Craftax: A Lightning-Fast Benchmark for Open-Ended Reinforcement Learning

Michael Matthews [1] **Michael Beukman** [1] **Benjamin Ellis** [1] **Mikayel Samvelyan** [2] **Matthew Jackson** [1]
Samuel Coward [1] **Jakob Foerster** [1]

Abstract

Benchmarks play a crucial role in the development and analysis of reinforcement learning (RL) algorithms. We identify that existing benchmarks used for research into open-ended learning fall into one of two categories. Either they are too slow for meaningful research to be performed without enormous computational resources, like Crafter, NetHack and Minecraft, or they are not complex enough to pose a significant challenge, like Minigrid and Procgen. To remedy this, we first present *Craftax-Classic*: a ground-up rewrite of Crafter in JAX that runs up to 250x faster than the Python-native original. A run of PPO using 1 billion environment interactions finishes in under an hour using only a single GPU and averages 90% of the optimal reward. To provide a more compelling challenge we present the main *Craftax* benchmark, a significant extension of the Crafter mechanics with elements inspired from NetHack[1]. Solving Craftax requires deep exploration, long term planning and memory, as well as continual adaptation to novel situations as more of the world is discovered. We show that existing methods including global and episodic exploration, as well as unsupervised environment design fail to make material progress on the benchmark. We believe that Craftax can for the first time allow researchers to experiment in a complex, open-ended environment with limited computational resources.

1. Introduction

Progress in reinforcement learning (RL) algorithms is driven in large part by the development and adoption of suitable benchmarks. Examples include the Arcade Learning Environment (Bellemare et al., 2013) for value based deep RL algorithms, Mujoco (Todorov et al., 2012) for continuous control and the StarCraft Multi-Agent Challenge (Samvelyan et al., 2019) for multi-agent RL. In the effort towards increasingly general agents, there has arisen a community focused on benchmarks that exhibit more open-ended dynamics (Stanley et al., 2017), in the form of procedural world generation, skill acquisition and reuse, long term dependencies and continual learning. This has motivated the development of environments like MALMO (Johnson et al., 2016) (Minecraft), The NetHack Learning Environment (Küttler et al., 2020), MiniHack (Samvelyan et al., 2021) and Crafter (Hafner, 2021). However, slow runtime has rendered them inaccessible to current methods without large-scale computational resources, limiting their practicality to the research community.

Figure 1. Pixel-based view from `Craftax`.

Simultaneously, as the speed of running an end-to-end compiled RL pipeline has become fully appreciated (Lu et al., 2022), there has been an explosion of RL environments implemented in JAX (Freeman et al., 2021; Lange, 2022; Bonnet et al., 2023; Rutherford et al., 2023; Nikulin et al.,

[1]University of Oxford [2]University College London. Correspondence to: Michael Matthews <michael.matthews@eng.ox.ac.uk>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

[1]Code provided at https://github.com/MichaelTMatthews/Craftax.

2023; Koyamada et al., 2023). The elimination of CPU-GPU transfer combined with efficient parallelisation and compilation means that experiments that would previously have taken a large compute cluster days to run can finish in minutes on a single GPU (Lu et al., 2022).

To bring these two paradigms together, we present the Craftax benchmark: a JAX-based environment exhibiting complex, open-ended dynamics and running orders of magnitude faster than comparable environments (Figure 2). Concretely, we first propose `Craftax-Classic`, a reimplementation of Crafter (Hafner, 2021) in JAX that runs 250 times faster than the Python-native original. We then go on to show that, with easy access to drastically more timesteps, `Craftax-Classic` is solved (to within 90% of maximum return) by a simple PPO (Schulman et al., 2017) agent in 51 minutes (see Appendix F.3 for further details).

We therefore also present `Craftax`, a significantly more challenging environment that incorporates dynamics inspired from NetHack (Küttler et al., 2020) and more broadly the Roguelike[2] genre as a whole. Our experiments show that existing methods make little progress on `Craftax`. We therefore hope that it presents a meaningful challenge for future RL research, while allowing experimentation with limited computational resources.

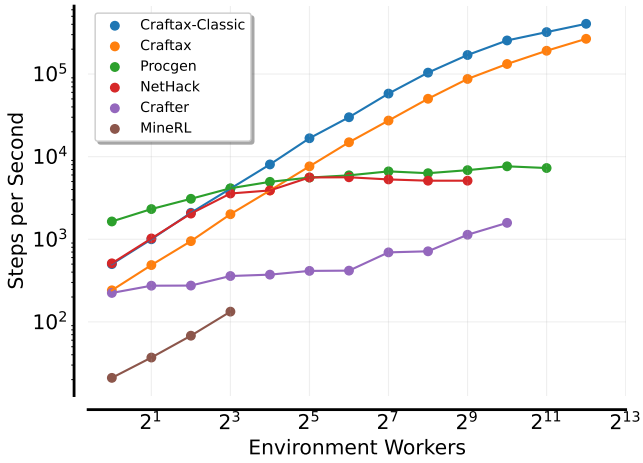


Figure 2. Speed comparison with popular benchmarks for open-ended learning. `Craftax-Classic` and `Craftax` are 257x and 169x faster than Crafter respectively. Details of the speed test are in Appendix B and best case results are in Table 1.

[2] `https://en.wikipedia.org/wiki/Roguelike` defines Roguelikes as a class of game characterised by "procedurally generated levels, turn-based gameplay, grid-based movement and permanent death of the player character"

2. Background

2.1. JAX-Based Environments

JAX (Bradbury et al., 2018) is a library that allows for code written in Python to be compiled down to Accelerated Linear Algebra[3] and run on hardware accelerators like GPUs and TPUs. While deep RL training has traditionally been split between collecting trajectories on CPU-based environments and then training policy and value networks on the GPU, the relatively new phenomenon of JAX-based environments allows for the whole RL pipeline to be run on the GPU. This allows for massive parallelisation of trajectory gathering (we use up to 4096 parallel environment workers), the elimination of the GPU-CPU transfer bottleneck and just-in-time (JIT) compilation of the whole training process.

2.2. Crafter

Crafter (Hafner, 2021) is a top-down, procedurally generated, open-world survival game that works like a simplified, 2D Minecraft. The player finds themselves in a world consisting of grassland, forests, lakes and mountains and must survive by gathering materials to craft tools with, maintaining hunger, thirst and energy levels, and fighting enemies.

The Crafter RL environment uses pixel-based observations and has a discrete action space made up of 4 movement directions and 13 interactions (e.g. `MAKE_STONE_PICKAXE`, `SLEEP`, `PLACE_FURNACE`). It has a set of 22 unique achievements (e.g. `DEFEAT_ZOMBIE`, `EAT_PLANT`, `COLLECT_DIAMOND`), each giving the agent a $+1$ reward the first time it is achieved in an episode.

Crafter was proposed to evaluate "strong generalization, deep exploration, and long-term reasoning" (Hafner, 2021).

While Crafter has become a popular benchmark, the evaluation protocol proposed allocates algorithms only 1 million environment interactions, a very limiting constraint when compared to other RL benchmarks. For instance, the standard protocol for the Atari benchmark (a set of environments that require limited generalisation (Cobbe et al., 2020)) is to use 200M environment interactions per game (Machado et al., 2018), while The NetHack Learning Environment suggests using 1 billion (Küttler et al., 2020). In practice, this has made Crafter a benchmark largely focused on testing sample efficiency (similar to other benchmarks like Atari 100k (Bellemare et al., 2013)).

While we reuse many of the Crafter dynamics, our aim is to provide a benchmark for investigations into open-endedness rather than sample efficiency. Open-endedness, by its very definition, should not be constrained by a fixed number of

[3] `https://www.tensorflow.org/xla`

samples. In practice we have to impose some limit, but this should be suitably high as to not impact the emergence of interesting phenomena.

2.3. The NetHack Learning Environment

NetHack is an infamously difficult video game, adapted into an RL benchmark through The NetHack Learning Environment (NLE) (Küttler et al., 2020) and MiniHack (Samvelyan et al., 2021). The game involves the player descending through around fifty floors of dungeons, collecting items, fighting enemies and gaining experience before retrieving an amulet and finally beating the game by 'ascending'. The game is known to be incredibly hard, with it typically taking years of practice to achieve an ascension (Küttler et al., 2020). Current algorithmic approaches (both rule-based and deep RL) fail to complete the game (Hambro et al., 2022). While NLE executes NetHack natively in C++ and is faster than Python-based Crafter, it is still significantly slower than GPU based environments (Figure 2).

2.4. Open-Ended Learning

Open-Ended Learning (Stanley et al., 2017) refers to a broad category of problem domains and algorithms that focus on learning in a perpetual and unguided manner. We investigate two subfields that are particularly relevant to `Craftax`.

Exploration through Intrinsic Rewards Exploration is a key part of RL performance, especially in environments with sparse or deceptive rewards, for which simple methods like max-entropy RL (Haarnoja et al., 2018) or ϵ-greedy policies are insufficient. One method to overcome these difficulties is through the idea of an *intrinsic reward* (Schmidhuber, 1991; Oudeyer et al., 2007; Barto, 2013; Bellemare et al., 2016), in addition to the extrinsic reward received from the environment. This signal should intuitively reward the agent for visiting states or performing actions that satisfy some novelty-like heuristic (Lehman & Stanley, 2008), causing the agent to actively explore the environment by pushing back the frontiers of its knowledge.

Unsupervised Environment Design Unsupervised environment design (UED) is a paradigm in RL where an *adversary* proposes environments configurations (referred to as *levels*) for an agent to train on (Dennis et al., 2020; Jiang et al., 2021a; Parker-Holder et al., 2022). The adversary is rewarded for choosing levels that maximise the agent's *regret*, defined as the difference in return between the current and optimal agent. This has been empirically shown to automatically induce a curriculum of progressively harder levels that aid the performance and generalisation properties of the learned agent. Different UED algorithms require different levels of access to the underlying environment state, ranging from simply being able to repeat seeds (Jiang et al., 2021a) to directly editing the levels (Parker-Holder et al., 2022). Due to the functional nature of `Craftax` necessitated by JAX, the entire environment state is exposed as a single object, making UED methods easy to apply.

3. Craftax

3.1. Craftax-Classic: A Reimplementation of Crafter in JAX

We first present `Craftax-Classic`, a JAX-based reimplementation of the Crafter benchmark. The aim of `Craftax-Classic` is to mimic the Crafter dynamics as closely as possible, while allowing orders of magnitude faster RL training (Figure 2). While most of the dynamics of Crafter were remade exactly in line with the original, some subtle changes were made for performance reasons (See Appendix A for a discussion of all the differences). Our hope is that `Craftax-Classic` provides an easy introduction to `Craftax` for those already familiar with the Crafter benchmark.

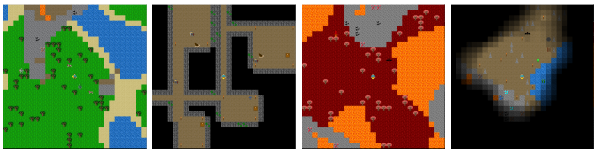
While Crafter uses pixel-based observations, many of the aspects that Crafter tests (exploration, memory) are indifferent to the exact nature of the observation and the use of pixels simply adds an extra layer of representation learning to the problem. For this reason, we provide versions of `Craftax` with both symbolic and pixel-based observations, with the former running around 10x faster than the latter.

3.2. Craftax: An Extension of Crafter with NetHack-Like Mechanics in JAX

As shown in Appendix F.3, `Craftax-Classic` is convincingly solved by a PPO agent running for under an hour on a single GPU. To provide a more compelling challenge we present the main `Craftax` environment, designed to be significantly harder, while retaining a fast runtime. `Craftax` adds a large and diverse range of new game mechanics. This section provides a brief overview, with Appendix D containing more details.

Multiple Floors While in Crafter the player is confined to a single 64x64 grid, `Craftax` contains 9 unique procedurally generated floors, including caves, dungeons, fire and ice floors and a final boss floor. The player can descend and ascend through the world by finding the ladders that connect adjacent floors. Each floor contains distinct challenges in the forms of different terrain generation, enemies and required skills, necessitating deep exploration and generalisation. While each floor is unique, many game mechanics are shared between them and, on a meta-level, exploration strategies that worked on earlier floors (for instance moving adjacent to a block and trying different actions to figure out its characteristics) will also work on later floors. In this way we hope to not only facilitate generalisation across different procedurally generated worlds but also generalisation of the

exploration strategy through time over the learning process.



Combat We overhaul the combat system with damage and defence split into physical, fire and ice categories. As well as being able to craft higher-tier weapons, the player can use a bow (found in a random chest on the first dungeon floor) for ranged attacks, and read books (also found in dungeon chests) to learn offensive spells to cast. The player can also craft armour to protect themselves from damage (essential when descending to the lower floors). As well as increasing the general complexity of the environment, the diversity in combat furthers the in-context learning element provided by the procedural level generation — an agent that stumbles upon a strong weapon or armour piece should suitably change its strategy. This further extends the exploration problem as, by design, there should not be one fixed strategy (for instance, always putting experience points into strength and defeating enemies with melee attacks) that works on every level, meaning that an agent will have to explore a diverse range of strategies to achieve consistently high return.

New Creatures The 3 creatures present in Crafter (cow, zombie and skeleton) have been increased to 19 different creatures, each with unique effects, behaviours and techniques required to defeat them. For instance, the Fire Elementals present on floor 7 either require an appropriate enchantment on the player's sword or magic to defeat, while beating the aquatic enemies on floor 6 typically requires ranged weaponry. This diversity of creatures further enhances the exploration problem, as no two are the same and strategies for dealing with one may entirely fail for others.

Potions and Enchantments The player can find potions of varying colours spread over the 9 floors; however the effects of these potions are randomly permuted every episode. This means that an agent will need to discover which potions correspond to which effects through trial and error each episode, further testing in-context learning and memory. Also, by spending mana (magical energy) and sacrificing one of the new gemstones, the player can enchant weapons and armour. This provides them with increased damage or protection against a certain element.

Attributes The player is rewarded with an experience point every time they descend to a new floor. These points can be spent to increase the players dexterity, strength or intelligence. Dexterity makes the player more capable with a bow, as well as allowing them to go longer between eating, drinking and sleeping. Strength increases the players health and melee damage, while intelligence increases the players magical powers. The assignment of experience points can drastically alter the players abilities and have a large influence on the strategy an agent should use, testing the agents long term reasoning capabilities, as successful runs typically take tens of thousands of frames.

