Combining Pre-Trained Models for Enhanced Feature Representation in Reinforcement Learning

Elia Piccoli[1,2,3], **Malio Li**[1], **Giacomo Carfì**[1], **Vincenzo Lomonaco**[1], **Davide Bacciu**[1]
{elia.piccoli, malio.li}@phd.unipi.it g.carfi1@studenti.unipi.it
{vincenzo.lomonaco, davide.bacciu}@unipi.it

[1]**Department of Computer Science, University of Pisa, Italy**
[2]**Department of Computing Science, University of Alberta, Canada** [3]**Amii**

Abstract

Reinforcement Learning (RL) focuses on maximizing the cumulative reward obtained via agents' interaction with the environment. RL agents do not have any prior knowledge about the world, and they either learn from scratch an end-to-end mapping between the observation and action spaces or, in more recent works, are paired with monolithic and computationally expensive Foundational Models. How to effectively combine and leverage the hidden knowledge of different pre-trained models simultaneously in RL is still an open and understudied question. In this work, we propose Weight Sharing Attention (WSA), a new architecture to combine embeddings of multiple pre-trained models to shape an enriched state representation, balancing the tradeoff between efficiency and performance. We run an extensive comparison between several combination modes showing that WSA obtains comparable performance on multiple Atari games compared to end-to-end models. Furthermore, we study the generalization capabilities of this approach and analyze how scaling the number of models influences agents' performance during and after training.

1 Introduction

In a typical RL setting, agents receive as input the raw representation of the state, without any additional information on the elements that characterize it. The learning process, focused on creating a policy - a mapping from states to actions that determines agent behavior - implicitly hides a significant effort related to understanding how to process the input representations. While one of the promises of deep learning algorithms is to automatically construct well-tuned features, such representation might not emerge from end-to-end training of deep RL agents. Agents learn how to solve the task while indirectly learning how to process and extract useful information from the input features. Although this approach has been the solution for several works, it adds an additional layer of complexity over RL algorithms, eventually requiring significant computational resources.

Several works analyze the differences between the learning process of humans and RL agents, highlighting how prior knowledge can influence the learning curve (Tenenbaum, 2018; Dubey et al., 2018). Humans and agents define the learning pattern by incrementally improving their abilities. Human learning often involves incremental skill development, leveraging prior knowledge, and adapting strategies based on experience. In new environments with few interactions, humans can leverage their prior knowledge to understand the effects of their actions. This allows them to focus mostly on learning a good policy, rather than also learning how to interpret the environment. How can we encode and represent these information in RL agents?

In light of this question, pre-trained models present themselves as good candidates. They are trained on extensive datasets in a self-supervised fashion, enabling them to learn rich and generalizable representations. These representations can be leveraged in RL to accelerate the learning process and improve performance on complex tasks. By using models' knowledge, RL agents can achieve higher efficiency and effectiveness, even in environments with sparse rewards or limited data. The objective of this paper is to study how different latent embeddings can be `combined` to compute a rich and informative representation of the environment. In this way, we can focus on `how` RL agents can exploit prior knowledge provided by multiple pre-trained models and improve the learning process of the actual policy. Here we report the main contributions of this work:

- We propose Weight Sharing Attention (WSA) as a combination module to incorporate different encodings coming from several pre-trained models. This approach achieves comparable performance with end-to-end solutions while also adding robustness and dynamic scalability.
- We run experiments across multiple Atari games studying the behavior of 7 different combination modules. To the best of our knowledge, this is the first analysis focused on the combination of multiple pre-trained models' latent representation in RL.
- We show that without any fine-tuning of hyperparameters, or exploiting a small search, our methodology can achieve comparable results with established RL end-to-end solutions. Moreover, we prove and address the problems of *distribution shifts*, *failing robustness*, and *adaptation to evolving knowledge*, which presents themselves as the key challenges.

2 Related Work

The idea of improving RL agents feature representation by providing information already processed by other models is not new. Some work focus on feature representations that can be detached from the learning process by having a module that is able to extract relevant information. Agents learn how to perform tasks by integrating different enriched elements, rather then needing to learn the state space from scratch. Some works use human-defined data (Bramlage & Cortese, 2022) or leverage pre-trained models' feature representation learned on different tasks, for example, computer vision (Shah & Kumar, 2021; Blakeman & Mareschal, 2022; Yuan et al., 2022) or other RL tasks (Kumar et al., 2023). Several approaches learn a general representation from collected data by applying different self-supervised methodologies, which can also be fine-tuned during policy learning (Goel et al., 2018; Anand et al., 2019; Kulkarni et al., 2019; Schwarzer et al., 2021; Xiao et al., 2022; Montalvo et al., 2023; Majumdar et al., 2023). In order to be effective, these representations should be as general as possible, without having any bias with respect to the task the agent should solve. Other works try to overcome this problem by presenting methodologies to build representations that are reward free - meaning they are unbiased with respect to the task (Stooke et al., 2021) - or by adding auxiliaries objectives to help shape the latent representation (Lan et al., 2023). Despite achieving good results in model-free RL, Schneider et al. (2024) studies the limitations of using one single model to gather pre-trained visual features in model-based RL. Appendix A provides an extended discussion comparing our proposed architecture with recent approaches that integrate multiple pre-trained models.

3 Weight Sharing Attention (WSA)

Many previous works share the idea of leveraging pre-trained models to enhance the state representation, allowing world knowledge transfer and simplifying the RL training process. Differently from these studies, our work focuses on how to use multiple models at the same time and how to effectively combine their latent representations to improve agents performance. Our methodology enhances the efficiency and effectiveness of RL agents by leveraging a set of pre-trained models tailored to the current task, serving as the prior knowledge needed to generate diverse representations of the environment.

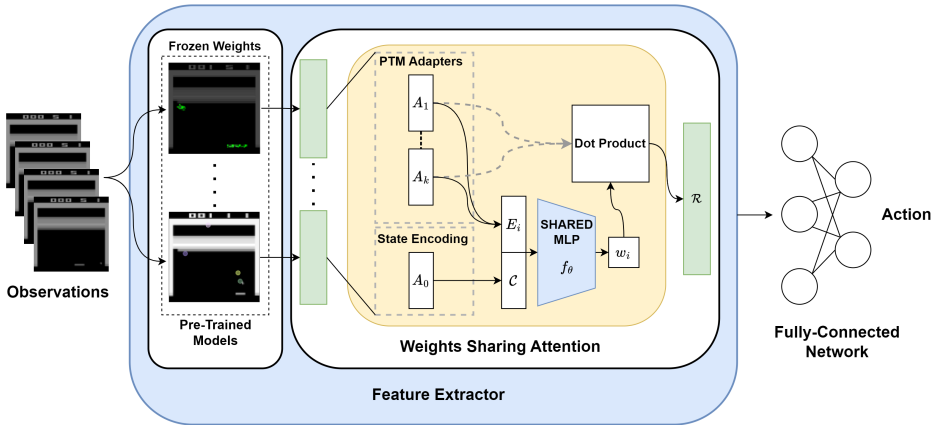


Figure 1: Schematic representation of the main pipeline from observations to actions and WSA architecture. The game frames are stacked together and passed as input to the pre-trained models and their adapters $\mathcal{A}_1, \ldots \mathcal{A}_k$ to obtain the embeddings of the current state. A State Encoder $\mathcal{E}$ and its adapter $\mathcal{A}_0$ compute an encoding of the state, that will be used as context $\mathcal{C}$. A shared MLP takes as input the context $\mathcal{C}$ and the representation of the i^{th} model, E_i, and predicts its weight. The final enriched representation $\mathcal{R}$ is obtained from the weighted summation between each weight w_i and the embedding E_i. Finally, $\mathcal{R}$ is given as input to agents' `Fully-Connected Network`.

The architecture of our agents is divided into two main components, as shown in Figure 1. The first one, delimited by the blue box, handles the processing of the current observation into a latent representation. While agents interact with the environment, the current observation is collected and processed through each pre-trained model. This process yields multiple perspectives or biases of the world, each encoded uniquely based on the specific model. The challenge lies in integrating these disparate views into a cohesive and enriched representation. To achieve this, we employ a specialized combination module, yellow box, that synthesizes the information from various models into a single latent representation. The intuition is that the enriched representation provides a comprehensive understanding of the current state of the environment, derived from the collective insights of the pre-trained models. By leveraging this information, our approach significantly reduces the burden on the RL agent to learn the environmental representation from scratch, allowing it to focus more on refining the action-mapping process. To facilitate the integration of different pre-trained models, we propose Weight Sharing Attention (WSA). The approach can be adapted and extended to any number and type of pre-trained models while maintaining its structure. Figure 1 outlines the main pipeline and components required by our module, while Algorithm 1 reports the pseudocode for the actual implementation.

Each pre-trained model is equipped with an adapter $\mathcal{A}$ - a linear layer followed by a non-linearity - that maps its latent representation into an embedding E of predefined size. This shared space ensures that different model outputs are compatible and can be seamlessly integrated. Adapters' parameters are optimized during training. Among the available models, one is designated as the State Encoder $\mathcal{E}$. The State Encoder processes the current state and produces a fixed-size embedding referred to as the *context* $\mathcal{C}$. The idea of introducing $\mathcal{C}$ is to guide the model in assigning contextual relevance to each pre-trained model based on the specific characteristics of the current input. Specifically, the context $\mathcal{C}$ is concatenated with each model's adapter embedding and passed through a shared Multi-Layer Perceptron (MLP) f_θ (referred to as the `Shared Weight Network`). This network computes a scalar weight w_i for each model, indicating its relative contribution to the final representation. By conditioning the weights on the context, it introduces a bias that the model can leverage to prioritize different sources of information. The State Encoder $\mathcal{E}$ can be replaced with any differentiable function, learned module, or even a heuristic-based mechanism that produces a suitable context, i.e. an insightful characterization of the current state. The weight vector W is

Algorithm 1 Weight Sharing Attention

In blue we highlight the components updated during training.

Require: Observation $\mathcal{O}$, Set of Pre-Trained Models Ψ, State Encoder $\mathcal{E}$

$\mathcal{C} = \mathcal{A}_0(\mathcal{E}(\mathcal{O}))$ ▷ Compute context representation $\mathcal{C}$

for pre-trained model ψ in Ψ **do**

  $x = \psi_i(\mathcal{O})$ ▷ Forward pass *i-th* pre-trained model using current observation

  $E_i = \mathcal{A}_i(x)$ ▷ Use the adapter to compute the resized embedding E_i

  $w_i = f_\theta(\mathcal{C}, E_i)$ ▷ Compute the weight w_i using the `Shared Weight Network` f_θ

end for

$W = \frac{W}{\|W\|_1}$ ▷ Normalize weights W using L1-norm

$\mathcal{R} = \sum_{i=0}^{|\Psi|} w_i * E_i$ ▷ Compute final representation: weights W, embeddings $E \rightarrow W \cdot E$

normalized by its L1 norm to produce a probability distribution $p = \frac{W}{\|W\|_1}$, ensuring the resulting values are non-negative and sum to one. By dynamically adjusting weights based on the current context, WSA emphasizes the most relevant models for any given situation, resulting in a more accurate and adaptable representation. The final representation $\mathcal{R}$ is computed using the weights for all pre-trained models. The `Fully-Connected Network` maps the final embedding $\mathcal{R}$ to actions, thereby enhancing the performance and learning efficiency of RL agents by integrating the strengths of multiple pre-trained models.

