Effective Sampling in Intrinsically Motivated Reinforcement Learning

Vis, Youri A.

Award date:
2023

Department of Mathematics and Computer Science
Uncertainty in Artificial Intelligence Research Group

EFFECTIVE SAMPLING IN INTRINSICALLY MOTIVATED REINFORCEMENT LEARNING

Master Thesis

Youri Vis
ID: 1002986
y.a.vis@student.tue.nl

Supervisors:

Maryam Tavakol
m.tavakol@tue.nl

Soroush Ghandi
s.ghandi@tue.nl

January 25, 2023

Eindhoven

ABSTRACT

Choosing what experiences to sample can have a great impact on the performance and rate of convergence in reinforcement learning. Plenty of approaches have been proposed and shown varying results in different environments. However, the use of non-uniform sampling strategies have not been studied before in intrinsic reward reinforcement learning. we use Prioritized Experience Replay (PER) in combination with Random Network Distillation (RND) to investigate whether alternative sampling will improve the performance of the agent. Also, the effect of the size of the replay buffer is studied. We use Montezuma's Revenge as test environment, the hardest exploration game of all 57 original Atari games, with the least extrinsic rewards. None of the alternative sampling approaches proposed in this article show improvement over the default RND with uniform sampling.

Contents

1 Introduction

In recent years Reinforcement Learning (RL) has yielded super-human achievements in increasingly complex games such as Starcraft II and DOTA II. However, training these models currently requires meticulously designed reward functions. In real-world settings, these rewards are often unknown or sparse. The engineering of these reward functions is an intensive task and often not feasible or can even be misleading. For example, imagine a search-and-rescue robot that only gets a reward once it has put a person out of harm's way. Is the distance to the person an accurate reward function? How should it deal with obstacles? How does it prevent getting stuck?

Many promising models have been proposed that aim to solve this problem and that function without extrinsic reward by using some type of *intrinsic* reward. One common factor is that these models still need to process many experiences to converge. This is a resource and time-consuming aspect and begs the question of how we can learn more efficiently from states and experiences inside the replay buffer.

From nature, we know that not all experiences are given the same weight. In the hippocampus, memories with higher importance, such as ones associated with rewarding locations or high reward-prediction errors, are replayed more often [1, 2, 3]. Also, participants with a more frequent replay of high-reward memories show better performance in memory tasks [4, 5].

This paper uses Random Network Distillation (RND) [6] as a way to give the agent intrinsic rewards and test alternative sampling approaches inspired by Prioritized Experience Replay (PER) [7] to make its learning more efficient. The original 57 Atari games are often used as a benchmark for these RL models. The hardest exploration game of these is *Montezuma's Revenge* which takes long-term planning to obtain a high score, as it requires the player to explore multiple rooms, save keys and sacrifice short-term rewards for long-term exploration. It is an environment with sparse rewards that takes a lot of experiences to converge.

First of all, a literature review will show what ways have been tried before with regard to intrinsic reward algorithms, sampling strategies, and the effect of the replay buffer size. Next, the methodology section shows what four sampling strategies were tested on the environment. The results present how none of these methods increased the performance of the agent. The conclusion and discussion will try to answer why these methods showed no improvement.

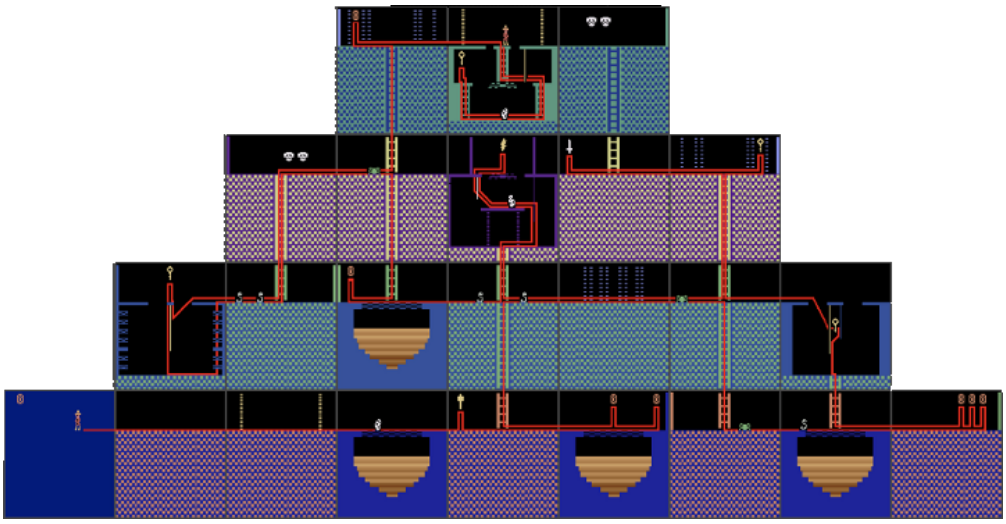

Figure 1: All rooms and the solution to Montezuma's Revenge

2 Literature Review

2.1 Intrinsic Reward Models

A common approach to handle sparse extrinsic reward is to augment the reward function with an additional intrinsic exploration bonus $r_t = r_t^e + \beta r_t^i$, where β is a hyperparameter adjusting the balance between intrinsic and extrinsic reward.

Count based exploration

The intuition of these methods is to add an intrinsic bonus when the agent goes into a state in which it usually never goes [8, 9]. This reward can be formalised with:

$$r_t^i(s_t) = \frac{1}{N(s_t)}$$

Where $N(s)$ is the number of times this state s has been visited. This method works well in tabular environments with discrete state spaces but breaks down when states are continuous since an agent never really returns to the exact same state. Instead, it is possible to use density models to approximate the frequency of state visits by deriving a *pseudo-count* $\hat{N}(s)$ [10].

$$r_t^i(s_t) = \frac{1}{\hat{N}(s_t)}$$

$$\hat{N}(s_t) = \frac{\rho(s)(1-\rho'(s))}{\rho'(s)-\rho(s)}$$

Where $\rho(s)$ is the density model which outputs a probability of observing s, and $\rho'(s)$ the probability to observe s after one more pass on s [11]. For example, these density models can be a Context Tree Switching model [10], PixelCNN[12], or a Gaussian Mixture Model [13].

Another method to limit continuous high dimensional states is to hash the state space $\phi : \mathcal{S} \mapsto \mathbb{Z}^k$ and update the reward function as $r^i(s) = N(\phi(s))^{-1}$. Tang et al. (2016) propose Locality-Sensitive Hashing as it preserves distance information between data points. For higher dimensional raw pixels they design an Autoencoder (AE), that takes the input state s to learn hash codes [14].

Prediction based exploration

Another method is to evaluate an agent's familiarity with the environment by its ability to predict it. An intuitive approach is to learn a forward dynamics model which predicts the environment's next state based on the current state and the agent's action: $f : \phi(s_t, a_t) \mapsto \phi(s_{t+1})$, with the reward being $r_t^i = \|f(s_t, a_t) - \phi(s_{t+1})\|$.

Predicting the next feature state is hard and might not even be desirable because some factors may not be dependent on the action of the agent or may not affect the agent's actions and thus there is no incentive in learning them. It also is susceptible to the 'Noisy TV' problem which occurs when stochastic elements in the environment cause the agent to be unable to predict the next state of these elements, resulting in a high intrinsic reward [6]. Because this prediction error is due to aleatoric uncertainty (which is caused by intrinsic stochasticity in the environment), instead of epistemic uncertainty (which stems from limited data or knowledge about the environment), it does not teach the agent anything useful about the environment. Moreover, the agent can get stuck in these states as it will exploit these rewards forever as it will never be able to predict stochasticity.

This has motivated literature on uncertainty quantification, which tries to separate aleatoric from epistemic uncertainty. Methods include using ensemble models, maximum likelihood estimation, Bayesian inference, generative models, and Gaussian processes [15]. For example Clements et al. (2019) use two networks with the same task and calculate the covariance between the two predictions to get the amount of variance caused by aleatoric uncertainty [16]. This is to correct for over-confident estimates neural networks often produce and get a sense of the aleatoric uncertainty of the model, although deep-ensembles methods do not always lead to improved calibration properties [17]. Huseljic et al. (2020) propose a model that learns a Dirichlet-Categorical distribution over all classes such that they can derive measures describing aleatoric and epistemic uncertainty [18]. Bayes-By-Backpropagation uses distributions as weights in a neural network and seems to be a good approach for Reinforcement Learning as it converges relatively fast [19].

Other methods for solving the Noisy TV problem are by avoiding or amending the forward dynamics model. The Intrinsic Curiosity Module (ICM) for example uses an inverse dynamics feature (IDF) model that predicts what action

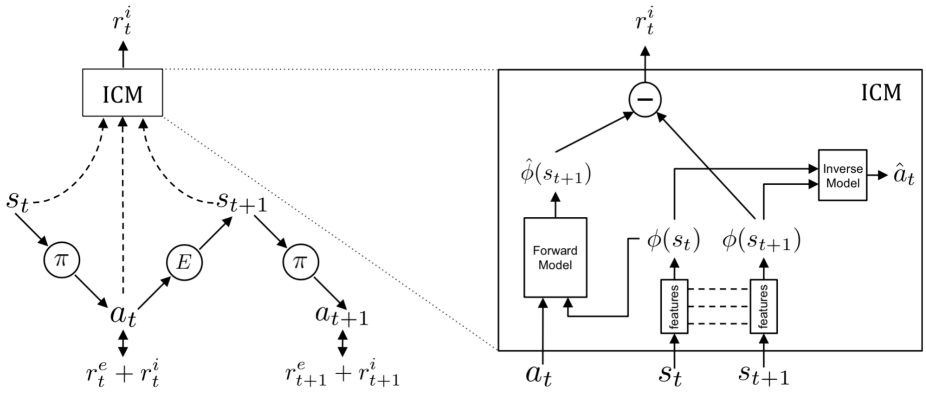

Figure 2: Architecture of the ICM [20].

led to the feature state transition $(\phi(s_t), \phi(s_{t+1})) \mapsto a_t$. Therefore IDF only captures the changes in the environment that are in control of the agent's actions [20]. The disadvantage of these encoded methods such as IDF or AE [21] is that they are not stable as their parameters are optimised by training.