Boss Floor The final floor contains a very challenging boss fight, with the player having to defeat waves of every existing enemy in the game. As well as serving as a final challenge, the boss fight tests how the agent can transfer its prior knowledge of fighting these enemies in different contexts to the unique structure of the boss floor.

Difficulty `Craftax` strikes a balance of difficulty, being significantly more challenging than Crafter, but not as hard as games like NetHack. For perspective, it took one of the authors (with extensive knowledge of the game mechanics) roughly 5 hours of gameplay to first achieve a 'perfect' run where every achievement was completed. This was playing in a GUI that allowed for unlimited time to pause and think before taking each action. We think that `Craftax` is hard enough to present a significant challenge beyond the capabilities of existing RL methods, while being ultimately achievable without domain knowledge.

3.3. RL Environment Interface

`Craftax` conforms to the Gymnax (Lange, 2022) wrapper for easy integration with existing frameworks.

Observation Space `Craftax` provides options for both pixel-based and symbolic observations. The pixel based observation is a downscaled $63 \times 63 \times 3$ image for `Craftax-Classic` (in line with the original Crafter) and $110 \times 130 \times 3$ for `Craftax`. The symbolic observations uses a one-hot encoding to represent both block types and creatures in the player's visual area, which is then appended to an array containing the rest of the players information (inventory, health, hunger, attributes, etc.), resulting in a flat observation space of 1345 for `Craftax-Classic` and 8268 for `Craftax`. A full description of the observation space is given in Appendix D.4.

Action Space `Craftax` uses a discrete action space of

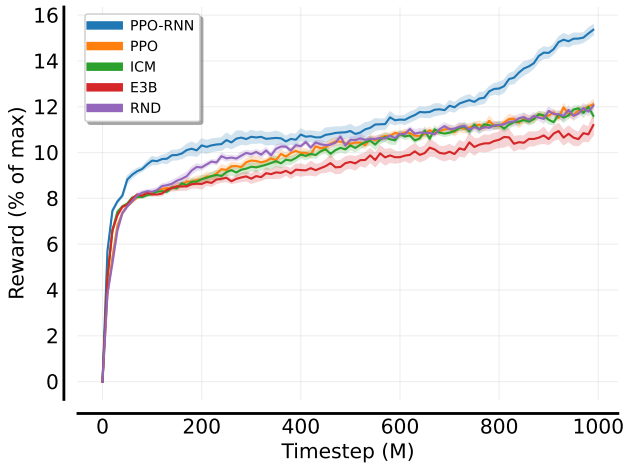


Figure 3. Rewards on Craftax-1B for PPO, PPO-RNN, ICM, E3B and RND. Each algorithm is run for 1 billion timesteps with 10 seeds. The shaded area denotes 1 standard error.

size 17 and 43 for `Craftax-Classic` and `Craftax` respectively. As in Crafter, every action can be taken at any timestep, so attempting an action without its specific prerequisites will effectively cause the agent to execute a no-op action, stepping the environment forward one timestep. For an exhaustive list of actions see Appendix D.5.

Reward We follow a similar reward structure to Crafter, defining a set of achievements and giving the agent a reward the first time each achievement is completed each episode. There are 22 achievements in `Craftax-Classic` and 65 in `Craftax`. We also penalise the agent by 0.1 for every point of damage taken and reward it by 0.1 for every point recovered. Early experiments on `Craftax` showed that providing a flat +1 reward for each achievement, did not provide enough incentive for the agent to start exploring the dungeons, as the agent would usually be killed quickly upon descending and learn to avoid them. To encourage progress towards harder tasks we grouped achievements into four categories: 'Basic', 'Intermediate', 'Advanced' and 'Very Advanced', each worth 1, 3, 5 and 8 reward respectively. An exhaustive listing and description of all the achievements is located in Appendix D.6.

3.4. Evaluation Framework

We propose two separate benchmarks to evaluate performance on `Craftax` (we leave `Craftax-Classic` mostly as a tool to help researchers iterate quickly on an environment very similar to the original Crafter). We also limit the benchmarks to considering symbolic observations, as these environments are significantly faster to run, and the pixel-based environments do not contribute anything significant to the core research directions we believe `Craftax` is useful for investigating.

First, we propose the **Craftax-1B Challenge**, where a budget of 1 billion environment interactions is permitted on the `Craftax-Symbolic` environment. This challenge is meant to provide a chance for algorithms focusing on exploration, continual learning, and long-term planning and reasoning to differentiate themselves on a hard benchmark, without being overly constrained by sample efficiency concerns. We set the benchmark at 1 billion steps to keep it manageable for researchers with limited computational resources (a run on a single RTX 4090 can finish in an hour), while providing enough timesteps for a substantial amount of meaningful exploration of the environment. Averaging around 5 steps per second (roughly the rate the authors moved at when playing), 1 billion timesteps corresponds to over 6 years of continual human gameplay. We use reward as the metric to compare agents.

Secondly, we propose the **Craftax-1M Challenge**, where a budget of 1 million environment interactions is permitted on the `Craftax-Symbolic` environment. This benchmark is meant as a test of sample efficiency, with the benefit that experiments can take only seconds to finish. We believe the main value of this benchmark is in the almost instantaneous results researchers can obtain, allowing for iteration of methods at an unprecedented speed. It should be noted that even with such a tight environment interaction budget, we found superior performance with massive parallelism of workers (see Table 11 in Appendix E).

4. Experiments

4.1. Exploration Baselines

We use the PPO implementations from PureJaxRL (Lu et al., 2022) as the foundation for our baselines, allowing us to run fully compiled end-to-end RL pipelines. We flatten the symbolic observations and use a 4 layer MLP of width 512 for both policy and value networks. We experimented initially with using convolutions for the map view part of the observation, but consistently found that fully connected networks performed better.

We run PPO with an MLP as well as PPO with memory (using a Gated Recurrent Unit (Chung et al., 2014)), which we refer to as PPO-RNN. We also experiment using two state of the art intrinsic rewards for additional exploration:

Random Network Distillation We use the popular exploration baseline Random Network Distillation (RND) (Burda et al., 2018) as our first method for global intrinsic reward. RND works by training a network that is given observations to match a network of the same architecture that was randomly initialised. Intuitively, the error of this network will be high on unseen observations, as the network will not yet

have been trained to match the random output. This error is then used as an additional intrinsic reward that incentivises the agent to explore states with novel observations.

Intrinsic Curiosity Module Our second baseline is the Intrinsic Curiosity Module (ICM) (Sukhbaatar et al., 2018). ICM learns a world model and provides intrinsic reward proportional to the error of this world model, incentivising the agent to visit areas of the environment which are badly modelled and therefore probably less visited. The latent space of this world model is constructed in such a way as to ignore aleatoric uncertainty in the environment, in order to focus exploration and avoid the so called 'Noisy TV problem' (Burda et al., 2018).

Exploration via Elliptical Episodic Bonuses Our third and final exploration baseline is the E3B algorithm (Henaff et al., 2022). Rather than considering exploration over the entire training process, E3B formulates its intrinsic reward to encourage exploration within each episode. An ellipse is fitted around the agents current trajectory in latent space and it is rewarded for moving far away from this ellipse. This approach has been shown to outperform global exploration methods, especially in non-singleton (i.e. procedurally generated) environments (Henaff et al., 2022; 2023).

4.2. UED Baselines

We assess PLR and ACCEL and compare them to Domain Randomisation (Jakobi, 1997; Tobin et al., 2017, DR), which corresponds to uniformly sampling randomly generated levels, as per usual. We use JaxUED (Coward et al., 2024) as the base implementation for all our UED experiments.

PLR Prioritised level replay (PLR) (Jiang et al., 2021a;b) is a curation-based UED method which maintains a buffer of training levels. At every step, one of the two options is randomly chosen: (a) the agent is evaluated on randomly generated levels, and those with high regret are added to the buffer; (b) the agent is trained on a sample of high-regret levels from the buffer. PLR performs gradient updates on the random levels, whereas robust PLR (denoted as $\text{PLR}^{\perp}$) only updates the agent on the levels sampled from the buffer. This is theoretically justified and (in some domains) empirically results in higher performance (Jiang et al., 2021a).

ACCEL ACCEL (Parker-Holder et al., 2022) is similar to PLR, but also considers randomly mutated levels from the buffer as candidate levels. ACCEL therefore increases the complexity of levels over time via iterative editing instead of having to rely solely on random generation and curation.

In this section, we provide results for the two proposed benchmarks using the previously discussed algorithms. We use JaxUED (Coward et al., 2024) for all UED experiments, and hyperparameter tuning is detailed in Appendix E.

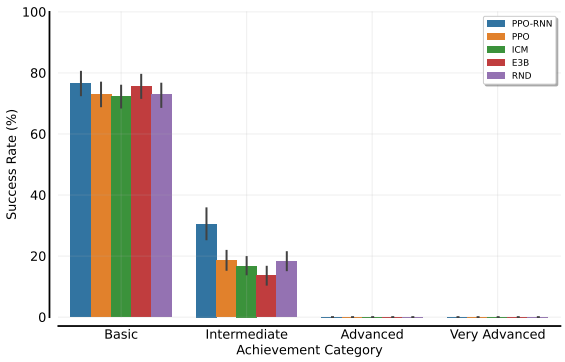


Figure 4. Achievement success rate on Craftax-1B split by achievement difficulty. Each algorithm is run on 10 seeds, with error bars showing 1 standard error.

4.3. Craftax-1B

The returns for the evaluated algorithms on Craftax-1B are summarised in Figure 3 with the final achievement yields split by difficulty shown in Figure 4. We report reward as a percentage of the maximum achievable reward (obtained by completing every achievement) of 226 for readability. Fine-grained achievement results are shown in Appendix F.1 with a selected set of achievements highlighted in Figure 5. We see that PPO and PPO-RNN both quickly master most of the basic achievements, with memory helping to push progress further into the intermediate category. Interestingly, none of the tested exploration methods improved performance and E3B in fact significantly reduced the reward. We hypothesise this is because the reward structure is dense enough for RL methods without intrinsic reward to learn on, with the intrinsic reward potentially serving to distract the agent. None of the agents make any progress in the harder achievement categories.

Taking a closer look at the achievements highlighted in Figure 5 we first see that all methods quickly and robustly learn basic achievements such as `MAKE_WOOD_PICKAXE`. While all methods enter the dungeon (the first floor beneath the overworld) an appreciable amount, we see that PPO-RNN does so significantly more than the others, giving it more access to the high-reward achievements and driving its strong performance. Interestingly, the success rate on some simple achievements like `DEFEAT_ZOMBIE` decrease over time, with PPO-RNN doing the worst of all. This is explained by the fact that zombies only appear in the overworld, so the stronger agents are trading off low-reward achievements in the overworld for high-reward ones underground. We see that no appreciable progress is made into the second floor (the gnomish mines), with only very rare

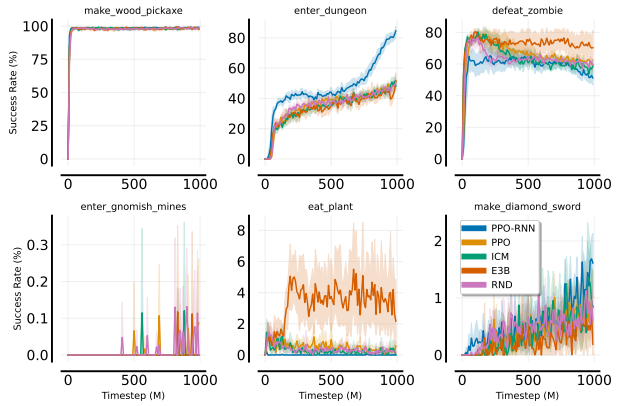


Figure 5. Achievement success rate on Craftax-1B for selected achievements over 10 seeds, with error bars denoting 1 standard error.

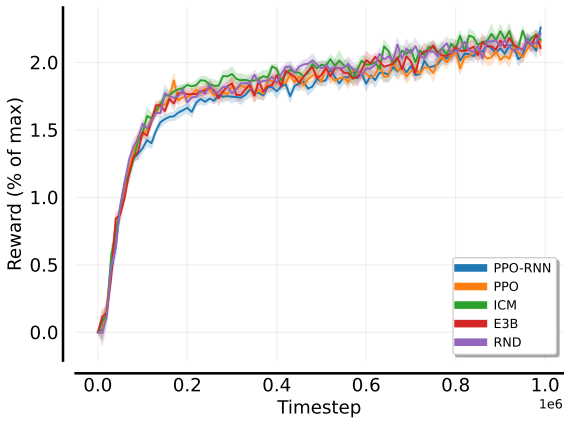


Figure 6. Rewards on Craftax-1M for PPO, PPO-RNN, ICM, E3B and RND. Each algorithm is run for 1 million timesteps with 10 seeds. The shaded area denotes 1 standard error.

approaches being made into it. Interestingly, we also see that the `EAT_PLANT` (notable for being perhaps the hardest achievement in the original Crafter) is entirely ignored by PPO-RNN and is actually best achieved by E3B, indicating that while the intrinsic reward may not be helping with overall return, it does incentivise the agent to explore different parts of the environment.

Although it looks like PPO-RNN in particular is still learning, we find that running for 10 billion timesteps barely improves performance (Appendix F.2), indicating that this is likely an artefact of learning rate decay (You et al., 2019).

4.4. Craftax-1M

The results for Craftax-1M are shown in Figure 6. Compared to Craftax-1B there is much less separation between the algorithms with all of them performing relatively similarly. Fine grained achievements are shown in Appendix F.1, showing that progress is made only on the very basic achievements (mostly corresponding to those carried over from Crafter), with agents entering the dungeons only on individual occurrences.

4.5. Craftax-1B: UED

We now investigate applying DR, PLR, Robust PLR and ACCEL to Craftax-1B. For ACCEL, we evaluate three different mutation operators applied to the overworld:

Noise Each level θ corresponds to the angle vectors that the perlin noise algorithm uses to generate worlds. Mutation adds a uniform random number to each element.

Swap We randomly swap two different tiles. To increase relevance, one of these is within the 16×16 central area of the level—close to where the agent starts.

Restricted Swap (RSwap) To obtain more coherent levels, we restrict the tiles that can be swapped. For instance, stone can be swapped with any ore tile, and grass can be swapped with trees, leading to more realistic and coherent levels.

