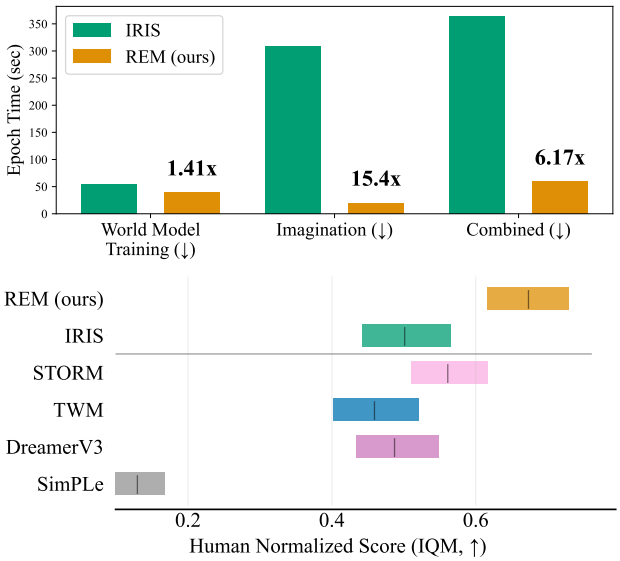

Figure 1. **Top:** comparison between the run times of token-based world model agents (IRIS and REM) during the world model training and imagination (actor-critic training). **Bottom:** interquantile mean (IQM) human-normalized score comparison between REM and state-of-the-art baselines on the Atari 100K benchmark with 95% stratified bootstrap confidence intervals (Agarwal et al., 2021). A line separates token-based methods from other baselines.

1. Introduction

Sample efficiency remains a central challenge in reinforcement learning (RL) due to the substantial data demands of successful RL algorithms (Berner et al., 2019; Mnih et al., 2015; Schrittwieser et al., 2020; Silver et al., 2016; Vinyals et al., 2019). One prominent model-based approach for addressing this challenge is known as world models. In world models, the agent's learning is solely based on simulated interaction data produced by a learned model of the environment through a process called imagination. World models are gaining increasing popularity due to their effectiveness, particularly in visual environments (Hafner et al., 2023).

In recent years, attention-based sequence models, most notably the Transformer architecture (Vaswani et al., 2017), demonstrated unmatched performance in language modeling tasks (Brown et al., 2020; Bubeck et al., 2023; Devlin et al., 2019; Touvron et al., 2023). The notable success of these models when applied to sequences of discrete tokens sparked motivation to employ these architectures to other data modalities by learning appropriate token-based representations. In computer vision, discrete representations are becoming a mainstream approach for various tasks (Dosovitskiy et al., 2021; Esser et al., 2021; Li et al., 2023; van den Oord et al., 2017). In RL, *token-based* world models were recently explored in visual environments (Micheli et al., 2023). The visual perception module in these methods is called a tokenizer, as it maps image observations to sequences of discrete symbols. This way, agent interaction is translated into a language-like sequence of discrete tokens, which are processed individually by the world model.

[1]Technion – Israel Institute of Technology [2]ByteDance. Correspondence to: Lior Cohen <liorcohen5@campus.technion.ac.il>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

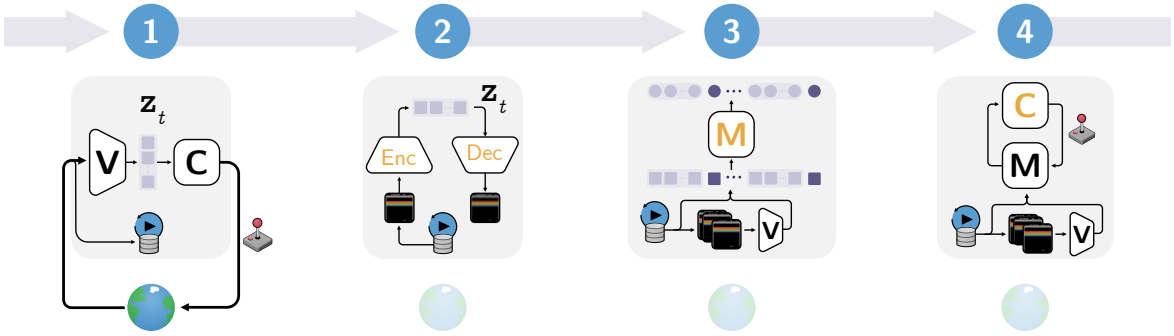

Figure 2. An overview of REM's training cycle. Each epoch has 4 steps: experience collection (1), tokenizer training (2), world model training (3), and controller training in imagination (4). Orange color represents component(s) that undergo training. Blue squares denote token inputs, where light blue is used for observation tokens and dark blue for actions. Replay buffer data at steps 3 and 4 contains observations, actions, rewards, and termination signals.

During imagination, to generate the tokens of the next observation with the auto-regressive model, the prediction is carried sequentially token-by-token. Effectively, this highly-sequential computation results in a severe bottleneck that pronouncedly hinders token-based approaches. Consequently, this bottleneck practically caps the length of the observation token sequences which in turn degrades performance. This limitation renders current token-based methods impractical for complex problems.

In this paper, we present Parallel Observation Prediction (POP), a novel mechanism that resolves the imagination bottleneck of token based world models (TBWMs). With POP, the enire next observation token sequence is generated in parallel during world model imagination. At its core, POP augments a Retentive Network (RetNet) sequence model (Sun et al., 2023) with a novel forward mode devised for retaining world model training efficiency. Additionally, we present REM (Retentive Environment Model), a TBWM agent driven by a POP-augmented RetNet architecture.

Our main contributions are summarized as follows:

- We propose Parallel Observation Prediction (POP), a novel mechanism that resolves the inference bottleneck of current token-based world models while retaining performance.
- We introduce REM, the first world model approach that incorporates the RetNet architecture. Our experiments provide first evidence of RetNet's performance in an RL setting.
- We evaluate REM on the Atari 100K benchmark, demonstrating the effectiveness of POP. POP leads to a 15.4x speed-up at imagination and trains in under 12 hours, while outperforming prior TBWMs.

2. Method

Notations. We consider the Partially Observable Markov Decision Process (POMDP) setting with image observations $\mathbf{o}_t \in \Omega \subseteq \mathbb{R}^{h \times w \times 3}$, discrete actions $a_t \in \mathcal{A}$, scalar rewards $r_t \in \mathbb{R}$, episode termination signals $d_t \in \{0, 1\}$, dynamics $\mathbf{o}_{t+1}, r_t, d_t \sim p(\mathbf{o}_{t+1}, r_t, d_t | \mathbf{o}_{\leq t}, a_{\leq t})$, and discount factor γ. The objective is to learn a policy π such that for every situation the output $\pi(a_t | \mathbf{o}_{\leq t}, a_{<t})$ is optimal w.r.t. the expected discounted sum of rewards from that situation $\mathbb{E}[\sum_{\tau=0}^{\infty} \gamma^\tau R_{t+\tau}]$ under the policy π.

2.1. Overview

REM builds on IRIS (Micheli et al., 2023), and similar to most prior works on world models for pixel input (Hafner et al., 2021; 2023; Kaiser et al., 2020), REM follows a $\mathcal{V}$-$\mathcal{M}$-$\mathcal{C}$ structure (Ha & Schmidhuber, 2018): a $\mathcal{V}$isual perception module that compresses observations into compact latent representations, a predictive $\mathcal{M}$odel that captures the environment's dynamics, and a $\mathcal{C}$ontroller that learns to act to maximize return. Additionally, a replay buffer is used to store environment interaction data. An overview of REM's training cycle is presented in Figure 2. A pseudo-code algorithm of REM is presented in Appendix A.2.

$\mathcal{V}$ - Tokenizer We instantiate the visual perception component as a tokenizer, mapping input observations into latent tokens. Following (Micheli et al., 2023), the tokenizer is a VQ-VAE discrete auto-encoder (Esser et al., 2021; van den Oord et al., 2017) comprised of an encoder, a decoder, and an embedding table. The embedding table $\mathcal{E} = \{\mathbf{e}_i\}_{i=1}^{N} \in \mathbb{R}^{N \times d}$ consists of N trainable vectors. The encoder first maps an input image $\mathbf{o}_t$ to a sequence of d-dimensional latent vectors $(\mathbf{h}_t^1, \mathbf{h}_t^2, \cdots, \mathbf{h}_t^K)$. Then, each latent vector $\mathbf{h}_t^k \in \mathbb{R}^d$ is mapped to the index of the nearest embedding in $\mathcal{E}$, , $z_t^k = \arg\min_i \|\mathbf{h}_t^k - \mathbf{e}_i\|$. Such indices are called *tokens*. For an input image $\mathbf{o}_t$, its latent

token sequence is denoted as $\mathbf{z}_t = (z_t^1, z_t^2, \cdots, z_t^K)$. To map a token sequence back to the input space, we first retrieve the embedding for each token and obtain a sequence $(\hat{\mathbf{h}}_t^1, \hat{\mathbf{h}}_t^2, \cdots, \hat{\mathbf{h}}_t^K)$ where $\hat{\mathbf{h}}_t^k = \mathbf{e}_{z_t^k}$. Then, inverse to the encoding process, the decoder is responsible for mapping this sequence to a reconstructed observation $\hat{\mathbf{o}}_t$.

The tokenizer is trained on frames sampled uniformly from the replay buffer. Its optimization objective, architecture, and other details are deferred to Appendix A.1.1.

$\mathcal{M}$ - World Model At the core of a world model is the component that captures the dynamics of the environment and makes predictions based on historical observations. Here, $\mathcal{M}$ is learned entirely in the latent token space, modeling the following distributions at each step t:

$$\text{Transition:} \quad p(\hat{\mathbf{z}}_{t+1} | \mathbf{z}_1, a_1, \ldots, \mathbf{z}_t, a_t), \tag{1}$$

$$\text{Reward:} \quad p(\hat{r}_t | \mathbf{z}_1, a_1, \ldots, \mathbf{z}_t, a_t), \tag{2}$$

$$\text{Termination:} \quad p(\hat{d}_t | \mathbf{z}_1, a_1, \ldots, \mathbf{z}_t, a_t). \tag{3}$$

To map observation tokens to embedding vectors, $\mathcal{M}$ uses the code vectors $\mathcal{E}$ learned by the tokenizer $\mathcal{V}$. Note that $\mathcal{E}$ is not updated by $\mathcal{M}$. In addition, $\mathcal{M}$ maintains dedicated embedding tables for mapping actions and special tokens (detailed in Section 2.3) to continuous vectors.