Random features that are not semantically meaningful, however, tend to be fairly competitive to meaningful features and do not require training. Burda et al. (2018) studied four different encoding methods with a simple PPO agent with only intrinsic reward [22].

- Raw pixels: $\phi(s) = s$.
- Random Features (RF): state is embedded by a CNN that is fixed after random initialisation.
- VAE: a Variational AutoEncoder $\phi(s) = q(z|s)$.
- IDF: as used in ICM.

As shown in Figure 3, raw pixel input is dominated by the other encoding methods. The disadvantage of VAE and IDF is that they are not stable as their parameters are optimised by training. Random Features are quite competitive to VAE and IDF and have the advantage of not requiring training and being stable [22]. In Montezuma's Revenge, they even outperform IDF and VAE features.

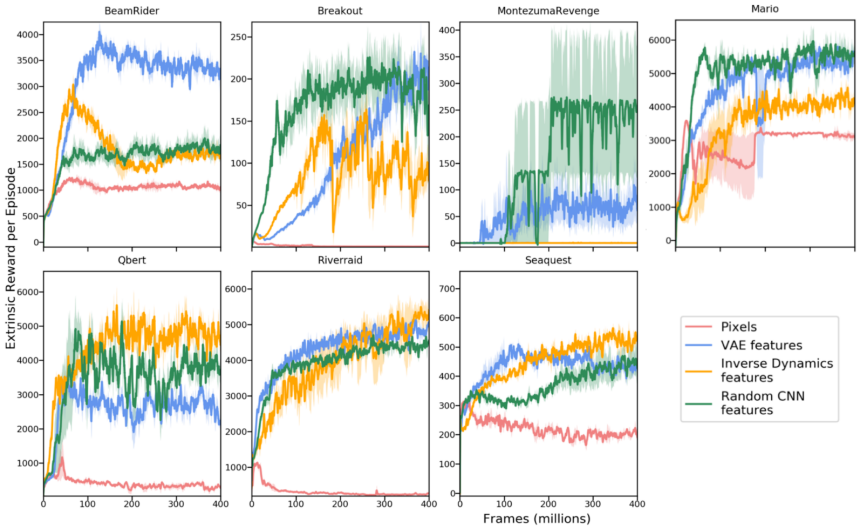

Figure 3: Intrinsic reward of four different encoding methods [22].

Random Network Distillation (RND) [6] is a relatively intuitive method that exploits this competitiveness of random features. It serves as the backbone of highly successful algorithms such as Never Give Up (NGU)[24] and Agent57 [25]. It creates a prediction task that is independent of the main task. It comprises a fixed randomly initialised Convolutional

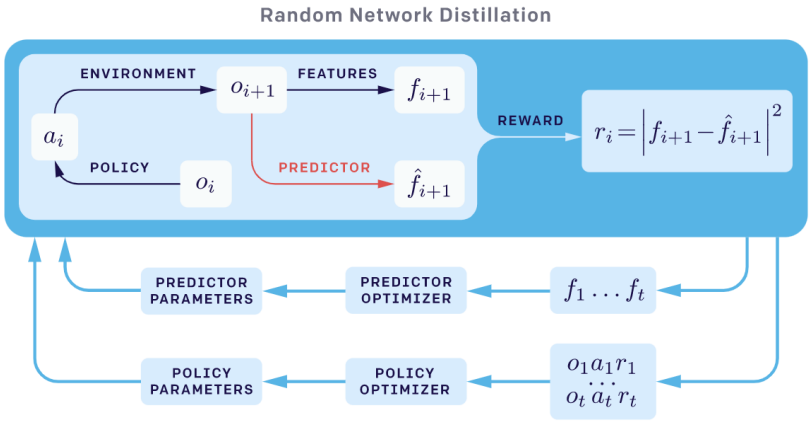

Figure 4: Architecture of Random Network Distillation, o_i is used for s_t [23].

Neural Network (CNN) f and a second CNN $\hat{f}$ that tries to reproduce the output of f. The first network encodes $f(s_t)$. The second network encodes $\hat{f}(s_t)$, where it optimises $(f(s_t) - \hat{f}(s_t))^2$. This loss is then simultaneously the intrinsic reward for state s_t, as it will be high if $\hat{f}$ has not learned the output of $f(s_t)$ since it is an infrequently visited novel state.

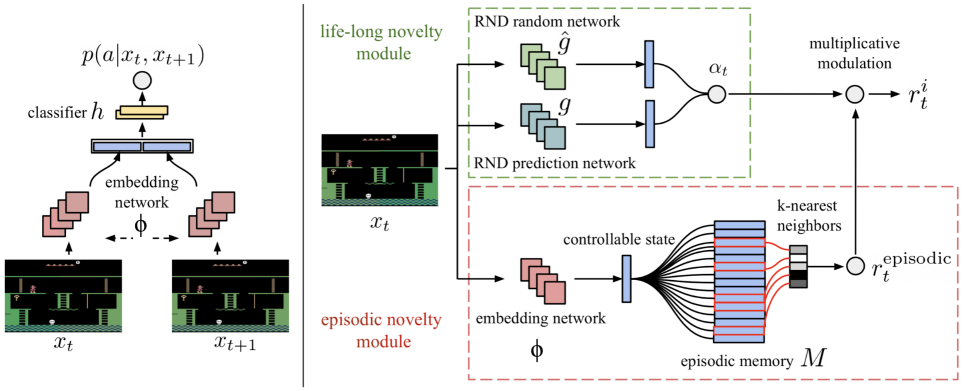

Figure 5: NGU's embedding function (left), and reward function (right) [24]

The Never Give UP (NGU) [24] takes RND a step further and uses two modules, a short-term *episodic novelty module*, and a *life-long novelty module*. It uses IDF as embedding function ϕ. For RND agents it is important not to truncate intrinsic reward at "game over", such that it is non-episodic and that normalisation is applied by subtracting the running estimate of the mean and dividing over the standard deviation of intrinsic return [22]. Therefore, NGU uses $\alpha_t = 1 + \frac{e^{\text{RND}(s_t) - \mu_e}}{\sigma_e}$ as normalised life-long non-episodic reward function. Agent57 [25] was built on top of NGU but used a population of policies with different exploration parameters $\{(\beta_j, \gamma_j)\}_{j=1}^{N}$. With reward function $r_{j,t} = r_t^e + \beta_j r_t^i$ and γ_j being the reward discounting factor. Moreover, it split the state-action value function in the following way to optimise intrinsic and extrinsic reward separately $Q(s, a; \theta_j) = Q\left(s, a; \theta_j^e\right) + \beta_j Q\left(s, a; \theta_j^i\right)$.

RND has some problems. The authors mention that the agent occasionally 'dances with the skulls', meaning it tries to evade enemies by a very small margin, probably because these states are visited very infrequently and therefore yield a large intrinsic reward. Furthermore, RND is limited because it may fail in global exploration with long time horizons, and only occasionally completes the first level of Montezuma's Revenge [23].

These shortcomings may be the result of a too slow convergence of the prediction network which causes the agent to receive high intrinsic rewards for a longer time than necessary. Ideally, in the environment of Montezuma's Revenge,

the agent would become accustomed to a room more quickly if prediction errors are solely the result of aleatoric uncertainty, in order to motivate the agent to keep exploring and finding intrinsic rewards due to epistemic uncertainty.

2.2 Sampling from Replay Buffer

The choice of what state transitions to train the model on can have a great impact on the convergence of the model [26]. Many approaches show improvement over uniform sampling. For example, gradually increasing the complexity of inputs when training a neural network [27], or filtering states from a given mini-batch of training data [28] may speed up convergence.

In the last decade, Experience Replay (ER) [29] has become prevalent in RL literature. ER first collects a batch of experiences before training the model and then interacts with the environment again. The idea is to break temporal correlations by mixing more and less recent experiences for the updates and using rare experiences for more than just a single update. ER has shown to be a lucrative method and has influenced many adaptations.

Hindsight Experience Replay (HER) [30] attempts to overcome sparse rewards by copying each trajectory experienced and replacing the actual rewards with rewards calculated assuming the goals are the steps achieved at the end of the trajectories. Competitive Experience Replay (CER) attempts to emphasize exploration by introducing competition between two agents attempting to learn the same task. One agent receives a penalty for visiting states that the competitor agent also visits, and vice versa [31].

Prioritized Experience Replay (PER) adds a way to prioritize some experiences over others. The idea is that an agent can learn more effectively from some transitions than from others. Transitions may be more or less surprising, redundant, or task-relevant [7]. Normally Temporal Difference (TD) error is used for determining the priorities, but other methods also take into account extrinsic rewards [32] or the agent's confidence in its prediction in combination with TD error [33]. PER has been shown to improve the convergence rate of the model [34] and clipping gradients that would change the current policy too much in PER might help in some environments [35].

Large Batch Experience Replay (LaBER) essentially does the same as PER, but first samples a large batch, computes importance sampling probability on this large batch, and down-samples the large batch to a mini-batch according to an optimal sampling distribution [36]. One can also use a separate model that tags relevant experience for these to be prioritized when sampling [37], or a Neural Experience Replay Sampler (NERS) [38], that combines local and global context to determine the relative importance of experiences.