Our approach by design guarantees two additional features. WSA is `scalable` and `adaptable` to any number of pre-trained models. Its shared component can handle an arbitrary number of models, making it a flexible solution for dynamically evolving agents. On this matter, Section 4.4 analyzes two different scenarios where the number of pre-trained models is increased or decreased over time. Additionally, WSA adds `explainability` to the model. By examining the weights assigned to different pre-trained models, we can gain insights into which models are most influential in a given context. This characteristic is important for achieving a more interpretable decision-making process for RL agents and for gaining valuable insights into the learning process.

4 Experiments & Results

In our experimental evaluation, we investigate the benefits and challenges that raise when combining multiple pre-trained models in RL. Appendix B-C provide detailed overviews of the experimental setup used in this study, including details about implementation, feature extractors definition, and pre-trained models employed as inductive bias. Through this empirical evaluation, we aim to answer the following research questions:

- (`Q1`) Does the combination of pre-trained models improve the performance?
- (`Q2`) Is static prior pre-trained knowledge enough for generalization?
- (`Q3`) Does learning to combine multiple representations improve agents' robustness?
- (`Q4`) What happens when the number of pretrained models changes over time?

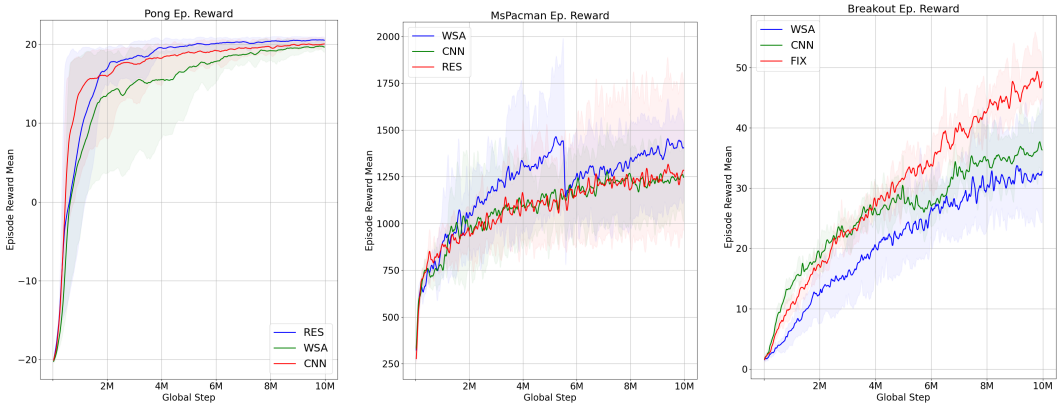
4.1 Comparison with End-to-End model (`Q1`)

The core idea is to assess if WSA is capable of combining out-of-the-box multiple pre-trained models and to evaluate if the enriched representation can be exploited for learning. It is important to note that the results are obtained without conducting any hyperparameter search. This analysis strictly focuses on the quality and insights of the computed representation. Figure 2 shows the average learning curves of agents using different combination modules during training. Table 1 reports the averaged results during evaluation. In this analysis, we compare WSA with the classical end-to-end model (E2E) used in Atari games and other combination modules (Appendix H). We also include

three additional baselines: (*i*) *choose-1* (C1), that uses Swin (Liu et al., 2021) - a single large pre-trained transformer model; (*ii*) *Ensamble* (ENS), where the representation is obtained by averaging the pre-trained models' embeddings; (*iii*) *Full Training* (FT), where the models' weights are learnt alongside policy ones. Looking at the evaluation results in Table 1, WSA matches the maximum reward (`21`) on `Pong` and achieves a higher score (`2530`) on `Ms.Pacman` than E2E. We address `Breakout` results in Section 4.2 to answer (`Q2`). All the additional baselines perform worse than our proposed architecture and the end-to-end model in the two more complex games, `Ms.Pacman` and `Breakout`; only ENS showcases competitive performance in Pong. A reason for the failure of C1 can be attributed to the out-of-distribution scenario of the Atari games compared to the training data of the Swin model, resulting in vague and incoherent embeddings. The poor performance of the ENS baseline further highlights the need for a smarter and more complex way to combine multiple embeddings. Simply averaging the embeddings' values fails to help learning competitive agents' policies. The full-training baseline shows how the pre-trained models effectively provide insightful representation that cannot otherwise be learned from scratch while optimizing agents' performance. As a sanity check, to ensure agents' performance does not depend on the particular RL algorithm, we also compare WSA effectiveness using `DQN`: training curves and evaluation scores are reported in Appendix D.1. Additionally, Appendix D.2 presents the performance of WSA on realistic robotic manipulation tasks (Tao et al., 2024), while Appendix D.3 extends the analysis to a broader set of Atari games, incorporating a targeted hyperparameter search to further optimize our solution.

Agent	**`Pong`**	**`Ms.Pacman`**	**`Breakout`**
C1	-21 ± 0.00	653.595 ± 47.14	8.38 ± 2.41
ENS	20.38 ± 0.46	642.87 ± 58.96	62.30 ± 7.50
FT	-20.29 ± 0.25	689.47 ± 28.04	8.89 ± 2.04
CNN	21 ± 0.00	1801.30 ± 20.95	65.98 ± 1.62
RES	20.85 ± 0.29	1369.27 ± 565.23	—
FIX	—	—	87.17 ± 6.87
WSA	21 ± 0.00	2530.20 ± 23.09	99.58 ± 6.66
WSA (M)	—	—	345.52 ± 6.47
E2E	20.51 ± 0.69	2145.80 ± 167.16	404.46 ± 13.49

Table 1: Performance during `evaluation` averaged across 5 different seeds. E2E model scores are computed using the `OpenAI/Baselines PPO` results collected in Huang et al. (2024).

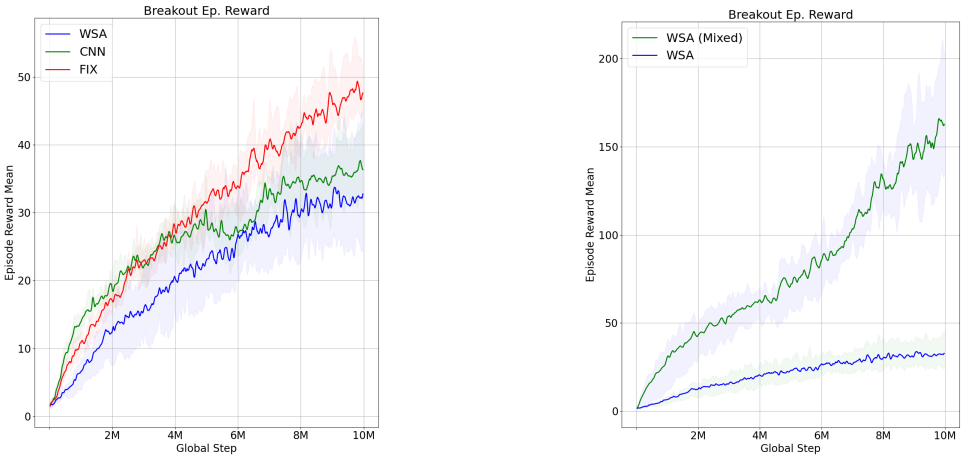


(a) `Pong` (b) `Ms.Pacman` (c) `Breakout`

Figure 2: Cumulative reward during `training` of different agents using WSA and other combination modules on three Atari games. Each subfigure shows the mean score, with shaded areas indicating the standard deviations across multiple agents.

4.2 Out of Distribution (Q2)

Unexpectedly on `Breakout`, WSA did not work straightaway. Both in Figure 2c and in Table 1, the performance of WSA is extremely low. Unlike `Pong` and `Ms.Pacman`, where the game screen remains relatively stable, `Breakout` dynamically evolves as more blocks are removed with score progression. This poses a **distributional shift** problem between training and test data. Our initial training data, collected from random agents, lacks late-game scenarios with few blocks remaining, which leads to limitations in the model's performance during inference. To tackle this problem and show the consequences of a limited training dataset, we gather new datasets from both random and expert agents to encompass early and late-game scenarios. We retrain all the pre-trained models and RL agents. Significant improvements are observed, as shown in Figure 3b and Table 1, using `(M)`. Notably, WSA demonstrates a substantial increase in the agent's final score (`99` $\rightarrow$ `345`), positioning itself slightly below the E2E model performance, and echoing the trends analyzed in Section 4.1 for the other two games.



(a) `Breakout` (b) Using Mixed Data

Figure 3: Performance comparison of WSA across different strategies on `Breakout`. Each subfigure displays the average score with the standard deviation shaded. They report the different experiments to improve the performance of WSA: (3a) shows the initial condition, (3b) pre-training models using random and expert data.

4.3 WSA Robustness (Q3)

Differently from end-to-end approaches, WSA leverages the embeddings provided by the pre-trained models, which are trained via supervised or unsupervised learning on a set of collected data and are kept frozen during the RL learning phase. For this reason, we expect the representations learned by these models to be more robust with respect to convolutional layers that are learned together with the policy (which tend to over-specialize to the training environment). To investigate this issue and measure the positive impact of WSA, we use `HackAtari` (Delfosse et al., 2024), which provides different variations of Atari gaming by changing game elements' color or altering the game settings. In our analysis, we consider two games to evaluate WSA in two different scenarios. The former is `Pong`, in its variation *Lazy Enemy* (LE) where the CPU behavior is altered allowing movement only after the player hits the ball. The latter is `Breakout`, where we use multiple variations that change the *color* either of the *player* (CP) or *blocks* (CB) (both can vary among five possible variation corresponding to the colors black, white, red, blue and green). In Table 2, we report the results of the evaluation on the two settings, averaged over 30 episodes. For `Breakout`, since there are no results in the original work, we train and evaluate both E2E and WSA agents. In Appendix E, we report the training curves of the agents, as well as a complete table showcasing the performance

on all 25 possible variations of Breakout. WSA demonstrates greater robustness and resilience to environmental changes compared to the E2E baseline across both scenarios. In `Pong`, WSA consistently maintains a **positive (winning)** score, while in `Breakout`, it outperforms E2E by more than **3×** in all configurations. Despite a significant drop in performance, likely related to research question (`Q2`), WSA still maintains a substantial advantage.