$\mathcal{C}$ - Controller REM's actor-critic controller $\mathcal{C}$ is trained to maximize return entirely in imagination (Hafner et al., 2021; Kaiser et al., 2020; Micheli et al., 2023). $\mathcal{C}$ comprises of a policy network π and a value function estimator V^π, and operates on latent tokens and their embeddings. In each optimization step, $\mathcal{M}$ and $\mathcal{C}$ are initialized with a short trajectory segment sampled from the replay buffer. Subsequently, the agent interacts with the world model for H steps. At each step t, the agent plays an action sampled from its policy $\pi(a_t | \mathbf{z}_1, a_1, \ldots, \mathbf{z}_{t-1}, a_{t-1}, \mathbf{z}_t)$. The world model evolves accordingly, generating $\hat{r}_t$, $\hat{d}_t$, and $\hat{\mathbf{z}}_{t+1}$ by sampling from the appropriate distributions (Eqn. (1-3)). The resulting trajectories are then used to train the agent. Following (Micheli et al., 2023), we adopted the actor-critic objectives of DreamerV2 (Hafner et al., 2021). We leave the full details of its architecture and optimization to Appendix A.1.3.

2.2. Retention Preliminaries

Similar to Transformers (Vaswani et al., 2017), a RetNet model (Sun et al., 2023) consists of a stack of layers, where each layer contains a multi-head Attention-like mechanism, called Retention, followed by a fully-connected network. A unique characteristic of the Retention mechanism is that it has a dual form of recurrence and parallelism, called "chunkwise", for improved efficiency when handling long

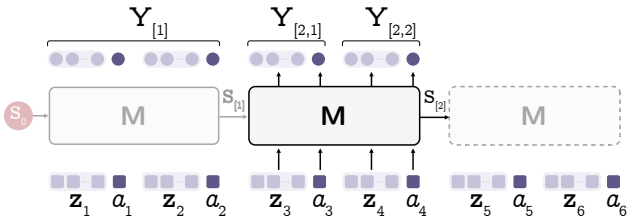

Figure 3. The "chunkwise" computation mode. Long sequences can be split into smaller "chunks" for enhanced training efficiency. Previous chunks are summarized by the recurrent state $\mathbf{S}$. Blue squares represents tokens, while circles denote output vectors. Crucially, RetNet's chunkwise mode does not natively support both a batched generation of tokens at imagination and an efficient world model training. These are achieved by our POP extension.

sequences. This form allows to split such sequences into smaller "chunks", where a parallel computation takes place within chunks and a sequential recurrent form is used between chunks, as shown in Figure 3. The information from previous chunks is summarized by a recurrent state $\mathbf{S} \in \mathbb{R}^{d \times d}$ maintained by the Retention mechanism.

Formally, consider a sequence of tokens $(x_1, x_2, \cdots, x_m)$. In our RL context, this sequence is a token trajectory composed of observation-action sub-sequences $(z_t^1, \cdots, z_t^K, a_t)$ we call *blocks*. As such trajectories are typically long, we split them into chunks of B tokens, where $B = c(K+1)$ is a multiple of $K+1$ so that each chunk only contains complete blocks. Here, the hyperparameter c can be tuned according to the size of the models, the hardware, and other factors to maximize efficiency. Let $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m) \in \mathbb{R}^{m \times d}$ be the d-dimensional token embedding vectors. The Retention output $\mathbf{Y}_{[i]} = \text{Retention}(\mathbf{X}_{[i]}, \mathbf{S}_{[i-1]}, i)$ of the i-th chunk is given by

$$\mathbf{Y}_{[i]} = \left(\mathbf{Q}_{[i]} \mathbf{K}_{[i]}^\top \odot \mathbf{D}\right) \mathbf{V}_{[i]} + (\mathbf{Q}_{[i]} \mathbf{S}_{[i-1]}) \odot \boldsymbol{\xi}, \tag{4}$$

where the bracketed subscript $[i]$ is used to index the i-th chunk, $\mathbf{Q} = (\mathbf{X}\mathbf{W}_Q) \odot \Theta$, $\mathbf{K} = (\mathbf{X}\mathbf{W}_K) \odot \bar{\Theta}$, $\mathbf{V} = \mathbf{X}\mathbf{W}_V$, and $\boldsymbol{\xi} \in \mathbb{R}^{B \times d}$ is a matrix with $\boldsymbol{\xi}_{ij} = \eta^{i+1}$. Here, $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d \times d}$ are learnable weights, η is an exponential decay factor, the matrix $\mathbf{D} \in \mathbb{R}^{B \times B}$ combines an auto-regressive mask with the temporal decay factor η, and the matrices $\Theta, \bar{\Theta} \in \mathbb{C}^{m \times d}$ are for relative position embedding (see Appendix A.3). Note the chunk index i argument of the Retention operator, which controls positional embedding information through Θ. The chunkwise update rule of the recurrent state is given by

$$\mathbf{S}_{[i]} = (\mathbf{K}_{[i]} \odot \boldsymbol{\zeta})^\top \mathbf{V}_{[i]} + \eta^B \mathbf{S}_{[i-1]} \tag{5}$$

where $\mathbf{S}_{[0]} = \mathbf{S}_0 = 0$, and $\boldsymbol{\zeta} \in \mathbb{R}^{B \times d}$ is a matrix with $\boldsymbol{\zeta}_{ij} = \eta^{B-i-1}$. On the right hand side of Equations 4 and

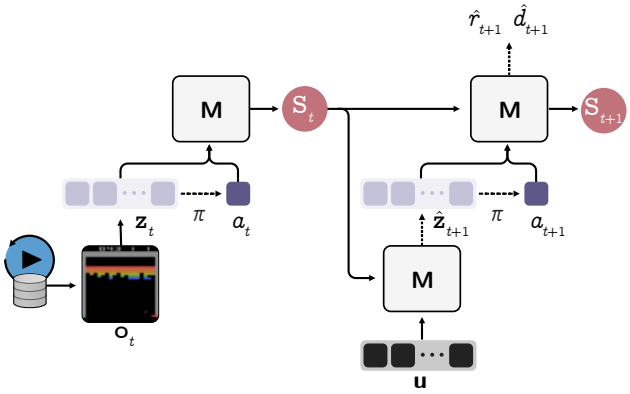

Figure 4. A single imagination step. Starting from a recurrent state $\mathbf{S}_t$, initially obtained from real experience, $\mathcal{M}$ computes all next-observation tokens $\hat{\mathbf{z}}_{t+1}$ in parallel using the prediction tokens $\mathbf{u}$ as inputs. Then, the agent observes $\hat{\mathbf{z}}_{t+1}$ and picks an action a_{t+1}. Finally, $\mathcal{M}$ takes $\mathbf{S}_t$, $\hat{\mathbf{z}}_{t+1}$, and a_{t+1} and outputs $\mathbf{S}_{t+1}, \hat{r}_{t+1}, \hat{d}_{t+1}$. Dashed arrows emphasize sampling operations.

5, the first term corresponds to the computation within the chunk while the second term incorporates the information from previous chunks, encapsulated by the recurrent state. Further details about the RetNet architecture are deferred to Appendix A.3.

2.3. World Model Imagination

As the agent's training relies entirely on world model imagination, the efficiency of the trajectory generation is critical. During imagination, predicting $\hat{\mathbf{z}}_{t+1}$ constitutes the primary non-trivial component and consumes the majority of processing time. In IRIS, the prediction of $\hat{\mathbf{z}}_{t+1}$ unfolds sequentially, as the model is limited to predicting only one token ahead at each step. This limitation arises since the identity of the next token, which remains unknown at the current step, is necessary for the prediction of later tokens. Thus, generating H observations costs KH sequential world model calls. This leads to poor GPU utilization and long computation time.

To overcome this bottleneck, POP maintains a set of K dedicated prediction tokens $\mathbf{u} = (u_1, \dots, u_K)$ together with their corresponding embeddings $\mathcal{E}_{\mathbf{u}} \in \mathbb{R}^{K \times d}$. To generate $\hat{\mathbf{z}}_{t+1}$ in one pass, POP simply computes the RetNet outputs starting from $\mathbf{S}_t$ using $\mathbf{u}$ as its input sequence, as illustrated in Figure 4. Note that at imagination, the chunk size is limited to a single block, i.e., to $K + 1$. Here, the notation $\mathbf{S}_t$ refers to the state that summarizes the first t observation-action blocks. To obtain $\mathbf{S}_t$, we use RetNet's chunkwise forward to summarize an initial context segment of blocks sampled from the replay buffer. Essentially, for every t, POP models the following distribution for next observation

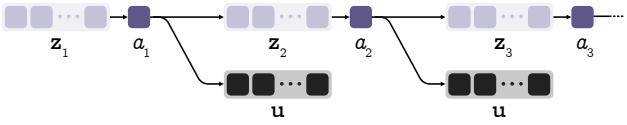

Figure 5. When appending $\mathbf{u}$ after an observation-action block, the sequence is no longer a prefix of the observation-action token trajectory. Thus, the recurrent state only summarizes observation and action tokens (top trajectory).

prediction:

$$p(\hat{\mathbf{z}}_{t+1} | \mathbf{z}_1, a_1, \dots, \mathbf{z}_t, a_t, \mathbf{u})$$

with

$$p(\hat{\mathbf{z}}^k_{t+1} | \mathbf{z}_1, a_1, \dots, \mathbf{z}_t, a_t, \mathbf{u}_{\leq k}).$$

It is worth noting that the tokens $\mathbf{u}$ are only intended for observation token predictions and are never employed in the update of the recurrent state.

This approach effectively reduces the total number of world model calls during imagination from KH to $2H$, eliminating the dependency on the number of observation tokens K. In fact, POP provides an additional generation mode that further reduces the number of sequential calls to H. We defer the details on this mode to Appendix A.1.2. Also, by using a recurrent state that summarizes long history sequences, POP improves efficiency further, as the per-token prediction cost reduces. Effectively, POP offers improved scalability at the expense of a higher overall computational cost ($(2K + 1)H$ compared to $(K + 1)H$). Our approach add to existing evidence suggesting that enhanced scalability is often favorable, even at the expense of additional computational costs, with Transformers (Vaswani et al., 2017) serving as a prominent example.