2.3 Replay Buffer Size

Since Montezuma's Revenge is an exploration game where the agent needs long-term planning, the size of the replay buffer may influence its performance. When the replay buffer is large, older experiences are stored, whereas a small replay buffer would only contain recent experiences. This means a larger or smaller replay buffer means a greater bias toward long-term memory or short-term respectively. The replay buffer size has not been exhaustively studied and most approaches use the same size as the seminal work that implemented it with the Deep-Q-Network (DQN) [39], such as Deep Deterministic Policy Gradient (DDPG)[40] and Hindsight Experience Replay (HER) [30], even though the performance can vary greatly [41, 42, 43] and despite that its size intuitively determines the ratio between long-term and short-term memory for the agent. Until now, the effect of the replay buffer size on the agent's performance in combination with an intrinsic reward model has not been previously studied.

Figure 6 shows how the states are stored in the memory in RND with more than 1 'default' unit of memory. (P)ER normally uses a simple sliding window strategy when appending new states and removing older ones. This also begs the question of if this could be more efficient, as there might be a better replacement policy.

2.4 Objectives

RND has some shortcomings regarding convergence and samples uniformly from the replay buffer, therefore the combination with PER could improve it and speed up training. Since training agents in an environment like Montezuma's Revenge usually takes days, this may be of great use.

Additionally, the agent 'dances with the skulls', likely because the RND predictor network $\hat{f}$ has not learned enough from these states and therefore they yield a great intrinsic reward, which causes the agent to return and exploit them. If these states are prioritized when training the RND predictor, it will be more adapted to these states and cause the agent to receive less intrinsic rewards from these states and therefore to ignore those.

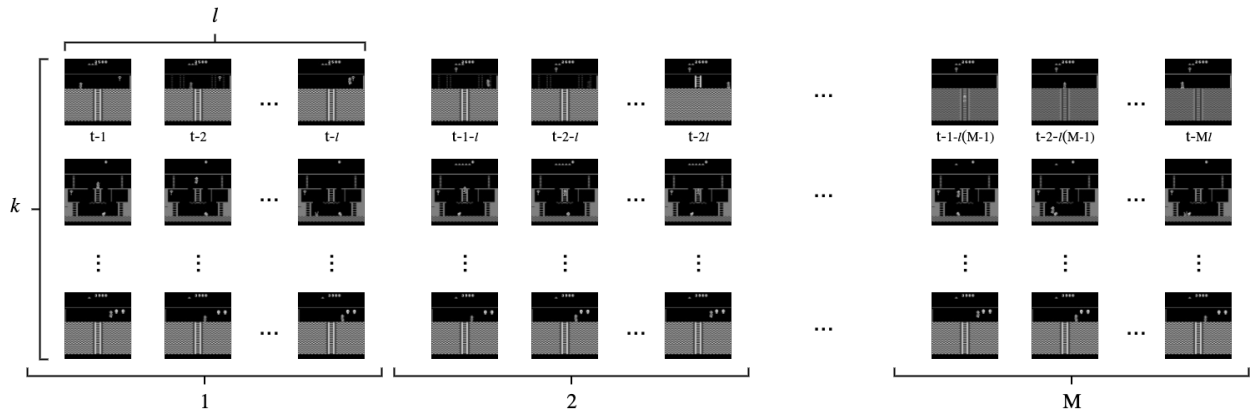

Figure 6: Visualisation of the contents of the replay buffer with what we consider the replay buffer size M and a memory unit. k parallel environments are run for roll-out length l states before being appended to the replay buffer.

There are also shortcomings in PER. It has been used on atari games and showed great improvements in some games, but no difference in Montezuma's Revenge. This could be because of the sparse rewards in Montezuma's Revenge and almost no model does meaningful exploration without some kind of intrinsic reward. Therefore a combination of PER with RND could have a synergistic effect and make them both better.

This leads to three questions that will be tested by running four different sampling strategies. First of all, what is the effect of the replay buffer's capacity on the performance of the agent? Secondly, does PER as sampling algorithm increase the performance of the RND agent, and is temporal bias or erroneous bias more important to increase its performance? Finally, does the replacement policy of the replay buffer increase performance if it is dependent on the error of the RND model?

3 Methodology

RND was originally implemented in combination with a Proximal Policy Optimization (PPO) CNN agent, so all runs are implemented with this same architecture to make for a fair comparison. The hyper-parameters of the agent and RND model are listed in the Appendix.

Four different methods of sampling from the replay buffer are implemented. Inspiration is drawn from Prioritized Experience Replay (PER), which gives states in the replay buffer that have a higher error δ a higher probability p of being sampled and also prioritizes recent states over older states. To determine the effect of PER on RND we will separately study the effect of proportionally prioritizing erroneous states and giving temporal bias to more recent states. A different replacement policy for PER is also tested and named PER-v2. These four methods are visualised in figure 7 and further elaborated below.

When processing new states, we have roll-out length l and parallel environment k, as shown in figure 6. These experiences get stacked to train the RND model as well as the policy model. In figure 7 this is visualised as the darkest new batch that gets appended.

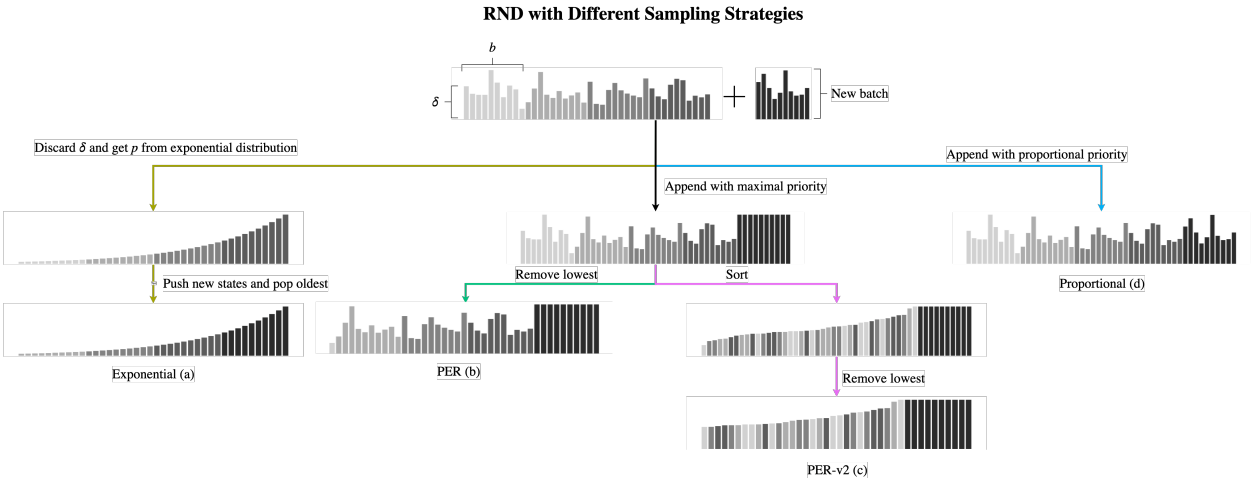

Figure 7: Visualisation of the four different sampling strategies.

3.1 Proportional Prioritization

Theoretically, when sampling states that have a higher error more often, the model converges more quickly to those states. In the case of RND, this means that the model converges in the direction of infrequently visited states since these have a higher error.

Intuitively, this means the agent becomes accustomed to the environment more quickly and the incentive of exploring this new environment will degrade more quickly. In other words, the agent will get 'bored' more quickly and is driven to explore more instead of exploiting this newly discovered environment longer.

Moreover, edge cases that cause the agent to 'dance with the skulls' might be tackled as the predictor can get more accustomed to these edge cases. Where it normally would have sampled these maybe once before it is already discarded from the memory.

The probability distribution is defined as follows

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

With $p_i = |\delta_i| + \epsilon$, and α determining how much prioritization is used, with $\alpha = 0$ corresponding to the uniform case.

3.2 Temporal Prioritization

To study the effect of temporal prioritization we have to discard the proportional priority given by PER. Normally, when states are added to the replay buffer, PER gives it maximal priority $p_t = \max_{i<t} p_i$, favouring the most recent

states that just were appended. To exclusively focus on the temporal bias, we have to choose a different approach, as this would result in a uniform distribution if we discard proportional priorities. Therefore we define the probability distribution exponentially

$$P(i) = \lambda e^{\frac{-\lambda i}{c}}$$

With c and λ being hyper-parameters to determine the shape of the distribution. For these experiments $\lambda = 0.5$ and $c = l \cdot k$ are chosen, where l is the roll-out length and k the number of parallel environments. In other words, we scale the exponential distribution according to the size of the replay buffer.

3.3 PER

PER creates both a bias towards recent states and states with erroneous predictions. Algorithm 1 shows the pseudo code with alterations from subsections 3.1, 3.2, and 3.4.

Since prioritized replay introduces a bias toward high erroneous states the authors use importance-sampling weights defined as

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^{\beta}$$

where this fully compensates the non-uniform distribution $P(i)$ if $\beta = 1$. Since the authors used an off-policy method, for an on-policy model we use (PPO with RND) we want this non-uniformity not to be annealed and so it is better to entirely discard the annealing. The pseudo-code of PER is shown in algorithm 1.

Algorithm 1: RND with four sample strategies. Code that is included in Exponential, PER-v2, and Proportional, is coloured. For PER the pseudo-code is as is without the coloured lines.

Objects: target feature encoder f, predictor feature encoder $\hat{f}$, total roll-outs T, number of transitions in one roll-out m, batch size b, the replay buffer $\mathcal{H}$, capacity of the replay buffer C.