Game	Random	E2E	WSA
Training **Testing**	- variation	original variation	original variation
Pong (LE)	-20.1 ± 0.4	-12.6 ± 2.4	13.56 ± 6.43
Breakout (CP 0) (CB $\star$)	1.19 ± 1.23	12.14 ± 7.50	45.64 ± 18.88
Breakout (CP 1) (CB $\star$)	1.18 ± 1.17	13.54 ± 9.37	45.35 ± 17.22
Breakout (CP 2) (CB $\star$)	1.31 ± 1.24	13.07 ± 9.06	44.04 ± 16.61
Breakout (CP 3) (CB $\star$)	1.00 ± 1.07	13.10 ± 8.82	43.49 ± 16.30
Breakout (CP 4) (CB $\star$)	1.16 ± 1.16	12.41 ± 7.96	44.70 ± 17.84
Breakout (CP $\star$) (CB 0)	1.18 ± 1.15	12.81 ± 7.78	44.93 ± 16.95
Breakout (CP $\star$) (CB 1)	1.15 ± 1.16	13.31 ± 9.54	45.20 ± 16.92
Breakout (CP $\star$) (CB 2)	1.24 ± 1.23	12.49 ± 8.15	44.48 ± 17.15
Breakout (CP $\star$) (CB 3)	1.07 ± 1.20	12.83 ± 8.19	44.75 ± 19.06
Breakout (CP $\star$) (CB 4)	1.20 ± 1.14	12.82 ± 9.07	43.86 ± 16.78

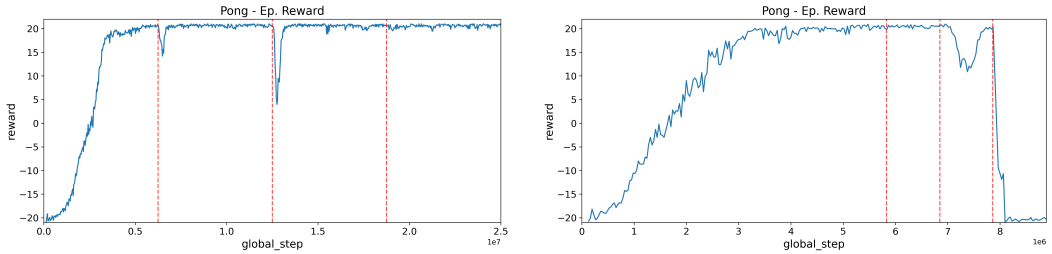
Table 2: Results averaged over multiple episodes for different configurations of the game as in Delfosse et al. (2024). The table presents performance for random, PPO, and WSA agents across variations of Pong and Breakout. Scores marked with $\star$ indicate averages over all possible combinations for that particular setting.

4.4 Scaling Number of Models (Q4)

Our approach is designed to ensure two key properties: WSA is both `scalable` and `adaptable` to any number of pre-trained models. The `Shared Weight Network` is capable of managing an arbitrary number of models, making it a versatile solution for dynamically evolving agents. This section examines two distinct scenarios in which the number of pre-trained models changes over time, either increasing or decreasing. In the *adding* experiment, Figures 4a, a new model becomes available at regular intervals, marked in red, and it is incorporated into the computation of the embedding while the agent continues training. This setup simulates a scenario where an agent must continuously integrate new knowledge without disrupting its learning process. On the other hand, in the *removal* experiment, Figure 4b, the agent is initially trained with the *complete set* of available models. Subsequently, we systematically remove models from the pool at regular intervals, shown in red. This experiment evaluates the agent's ability to retain and generalize knowledge as resources diminish, assessing its adaptability to a progressively reduced set of pre-trained models while maintaining performance. For more details about these scenarios, please refer to Appendix F.

WSA Explainability. Lastly, Figure 5 illustrates the `explainability` feature provided by WSA. By examining the current frames and the corresponding weights allocated by the shared network to each pre-trained model, one can appreciate the agents' decision-making process. Moreover, this information helps in debugging and understanding agents behavior in dynamic environments, as the one presented in Appendix F. Figure 5 shows how WSA's weights change throughout an episode on `Breakout`. WSA assigns the highest weight to the Video Object Segmentation (VOS) (Goel et al., 2018) model and combines it with contributions from other two models: State Representation (SR) (Anand et al., 2019) and Object Keypoints (OK) (Kulkarni et al., 2019). In the initial stages of the games where the majority of the blocks are still present, WSA combines multiple embeddings.

Once the agent gets access to the upper portion of the screen, it starts eliminating most of the blocks and WSA gradually increases the importance of the VOS model. This behavior may be attributed to VOS embeddings that are pre-trained to generate segmentation masks across multiple frames. During this phase of the game, the agent must wait for the ball to hit the blocks and fall, making the ability to track objects over time particularly important.



(a) Adding Scenario (b) Removal Scenario

Figure 4: Figures illustrate the adding and removing experiments, red lines mark when a model is added or removed.

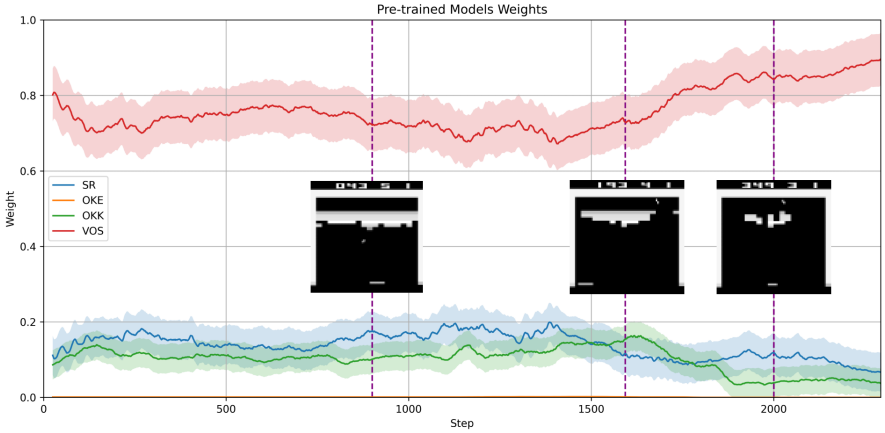


Figure 5: Weights assigned by WSA to different pre-trained models during rollouts in `Breakout`. In the early stages, WSA distributes attention across multiple models, whereas in the later stages, it converges towards relying predominantly on a single model (VOS).

5 Conclusion

In this work, we explored how prior knowledge encoded in multiple pre-trained models can be integrated as inductive biases to guide representation learning in Reinforcement Learning agents. We addressed the challenge of combining diverse latent representations into a unified feature space and proposed the Weight Sharing Attention (WSA) module, which uses attention mechanisms and parameter sharing to dynamically balance the contributions of each model in a scalable and model-agnostic way. Our evaluation across a range of Atari games demonstrates that WSA match or outperform traditional end-to-end architectures, effectively leveraging prior knowledge from multiple sources. These results highlight the potential of combining multiple inductive priors to improve efficiency and policy robustness, particularly in dynamic or changing environments. Looking ahead, we plan to extend our analysis to more diverse and complex scenarios. We also aim to scale our approach by incorporating richer forms of prior knowledge, e.g., Large Language Models (LLMs) or Vision-Language Models (VLMs).

Acknowledgments

We would like to thank the reviewers and Levi H. S. Lelis for their valuable comments. Research partly funded by PNRR - M4C2 - Investimento 1.3, Partenariato Esteso PE00000013 - "FAIR - Future Artificial Intelligence Research" - Spoke 1 "Human-centered AI", funded by the European Commission under the NextGeneration EU programme.

References

Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R. Devon Hjelm. Unsupervised state representation learning in atari. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 8766–8779, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/6fb52e71b837628ac16539c1ff911667-Abstract.html.

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, et al. Agent57: Outperforming the atari human benchmark. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 507–517. PMLR, 2020. URL http://proceedings.mlr.press/v119/badia20a.html.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.*, 47:253–279, 2013. DOI: 10.1613/jair.3912. URL https://doi.org/10.1613/jair.3912.

Lukas Biewald. Experiment tracking with weights and biases, 2020. URL https://www.wandb.com/. Software available from wandb.com.

Sam Blakeman and Denis Mareschal. Selective particle attention: Rapidly and flexibly selecting features for deep reinforcement learning. *Neural Networks*, 150:408–421, 2022. DOI: 10.1016/j.neunet.2022.03.015. URL https://doi.org/10.1016/j.neunet.2022.03.015.

Lennart Bramlage and Aurelio Cortese. Generalized attention-weighted reinforcement learning. *Neural Networks*, 145:10–21, 2022. DOI: 10.1016/j.neunet.2021.09.023. URL https://doi.org/10.1016/j.neunet.2021.09.023.

Quentin Delfosse, Jannis Blüml, Bjarne Gregori, and Kristian Kersting. Hackatari: Atari learning environments for robust and continual reinforcement learning. *arXiv preprint arXiv:2406.03997*, 2024.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. URL https://arxiv.org/abs/2010.11929.

Rachit Dubey, Pulkit Agrawal, Deepak Pathak, Tom Griffiths, and Alexei A. Efros. Investigating human priors for playing video games. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1348–1356. PMLR, 2018. URL http://proceedings.mlr.press/v80/dubey18a.html.

Abraham J. Fetterman, Ellie Kitanidis, Joshua Albrecht, Zachary Polizzi, Bryden Fogelman, Maksis Knutins, et al. Tune as you scale: Hyperparameter optimization for compute efficient training. *CoRR*, abs/2306.08055, 2023. DOI: 10.48550/ARXIV.2306.08055. URL https://doi.org/10.48550/arXiv.2306.08055.

Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. Deep reservoir computing: A critical experimental analysis. *Neurocomputing*, 268:87–99, 2017. DOI: 10.1016/J.NEUCOM.2016.12.089. URL https://doi.org/10.1016/j.neucom.2016.12.089.

Vikash Goel, Jameson Weng, and Pascal Poupart. Unsupervised video object segmentation for deep reinforcement learning. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 5688–5699, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/96f2b50b5d3613adf9c27049b2a888c7-Abstract.html.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. URL http://jmlr.org/papers/v23/21-1342.html.

Shengyi Huang, Quentin Gallouédec, Florian Felten, Antonin Raffin, Rousslan Fernand Julien Dossa, et al. Open RL benchmark: Comprehensive tracked experiments for reinforcement learning. *CoRR*, abs/2402.03046, 2024. DOI: 10.48550/ARXIV.2402.03046. URL https://doi.org/10.48550/arXiv.2402.03046.

Tomas Jakab, Ankush Gupta, Hakan Bilen, and Andrea Vedaldi. Unsupervised learning of object landmarks through conditional image generation. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 4020–4031, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/1f36c15d6a3d18d52e8d493bc8187cb9-Abstract.html.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, et al. Openvla: An open-source vision-language-action model. In *Conference on Robot Learning, 6-9 November 2024, Munich, Germany*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 2024. URL https://proceedings.mlr.press/v270/kim25c.html.

Tejas D. Kulkarni, Ankush Gupta, Catalin Ionescu, Sebastian Borgeaud, Malcolm Reynolds, et al. Unsupervised learning of object keypoints for perception and control. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 10723–10733, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/dae3312c4c6c7000a37ecfb7b0aeb0e4-Abstract.html.

Aviral Kumar, Rishabh Agarwal, Xinyang Geng, George Tucker, and Sergey Levine. Offline q-learning on diverse multi-task data both scales and generalizes. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/pdf?id=4-k7kUavAj.

Charline Le Lan, Stephen Tu, Mark Rowland, Anna Harutyunyan, Rishabh Agarwal, et al. Bootstrapped representations in reinforcement learning. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 18686–18713. PMLR, 2023. URL https://proceedings.mlr.press/v202/le-lan23a.html.

Hao Liu, Lisa Lee, Kimin Lee, and Pieter Abbeel. Instruction-following agents with jointly pre-trained vision-language models. *CoRR*, abs/2210.13431, 2022. DOI: 10.48550/ARXIV.2210.13431. URL https://doi.org/10.48550/arXiv.2210.13431.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, et al. Swin transformer: Hierarchical vision transformer using shifted windows. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pp. 9992–10002. IEEE, 2021. DOI: 10.1109/ICCV48922.2021.00986. URL https://doi.org/10.1109/ICCV48922.2021.00986.