Results After training for 1B timesteps, we evaluate the saved checkpoints on a fixed set of 20 evaluation levels, which were generated normally. We plot the mean and standard error over 10 independent runs.

The primary results are presented in Figure 7, and success rates on all achievements are provided in Figure 34. We find that PLR performs best, followed by DR, PLR$^{\perp}$, restricted swap and noise ACCEL, which all perform roughly the same. Unrestricted swap ACCEL performs the worst.

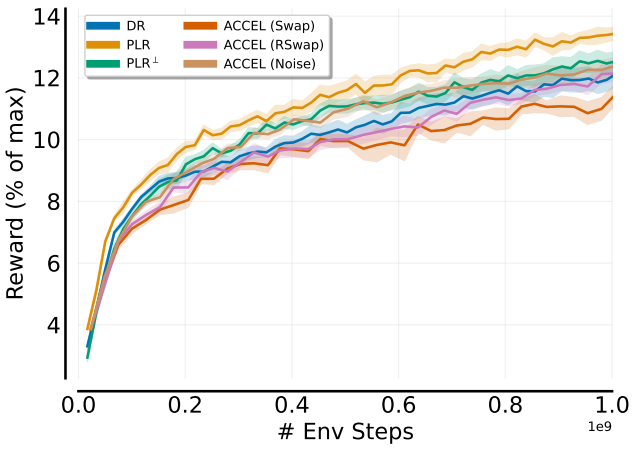


Figure 7. Evaluation reward for various UED methods on Craftax-1B on 10 seeds, with the shaded area denoting 1 standard error.

PLR vs Robust PLR We find that Robust PLR performs worse than vanilla PLR, and only slightly outperforms

DR, despite previous evidence to the contrary (Jiang et al., 2021a). One reason could be that, unlike the gridworld mazes it has previously been tested on, `Craftax` consistently generates high-quality and solvable levels. Therefore, the agent is not penalised by training on these DR levels. Since we compared each algorithm when using the same number of environment interactions, robust PLR performs fewer gradient updates than DR or PLR. This result shows how important it is to expand the pool of environments UED is tested on for thorough analysis of the methods.

Distribution Shift Since UED alters the training distribution (a phenomenon named curriculum-induced covariate shift (CICS) by Jiang et al. (2022)), we make the distinction between DR levels sampled from the normal generator and *replay* levels sampled from the shifted UED distribution.

We find that ACCEL-based methods perform differently in replay levels compared to DR. For instance, the unrestricted swap method collect diamonds more than 30% of the time during replay, compared to 7% on DR levels (see Figures 32 and 33). In addition, restricted swap ACCEL reaches the gnomish mines 1% of the time in replay levels, compared to almost never in DR levels.

The average return (shown in Figure 8) is also different between replay and DR levels. The two swapping-based ACCEL methods have significantly higher returns during replay. This may indicate that the generated levels are easier (e.g., with more resources near the start), and the agent is able to perform more reward-generating activities. The other methods have similar episode lengths during both phases.

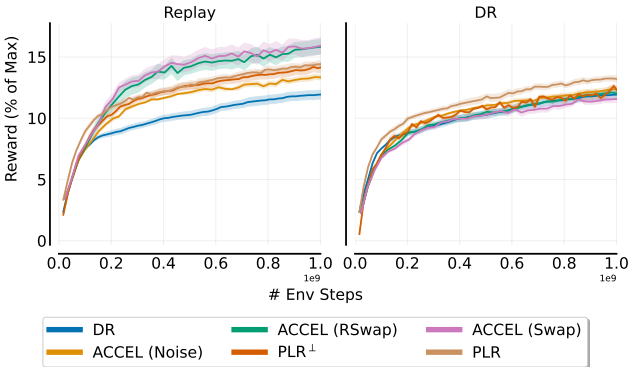


Figure 8. Average episode length for replay and DR levels on Craftax-1B over 10 seeds, with the shaded area denoting 1 standard error.

UED Results Analysis We find that DR performs competitively to the other UED methods, with only normal PLR significantly outperforming it. We believe this is partly due to `Craftax` evaluating on in distribution levels, rather than a set of hold-out levels, where UED generally performs better (Jiang et al., 2021a; Parker-Holder et al., 2022).

5. Related Work

5.1. Environments for Open-Endedness

Exploration The most widely known hard-exploration environments are those from the Atari benchmark with sparse rewards, most famously Montezuma's Revenge and Pitfall (Bellemare et al., 2013). While these environments once posed a significant challenge to the community, they have been solved by methods like Go-Explore (Ecoffet et al., 2021), which exhaustively explore the entire game tree using state resets or goal-conditioned policies. This technique is only possible due to these being singleton environments with mostly deterministic mechanics. As was similarly argued in Küttler et al. (2020), we believe that these environments are therefore no longer suitable for furthering research into exploration and that focus should instead be applied to more complex environments with randomised levels and stochastic mechanics.

Many recent environments have subscribed to this view and introduced elements like procedural generation and more complex world mechanics. Procgen (Cobbe et al., 2020) addresses the singleton and determinism problems seen in the Arcade Learning Environment. It randomly samples an initial 'context' at the start of each episode, corresponding to concepts like obstacle placement or maze layouts.

This idea is taken even further with environments like Minecraft (Johnson et al., 2016) which generates entirely unique (and effectively infinite) worlds each episode. As discussed in Section 2, both Crafter (Hafner, 2021) and NetHack (Küttler et al., 2020) make use of procedural generation to produce finite worlds each episode. MiniHack (Samvelyan et al., 2021) provides a set of small levels that make use of the NetHack mechanics, but are less complicated than attempting to solve the entire NetHack game. XLand (Open Ended Learning Team et al., 2021) can represent a huge array of 3D games with different goals and emergent strategies.

UED The most commonly used environments for UED are MiniHack (Samvelyan et al., 2021) and Minigrid (Chevalier-Boisvert et al., 2023), BipedalWalker (Wang et al., 2019) and Car Racing (Jiang et al., 2021a). All of these environments are relatively simple and while they were vital for getting the field off the ground, for UED to be used in more advanced settings it needs to be shown to generalise to more complex environments, for which we believe `Craftax` is a suitable candidate.

5.2. JAX-Based Environments

Brax (Freeman et al., 2021) arguably kickstarted the development of JAX-based environments by reimplementing MuJoCo-like mechanics (Todorov et al., 2012) and showing that training with large numbers of environment workers

could drastically reduce wall clock time without unduly affecting performance. Gymnax (Lange, 2022) implemented classic control (Brockman et al., 2016), BSuite (Osband et al., 2020) and MinAtar environments (Young & Tian, 2019) among others. JaxMARL (Rutherford et al., 2023) remade many popular environments for multi-agent RL like StarCraft (Samvelyan et al., 2019; Ellis et al., 2024) and Hanabi (Bard et al., 2020). Pgx (Koyamada et al., 2023) provides implementations of classic board games like Chess and Go. Jumanji (Bonnet et al., 2023) recreated many combinatorial problems like Sudoku and graph colouring.

The most similar to our own work is that of XLand-Minigrid (Nikulin et al., 2023), which can represent a huge diversity of tasks in a fast gridworld-based environment. While this provides an incredible tool for open-ended research, we see `Craftax` as filling a different niche. Specifically, `Craftax` can be seen as providing a very difficult exploration task in the form of a distribution of meaningfully similar levels. Whereas, XLand-Mingrid encompasses a very wide range of semantically unique levels of which each one is significantly simpler than an instance of `Craftax`. In other words, `Craftax` assesses deep exploration whereas XLand-Minigrid assesses wide generalisation.

6. Conclusion

We present `Craftax`, a blazing fast environment filled with complex mechanics that cannot be solved by existing RL algorithms. We hope that `Craftax` will facilitate research into areas including exploration, continual learning, generalisation, skill acquisition and long term reasoning. We believe an agent that could solve `Craftax` would represent a substantial step forward for the field and look forward to seeing what the community uses the benchmark to develop.

Acknowledgements

We would like to thank Alex Goldie and Sebastian Towers for their invaluable contributions to game design, our anonymous reviewers for their useful feedback and to the open-source software community who have already contributed numerous improvements to the codebase.

Impact Statement

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

References

Bard, N., Foerster, J. N., Chandar, S., Burch, N., Lanctot, M., Song, H. F., Parisotto, E., Dumoulin, V., Moitra, S., Hughes, E., et al. The hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, 280:103216, 2020.

Barto, A. G. Intrinsic motivation and reinforcement learning. *Intrinsically motivated learning in natural and artificial systems*, pp. 17–47, 2013.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

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, 2013.

Bonnet, C., Luo, D., Byrne, D., Surana, S., Coyette, V., Duckworth, P., Midgley, L. I., Kalloniatis, T., Abramowitz, S., Waters, C. N., Smit, A. P., Grinsztajn, N., Sob, U. A. M., Mahjoub, O., Tegegn, E., Mimouni, M. A., Boige, R., de Kock, R., Furelos-Blanco, D., Le, V., Pretorius, A., and Laterre, A. Jumanji: a diverse suite of scalable reinforcement learning environments in jax, 2023. URL https://arxiv.org/abs/2306.09884.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Burda, Y., Edwards, H., Storkey, A. J., and Klimov, O. Exploration by random network distillation. *CoRR*, abs/1810.12894, 2018. URL http://arxiv.org/abs/1810.12894.

Chevalier-Boisvert, M., Dai, B., Towers, M., de Lazcano, R., Willems, L., Lahlou, S., Pal, S., Castro, P. S., and Terry, J. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *CoRR*, abs/2306.13831, 2023.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

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.

Coward, S., Beukman, M., and Foerster, J. Jaxued: A simple and useable ued library in jax. *arXiv preprint*, 2024.

Dennis, M., Jaques, N., Vinitsky, E., Bayen, A. M., Russell, S., Critch, A., and Levine, S. Emergent complexity and zero-shot transfer via unsupervised environment design. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/985e9a46e10005356bbaf194249f6856-Abstract.html.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. First return, then explore. *Nature*, 590(7847): 580–586, 2021.

Ellis, B., Cook, J., Moalla, S., Samvelyan, M., Sun, M., Mahajan, A., Foerster, J., and Whiteson, S. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Freeman, C. D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., and Bachem, O. Brax – a differentiable physics engine for large scale rigid body simulation, 2021. URL https://arxiv.org/abs/2106.13281.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Hafner, D. Benchmarking the spectrum of agent capabilities. In *International Conference on Learning Representations*, 2021.

Hambro, E. et al. Insights from the neurips 2021 nethack challenge. In *NeurIPS 2021 Competitions and Demonstrations Track*, pp. 41–52. PMLR, 2022.

Henaff, M., Raileanu, R., Jiang, M., and Rocktäschel, T. Exploration via elliptical episodic bonuses. *Advances in Neural Information Processing Systems*, 35:37631–37646, 2022.

Henaff, M., Jiang, M., and Raileanu, R. A study of global and episodic bonuses for exploration in contextual mdps. *arXiv preprint arXiv:2306.03236*, 2023.

Jakobi, N. Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive behavior*, 6(2):325–368, 1997.

Jiang, M., Dennis, M., Parker-Holder, J., Foerster, J. N., Grefenstette, E., and Rocktäschel, T. Replay-guided adversarial environment design. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, pp. 1884–1897, 2021a. URL https://proceedings.neurips.cc/paper/2021/hash/0e915db6326b6fb6a3c56546980a8c93-Abstract.html.

Jiang, M., Grefenstette, E., and Rocktäschel, T. Prioritized level replay. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 4940–4950. PMLR, 2021b. URL http://proceedings.mlr.press/v139/jiang21b.html.

Jiang, M., Dennis, M., Parker-Holder, J., Lupu, A., Küttler, H., Grefenstette, E., Rocktäschel, T., and Foerster, J. Grounding aleatoric uncertainty for unsupervised environment design. *Advances in Neural Information Processing Systems*, 35:32868–32881, 2022.

Jiang, M., Dennis, M., Grefenstette, E., and Rocktäschel, T. minimax: Efficient baselines for autocurricula in jax. In *Agent Learning in Open-Endedness Workshop at NeurIPS*, 2023.

Johnson, M., Hofmann, K., Hutton, T., and Bignell, D. The malmo platform for artificial intelligence experimentation. In *Ijcai*, pp. 4246–4247, 2016.

Koyamada, S., Okano, S., Nishimori, S., Murata, Y., Habara, K., Kita, H., and Ishii, S. Pgx: Hardware-accelerated parallel game simulators for reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

Küttler, H., Nardelli, N., Miller, A., Raileanu, R., Selvatici, M., Grefenstette, E., and Rocktäschel, T. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684, 2020.

Lange, R. T. gymnax: A JAX-based reinforcement learning environment library, 2022. URL http://github.com/RobertTLange/gymnax.

Lehman, J. and Stanley, K. Exploiting open-endedness to solve problems through the search for novelty. *Artificial Life - ALIFE*, 01 2008.

Lu, C., Kuba, J., Letcher, A., Metz, L., Schroeder de Witt, C., and Foerster, J. Discovered policy optimisation. *Advances in Neural Information Processing Systems*, 35: 16455–16468, 2022.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

Nikulin, A., Kurenkov, V., Zisman, I., Sinii, V., Agarkov, A., and Kolesnikov, S. XLand-minigrid: Scalable meta-reinforcement learning environments in JAX. In *Intrinsically-Motivated and Open-Ended Learning Workshop, NeurIPS2023*, 2023. URL https://openreview.net/forum?id=xALDC4aHGz.

Open Ended Learning Team et al. Open-ended learning leads to generally capable agents, 2021. URL https://arxiv.org/abs/2107.12808.

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.

Oudeyer, P.-Y., Kaplan, F., and Hafner, V. V. Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 11(2): 265–286, 2007.

Parker-Holder, J., Jiang, M., Dennis, M., Samvelyan, M., Foerster, J., Grefenstette, E., and Rocktäschel, T. Evolving curricula with regret-based environment design. In *Proceedings of the International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 17473–17498. PMLR, 2022. URL https://proceedings.mlr.press/v162/parker-holder22a.html.

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.

Rutherford, A., Ellis, B., Gallici, M., Cook, J., Lupu, A., Ingvarsson, G., Willi, T., Khan, A., de Witt, C. S., Souly, A., Bandyopadhyay, S., Samvelyan, M., Jiang, M., Lange, R. T., Whiteson, S., Lacerda, B., Hawes, N., Rocktaschel, T., Lu, C., and Foerster, J. N. Jaxmarl: Multi-agent rl environments in jax. *arXiv preprint arXiv:2311.10090*, 2023.