1 Initialize replay memory $\mathcal{H} \leftarrow \emptyset$, gradient $\Delta \leftarrow 0$, and maximal priority $p_{\text{max}} \leftarrow 1$
2 **for** $t \leftarrow 1$ **to** T **do**
3 Run environment until we collect m transitions
4 **for** $i \leftarrow 1$ **to** m **do**
5 Store transition $(s_i, a_i, r_i ...)$ in $\mathcal{H}$ with maximal priority $p_i = p_{\text{max}}$
6 $p_i = \delta_i + \epsilon$ // Proportional
7 **end**
8 **if** $|\mathcal{H}| + m > C$ **then**
 // PER-v2
9 Sort $\mathcal{H}$ on p
10 $\mathcal{H} \leftarrow \{s \in \mathcal{H} \mid p(s) > p(\mathcal{H}_m)\}$
11 **end**
12 **foreach** epoch **do**
13 **foreach** batch **do**
14 Sample transitions $j \sim P(j) = p_j^{\alpha} / \sum_i p_i^{\alpha}$
15 Sample transitions $j \sim P(j) = \lambda e^{-\lambda x / c}$ // Exponential replaces line 14
16 Compute L2-error $\delta_j = (f(s_j) - \hat{f}(s_j))^2$
17 Update transition priority $p_j \leftarrow \delta_j + \epsilon$
18 Accumulate weight-change $\Delta \leftarrow \Delta + \delta_j \cdot \nabla_\theta \hat{f}(s_{j-1})$
19 **end**
20 Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$
21 $\Delta \leftarrow 0$
22 **end**
23 $p_{\text{max}} \leftarrow \max_i p_i$
24 **end**

3.4 PER-v2

Lastly, there also surfaces a choice of replacement policy. Normally PER uses a FIFO replacement policy as shown in figure 7.b, with a simple sliding window. However, this replacement policy is indifferent to the error of the states and does not treat frequently visited states differently from infrequently visited states within one batch of experiences. We can take this error into account by first sorting on the error before replacing the least erroneous states. This way, states that are older but still highly erroneous can be continued to be sampled in further iterations, and more recent states that are less erroneous are filtered out more quickly.

Figure 7.c shows a different way where we first append the most recent states, sort, and discard the least erroneous states. This way the RND model keeps the states it can still learn from the most and gives the RND model the possibility to keep useful very old states or discard very recent states that have become of low use.

3.5 Experiments

The RND paper describes two policy networks, one a CNN and one an RNN. The RNN slightly outperforms the CNN but will take more training time. Since the comparison between default RND and RND in combination with PER will still be valid with CNN and most experiments were run on a CNN in the original article, these experiments will also be implemented with a CNN.

The original RND paper used 30K rollouts of length 128 per environment with 128 parallel environments for most experiments, processing about 2B frames. Since performance plateaus before 1B frames, we chose to run the experiments for 1B frames to save time, as a single run already takes approximately 5 days on the High-Performance Cluster (HPC) that was used to run these experiments.

These runs were executed with a roll-out length of 128 states per environment with 56 parallel environments per run, resulting in approximately 35K iterations to reach 1B frames. For evaluation, the exploratory performance of the agent in Montezuma's Revenge can be evaluated using the number of rooms it has discovered. There is no cap on this score as the level resets itself once the agent has discovered all 24 rooms.

Both the speed at which the agent reaches new rooms and the number of rooms it discovers are important for the overall 'skill' of the agent. We evaluate the total performance by fitting a non-linear regression model and evaluating the coefficients to determine which sampling strategies have a beneficial effect.

Furthermore, we chose the following replay buffer sizes to run the sampling algorithms on. These experiments were run on an HPC with a 2.20 GHz Intel Xeon Gold 6238R (56 cores) and 8 Nvidia GeForce RTX 2080 Ti with a total of 256 GB RAM, with 8 experiments running in parallel at a time. The number of runs for each combination of algorithm and replay buffer size is listed in table 1 below.

Table 1: Experiments

Sampling Algorithm	**Replay Buffer Size**							**Figure**
	1	**2**	**4**	**6**	**8**	**16**	**64**	
Uniform	5	3	3	3	3			
Exponential	2	2	2	3	3			7.a
PER			3	3	3	3	3	7.b
PER-v2		2	2		3	3	3	7.c
Proportional	3	3	3	3	3			7.d

For PER and PER-v2 it is less interesting to run it on tiny replay buffer sizes as it then just becomes a proportional approach and their value will only be apparent once we increase the replay buffer size. This is why replay buffer size 1 and 2 are sometimes omitted and PER-v2 6 omits size 6 for reasons of time constraints and this increment is of less importance. All code is available on GitHub and logged runs are available on Weights & Biases.

4 Results

4.1 Processed Frames vs. Visited Rooms

Multiple non-linear models were fitted to predict the number of rooms the agent has visited after processing a number of frames. The best three fitted models are shown in table 2 in the Appendix and have a R^2 of 0.640, 0.678, 0.658. The first two models treat the replay buffer size as ordinal. Models 2 and 3 add interaction between the replay buffer and algorithm. Model 3 also adds polynomial variables to the order of 3 regarding frames. Since model 2 had the highest R^2, this one was plotted, although the other models had a similar curve. Figure 8 shows all runs together with the best-fitted nonlinear model. All algorithms had a significant negative effect ($p < 0.01$). The effect of the replay buffer size was more tentative, but in general also negative. To make these coefficients of the regression model and thereby the effect of the sampling strategies and memory size more salient, these are plotted in figure 9.

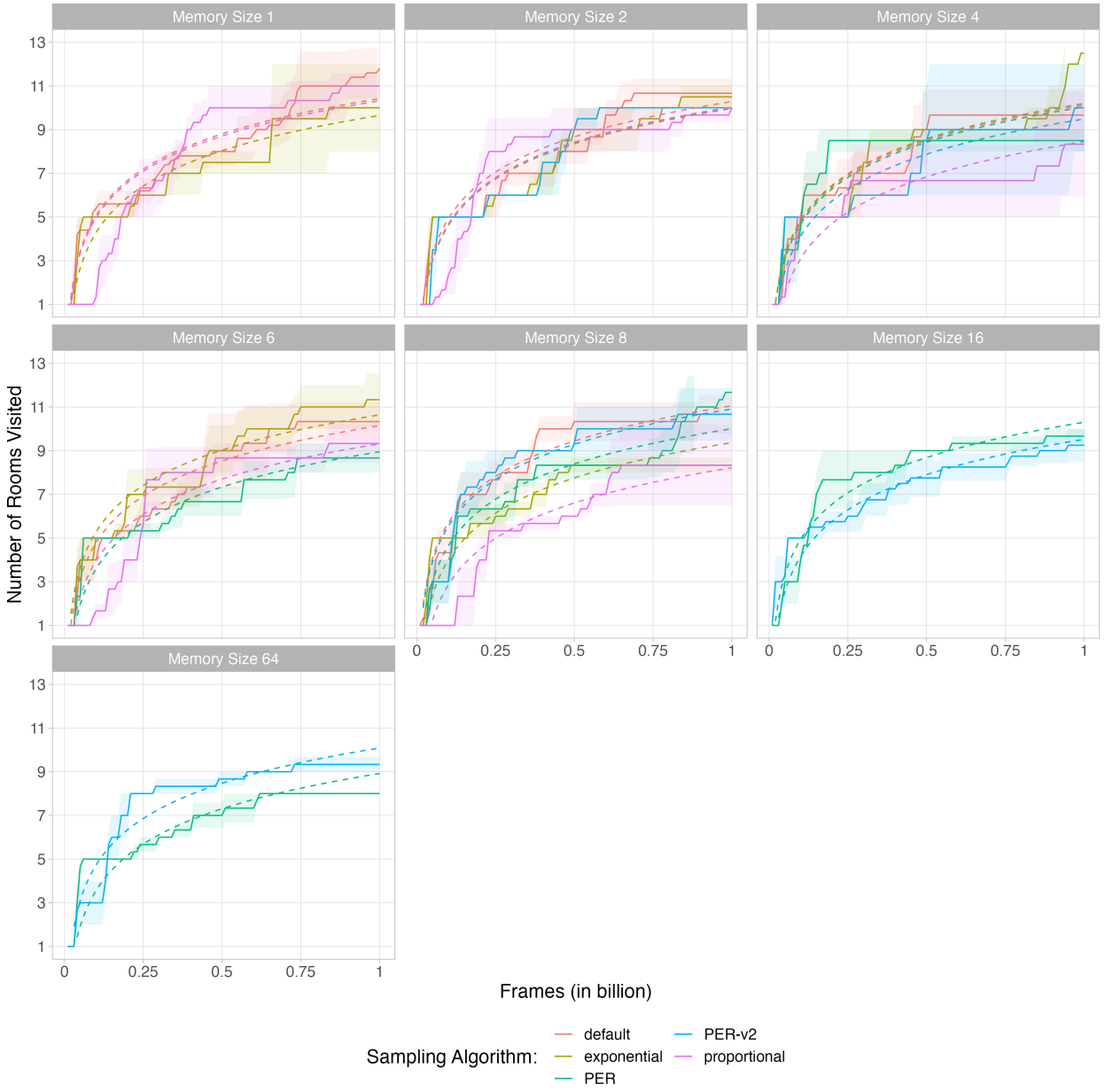

Figure 8: Number of rooms visited by a combination of algorithm and memory size plotted against the number of frames were processed. The standard error is plotted as bands. Fitted regression model estimates are the dashed lines.