Arjun Majumdar, Karmesh Yadav, Sergio Arnaud, Yecheng Jason Ma, Claire Chen, et al. Where are we in the search for an artificial visual cortex for embodied intelligence? In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/022ca1bed6b574b962c48a2856eb207b-Abstract-Conference.html.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, et al. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, et al. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015. DOI: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

Javier Montalvo, Álvaro García-Martín, and Jesús Bescós. Exploiting semantic segmentation to boost reinforcement learning in video game environments. *Multim. Tools Appl.*, 82(7):10961–10979, 2023. DOI: 10.1007/S11042-022-13695-1. URL https://doi.org/10.1007/s11042-022-13695-1.

Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3m: A universal visual representation for robot manipulation, 2022. URL https://arxiv.org/abs/2203.12601.

Johan Samir Obando-Ceron, Ghada Sokar, Timon Willi, Clare Lyle, Jesse Farebrother, Jakob Nicolaus Foerster, Gintare Karolina Dziugaite, Doina Precup, and Pablo Samuel Castro. Mixtures of experts unlock parameter scaling for deep RL. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=X9VMhfFxwn.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

Antonin Raffin. Rl baselines3 zoo. https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Moritz Schneider, Robert Krug, Narunas Vaskevicius, Luigi Palmieri, and Joschka Boedecker. The surprising ineffectiveness of pre-trained visual representations for model-based reinforcement learning. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/39b1126cdba0a37986fa14f568471ae8-Abstract-Conference.html.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Max Schwarzer, Nitarshan Rajkumar, Michael Noukhovitch, Ankesh Anand, Laurent Charlin, et al. Pretraining representations for data-efficient reinforcement learning. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 12686–

12699, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/69eba34671b3ef1ef38ee85caae6b2a1-Abstract.html.

Rutav M. Shah and Vikash Kumar. RRL: resnet as representation for reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9465–9476. PMLR, 2021. URL http://proceedings.mlr.press/v139/shah21a.html.

Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9870–9879. PMLR, 2021. URL http://proceedings.mlr.press/v139/stooke21a.html.

Joseph Suarez. Pufferlib: Making reinforcement learning libraries and environments play nice, 2024. URL https://arxiv.org/abs/2406.12905.

Joseph Suarez. Pufferlib 2.0: Reinforcement learning at 1m steps/s. In *Reinforcement Learning Conference*, 2025. URL https://openreview.net/forum?id=qRyteMTgn0.

Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse kai Chan, Yuan Gao, Xuanlin Li, Tongzhou Mu, Nan Xiao, Arnav Gurha, Zhiao Huang, Roberto Calandra, Rui Chen, Shan Luo, and Hao Su. Maniskill3: A versatile, scalable, and flexible machine learning framework for medical imaging. *arXiv preprint arXiv:2410.00425*, 2024.

SIMA Team, Maria Abi Raad, Arun Ahuja, Catarina Barros, Frederic Besse, et al. Scaling instructable agents across many simulated worlds. *CoRR*, abs/2404.10179, 2024. DOI: 10.48550/ARXIV.2404.10179. URL https://doi.org/10.48550/arXiv.2404.10179.

Josh Tenenbaum. Building machines that learn and think like people. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pp. 5. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2018. URL http://dl.acm.org/citation.cfm?id=3237389.

Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, et al. Gymnasium, March 2023. URL https://zenodo.org/record/8127025.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, et al. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su. SAPIEN: A simulated part-based interactive environment. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Tete Xiao, Ilija Radosavovic, Trevor Darrell, and Jitendra Malik. Masked visual pre-training for motor control. *CoRR*, abs/2203.06173, 2022. DOI: 10.48550/arXiv.2203.06173. URL https://doi.org/10.48550/arXiv.2203.06173.

Zhecheng Yuan, Zhengrong Xue, Bo Yuan, Xueqian Wang, Yi Wu, et al. Pre-trained image encoder for generalizable visual reinforcement learning. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/548a482d4496ce109cddfbeae5defa7d-Abstract-Conference.html.

A Related Works: Comparison

Several state-of-the-art methodologies and works are employing Mixture of Experts to their models, suggesting that combining multiple sources of information improve the performance (Obando-Ceron et al., 2024). However, *how* to combine multiple latent features in a single representation is still an open problem. There is almost non-existing literature about works that try to apply more than one model at the same time to enhance the representation. Team et al. (2024) provide the agent with multiple instances of pre-trained models that solve auxiliary tasks and combine their information. They showcase that combining pre-trained models with trained-from-scratch components leads to better-performing agents. Unfortunately, due to the large scale of their work, there is no particular attention towards how models' representations are combined. Recent advances in large-scale language models have increased the research community's interest in combining visual information with textual data. In these scenarios, agents must handle different data modalities at the same time. This can be obtained by using either a single multi-modal pre-trained model (Liu et al., 2022) or multiple models for each data source. A recent work that follows the latter idea is OpenVLA (Kim et al., 2024). In this model visual representations are embedded using multiple vision encoders. The embeddings are then combined and projected, using a small MLP, into a common latent space with the language model. In our work, we aim to focus specifically on how to combine pre-trained models. In this context, our work could be used as a component within the aforementioned works to improve the effectiveness of combining different pre-trained models that possibly operate across different data types. We propose WSA as an effective and efficient solution to merge different pre-train models' embeddings while guaranteeing the ability to scale over time.

B Experimental Setup

Code Implementation. To simulate and interact with the Atari environments (Bellemare et al., 2013) we leverage the API provided by Gymnasium (Towers et al., 2023). The training performance was tracked and recorded using `Weights and Biases` (Biewald, 2020). We build our work[1] using `Stable-Baselines3` (SB3) (Raffin et al., 2021) and CleanRL (Huang et al., 2022) as backbones to create agents' architecture and use reliable implementations of RL algorithms. In particular, we exploit the unconstrained definition of the agent model to decompose the agents' network in `Feature Extractor` and `Fully-Connected Network`, as it also highlighted in Figure 1. We use the `Feature Extractor` to define the different combination modules that provide as output the combined latent representation. The `Fully-Connected Network` receives as input the pre-trained models' unified encoding and maps it to actions (or values). We keep the latter component as simple as possible since most of the information should be provided by the enriched representations. Throughout all the initial experiments and the first comparison analysis, we use the default hyperparameters for different RL algorithms, i.e. PPO and DQN (Raffin, 2020). We do not conduct any hyperparameter search for our agents, unless otherwise stated. Instead, we only modify the model architecture. Even though this setup could be sub-optimal for our solution, we believe is an important test to evaluate the actual influence of combining pre-trained models' representations. Hyperparamters used for different RL algorithms in the experiments: `PPO` (Table 3) and `DQN` (Table 4). The values are the default ones and are also available on rl-zoo (Raffin, 2020). For each game, multiple agents instances are trained using different seeds. For a comprehensive view of agents' performance, the score is averaged over different versions, agents are tested across `5 seeds` (different from the training ones) for `50 episodes`. The set of seeds used during evaluation is the following: 47695, 32558, 94088, 71782 and 66638.

Feature Extractors. Besides our proposed methodology, we tested and compared the performance of several combination modules. All solutions act as interchangeable modules inside the `Feature Extractor`, i.e. only need to replace the yellow component in Figure 1. Feature Extractors' output

[1] Code is available at `https://github.com/EliaPiccoli/Collas-WSA`

is a linear representation that is provided as input to the `Fully-Connected Network`. Figures 14-15 showcase graphically all the extractors; here, we report a brief rundown across all solutions:

- `Linear` (LIN): pre-trained models' representations are linearized and concatenated.
- `Fixed Linear` (FIX): embeddings are linearized to a predefined size, possibly using adapters to scale the information.
- `Convolutional` (CNN): pre-trained models' outputs are concatenated along the channel dimension. They are processed by a predefined number of convolutional layers, and the resulting information is flattened to linear.
- `Mixed` (MIX): this is a combination of the previous methods. Data coming from different spaces are dealt separately and then combined.
- `Reservoir` (RES): inspired by Gallicchio et al. (2017), this approach leverages reservoir layers to combine models' representations.
- `Dot Product Attention` (DPA): via `scaled dot product attention` (Vaswani et al., 2017) we compute the final weighted representation using as query vector the representation obtained from the State Encoder.

Hyperparameter	Value
N. Envs.	8
N. Stacks	4
N. Steps	128
N. Epochs	4
Batch Size	256
N. Timesteps	10.000.000
Learning Rate	2.5e-4
Clip Range	0.1
VF. Coef.	0.5
Ent. Coef.	0.01
Normalize	True

Table 3: PPO hyperparameters.

Hyperparameter	Value
N. Envs.	8
N. Stacks	4
Buffer Size	100.000
Target Update Interval	1.000
Batch Size	32
N. Timesteps	10.000.000
Learning Rate	1e-4
Learning Starts	100.000
Train Freq.	4
Gradient Steps	1
Exploration Fraction	0.1
Exploration Final Eps	0.01
Optimize Memory Usage	False

Table 4: DQN hyperparameters.

Initial Experiments. We chose three different Atari games as benchmark, with different levels of difficulty: `Pong`, `Ms.Pacman`, and `Breakout`. After training each pre-trained model, we run a first round of experiments for the three games to select the three best performing extractors and their respective embedding size, which will be used for our empirical analysis along with our proposed method. Agents are trained using PPO (Schulman et al., 2017) for `10M` steps using the default parameters. In particular, the `Fully-Connected Network` and the `Shared Weight Network` are defined as a single layer MLP. Throughout the learning process, agents are repeatedly evaluated for 100 episodes. Appendix H reports all the possible configurations, a detailed analysis on the computational cost and their performance in this initial experimental phase. We report the best performing modules for each game, the chosen embedding size is detailed between parentheses: **`Pong`**: WSA (1024), RES (1024), CNN (2); **`Ms.Pacman`**: WSA (256), RES (1024), CNN (2); **`Breakout`**: WSA (256), FIX (512), CNN (3).

C Pre-Trained Models

C.1 Data and Training

To pretrain the different models, we first create a specific dataset for each game. For each environment, we collect `1M` frames via random agents interacting with the environment. The dataset contains grayscale game frames with size of 84x84 pixels. During the training process, data is randomly sampled to avoid any correlation between elements due to the collection phase. For the experiments concerning our approach, we use three different models, which provide us with four different pre-trained networks: `Video Object Segmentation` (Goel et al., 2018), `State Representation` (Anand et al., 2019), and `Object Keypoints` (Kulkarni et al., 2019). We implement and train the models for all the games using the default architecture and hyperparameters, the values and additional details, e.g. the embedding representation shape, are reported in Appendix C.2. Additionally, we train a deep autoencoder - inspired by Nature CNN (Mnih et al., 2015) - to encode the current state and leverage its representation to act as the context $\mathcal{C}$ in our approach. As previously mentioned, the main objective of this work is `combining` different representations, thus, we exploit these models, which can be easily trained and adapted compared to monolithic foundational models but still provide insightful and useful information. Nevertheless, the chosen pre-trained models can be arbitrarily changed to more complex or larger ones without affecting the structure of our work. While training the agent, pre-trained models' weights are frozen and no longer updated.