Samvelyan, M., Rashid, T., De Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G., Hung, C.-M., Torr, P. H., Foerster, J., and Whiteson, S. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

Samvelyan, M., Kirk, R., Kurin, V., Parker-Holder, J., Jiang, M., Hambro, E., Petroni, F., Küttler, H., Grefenstette, E., and Rocktäschel, T. Minihack the planet: A sandbox for open-ended reinforcement learning research, 2021. URL https://arxiv.org/abs/2109.13202.

Schmidhuber, J. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pp. 222–227, 1991.

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

Stanley, K. O., Lehman, J., and Soros, L. Open-endedness: The last grand challenge you've never heard of. *While open-endedness could be a force for discovering intelligence, it could also be a component of AI itself*, 2017.

Sukhbaatar, S., Lin, Z., Kostrikov, I., Synnaeve, G., Szlam, A., and Fergus, R. Intrinsic motivation and automatic curricula via asymmetric self-play. In *6th International Conference on Learning Representations, ICLR 2018*, 2018. URL https://openreview.net/forum?id=SkT5Yg-RZ.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. Domain randomization for transferring deep neural networks from simulation to the real world. In *International Conference on Intelligent Robots and Systems*, pp. 23–30. IEEE, 2017. doi: 10.1109/IROS.2017.8202133. URL https://doi.org/10.1109/IROS.2017.8202133.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033. IEEE, 2012.

Wang, R., Lehman, J., Clune, J., and Stanley, K. O. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019.

You, K., Long, M., Wang, J., and Jordan, M. I. How does learning rate decay help modern neural networks? *arXiv preprint arXiv:1908.01878*, 2019.

Young, K. and Tian, T. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.

A. Differences between Craftax-Classic and Crafter

While `Craftax-Classic` was designed to be as similar to Crafter as possible, the realities of coding in JAX necessitated some technical deviations.

A.1. Creatures

In Crafter, the creatures (zombies, skeletons and cows) are continually spawned around the player and despawned when the player moves far away from them. This allows for the appearance of a world full of creatures, without having to simulate far-away entities that will have no effect on the player.

We broadly follow this system with `Craftax-Classic`, with the additional constraint that we have to pre-specify the maximum number of each creature that can exist simultaneously. This is because JAX requires that when we compile a function, the shapes of every array in the compute graph must be known a priori, preventing the use of arbitrarily sized arrays. This means that every timestep we have to simulate the steps for a fixed number of creatures. It should be noted that this has the extra side effect that we actually always have to simulate the maximum number of each creature every timestep, even if less are currently spawned. In order to allow for a dynamic number of creatures (up to the maximum) we mask out the influences of unspawned creatures.

The maximum number of creatures are 3, 3, 2 and 3 for zombies, cows, skeletons and arrows (which we treat as creatures for purpose of code reuse), respectively. In practice it is quite rare to observe more than these amounts of each creature simultaneously in Crafter.

A similar reasoning applies to cultivating plants, where only a finite number of growing plants can be tracked. This was set to 10 in `Craftax-Classic`.

A.2. World Generation

While Crafter uses Simplex Noise as the base of its world generation, we used the very similar Perlin Noise, adapted from the Perlin-Numpy repository[4]. This produces qualitatively similar results to Crafter (Figures 9 and 10).

B. Speed Comparison Details

All experiments were run on a single machine with a GeForce RTX 4090 (24GB of VRAM), i9-13900K (24 cores with 32 threads) and 32GB of RAM. For each environment we started with a single worker and proceeded to double the number of environment workers until the machine crashed from an out-of-memory error. We measure the average steps per second over the **entire RL learning process**, unlike other works which report times from running the environment with a static/random agent. We believe this provides a more useful indication of the times a researcher could actually expect to see when experimenting on these benchmarks.

For `Craftax` we tested on the `Craftax-Classic-Symbolic` and `Craftax-Symbolic` environments, using an end-to-end compiled RL pipeline based off PureJaxRL (Lu et al., 2022). For NetHack we used the built-in IMPALA baseline. For Crafter and Procgen we used the stable-baselines3 (Raffin et al., 2021) PPO implementation, with the SubprocVecEnv for parallelised workers on multiple threads. We were unable to get any baselines working on MineRL so, in order to be as generous as possible, we simply ran the environment on multiple threads with random actions. For this reason the MineRL numbers are likely overestimates of the true expected end-to-end steps-per-second. Since this was by far the slowest environment anyway, we didn't feel the need to explore this further.

For Craftax, Crafter, Procgen and NetHack we tried to match the magnitude of the value/policy network learning as much as possible by matching number of network parameters (different observation structures made it impossible to completely match network architectures), epochs and minibatch size. There is no way to perfectly compare all these environments, but we felt that our decisions represent the speeds that a researcher using these environments is likely to see in practice and therefore is a fair and useful metric.

[4] https://github.com/pvigier/perlin-numpy

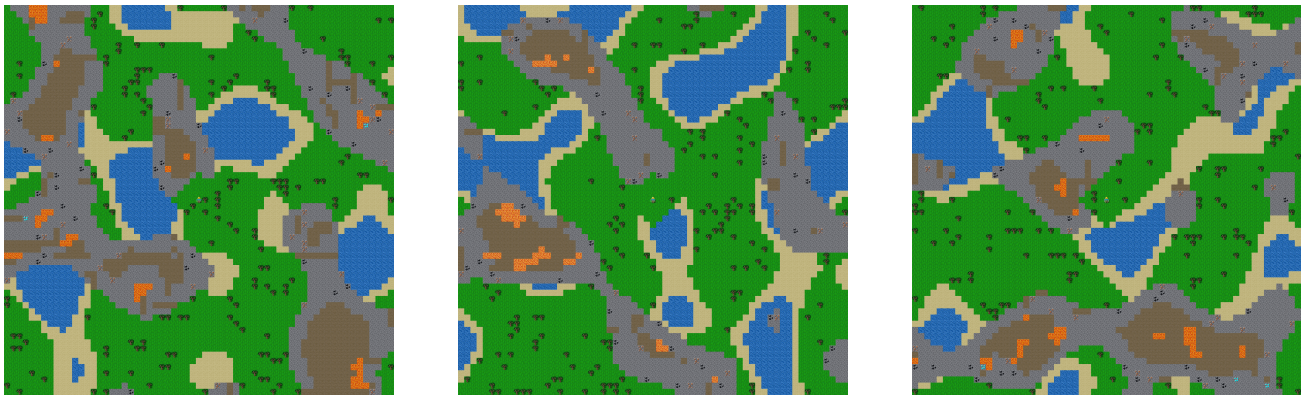


Figure 9. Levels from `Craftax-Classic`

Figure 10. Levels from Crafter. Figure reprinted from Hafner (2021).

Environment	Steps Per Second (Best case)	Environment Workers (Best Case)
`Craftax-Classic`	405618	4096
`Craftax`	266961	4096
Procgen	7638	1024
NetHack	5628	64
Crafter	1580	1024
MineRL	133	8

Table 1. Best case speed comparison for each environment.

C. Optimistic Environment Resets

One of the features of compilation in JAX is that branching cannot occur inside a function that is parallelised with the *vmap* operator. In order to achieve branching-like behaviour, both branches must always be evaluated and then the appropriate output can be selected with the branching condition. The most significant place that this occurs within JAX RL environments is with resets, where applying this constraint involves both resetting the environment and stepping it every timestep, and then selecting the desired next state based on the outputted done flag. Indeed, this is the standard approach taken in Gymnax (Lange, 2022). While this approach is fine when dealing with simple environments, resetting in `Craftax` is significantly more expensive as it involves generating 9 floors with multiple passes of Perlin Noise and room generation. Furthermore, episode lengths tend to be in the hundreds of timesteps, so over 99% of generated worlds are simply discarded.

To overcome this bottleneck, we instead perform what we term 'optimistic environment resets'. When running N parallel environment workers, we only generate M new worlds every timestep, where $M << N$. Then, for the environments that do need resetting, we sample without replacement from our set of M generated new states. Typically this will make no tangible difference to the RL training process, except in the exceedingly rare case where more than M environments reset simultaneously. In this case, some environment workers will be allocated the same new state. It should be noted that, while this will reduce the diversity of experiences seen by the agent, the stochasticity in the agent and the environment will mean the trajectories will still be different.

In practice, we use a ratio of 1 new state generated for each 16 environment workers. In the case of 1024 environment workers, this means we generate 64 new states every timestep. Assuming an average episode length of around 200 (in practice we usually see much longer), we can model the number of done flags as a binomial distribution $X \sim B(1024, \frac{1}{200})$. The probability that we end up with more resets than generated levels is therefore $P(X > 64)$ which is less than $\frac{1}{10^{10}}$. This shows that, when running for 1 billion timesteps, it is extremely unlikely we would even see a single duplicated world.

We have shown that this optimisation has essentially no effect on the RL process, while providing a significant speedup (around 2x in `Craftax`).

D. Further Environment Details

The maximum episode length is 100,000 at which the episode is truncated.

D.1. Block Types

The list of block types in `Craftax` is shown in Table 2.

D.2. Creatures

The list of creatures in `Craftax` is shown in Table 3.

D.3. World Generation

Examples of procedurally generated floors are shown in Figures 11, 12, 13, 14, 15, 16, 17 and 18. Floors 2, 5, 7 and 8 have been artificially lit up to make them visible, whereas in gameplay torches need to be placed to reveal them. Floor 0 is the overworld and is essentially equivalent to the world from `Craftax-Classic`.

