

When Source Selection Fails: Expert-free Online Transfer Learning Beyond its Validated Regime

Kevin Sánchez^{1,*}, Jorge Hernandez² and Diego H. Peluffo-Ordóñez¹

¹*Yachay Tech University, San Miguel de Urcuquí, Ecuador*

²*Universidad Distrital Francisco José de Caldas, Bogotá, Colombia*

Abstract

Expert-free online transfer learning (EF-OnTL) allows multiple agents to learn in parallel and share experiences with one another. This eliminates the need for a pre-trained expert. Instead, every certain number of episodes, it selects an agent as a source—for example, the one with the lowest average epistemic uncertainty. Our hypothesis is that low uncertainty implies high competence; we took the published code based on the original work by [1] and ran it on a consumer laptop. In this reimplementation, five corrections were necessary to make it work, one of which was a deviation from the documented protocol. Subsequently, we adapted the configuration validated on Cart-Pole to ALE/MsPacman-v5 under RAM observations, without modifying anything, where MsPacman has 128 observation dimensions instead of four. The algorithm worked correctly, and the agents learned. Furthermore, in both runs, they outperformed a random-action baseline that we measured using the same configuration. However, the selection rule ceased to function, as in Cart-Pole it achieved better results than random selection, with an average score of 2.50 out of 5, and uncertainty was negatively correlated with performance in five of the six transfer events. On the other hand, MsPacman performed worse than random selection in two independent runs, with an average score of 3.83, and the correlation was never negative—the best agent was never chosen as the source. We also measured the cost in terms of real time, showing that it increases linearly with the number of steps in the environment, at nearly the same rate in both environments. The same 1,800 episodes took 3.5 hours in Cart-Pole and 15.6 hours in MsPacman. Finally, we ran one simulation per configuration and no control runs without transfer, so we do not take a position on whether transfer helped or hindered performance. Our contribution is this observation: a source selection rule may fail to select competent agents while the algorithm continues to function normally, with nothing in its output to indicate this. You can find our work on: <https://github.com/KevJoss/Expert-Free-Online-Transfer-Learning-Algorithm>

Keywords

Autonomous agents, Multi-agent reinforcement learning, Transfer learning, Source selection, Uncertainty estimation, Reproducibility, Empirical evaluation

1. Introduction

When multiple reinforcement learning agents are trained in parallel on the same task, each one independently rediscovers what the others have already learned. That is why sharing experience among them is a good idea—it accelerates learning—but it raises the question of what to share and, more importantly, who should share it. Classical transfer methods answer the second question by using a pre-trained expert, which is restrictive, since in many contexts no such expert exists, and creating one is precisely the problem being solved. In contrast, expert-free methods dynamically select a source from among the learners themselves. Expert-free online transfer learning (EF-OnTL)[1] does this by classifying agents according to their average epistemic uncertainty, estimated using a state-action-reward random network distillation (sars-RND) module, and designating the least uncertain agent as the source for that round. That agent then transfers a predetermined batch of its experiences—filtered according to a transfer criterion—to the replay buffers of the remaining agents.

ICAIW 2026: Workshops at the 9th International Conference on Applied Informatics 2026, October 13–17, 2026, Orlando, United States

*Corresponding author.

✉ kevin.sanchez@yachaytech.edu.ec (K. Sánchez); jehernandezr@udistrital.edu.co (J. Hernandez);

diego.peluffo@sdas-group.com (D.H. Peluffo-Ordóñez)

🌐 <https://github.com/KevJoss> (K. Sánchez); <https://www.diegopeluffo.com> (D.H. Peluffo-Ordóñez)

🆔 0009-0003-0311-9170 (K. Sánchez); 0000-0002-7129-7919 (J. Hernandez); 0000-0002-9045-6997 (D.H. Peluffo-Ordóñez)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

When we talk about algorithms of this nature, they generally include two elements worth paying attention to: the configuration of their hyperparameters and the protocol that establishes the experimental budget. In both cases, there is no guarantee that either will withstand a change in domain. Both cases are addressed in greater detail in 2. However, to provide some brief context, studies comparing different environments indicate a decline in performance when a configuration tuned for one set of environments is applied to another [2]; furthermore, it has been shown that optimal hyperparameter values do not transfer even within the same domain when the training budget changes [3]. In multi-agent reinforcement learning (MARL) systems with experience sharing, the source selection rule is an additional component exposed to this risk, and its designers acknowledge that such rules are heuristic approximations of utility that may fail in different domains [4]. In this context, to the best of our knowledge, no direct study has been conducted on how a source selection criterion behaves when its configuration and protocol are transferred without modification to a domain with substantially higher observational dimensionality. In MARL transfer studies that do address domain shifts, this has been achieved by pre-training representations in agents from unprocessed observations in an unsupervised manner and transferring them through weight initialization or progressive networks [5], rather than relying on the robustness of the selection rule itself. Given this gap, this work takes the configuration and protocol validated by Castagna for Cart-Pole [1], an environment that is not natively multi-agent and for which five parallel instances were run to construct a multi-agent scenario—and examines what happens when they are transferred without modification to another context. Both the documented configuration and the source code are publicly available, making such an examination possible. We address three questions. First, whether the published implementation reproduces the documented protocol and what corrections are required before it can run. Second, what results are obtained when porting that configuration, without adjustments, to a non-native multi-agent environment of substantially higher dimensionality: five parallel instances of ALE/MsPacman-v5 under RAM observations. Third, what is the cost of conducting these experiments on a consumer laptop? rather than the dedicated server used in the original study, and what does that cost imply for anyone attempting a replication?

The main conclusion of this study concerns the second of these questions. The transfer mechanism continues to function after transfer, as it selects sources within the configured intervals, transfers the configured number of tuples, and the agents learn, outperforming a random action baseline measured under the same configuration. However, the source selection criterion no longer identifies competent agents. In Cart-Pole, the selection outperforms random selection, with an average score of 2.50 out of 5. In MsPacman, the selection underperforms random selection in two independent runs, with an identical average score of 3.83, and the agent that placed first in nine of the twelve transfer events was never selected. The failure is silent, since the algorithm generates no warnings, no errors, and no visible deterioration, and anyone reusing it would observe that the agents perform better than random play and conclude that the method works. Our evidence is limited to one run per configuration and two seeds for MsPacman, compared to the twenty runs for Cart-Pole in the original work. That is why we focus on identifying the behavior, but do not establish effects or hypotheses regarding it.