C.2 Representation and Hyperparameters

In this section, we provide additional details on the models leveraged as prior knowledge by agents. Tables 5a, 5b, 5c report the hyperparameters used to train all models; the architecture presented in the original works is kept as the backbone of the models. We re-implement all models and training scripts in PyTorch, which will be released together with our codebase and checkpoints of the models to replicate all the experiments. `State Representation`'s RAM information is not available for all games; thus, in our experiments, `Beam Rider` and `Enduro` have one less pre-trained model due to the unavailabiliy of the information in Anand et al. (2019).
Next we provide a brief description of each model and the shape of their latent representation.

- `State Representation` (Anand et al., 2019). Given the current state, it computes a linear representation exploiting the spatial-temporal nature of visual observations.
  Embedding shape: $[512]$.
- `Object Keypoints` (Kulkarni et al., 2019). The model is composed by two heads that are used during inference: a convolutional network to compute spatial feature maps and a KeyNet (Jakab et al., 2018) to predict the keypoint coordinates.
  Embedding shape: CNN $[128 \times 21 \times 21]$, KeyNet $[4 \times 21 \times 21]$.
- `Video Object Segmentation` (Goel et al., 2018). This model takes as input the last two frames of the game and computes K feature maps that highlights the moving objects.
  Embedding shape: $[20 \times 84 \times 84]$.
- `Deep Autoencoder`. It is used to encode the current state and is used in WSA to compute the context. Its architecture is inspired by NatureCNN (Mnih et al., 2015).
  Embedding shape: $[64 \times 16 \times 16]$.

Hyperparameters	Value
Episodes	1000
Max Iter	1e6
Batch Size	64
Image Channels	1
K (n. keypoints)	4
Learning Rate	1e-3
LR Decay	0.95
Decay Length	1e5

(a)

Hyperparameters	Value
Num. Frames	2
Steps	500,000
Batch Size	64
Learning Rate	1e-4
Max Grad. Norm	5.0
Decay Length	1e5
Optimizer	Adam
K (n. masks)	20

(b)

Hyperparameters	Value
Image Width	160
Image Height	210
Grayscaling	Yes
Action Repetitions	4
Max-pool (last N)	2
Frame Stacking	4
Batch Size	64
LR (Training)	5e-4
LR (Probing)	3e-4
Entropy Threshold	0.6
Encoder steps	80,000
Probe train steps	30,000
Probe test steps	10,000
Epochs	100
Feature Size	512
Pretraining steps	100,000
Num Parallel Envs.	8

(c)

Table 5: Hyperparameter used for three unsupervised pre-trained models, (5a) Object Keypoints (Kulkarni et al., 2019), (5b) Video Object Segmentation (Goel et al., 2018), (5c) State Representation (Anand et al., 2019).

D Additional Experiments

D.1 DQN Comparison

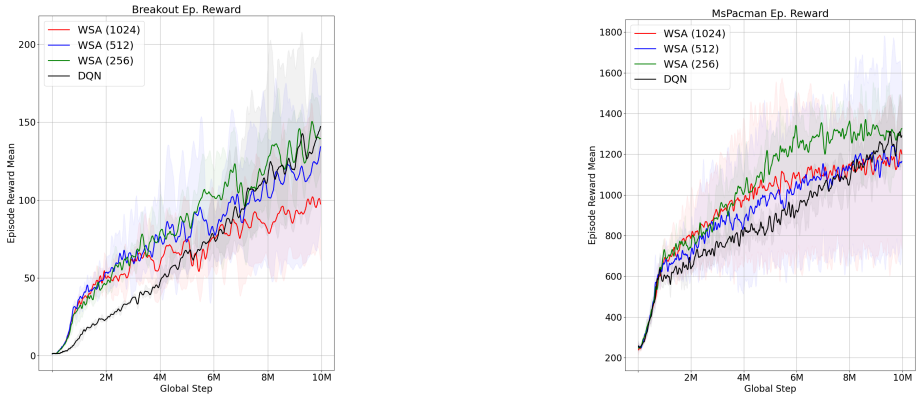
Using the same methodology and structure of experiments of the main work, we study the performance of WSA while using a different RL algorithm, i.e. `DQN`. This analysis proves that WSA works well regardless of the algorithm used. In particular, in `Pong` WSA achieves a perfect score, it *outperforms* the end-to-end architecture in `Breakout` and `MsPacman` and in `SpaceInvaders` WSA achieves a *comparable mean performance* to E2E but with significantly lower variability. Figure 6 depicts the average training trends across different instances with various latent-representation sizes, while Table 6 shows the evaluation performance of the best configuration.

	Breakout	MsPacman	Pong	Space Invaders
`WSA`	213.14 ± 39.37	2047.27 ± 231.18	21 ± 0.00	649.00 ± 136.85
`E2E`	166.65 ± 20.19	1701.00 ± 490.41	20.70 ± 0.46	680.00 ± 231.26

Table 6: Performance during `evaluation` averaged across 5 different seeds, WSA embedding size is 256.

D.2 ManiSkill additional experiments

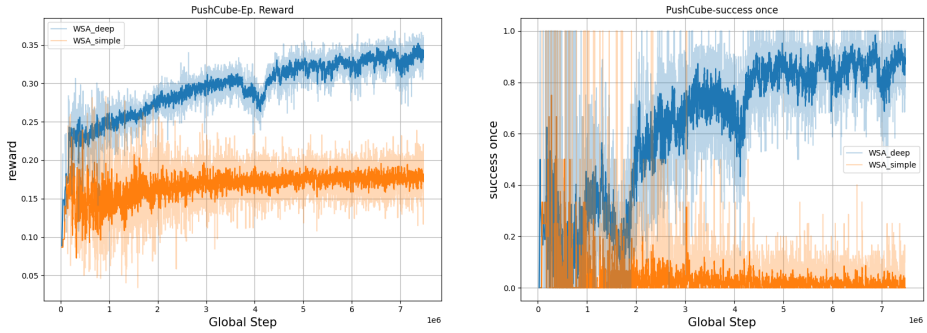
We conducted an additional experiment using `ManiSkill` (Tao et al., 2024), a powerful framework for robot simulation, especially manipulation skills, and training powered by SAPIEN (Xiang et al., 2020). In this experiment we consider the `Push-Cube-V1` environment with an episode length of 50, where the robot arm has to move a block to a target location. We use the RGB representation to define the observation space and use PPO as an RL algorithm. To process the visual



(a) `Breakout` (b) `MsPacman`

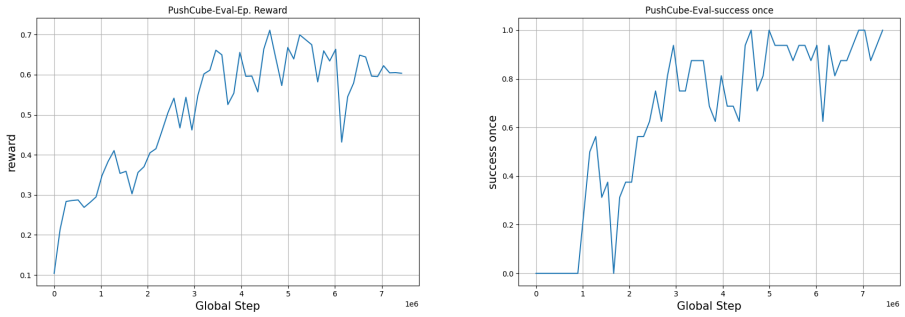
Figure 6: Cumulative reward during `training` of different agents, comparing WSA and end-to-end model using DQN. Each subfigure shows the mean score, shaded areas indicate the standard deviations across multiple agents.

representation, we employed different open-source and off-the-shelf pre-trained models. In particular we use: (*i*) `SwinTransformer` (Liu et al., 2021); (*ii*) a fine-tuned `resnet18` from R3M (Nair et al., 2022) tailored for robotic tasks; (*iii*) a `CLIP` model (Radford et al., 2021) with a ViT backbone (Dosovitskiy et al., 2021) for the state representation. Among these pre-trained models, only `resnet18` was already fine-tuned for robotics tasks, while the others remained general-purpose pre-trained vision encoders. Unlike the `Atari` environments, this setting involves continuous control and presents a more challenging task. In fact, WSA architecture using single-layer MLPs for the `Shared Weight Network` collapses and fails to learn a meaningful policy for the task. To better handle the continuous action space and complexity of the problem, we deepened the architecture of the `Shared Weight Network` to three hidden layers, keeping its size relatively small. Despite the increased complexity of this setting, `WSA` demonstrates strong performance. Figure 7 reports the training curves for the reward and success of the task, comparing the basic and deep WSA versions. While we fixed the number of training steps to 7.5M, the performance increases steadily as training progresses, suggesting that with longer training time, WSA can converge to optimal performance as shown in Figure 8, where deep WSA during evaluation on 16 episodes reaches a mean reward near 0.6 per step and success rate close to 90%. These experiments demonstrate that our architecture generalizes effectively to diverse environments — including those more realistic and complex than Atari games — achieving strong performance with minimal tuning.



(a) WSA agent mean reward per episode (b) WSA agent success once

Figure 7: Training performance of WSA agent on `Push-Cube-v1` environment, reporting episode reward and success during training. Comparing performance of WSA using a single layer (*simple*) or three layer (*deep*) `Shared Weight Network`.



(a) deep WSA mean reward on evaluation (b) deep WSA success once on evaluation

Figure 8: Evaluation of deep WSA agent on `Push-Cube-V1`. Evaluation is performed over 16 episodes with duration of 50 steps.

D.3 Extended Experiments: Optimized WSA

In Section 4.1, we analyzed whether WSA could serve as a valid alternative to the convolutional layers that characterize end-to-end agents on Atari. We showed that the information obtained by the combination of multiple pre-trained models enhances the performance of agents. In this section, we aim to evaluate the potential of WSA on a broader set of Atari games by also optimizing the hyperparameters of the RL algorithm. To perform model optimization we leverage the CleanRL implementation of our method to incorporate WSA agents inside PufferLib (Suarez, 2024; 2025), which provides a faster vectorization reducing significantly the training time, and use CARBS (Fetterman et al., 2023) as hyperparameter optimization algorithm. For each game, we run a set of sweeps consisting of 50/100 runs (2-3 days of compute time) to identify the optimal configurations. The search space is reported in Appendix G. We train the best-performing configuration and, in Table 7, we report the evaluation performance across multiple seeds. Between parenthesis we include Human Normalized Score (HNS) and Capped-HNS (CNHS) (Badia et al., 2020) of the best agent.[2] For the E2E architecture we report the score from Huang et al. (2024).