2.4. World Model Training

While applying POP during imagination is fairly straightforward, it requires modification of the training data. Consider an input trajectory segment $(\mathbf{z}_1, a_1, \dots, \mathbf{z}_T, a_T)$ sampled from the replay buffer. To make meaningful observation predictions at imagination, the model should be trained to predict $\mathbf{z}_t$ given $(\mathbf{z}_1, a_1, \dots, \mathbf{z}_{t-1}, a_{t-1}, \mathbf{u})$, for each time step t of every input segment. Hence, for every t, the input sequence should contain $\mathbf{u}$ at block t. However, replacing $\mathbf{z}_t$ with $\mathbf{u}$ in the original sequence is inadequate, as the prediction of future observations, rewards, and termination signals depends on $\mathbf{z}_t$. Thus, the standard approach of computing all outputs from the same input sequence is not viable, as in this case these two requirements contradict each other (Figure 5). The challenge then lies in devising an efficient method for computing the outputs for all time steps in parallel.

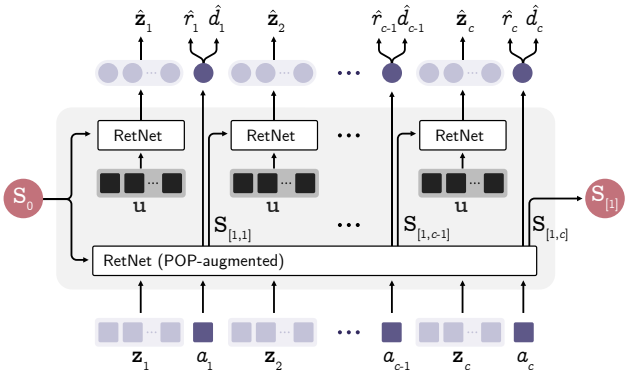

Figure 6. An illustration of the POP chunkwise forward algorithm (Alg. 1 and 2) for a single-layer model. During training, $\mathcal{M}$ computes the outputs of c observation-action blocks in parallel. Blue squares represent token inputs, while the corresponding RetNet outputs are denoted by circles. Each RetNet block represents a forward call to the same RetNet model. The bottom RetNet call uses our POP extension for computing the additional recurrent states at the end of every observation-action block (Alg. 2, lines 2-7). The top row of RetNet calls are batch-computed in parallel (Alg. 2, line 8). Finally, the output combines the observation token outputs produced by the top RetNet call with the rewards and termination outputs computed by the bottom one (Alg. 1, lines 7-9).

To tackle this challenge, we first note that each trajectory prefix can be summarized into a single recurrent state. For example, for the first input chunk $(\mathbf{z}_1, a_1, \ldots, \mathbf{z}_c, a_c)$, $(\mathbf{z}_1, a_1)$ can be summarized into $\mathbf{S}_{[1,1]}$ and $(\mathbf{z}_1, a_1, \mathbf{z}_2, a_2)$ can be summarized into $\mathbf{S}_{[1,2]}$. Here, we use the subscript $[i, j]$ to conveniently refer to the j-th block within the i-th chunk (this notation is demonstrated in Figure 3), with $\mathbf{S}_{[i,0]} = \mathbf{S}_{[i-1]}$ and $\mathbf{S}_{[i,c]} = \mathbf{S}_{[i]}$. Thus, our plan is to first compute all states $\mathbf{S}_{[i,1]}, \ldots, \mathbf{S}_{[i,c]}$ in parallel, and then predict all next observations from all $(\mathbf{S}_{[i,j]}, \mathbf{u})$ tuples.

To compute all recurrent states $\mathbf{S}_{[i,j]}$ in parallel, a two-step computation is carried. First, intermediate states $\tilde{\mathbf{S}}_{[i,j]}$ are computed in parallel for all j with

$$\tilde{\mathbf{S}}_{[i,j]} = \left(\mathbf{K}_{[i,j]} \odot \boldsymbol{\zeta}\right)^{\top} \mathbf{V}_{[i,j]}, \tag{6}$$

where $\boldsymbol{\zeta} \in \mathbb{R}^{(K+1)\times d}$ is a matrix with $\boldsymbol{\zeta}_{ij} = \eta^{K-i}$. Then, each recurrent state is computed sequentially by

$$\mathbf{S}_{[i,j]} = \tilde{\mathbf{S}}_{[i,j]} + \eta^{K+1} \mathbf{S}_{[i,j-1]}. \tag{7}$$

As the majority of the computational burden lies in the first step, the sequential computation in the second step has minimal impact on the overall speedup.

Once we have all states ready, the output of $(\mathbf{S}_{[i,j]}, \mathbf{u})$ for all $1 \leq j \leq c$ is computed in parallel. Here, we stress that the existing Retention mechanism can only perform batched input computation with recurrent states $\mathbf{S}_t$ of the same time step t. This is due to the shared positional embedding information applied to every input sequence in the batch. To overcome this, we devise a mechanism which extends RetNet to support the batched computation of the $(\mathbf{S}_{[i,j]}, \mathbf{u})$ tuples, while applying the appropriate positional encoding information. A pseudo code of our novel POP extension of RetNet is given in Algorithms 1 and 2. The latter presents the core of the mechanism (described above), while the former describes the higher level layer-by-layer computation with a final aggregation for combining the produced outputs. Figure 6 illustrates a simplified example of the POP Forward mechanism (Algorithms 1 and 2) for a single-layer model. For brevity, our pseudo code and illustrations only considers Retention layers, omitting other modules of RetNet (Appendix A.3).

To train the world model, trajectory segments of H steps from past experience are uniformly sampled from the replay buffer and translated into token sequences. These sequences are processed in chunks of c observation-action blocks to produce the modeled distributions, as depicted in Figure 6. Optimization is carried by minimizing the cross-entropy loss of the transitions and termination outputs, and the appropriate loss of the reward outputs, depending on the task. For continuous rewards, the mean-squared error loss is used while for discrete ones cross-entropy is used instead.

Algorithm 1 RetNet POP Chunkwise Forward

Input: chunk size $1 \leq c \leq H$, token embeddings $\mathbf{X}_{[i]}$ of chunk i, per-layer recurrent states $\{\mathbf{S}^l_{[i-1]}\}_{l=1}^L$.
Initialize $\mathbf{A}^0_{[i]} \leftarrow \mathbf{X}$
Initialize $\mathbf{B}^0_{[i,1]}, \ldots, \mathbf{B}^0_{[i,c]} \leftarrow \mathcal{E}_{\mathbf{u}}, \ldots, \mathcal{E}_{\mathbf{u}}$
for $l = 1$ **to** L **do**
  $\mathbf{A}^l_{[i]}, \mathbf{B}^l_{[i]}, \mathbf{S}^l_{[i]} \leftarrow \text{POPLayer}(\mathbf{A}^{l-1}_{[i]}, \mathbf{B}^{l-1}_{[i]}, \mathbf{S}^l_{[i-1]}, i)$
end for
for $j = 1$ **to** c **do**
  $\mathbf{Y}_{[i,j]} \leftarrow \text{Concat}(\mathbf{B}^L_{[i,j]}, \mathbf{A}^L_{[i,j,K+1]})$
end for
Return $\mathbf{Y}, \{\mathbf{S}^l_{[i]}\}_{l=1}^L$

Algorithm 2 POPLayer Chunkise Forward

Input: Chunk latents $\mathbf{A}_{[i]}$, observation prediction latents $\mathbf{B}_{[i]}$, recurrent state $\mathbf{S}_{[i-1]}$, chunk index i.
$\mathbf{A}_{[i]} \leftarrow \text{Retention}(\mathbf{A}_{[i]}, \mathbf{S}_{[i-1]}, i)$ (Eqn. 4)
Compute $\tilde{\mathbf{S}}_{[i,1]}, \ldots, \tilde{\mathbf{S}}_{[i,c]}$ in parallel (Eqn. 6)
for $j = 1$ **to** c **do**
  $\mathbf{S}_{[i,j]} \leftarrow \tilde{\mathbf{S}}_{[i,j]} + \eta^{K+1} \mathbf{S}_{[i,j-1]}$ (Eqn. 7)
end for
$\mathbf{S}_{[i]} \leftarrow \mathbf{S}_{[i,c]}$
$\mathbf{B}_{[i,j]} \leftarrow \text{Retention}(\mathbf{B}_{[i,j]}, \mathbf{S}_{[i,j-1]}, [i, j])$ in parallel for $j = 1, \ldots, c$ (Eqn. 4)
Return $\mathbf{A}_{[i]}, \mathbf{B}_{[i]}, \mathbf{S}_{[i]}$

Table 1. Mean agent returns on the 26 games of the Atari 100k benchmark followed by averaged human-normalized performance metrics. Each game score is computed as the average of 5 runs with different seeds, where the score of each run is computed as the average over 100 episodes sampled at the end of training. Bold face and underscores mark the highest score among token-based methods and among all baselines, respectively.

			Non-Token-Based				Token-Based	
Game	Random	Human	SimPLe	DreamerV3	TWM	STORM	IRIS	REM (ours)
Alien	227.8	7127.7	616.9	959.4	674.6	983.6	420.0	**607.2**
Amidar	5.8	1719.5	74.3	139.1	121.8	204.8	**143.0**	95.3
Assault	222.4	742.0	527.2	705.6	682.6	801.0	1524.4	**1764.2**
Asterix	210.0	8503.3	1128.3	932.5	1116.6	1028.0	853.6	**1637.5**
BankHeist	14.2	753.1	34.2	648.7	466.7	641.2	**53.1**	19.2
BattleZone	2360.0	37187.5	4031.2	12250.0	5068.0	13540.0	**13074.0**	11826.0
Boxing	0.1	12.1	7.8	78.0	77.5	79.7	70.1	**87.5**
Breakout	1.7	30.5	16.4	31.1	20.0	15.9	83.7	**90.7**
ChopperCommand	811.0	7387.8	979.4	410.0	1697.4	1888.0	1565.0	**2561.2**
CrazyClimber	10780.5	35829.4	62583.6	97190.0	71820.4	66776.0	59324.2	**76547.6**
DemonAttack	152.1	1971.0	208.1	303.3	350.2	164.6	2034.4	**5738.6**
Freeway	0.0	29.6	16.7	0.0	24.3	0.0	31.1	**32.3**
Frostbite	65.2	4334.7	236.9	909.4	1475.6	1316.0	**259.1**	240.5
Gopher	257.6	2412.5	596.8	3730.0	1674.8	8239.6	2236.1	**5452.4**
Hero	1027.0	30826.4	2656.6	11160.5	7254.0	11044.3	**7037.4**	6484.8
Jamesbond	29.0	302.8	100.5	444.6	362.4	509.0	**462.7**	391.2
Kangaroo	52.0	3035.0	51.2	4098.3	1240.0	4208.0	**838.2**	467.6
Krull	1598.0	2665.5	2204.8	7781.5	6349.2	8412.6	**6616.4**	4017.7
KungFuMaster	258.5	22736.3	14862.5	21420.0	24554.6	26182.0	21759.8	**25172.2**
MsPacman	307.3	6951.6	1480.0	1326.9	1588.4	2673.5	**999.1**	962.5
Pong	-20.7	14.6	12.8	18.4	18.8	11.3	14.6	**18.0**
PrivateEye	24.9	69571.3	35.0	881.6	86.6	7781.0	**100.0**	99.6
Qbert	163.9	13455.0	1288.8	3405.1	3330.8	4522.5	**745.7**	743.0
RoadRunner	11.5	7845.0	5640.6	15565.0	9109.0	17564.0	9614.6	**14060.2**
Seaquest	68.4	42054.7	683.3	618.0	774.4	525.2	661.3	**1036.7**
UpNDown	533.4	11693.2	3350.3	7567.1	15981.7	7985.0	3546.2	**3757.6**
#Superhuman (↑)	0	N/A	1	9	8	9	10	**12**
Mean (↑)	0.000	1.000	0.332	1.124	0.956	1.222	1.046	**1.222**
Median (↑)	0.000	1.000	0.134	0.485	0.505	0.425	**0.289**	0.280
IQM (↑)	0.000	1.000	0.130	0.487	0.459	0.561	0.501	**0.673**
Optimality Gap (↓)	1.000	0.000	0.729	0.510	0.513	0.472	0.512	**0.482**