Figure 11. `Craftax`: Floor 1 - Dungeons

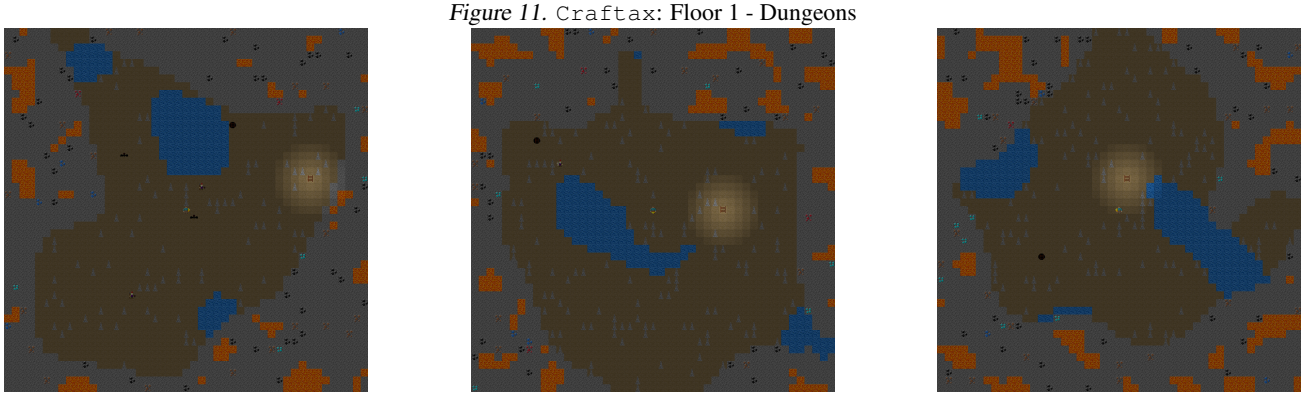


Figure 12. `Craftax`: Floor 2 - Gnomish Mines

Figure 13. `Craftax`: Floor 3 - Sewers

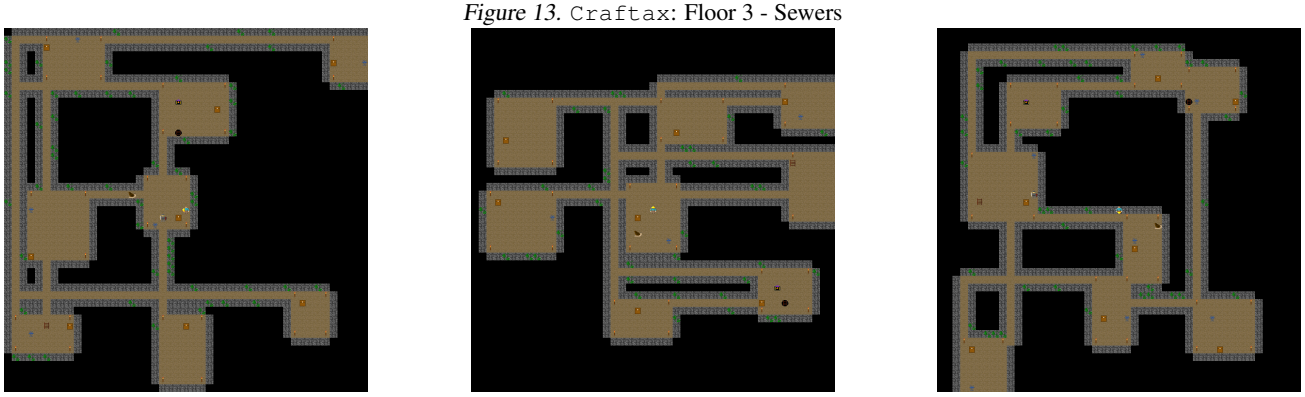


Figure 14. `Craftax`: Floor 4 - Vaults

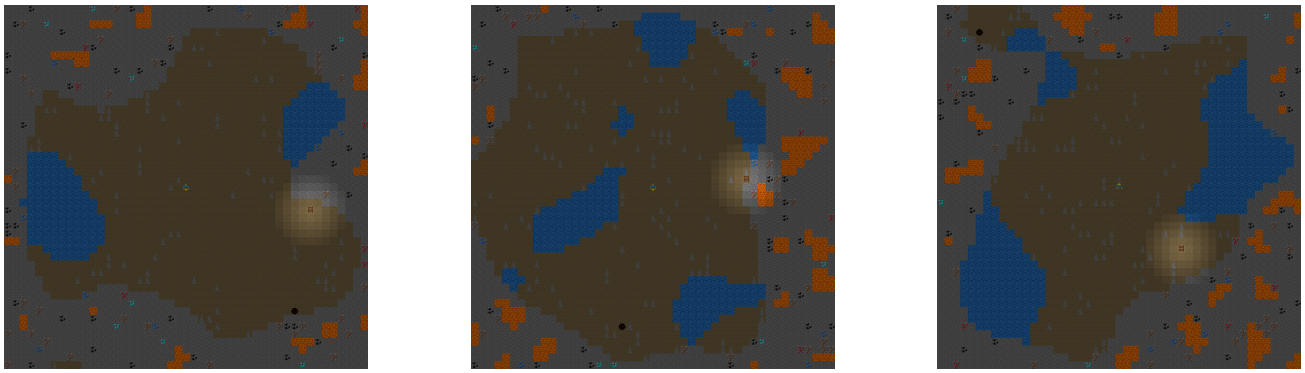


Figure 15. `Craftax`: Floor 5 - Troll Mines

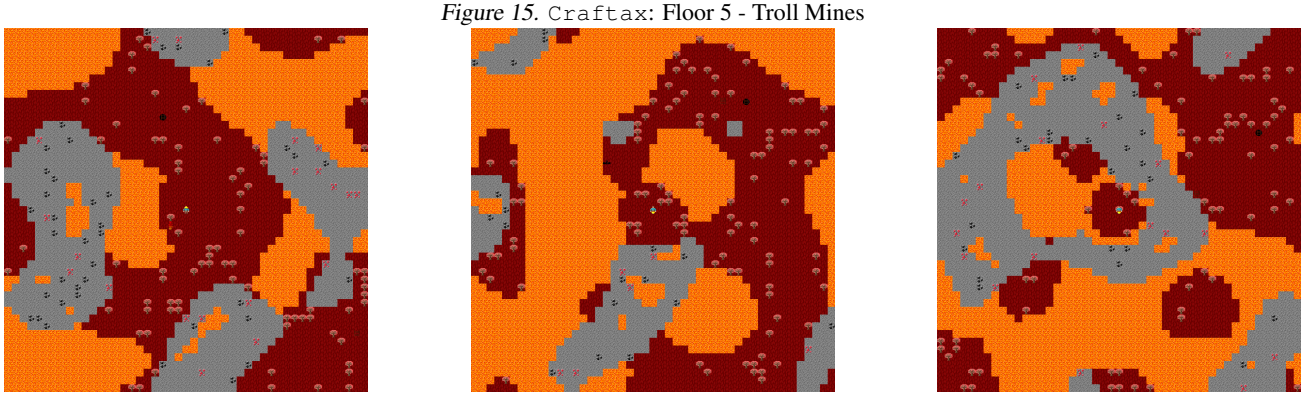


Figure 16. `Craftax`: Floor 6 - Fire Realm

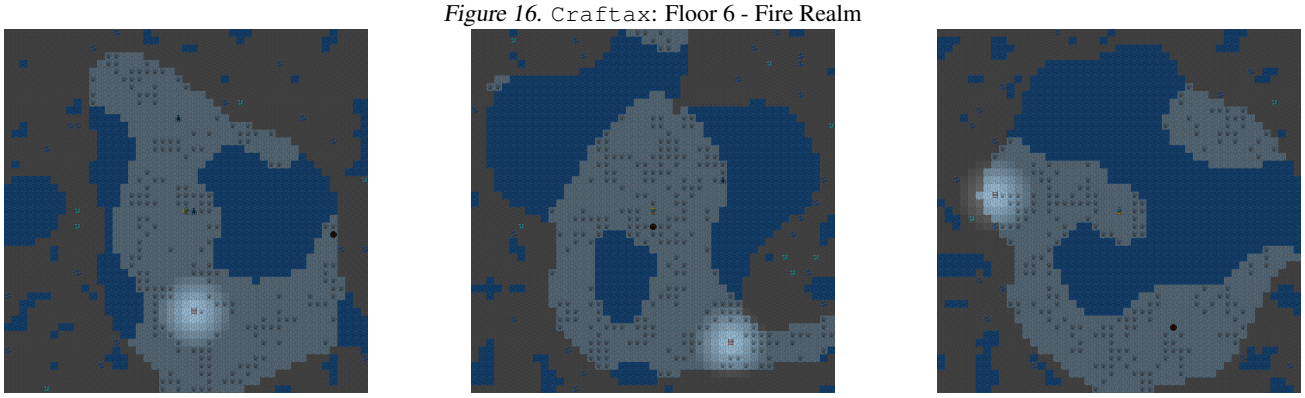


Figure 17. `Craftax`: Floor 7 - Ice Realm

Figure 18. `Craftax`: Floor 8 - Graveyard

D.4. Observation Space

Both Craftax and Craftax-Classic have options for pixel-based and symbolic observations. Each consist of a view of the map and the players stats and inventory. The map view in Craftax-Classic is 7x9 squares, while Craftax uses 9x11.

Pixels The pixel-based observations for Craftax-Classic take the same form as those in Crafter, with each 16x16 square downscaled to 7x7. For Craftax, we downscale only to 10x10. This is because we deal with numbers greater than 9, meaning that the digits had to be rendered in a smaller font size. At 7x7 downscaling many of these digits were indistinguishable. Pixel observations from Crafter, Craftax-Classic and Craftax are shown in Figure 25.

Symbolic The primary observation space we experiment on is symbolic, as it allows for significantly faster runtime. Each square on the map encodes three one-hot vectors and a light level. The first one-hot vector is of size 37 and encodes the block type, the second is of size 5 and encodes the item type, while the third is of size 36 and encodes the creature type. The light level is a float between 0 and 1. If the light level for a block is < 0.05 then the block, item and creature are masked out.

The inventory/stats bar is represented as an array with each element representing the amount corresponding to a particular item or attribute. All values are scaled to be roughly in the range 0-1. All inventory items (which the player can carry up to 99 of) are represented as $\frac{\sqrt{n}}{10}$. Table 4 lists the symbolic inventory observation for Craftax. The observation for Craftax-Classic is the subset of these elements that exist in the classic version.

Textual Although we don't make use of it for our experiments, we also provide a textual 'renderer'. This encodes the grid in the agents view as "<CREATURE> on <ITEM> on <BLOCK>" for each viewable block, so a troll standing on a path block with a torch on it would be encoded as "troll on torch on path". We encode the players inventory and intrinsics as "<INVENTORY_ITEM>: X", so an agent with 2 diamonds in its inventory would see "DIAMOND: 2". Use in conjunction with the text tutorial found in the codebase, this could be use for language-conditioned learning.

D.5. Action Space

The action space for Craftax is shown in Table 5.

D.6. Achievements

The achievements for Craftax are listed in Table 6. Achievements for Craftax-Classic are those with an ID of up to and including 21.

E. Hyperparameter Tuning

E.1. Craftax-1B

The hyperparameters and considered values for PPO on Craftax-1B are shown in Table 7. Each run was performed for 1 billion timesteps with 1 seed. We started from a set of values picked informally and then tuned each hyperparameter individually. These hyperparameters were then taken as the baseline for the other methods, which all build off of the PPO implementation. For PPO-RNN, we kept the same hyperparameters as PPO, with the hidden size being equal to the MLP layer size. The hyperparameters for ICM, E3B and RND are shown in Tables 8, 9 and 10 respectively.

E.2. Craftax-1M

The hyperparameters for Craftax-1M were tuned more thoroughly, as the experiments were very fast to run. We performed a random search over the PPO hyperparameters to obtain a baseline PPO implementation, before then running a random search for ICM and E3B. The hyperparameters for PPO, ICM and E3B are shown in Tables 11, 12 and 13 respectively.

F. Further Results

F.1. Individual Achievements

The success rates through training for Craftax-1B and Craftax-1M are shown in Figures 19 and 20 respectively.

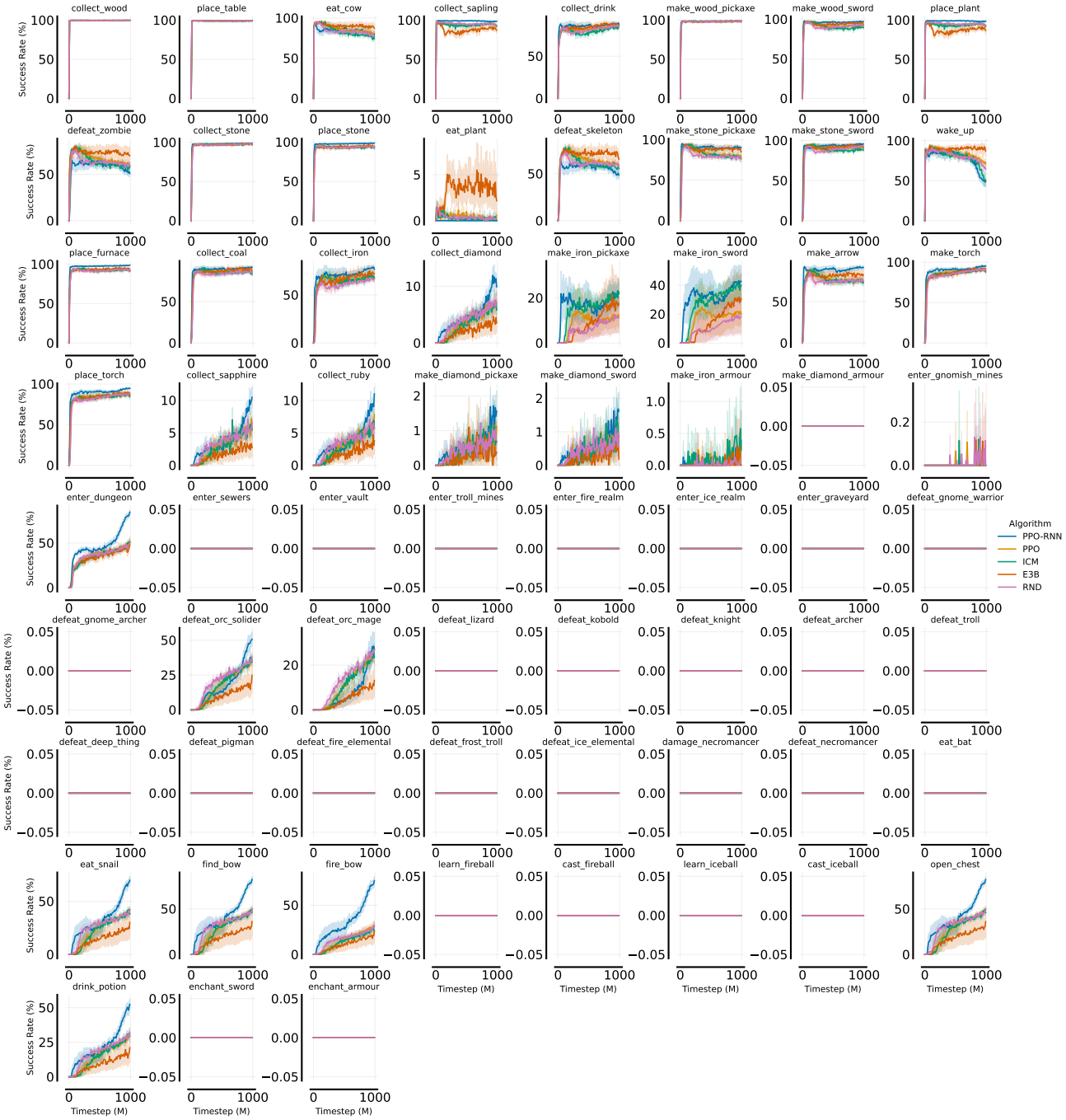


Figure 19. Success rate curves for all achievements on Craftax-1B. Each algorithm was run on 10 seeds for 1 billion timesteps. The shaded area denotes 1 standard error.

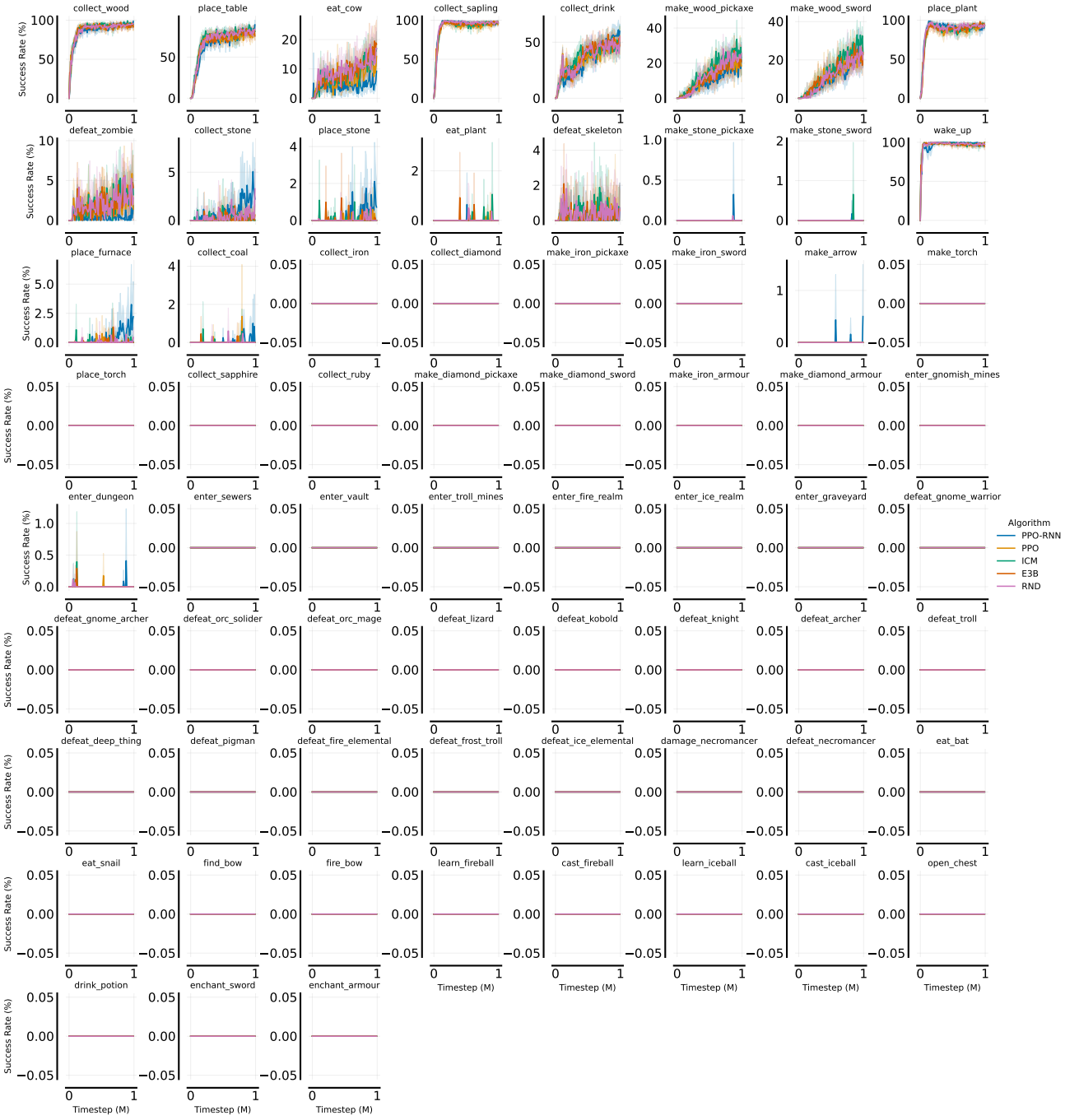


Figure 20. Success rate curves for all achievements on Craftax-1M. Each algorithm was run on 10 seeds for 1 million timesteps. The shaded area denotes 1 standard error.