Game	**E2E**	**WSA**
Asteroids	$1511.16 \pm 117.01_{(0.0195/0.0195)}$	$2130.95 \pm 139.96_{(0.0333/0.0333)}$
Beam Rider	$2842.55 \pm 423.97_{(0.1753/0.1753)}$	$1443.85 \pm 233.17_{(0.0793/0.0793)}$
Breakout	$404.46 \pm 13.49_{(14.4531/1.0)}$	$395.93 \pm 27.59_{(14.6465/1.0)}$
Enduro	$1061.93 \pm 41.05_{(1.2818/1.0)}$	$1113.23 \pm 69.29_{(1.3742/1.0)}$
MsPacman	$2145.80 \pm 167.16_{(0.3019/0.3019)}$	$2502.55 \pm 141.23_{(0.3517/0.3517)}$
Pong	$20.51 \pm 0.69_{(1.1810/1.0)}$	$20.62 \pm 0.37_{(1.1810/1.0)}$
Qbert	$15298.00 \pm 1011.65_{(1.2148/1.0)}$	$9194.75 \pm 503.5_{(0.7173/0.7173)}$
Seaquest	$1517.73 \pm 399.81_{(0.0440/0.0440)}$	$2795.31 \pm 288.35_{(0.0718/0.0718)}$
Space Invaders	$1038.64 \pm 72.93_{(0.6336/0.6336)}$	$833.94 \pm 65.06_{(0.4939/0.4939)}$
Mean HNS/CHNS	2.1450/0.5749	2.1054/0.5275

Table 7: Performance comparison of PPO and WSA on various Atari games, between parenthesis are reported HNS and CHNS, respectively. E2E model scores are computed using the `OpenAI/Baselines PPO` results collected in Open RL Benchmark (Huang et al., 2024).

Out of the nine games considered in this evaluation, WSA outperforms E2E in five of them (`Asteroids`, `Enduro`, `MsPacman`, `Pong`, `Seaquest`) and shows comparable performance in `Breakout`, as indicated by overlapping confidence intervals. On the other hand, in three games (`Beam Rider`, `Qbert`, `Space Invaders`) WSA performs worse then E2E. The small difference in the averaged human normalized scores, less than 0.05 in the capped case, suggests that

[2]Data used for computation is taken from CleanRL-HNS.

WSA performs in the same range as the baseline, while providing benefits such as modularity, interpretability (Section 4.4), and generalization advantages (Section 4.3, Appendix E). Given the characteristics of the games, we hypothesize that the performance is influenced by a similar problem to the one analyzed in section 4.2. For example, in later stages of `Space Invaders`, aliens are removed from the screen and the structures are damaged, picturing a different scenario compared to the initial conditions, leading to poor performance of the pre-trained models.

D.4 Breakout Additional Experiments

A first attempt to overcome the problem was to increase the number of parameters of the model. We increased the size of the `Fully-Connected Network` to three linear layers, using ReLU as activation function. Figure 9 and Table 8 - referenced as `(P)` - report the results for this experiment. With respect to the default scenario, there is an improvement in performance, but WSA is still far from the E2E model (`156 vs 404`).

The additional analysis, reported in Section 4.2, was also extended to the other two combination modules studied for the game Breakout (Appendix B and Figure 2c). Figures 10, 11 showcase the training trend while Table 8 evaluation scores (we recall `(P)` indicates that the `(Fully-Connected Network)` was increased to more layers, while `(M)` show the result while re-training the models using both random and expert data). Differently from WSA, these modules do not scale as good their final performance, they are still far from PPO final score. Nevertheless, `CNN` seems to gain advantage from an increase amount of hyperparameter, achieving the highest score in the `Policy` (P) version, while `FIX` shows a similar trend of improvement that was observed with WSA.

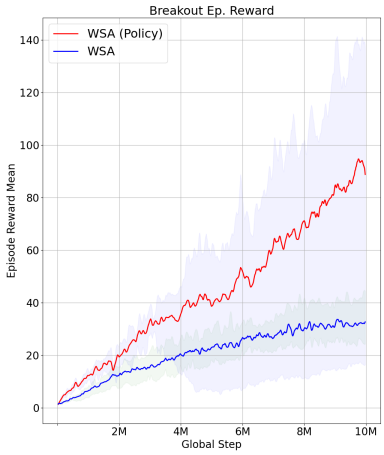
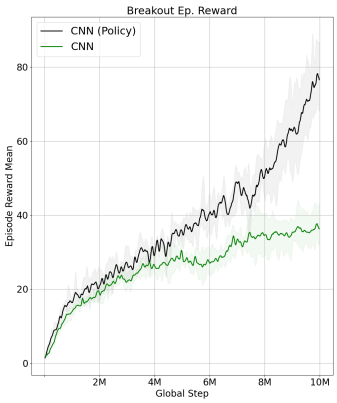
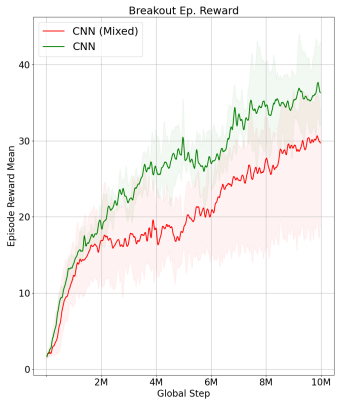


Figure 9: Increasing MLP size

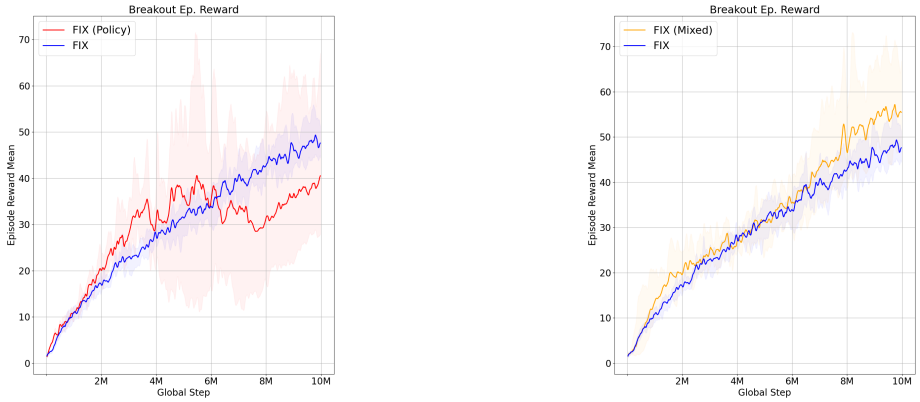


(a) Increasing MLP size



(b) Using Mixed Data

Figure 10: Performance comparison of `CNN` with PPO across various strategies in `Breakout`. Each subfigure displays the average score with the standard deviation shaded.



(a) Increasing MLP size (b) Using Mixed Data

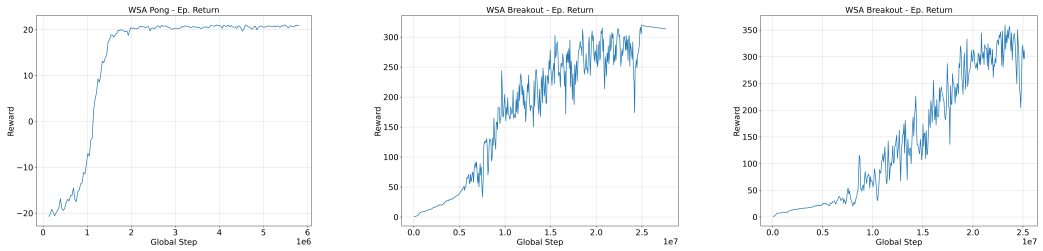
Figure 11: Performance comparison of `FIX` with PPO across various strategies in `Breakout`. Each subfigure displays the average score with the standard deviation shaded.

Environment	**Agent**	**Reward**
`Breakout`	CNN	65.98 ± 1.62
	CNN `(P)`	118.71 ± 4.30
	CNN `(M)`	62.21 ± 1.98
	FIX	87.17 ± 6.87
	FIX `(P)`	106.46 ± 10.84
	FIX `(M)`	199.08 ± 7.46
	WSA `(P)`	156.17 ± 3.59
	E2E	404.46 ± 13.49

Table 8: Performance during `evaluation`. This completes the results reported in Table 1.

E HackAtari Full Evaluation

In this section, we report the training curves of the three models we trained to be evaluated in the HackAtari scenarios. Figures 12a-12c outline the training curves for the WSA agents on Pong, reaching over 20 as the final score, and Breakout getting close to 300 points. Since in Delfosse et al. (2024) no information on the performance of PPO on Breakout variations is provided, we also train an end-to-end (E2E) PPO agent matching the training steps of WSA (Figure 12b). Table 9 details the performance of PPO and WSA across all 25 possible variations of Breakout. We recall that CP and CB stand for *color_player* and *color_blocks*, respectively. Both can range between five different values $X \in [0, 4]$ matching the colors black, white, red, blue, and green.



(a) WSA Pong (b) E2E Breakout (c) WSA Breakout

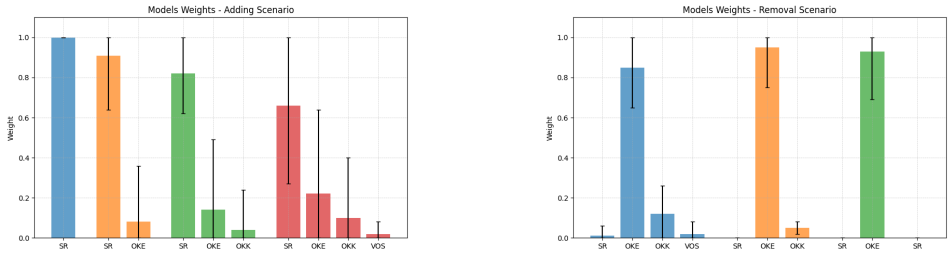
Figure 12: Training curves for the agents used in the HackAtari evaluations on Pong and Breakout.

Game	Random	E2E	WSA
Training **Testing**	- variation	original variation	original variation
Breakout (CP 0) (CB 0)	0.98 ± 1.05	12.58 ± 7.91	44.92 ± 18.27
Breakout (CP 0) (CB 1)	1.40 ± 1.22	12.28 ± 5.67	46.62 ± 25.13
Breakout (CP 0) (CB 2)	1.36 ± 1.51	11.14 ± 7.11	43.48 ± 14.19
Breakout (CP 0) (CB 3)	0.92 ± 1.04	11.98 ± 6.00	47.10 ± 15.18
Breakout (CP 0) (CB 4)	1.32 ± 1.27	12.76 ± 10.01	46.08 ± 19.63
Breakout (CP 1) (CB 0)	0.96 ± 0.92	13.04 ± 7.46	50.06 ± 18.65
Breakout (CP 1) (CB 1)	1.10 ± 1.17	14.66 ± 12.33	43.28 ± 16.37
Breakout (CP 1) (CB 2)	1.18 ± 1.13	13.5 ± 9.99	43.96 ± 17.17
Breakout (CP 1) (CB 3)	1.32 ± 1.42	13.48 ± 7.23	44.56 ± 17.16
Breakout (CP 1) (CB 4)	1.36 ± 1.20	13.06 ± 8.93	44.90 ± 16.68
Breakout (CP 2) (CB 0)	1.54 ± 1.34	12.92 ± 7.68	45.50 ± 15.40
Breakout (CP 2) (CB 1)	1.12 ± 1.11	13.46 ± 8.84	44.56 ± 19.90
Breakout (CP 2) (CB 2)	1.40 ± 1.23	13.40 ± 9.18	43.60 ± 16.50
Breakout (CP 2) (CB 3)	1.26 ± 1.35	12.92 ± 10.40	43.46 ± 16.69
Breakout (CP 2) (CB 4)	1.26 ± 1.18	12.68 ± 9.02	43.10 ± 14.02
Breakout (CP 3) (CB 0)	0.98 ± 1.12	13.42 ± 8.76	39.06 ± 16.54
Breakout (CP 3) (CB 1)	0.98 ± 1.12	12.8 ± 7.91	45.88 ± 15.49
Breakout (CP 3) (CB 2)	0.96 ± 0.94	12.72 ± 8.24	45.84 ± 14.17
Breakout (CP 3) (CB 3)	0.96 ± 1.06	13.12 ± 8.83	44.38 ± 17.33
Breakout (CP 3) (CB 4)	1.14 ± 1.10	13.44 ± 10.23	42.30 ± 17.72
Breakout (CP 4) (CB 0)	1.46 ± 1.28	12.10 ± 7.02	42.86 ± 16.69
Breakout (CP 4) (CB 1)	1.16 ± 1.22	13.38 ± 11.42	43.42 ± 16.81
Breakout (CP 4) (CB 2)	1.32 ± 1.27	11.72 ± 5.50	47.80 ± 21.65
Breakout (CP 4) (CB 3)	0.90 ± 1.08	12.66 ± 7.85	46.52 ± 18.12
Breakout (CP 4) (CB 4)	0.96 ± 0.96	12.2 ± 6.77	42.94 ± 15.29

Table 9: Evaluations over 30 episodes, reporting mean and std, analyzing the performance over all possible variations of Breakout.