3. Experiments

We follow most prior works on world models and evaluate REM on the widely-recognized Atari 100K benchmark (Kaiser et al., 2020) for sample-efficient reinforcement learning. The Atari 100K benchmark considers a subset of 26 Atari games. For each game, the agent is limited to 100K interaction steps, corresponding to 400K game frames due to the standard frame-skip of 4. In total, this amounts to roughly 2 hours of gameplay. To put in perspective, the original Atari benchmark allows agent to collect 200M steps, that is, 500 times more experience.

Experimental Setup The full details of the architectures and hyper-parameters used in our experiments are presented in Appendix A.1. Notably, our tokenizer uses $K = 64$ (i.e., a grid of 8×8 latent tokens per observation), whereas IRIS uses only $K = 4 \times 4 = 16$. To ensure a meaningful comparison of the run times of REM and IRIS, REM's configuration was chosen so that the amount of computation carried by each component at each epoch remains (roughly) equivalent to that of the corresponding component in IRIS. For benchmarking agents run times, we used a workstation with an Nvidia RTX 4090 GPU. The rest of our experiments were conducted on Nvidia V100 GPUs.

Baselines Since the contributions of this paper relate to token-based approaches, and to IRIS in particular, our evaluation focuses on token-based methods. To enrich our results, as well as to facilitate future research, we have included the following additional baselines: SimPLe (Kaiser et al., 2020), DreamerV3 (Hafner et al., 2023), TWM (Robine et al., 2023), and STORM (Zhang et al., 2023). In these approaches, observations are processed as a single sequence element by the world model. Following prior works on world models, lookahead search methods such as MuZero

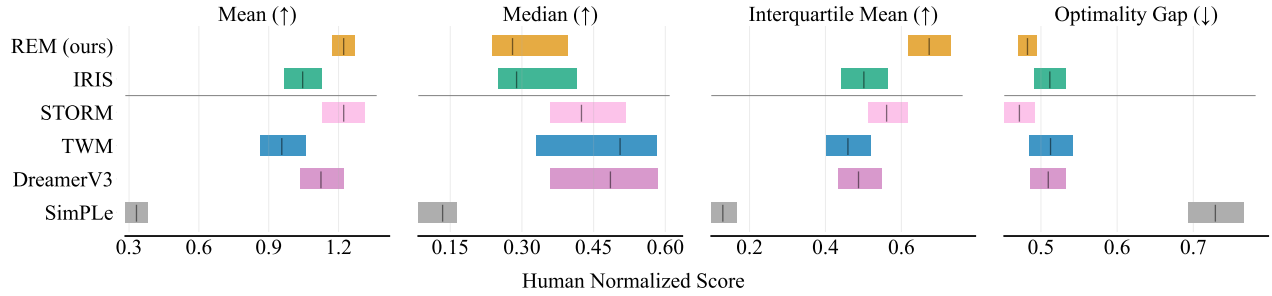

Figure 7. Atari 100K aggregated metrics with 95% stratified bootstrap confidence intervals of the mean, median, and inter-quantile mean (IQM) human-normalized scores and optimality gap (Agarwal et al., 2021). A line separates token-based methods from other baselines.

(Schrittwieser et al., 2020) and EfficientZero (Ye et al., 2021) are not included as lookahead search operates on top of a world model. Here, our aim is to improve the world model component itself.

3.1. Results

On Atari, it is standard to use human-normalized scores (HNS) (Mnih et al., 2015), calculated as $\frac{\text{agent\_score} - \text{random\_score}}{\text{human\_score} - \text{random\_score}}$, rather than raw game scores. Here, the final score of each training run is computed as an average over 100 episodes collected at the end of training. In the work of (Agarwal et al., 2021), the authors found discrepancies between conclusions drawn from point estimate statistics such as mean and median and a more thorough statistical analysis that also considers the uncertainty in the results. Adhering to their established protocol and utilizing their toolkit, we report the mean, median, and interquantile mean (IQM) human-normalized scores, and the optimality gap, with 95% stratified bootstrap confidence intervals in Figure 7. Performance profiles are presented in Figure 8. Average scores of individual games are reported in Table 1.

REM attains an IQM human normalized score of 0.673, outperforming all baselines. Additionally, REM improves over IRIS on 3 out of the 4 metrics (i.e., mean, optimality gap, and IQM), while being comparable in terms of its median score. Remarkably, REM achieves superhuman performance on 12 games, more than any other baseline (Table 1). REM also exhibits state-of-the-art scores on several games, including Assault, Boxing, and Chopper Command. These findings support our empirical claim that REM performs similarly or better than previous token-based approaches while running significantly faster.

3.2. Ablation Studies

To analyze the impact of different components of our approach on REM's performance, we conduct a series of ablation studies. For each component, we compare the final algorithm to a version where the component of interest is

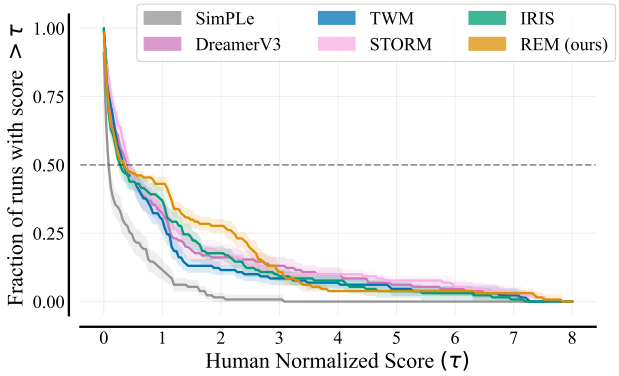

Figure 8. Performance profiles. For every human-normalized score value (x axis), each algorithm's curve shows the fraction of its runs with score grater than the given score value. The shaded area indicates pointwise 95% confidence bands based on percentile bootstrap with stratified sampling (Agarwal et al., 2021).

disabled. Due to computational resource constraints, the evaluation is performed on a subset of 8 games from the Atari 100K benchmark using 5 random seeds for each game. This subset includes games with large score differences between IRIS and REM, as we are particularly interested in studying the impact of each component in these games. Concretely, this subset includes the games "Assult", "Asterix", "Chopper Command", "Crazy Climber", "Demon Attack", "Gopher", "Krull", and "Road Runner". We performed ablation studies on the following aspects: the POP mechanism, the latent space architecture of $\mathcal{C}$ and its action inputs, the latent resolution of $\mathcal{V}$, and the observation token embeddings used by $\mathcal{M}$.

The probability of improvement (Agarwal et al., 2021) and IQM human-normalized scores are presented in Figure 9. Figure 10 offers a comparison of the training times, juxtaposing REM with its efficiency-related ablations.

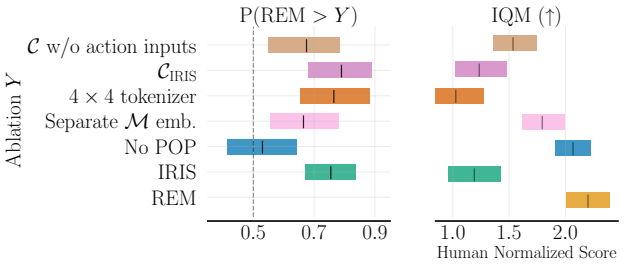

Figure 9. **Left**: The probability of improvement (Agarwal et al., 2021) shows the probability of REM outperforming each ablation on a randomly selected game from the subset of 8 games used for the ablation studies. **Right**: interquantile mean (IQM) human normalized score. Each band indicate a 95% stratified bootstrap confidence interval.

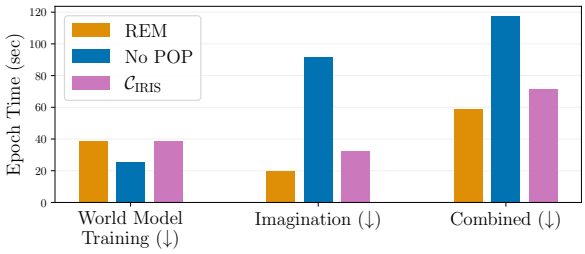

Figure 10. Epoch run time comparison between REM and two of its ablations: "No POP", and $\mathcal{C}_{\text{IRIS}}$.

Analyzing POP To study the impact of POP on REM's performance, we replaced the POP-augmented RetNet of $\mathcal{M}$ with a vanilla RetNet. In this version, denoted as "No POP", the prediction of next observation tokens is performed sequentially token-by-token, as done in IRIS.

Our results suggest that POP retains the agent's performance (Figure 9) while significantly reducing the overall computation time (Figure 10). In Appendix A.4, we provide additional results indicating that the world model's performance are also retained. POP achieves lower total computation time by expediting the actor-critic learning phase, despite the increased computational cost implied by the observation prediction tokens.

Actor-Critic Architecture and Action Inputs For $\mathcal{C}$, we considered an incremental ablation. First, we replaced the architecture of REM's controller $\mathcal{C}$ with that of IRIS (denoted "$\mathcal{C}_{\text{IRIS}}$"). In contrast to REM, this version processes fully reconstructed pixel frames and does not incorporate action inputs. Formally, $\mathcal{C}_{\text{IRIS}}$ models $\pi(a_t|\hat{\mathbf{o}}_{\leq t}), V^{\pi}(\hat{\mathbf{o}}_{\leq t})$. In the second ablation, REM was modified so that only the action inputs of $\mathcal{C}$ were disabled. This ablation corresponds to $\pi(a_t|\hat{\mathbf{z}}_{\leq t}), V^{\pi}(\hat{\mathbf{z}}_{\leq t})$.

