An Investigation into the Open World Survival Game Crafter

Aleksandar Stanić [† 1] **Yujin Tang** [2] **David Ha** [2] **Jürgen Schmidhuber** [1 3]

Abstract

We share our experience with the recently released Crafter benchmark, a 2D open world survival game. Crafter allows tractable investigation of novel agents and their generalization, exploration and long-term reasoning capabilities. We evaluate agents on the original Crafter environment, as well as on a newly introduced set of generalization environments, suitable for evaluating agents' robustness to unseen objects and fast-adaptation (meta-learning) capabilities. Through several experiments we provide a couple of critical insights that are of general interest for future work on Crafter. We find that: (1) Simple agents with tuned hyper-parameters outperform all previous agents. (2) Feedforward agents can unlock almost all achievements by relying on the inventory display. (3) Recurrent agents improve on feedforward ones, also without the inventory information. (4) Baseline agents fail to generalize to OOD objects, object-centric agents improve over them. We will open-source our code.

1. Introduction

Common benchmarks and solid baselines are essential for developing new models and correctly measuring progress in machine learning. Datasets in supervised learning such as ImageNet (Deng et al., 2009) and environments in reinforcement learning (RL) such as Atari (Bellemare et al., 2013), ProcGen (Cobbe et al., 2020) and MineRL (Guss et al., 2021) have played a crucial role in improving the existing and creating novel models. Equally importantly, it is critical to compare with solid, yet simple baselines as the ones we introduce in this paper, to judge the improvements offered by novel methods. This is particularly the case in deep RL, where reproducing existing work is often hard due to a plethora of difficulties (Henderson et al., 2018).

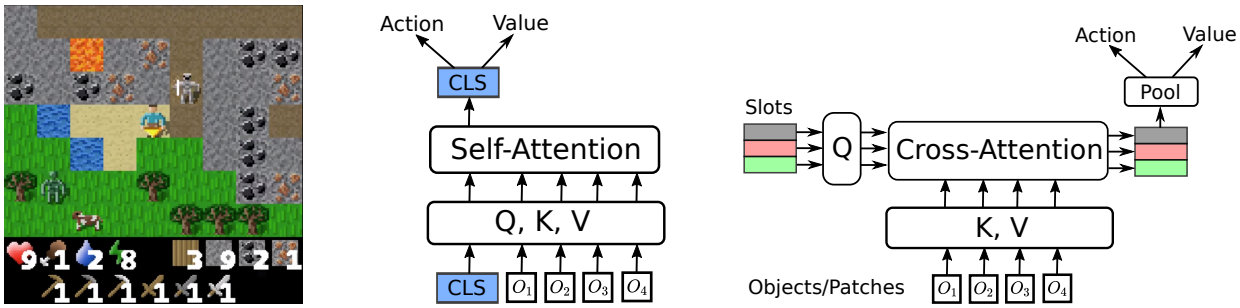
Early benchmarks were an important milestone to show that deep RL methods can learn control from high-dimensional images (Koutník et al., 2013; Mnih et al., 2015). However, they focused on a narrow set of tasks and IID setting between training and evaluation environments, where the agents could memorize a sequence of actions that leads to high rewards, without *understanding* the mechanics of the world. For better evaluations, the focus has been recently shifted to closely study the agents' behaviors through their performance on carefully designed benchmarks (Osband et al., 2020) and to learning agents that can generalize to environments with distributions beyond what they are trained on (e.g. ProcGen or Crafter).

In this work we focus on Crafter (Hafner, 2021), a recently introduced open world survival game, that allows tractable investigation of new agents and their generalization, exploration and long-term reasoning capabilities. Compared to other benchmarks, Crafter has a set of advantages that make it suitable for RL research such as fast iteration speed (agents can be trained in a few hours on a standard GPU and a single CPU core), model evaluation by inspecting semantically meaningful achievements the agents unlocked, and controllable environment objects that facilitates systematic studies. On the original Crafter environment our experiments show important, previously unpublished insights into baselines and environment workings. Crucially we establish solid baselines by showing that simple agents with tuned hyper-parameters outperform all previous agents. We also introduce a set of new environments, CrafterOOD, that test agents' generalization and robustness to unseen objects, setting the stage for development of fast-adaptation algorithms such as meta-learning.

Our main findings can be summarised as follows: (1) Simple agents with tuned hyper-parameters outperform all previous agents. (2) Feedforward (FF) agents can unlock almost all achievements by relying on the inventory display (bottom part of the image in Figure 1a) as a form of a "scratchpad". (3) Recurrent agents improve on FF ones, even without the inventory information. (4) Baseline agents fail to generalize to out-of-distribution (OOD) objects, object-centric agents improve over them. Researchers should be aware of these baseline and environment insights when testing new ideas on the Crafter environment.

† The majority of this work was carried out as a student researcher at Google Brain. [1]IDSIA, USI, SUPSI, Lugano, Switzerland [2]Google Brain [3]AI Initiative, KAUST, Thuwal, Saudi Arabia. Correspondence to: Aleksandar Stanić <aleksandar@idsia.ch>.

Decision Awareness in Reinforcement Learning Workshop at the 39th International Conference on Machine Learning (ICML), Baltimore, Maryland, USA, 2022. Copyright 2022 by the author(s).



(a) (b) (c)

Figure 1: *(a) Crafter gameplay. (b) Self-attention (SA) model with a CLS token. (c) Slot-based cross-attention (CA) model.*

2. Environments

Here we briefly summarize essential properties of the Crafter environment and the newly introduced CrafterOOD variant. For additional details we refer to (Hafner, 2021).

Crafter. Crafter is an open world survival game for RL research, whose game dynamics are inspired by popular game Minecraft. The benchmark is designed to facilitate existing research challenges, such as strong generalization via procedural generation, deep exploration via achievements conditioned on one another, learning from high-dimensional image observations and sparse rewards that require long-term reasoning and credit assignment. It facilitates evaluation by unlocking semantically meaningful achievements and fast iteration speed. In Crafter a unique terrain is generated for every episode with grasslands, lakes and mountains, that contain forests, caves, ores and lava. The agent needs to collect food and water, protect against enemies and collect resources to craft tools to then unlock all achievements, with collecting the diamond being the last and the most difficult one (see example gameplay in Figure 1a). There are 22 achievements and the main evaluation metric is the *crafter score* S computed by averaging unlocked achievements in the log space (to account for differences in their difficulties): $S = \exp(\frac{1}{N}\sum_{i=1}^{N} \ln(1+s_i))$, where $s_i \in [0, 100]$ is percentage of episodes in which achievement i was unlocked.