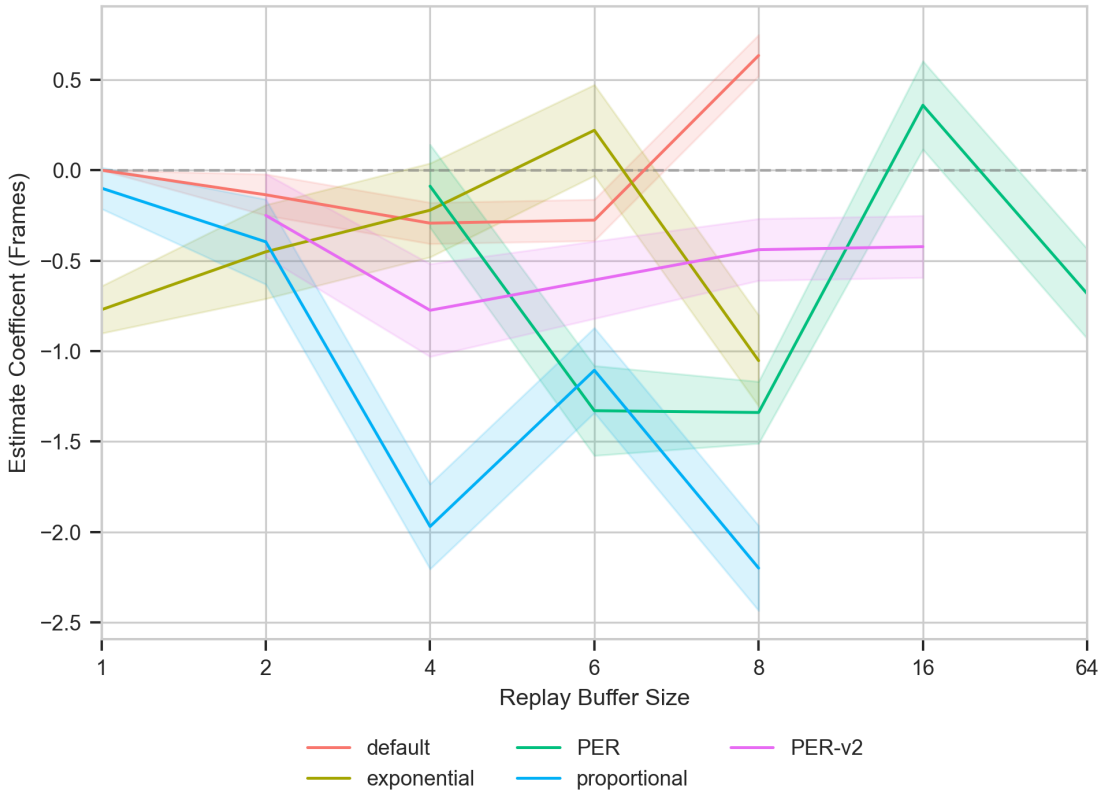

Figure 9: Coefficients for sampling algorithm in combination with memory size and their interaction effect for the regression model. Bands show the standard deviation.

From these coefficients, we see that most non-uniform algorithms have a poorer performance than the default strategy with a replay buffer size of 1, except for PER with size 16, exponential with size 6, and default with size 8, although these do not greatly exceed the default. It seems that only the proportional sampling strategy has a trend that indicates worse performance with a larger replay buffer.

4.2 Time vs. Visited Rooms

From the previous section, one might conclude that these three exceptions do in fact increase the performance and convergence rate of the model, however, we should take into account the additional time these algorithms take to get a more realistic picture. For PER this means updating the priorities and exponential sampling from this exponential distribution. Therefore figure 10 shows the number of rooms visited against the number of hours the model ran.

Again, the best three models that were fitted are listed in Appendix table 3. The R^2 of these models were 0.643, 0.669, 0.661 and were made up of the same variables as the models in the previous section (except for time as opposed to frames). The coefficients are again plotted in figure 11. This shows that when taking the actual run-time into account the advantage of the previously mention three combinations of algorithm and replay buffer size disappears. Counterintuitively, the default algorithm with replay buffer size 2 is faster than with replay buffer size 1. There is no reason to assume this is more than an anomaly as theoretically this only takes more memory and processing time.

For PER-v2 we can analyse if it has different behaviour from PER by checking the relative age of experiences in the replay buffer, as defined by how many training iterations they have stayed in the replay buffer without being removed. Figure 12 shows this for different replay buffer sizes. No clear pattern can be made up except for the fact that more recent states make up for a larger portion of the replay buffer, which is no surprise since the longer a state stays in the replay buffer, the more times it can be sampled and the model to be adapted to it, resulting in it being discarded for yielding a too low error. Arguably the distribution has a higher kurtosis once the model converges, as the figure shows a greater spread of grey towards the end. This indicates that less weight is given to a temporal bias.

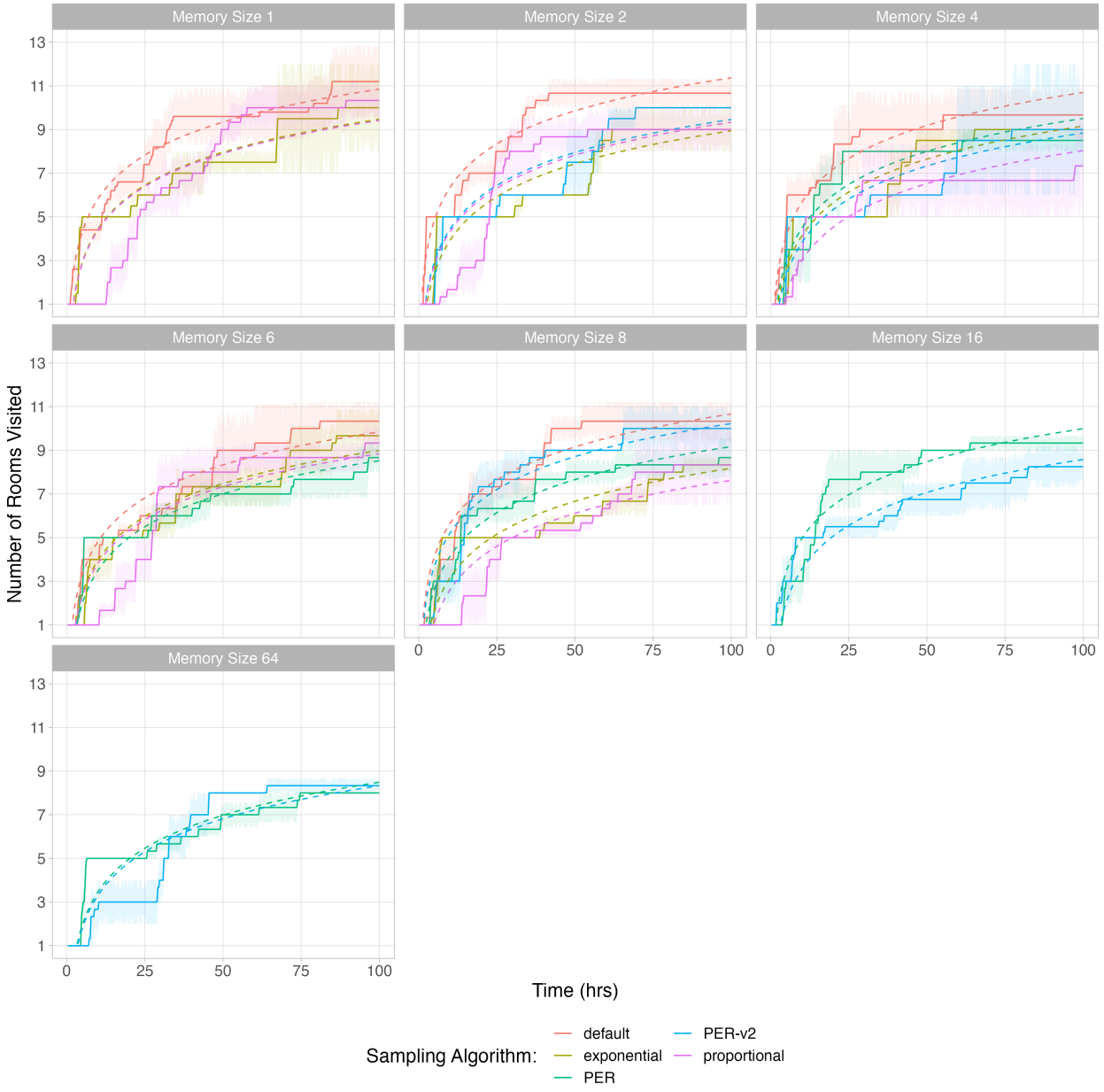

Figure 10: Number of rooms visited by a combination of algorithm and memory size plotted against the number of frames were processed. The standard error is plotted as bands. Fitted regression model estimates are the dashed lines.