2. Background and related work

Reinforcement learning (RL) is a field of machine learning that enables the development of autonomous agents [6]. Thus, RL distinguishes between two sources of uncertainty. Random uncertainty arises from the stochasticity of the environment itself. On the other hand, epistemic uncertainty reflects the agent’s lack of knowledge about a region of the state space, and it decreases as that region is explored. Estimating the latter has traditionally relied on counting visits, which is impractical when the state space increases in dimensionality or is continuous. The use of random network distillation (RND) replaces counting with a function approximation: a fixed target network, initialized randomly, and a trainable predictor network both map an input to an output; furthermore, during this process, the predictor network is trained to match the target network, and the prediction error is taken as the epistemic uncertainty of that input. RND was introduced as a novel signal for exploration: the error is

high for inputs different from those previously observed and decreases as an input becomes familiar [7]. Despite the good results that RND can yield, it only evaluates the current state, disregarding the action taken, the reward obtained, and the resulting transition.

That is why the authors of the paper proposing the EF-OnTL algorithm opt to use SARS-RND, which retains the RND architecture but takes the complete transition (s, a, r, s') as input; as reported in that paper, this provides better uncertainty estimates than RND based solely on the state [1, 8, 9]. With the help of this estimator, MARL eliminates the need for a fixed expert, as it follows the process mentioned in Section 1 by using the uncertainty estimate as a selection criterion for the source agent (teacher) for a specific round of knowledge transfer to the other agents. Figure 1 illustrates the cycle.

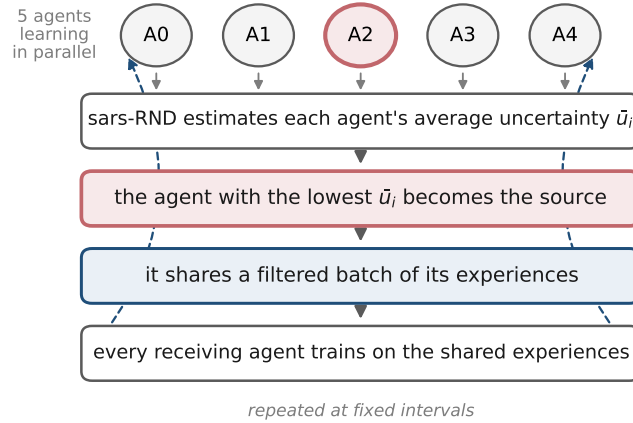


Figure 1: A simple diagram detailing the general operation of the EF-OnTL, section by section.

Now, as detailed above, the context is SARS-RND; however, since it consists of two neural networks for estimating the uncertainty value, its learning depends on a specific hyperparameter configuration. On the other hand, the EF-OnTL algorithm consists not only of an uncertainty estimator but also of various components that depend on the configuration of different parameters for proper implementation. Given this premise, it is well known that empirical results in deep reinforcement learning are fragile; for example, performance can depend on random seeds, implementation details not described in the papers, and specific source code decisions—to such an extent that it complicates comparisons between published methods [10]. Subsequent studies have quantified how many independent runs are required before a performance difference can be asserted with adequate statistical power [11], and have documented variability across runs, specifically in Atari environments [12]. Given this literature review, it is worth considering the independent re-execution of published results and establishing that conclusions drawn from a single run should be expressed as observations rather than absolute truths. On the other hand, there is a line of research that examines whether a configuration validated in one environment remains valid in others. Comparisons conducted across environments reveal a drop in performance when hyperparameter configurations selected in one environment are applied to another [2]. Other studies, meanwhile, propose protocols for quantifying how sensitive an algorithm is to its hyperparameters, rather than simply reporting the optimized performance [13][14]. Sensitivity is not limited solely to the choice of different environments; when considering identical domains, it has been shown that optimal hyperparameter values do not persist across different training budgets. For example, one study details that the configurations selected under the Atari 100k regime, for the most part, cease to be optimal over longer time horizons [3]. Given all the limitations and considerations discussed previously, it is clear that a protocol that fixes the training budget and transfers it unchanged to a new domain does not preserve the conditions under which the original configuration was validated.

Based on what was mentioned in the preceding paragraphs, the portability of configurations between different or similar environments is unstable and degrades the performance of agents that have operated with a specific configuration. Given this premise and knowledge, this paper draws on two works by the

same author in his doctoral thesis and a conference paper [1][8] as its foundation and reference. Our objective is to examine the system’s behavior when both the configuration and the selection protocol are transferred to a domain of substantially higher dimensionality; that is, we wish to verify whether the rule selects competent master agents despite changes in the configuration.

3. Experimental framework

Our methodology began with the reimplementing of the author’s original simulation in the cart-pole environment. We used the same configuration, with the sole difference being the number of simulations due to the computational cost required to run them. An initial exploration of the reimplementing revealed certain issues in the configuration of the source code provided by the author. We therefore determined that the published implementation would require five corrections before the documented experiments could be carried out. Each of the five corrections was classified according to its level of impact on the experimental results. Each classification, along with its respective details, is shown in Table 1

Class	Issue	Component	Affects results?
A. Packaging	Missing package initialiser prevents module import	agents package	No
A. Packaging	Hard dependency on an optional serialisation library	uncertainty estimator	No
B. Latent defect	Six stored fields unpacked into seven targets when n -step returns are disabled	learning routine	Yes: blocks the transfer-driven update path
B. Latent defect	The initialization calls a class method that has not been linked, causing the seed to be set	agent module	Yes: runs execute unseeded
C. Protocol divergence	The transfer start restriction is missing when using the average uncertainty criteria	transfer engine	Yes: see Sec. 3.1

Table 1

Corrections required to execute the released EF-OnTL implementation, classified by their potential to affect experimental outcomes.

Each of the classifications is addressed as follows: Class A defects prevent execution but cannot alter the results once resolved. The two Class B defects are latent—that is, each is located in a code path that is not executed in all configurations. Finally, the Class C problem is of a different nature, and we describe it separately in the following paragraph.