CrafterOOD. To stay alive in Crafter, the player eats food (plants or cows), drinks water and protects itself from enemies. To craft tools the player needs to collect resources such as wood, stone, coal and iron. All these agents and resources are represented as objects in the environment. We introduce a set of new environments, CrafterOOD, where an agent is trained on one distribution of such objects and then evaluated on another one, possibly one which contains objects never seen during training. In this way CrafterOOD facilitates testing agents' generalization and robustness to unseen objects and sets the stage for development of fast-adaptation algorithms such as meta-learning. We introduce new variants for objects the agent most frequently interacts with: trees, cows, zombies, stones, coal and skeletons. For a detailed overview of newly introduced objects see Figure 3.

3. Methods

We base all our methods on the PPO (Schulman et al., 2017) implementation in the stable baselines (Raffin et al., 2021).

3.1. Linear and Recurrent PPO

PPO learns to map images to actions via policy gradients. Two FF variants are investigated, that differ by the CNN policy, and a recurrent version based on LSTM (Wierstra et al., 2007; Hochreiter & Schmidhuber, 1997) (see Appendix G).

PPO with NatureCNN (PPO-CNN). This baseline is architecturally identical to the one used in (Hafner, 2021), with the CNN policy from the DQN paper (Mnih et al., 2015).

PPO with size-preserving CNN (PPO-SPCNN). Size-preserving CNN (SPCNN) differs from the PPO-CNN baseline by the CNN architecture. We introduce it as a baseline for attention-based agents that use SPCNN for "feature mixing", inspired by SlotAttention (Locatello et al., 2020). SPCNN does not have pooling layers, so the resulting output tensor is of the same height and width as the input image. The flattened tensor that is fed into a linear layer is much larger (64x64x64 instead of 8x8x64 for CNN). Therefore, the policy is also very large (134M parameters).

Recurrent PPO (RecPPO). Crafter is a partially observable environment. The agent observes only a part of the world (see Figure 1a). To perform well it needs to remember the location of resources, e.g. food, mining materials and where it placed objects for crafting new tools, e.g. table or furnace. Additionally, some achievements require the agent to perform a long chain of reasoning (see Figure 4 in (Hafner, 2021) for the complete overview). For these reasons, we introduce recurrent agents (RecPPO-CNN and RecPPO-SPCNN) where we use LSTMs as the critic and the actor networks. These networks in theory entitle the agent to have memories and can help unlock achievements.

3.2. Attention-based PPO

Our experiments show that CNN agents fail to generalize to OOD environments. Recently, object-centric methods have successfully been used for OOD generalization in supervised and unsupervised learning (Greff et al., 2017; van Steenkiste et al., 2018; Kosiorek et al., 2018; Stanić & Schmidhuber, 2019; Greff et al., 2019; Locatello et al., 2020; Stanić et al., 2020; Kipf et al., 2021) and in RL (Watters et al., 2019; Veerapaneni et al., 2020; Kipf et al., 2020; Carvalho et al., 2021) though previous work did not consider procedurally generated open world games. Since Crafter is composed of objects, we expected these methods to facilitate learning and show stronger generalization capabilities. In this vein, we designed two attention-based agents and investigated them on CrafterOOD. They take as input either patches extracted from the image through a size-preserving CNN (Figure 5) or learn their own representation of an object (by attending over the whole input tensor Figure 9). In Crafter, it is possible to crop patches corresponding to individual objects, which is convenient for complexity control when developing new methods (see Appendix G for details).

PPO with self-attention (PPO-SA) (Figure 1b) learns a policy via a dot-product self-attention between the input patches and a learned "CLS" token (a vector initialized to unit Gaussian parameters and optimized via backprop). This is similar to using the CLS token in BERT (Devlin et al., 2018). Let $(x_1, ..., x_k, CLS) \in \mathbb{R}^{(k+1)\times d_{in}}$ be the input sequence, where k is the number of input patches and d_{in} is their dimensionality. Dot-product self-attention is defined as $Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$, where $Q \in \mathbb{R}^{(k+1)\times d}$, $K \in \mathbb{R}^{(k+1)\times d}$, $V \in \mathbb{R}^{(k+1)\times d}$, are the query, key and value matrices, resulting from linear mapping of the input sequence onto a space of dimension d. We add positional embeddings to the input, which enables learning relative patch positions, e.g. if an enemy is nearby.

PPO with cross-attention networks (PPO-CA) (Figure 1c) learns a set of vectors, which we refer to as 'slots'. This idea was proposed in SetTransformer (Lee et al., 2019) and successfully used for unsupervised learning of objects (Locatello et al., 2020), learning permutation-invariant agents (Tang et al., 2020; Tang & Ha, 2021) and general perception modules (Jaegle et al., 2021b;a; Alayrac et al., 2022). PPO-CA computes attention between queries, keys and values as in self-attention, with the queries coming from the n learned slots, so we have $Q \in \mathbb{R}^{n\times d}$, $K \in \mathbb{R}^{k\times d}$, $V \in \mathbb{R}^{k\times d}$, where k is the number of input patches. In the extreme case, the patch size is 1, which allows higher expressiveness as each slot can attend to variable-size or further apart image regions. After cross-attention, slots are pooled by a mean operation and then fed into the policy.

Table 1: Scores on Crafter for agents trained on 1M environment steps. Reported are mean scores and standard deviations of 10 random seeds. *score from (Hafner, 2021).

METHOD	CRAFTER SCORE
PPO*	4.6 ± 0.3
DREAMERV2*	10.0 ± 0.2
PPO-CNN	10.3 ± 0.6
PPO-SPCNN	11.6 ± 0.6
RECPPO-CNN	10.4 ± 0.2
RECPPO-SPCNN	$\mathbf{12.1} \pm 0.8$
PPO-SA	11.1 ± 0.7
PPO-CA	11.0 ± 0.4
PPO-CNN (NO INVENTORY)	6.9 ± 0.4
RECPPO-CNN (NO INVENTORY)	$\mathbf{7.7} \pm 0.5$

4. Experiments and Core Findings

Improved baselines. By tuning a few hyper-parameters, we find that the simple PPO-CNN baseline outperforms the model-based DreamerV2 (Hafner et al., 2020) (Table 1). Surprisingly, even though PPO-SPCNN model has 134M parameters, PPO is not only able to train it, but it also outperforms all other variants in both IID and OOD settings (Tables 1 and 2). This holds for both FF (PPO-SPCNN) and the recurrent (RecPPO-SPCNN) models. We tried tuning DreamerV2, but did not improve on results in (Hafner, 2021) (see Appendix E for hyper-parameters we investigated).

Agents use inventory display as a "scratchpad." Although purely FF, PPO-SPCNN is able to unlock almost all achievements (19/22). This essentially requires the agent to remember the unlocked achievements (e.g I have an iron pick-axe now, need to dig a diamond next). However, the fact that the memory-less FF models unlocked most achievements leads us to hypothesize that it is using the inventory on the input image as a "scratchpad." We test this by training agents with the inventory removed.The results speak in favor of our hypothesis, as the performance significantly drops (compare PPO-CNN and "PPO-CNN (no inventory)" in Table 1).