F Scaling Number of Models: Analysis

The Adding and Removal scenarios, Section 4.4 - Figure 4, provide insight into WSA's flexibility in managing dynamic model availability. In both scenarios, the pre-trained models are ordered as follows: (1) State-Representation (SR), (2) Object Key-Points Encoder (OKE), (3) Object Key-Points KeyNet (OKK), and (4) Video Object Segmentation (VOS). In the adding experiments, pre-trained models are added following the order $1 \rightarrow 4$, while in the removal models are removed following the order $4 \rightarrow 1$, as also suggested by the labels in Figures 13a-13b. Looking at Figure 13a, as new models are added to the pool, WSA slowly adapts its weights to leverage embeddings from all the models while maintaining optimal performance. In the removal scenario, Figure 13b, at the end of the training (before the first red mark), WSA assigns the highest importance to the OKE model. Removing other models does not affect critically performance and WSA is able to adapt, once the OKE model is removed, the performance collapses because the remaining model does not provide useful information.



(a) Adding Scenario

(b) Removal Scenario

Figure 13: Average and standard deviation of the weights assigned by our model to each component, represented as bar plots with error bars. Each group corresponds to an evaluation phase, aligned with the red vertical lines in Figure 4. (13a) shows the Adding Scenario, while (13b) depicts the Removal Scenario.

G CARBS Hyperparameters

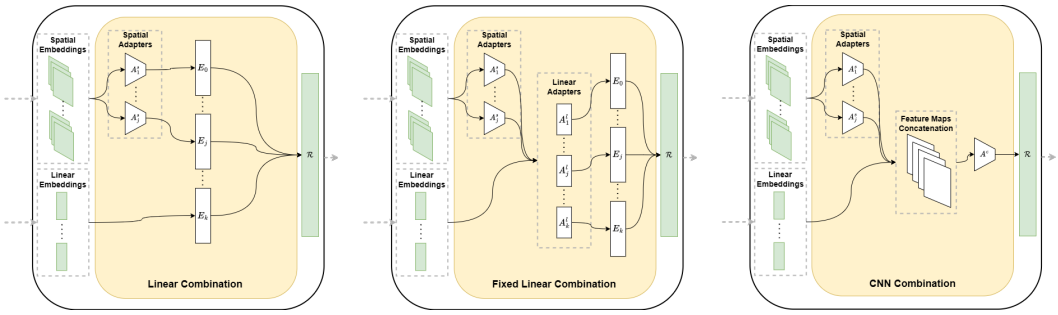
The hyperparameter search for the model training is performed using CARBS (Fetterman et al., 2023), which systematically explores different combinations of hyperparameters to optimize performance. Table 10 reports the ranges for the key hyperparameters used in the search.

Hyperparameter	**Range**
`Batch Size`	16,384 – 65,536
`BPTT Horizon`	$\{1, 2, 4, 8, 16\}$
`Clip Coefficient`	0.0 – 1.0
`Embedding Size`	128 – 1,024
`Entropy Coefficient`	$1e-5$ – $1e-1$
`GAE Lambda`	0.0 – 1.0
`Gamma`	0.0 – 1.0
`Learning Rate`	$1e-5$ – $1e-1$
`Max Gradient Norm`	0.0 – 10.0
`Minibatch Size`	512 – 2,048
`Total Timesteps`	5M – 25M
`Update Epochs`	1 – 4
`Value Function Clip Coefficient`	0.0 – 1.0
`Value Function Coefficient`	0.0 – 1.0

Table 10: Hyperparameter ranges used for the search.

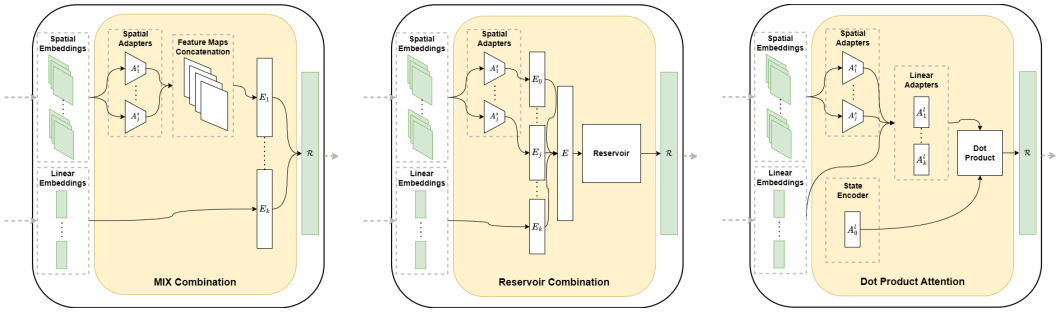
H Combination Modules Analysis: Definition, Cost and Performance

In this section, we provide additional insights for the results of the initial analysis. The experimental setup, outlined in Appendix B, structures a robust framework for evaluating the various combination modules.



(a) LIN (b) FIX (c) CNN

Figure 14: Graphical representation of different embedding combination modes that have been evaluated. The blocks representing the module can be placed in Figure 1 replacing WSA.



(a) MIX (b) RES (c) DPA

Figure 15: Graphical representation of different embedding combination modes that have been evaluated. The blocks representing the module can be placed in Figure 1 replacing WSA.

As shown in Table 11, we explore a variety of configurations for each module; visual representations are depicted in Figures 14,15, ensuring a comprehensive assessment of their performance. Table 12 reports the number of trainable parameter for each configuration. Table 13 presents the time needed to compute the embeddings for each pre-trained model used by WSA in our empirical evaluation. Tables 14-15 report respectively the timings during training and evaluation of different architectures. The E2E baseline, which uses the NatureCNN architecture (Mnih et al., 2013) (three convolutional layers and a linear layer), achieves the shortest average execution time, as expected. ENS and WSA show similar performance, with ENS being slightly faster due to the simpler combination of pre-trained model embeddings. The **computational overhead** introduced by WSA, mainly for weight computation and representation, is *minimal* and *comparable* to the ensemble approach, which only performs a concatenation. Moreover, the weight computation via the shared network and the construction of the final representation are efficient, requiring only a *fraction of a millisecond* more than the simple concatenation operation used in ENS. Nevertheless, WSA delivers a meaningful and informative representation that leads to improved agents performance, as shown in Table 1 (WSA vs ENS). For both methods, the majority of the time is spent on model inference. In fact, the primary bottleneck lies in the pre-trained models themselves, as shown in Table 13. The inference of the models takes on average *3.423 ms* that corresponds to **87%** of the time needed by WSA to encode observations. The pre-trained models are an arbitrary and replaceable design choice, independent of

the focus of our work. To further prove this point, C1, which uses a single SwinTransformer model (Liu et al., 2021), is the slowest due to its high computational cost. Finally, Figures 16, 17, and 18 illustrate the training performance of each agent, offering a visual representation of their learning trajectories. Early stopping was applied for agents that showed no improvement over multiple evaluations. This strategy allowed us to promptly identify the best-performing configurations.

Feature Extractor	**Embedding Size**
Linear Concatenation	-
Fixed Linear Concatenation	256, 512, 1024
Convolutional Concatenation	1, 2, 3
Mixed	-
Reservoir Concatenation	512, 1024, 2048
Dot Product Attention	256, 512, 1024
Weights Sharing Attention	256, 512, 1024

Table 11: All the extractors' configurations we tested in the initial phase of experiments. For Fixed Linear, Dot Product Attention, and Weights Sharing Attention, the values indicate the fixed dimensions of embeddings and context; for Reservoir, the size of the reservoir; and for Convolutional, the number of convolutional layers.

Feature Extractor	**Parameters**
Linear Concatenation	8.7M
Fixed Linear Concatenation	4.9M–19.4M
Convolutional Concatenation	4.2M
Mixed	4.5M
Reservoir Concatenation	0.3M–1.1M
Dot Product Attention	8.7M–34.7M
Weights Sharing Attention	8.7M–34.7M
End-to-end	1.6M

Table 12: Number of lear parameters, from the smallest to the biggest configuration, based on the embedding size presented in Tab. 11.

Model	**Inference Time (ms)**
State-Representation (Anand et al., 2019)	0.812 ± 0.082
Object Key-Points Encoder (Kulkarni et al., 2019)	0.767 ± 0.121
Object Key-Points KeyNet (Jakab et al., 2018; Kulkarni et al., 2019)	0.757 ± 0.095
Video Object Segmentation (Goel et al., 2018)	0.819 ± 0.126
Autoencoder	0.268 ± 0.028
Average Total	*3.423 ms*

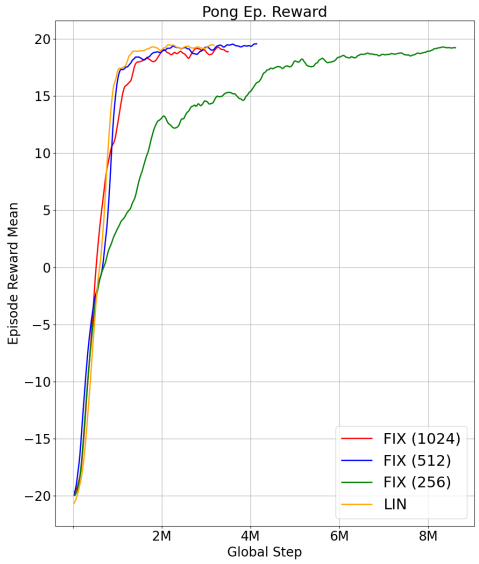
Table 13: Inference timings of pre-trained models used in our work.