Our findings indicate that both the latent codes based architecture and the added action inputs contribute to the final performance of REM (Figure 9). Additionally, the latent codes based architecture of $\mathcal{C}$ leads to reduced computational overhead and shorter actor-critic learning times (Figure 10).

Tokenizer Resolution Here, we compare REM to a version with a reduced latent resolution of 4×4, similar to that of IRIS. The results in Figure 9 provides clear evidence that the latent resolution of the tokenizer has a significant impact on the agent's performance. Our results demonstrates that POP enables REM to utilize higher latent resolutions while incurring shorter computation times than prior token-based approaches.

World Model Embeddings In REM, $\mathcal{M}$ translates observation tokens to embedding vectors using the embedding table $\mathcal{E}$ learned by $\mathcal{V}$. These embeddings encode the visual information as learned by $\mathcal{V}$. In contrast, IRIS maintains a separate embedding table learned by the world model for that purpose. Here, the results in Figure 9 provide empirical evidence indicating that leveraging this encoded visual information leads to improved performance. In Appendix A.4, we provide additional evidence suggesting that the world model's next-observation predictions are also improved.

4. Related Work

Model-based reinforcement learning (RL), with its roots in the tabular setting (Sutton, 1991), has been a focus of extensive research in recent decades. The deep RL agent of (Ha & Schmidhuber, 2018) leveraged an LSTM (Hochreiter & Schmidhuber, 1997) sequence model with a VAE (Kingma & Welling, 2014) to model the dynamics in visual environments, demonstrating that successful policies can be learned entirely from simulated data. This approach, commonly known as world models, was later applied to Atari games (Kaiser et al., 2020) with the PPO (Schulman et al., 2017) RL algorithm. Later, a series of works (Hafner et al., 2020; 2021; 2023) proposed the Dreamer algorithms, which are based on a recurrent state space model (RSSM) (Hafner et al., 2019) to model dynamics. The latest DreamerV3 was evaluated on a variety of challenging environments, providing further evidence of the promising potential of world models. In contrast to token-based approaches, where each token serves as a standalone sequence element, Dreamer encodes each frame as a vector of categorical variables, which are processed at once by the RSSM.

Following the success of the Transformer architecture (Vaswani et al., 2017) in language modeling (Brown et al., 2020), and motivated by their favorable scaling properties compared to RNNs, Transformer were recently explored in

RL (Chen et al., 2021; Parisotto et al., 2020; Reed et al., 2022; Shridhar et al., 2023). World model approaches also adopted the Transformer architecture. (Micheli et al., 2023) blazed the trail for token-based world models with IRIS, representing agent trajectories as language-like sequences. By treating each observation as a sequence, its Transformer-based world model gains an explicit sub-observation attention resolution. Despite IRIS's high performance, its imagination bottleneck results in a substantial disadvantage.

In addition to IRIS, non-token-based world models driven by Transformers were proposed. TWM (Robine et al., 2023) utilizes the Transformer-XL architecture (Dai et al., 2020) and a non-uniform data sampling. STORM (Zhang et al., 2023) proposes an efficient Transformer based world model agent which sets state-of-the-art result for the Atari 100K benchmark. STORM has a significantly smaller 2-layer Transformer compared to the 10-layer models of TWM and IRIS, demonstrating drastically reduced training times and improved agent performance.

5. Conclusions

In this work, we presented a novel parallel observation prediction (POP) mechanism augmenting Retention networks with a dedicated forward mode to improve the efficiency of token-based world models (TBWMs). POP effectively solves the imagination bottleneck of TBWMs and enables them to deal with longer observation sequences. Additionally, we introduced REM, a TBWM agent equipped with POP. REM is the first world model agent driven by the RetNet architecture. Empirically, we demonstrated the superiority of REM over prior TBWMs on the Atari 100K benchmark, rendering REM competitive with the state-of-the-art, both in terms of agent performance and overall run time.

Our work opens up many promising avenues for future research by making TBWMs practical and cost-efficient. One such direction could be to explore a modification of REM where the recurrent state of the world model summarizes the entire history of the agent. Similarly, a history-preserving RetNet architecture should be considered for the controller as well. Another promising avenue would be to leverage the independent optimization of the tokenizer to enable REM to use pretrained visual perception models in environments where visual data is abundant, for example, the real world. Such perceptual models could be trained at scale, and allow REM to store only compressed observations in its replay buffer, further improving its efficiency. Lastly, token-based methods for video generation tasks can benefit from using the POP mechanism for generating entire frames in parallel conditioned on the past context. We believe that this is an exciting avenue to explore with a potentially high impact.

Acknowledgements

This project has received funding from the European Union's Horizon Europe Programme under grant agreement No. 101070568.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. Deep reinforcement learning at the edge of the statistical precipice. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 29304–29320. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/f514cec81cb148559cf475e7426eed5e-Paper.pdf.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., de Oliveira Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019. URL http://arxiv.org/abs/1912.06680.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J.,

Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S. M., Nori, H., Palangi, H., Ribeiro, M. T., and Zhang, Y. Sparks of artificial general intelligence: Early experiments with GPT-4. *CoRR*, abs/2303.12712, 2023. doi: 10.48550/ARXIV.2303.12712. URL https://doi.org/10.48550/arXiv.2303.12712.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision Transformer: Reinforcement Learning via Sequence Modeling. In *Advances in Neural Information Processing Systems*, volume 18, 2021.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-XL: Attentive language models beyond a fixed-length context. In *ACL 2019 - 57th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, 2020. doi: 10.18653/v1/p19-1285.

Devlin, J., Chang, M. W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, volume 1, 2019.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Esser, P., Rombach, R., and Ommer, B. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12873–12883, 2021.

Ha, D. and Schmidhuber, J. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems 31*, pp. 2451–2463. Curran Associates, Inc., 2018. URL https://papers.nips.cc/paper/7512-recurrent-world-models-facilitate-policy-evolution. https://worldmodels.github.io.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. In *36th International Conference on Machine Learning, ICML 2019*, volume 2019-June, 2019.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1lOTC4tDS.

Hafner, D., Lillicrap, T. P., Norouzi, M., and Ba, J. Mastering atari with discrete world models. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=0oabwyZbOu.

Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

Hendrycks, D. and Gimpel, K. Bridging nonlinearities and stochastic regularizers with gaussian error linear units, 2017. URL https://openreview.net/forum?id=Bk0MRI5lg.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Kaiser, Ł., Babaeizadeh, M., Miłos, P., Osiński, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Mohiuddin, A., Sepassi, R., Tucker, G., and Michalewski, H. Model based reinforcement learning for atari. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1xCPJHtDB.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 2014.

Li, T., Chang, H., Mishra, S., Zhang, H., Katabi, D., and Krishnan, D. Mage: Masked generative encoder to unify representation learning and image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2142–2152, 2023.

Micheli, V., Alonso, E., and Fleuret, F. Transformers are sample-efficient world models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/pdf?id=vhFu1Acb0xb.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.

Parisotto, E., Song, F., Rae, J., Pascanu, R., Gulcehre, C., Jayakumar, S., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., Botvinick, M., Heess, N., and

Hadsell, R. Stabilizing transformers for reinforcement learning. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 7487–7498. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/parisotto20a.html.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions, 2018. URL https://openreview.net/forum?id=SkBYYyZRZ.

Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-maron, G., Giménez, M., Sulsky, Y., Kay, J., Springenberg, J. T., Eccles, T., Bruce, J., Razavi, A., Edwards, A., Heess, N., Chen, Y., Hadsell, R., Vinyals, O., Bordbar, M., and de Freitas, N. A generalist agent. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL https://openreview.net/forum?id=1ikK0kHjvj. Featured Certification, Outstanding Certification.

Robine, J., Höftmann, M., Uelwer, T., and Harmeling, S. Transformer-based world models are happy with 100k interactions. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=TdBaDGCpjly.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., and Silver, D. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, dec 2020. ISSN 14764687. doi: 10.1038/s41586-020-03051-4. URL https://www.nature.com/articles/s41586-020-03051-4.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017. URL https://api.semanticscholar.org/CorpusID:28695052.

Shridhar, M., Manuelli, L., and Fox, D. Perceiver-actor: A multi-task transformer for robotic manipulation. In Liu, K., Kulic, D., and Ichnowski, J. (eds.), *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 785–799. PMLR, 14–18 Dec 2023. URL https://proceedings.mlr.press/v205/shridhar23a.html.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016. URL http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

Sun, Y., Dong, L., Huang, S., Ma, S., Xia, Y., Xue, J., Wang, J., and Wei, F. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

Sutton, R. S. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4), 1991. ISSN 0163-5719. doi: 10.1145/122344.122377.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

Touvron, H., Martin, L., Stone, K. R., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D. M., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A. S., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I. M., Korenev, A. V., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T. Llama 2: Open foundation and fine-tuned chat models. *ArXiv*, abs/2307.09288, 2023. URL https://api.semanticscholar.org/CorpusID:259950998.

van den Oord, A., Vinyals, O., and kavukcuoglu, k. Neural discrete representation learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/7a98af17e63a0ac09ce2e96d03992fbc-Paper.pdf.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782), 2019. ISSN 14764687. doi: 10.1038/s41586-019-1724-z.

Ye, W., Liu, S., Kurutach, T., Abbeel, P., and Gao, Y. Mastering Atari Games with Limited Data. In *Advances in Neural Information Processing Systems*, volume 30, 2021.

Zhang, W., Wang, G., Sun, J., Yuan, Y., and Huang, G. STORM: Efficient stochastic transformer based world models for reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=WxnrX42rnS.

A. Appendix

A.1. Models and Hyperparameters

Tables 2 and 3 detail hyperparameters of the optimization and environment, as well as hyperparameters shared by multiple components.