Recurrent improve over FF agents, even without inventory information. Agent's reliance on inventory display as a "scratchpad" lead us to investigating whether recurrent agents could improve on FF ones by memorizing actions, map, or unlocked achievements. With observable inventory, RecPPO-CNN does not improve upon FF variant PPO-CNN (10.3 vs 10.4 in Table 1). However, when the inventory is not observable RecPPO-CNN outperforms PPO-CNN (7.7 vs 6.9). This might indicate that the recurrent agents learn to store achievements in the memory, although not perfectly as we observe a drop from the case with observable inventory. Note also that RecPPO-SPCNN outperforms its FF variant PPO-SPCNN in both IID and OOD settings (Table 2). Here the inventory is observed, so the largest benefit must arise from memorizing the map layout.

Table 2: Scores on CrafterOOD (mean and standard deviations over 10 random seeds) for agents trained for 1M environment steps. Each setting has two rows, denoting training (e.g. $O_{1-4}:25\%$) and evaluation ($O_1:0\%, O_{2-4}:33\%$) scores.

TRAIN/EVAL DIST	PPO-CNN	PPO-SPCNN	RECPPO-CNN	RECPPO-SPCNN	PPO-SA	PPO-CA
TRAINING: $O_1:100\%$	10.3 ± 0.6	11.6 ± 0.6	10.4 ± 0.2	12.1 ± 0.8	11.1 ± 0.7	10.0 ± 0.4
EVALUATION: $O_1:100\%$	10.3 ± 0.6	11.6 ± 0.6	10.4 ± 0.2	$\mathbf{12.1} \pm 0.8$	11.1 ± 0.7	10.0 ± 0.4
$O_{1-4}:25\%$	9.2 ± 0.5	10.7 ± 0.6	10.7 ± 0.6	11.5 ± 0.4	9.7 ± 1.1	9.9 ± 0.6
$O_1:0\%,\ O_{2-4}:33.3\%$	9.2 ± 0.7	11.0 ± 1.1	11.0 ± 1.0	$\mathbf{11.6} \pm 0.6$	9.7 ± 1.2	9.2 ± 0.7
$O_1:52\%, O_{2-4}:16\%$	9.9 ± 0.5	11.1 ± 0.7	11.5 ± 1.4	11.1 ± 0.5	9.6 ± 0.8	9.4 ± 0.9
$O_1:0\%,\ O_{2-4}:33.3\%$	10.0 ± 0.7	11.2 ± 1.1	$\mathbf{11.4} \pm 1.6$	11.0 ± 0.5	10.6 ± 0.9	9.9 ± 0.9
$O_1:76\%, O_{2-4}:8\%$	9.9 ± 0.4	11.5 ± 0.6	10.7 ± 1.0	11.6 ± 0.6	9.8 ± 0.8	11.3 ± 0.5
$O_1:0\%,\ O_{2-4}:33.3\%$	9.2 ± 0.6	10.5 ± 0.8	10.4 ± 1.0	$\mathbf{10.7} \pm 0.7$	9.2 ± 0.8	10.5 ± 0.7
$O_1:88\%, O_{2-4}:4\%$	10.1 ± 0.6	12.2 ± 0.8	11.5 ± 1.4	11.3 ± 0.4	10.5 ± 1	11.2 ± 0.9
$O_1:0\%,\ O_{2-4}:33.3\%$	9.1 ± 0.7	$\mathbf{10.2} \pm 0.7$	10.1 ± 1.3	9.8 ± 0.8	9.4 ± 1.3	9.4 ± 1.0
$O_1:94\%, O_{2-4}:2\%$	10.9 ± 0.7	12.0 ± 0.8	11.4 ± 1.2	11.7 ± 0.6	10.5 ± 0.6	10.8 ± 1.1
$O_1:0\%,\ O_{2-4}:33.3\%$	8.6 ± 0.7	9.2 ± 1.1	9.1 ± 1.4	9.8 ± 0.8	$\mathbf{9.9} \pm 0.8$	8.8 ± 0.9
$O_1:97\%, O_{2-4}:1\%$	10.5 ± 0.6	11.8 ± 0.7	11.9 ± 1.4	12.0 ± 0.3	10.3 ± 1.1	10.8 ± 0.6
$O_1:0\%,\ O_{2-4}:33.3\%$	7.3 ± 0.5	7.7 ± 1.0	8.2 ± 1.0	8.6 ± 1.0	$\mathbf{9.3} \pm 0.8$	8.8 ± 0.8
$O_1:100\%, O_{2-4}:0\%$	10.5 ± 0.6	11.8 ± 0.6	10.7 ± 0.2	11.9 ± 0.8	11.1 ± 1.3	10.7 ± 0.6
$O_1:0\%,\ O_{2-4}:33.3\%$	7.3 ± 0.5	7.7 ± 0.9	5.8 ± 0.2	6.8 ± 1.0	$\mathbf{8.0} \pm 1.2$	7.6 ± 0.5

CrafterOOD generalization. We generate a collection of increasingly difficult adaptation scenarios. Agents are trained and evaluated on progressively more distinct environments. Environments contain four different object variants for trees, cows, iron, stones, zombies and skeletons. The settings differ in object distributions in training and evaluation environments. For example, the IID case corresponds to observing only the first object variant O_1 during training and evaluation (first two rows in Table 2). Starting from training with uniformly distributed objects ($O_{1-4} = 25\%$) we make generalization progressively more difficult by skewing the training distribution towards the first object. The evaluation environment always contains only the last three objects uniformly distributed. In Table 2 we see that up to the point of observing evaluation objects 16% of the time agents generalize fairly well (see also Figure 4 in Appendix B). Decreasing the percentage of evaluation objects further, the performance of all agents consistently drops. Finally, when trained only on the first object ($O_1 = 100\%$) the agent relies on pure chance or interacting with objects that do not change (e.g. water, iron). Our experiments show that the object-centric agents PPO-SA and PPO-CA match the vanilla PPO agents on the IID and easy OOD generalization cases (the last two columns in Table 2). In the most difficult OOD generalization cases though (the last three pairs of rows in Table 2), PPO-SA and PPO-CA show better generalization compared to the PPO-CNN and PPO-SPCNN. To the best of our knowledge, this is the first time cross-attention-based methods (e.g. SlotAttention- and Perceiver-like) are applied to an open-world RL survival game, and we found their vanilla versions not to work out-of-the-box. To achieve good level of performance we analyzed their components and present our findings in Appendix F. This analysis led us to converge to an architecture with a single cross-attention over the input, no latent self-attention, no layer normalization, no residual connections and (unlike SlotAttention) no competition between the slots via softmax over queries. Additionally, these methods let us inspect their attention, making them interpretable and potentially easier to build on in future work (see Appendix C and D).