3.1. Documented discrepancy in the system

The documented and used configuration for the Cart-Pole environment specifies average uncertainty as the selection criterion for the source (expert) agent and defers the first transfer event until episode 600, on the grounds that the agents lack sufficient knowledge to estimate a utility confidence before that point [1]. In the published implementation, the condition that configures this delay is applied to the best-performance criterion for selecting the source agent, but this is not the case for the average uncertainty criterion. This is why, when running the system with the documented configuration, transfer events occurred starting in episode 200 and not in episode 600 as mentioned. As a result, in episode 200, the selected source agent had 4,467 tuples in its transfer buffer—a value lower than the configured budget of 5,000—so the transfer could not even deliver a complete batch; on the other hand, in episode 400, the buffer contained 8,422 tuples, of which the budgeted 5,000 were transferred. Once we identified the problem, we decided to reinstate the restriction, and our execution logs confirm the correct behavior documented in both environments: source selection is skipped in episodes 200 and 400, and the first transfer occurs in episode 600, followed by five more events at the configured interval of

200 episodes. We are reporting this as a discrepancy between the published system and the documented protocol. We make no claims regarding the code used to obtain the originally reported results, as we do not have access to it.

3.2. Experimental protocol and baseline

In this work, we have modeled each run as training five agents over 1,800 episodes. Due to the computational cost on the available hardware, we present one run with a seed of 42 for the Cart-Pole environment, and two runs with seeds of 7 and 42, respectively, for the MsPacman environment, rather than the twenty reported in the original study. For each run, the seed was initialized using `torch.manual_seed` and Python’s `random` module; however, for the NumPy, CUDA, and ALE environments, no initialization was performed, so bit-for-bit reproducibility is not guaranteed. On the other hand, to establish a minimum performance threshold in the MsPacman environment, we measured a baseline based on taking random actions under the same configuration. In this case, we configured five instances of 500 episodes each, resulting in a total of 2,500 records as data, selected at random. The benchmark for MsPacman is not obvious. There are published scores from random play for this game; [15] reports 307.3 points for a uniformly random policy, but these were obtained before stochasticity was introduced into the Arcade Learning Environment. It was verified through code that the v5 environment used in this work applies “sticky” actions, meaning that with a probability of 0.25, the emulator ignores the sent action and repeats the previous one. Since the published scores measured without this mechanism are not directly comparable, we measured our own minimum, where the resulting average of 261.2 points is below the published figure and serves as the benchmark used throughout section 5. In addition, we present Figure 2, which details how EF-OnTL operates as it runs in our configuration, showing the agent-level loop and the points at which the transfer mechanism intervenes.

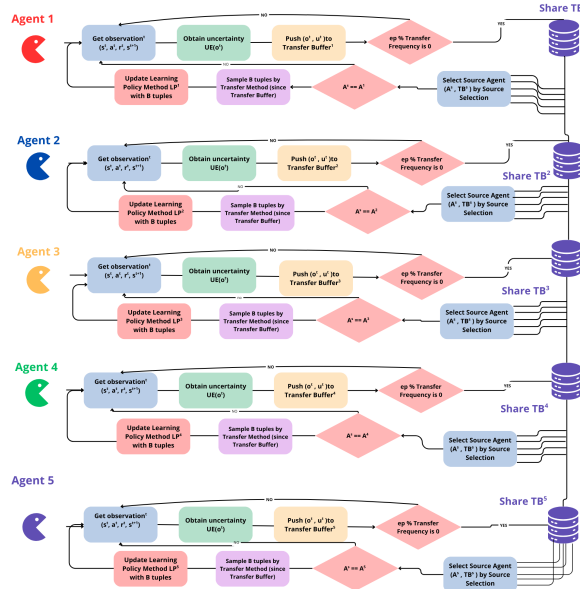


Figure 2: A detailed description of how EF-OnTL works as implemented in our MsPacman setup. Each of the five agents runs its own instance of the environment and, at each time step, obtains a transition (s^t, a^t, r^t, s^{t+1}) , calculates its uncertainty using the sars-RND estimator $UE(s^t)$, and sends the pair (s^t, u^t) to its own transfer buffer TB^i . Once the episode index is a multiple of the transfer frequency, all buffers are made available, and the source selector designates a source agent A^s ; subsequently, each of the other agents samples B tuples from TB^s according to the transfer criterion and updates its learning policy LP^i with them. Diagram produced from the algorithm description in [1]

In this paper, we have identified three questions that determine our path forward, and each requires a different type of measurement. The first, addressed in the previous subsection, is what changes the published code requires so that the documented experiments can be run. The second is how much

it costs to reproduce this algorithm on a standard computer rather than the dedicated server used in the original study. And finally, the third is whether the configuration used by the author retains the properties with which it was validated once transferred to MsPacman, a non-native multi-agent environment with a higher dimensionality than that of Cart-Pole.

3.3. Metrics

To answer the third question, we need to know whether the agents learned anything; otherwise, any statement about the transfer mechanism doesn't make much sense. Here, the reward distribution in the MsPacman environment complicates matters, as some episodes can exceed 2,000 points compared to a median of around 250 points per episode, and the mean is easily skewed by just a few of them. Therefore, in this paper we report the mean along with the median and the interquartile range, which describe the frequency of episodes and their dispersion, which can be affected by the tails. Using these three indicators, we can determine whether an improvement raises the entire distribution or simply broadens it. On the other hand, we obtained the confidence intervals using bootstrap resampling with 20,000 replicates. Finally, the interval in which we place the most confidence is where we evaluate the difference in means relative to the baseline. We decided to supplement all these metrics with Cliff's delta,

$$\delta = \frac{\#(x_i > y_j) - \#(x_i < y_j)}{mn}, \quad (1)$$

calculated across all episode pairs extracted from two scenarios (baseline or seeds). This metric counts how many times one condition outperforms the other without taking into account the magnitude of the individual differences, so a single lucky episode contributes no more than any other. With these two statistics, we answer different questions: the interval indicates how much better our results are on average, while the delta assesses how consistently they outperform.