Step	**E2E**	**ENS**	**WSA**	**C1**
Encode Observations	0.5129 ± 0.0561	3.7425 ± 0.5268	3.9020 ± 0.4017	9.6541 ± 1.0187
Policy Network	0.1321 ± 0.0103	0.1191 ± 0.0140	0.1113 ± 0.0073	0.1128 ± 0.0122

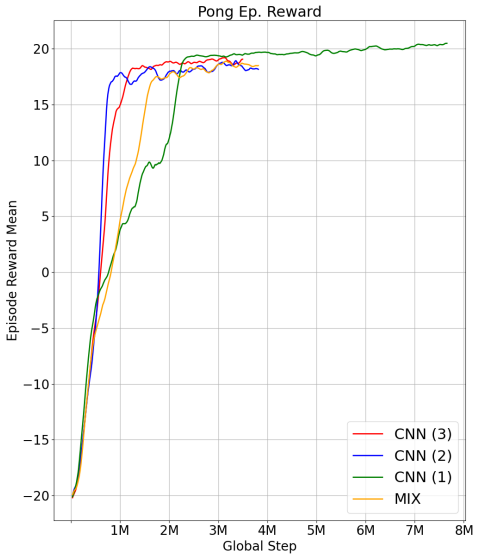
Table 14: Training timings breakdown (in *ms*) of the two main step comparing different architectures.

Step	**E2E**	**ENS**	**WSA**	**C1**
Encode Observations	0.3896 ± 0.0670	3.6156 ± 0.3875	3.9071 ± 0.4153	10.9849 ± 1.1776
Policy Network	0.1127 ± 0.0049	0.1162 ± 0.0084	0.1120 ± 0.0102	0.1191 ± 0.0101

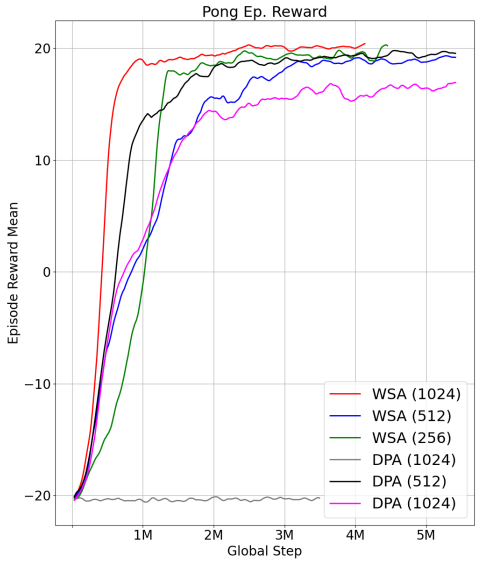
Table 15: Evaluation timings breakdown (in *ms*) of the two main step comparing different architectures.



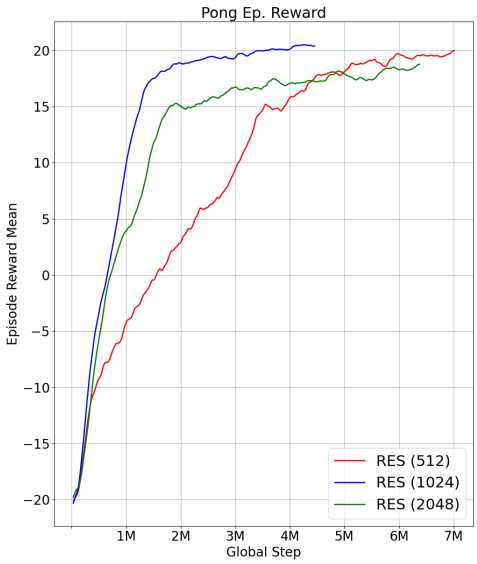
(a) `Linear` and `Fixed Linear`



(b) `Convolutional` and `Mixed`

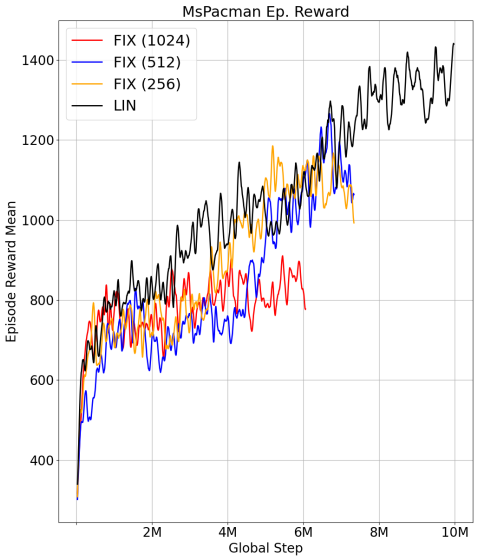


(c) `Weight Sharing Attention` and `Dot Product Attention`

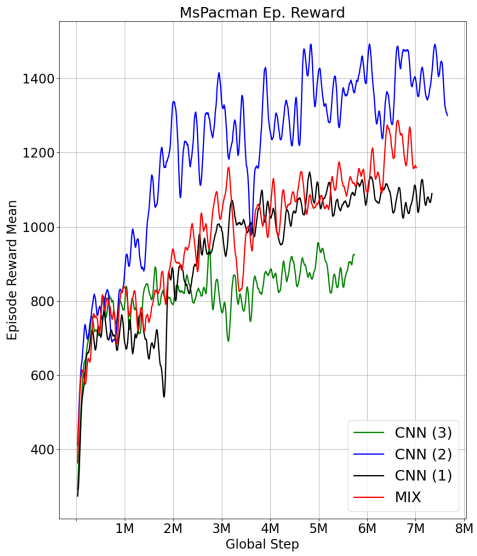


(d) `Reservoir`

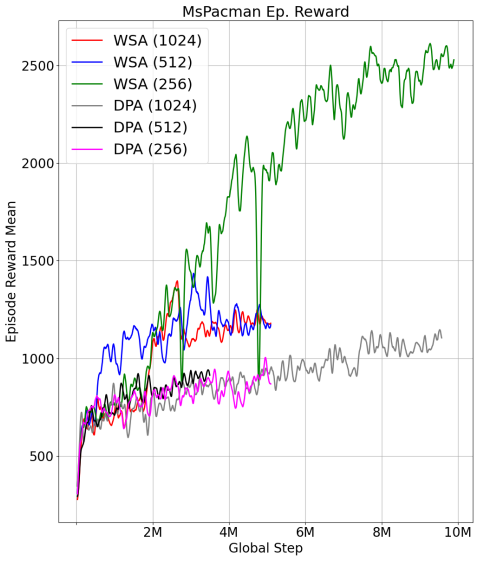
Figure 16: Initial analysis between different combination modules configurations in `Pong`.



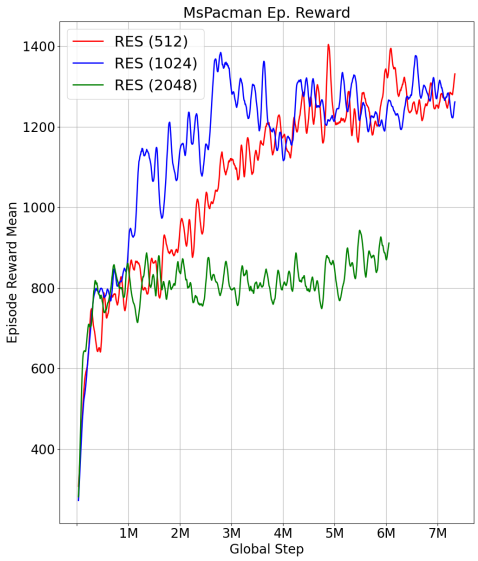
(a) `Linear` and `Fixed Linear`



(b) `Convolutional` and `Mixed`

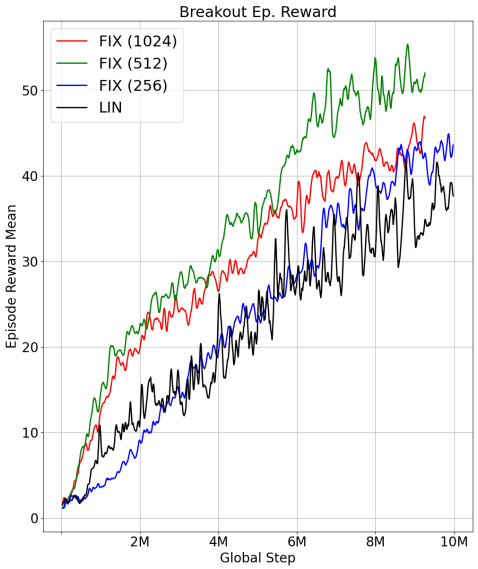


(c) `Weight Sharing Attention` and `Dot Product Attention`

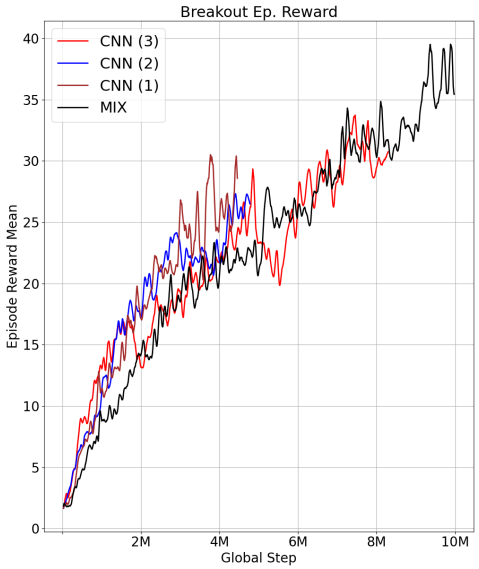


(d) `Reservoir`

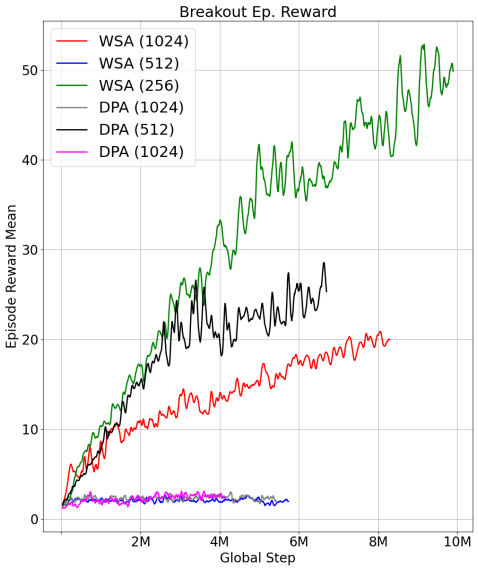
Figure 17: Initial analysis between different combination modules configurations in `Ms.Pacman`.



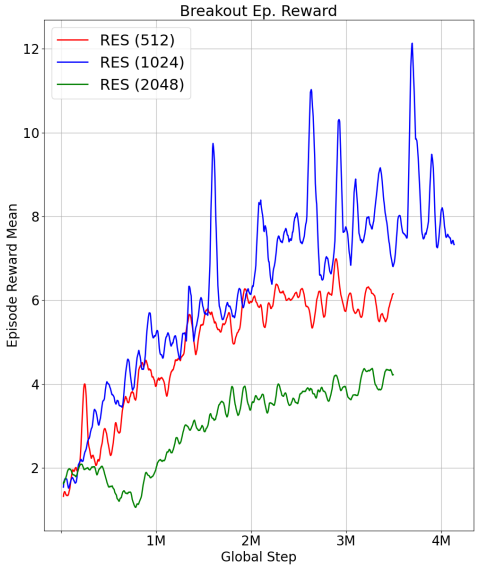
(a) `Linear` and `Fixed Linear`



(b) `Convolutional` and `Mixed`



(c) `Weight Sharing Attention` and `Dot Product Attention`



(d) `Reservoir`

Figure 18: Initial analysis between different combination modules configurations in `Breakout`.