5. Conclusion and Discussion

For research on new machine learning methods and proper progress measuring, it is crucial to start from strong baselines. We reported a couple of important observations from our experiments with the Crafter baselines. These insights indicate that a systematic study of baselines and environment workings (e.g. inventory acting as a "scratchpad") is needed. Furthermore, we introduced CrafterOOD, an environment variant that is OOD in objects appearance and show that when the evaluation environment contains unseen objects the baseline agents fail to adapt, but the object-centric ones improve over them. This might be a fruitful environment for developing fast-adaptation and meta-learning agents. Also it would be interesting to consider other OOD variants of Crafter, such as ones with objects of varying sizes or frequencies of appearance. We see Crafter as a good candidate to explore all these direction as one has control over the environment configuration, can inspect models visually and have fast iteration cycles with our solid baselines. We hope our findings will be of interest and taken into consideration for future work on Crafter.

Acknowledgements

We thank Danijar Hafner, Shixiang Shane Gu and Yingtao Tian for their useful comments and suggestions on an earlier version of this paper. This research was partially funded by the ERC Advanced grant no: 742870, project AlgoRNN.

References

Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., et al. Flamingo: a visual language model for few-shot learning. *arXiv preprint arXiv:2204.14198*, 2022.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

Carvalho, W., Lampinen, A., Nikiforou, K., Hill, F., and Shanahan, M. Feature-attending recurrent modules for generalization in reinforcement learning. *arXiv preprint arXiv:2112.08369*, 2021.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pp. 2048–2056. PMLR, 2020.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Greff, K., van Steenkiste, S., and Schmidhuber, J. Neural expectation maximization. In *Advances in Neural Information Processing Systems*, pp. 6691–6701, 2017.

Greff, K., Kaufman, R. L., Kabra, R., Watters, N., Burgess, C., Zoran, D., Matthey, L., Botvinick, M., and Lerchner, A. Multi-object representation learning with iterative variational inference. In *International Conference on Machine Learning*, pp. 2424–2433, 2019.

Guss, W. H., Castro, M. Y., Devlin, S., Houghton, B., Kuno, N. S., Loomis, C., Milani, S., Mohanty, S., Nakata, K., Salakhutdinov, R., et al. The minerl 2020 competition on sample efficient reinforcement learning using human priors. *arXiv preprint arXiv:2101.11071*, 2021.

Hafner, D. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*, 2021.

Hafner, D., Lillicrap, T., Norouzi, M., and Ba, J. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

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

Jaegle, A., Borgeaud, S., Alayrac, J.-B., Doersch, C., Ionescu, C., Ding, D., Koppula, S., Zoran, D., Brock, A., Shelhamer, E., et al. Perceiver io: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795*, 2021a.

Jaegle, A., Gimeno, F., Brock, A., Vinyals, O., Zisserman, A., and Carreira, J. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning*, pp. 4651–4664. PMLR, 2021b.

Kipf, T., van der Pol, E., and Welling, M. Contrastive learning of structured world models. In *International Conference on Learning Representations*, 2020.

Kipf, T., Elsayed, G. F., Mahendran, A., Stone, A., Sabour, S., Heigold, G., Jonschkowski, R., Dosovitskiy, A., and Greff, K. Conditional object-centric learning from video. *arXiv preprint arXiv:2111.12594*, 2021.

Kosiorek, A. R., Kim, H., Posner, I., and Teh, Y. W. Sequential attend, infer, repeat: Generative modelling of moving objects. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 8615–8625, USA, 2018. Curran Associates Inc.

Koutník, J., Cuccu, G., Schmidhuber, J., and Gomez, F. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pp. 1061–1068, 2013.

Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pp. 3744–3753. PMLR, 2019.

Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T. Object-centric learning with slot attention. *Advances in Neural Information Processing Systems*, 33:11525–11538, 2020.

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

Osband, I., Doron, Y., Hessel, M., Aslanides, J., Sezener, E., Saraiva, A., McKinney, K., Lattimore, T., Szepesvári, C., Singh, S., Van Roy, B., Sutton, R., Silver, D., and van Hasselt, H. Behaviour suite for reinforcement learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygf-kSYwH.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. 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.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Stanić, A. and Schmidhuber, J. R-sqair: Relational sequential attend, infer, repeat. *NeurIPS PGR Workshop*, 2019.

Stanić, A., van Steenkiste, S., and Schmidhuber, J. Hierarchical relational inference. *arXiv preprint arXiv:2010.03635*, 2020.

Tang, Y. and Ha, D. The sensory neuron as a transformer: Permutation-invariant neural networks for reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Tang, Y., Nguyen, D., and Ha, D. Neuroevolution of self-interpretable agents. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 414–424, 2020.

van Steenkiste, S., Chang, M., Greff, K., and Schmidhuber, J. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. In *International Conference on Learning Representations*, 2018.

Veerapaneni, R., Co-Reyes, J. D., Chang, M., Janner, M., Finn, C., Wu, J., Tenenbaum, J., and Levine, S. Entity abstraction in visual model-based reinforcement learning. In *Conference on Robot Learning*, pp. 1439–1456. PMLR, 2020.

Watters, N., Matthey, L., Bosnjak, M., Burgess, C. P., and Lerchner, A. Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration. *ICML workshop on Generative Modeling and Model-Based Reasoning for Robotics and AI*, 2019.

Wierstra, D., Foerster, A., Peters, J., and Schmidhuber, J. Solving deep memory pomdps with recurrent policy gradients. In *International conference on artificial neural networks*, pp. 697–706. Springer, 2007.

A. OOD environment textures

Figure 2: Original Crafter objects. Figure from (Hafner, 2021).