With that established, we can examine the transfer mechanism. The EF-OnTL algorithm designates as the source the agent with the lowest average uncertainty, so it is valid to ask whether the agent it chooses is, in fact, a competent agent. That is why, at each transfer event. That is why, at each transfer event, we calculate each agent's average reward per episode over the previous 100 episodes and rank the agents from 1 (highest) to 5 (lowest); this ranking of the selected agent helps us determine how well the criterion performed. The general idea is that a criterion that always identifies the best agent would yield an average ranking of 1, whereas a uniform random selection among five agents would yield an expected average ranking of 3. Additionally, we would like to note that the 100-episode window episodes is our choice, and with only six events per run, the average ranking per run is an approximate measure; that is why we combine it with a second measure, where selecting the minimum of a quantity (uncertainty) identifies a competent agent only if that quantity decreases as competence increases, and we verify this by calculating, among the five agents in each transfer event, the Spearman correlation between the average uncertainty and the recent average reward, where our criterion is based on the fact that a negative correlation is the condition under which the criterion works.

The second question, which focuses on cost, requires a unit that remains constant despite changes in the environment. For example, episodes do not meet this criterion because their duration depends on the environment's policy and termination rules; therefore, the same number of episodes can correspond to very different amounts of computation. That is why we record the cumulative number of steps in the environment per agent, which is comparable across environments and with the training budgets reported in other publications in the literature. By dividing the actual execution time by the environment's steps in successive blocks of 50 episodes, we obtain a cost per step, expressed as the mean and standard deviation across those blocks; furthermore, using the standard deviation, we can infer whether that cost is stable enough for the metric to predict the cost of a longer execution. Finally, we compare the average episode duration between the first and last blocks of each run, which shows how the cost per episode evolves as the policy improves and, therefore, whether the number of episodes can serve as a unit for planning the computation.

4. Materials and methods

We conducted the evaluation in two environments, with Cart-Pole serving as the primary reference. This is because Cart-Pole is the environment in which the configuration under study was validated, and it is the environment for which the original work presents a non-native multi-agent formulation, in which five parallel instances of a single-agent environment replace a cooperative system [1]. On our end, ALE/MsPacman-v5 is the target; with this environment, we have reproduced the same non-native setup with five parallel instances, but the observation space increases from 4 to 128 dimensions, and the reward becomes the arcade score, unlike in Cart-Pole. For the environment studied in this paper, observations are taken from the Atari 2600’s RAM rather than from pixels, resulting in a 128-dimensional vector composed of 8-bit integers that encode internal game variables, the individual meanings of which are not documented [16]. By selecting this environment, we ensure that the comparison with Cart-Pole remains meaningful, since both environments present the agent with a feature vector rather than an image, and it avoids introducing a more complex learning process, such as the use of algorithms that employ convolutional networks for the recognition of complex states from images.

4.1. Execution environment

Table 2 shows the hardware specifications. All experiments were executed on a consumer laptop, running on Windows 11, with Python 3.13.10, PyTorch 2.6.0+cu124, Gymnasium 1.3.0 version and ale-py 0.12.0. This contrasts with the original study, which reports training on a server equipped with a Xeon Gold 5220R, four RTX 3070 GPUs and 1,535 GiB of RAM, while stating that a GPU is not mandatory for replication because the bottleneck lies in CPU capacity [1]. Quantifying the cost of that replication on non-dedicated hardware is one objective of this work.

Table 2

Hardware specifications for simulation execution.

Component	Specification
CPU	AMD Ryzen 7 7445HS w/ Radeon 740M Graphics
RAM	32 GB
GPU	NVIDIA GeForce RTX 4050 Laptop GPU

4.2. Configuration

The Dueling DQN architecture, its optimizer configuration, and the EF-OnTL transfer parameters used are those documented for Cart-Pole in the original paper, adopted without any adjustments. However, it is important to note that three parameters necessarily differ because they are determined by the nature of the environment. These are: the network’s input width (from 4 to 128), the number of actions (from 2 to 9), and the sars-RND input, whose size in the implementation is $2n_s + n_a + n_{\text{rew}}$ and therefore ranges from 10 to 258. On the other hand, the hidden layers, the learning rate, the target network replacement interval, the exploration rule, the buffer sizes, the transfer budget, the transfer frequency, and the transfer start episode remain unchanged.¹ Because no re-tuning was performed, any difference in behaviour between the two domains cannot be attributed to network tuning.

¹An intentional modification was made: the RAM observations were divided by 255 to place the state components in the interval $[0, 1]$. On the other hand, the rewards were left unscaled and uncapped. As a result, the reward for the sars-RND input can exceed the normalized states by three orders of magnitude—an asymmetry that does not exist in Cart-Pole, where the reward per step is 1. We will return to this in section 6.

Parameter	Cart-Pole	Pacman
Agents (N)	5	5
Transfer Frequency	200	200
TB Capacity	10,000	10,000
Source Selection (SS)	avg. uncertainty	avg. uncertainty
Filtering Criteria (TCS)	high Δ -conf	high Δ -conf
Transfer Budget (B)	5,000	5,000
Ep. Start Transfer	600	600
Max Episode	1,800	1,800
Max Timestep	400	not enforced [†]

Table 3: EF-OnTL parameters. [†]No episode-length cap was applied beyond the ALE default; the longest observed episode was 1,317 agent steps (see Section 6).

Parameter	Cart-Pole	Pacman
Input Layer	FC(4, 128)	FC(128, 128)
Hidden Layer	FC(128, 64)	FC(128, 64)
Value Layer	FC(64, 1)	FC(64, 1)
Adv. Layer	FC(64, 2)	FC(64, 9)
Activation	ReLU	ReLU
Loss	MSE	MSE
Optimiser	Adam	Adam
Learning Rate	1×10^{-5}	1×10^{-5}
Betas	(0.9, 0.999)	(0.9, 0.999)
Gamma	0.999	0.999
Target Replacement	1,000	1,000
Policy Update	every timestep	every timestep
Exploration	thompson ($T=1$)	thompson ($T=1$)
Replay Buffer Size	10,000	10,000
Batch Size	32	32
Ep. Start Training	100	100

Table 4: Dueling DQN parameters. Only the input and advantage-layer widths differ, both fixed by the environment.