F.2. Craftax-Extended with 10B Environment Interactions

To really push the limits of the environment, we ran PPO-RNN on 10B environment interactions, with the results shown in Figures 22 and 21. The results show that learning for 10B iterations barely affects performance, with the agent still not making it into the gnomish mines. Interestingly, as in the main results on Craftax-1B, the reward increases more quickly towards the end of the experiment. The fact that this occurs at roughly the same value for both 1B and 10B implies that this is an artifact of the learning rate schedule.

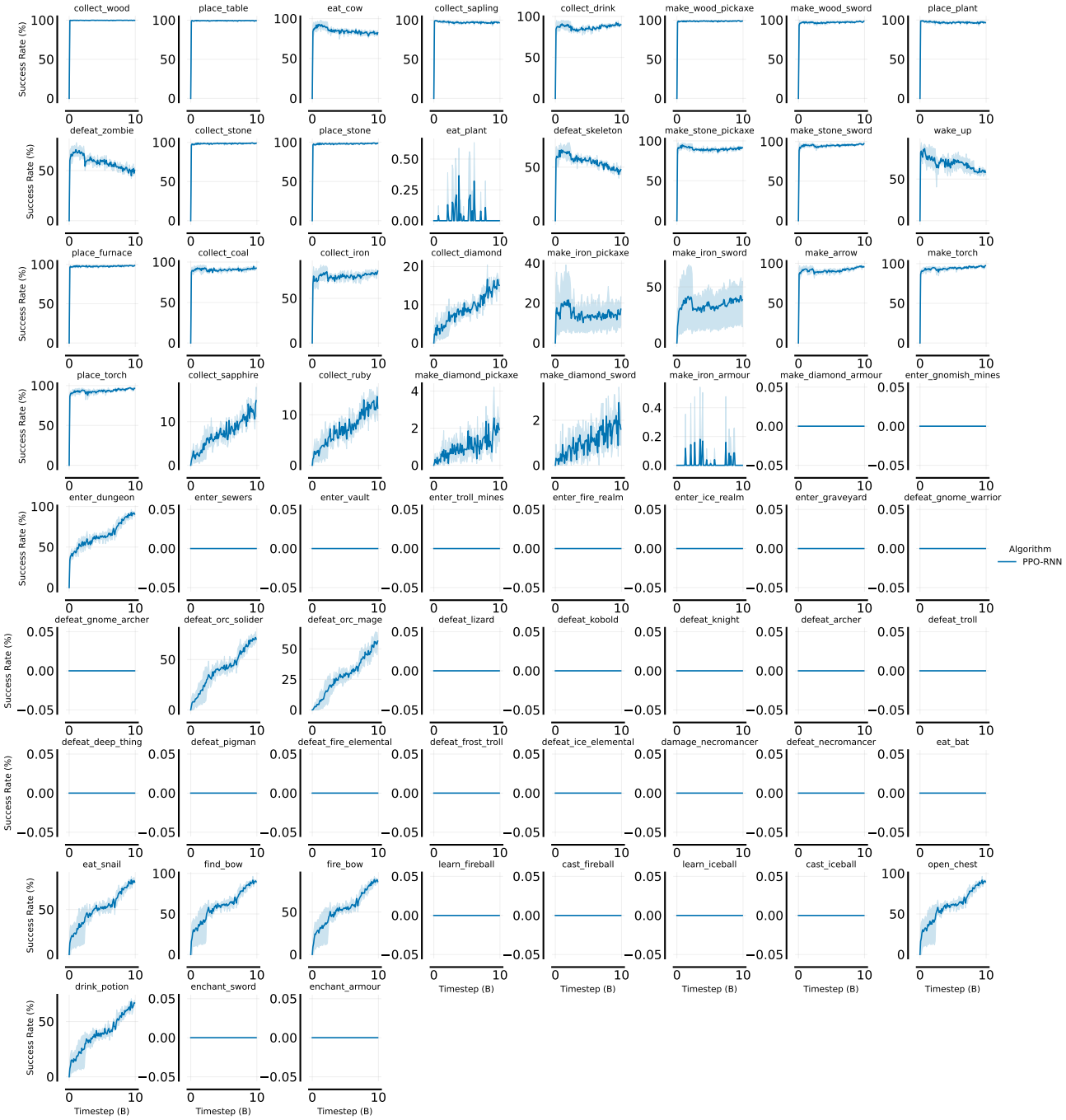


Figure 21. Achievement yields for PPO-RNN on `Craftax-Extended` given 10 billion environment interactions. The experiment was repeated with 4 seeds and the shaded areas shows 1 standard error.

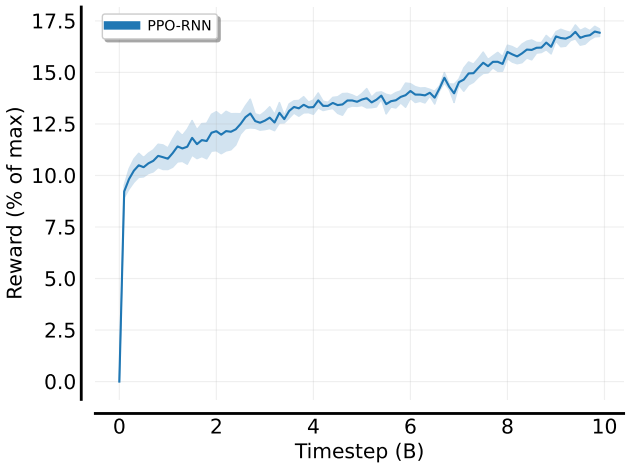


Figure 22. Reward for PPO-RNN on `Craftax-Extended` given 10 billion environment interactions. The experiment was repeated with 4 seeds and the shaded areas shows 1 standard error.

F.3. Craftax-Classic

Figures 23 and 24 show the results of running PPO-RNN on `Craftax-Classic` for 1 billion timesteps. As can be seen, all achievements excepting `COLLECT_DIAMOND` and `EAT_PLANT` are reliably fulfilled. Collecting diamonds is achieved somewhat reliably while eating plants is only achieved very rarely. While the agent hasn't 'solved' the benchmark by reliably completing every achievement, it is so close to doing so that releasing only the `Craftax-Classic` would be of little use, as it would likely be solved very quickly.

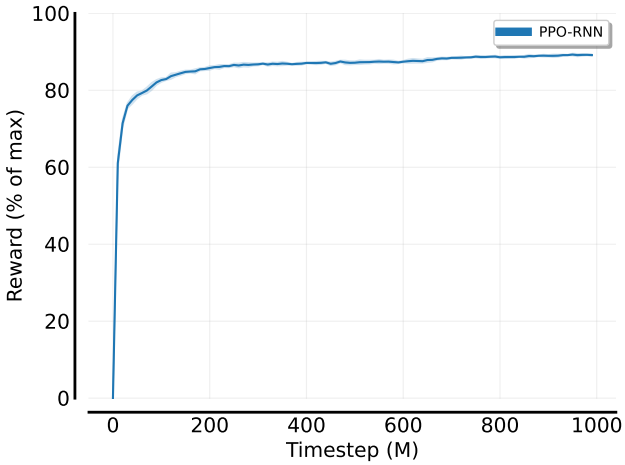


Figure 23. Reward for PPO-RNN on `Craftax-Classic` given 1 billion environment interactions. The experiment was repeated with 10 seeds and the shaded areas shows 1 standard error.

(a) Crafter (b) `Craftax-Classic` (c) `Craftax`

Figure 25. Comparison of downscaled pixel views for agent observations.

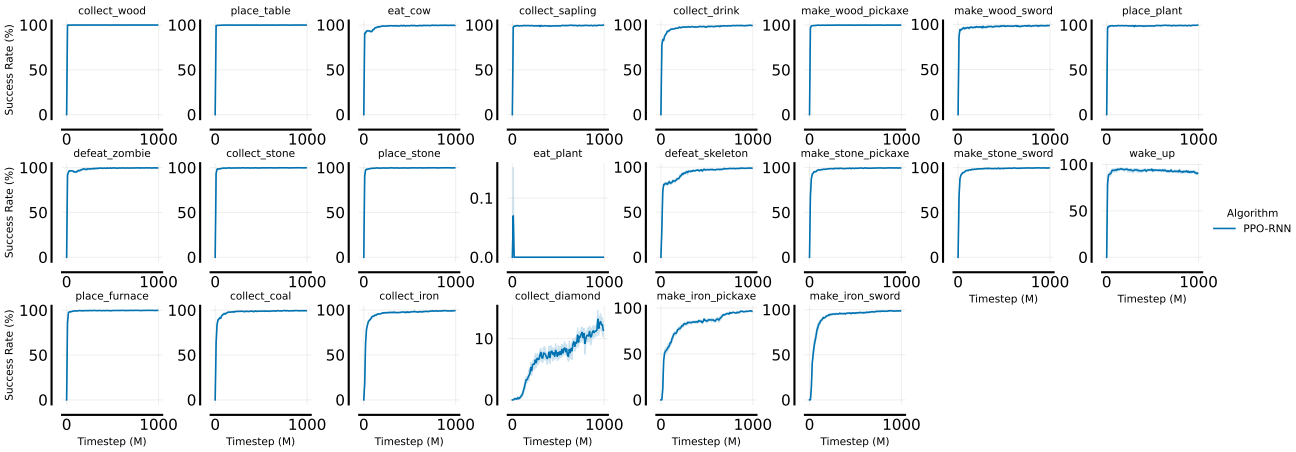


Figure 24. Achievement yields for PPO-RNN on `Craftax-Classic` given 1 billion environment interactions. The experiment was repeated with 10 seeds and the shaded areas shows 1 standard error.

G. UED

G.1. Implementation Changes

In standard implementations of UED (Jiang et al., 2023), the PPO rollout length is determined by the episode's length. This is because, whenever new levels are sampled, one rollout is performed. This is used to both (a) learn and (b) to approximate a regret value for this level. This does not pose a problem if the maximum episode length is short (such as in Minigrid); however, with `Craftax` and its long episodes, this approach is limited, as there must either be a long time between subsequent agent updates, or the score would only take into account part of an episode.

We modify the implementation to have multiple short inner rollouts, which the agent learns on as normal. After a certain number of these, we use the entire trajectory information to approximate regret, which then updates the adversary. This allows us to completely decouple these two aspects. Although this induces some non-stationarity—as the agent is learning on the trajectories that the adversary uses for scores—we found this does not significantly hamper performance. Finally, since the agent rarely runs out of time, we restrict the episode length to be 4096 instead of the full 10000.

G.2. Additional UED Results

Qualitative Results Figures 26 to 31 in show several levels curated by each method. The curation-based methods generally cause resources to be more centrally-located, near where the player spawns. The unrestricted ACCEL swapping method generates levels that are visually very different to the normally generated ones, as there is no restriction to where it can place tiles. The other mutations and PLR generate more plausible levels.

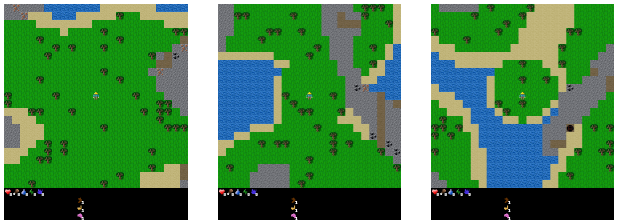


Figure 26. ACCEL Noise replay levels at the start, middle and end of training

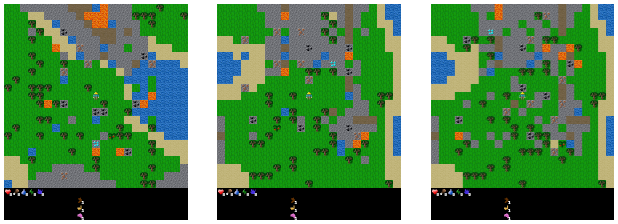


Figure 27. ACCEL Swap replay levels at the start, middle and end of training

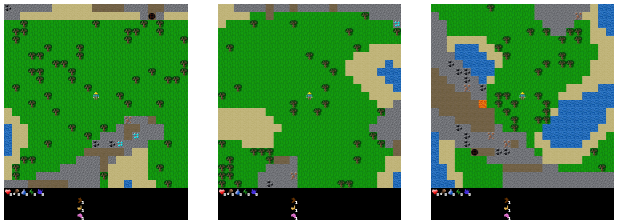


Figure 28. ACCEL Swap (Restrict) replay levels at the start, middle and end of training

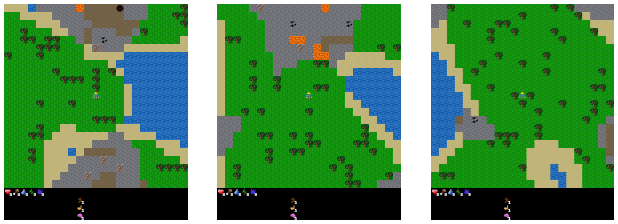


Figure 29. Normal PLR replay levels at the start, middle and end of training

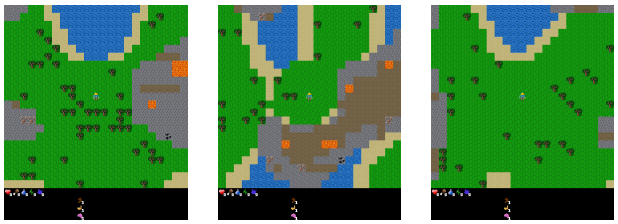


Figure 30. Robust PLR replay levels at the start, middle and end of training

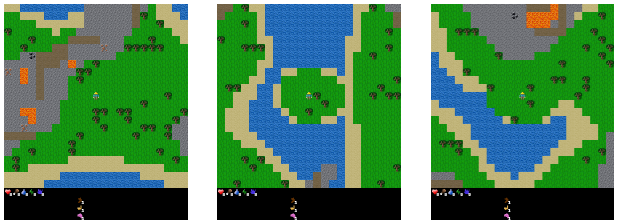


Figure 31. DR levels at the start, middle and end of training

G.3. Hyperparameters

Table 14 contains the UED hyperparameters used. We tuned these as follows.