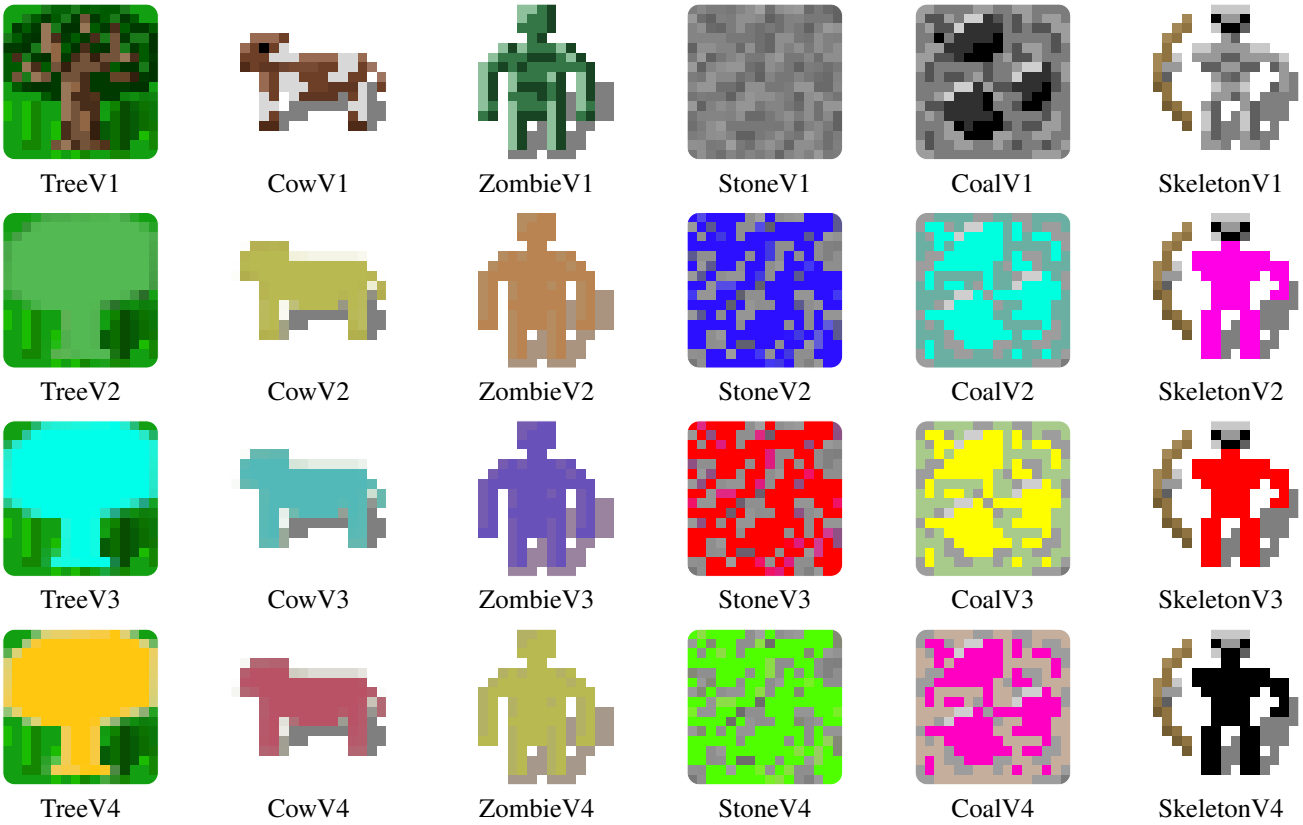


Figure 3: In CrafterOOD there are four variants of objects for trees, cows, zombies, stone coal and skeletons.

B. CrafterOOD performance

In Figure 4 we provide heatmap of the CrafterOOD scores previously reported in **??**.

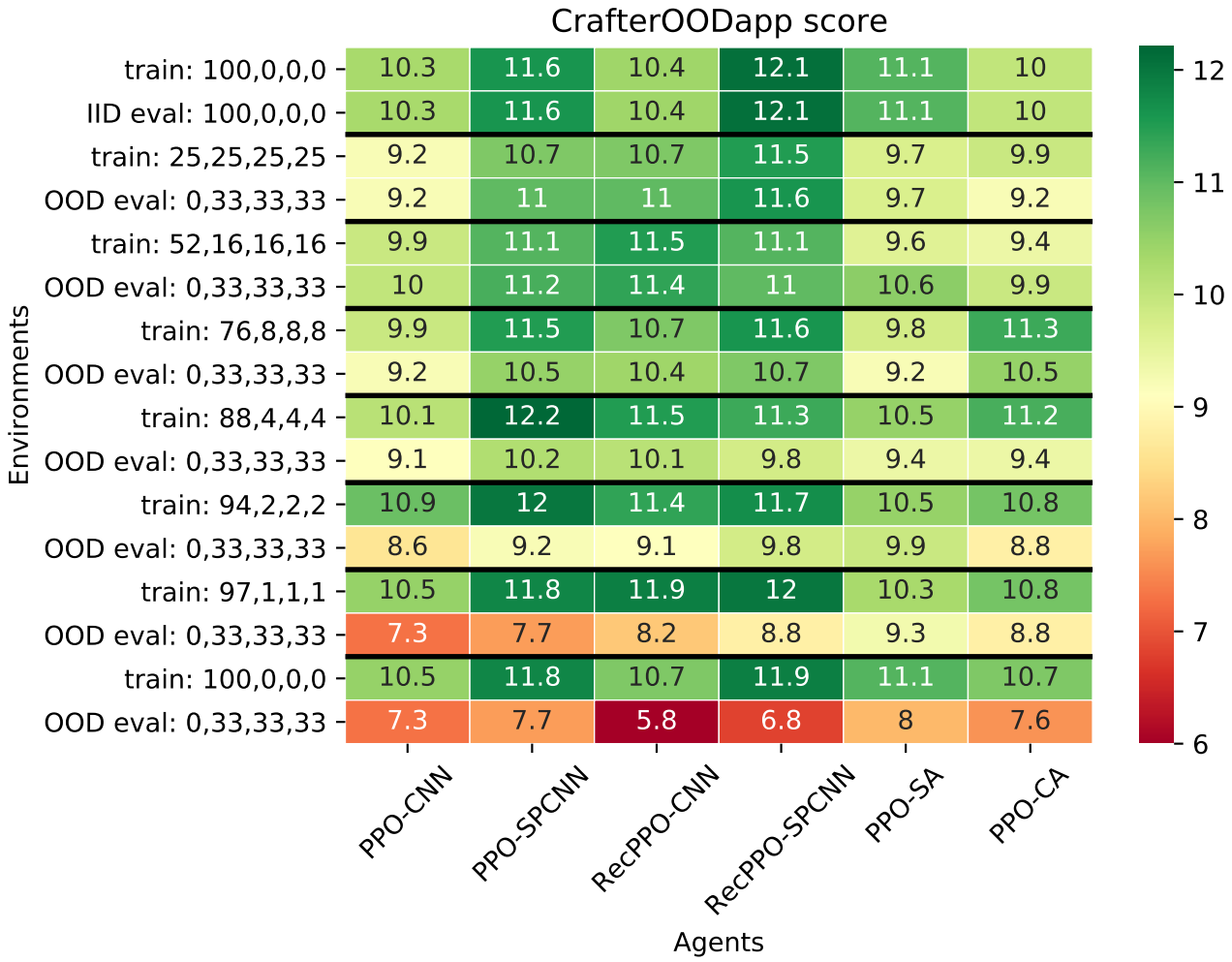


Figure 4: *Scores on CrafterOODnum for agents trained for 1M environment steps. Mean over 10 random seeds are reported. For standard deviations see Table 2. Each setting has two rows, denoting scores in training (e.g.* $25, 25, 25, 25$ *for* $O_{1-4} : 25\%$*) and evaluation (*$0, 33, 33, 33$*, for* $O_1 : 0\%, O_{2-4} : 33.3\%$*) environments.*

C. Attention Visualization in Self-Attention (SA)

In this section we visualize what the CLS token attends. Note that all the below visualizations stem from a single agent. They are representative to what we observed in most episodes of a trained agent.