Setting	Value
Environment ID	ALE/MsPacman-v5
Observation type	ram
Observation space	Box(0, 255, (128,), uint8)
Action space	Discrete(9)
Frame skip	4
Sticky actions (p)	0.25
max_episode_steps	None
State normalisation	division by 255
Reward clipping	none

Table 5: ALE/MsPacman-v5 configuration. All values are Gymnasium/ALE defaults except state normalisation.

5. Results

Policy training begins in episode 100; during the preceding episodes, the agents act according to a randomly initialized network, sampling actions using the Thompson method (with Temperature = 1) on their outputs. Therefore, their performance should be similar to that of the uniform random game, though not identical to it. The average reward per episode we obtained during the first 100 episodes was 270.4 (seed 42) and 261.6 (seed 7), compared to 261.2 for the measured random-action baseline; furthermore, the median was 240 in all three cases. On the other hand, the differences in the means were +9.18 points (95% bootstrap CI $[-3.4, +22.0]$) and +0.38 points ($[-10.8, +12.0]$), with a Cliff δ of +0.0572 and +0.0099.

Once we had applied the metrics between the two simulations run for this study in comparison to the baseline, we decided to apply the same process with a different approach, since we analyzed the final state of each run using its last 200 episodes (1,000 episode records per run) and compared it to the baseline measured using random actions. The goal is to evaluate the difference in learning between the system that uses EF-OnTL and the baseline system of random decisions; in simple terms, to assess whether our system performs better than simply making random decisions. Table 6 below details the results of this analysis.

Scenario	Mean [95% CI]	Median	Q25–Q75	Δ mean [95% CI]	Cliff’s δ
Random baseline ($n=2,500$)	261.2 [256, 267]	240	200–290	—	—
Seed 42, final 200 ep ($n=1,000$)	355.2 [341, 370]	290	230–430	+93.9 [+79, +109]	+0.328
Seed 7, final 200 ep ($n=1,000$)	339.0 [325, 354]	280	190–450	+77.8 [+63, +93]	+0.195

Table 6

Final performance against the measured random-action baseline. Confidence intervals are bootstrap estimates over 20,000 replicates; Cliff’s δ is computed over all pairs of episodes between each condition and the baseline.

The results show that both runs end up above the baseline. The median increases from 240 to 290 and 280, respectively; the difference between the means is +93.9 points [+79, +109] and +77.8 points [+63, +93], and Cliff’s δ is +0.328 and +0.195. None of the difference intervals include zero, unlike the pre-training intervals mentioned in the opening paragraph of this section. Furthermore, we note that the two seeds clearly differ, illustrating the variability across runs that the original study addressed by including twenty runs. Focusing specifically on the differences observed for each of the seeds, we note that for seed 42, all three quartiles increase from 200/240/290, in contrast to the baseline, which is 230/290/430; thus, the entire distribution shifts upward, although the gain is greater at the upper tail. On the other hand, for seed 7, the improvement is limited to the upper end: the quartiles shift to 190/280/450, and the lower quartile falls slightly below the baseline. Finally, we observed that in both runs, the difference between the first and third quartiles more than doubles. Therefore, seed 7 improves primarily by generating occasional high-scoring episodes, while seed 42 also increases the reward from its weaker episodes.

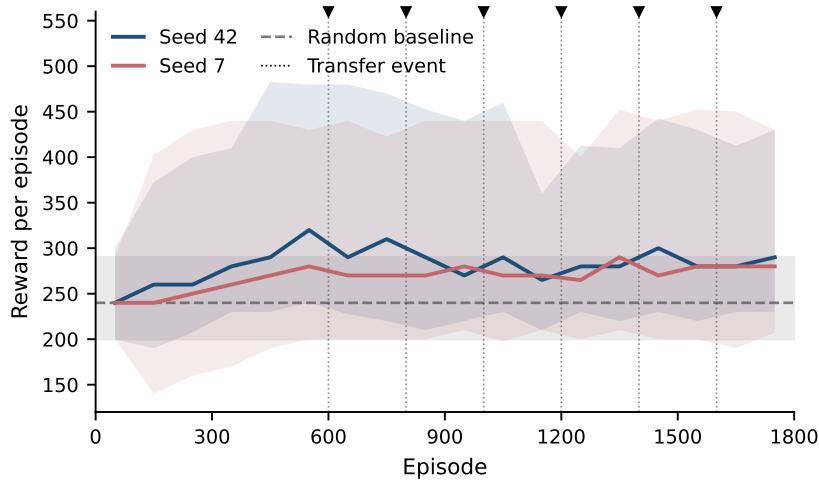


Figure 3: Reward per episode during training for both MsPacman and baseline runs. Each point was calculated based on a block of 100 consecutive episodes from the five agents—that is, 500 episode records per block. The solid lines represent the median of that block, and the shaded band ranges from the first to the third quartile. The dashed line and gray band refer to the baseline used. Finally, the dotted lines mark the six transfer events.

Figure 3 visually illustrates what we discussed earlier: both runs reach their final level early and remain there, while the median per block stays within the interquartile range of the random game. The bands also widen as training progresses, while the reference band remains constant: the agents become more variable, not just better. Additionally, for the purposes of this study, it is useful to compare our results with those obtained in other studies on DQN for this game, since they are approximately one order of magnitude higher than the scores observed in those studies [17], which were obtained with 5×10^7 steps reported in [18] environment. In contrast, our agents each accumulate approximately 1.02×10^6 steps, about 49 times fewer. This suggests—and allows us to hypothesize—that the observed performance is consistent with this difference in the training budget rather than with a failure of the learning algorithm.

5.1. Source selection

We have addressed an initial question: evaluating how well the agents have learned by using the configuration transferred from the Cart-Pole environment to the MsPacman environment. However, it is not enough to analyze this based solely on previously measured metrics such as means, differences, medians, interquartile ranges, etc. In this work, we also focus on evaluating how well the selection criterion applied in the MsPacman environment works. As we know, EF-OnTL designates as the transfer source the agent with the lowest average epistemic uncertainty regarding its recent experience. Therefore, we decided to evaluate whether this criterion identifies competent sources by comparing, in each transfer event, the selected agent with the other four. Agent performance is rated from 1 (highest) to 5 (lowest); the method used to calculate this performance is detailed in our section 3. Table 7 below shows the rankings assigned to the different agents at each transfer step and their final mean.