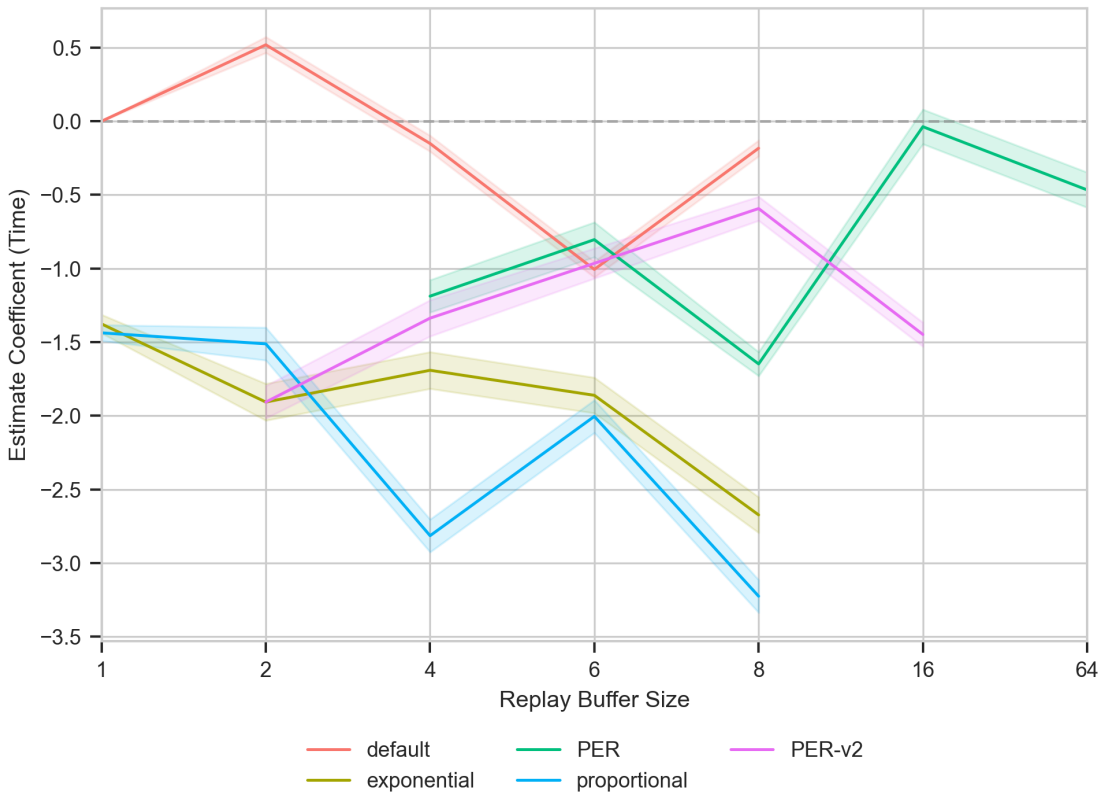

Figure 11: Coefficients for sampling algorithm in combination with memory size and their interaction effect for the regression model. Bands show the standard deviation.

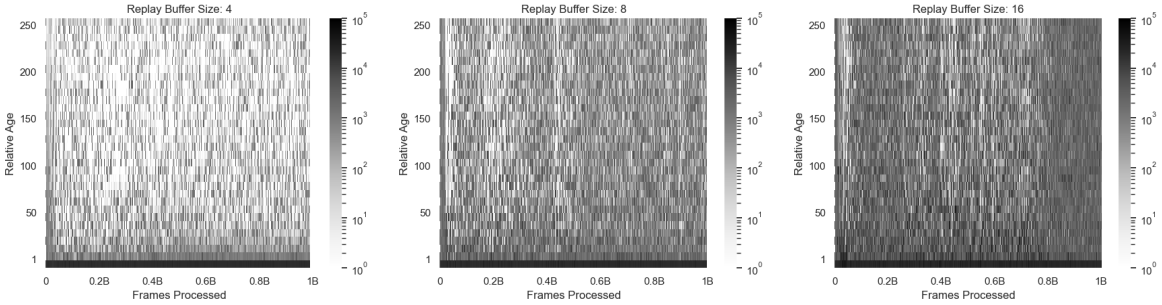

Figure 12: Relative age of states in the replay buffer with count as the colour intensity of PER-v2 with replay buffer size 4, 8, and 16.

5 Conclusion

None of the algorithms proposed in this article outperform the default RND algorithm. PER and exponential sampling seemed to outperform the default with some replay buffer size, but when comparing this to the additional time it takes to run PER, the added benefit disappears.

Figure 12 shows that PER-v2 does have different behaviour than PER, as it shows that even states 250 iterations old are still in the replay buffer. The distribution is heavily skewed to the more recent ones, but this might be of value for further research if more weight is given to these older states when they do get sampled.

PER-v2 lacks scalability when the replay buffer increases. When run with a replay buffer size of 256, it processed 300M frames in 122 hours whereas the default with replay buffer size 1 already processed approximately 1B frames in the same time. This means that PER-v2 would only be viable in a sweet spot somewhere between 2 and 8 in replay buffer size. When the replay buffer is too small, there is no use in sorting as most states will get replaced regardless of their priority. When it is too large, the sorting takes too much time.

There is a similar problem with PER, as the priorities of states have to be updated after each training step. The implementation we used, uses linear time, but it is possible to implement this using a Sum Tree, making the updating logarithmic in time complexity. However, a cost remains, and PER does not show many advantages over the default RND version.

Figure 8 shows that the exponential sampling does not decrease as much as the proportional sampling when the replay buffer increases. If we assume that PER is similar to the sum of these two methods, we can conclude that the updating of the priorities mainly causes detriment in performance. The temporal bias of giving the most recent states maximal priority causes less detriment. This is also justified by the intuition that proportionally sampling gives less weight to the most recent states when the replay buffer increases, simply because the fraction of most recent states is smaller.

6 Discussion

First of all, the differences in replay buffer sizes could have been too small to show a salient effect, it might have been better to scale it in factors of 10, however, the scaling of replay buffer size came with the expense of a significantly longer processing time. Ideally, we would have run more experiments per combination for stronger results, but this also came with a cost. One might argue to have tested the models in a simpler environment than Montezuma's Revenge. However, making the environment simpler would inevitably result in less aleatoric uncertainty and the long horizon of Montezuma's Revenge gave this environment enough challenge to make intrinsic rewards absolutely necessary to solve it. Less complex environments are often solvable without intrinsic rewards.

Secondly, there might be a flaw in combining PER with multiple parallel environments. The idea of PER is that the agent favours some states over others. However, since we ran 56 parallel environments within each experiment, the states that could be useful to prioritize for one agent, might not be useful to another, or might even be detrimental. We ran the experiments with just one agent to test this hypothesis, but it failed to explore further than the first room after 1B frames (even with varying hyper-parameters such as the learning rate), and so no useful insight can be drawn from those runs.

Another reason why PER might not work for Montezuma's Revenge is that the agent has to have a sense of older states when exploring the newer ones, as it sometimes has to return to an older room to pick up a key or open a door before being able to continue. When focusing too much on the more recent states, the agent may 'forget' about the older states, or even consider these older rooms as new again as the RND model converged to the newer ones and has to converge to the older rooms once again. We tested giving more weight in the replay buffer to the states that correspond to the room the agent is currently in, in order to avoid this problem. Yet again, this resulted in no improvement.

This study investigated the effect of different sampling approaches on RND, we chose to focus solely on implementing and testing it. However, the PPO was still trained by uniform sampling. It would still be of value to study the effect of using a non-uniform method for this part of the algorithm.

References

[1] F. Michon, J. J. Sun, C. Y. Kim, D. Ciliberti, and F. Kloosterman, "Post-learning Hippocampal Replay Selectively Reinforces Spatial Memory for Highly Rewarded Locations," *Current Biology*, vol. 29, pp. 1436–1444, 5 2019.

[2] E. L. Roscow, M. W. Jones, and N. F. Lepora, "Behavioural and computational evidence for memory consolidation biased by reward-prediction errors," *bioRxiv*, vol. 716290, 2019.

[3] B. Salvetti, R. G. Morris, and S. H. Wang, "The role of rewarding and novel events in facilitating memory persistence in a separate spatial memory task," *Learning and Memory*, vol. 21, pp. 61–72, 2 2014.

[4] M. J. Gruber, M. Ritchey, S. F. Wang, M. K. Doss, and C. Ranganath, "Post-learning Hippocampal Dynamics Promote Preferential Retention of Rewarding Events," *Neuron*, vol. 89, pp. 1110–1120, 3 2016.

[5] A. C. Schapiro, E. A. McDevitt, T. T. Rogers, S. C. Mednick, and K. A. Norman, "Human hippocampal replay during rest prioritizes weakly learned information and predicts memory performance," *Nature Communications*, vol. 9, 12 2018.

[6] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by Random Network Distillation," *arXiv preprint arXiv:1810.12894*, 10 2018.

[7] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," *ICLR*, 11 2015.

[8] R. I. Brafman and M. Tennenholtz, "R-max-A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning," tech. rep., 2002.

[9] M. Kearns and S. Singh, "Near-Optimal Reinforcement Learning in Polynomial Time," tech. rep., 2002.

[10] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying Count-Based Exploration and Intrinsic Motivation," 6 2016.

[11] A. Aubret, L. Matignon, and S. Hassas, "A survey on intrinsic motivation in reinforcement learning," *arXiv preprint arXiv:1908.06976*, 8 2019.

[12] A. v. d. Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, "Conditional Image Generation with PixelCNN Decoders," 6 2016.

[13] R. Zhao and V. Tresp, "Curiosity-Driven Experience Prioritization via Density Estimation," 2 2019.

[14] H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, "#Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning," 11 2016.

[15] E. Hüllermeier and W. Waegeman, "Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods," *Machine Learning*, vol. 110, pp. 457–506, 3 2021.

[16] W. R. Clements, B. Van Delft, B.-M. Robaglia, R. B. Slaoui, and S. Toth, "Estimating Risk and Uncertainty in Deep Reinforcement Learning," in *ICML 2020 Workshop on Uncertainty and Robustness in Deep Learning*, ICML, 5 2019.

[17] R. Rahaman and A. H. Thiery, "Uncertainty Quantification and Deep Ensembles," *Advances in Neural Information Processing Systems*, vol. 34, pp. 20063–20075, 7 2020.