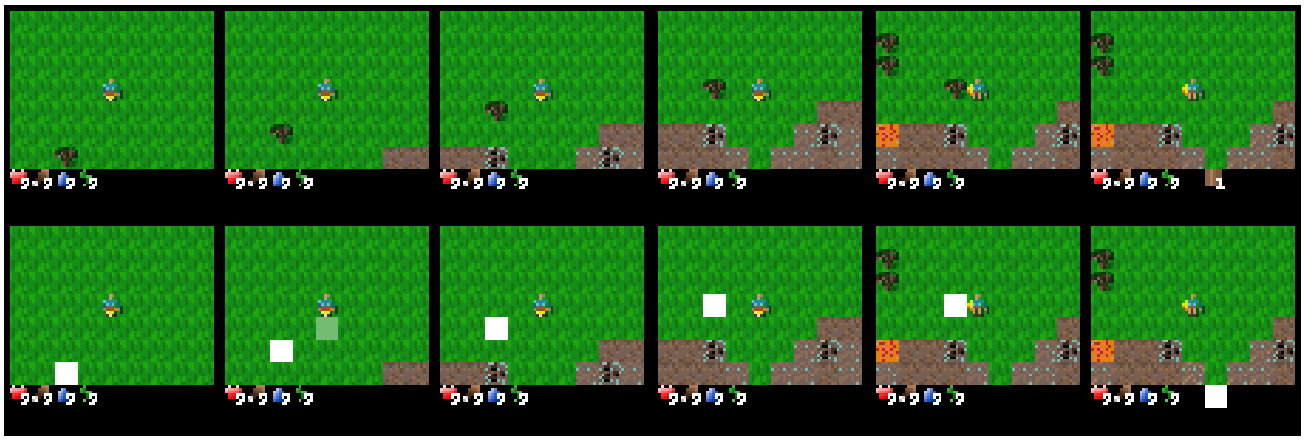


Figure 5: *Top row: input images. Bottom row: visualized attention by it's intensity. Horizontally are episode steps. The agent collects resources. The agent notices a tree in its vicinity. Through the frames it focuses consistently on the tree and collects it. In the final frame the agent attends to the (newly crafted) wood resource (sequence continued in Figure 6).*

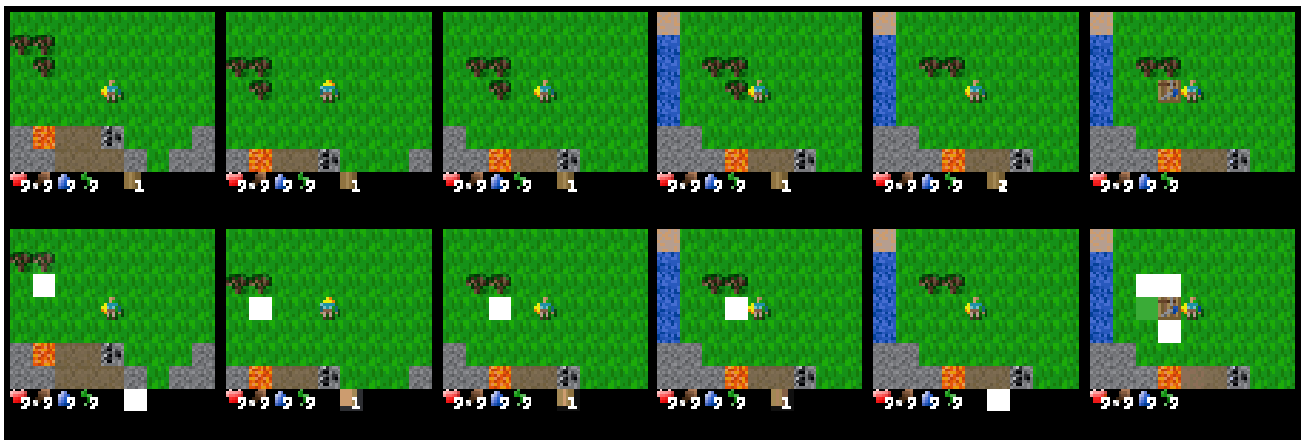


Figure 6: *Top row: input images. Bottom row: visualized attention by it's intensity. Horizontally are episode steps. Building a table (continued from Figure 5). After collecting one tree, the agent needs to collect another one to build a tree. It spots more trees in it's vicinity, goes to them, collects them and builds a table, such that it can build weapons on the table (continued in Figure 7.*

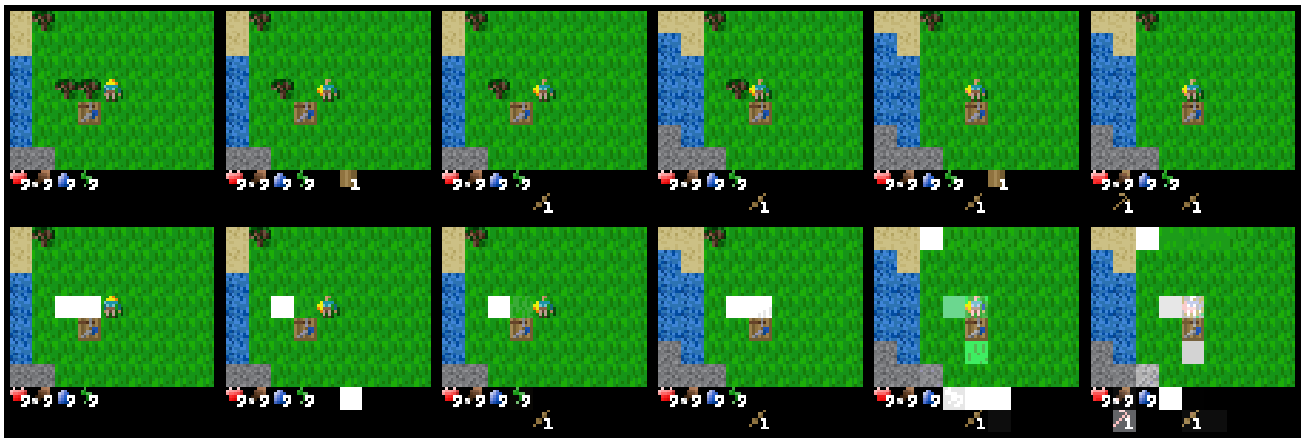


Figure 7: *Top row: input images. Bottom row: visualized attention by it's intensity. Horizontally are episode steps. Crafting weapons (continued from Figure 6). Once it has built a table, the agent needs to collect more resources to craft weapons. It collects the two nearby trees and uses them to build a wooden sword and a wooden pickaxe (shown in the bottom inventory). It can later use these weapons to defend against enemies, collect stones and collect coal.*