Run	Environment	Rank of selected source	Mean rank
CP-C	Cart-Pole	4, 3, 3, 1, 1, 3	2.50
MP-1	MsPacman (seed 42)	4, 4, 5, 5, 3, 2	3.83
MP-2	MsPacman (seed 7)	2, 4, 2, 5, 5, 5	3.83

Table 7

Ranking of the agent selected as the transfer source in each of the six transfer events, based on its recent performance. More details on the meaning of the values can be found in the section 3.

In Cart-Pole, the criterion performs better than random selection (average rank of 2.50) and selects

the best-performing agent in episodes 1,200 and 1,400. MsPacman, on the other hand, performs worse than chance in both runs, with a mean rank of 3.83, and selects the agent with the lowest rank in five of the six events measured in seed 7. In our evaluations, Agent 3 ranked first in nine of the twelve events in both runs of MsPacman and was never selected as a source. To evaluate this observed difference using a metric based on ranges, we calculated Spearman’s correlation between the average uncertainty and the recent performance of the five agents in each event; this metric is presented in greater detail in section 3. The data obtained from the correlation for each transfer event are shown in Table 8

Run	600	800	1000	1200	1400	1600	Mean	Negative
CP-C (Cart-Pole)	−0.20	−0.50	+0.60	−0.10	−0.46	−0.60	−0.210	5/6
MP-2 (MsPacman)	+0.10	+0.30	0.00	+0.60	0.00	+1.00	+0.333	0/6

Table 8

Spearman correlation between average uncertainty and recent average reward among the five agents, for each transfer event; we note that uncertainty values were not recorded for the MP-1 run and are therefore omitted.

The results show that, in Cart-Pole, the correlation is negative in five of the six events, so lower uncertainty corresponds to higher performance, and the criterion selects accordingly. In contrast, in MsPacman, the correlation is non-negative in all events; therefore, the criterion selects based on lower performance, which is consistent with the rankings in Table 7. It is also interesting to note that the magnitude of the estimated uncertainty behaves differently in the two environments. In Cart-Pole, the average across the five agents drops from 1.73×10^{-2} in the first transfer event to 6.4×10^{-6} in the last, a factor of approximately 2,700, and decreases uniformly across the six events. In MsPacman, it remains between 3.5×10^{-3} and 4.4×10^{-3} at all times, with no downward trend. We mention this final result without attributing an apparent cause to it, since determining why the estimator converges in one environment and not in the other would require measurements that we did not collect and is beyond the scope of this work.

5.2. Computational cost

Finally, we focused on measuring the cost of implementing the consumer hardware described in section 4, based on the resources used by the author, as described in that same section.

Run	elapsed time	Env. steps	Steps/agent	s per step	Episode length
CP-C (Cart-Pole)	3.52 h	1,194,288	238,857	0.0121 ± 0.0022	21.7 → 394.2
MP-1 (MsPacman)	15.58 h	5,123,396	1,024,679	0.0113 ± 0.0006	503.1 → 584.3
MP-2 (MsPacman)	15.55 h	5,105,530	1,021,106	0.0113 ± 0.0001	510.8 → 550.6

Table 9

Computational cost of the three runs of EF-OnTL, each consisting of 1,800 episodes with five agents. The table details values derived from these runs, such as the cost per step of the environment, measured using the mean and standard deviation, and also shows the episode duration for the first and last blocks.

The results show, first of all, that the cost per step in the environment is similar in both environments, 0.0121 s and 0.0113 s even though MsPacman has a 128-dimensional input compared to four. Therefore, we can conclude that the cost per step is determined primarily by the learning mechanism and not by the environment used. In terms of variations, the Cart-Pole runs are much shorter: approximately 5,000 steps compared to 135,000 for MsPacman. Second, the two runs of MsPacman show a difference of less than 0.35% in cumulative steps and 0.22% in real time, despite the different seeds, which allows us to partially predict the cost, since more runs with different seeds are needed to confirm a conclusion. Therefore, the cost is predictable based on the step count, rather than being merely describable a posteriori. Third, and most relevant for anyone planning a replication, it is evident that the same 1,800-episode protocol takes 3.52 hours in one environment and 15.6 hours in the other, due to episode duration; in Cart-Pole, the average episode duration increases by a factor of 18.2 throughout training, while in MsPacman it increases by a factor of 1.16. As a preliminary observation, we note that the main

reason is that in Cart-Pole, the reward is the episode duration; thus, a policy that improves produces longer—and therefore more computationally expensive—episodes. In MsPacman, on the other hand, episode duration is limited by the three available lives and is largely independent of the quality of the policy. Therefore, an episode budget does not determine computational cost and does not transfer between environments, whereas a step budget does. Finally, a difference is identified that affects the training budget. Using 1,800 episodes, 238,857 environment steps per agent are generated in Cart-Pole, but 1,024,679 in MsPacman—a fourfold difference that is due to the fact that the number of episodes is kept fixed by the nature of EF-OnTL. Compared to the training budgets typically used in this field, the figure for MsPacman is still approximately fifty times smaller than the 5×10^7 steps reported for DQN [18].

Finally, we noticed two patterns in one of the test series that did not appear in the other, and we pointed them out to clarify what a single test series allows us to add as a preliminary observation. In seed 42, the average reward peaks at 395.5 in episodes 400–599, then declines throughout the process to 374.0, 358.2, and 356.2, stabilizing around 350 during the remaining 1,200 episodes. On the other hand, we can see that in seed 7, the average reward rises to 347.5 during the same interval and remains between 335 and 352 without any decline. It is interesting to note these changes in the pattern of the average reward throughout the algorithm; however, since we do not have a transfer-free evaluation, we cannot draw conclusions regarding performance due to transfer. From another perspective, we can see that the two series also differ in terms of whether or not the agents diverge, since the Spearman correlations—applied this time for each agent—between the episode’s outcome and the reward obtained for that episode show mixed signs in seed 42 (-0.119 , -0.077 , -0.013 , $+0.191$, $+0.230$: three agents perform worse, while two perform better), but are uniformly positive in seed 7 ($+0.023$ to $+0.110$: all five perform better). Therefore, our data do not allow us to determine whether the exchange of experiences standardizes the agents or differentiates them.