Table 2. Shared Hyperparameters

Description	Symbol	Value
Horizon	H	10
Tokens per observation	K	64
Tokenizer vocabulary size	N	512
Epochs	-	600
Experience collection epochs	-	500
Environment steps per epoch	-	200
Collection epsilon-greedy	-	0.01
Eval sampling temperature	-	0.5
Optimizer	-	AdamW
AdamW β_1	-	0.9
AdamW β_2	-	0.999
Frame resolution	-	64×64
Frame Skip	-	4
Max no-ops (train, test)	-	(30, 1)
Max episode steps (train, test)	-	(20K, 108K)
Terminate on live loss (train, test)	-	(No, Yes)

Table 3. Per-Component Hyperparameters

Description	Symbol	Tokenizer	World Model	Actor-Critic
Learning rate	-	0.0001	0.0002	0.0001
Batch size	-	128	64	128
Gradient Clipping Threshold	-	10	100	3
Start after epochs	-	5	25	50
Training Steps per epoch	-	200	200	100
AdamW Weight Decay	-	0.01	0.05	0.01

A.1.1. TOKENIZER ($\mathcal{V}$)

Tokenizer Architecture Our tokenizer is based on the implementation of VQ-GAN (Esser et al., 2021). However, we simplified the architectures of the encoder and decoder networks. A description of the architectures of the encoder and decoder networks can be found in table 4.

Tokenizer Learning Following IRIS (Micheli et al., 2023), our tokenizer is a VQ-VAE (van den Oord et al., 2017) based on the implementation of (Esser et al., 2021) (without the discriminator). The training objective is given by

$$\mathcal{L}(E, D, \mathcal{E}) = \|x - D(z)\|_1 + \|\text{sg}(E(x)) - \mathcal{E}(z)\|_2^2 + \|\text{sg}(\mathcal{E}(z)) - E(x)\|_2^2 + \mathcal{L}_{\text{perceptual}}(x, D(z)) \tag{8}$$

where E and D are the encoder and decoder models, respectively, and $\text{sg}(\cdot)$ is the stop-gradient operator. The first term on the right hand side of Equation 8 above is the reconstruction loss, the second and third terms correspond to the commitment loss, and the last term is the perceptual loss.

Table 4. The encoder and decoder architectures. "Conv(a,b,c)" represents a convolutional layer with kernel size $a \times a$, stride of b and padding c. A value of $c =$ Asym. represents an asymmetric padding where a padding of 1 is added only to the right and bottom ends of the image tensor. "GN" represents a GroupNorm operator with 8 groups, $\epsilon = 1e-6$ and learnable per-channel affine parameters. SiLU is the Sigmoid Linear Unit activation (Hendrycks & Gimpel, 2017; Ramachandran et al., 2018). "Interpolate" uses PyTorch's interpolate method with scale factor of 2 and the "nearest-exact" mode.

Module	Output Shape
Encoder	
Input	$3 \times 64 \times 64$
Conv(3, 1, 1)	$32 \times 64 \times 64$
EncoderBlock1	$64 \times 32 \times 32$
EncoderBlock2	$128 \times 16 \times 16$
EncoderBlock3	$256 \times 8 \times 8$
GN	$256 \times 8 \times 8$
SiLU	$256 \times 8 \times 8$
Conv(3, 1, 1)	$256 \times 8 \times 8$
EncoderBlock	
Input	$c \times h \times w$
GN	$c \times h \times w$
SiLU	$c \times h \times w$
Conv(3, 2, Asym.)	$2c \times \frac{h}{2} \times \frac{w}{2}$
Conv(3, 1, 1)	$2c \times \frac{h}{2} \times \frac{w}{2}$
Decoder	
Input	$256 \times 8 \times 8$
Conv(3, 1, 1)	$256 \times 8 \times 8$
DecoderBlock1	$128 \times 16 \times 16$
DecoderBlock2	$64 \times 32 \times 32$
DecoderBlock3	$32 \times 64 \times 64$
GN	$32 \times 64 \times 64$
SiLU	$32 \times 64 \times 64$
Conv(3, 1, 1)	$3 \times 64 \times 64$
DecoderBlock	
Input	$c \times h \times w$
GN	$c \times h \times w$
SiLU	$c \times h \times w$
Interpolate	$c \times 2h \times 2w$
Conv(3, 1, 1)	$\frac{c}{2} \times 2h \times 2w$
Conv(3, 1, 1)	$\frac{c}{2} \times 2h \times 2w$

A.1.2. Retentive World Model ($\mathcal{M}$)

The hyperparameters of $\mathcal{M}$ are presented in Table 5.

Implementation Details We use the "Yet-Another-RetNet" RetNet implementation[1], as its code is simple and convenient while its performance remain competitive with the official implementation in terms of run time and efficiency.

Originally, the IRIS algorithm provides the world model with a single observation to make forward predictions. Our implementation considers a context of two frames for making forward predictions.

POP Observation-Generation Modes The simpler observation generation mode presented in Section 2.3 requires two sequential world model calls to generate the next observation. The first call consumes the previous observation-action block to compute the recurrent state at the current time step, while the second call uses $\mathbf{u}$ to generate the next observation tokens. Note that the next observation tokens sampled in the second call have not been processed by the world model at this point. To incorporate these tokens into the recurrent state, an additional world model call is required. Here, the cost induced by the

[1] https://github.com/fkodom/yet-another-retnet

Table 5. The world model hyper-parameters.

Description	Symbol	Value
Number of layers	-	5
Number of Retention heads	-	4
Embedding dimension	d	256
Dropout	-	0.1
RetNet feed-forward dimension	-	1024
RetNet LayerNorm epsilon	-	1e-6
Blocks per chunk	c	3
World model context length	-	2

first world model call is $K+1$, while the second call costs K.

Alternatively, it is also possible to combine these two calls into one by concatenating the previous observation-action block and $\mathbf{u}$. However, to avoid having $\mathbf{u}$ incorporated in the recurrent state computed by this call, a modified forward call should be used. Concretely, the resulting recurrent state should only summarize the previous observation-action block, neglecting the suffix $\mathbf{u}$. In practice, we use the backbone of the POP chunkwise forward mode (Alg. 1) for this computation. This alternative mode induces only H sequential world model calls, while each call processes $2K+1$. Hence, this alternative reduces the number of sequential calls while maintaining the same total cost.

In practice, the optimal mode to use depends on the configuration, model sizes, and hardware. In our configuration, we opted for a larger batch size for the imagination phase. Hence, we found the first (simpler) mode to be slightly more efficient in this case. However, the second mode could be more efficient in other settings.

Table 6. A comparison between POP and the "No POP" ablation in terms of their computational costs at training and their number of sequential model forward calls at inference (imagination). For brevity, we only consider observation-prediction related costs, neglecting costs related to the processing of action tokens. POP provides two modes of operation during imagination. Here, we consider the costs for a single input sequence.

Algorithm	Observation Prediction Cost	World Model Training Cost	Imagination Sequential Calls	Imagination Cost per Call
POP (default mode)	$2K$	$2KH$	$2H$	K
POP (alternative mode)	$2K$	$2KH$	H	$2K$
No POP	K	KH	KH	1

A.1.3. CONTROLLER ($\mathcal{C}$)

Actor-Critic Learning Our learning algorithm follows IRIS and Dreamer (Hafner et al., 2020; 2021; Micheli et al., 2023), which uses λ-returns defined recursively as

$$G_t = \begin{cases} \hat{r}_t + \gamma(1-\hat{d}_t)\left((1-\lambda)V^\pi(\hat{\mathbf{z}}_{t+1}) + \lambda G_{t+1}\right) & t < H \\ V^\pi(\hat{\mathbf{z}}_H) & t = H \end{cases}$$

where V^π is the value network learned by the critic, and $(\hat{\mathbf{z}}_0, a_0, \hat{r}_0, \hat{d}_0, \ldots, \hat{\mathbf{z}}_{H-1}, a_{H-1}, \hat{r}_{H-1}, \hat{d}_{H-1}, \hat{\mathbf{z}}_H)$ is a trajectory obtained through world model imagination.

To optimize V^π, the following loss is minimized:

$$\mathcal{L}_{V^\pi} = \mathbb{E}_\pi\left[\sum_{t=0}^{H-1} V^\pi(\hat{\mathbf{z}}_t) - \mathrm{sg}(G_t)^2\right]$$

The policy optimization follows a simple REINFORCE (Sutton & Barto, 2018) objective, with V^π used as a baseline for

variance reduction. The objective is given by

$$\mathcal{L}_\pi = -\mathbb{E}_\pi[\sum_{t=0}^{H-1} \log(\pi(a_t|\hat{\mathbf{z}}_1, a_1, \ldots, \hat{\mathbf{z}}_{t-1}, a_{t-1}, \hat{\mathbf{z}}_t))\text{sg}(G_t - V^\pi(\hat{\mathbf{z}}_t)) + \alpha\mathcal{H}(\pi(a_t|\hat{\mathbf{z}}_1, a_1, \ldots, \hat{\mathbf{z}}_{t-1}, a_{t-1}, \hat{\mathbf{z}}_t))]$$

where α The values of the hyperparameters used in our experiments are detailed in Table 7

Table 7. Actor-Critic Hyperparameters.

Description	Symbol	Value
Discount factor	γ	0.995
λ-return	λ	0.95
Entropy loss weight	α	0.001

Table 8. The actor-critic observation representation architecture.

Module	Output Shape
Input	$256 \times 8 \times 8$
Conv(3, 1, 1)	$128 \times 8 \times 8$
SiLU	$128 \times 8 \times 8$
Conv(3, 1, 1)	$64 \times 8 \times 8$
SiLU	$64 \times 8 \times 8$
Flatten	4096
Linear	512
SiLU	512

Agent Architecture The architecture of the agent module comprises of a shared backbone and two linear maps for the actor and critic heads, respectively. The shared backbone first maps the input to a latent representation which takes the form of a 512-dimensional vector. For action token inputs, a learned embedding table is used to map the token to its latent representation. For observation inputs, the K tokens are first mapped to their corresponding code vectors learned by the tokenizer and reshaped according to their original spatial order. Then, the resulting tensor is processed by a convolutional neural network followed by a fully connected network. The architecture details of these networks are presented in Table 8. Lastly, a long-short term memory (LSTM) (Hochreiter & Schmidhuber, 1997) network of dimension 512 maps the processed input vector to an history-dependant latent vector, which serves as the output of the shared backbone.

Action Dependant Actor Critic In the IRIS algorithm, the actor and critic networks share an LSTM (Hochreiter & Schmidhuber, 1997) backbone and model $\pi(a_t|\mathbf{o}_{\leq t}), V^\pi(\mathbf{o}_{\leq t})$. Notice that the output of the policy models the *distribution* of actions at step t. Importantly, the model has no information about the sampled actions. In REM, the input of $\mathcal{C}$ contains the sampled actions, i.e., our algorithm models $\pi(a_t|\hat{\mathbf{z}}_1, a_1, \ldots, \hat{\mathbf{z}}_{t-1}, a_{t-1}, \hat{\mathbf{z}}_t), V^\pi(\hat{\mathbf{z}}_1, a_1, \ldots, \hat{\mathbf{z}}_{t-1}, a_{t-1}, \hat{\mathbf{z}}_t)$.