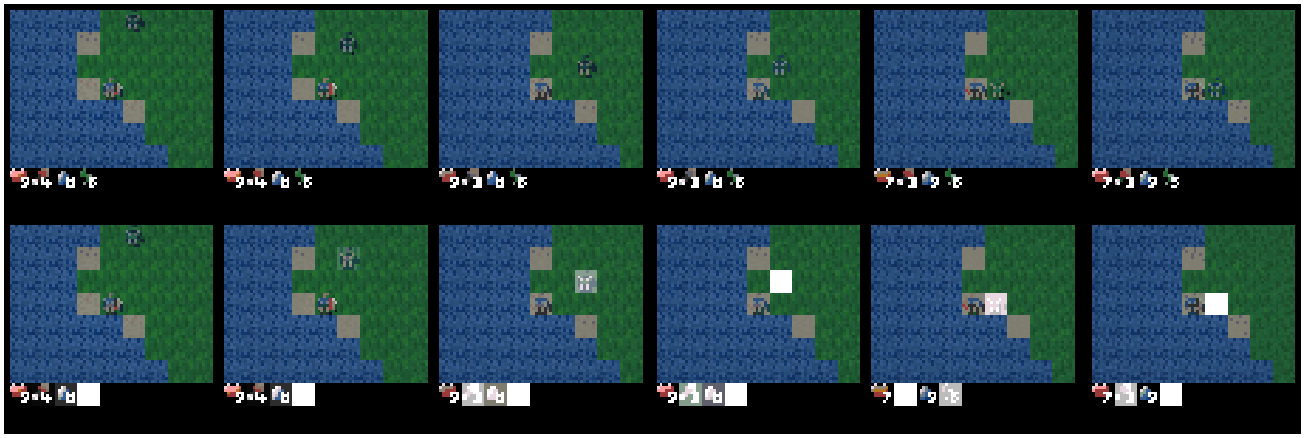


Figure 8: *Top row: input images. Bottom row: visualized attention by it's intensity. Horizontally are episode steps. The agent defends against enemies. During the night (the reason why the frame color is darker), a zombie enters the scene. In the first frame, the agent does not attend to the zombie, as it is still far away. As the zombie gets closer, it attends more and more to it.*

D. Learned Attention Visualization in Cross-Attention (CA)

In Figure 9 we visualize the learned attention patterns (not fixed to patches, but any slot can attend over the *entire* input image). We observe more numerous, but smaller attention patterns compared to patch-based attention that is typically focused on 2-5 regions in the image. Although in theory more powerful (because they can choose to ignore most parts of the image, or even the object at hand), in practice learned attention models underperform the patch-based ones. We suspect the reason could be that the learned attention agents first need to find out what an object actually is, whereas patch-based ones have a more "guided" learning process. We speculate that greater gains of learned attention would be in scenarios with objects of varying sizes, in which case a fixed grid is too rigid representation.

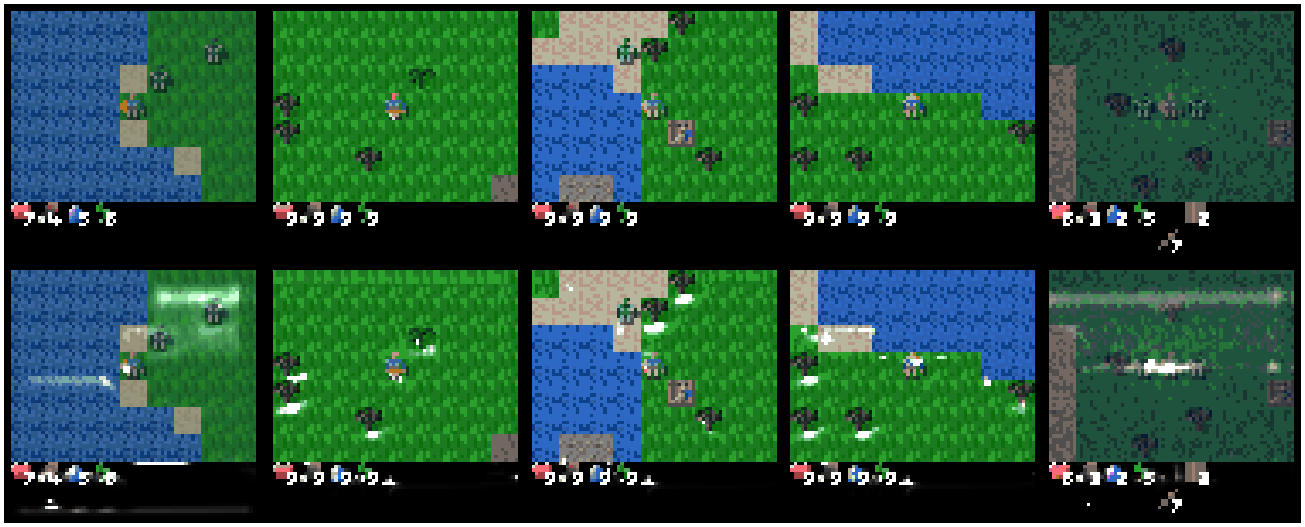


Figure 9: *Top row: input images. Bottom row: visualized attention by it's intensity. We find that the learned attention also attends to the salient objects in the scene: zombies (1st, 3rd and 5th columns), trees (2nd, 3rd and 4th columns), resources (water overall) and the inventory. We observe more numerous, but smaller attention patterns than in patch-based attention, which is typically focused on 2-5 regions in the image.*

E. Hyper-parameter Configurations

Table 3: PPO hyper-parameters we searched over (Sweep) and final values used in our experiments for FF and recurrent (Rec) agents. For most hyper-parameters we used default values (that were tuned for Atari in previous work), except for the bolded ones.

HYPER-PARAMETER	FINAL FF	FINAL REC	DEFAULT	SWEEP
LEARNING RATE	0.0003	0.0003	0.0003	[0.001, 0.0005, 0.0003, 0.0001, 0.00005]
BATCH SIZE	64	**128**	64	[16, 32, 64, 128]
NUMBER OF ROLLOUTS	2048	**4096**	2048	[1024, 2048, 4096]
NUMBER OF EPOCHS	**4**	**4**	10	[3, 4, 5, 7, 10]
DISCOUNT FACTOR	**0.95**	**0.95**	0.99	[0.95, 0.97, 0.99, 0.999]
GAE λ	**0.65**	**0.65**	0.95	[0.6, 0.65, 0.7, 0.8, 0.9, 0.95]
CLIP RANGE	0.2	0.2	0.2	[0.1, 0.2, 0.3]
MAX GRAD NORM	0.5	0.5	0.5	[0.1, 0.3, 0.5, 1.0]

Table 4: Dreamer hyper-parameters we searched over.