[18] D. Huseljic, B. Sick, M. Herde, and D. Kottke, "Separation of aleatoric and epistemic uncertainty in deterministic deep neural networks," in *Proceedings - International Conference on Pattern Recognition*, pp. 9172–9179, Institute of Electrical and Electronics Engineers Inc., 2020.

[19] C. Tegho, P. Budzianowski, and M. Gasic, "Benchmarking Uncertainty Estimates with Deep Reinforcement Learning for Dialogue Policy Optimisation," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 2018-April, pp. 6069–6073, Institute of Electrical and Electronics Engineers Inc., 9 2018.

[20] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven Exploration by Self-supervised Prediction," 5 2017.

[21] S. Alvernaz and J. Togelius, "Autoencoder-augmented Neuroevolution for Visual Doom Playing," 7 2017.

[22] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros, "Large-Scale Study of Curiosity-Driven Learning," *arXiv preprint arXiv:1808.04355*, 8 2018.

[23] Y. Burda and H. Edwards, "Reinforcement Learning with Prediction-Based Rewards," 10 2018.

[24] A. P. Badia, P. Sprechmann, A. Vitvitskyi, D. Guo, B. Piot, S. Kapturowski, O. Tieleman, M. Arjovsky, A. Pritzel, A. Bolt, and C. Blundell, "Never Give Up: Learning Directed Exploration Strategies," *arXiv preprint arXiv:2002.06038*, 2 2020.

[25] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, and C. Blundell, "Agent57: Outperforming the Atari Human Benchmark," in *Proceedings of the 37th International Conference on Machine Learning*, pp. 507–517, PMLR, 3 2020.

[26] B. Anenberg and B. Raghavan, "Sampling Strategies for Deep Reinforcement Learning," *arXiv preprint arXiv:1312.5602*, 2013.

[27] Y. Bengio, u. Jérôme Louradour, R. Collobert, and J. Weston, "Curriculum Learning," in *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, ICML, 2009.

[28] Y. Fan, F. Tian, T. Qin, J. Bian, and T.-Y. Liu, "Learning What Data to Learn," *arXiv preprint arXiv:1702.08635*, 2 2017.

[29] L.-J. Lin, "Self-Improving Reactive Agents Based On Reinforcement Learning, Planning and Teaching," *Machine Learning*, vol. 8, pp. 293–321, 1992.

[30] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight Experience Replay," tech. rep.

[31] H. Liu, A. Trott, R. Socher, and C. Xiong, "Competitive Experience Replay," in *Seventh International Conference on Learning Representations*, ICLR, 2 2019.

[32] J. Gao, X. Li, W. Liu, and J. Zhao, "Prioritized Experience Replay Method Based on Experience Reward," in *Proceedings - 2021 International Conference on Machine Learning and Intelligent Systems Engineering, MLISE 2021*, pp. 214–219, Institute of Electrical and Electronics Engineers Inc., 2021.

[33] M. Ramicic, V. Smidl, and A. Bonarini, "Informed Sampling of Prioritized Experience Replay," in *2022 IEEE International Conference on Development and Learning (ICDL)*, pp. 215–222, Institute of Electrical and Electronics Engineers Inc, 9 2022.

[34] Y. Pan, J. Mei, A.-M. Farahmand, M. White, H. Yao, M. Rohani, and J. Luo, "Understanding and Mitigating the Limitations of Prioritized Experience Replay," in *Uncertainty in Artificial Intelligence*, pp. 1561–1571, PMLR, 2022.

[35] G. Novati and P. Koumoutsakos, "Remember and Forget for Experience Replay," in *Proceedings of the 36th International Conference on Machine Learning, PMLR*, vol. 97, pp. 4851–4860, PLMR, 2019.

[36] T. Lahire, M. Geist, and E. Rachelson, "Large Batch Experience Replay," *arXiv preprint arXiv:2110.01528*, 10 2021.

[37] G. Dao and M. Lee, "Relevant Experiences in Replay Buffer," in *IEEE symposium series on computational intelligence (SSCI)*, pp. 94–101, Institute of Electrical and Electronics Engineers Inc, 2019.

[38] Y. Oh, K. Lee, J. Shin, E. Yang, and S. J. Hwang, "Learning to Sample with Local and Global Contexts in Experience Replay Buffer," *arXiv preprint arXiv:2007.07358*, 7 2020.

[39] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2 2015.

[40] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 9 2015.

[41] S. Zhang and R. S. Sutton, "A Deeper Look at Experience Replay," 12 2017.

[42] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, and W. Dabney, "Revisiting Fundamentals of Experience Replay," in *International Conference on Machine Learning*, pp. 3061–3071, PMLR, 2020.

[43] R. Liu and J. Zou, "The Effects of Memory Replay in Reinforcement Learning," in *56th annual allerton conference on communication, control, and computing (Allerton)*, pp. 478–485, IEEE, 2018.

[44] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd International Conference for Learning Representations*, arXiv preprint arXiv:1412.6980, 12 2015.

Appendix

Table 2: Regression models (frames). Model 1 and 2 treats replay buffer size as ordinal. Models 2 and 3 add interaction between the replay buffer and algorithm. Model 3 also adds polynomial variables to the order of 3 regarding frames.

	Dependent variable:		
	Visited Number of Rooms		
	(1)	(2)	(3)
exponential	-0.394^{***}	-0.771^{***}	-0.153
	(0.063)	(0.130)	(0.118)
PER	-0.539^{***}	-1.047^{***}	-0.715^{***}
	(0.069)	(0.127)	(0.166)
PER-v2	-0.178^{**}	-0.147	-0.541^{***}
	(0.070)	(0.127)	(0.147)
proportional	-1.110^{***}	-0.099	0.193^{*}
	(0.059)	(0.114)	(0.106)
log(frame)	2.328^{***}	2.328^{***}	1.213^{***}
	(0.021)	(0.020)	(0.122)
poly(frame, 1)			84.249^{***}
			(8.702)
poly(frame, 2)			-28.885^{***}
			(3.802)
poly(frame, 3)			7.523^{***}
			(2.475)
poly(mem_size, 1)			-462.828^{***}
			(37.069)
poly(mem_size, 2)			-10.122^{***}
			(3.280)
poly(mem_size, 3)			7.581^{**}
			(3.140)
default:mem_size			0.324^{***}
			(0.022)
exponential:mem_size			0.252^{***}
			(0.024)
PER:mem_size			0.318^{***}
			(0.027)
PER-v2:mem_size			0.331^{***}
			(0.027)
proportional:mem_size			
mem_size2	-0.107	-0.136	
	(0.075)	(0.114)	
mem_size4	-0.586^{***}	-0.293^{**}	
	(0.073)	(0.114)	
mem_size6	-0.375^{***}	-0.276^{**}	
	(0.073)	(0.114)	
mem_size8	-0.292^{***}	0.634^{***}	
	(0.071)	(0.114)	
mem_size16	-0.461^{***}	-0.754^{***}	
	(0.097)	(0.165)	
mem_size64	-0.787^{***}	-0.186	
	(0.100)	(0.171)	
Constant	-37.597^{***}	-37.825^{***}	-19.229^{***}
	(0.420)	(0.399)	(2.417)
Observations	7,200	7,200	7,200
R^2	0.640	0.678	0.658
Adjusted R^2	0.640	0.677	0.657
Residual Std. Error	1.648 (df = 7188)	1.559 (df = 7174)	1.608 (df = 7184)
F Statistic	$1{,}162.183^{***}$ (df = 11; 7188)	605.595^{***} (df = 25; 7174)	920.753^{***} (df = 15; 7184)
Note:			*p<0.1; **p<0.05; ***p<0.01

Regression models (frames) continued.

	Dependent variable:		
	Visited Number of Rooms		
	(1)	(2)	(3)
exponential:mem_size2		0.456^{**}	
		(0.193)	
PER:mem_size2			
PER-v2:mem_size2		-0.103	
		(0.191)	
proportional:mem_size2		-0.161	
		(0.171)	
exponential:mem_size4		0.843^{***}	
		(0.193)	
PER:mem_size4		0.958^{***}	
		(0.191)	
PER-v2:mem_size4		-0.492^{**}	
		(0.191)	
proportional:mem_size4		-1.577^{***}	
		(0.171)	
exponential:mem_size6		1.268^{***}	
		(0.182)	
PER:mem_size6		-0.147	
		(0.180)	
PER-v2:mem_size6			
proportional:mem_size6		-0.731^{***}	
		(0.171)	
exponential:mem_size8		-0.916^{***}	
		(0.182)	
PER:mem_size8			
PER-v2:mem_size8			
proportional:mem_size8		-2.734^{***}	
		(0.171)	
exponential:mem_size16			
PER:mem_size16		1.682^{***}	
		(0.174)	
PER-v2:mem_size16			
proportional:mem_size16			
exponential:mem_size64			
PER:mem_size64		-0.270	
		(0.180)	
PER-v2:mem_size64			
proportional:mem_size64			
Constant	-37.597^{***}	-37.825^{***}	-19.229^{***}
	(0.420)	(0.399)	(2.417)
Observations	7,200	7,200	7,200
R^2	0.640	0.678	0.658
Adjusted R^2	0.640	0.677	0.657
Residual Std. Error	1.648 (df = 7188)	1.559 (df = 7174)	1.608 (df = 7184)
F Statistic	$1{,}162.183^{***}$ (df = 11; 7188)	605.595^{***} (df = 25; 7174)	920.753^{***} (df = 15; 7184)
Note:			*p<0.1; **p<0.05; ***p<0.01

Table 3: Regression Models (Hours). Model 1 and 2 treats replay buffer size as ordinal. Models 2 and 3 add interaction between the replay buffer and algorithm. Model 3 also adds polynomial variables to the order of 3 regarding hours.