A.2. REM Algorithm

Here, we present a pseudo-code of REM. The high-level loop is presented in Algorithm 3, while the pseudo-codes of the training of each component are presented in algorithms 4-7.

Algorithm 3 REM Training Overview

Input:
repeat
  `collect_experience()` (Alg. 4)
  `train_V()` (Alg. 5)
  `train_M()` (Alg. 6)
  `train_C()` (Alg. 7)
until stopping criterion is met

Algorithm 4 `collect_experience`

Input:
$\mathbf{o}_1 \leftarrow$ `env.reset()`
for $t = 1$ **to** T **do**
  $\mathbf{z}_t \leftarrow \mathcal{V}_{\text{Enc}}(\mathbf{o}_t)$
  $a_t \sim \pi(a_t|\mathbf{z}_1, a_1, \ldots, \mathbf{z}_{t-1}, a_{t-1}, \mathbf{z}_t)$
  $\mathbf{o}_{t+1}, r_t, d_t \leftarrow$ `env.step`(a_t)
  if $d_t = 1$ **then**
    $\mathbf{o}_{t+1} \leftarrow$ `env.reset()`
  end if
end for
`replay_buffer.store`$(\{\mathbf{o}_t, a_t, r_t, d_t\}_{t=1}^{T})$

Algorithm 5 `train_V`

$\mathbf{o} \leftarrow$ `replay_buffer.sample_obs()`
$\mathbf{z} \leftarrow \mathcal{V}_{\text{Enc}}(\mathbf{o})$
$\hat{\mathbf{o}} \leftarrow \mathcal{V}_{\text{Dec}}(\mathbf{z})$
Compute loss (Eqn. 8)
Update $\mathcal{V}$

Algorithm 6 `train_M`

$\{\mathbf{o}_t, a_t, r_t, d_t\}_{t=1}^{H} \leftarrow$ `replay_buffer.sample()`
for $t = 1$ **to** H **do**
  $\mathbf{z}_t, \mathbf{h}_t \leftarrow \mathcal{V}_{\text{Enc}}(\mathbf{o}_t)$
  $\mathbf{a}_t \leftarrow \mathcal{M}$`.embed_action`$(a_t)$
end for
$\mathbf{X} \leftarrow (\mathbf{h}_1, \mathbf{a}_1, \ldots, \mathbf{h}_H, \mathbf{a}_H)$
$\mathbf{S}_0^1, \ldots, \mathbf{S}_0^L \leftarrow 0, \ldots, 0$
for $i = 1$ **to** $\lceil \frac{H}{c} \rceil$ **do**
  $\mathbf{Y}_{[i]}, \{\mathbf{S}_i^l\}_{l=1}^{L} \leftarrow$ `POP_forward`$(\mathbf{X}_{[i]}, \{\mathbf{S}_{[i-1]}^l\}_{l=1}^{L})$ (Alg. 1)
end for
$(\hat{\mathbf{z}}_1, \ldots, \hat{\mathbf{z}}_H) \leftarrow \mathcal{M}$`.obs_pred_head`$(\mathbf{Y}$`[:, :-1]`$)$
$(\hat{r}_1, \hat{d}_1, \ldots, \hat{r}_H, \hat{d}_H) \leftarrow \mathcal{M}$`.reward_done_head`$(\mathbf{Y}$`[:, -1]`$)$
Compute Losses and update $\mathcal{M}$

Algorithm 7 `train_C`

$\{\mathbf{o}_t, a_t, r_t, d_t\}_{t=1}^{H} \leftarrow$ `replay_buffer.sample()`
for $t = 1$ **to** H **do**
  $\mathbf{z}_t, \mathbf{h}_t \leftarrow \mathcal{V}_{\text{Enc}}(\mathbf{o}_t)$
end for
$\mathbf{S} \leftarrow 0$
$c \leftarrow 1$
Initialize context $\tau \leftarrow (\mathbf{z}_1, a_1, \ldots, \mathbf{z}_H)$
for $t' = H + 1$ **to** $2H$ **do**
  $a_{t'} \sim \pi(a_{t'} | \mathbf{z}_1, a_1, \ldots, \mathbf{z}_{t'})$
  $V_{t'} \leftarrow V(\mathbf{z}_1, a_1, \ldots, \mathbf{z}_{t'})$
  $\mathbf{Y}, \mathbf{S} \leftarrow \mathcal{M}$`.retnet_chunkwise_forward`$((\tau, a_{t'}), \mathbf{S}, t')$
  $r_{t'}, d_{t'} \sim \mathcal{M}$`.reward_done_head`$(\mathbf{Y}$`[-1]`$)$
  $\mathbf{Y}, \_ \leftarrow \mathcal{M}$`.retnet_chunkwise_forward`$(\mathbf{u}, \mathbf{S}, t' + 1)$
  $\mathbf{z}_{t'+1} \sim \mathcal{M}$`.obs_pred_head`$(\mathbf{Y}$`[:-1]`$)$
  $\tau \leftarrow \mathbf{z}_{t'+1}$
end for
Update π, V (detailed in Section A.1.3)

A.3. Retentive Networks

In this section, we give detailed information regarding the RetNet architecture for the completeness of this paper. For convenience reasons, we defer to the notations of (Sun et al., 2023), rather than the notation presented in

(Sun et al., 2023) is a recent alternative to Transformers (Vaswani et al., 2017). It is highly parallelizable, has lower cost inference than Transformers, and is empirically claimed to perform competitively on language modelling tasks. The RetNet model is a stack of L identical layers. Here, we denote the output of the l-th layer by $\mathbf{Y}^l$. Given an embedded input sequence $\mathbf{X} = \mathbf{Y}^0 \in \mathbb{R}^{m \times d}$ of m d-dimensional vectors, each RetNet layer can be described as

$$\mathbf{X}^l = \mathrm{MSR}(\mathrm{LN}(\mathbf{Y}^l)) + \mathbf{Y}^l \tag{9}$$

$$\mathbf{Y}^{l+1} = \mathrm{FFN}(\mathrm{LN}(\mathbf{X}^l)) + \mathbf{X}^l \tag{10}$$

where $\mathrm{LN}(\cdot)$ is layer-norm (Ba et al., 2016), $\mathrm{FFN}(\mathbf{X}) = \mathrm{gelu}(\mathbf{X}\mathbf{W}_1)\mathbf{W}_2$ is a feed-forward network (FFN), and $\mathrm{MSR}(\cdot)$ is a multi-scale retention (MSR) module with multiple Retention heads. The output of the RetNet model is given by $\mathrm{RetNet}(\mathbf{Y}^0) = \mathbf{Y}^L$.

As presented in the main text, the chunkwise equations are

$$\mathbf{S}_{[i]} = (\mathbf{K}_{[i]} \odot \boldsymbol{\zeta})^\top \mathbf{V}_{[i]} + \eta^B \mathbf{S}_{[i-1]}$$

$$\mathbf{Y}_{[i]} = \left(\mathbf{Q}_{[i]}\mathbf{K}_{[i]}^\top \odot \mathbf{D}\right)\mathbf{V}_{[i]} + (\mathbf{Q}_{[i]}\mathbf{S}_{[i-1]}) \odot \boldsymbol{\xi}$$

where $\mathbf{Q} = (\mathbf{X}\mathbf{W}_Q) \odot \Theta$, $\mathbf{K} = (\mathbf{X}\mathbf{W}_K) \odot \bar{\Theta}$, $\mathbf{V} = \mathbf{X}\mathbf{W}_V$, and $\boldsymbol{\xi} \in \mathbb{R}^{B \times d}$ is a matrix with $\boldsymbol{\xi}_{ij} = \eta^{i+1}$. Here, $\Theta_n = e^{in\theta}$, $\mathbf{D}_{n,m} = \begin{cases} \eta^{n-m} & n \geq m \\ 0 & n < m \end{cases}$, and $\theta, \eta \in \mathbb{R}^d$,

where η is an exponential decay factor, and the matrices $\Theta, \bar{\Theta} \in \mathbb{C}^{m \times d}$ are for relative position encoding, and $\mathbf{D} \in \mathbb{R}^{B \times B}$ combines an auto-regressive mask with the temporal decay factor η.

In each RetNet layer, $h = \frac{d}{d_{\text{head}}}$ heads are used, where d_{head} is the dimension of each head. Head Retention head uses different parameters W_K, W_Q, W_V. Additionally, Retention head uses a different value of η. Among different RetNet layers, the values of η are fixed. Each layer is defined as follows:

$$\eta = 1 - 2^{-5-\text{arange}(0,h)} \in \mathbb{R}^h$$

$$\text{head}_i = \text{Retention}(X, \eta_i)$$

$$Y = \text{GroupNorm}_h(\text{Concat}(\text{head}_1, \ldots, \text{head}_h))$$

$$\text{MSR}(X) = (\text{swish}(XW_G) \odot Y)W_O$$

where $W_G, W_O \in \mathbb{R}^{d \times d}$ are learnable parameters.

A.4. Additional Results

In addition to comparing the run times of REM and IRIS, we also conducted a comparison to an improved version of IRIS that uses REM's configurations. These results are presented in Figure 11. These results clearly show the effectiveness of our novel POP mechanism.

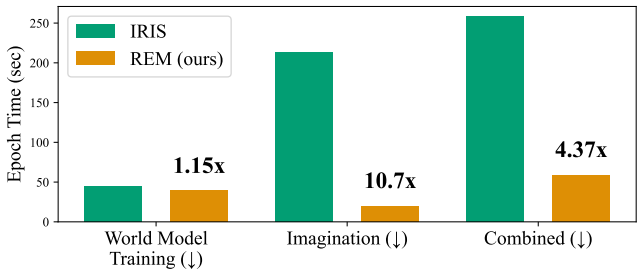

Figure 11. A comparison between the run times of REM and an improved version of IRIS that uses REM's configurations (detailed in A.1) during the world model training and imagination phases (actor-critic training).

The probability of improvement results from our Atari 100K benchmark experiment are presented in Figure 12. Importantly, REM outperforms previous token-based methods, namely, IRIS, while competitive with all baselines except STORM on this metric. We highlight that our main contributions address the computational bottleneck of token-based methods, and thus we focus on comparing REM to these approaches.