1. We used PPO hyperparameters found in an earlier hyperparameter sweep and did a small search over learning rate $\{0.0002, 0.0003\}$.
2. We then performed a grid search over the following parameters for both PLR variations: temperature $\{0.3, 1.0\}$, replay probability $\{0.5, 0.8\}$, score function $\{\text{MaxMC}, \text{PVL}\}$, prioritisation $\{\text{rank}, \text{topk}\}$ and topk's k in $\{32, 256\}$.
3. We then used the best parameters for the ACCEL-based methods, and further tuned over the number of mutations in $\{10, 100, 200\}$. In this case, PLR outperformed robust PLR, so all of our ACCEL methods also perform gradient updates on DR levels, but not on mutated levels.

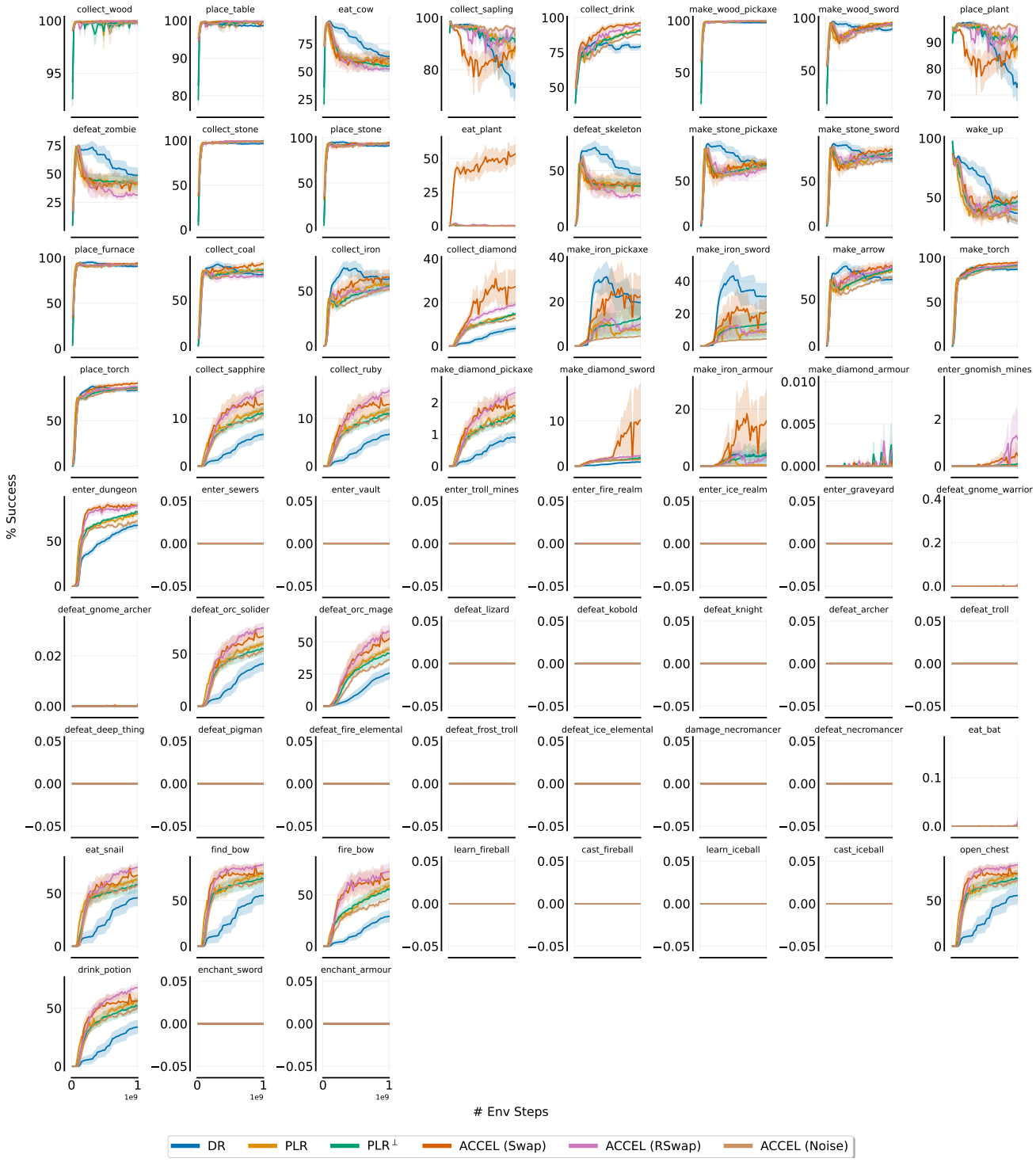


Figure 32. Replay achievement success rates for UED

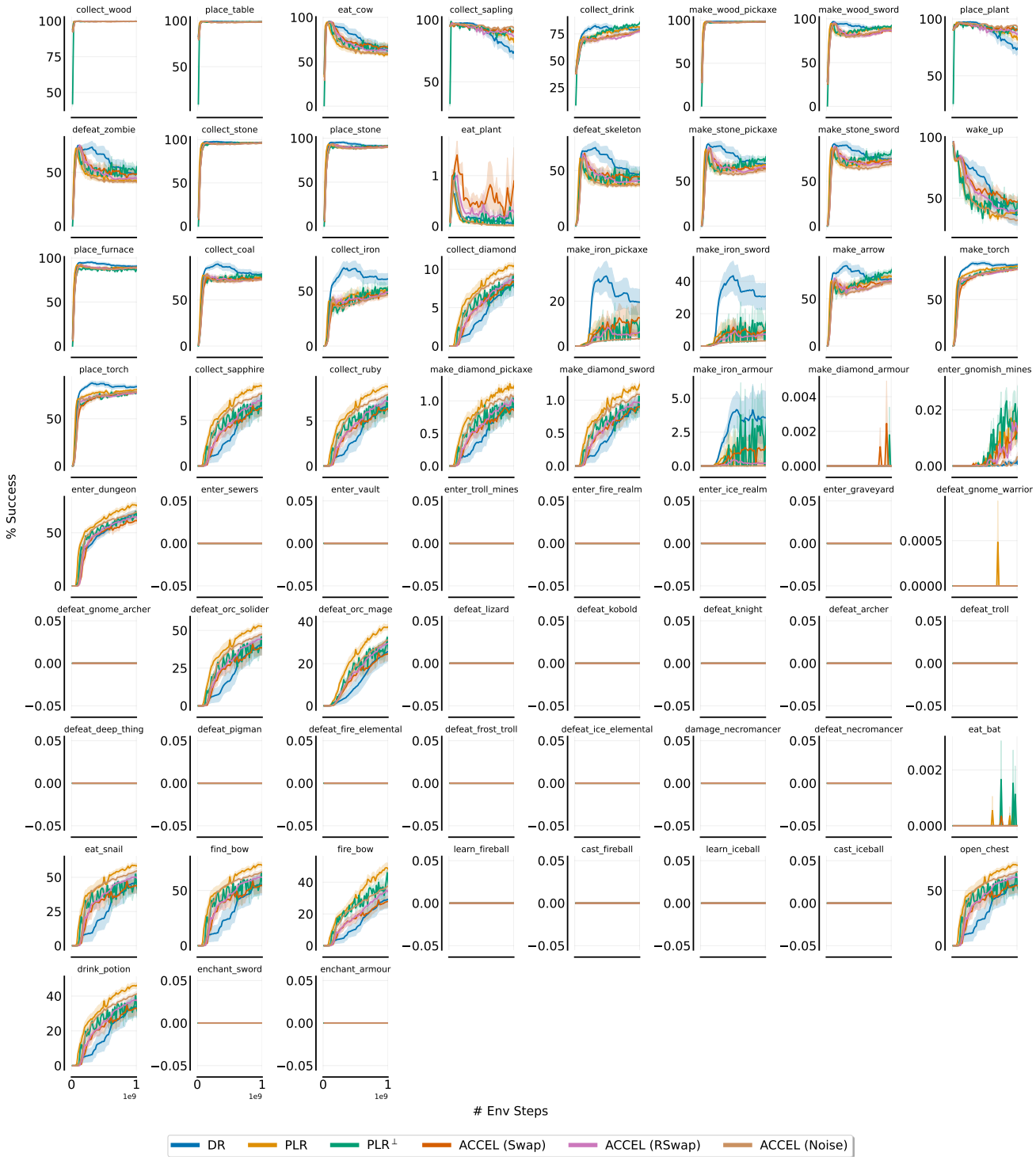


Figure 33. DR achievement success rates for UED

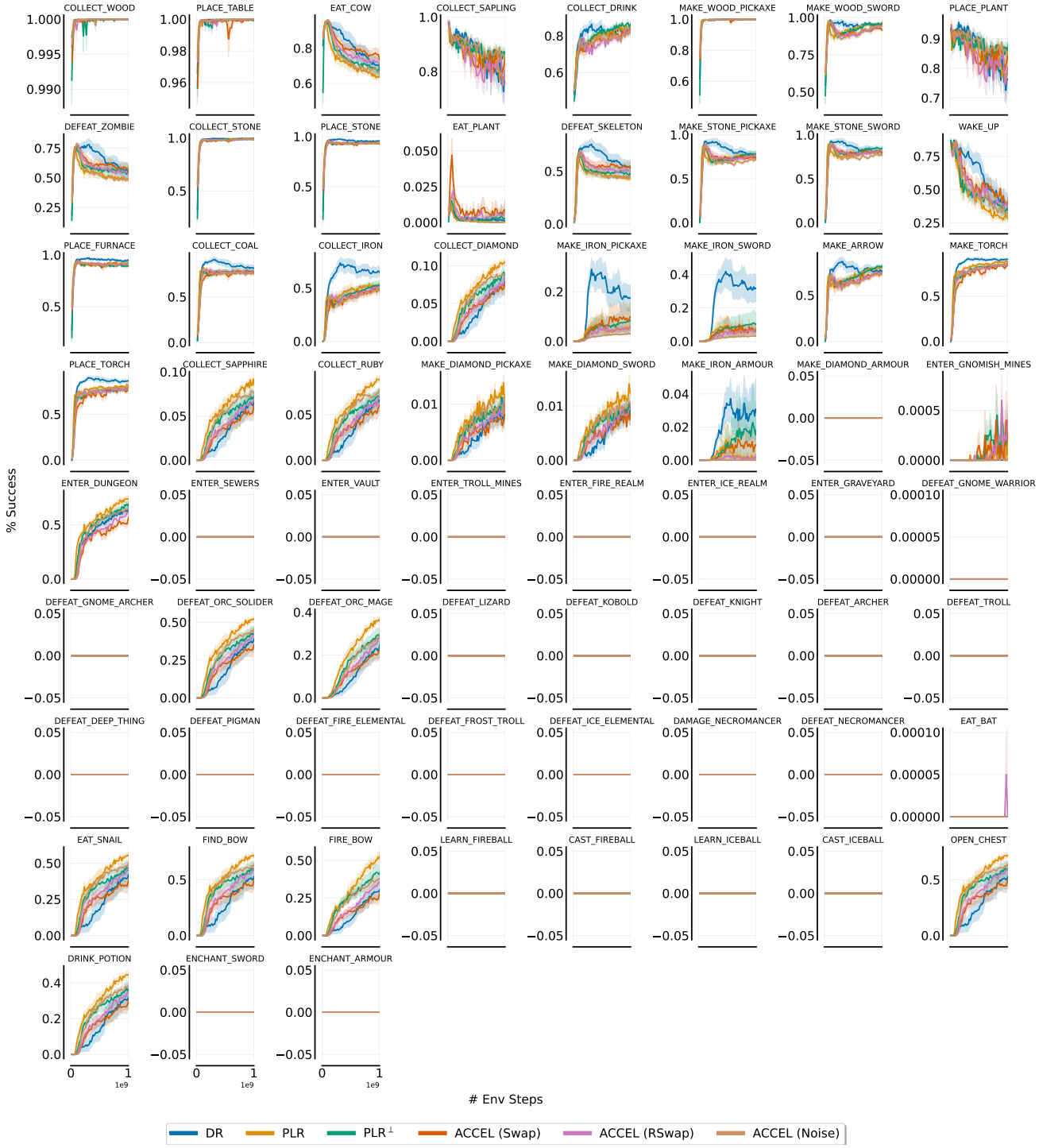


Figure 34. Evaluation achievement success rates for UED

All ACCEL-methods use exploratory gradient updates.

Table 14. UED Hyperparameters. We use a replay rate of 0.8 for PLR$^{\perp}$ and a replay rate of 0.5 for PLR and ACCEL.

Parameter	DR	PLR/PLR$^{\perp}$/ACCEL
PPO		
Network Width	512	
# Environments	1024	
γ	0.99	
λ_{GAE}	0.9	
PPO inner rollout	64	
PPO outer rollout	64	
PPO epochs	5	
PPO minibatches per epoch	2	
PPO clip range	0.2	
Adam learning rate	0.0002	0.0003
Anneal LR	yes	
Adam ϵ	1e-5	
PPO max gradient norm	1.0	
PPO value clipping	yes	
return normalization	no	
value loss coefficient	0.5	
student entropy coefficient	0.01	
PLR		
Replay rate, p	-	0.8 (PLR$^{\perp}$), 0.5 (PLR, ACCEL)
Buffer size, K	-	4000
Scoring function	-	MaxMC
Prioritization	-	Rank
Temperature, β	-	1.0
Staleness coefficient	-	0.3
ACCEL		-
Number of Mutations (Swap)	-	10
Number of Mutations (RSwap)	-	200
Number of Mutations (Noise)	-	100

ID	Name	Texture
2	Grass	
3	Water	
4	Stone	
5	Tree	
6	Wood	
7	Path	
8	Coal	
9	Iron	
10	Diamond	
11	Crafting Table	
12	Furnace	
13	Sand	
14	Lava	
15	Plant	
16	Ripe Plant	
17	Wall	
18	Darkness	
19	Mossy Wall	

ID	Name	Texture
20	Stalagmite	
21	Sapphire	
22	Ruby	
23	Chest	
24	Fountain	
25	Fire Grass	
26	Ice Grass	
27	Gravel	
28	Fire Tree	
29	Ice Shrub	
30	Enchantment Table (Fire)	
31	Enchantment Table (Ice)	
32	Necromancer	
33	Grave 1	
34	Grave 2	
35	Grave 3	
36	Necromancer (Vulnerable)	

Table 2. Listing of block types. Most textures were either taken directly or adapted from those in Crafter.