6. Discussion and Limitations

Our results distinguish between two aspects that are often confused when reusing an algorithm: whether it runs and whether its components continue to do what they were designed to do. In this work, we succeeded in executing the EF-OnTL algorithm based on the assumption that we had corrected the problems listed in section 3; furthermore, the algorithm ran to completion in both environments, selected sources within the configured intervals, and transferred the configured number of experiences. We also demonstrated that the agents learned, since in both MsPacman runs we outperformed the random-action baseline measured under the same configuration, based on the metrics presented in the section 5, thus allowing us to conclude that no problems occurred during execution. However, toward the end of obtaining our results, we encountered certain points worth discussing regarding our implementation.

The first point concerns an inconsistency in the descriptions published by the original author, since the original work reports different learning rates for the same Cart-Pole experiment: a value of 1×10^{-4} in the thesis [1] and 1×10^{-5} in the conference paper [8]. Ultimately, in this paper we chose to use the value of 1×10^{-5} , which matches the default value of the published configuration.

The last issue relates to a flaw in our own experimental environment. While analyzing our source code, we noticed and reported a corresponding defect on our part. Specifically, regarding the maximum episode duration that was loaded from the configuration file and displayed in the experiment summary, but was never passed to the environment constructor, so no episode duration limit was applied beyond the ALE default value of 108,000 emulator frames, equivalent to 27,000 agent steps with a four-frame step size. We initially set a maximum of 1,600 agent steps; however, we did not notice this problem during execution, since the longest episode observed in both runs was 1,317 agent steps, meaning no episodes were affected; therefore, we note that the parameter was not applied based on the configuration set in our code (Table 3)

Despite the two issues mentioned, they did not affect the final results reported in this paper in any

way, and they allowed us to observe that the source selection criterion did not retain the property on which the author originally based it; furthermore, we were able to corroborate this in the results we obtained for Cart-Pole and MsPacman. The results we obtained for MsPacman do not indicate a specific type of failure, since the algorithm does not crash, does not generate warnings, and does not report reduced performance; anyone reusing the algorithm would observe that the agents perform better than in the random game and would conclude that the method works. However, the change in criteria is not so obvious and is only visible if the selection step is instrumented and documented and the chosen source is compared with the alternatives, which is not part of the algorithm’s normal output. This is consistent with what has been discussed for similar methods, since the rules determining which agent should share its experience are heuristic approximations of utility, and their authors note that such approximations can fail in asymmetric domains [4]. Our results provide an empirical example of this type of failure for a different approximation in a domain outside its previously validated regime. The cost analysis reveals a second problem. While it is clear that the elapsed time depends linearly on the cumulative number of steps in the environment, with a constant that varies very little between the two environments, the same protocol consisting of 1,800 episodes takes 3.5 hours in one environment and 15.6 hours in the other, where this discrepancy is due to the duration of the episodes, which is multiplied by 18 during training on Cart-Pole and by 1.2 on MsPacman. Therefore, an episode-based protocol does not determine a fixed amount of computation, and its transfer to a new domain modifies the training budget, thus complementing existing findings that optimal hyperparameter values do not persist across different training budgets [3].

Beyond describing the previous problems, our measurements also highlight the areas where the criterion falls short. The rule classifies agents solely based on uncertainty, which means it can only work as long as uncertainty and performance evolve in opposite directions—and that is precisely what ceases to happen in MsPacman. The rule itself never changes, so what changes is whether the value it reads still contains the information it is supposed to contain. Viewed this way, the difficulty does not lie in the fact that uncertainty is a negative signal, but rather in the fact that it is used in isolation, since the selection phase does not take into account each agent’s actual performance and, therefore, there is no way to detect when the hypothesis ceases to be valid. A rule that combines uncertainty with some measure of recent performance would not have this blind spot, and our own data suggest as much, since the agent that performed best in nine of the twelve transfer events was never chosen; therefore, any weight assigned to the reward would have skewed the selection of the chosen agents. This would not even require a new component, since the original implementation already provides a criterion for better performance, suggesting that the two signals could be combined rather than treated as alternatives. However, we do not propose a specific combination, and our data cannot tell us what weighting would be appropriate or whether such a criterion would actually improve the final performance, since establishing it would require the control without transfer and the additional runs discussed below. What our results do support is more modest: when a selection rule depends on a single indicator, it is worthwhile to combine it with some way of verifying that the indicator continues to behave as expected, especially when the algorithm is applied to a domain other than the one in which it was validated.

6.1. Limitations

In this paper, we present one simulation per configuration and two seeds for MsPacman, compared to the twenty simulations in the original study. The consequences are evident in our own data: the two runs of MsPacman differ in effect size relative to the baseline ($\delta = +0.328$ and $+0.195$), and two patterns present in one run were absent in the other, as discussed in this section. As mentioned in the section, studies on statistical power in deep reinforcement learning show that many more independent runs are needed before we can state with certainty that a performance difference exists [11], and our two runs are not sufficient to estimate the variance required for such a calculation. Properly quantifying this requirement would involve running enough seeds to obtain a usable estimate for each condition, along with the no-transfer control described below, making this limitation one of the strongest and

most important points to be addressed in future work. Therefore, we make no claims about the effect of transfer on performance given the nature of our data. Furthermore, we did not conduct a control experiment without transfer, which is why nothing stated in this article should be interpreted as a claim as to whether the exchange of experiences helped or hindered the agents in MsPacman. Similarly, we did not determine why the uncertainty estimator behaves differently in the two environments, since identifying the cause would require measurements that we did not collect. On the other hand, it is important to note two deviations from the original configuration: RAM observations were divided by 255, and rewards were not scaled, meaning that the reward component of the sars-RND input can exceed the normalized state components by three orders of magnitude—an asymmetry absent in Cart-Pole. Furthermore, the maximum episode length configured for Cart-Pole was not applied, so episodes were truncated at 500 steps instead of 400. Neither of these situations was intentional; both are described in this section. Finally, our MsPacman agents accumulate approximately 1.02×10^6 environment steps each, a value far lower than the budgets typically used in this field. Therefore, the observed source-selection behavior is established under a short training budget, and we cannot say whether it would persist under a longer one.