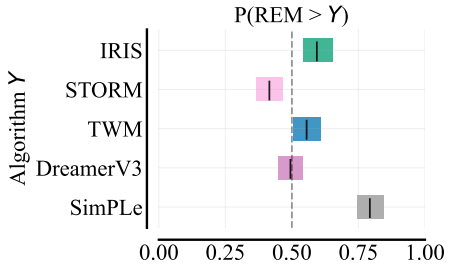

Figure 12. The probability of improvement (Agarwal et al., 2021) shows the probability of REM outperforming each baseline on a randomly selected game from the 26 games of Atari 100K with 95% stratified bootstrap confidence intervals.

The complete set of ablation results are presented in Figure 13 and Table 9. The performance profiles for the ablations are presented in Figure 14.

Ablations World Model Observation Prediction Losses To investigate the contribution of each ablation to the quality of world model observation predictions, we measured the corresponding loss values during training and during test episodes with a frequency of 50 epochs. The results are presented in Figure 15, including results for each of the 8 games used in our ablation studies.

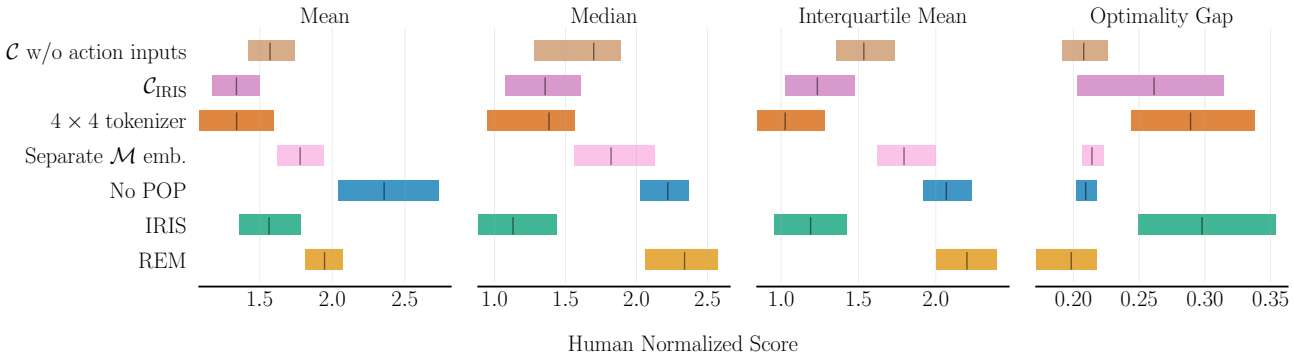

Figure 13. Aggregated metrics with 95% stratified bootstrap confidence intervals of the mean, median, and inter-quantile mean (IQM) human-normalized scores and optimality gap (Agarwal et al., 2021) for each ablation on a subset of 8 games from the Atari 100K benchmark. The results are based on 5 random seeds.

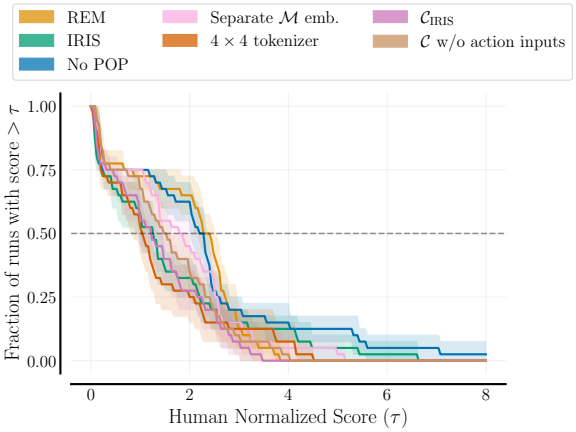

Figure 14. The Performance profiles of the ablations. For every human-normalized score value (x axis), each algorithm's curve shows the fraction of its runs with score grater than the given score value. The shaded area indicates pointwise 95% confidence bands based on percentile bootstrap with stratified sampling (Agarwal et al., 2021). The results of each algorithm and each game from the subset of 8 Atari games used in our ablations are based on 5 random seeds

Table 9. Mean agent returns on a subset of 8 games from the Atari 100k benchmark followed by averaged human-normalized performance metrics. Each game score is computed as the average of 5 runs with different seeds, where the score of each run is computed as the average over 100 episodes sampled at the end of training. The best score on each game is indicated with bold face.

Game	Random	Human	REM	IRIS	No POP	Separate $\mathcal{M}$ emb.	4 × 4 tokenizer	$\mathcal{C}_{\text{IRIS}}$	$\mathcal{C}$ w/o action inputs
Assault	222.4	742.0	**1764.2**	1524.4	1472.2	1269.2	1288.9	1221.5	1498.5
Asterix	210.0	8503.3	1637.5	853.6	1603.4	1185.9	909.6	1376.4	**1656.3**
ChopperCommand	811.0	7387.8	**2561.2**	1565.0	1848.0	1928.3	1958.9	2517.6	2302.7
CrazyClimber	10780.5	35829.4	**76547.6**	59324.2	62964.8	74791.3	57814.7	30952.7	42441.2
DemonAttack	152.1	1971.0	5738.6	2034.4	**12316.0**	4389.9	3863.3	5159.0	5827.0
Gopher	257.6	2412.5	**5452.4**	2236.1	5338.4	3764.2	2174.9	2891.2	4365.3
Krull	1598.0	2665.5	4017.7	**6616.4**	5138.6	5779.9	4612.8	3866.2	3659.6
RoadRunner	11.5	7845.0	**14060.2**	9614.6	13161.6	11723.5	6161.7	11692.9	11692.9
#Superhuman (↑)	0	N/A	**6**	5	6	6	4	5	6
Mean (↑)	0.000	1.000	1.947	1.564	**2.357**	1.778	1.341	1.340	1.571
Median (↑)	0.000	1.000	**2.339**	1.130	2.221	1.821	1.384	1.357	1.699
IQM (↑)	0.000	1.000	**2.201**	1.191	2.068	1.794	1.026	1.234	1.535
Optimality Gap (↓)	1.000	0.000	**0.198**	0.298	0.209	0.214	0.289	0.261	0.208

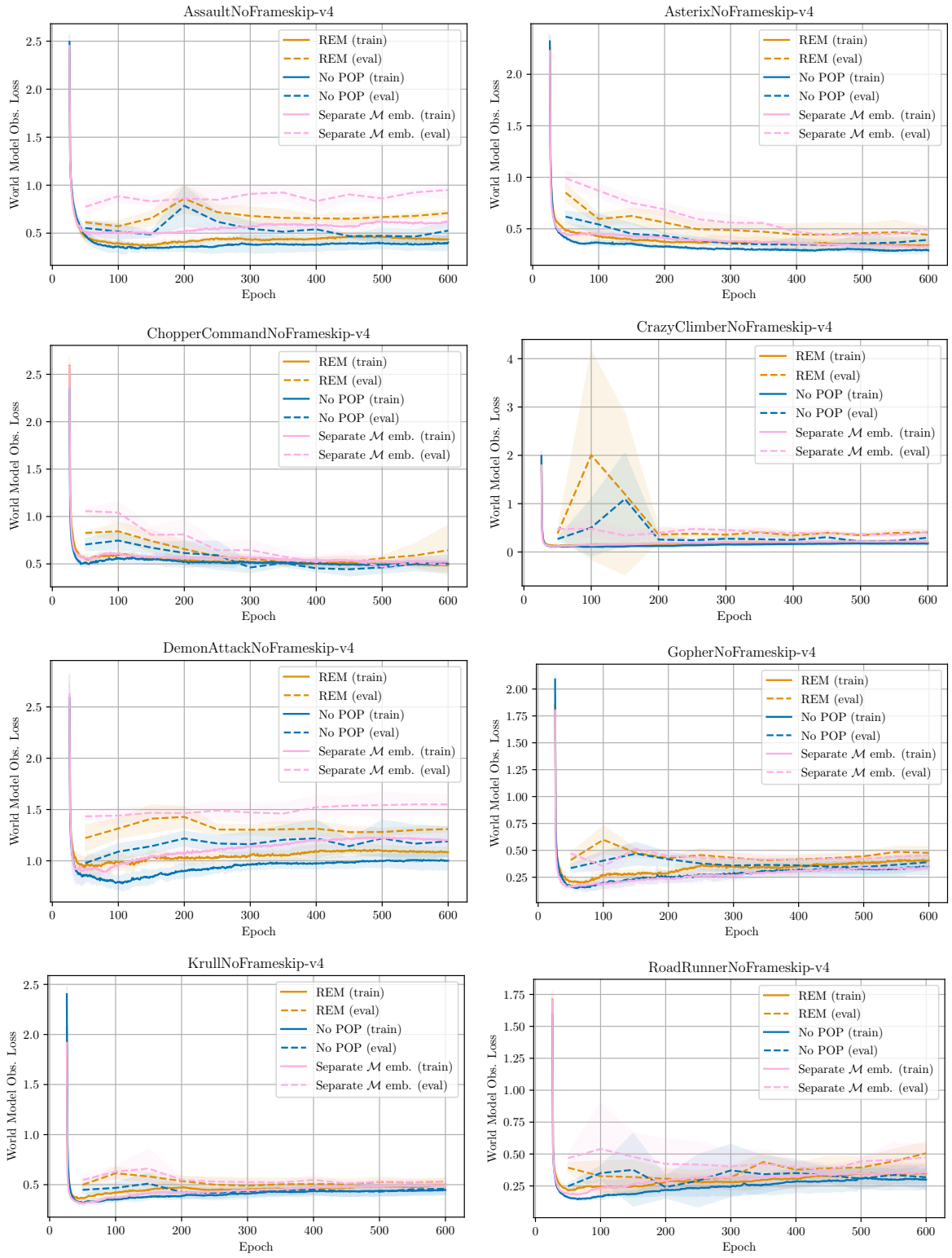

Figure 15. Comparison between the world model observation prediction loss of REM and two of its ablations for each of the 8 games considered in our ablations. For each algorithm, the mean and standard deviation of the training and evaluation losses are shown. The observation prediction loss is computed as the average observation token cross-entropy loss. Note that the evaluation frequency is 50 epochs.

A.5. Setup in Freeway

For a fair comparison, we followed the actor-critic configurations of IRIS (Micheli et al., 2023) for Freeway. Specifically, the sampling temperature of the agent is modified from 1 to 0.01, a heuristic that guides the agent towards non-zero reward trajectories. We highlight that different methods use other mechanisms such as epsilon-greedy schedules and "argmax" action selection policies to overcome this exploration challenge (Micheli et al., 2023).