	Dependent variable:		
	Visited Number of Rooms		
	(1)	(2)	(3)
exponential	-1.711^{***}	-1.376^{***}	-1.538^{***}
	(0.030)	(0.063)	(0.056)
PER	-1.089^{***}	-1.497^{***}	-2.080^{***}
	(0.032)	(0.061)	(0.078)
PER-v2	-1.288^{***}	-0.442^{***}	-1.941^{***}
	(0.033)	(0.061)	(0.070)
proportional	-2.003^{***}	-1.437^{***}	-1.507^{***}
	(0.027)	(0.054)	(0.049)
log(hrs)	2.181^{***}	2.181^{***}	0.704^{***}
	(0.009)	(0.009)	(0.045)
poly(hrs, 1)			253.586^{***}
			(7.305)
poly(hrs, 2)			-100.008^{***}
			(3.402)
poly(hrs, 3)			23.578^{***}
			(2.353)
poly(mem_size, 1)			-817.011^{***}
			(37.708)
poly(mem_size, 2)			-19.075^{***}
			(3.420)
poly(mem_size, 3)			9.224^{***}
			(3.242)
mem_size2	0.018	0.519^{***}	
	(0.035)	(0.054)	
mem_size4	-0.635^{***}	-0.151^{***}	
	(0.034)	(0.054)	
mem_size6	-0.835^{***}	-1.006^{***}	
	(0.034)	(0.054)	
mem_size8	-0.700^{***}	-0.183^{***}	
	(0.033)	(0.054)	
mem_size16	-0.690^{***}	-1.827^{***}	
	(0.046)	(0.079)	
mem_size64	-1.488^{***}	-2.079^{***}	
	(0.047)	(0.082)	
default:mem_size			0.123^{***}
			(0.010)
exponential:mem_size			0.081^{***}
			(0.011)
PER:mem_size			0.260^{***}
			(0.013)
PER-v2:mem_size			0.252^{***}
			(0.013)
proportional:mem_size			
Constant	-16.822^{***}	-17.057^{***}	-2.007^{***}
	(0.114)	(0.111)	(0.539)
Observations	34,075	34,075	34,075
R^2	0.643	0.669	0.661
Adjusted R^2	0.643	0.669	0.661
Residual Std. Error	1.693 (df = 34063)	1.629 (df = 34049)	1.649 (df = 34059)
F Statistic	$5{,}575.848^{***}$ (df = 11; 34063)	$2{,}758.521^{***}$ (df = 25; 34049)	$4{,}432.270^{***}$ (df = 15; 34059)
Note:			*p<0.1; **p<0.05; ***p<0.01

Regression models (hours) continued.

	Dependent variable:		
	Visited Number of Rooms		
	(1)	(2)	(3)
exponential:mem_size2		-1.049^{***}	
		(0.093)	
PER:mem_size2			
PER-v2:mem_size2		-1.465^{***}	
		(0.092)	
proportional:mem_size2		-0.594^{***}	
		(0.081)	
exponential:mem_size4		-0.163^{*}	
		(0.093)	
PER:mem_size4		0.309^{***}	
		(0.091)	
PER-v2:mem_size4		-1.414^{***}	
		(0.092)	
proportional:mem_size4		-1.226^{***}	
		(0.081)	
exponential:mem_size6		0.521^{***}	
		(0.088)	
PER:mem_size6		0.174^{**}	
		(0.086)	
PER-v2:mem_size6			
proportional:mem_size6		0.439^{***}	
		(0.081)	
exponential:mem_size8		-1.113^{***}	
		(0.088)	
PER:mem_size8			
PER-v2:mem_size8			
proportional:mem_size8		-1.605^{***}	
		(0.081)	
exponential:mem_size16			
PER:mem_size16		2.465^{***}	
		(0.085)	
PER-v2:mem_size16			
proportional:mem_size16			
exponential:mem_size64			
PER:mem_size64		1.214^{***}	
		(0.087)	
PER-v2:mem_size64			
proportional:mem_size64			
Constant	-16.822^{***}	-17.057^{***}	-2.007^{***}
	(0.114)	(0.111)	(0.539)
Observations	34,075	34,075	34,075
R^2	0.643	0.669	0.661
Adjusted R^2	0.643	0.669	0.661
Residual Std. Error	1.693 (df = 34063)	1.629 (df = 34049)	1.649 (df = 34059)
F Statistic	$5{,}575.848^{***}$ (df = 11; 34063)	$2{,}758.521^{***}$ (df = 25; 34049)	$4{,}432.270^{***}$ (df = 15; 34059)
Note:			*p<0.1; **p<0.05; ***p<0.01

Table 4: PPO and RND hyperparameters

Hyperparameter	Value
Rollout length	128
Total number of rollouts per environment	35K
Number of minibatches	4
Number of optimization epochs	4
Coefficient of extrinsic reward	2
Coefficient of intrinsic reward	1
Number of parallel environments	56
Learning rate	0.0001
Optimization algorithm	Adam [44]
λ	0.95
Entropy coefficient	0.001
Proportion of experience used for training predictor	0.25
γ_E	0.999
γ_I	0.99
Clip range	[0.9,1.1]
Policy architecture	CNN

Table 5: CNN policy network with this output going to an action Dense network (448x18), and a critic Dense network (448x2) outputting intrinsic and extrinsic estimates.

Layer Type	Size	Additional Remarks
Conv2d	32	kernel size 8, stride 2
ReLU		
Conv2d	64	kernel size 4, stride 2
ReLU		
Conv2d	64	kernel size 3, stride 1
ReLU		
Flatten		
Dense	256	
ReLU		

Table 6: RND target network f

Layer Type	Size	Additional Remarks
Conv2d	32	kernel size 8, stride 4
LeakyReLU		
Conv2d	64	kernel size 4, stride 2
LeakyReLU		
Conv2d	64	kernel size 3, stride 1
LeakyReLU		
Flatten		
Dense	512	

Table 7: RND predictor network $\hat{f}$

Layer Type	Size	Additional Remarks
Conv2d	32	kernel size 8, stride 4
Conv2d	64	kernel size 4, stride 2
Conv2d	64	kernel size 3, stride 1
Flatten		
Dense	512	
Dense	512	
Dense	512	

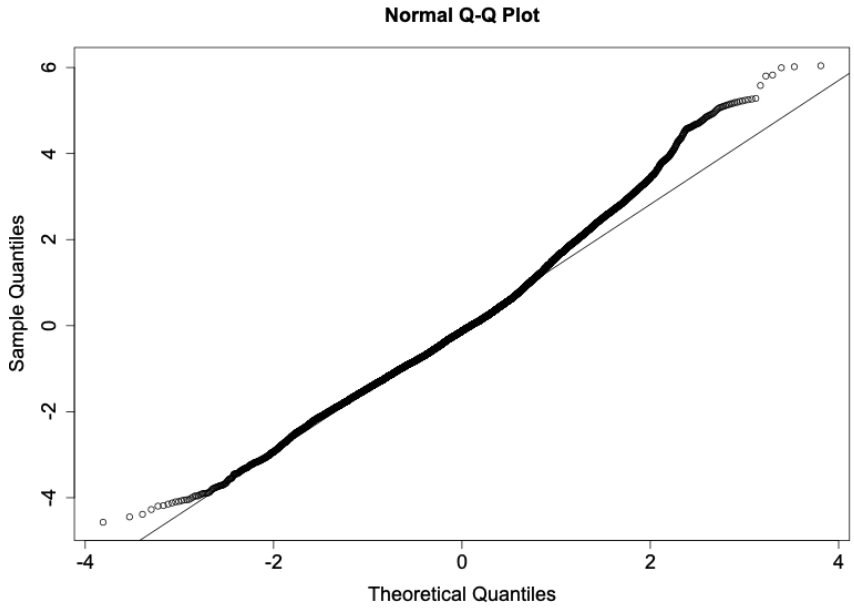

Figure 13: Residuals of the regression model (frames).

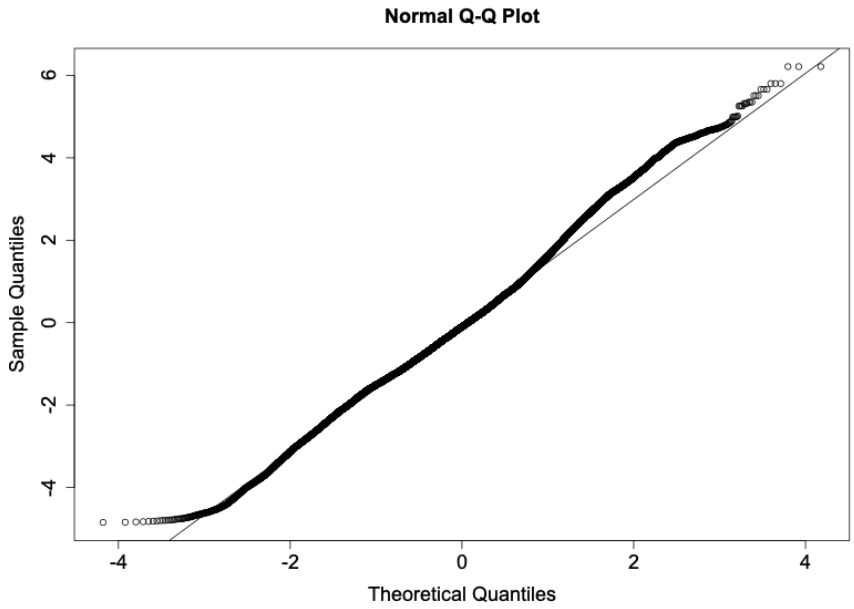

Figure 14: Residuals of the regression model (hours).