HYPER-PARAMETER	SWEEP
BATCH SIZE	$16, 32, 64$
DISCOUNT FACTOR	$[0.9, 0.95, 0.99, 0.999]$
ACTOR ENTROPY	$[0.003, 0.001, 0.0003, 0.0001, 0.00003]$
KL LOSS SCALE	$[0.1, 0.3, 1, 3]$
REWARD LOSS SCALE	$[0.5, 1, 2]$
DISCOUNT LOSS SCALE	$[0.5, 1, 2]$
KL BALANCE	$[0.5, 0.8, 1, 2]$
DISCOUNT λ	$[0.8, 0.9, 0.95]$

F. Cross-Attention Network Ablation

In this section we present an ablation study on the object-centric PPO-CA agent in Table 5. We found vanilla versions of methods based on cross-attention (SlotAttention (Locatello et al., 2020) and Perceiver (Jaegle et al., 2021b)) not to work out of the box. For example, these methods typically employ LayerNorm and residual MLPs, but in Table 5 we can see that the variant using these "CA + Residual MLP + LayerNorm" underperforms. Also any of these two elements included individually ("CA + Residual MLP" and "CA + LayerNorm") did not improve performance. Moreover, introducing competition over slots (via a softmax over queries) "CA + Slot Competition" (akin to SlotAttention) also hurt the downstream performance. This analysis led us to converge to an architecture that uses a single cross-attention over the input image, with no latent self-attention, no layer normalization, no residual connections and (unlike SlotAttention) no competition between the slots via softmax over queries.

We also observed that there is a sweet spot of number of heads and number of slots where both have value 8. Although we cannot say what the direct relationship between these network parameters and the final RL performance is, we can speculate that the higher number of slots lets each of them to specialize in parts (objects) of the input image, and higher number of heads allows for operation specialization for each of the heads. Finally, we observe that larger patch size (with the appropriate stride) improves performance. The learned attention over the whole input image "CA, Patch Size=1, Stride=1" does not perform as well as attention with larger patches. Although in theory more powerful as it can attend to varying object sizes, this variant needs first to *learn* what parts of the image belong together into a single object. In its current form, Crafter objects are all of equal size, so the patches can correspond to these. We suspect attention over the whole image (patch size of 1) would be beneficial if we would have objects with varying sizes.

Table 5: Cross-attention (CA) Network Ablation.

Variant	Crafter Score
CA	**10.0 ± 0.4**
CA + Residual MLP	7.3 ± 0.4
CA + LayerNorm	4.1 ± 0.3
CA + Residual MLP + LayerNorm	3.1 ± 0.6
CA + Slot Competition	7.1 ± 0.3
CA, Number of Slots=1	8.2 ± 0.9
CA, Number of Slots=2	7.8 ± 0.4
CA, Number of Slots=4	8.7 ± 0.7
CA, Number of Slots=8	**10.0 ± 0.4**
CA, Number of Slots=16	9.1 ± 0.9
CA, Number of Heads=1	6.6 ± 0.5
CA, Number of Heads=2	7.3 ± 0.9
CA, Number of Heads=4	8.0 ± 0.6
CA, Number of Heads=8	**10.0 ± 0.4**
CA, Number of Heads=16	7.2 ± 0.8
CA, Patch Size=1, Stride=1	6.9 ± 0.9
CA, Patch Size=8, Stride=8	7.2 ± 0.4
CA, Patch Size=12, Stride=8	8.7 ± 0.7
CA, Patch Size=12, Stride=12	8.9 ± 0.7
CA, Patch Size=16, Stride=8	6.2 ± 0.5
CA, Patch Size=16, Stride=16	**10.0 ± 0.4**

G. Network Configurations

Table 6: PPO-CNN: baseline agent with the CNN from DQN (Mnih et al., 2015).

Feature Extractor
8×8 conv, 32 ReLU units, stride 4
4×4 conv, 64 ReLU units, stride 2
3×3 conv, 64 ReLU units, stride 1
Flatten
Linear, 512 ReLU units.
Action Network
Linear, 17 units.
Value Network
Linear, 1 units.

Table 7: PPO-SPCNN: agent with the size-preserving CNN.

Feature Extractor
5×5 conv, 64 ReLU units, stride 1, padding 2
5×5 conv, 64 ReLU units, stride 1, padding 2
5×5 conv, 64 ReLU units, stride 1, padding 2
5×5 conv, 64 ReLU units, stride 1, padding 2
Flatten
Linear, 512 ReLU units.
Action Network
Linear, 17 units.
Value Network
Linear, 1 units.

Table 8: PPO-SA: agent with the self-attention module.

Feature Extractor
5×5 conv, 64 ReLU units, stride 1, padding 2
5×5 conv, 64 ReLU units, stride 1, padding 2
5×5 conv, 64 ReLU units, stride 1, padding 2
5×5 conv, 64 ReLU units, stride 1, padding 2
Split into 8×8 patches and flatten the patch grid
Self-Attention Network
Learned CLS token of 256 size.
Slot-wise projection: Linear, 256 units.
Query map: Linear, 256 units.
Key map: Linear, 256 units.
Values map: Linear, 256 units.
Action Network
Linear, 17 units.
Value Network
Linear, 1 units.

Table 9: PPO-CA: agent with the cross-attention module.

Feature Extractor
5×5 conv, 64 ReLU units, stride 1, padding 2
5×5 conv, 64 ReLU units, stride 1, padding 2
5×5 conv, 64 ReLU units, stride 1, padding 2
5×5 conv, 64 ReLU units, stride 1, padding 2
Split into 16×16 patches and flatten the patch grid
Self-Attention Network
Learned slots 8×256 size.
Query map: Linear, 256 units.
Key map: Linear, 256 units.
Values map: Linear, 256 units.
Action Network
Linear, 17 units.
Value Network
Linear, 1 units.

Table 10: RecPPO-CNN: recurrent agent with an LSTM and CNN from DQN (Mnih et al., 2015).

Feature Extractor
8×8 conv, 32 ReLU units, stride 4
4×4 conv, 64 ReLU units, stride 2
3×3 conv, 64 ReLU units, stride 1
Flatten
Linear, 512 ReLU units.
Action Network
LSTM, 256 units.
Linear, 17 units.
Value Network
LSTM, 256 units.
Linear, 1 units.

Table 11: RecPPO-SPCNN: recurrent agent with an LSTM and size-preserving CNN.

Feature Extractor
5×5 conv, 64 ReLU units, stride 1, padding 2
5×5 conv, 64 ReLU units, stride 1, padding 2
5×5 conv, 64 ReLU units, stride 1, padding 2
5×5 conv, 64 ReLU units, stride 1, padding 2
Flatten
Linear, 512 ReLU units.
Action Network
LSTM, 256 units.
Linear, 17 units.
Value Network
Linear, 256 units.
Linear, 1 units.