Name	Texture	Type	Health	Damage	Defense (%)	Collision Type	Floor
Zombie		Melee	5	2	0	Ground	0
Skeleton		Ranged	3	2	0	Ground	0
Cow		Passive	3	-	0	Ground	0
Orc Solider		Melee	7	3	0	Ground	1
Orc Mage		Ranged	5	3	0	Ground	1
Snail		Passive	6	-	0	Ground	1,3,4
Gnome Warrior		Melee	9	4	0	Ground	2
Gnome Archer		Ranged	6	2	0	Ground	2
Bat		Passive	4	-	0	Flying	2,5,6
Lizard		Melee	11	5	0	Amphibian	3
Kobold		Ranged	8	4	0	Ground	3
Knight		Melee	12	6	50	Ground	4
Archer		Ranged	12	4	50	Ground	4
Troll		Melee	20	6 + 1 + 1	20	Ground	5
Deep Thing		Ranged	6	4 + 3 + 3	0	Aquatic	5
Pig Man		Melee	20	3 + 5	90, 100	Ground	6
Fire Elemental		Ranged	14	3 + 5	90, 100	Flying	6
Frost Troll		Melee	24	4 + 5	90, 100	Ground	7
Ice Elemental		Ranged	16	4 + 4	90, 100	Flying	7

Table 3. Listing of creatures. Some textures were taken directly or adapted from Crafter. Text in red and blue denote fire and ice attack/defence respectively.

Item	Representation	Min	Max
Wood	$\frac{\sqrt{n}}{10}$	0	1
Stone	$\frac{\sqrt{n}}{10}$	0	1
Coal	$\frac{\sqrt{n}}{10}$	0	1
Iron	$\frac{\sqrt{n}}{10}$	0	1
Diamond	$\frac{\sqrt{n}}{10}$	0	1
Sapphire	$\frac{\sqrt{n}}{10}$	0	1
Ruby	$\frac{\sqrt{n}}{10}$	0	1
Sapling	$\frac{\sqrt{n}}{10}$	0	1
Torch	$\frac{\sqrt{n}}{10}$	0	1
Arrow	$\frac{\sqrt{n}}{10}$	0	1
Potion (x6)	$\frac{\sqrt{n}}{10}$	0	1
Book	$\frac{n}{2}$	0	1
Pickaxe	$\frac{\text{level}}{4}$	0	1
Sword	$\frac{\text{level}}{4}$	0	1
Sword Enchantment	$\mathbb{I}_{\text{fire}} + 2 \cdot \mathbb{I}_{\text{ice}}$	0	2
Bow	$\mathbb{I}_{\text{bow}}$	0	1
Armour (x4)	$\frac{\text{level}}{2}$	0	1
Armour Enchantment (x4)	$\mathbb{I}_{\text{fire}} + 2 \cdot \mathbb{I}_{\text{ice}}$	0	2
Health	$\frac{x}{10}$	0	1.4
Food	$\frac{x}{10}$	0	1.7
Drink	$\frac{x}{10}$	0	1.7
Energy	$\frac{x}{10}$	0	1.7
Mana	$\frac{x}{10}$	0	2.1
XP	$\frac{x}{10}$	0	0.8
Dexterity	$\frac{x}{10}$	0.1	0.5
Strength	$\frac{x}{10}$	0.1	0.5
Intelligence	$\frac{x}{10}$	0.1	0.5
Direction	$\text{onehot}(n)$	–	–
Day/Night	x	0	1
Sleeping	$\mathbb{I}_{\text{sleep}}$	0	1
Resting	$\mathbb{I}_{\text{rest}}$	0	1
Learned Fireball	$\mathbb{I}_{\text{fireball}}$	0	1
Learned Iceball	$\mathbb{I}_{\text{iceball}}$	0	1
Floor	$\frac{n}{10}$	0	0.9
Floor Cleared	$\mathbb{I}_{\text{cleared floor}}$	0	1
Boss Vulnerable	$\mathbb{I}_{\text{boss vulnerable}}$	0	1

Table 4. Symbolic observations for player inventory.

Action ID	Action Name	Key
0	NOOP	Q
1	LEFT	A
2	RIGHT	D
3	UP	W
4	DOWN	S
5	DO	Space
6	SLEEP	Tab
7	PLACE_STONE	R
8	PLACE_TABLE	T
9	PLACE_FURNACE	F
10	PLACE_PLANT	P
11	MAKE_WOOD_PICKAXE	1
12	MAKE_STONE_PICKAXE	2
13	MAKE_IRON_PICKAXE	3
14	MAKE_WOOD_SWORD	5
15	MAKE_STONE_SWORD	6
16	MAKE_IRON_SWORD	7
17	REST	e
18	DESCEND	.
19	ASCEND	,
20	MAKE_DIAMOND_PICKAXE	4

Action ID	Action Name	Key
21	MAKE_DIAMOND_SWORD	8
22	MAKE_IRON_ARMOUR	Y
23	MAKE_DIAMOND_ARMOUR	U
24	SHOOT_ARROW	I
25	MAKE_ARROW	O
26	CAST_FIREBALL	G
27	CAST_ICEBALL	H
28	PLACE_TORCH	J
29	DRINK_POTION_RED	Z
30	DRINK_POTION_GREEN	X
31	DRINK_POTION_BLUE	C
32	DRINK_POTION_PINK	V
33	DRINK_POTION_CYAN	B
34	DRINK_POTION_YELLOW	N
35	READ_BOOK	M
36	ENCHANT_SWORD	K
37	ENCHANT_ARMOUR	L
38	MAKE_TORCH	[
39	LEVEL_UP_DEXTERITY	]
40	LEVEL_UP_STRENGTH	-
41	LEVEL_UP_INTELLIGENCE	=
42	ENCHANT_BOW	;

Table 5. Actions for `Craftax`.

ID	Name	Category
0	COLLECT_WOOD	Basic (1)
1	PLACE_TABLE	Basic (1)
2	EAT_COW	Basic (1)
3	COLLECT_SAPLING	Basic (1)
4	COLLECT_DRINK	Basic (1)
5	MAKE_WOOD_PICKAXE	Basic (1)
6	MAKE_WOOD_SWORD	Basic (1)
7	PLACE_PLANT	Basic (1)
8	DEFEAT_ZOMBIE	Basic (1)
9	COLLECT_STONE	Basic (1)
10	PLACE_STONE	Basic (1)
11	EAT_PLANT	Basic (1)
12	DEFEAT_SKELETON	Basic (1)
13	MAKE_STONE_PICKAXE	Basic (1)
14	MAKE_STONE_SWORD	Basic (1)
15	WAKE_UP	Basic (1)
16	PLACE_FURNACE	Basic (1)
17	COLLECT_COAL	Basic (1)
18	COLLECT_IRON	Basic (1)
19	COLLECT_DIAMOND	Basic (1)
20	MAKE_IRON_PICKAXE	Basic (1)
21	MAKE_IRON_SWORD	Basic (1)
22	MAKE_ARROW	Basic (1)
23	MAKE_TORCH	Basic (1)
24	PLACE_TORCH	Basic (1)
25	MAKE_DIAMOND_SWORD	Intermediate (3)
26	MAKE_IRON_ARMOUR	Intermediate (3)
27	MAKE_DIAMOND_ARMOUR	Intermediate (3)
28	ENTER_GNOMISH_MINES	Intermediate (3)
29	ENTER_DUNGEON	Intermediate (3)
30	ENTER_SEWERS	Advanced (5)
31	ENTER_VAULT	Advanced (5)
32	ENTER_TROLL_MINES	Advanced (5)
33	ENTER_FIRE_REALM	Very Advanced (8)
34	ENTER_ICE_REALM	Very Advanced (8)
35	ENTER_GRAVEYARD	Very Advanced (8)
36	DEFEAT_GNOME_WARRIOR	Intermediate (3)
37	DEFEAT_GNOME_ARCHER	Intermediate (3)
38	DEFEAT_ORC_SOLIDER	Intermediate (3)
39	DEFEAT_ORC_MAGE	Intermediate (3)
40	DEFEAT_LIZARD	Advanced (5)
41	DEFEAT_KOBOLD	Advanced (5)
42	DEFEAT_TROLL	Advanced (5)
43	DEFEAT_DEEP_THING	Advanced (5)
44	DEFEAT_PIGMAN	Very Advanced (8)
45	DEFEAT_FIRE_ELEMENTAL	Very Advanced (8)
46	DEFEAT_FROST_TROLL	Very Advanced (8)
47	DEFEAT_ICE_ELEMENTAL	Very Advanced (8)
48	DAMAGE_NECROMANCER	Very Advanced (8)
49	DEFEAT_NECROMANCER	Very Advanced (8)
50	EAT_BAT	Intermediate (3)
51	EAT_SNAIL	Intermediate (3)
52	FIND_BOW	Intermediate (3)
53	FIRE_BOW	Intermediate (3)
54	COLLECT_SAPPHIRE	Intermediate (3)
55	LEARN_FIREBALL	Advanced (5)
56	CAST_FIREBALL	Advanced (5)
57	LEARN_ICEBALL	Advanced (5)
58	CAST_ICEBALL	Advanced (5)
59	COLLECT_RUBY	Intermediate (3)
60	MAKE_DIAMOND_PICKAXE	Intermediate (3)
61	OPEN_CHEST	Intermediate (3)
62	DRINK_POTION	Intermediate (3)
63	ENCHANT_SWORD	Advanced (5)
64	ENCHANT_ARMOUR	Advanced (5)
65	DEFEAT_KNIGHT	Advanced (5)
66	DEFEAT_ARCHER	Advanced (5)

Table 6. Achievements listing for `Craftax`. The category column indicates the difficulty classification of the achievement and the associated reward.

Hyperparameter	Considered Values	Value
Entropy Coefficient	$\{0, 0.01\}$	0.01
GAE λ	$\{0.8, 0.9, 0.95\}$	0.8
γ	$\{0.98, 0.99, 0.995, 0.999, 0.9995\}$	0.99
MLP Layer Size	$\{256, 512, 1024\}$	512
Learning Rate	$\{0.0002, 0.0003, 0.0004\}$	0.0002
Environment Workers	$\{256, 512, 1024\}$	1024
# Minibatches	$\{4, 8, 16\}$	8
# Steps	$\{64\}$	64
# Update Epochs	$\{4\}$	4
Clip ϵ	$\{0.2\}$	0.2
Value Function Coefficient	$\{0.5\}$	0.5
Activation Function	{tanh}	tanh
Anneal Learning Rate	{True}	True

Table 7. Hyperparameters for PPO for Craftax-1B.

Hyperparameter	Considered Values	Value
Intrinsic Reward Coefficient β	$\{0.1, 0.5, 1.0\}$	1.0
World Models Learning Rate	$\{0.0003\}$	0.0003
Forward Loss Coefficient	$\{1.0\}$	1.0
Inverse Loss Coefficient	$\{1.0\}$	1.0
World Models Layer Size	$\{256\}$	256
Latent Size	$\{32, 64, 128\}$	32

Table 8. Hyperparameters for ICM for Craftax-1B.

Hyperparameter	Considered Values	Value
Intrinsic Reward Coefficient β	$\{0.0005, 0.001, 0.005\}$	0.001
Ridge Regulariser λ	$\{0.005, 0.1, 0.5\}$	0.1

Table 9. Hyperparameters for E3B for Craftax-1B.

Hyperparameter	Considered Values	Value
Intrinsic Reward Coefficient β	$\{0.01, 0.1, 1.0\}$	1.0
RND Output Dimension	$\{4, 32, 256, 2048\}$	256
Distillation Network Learning Rate	$\{0.00003, 0.0003, 0.003\}$	0.0003
RND Loss Coefficient	$\{0.0001, 0.001, 0.01\}$	0.01
RND GAE Coefficient	$\{0.01, 0.1, 1.0\}$	0.01

Table 10. Hyperparameters for RND for Craftax-1B and Craftax-1M.

Hyperparameter	Considered Values	Value
Entropy Coefficient	$\{0.01\}$	0.01
GAE λ	$\{0.7, 0.8, 0.9, 0.95\}$	0.8
γ	$\{0.99\}$	0.99
MLP Layer Size	$\{128, 256, 512, 1024\}$	512
Learning Rate	$\{0.0002, 0.0003, 0.0004\}$	0.0003
Environment Workers	$\{16, 64, 256\}$	256
# Minibatches	$\{2, 4, 8, 16\}$	8
# Steps	$\{16, 64, 256\}$	16
# Update Epochs	$\{1, 2, 4, 8, 16\}$	4
Clip ϵ	$\{0.2\}$	0.2
Value Function Coefficient	$\{0.5\}$	0.5
Activation Function	$\{\text{tanh}, \text{relu}\}$	tanh
Anneal Learning Rate	$\{\text{True}\}$	True

Table 11. Hyperparameters for PPO for Craftax-1M.

Hyperparameter	Considered Values	Value
Intrinsic Reward Coefficient β	$\{0.01, 0.1, 1.0, 10.0, 100.0\}$	10.0
World Models Learning Rate	$\{0.0003\}$	0.0003
Forward Loss Coefficient	$\{0.1, 0.5, 1.0\}$	0.1
Inverse Loss Coefficient	$\{0.1, 0.5, 1.0\}$	0.5
World Models Layer Size	$\{256\}$	256
Latent Size	$\{16, 32, 64, 128\}$	16

Table 12. Hyperparameters for ICM for Craftax-1M.

Hyperparameter	Considered Values	Value
Intrinsic Reward Coefficient β	$\{0.0001, 0.001, 0.01, 0.1, 1.0\}$	0.01
Ridge Regulariser λ	$\{0.001, 0.01, 0.1, 0.5\}$	0.5

Table 13. Hyperparameters for E3B for Craftax-1M.