Regarding the statistical analysis, because the results were obtained at the episode level, and because learning is continuous, meaning that learning can improve in a given episode, the observations of each performance are not independent. Therefore, the results can be considered descriptive, and caution should be exercised when generalizing the data. Given the continuous nature of the learning process, it is expected that more performances will be conducted to achieve statistical consistency.

7. Conclusion

In this work, we ran the published implementation of EF-OnTL on consumer hardware and transferred its configuration and protocol—validated using the Cart-Pole problem—without modifications to ALE/MsPacman-v5 under RAM observations. The system’s execution required five corrections, one of which involved a discrepancy from the originally documented protocol, meaning that the transfer began earlier than specified in the published experimental description. Once the protocol was restored, the algorithm ran correctly in both environments, and the agents learned; we verified through metrics that both MsPacman runs ended with a result significantly better than a random-action baseline established under the same configuration. However, the source selection criterion does not preserve the proposed property. While in Cart-Pole it performs better than random selection and the correlation between uncertainty and recent performance is negative in five of the six transfer events, MsPacman, on the other hand, performs worse than random selection in both runs, with an identical average score of 3.83 out of 5, and the correlation is non-negative in all events. On the other hand, cost measurements show that real-time performance is linear with respect to the cumulative number of environment steps, at a rate that differs only slightly between the two environments, while the same protocol consisting of 1,800 episodes takes 3.5 and 15.6 hours, respectively. Furthermore, we present a simulation for each configuration and no control group without transfer, so we make no claims about whether the exchange of experiences helped or hindered performance. However, we propose as a conclusion some measures that would help resolve the open questions raised by this study. One of these is the implementation of a no-transfer control condition in MsPacman, which would allow us to separate the effect of transfer from that of training. And the last of these is to repeat both experiments with a larger sample size, which would allow us to determine whether the reversal is a property of the domain or of the regime in which we observe it.

Declaration on Generative AI

During the preparation of this work, the authors used Claude Opus 5 and Elicit to perform grammar and spelling checks and content enhancement. After using these tools/services, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

- [1] A. Castagna, Expert-free online transfer learning in multi-agent reinforcement learning, 2025. URL: <https://arxiv.org/abs/2501.15495>. arXiv: 2501.15495.
- [2] A. Patterson, S. Neumann, R. Kumaraswamy, M. White, A. White, The cross-environment hyperparameter setting benchmark for reinforcement learning, 2024. URL: <https://arxiv.org/abs/2407.18840>. arXiv: 2407.18840.
- [3] J. Obando-Ceron, J. G. M. Araújo, A. Courville, P. S. Castro, On the consistency of hyper-parameter selection in value-based deep reinforcement learning, 2024. URL: <https://arxiv.org/abs/2406.17523>. arXiv: 2406.17523.
- [4] M. Gerstgrasser, T. Danino, S. Keren, Selectively sharing experiences improves multi-agent reinforcement learning, 2024. URL: <https://arxiv.org/abs/2311.00865>. arXiv: 2311.00865.
- [5] W. Shang, L. Espeholt, A. Raichuk, T. Salimans, Agent-centric representations for multi-agent reinforcement learning, ArXiv abs/2104.09402 (2021). URL: <https://api.semanticscholar.org/CorpusID:233296399>.
- [6] J. Hernandez, M. F. Pollo-Cattaneo, D. H. Peluffo-Ordóñez, H. Florez, Cognitive architecture for learning in autonomous agents: A study in discrete navigation, in: International Conference on Applied Informatics, Springer, 2025, pp. 3–19.
- [7] Y. Burda, H. Edwards, A. Storkey, O. Klimov, Exploration by random network distillation, 2018. URL: <https://arxiv.org/abs/1810.12894>. arXiv: 1810.12894.
- [8] A. Castagna, I. Dusparic, Expert-free online transfer learning in multi-agent reinforcement learning, 2023. URL: <https://arxiv.org/abs/2303.01170>. arXiv: 2303.01170.
- [9] J. Hernandez, D. H. Peluffo-Ordóñez, H. Florez, How do autonomous software agents learn? a systematic literature review, PeerJ Computer Science 12 (2026) e3756.
- [10] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, D. Meger, Deep reinforcement learning that matters, 2019. URL: <https://arxiv.org/abs/1709.06560>. arXiv: 1709.06560.
- [11] C. Colas, O. Sigaud, P.-Y. Oudeyer, How many random seeds? statistical power analysis in deep reinforcement learning experiments, 2018. URL: <https://arxiv.org/abs/1806.08295>. arXiv: 1806.08295.
- [12] K. Clary, E. Tosch, J. Foley, D. Jensen, Let's play again: Variability of deep reinforcement learning agents in atari environments, 2019. URL: <https://arxiv.org/abs/1904.06312>. arXiv: 1904.06312.
- [13] J. Adkins, M. Bowling, A. White, A method for evaluating hyperparameter sensitivity in reinforcement learning, 2025. URL: <https://arxiv.org/abs/2412.07165>. arXiv: 2412.07165.
- [14] T. Eimer, M. Lindauer, R. Raileanu, Hyperparameters in reinforcement learning and how to tune them, 2023. URL: <https://arxiv.org/abs/2306.01324>. arXiv: 2306.01324.
- [15] T. Fiskus, U. Shaham, Turning sand to gold: Recycling data to bridge on-policy and off-policy learning via causal bound, 2025. URL: <https://arxiv.org/abs/2507.11269>. arXiv: 2507.11269.
- [16] Farama Foundation, Arcade Learning Environment Documentation: MsPacman, https://ale.farama.org/environments/ms_pacman/, 2023. Accedido en 2026.
- [17] H. van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, 2015. URL: <https://arxiv.org/abs/1509.06461>. arXiv: 1509.06461.
- [18] K. Young, T. Tian, Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments, 2019. URL: <https://arxiv.org/abs/1903.03176>. arXiv: 1903.03176.