XLand-MiniGrid: Scalable Meta-Reinforcement Learning Environments in JAX

Alexander Nikulin * Tinkoff
Vladislav Kurenkov Tinkoff
Ilya Zisman Tinkoff
Artem Agarkov Tinkoff
Viacheslav Sinii Tinkoff
Sergey Kolesnikov Tinkoff

Abstract

We present XLand-MiniGrid, a suite of tools and grid-world environments for meta-reinforcement learning research inspired by the diversity and depth of XLand and the simplicity and minimalism of MiniGrid. XLand-Minigrid is written in JAX, designed to be highly scalable, and can potentially run on GPU or TPU accelerators, democratizing large-scale experimentation with limited resources. To demonstrate the generality of our library, we have implemented some well-known single-task environments as well as new meta-learning environments capable of generating 10^8 distinct tasks. We have empirically shown that the proposed environments can scale up to 2^{13} parallel instances on the GPU, reaching tens of millions of steps per second.

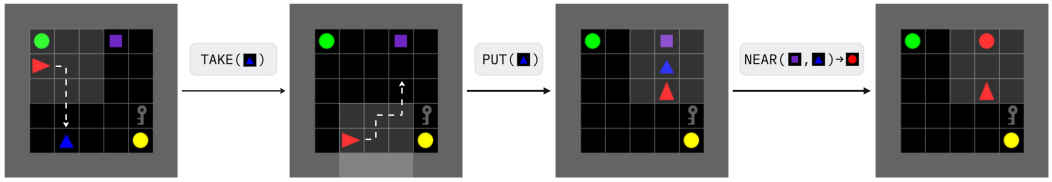


Figure 1: A visualization of how the production rules in XLand-MiniGrid work, exemplified by a few steps in the environment. On the first steps the agent picks up the blue pyramid, then places it next to the purple square. The `NEAR` production rule is then triggered, which transforms both objects into a new red circle. See Section 2.1 for a detailed description.

1 Introduction

Reinforcement learning is known to be extremely sample inefficient and prone to overfitting, sometimes failing to generalize to even subtle variations in environmental dynamics or goals [Rajeswaran et al., 2017, Zhang et al., 2018, Henderson et al., 2018]. One way to address these shortcomings can be a meta-RL approach, where adaptive agents are pre-trained on diverse task distributions to significantly increase the sampling efficiency on new problems [Wang et al., 2016, Duan et al., 2016]. With sufficient scaling and task diversity, this can lead to astonishing results, reducing the adaptation time on new problems to the human level and beyond [Team et al., 2021, 2023].

At the same time, meta-RL introduces a trade-off, significantly increasing the pre-training requirements at the expense of faster adaptation during inference, as the agent should experience thousands

*Correspondence to: <a.p.nikulin@tinkoff.ai>

Intrinsically Motivated Open-ended Learning Workshop at 37th Conference on Neural Information Processing Systems (NeurIPS 2023).

of different tasks for generalization. For example, a single training of the Ada agent [Team et al., 2023] requires five weeks, which is out of reach for most academic labs and practitioners. And even if they had the resources, the XLand environment is not publicly available. It seems to us, and has been pointed out by Wang et al. [2021], that such high requirements are the reason why most of the recent work on adaptive agents [Laskin et al., 2022, Lee et al., 2023, Lu et al., 2023, Norman and Clune, 2023] uses rather simplistic environments.

While simple environments are affordable and convenient for theoretical analysis, they do not allow researchers to discover the limits and scaling properties of proposed algorithms in practice. To make this possible, we continue the successful efforts [Bonnet et al., 2023, Koyamada et al., 2023, Freeman et al., 2021, Lu et al., 2022] at accelerating environments through JAX [Bradbury et al., 2018] and introduce XLand-MiniGrid, a library of grid world environments for meta-RL research. It does not compromise on task complexity in favour of affordability, democratizing large scale experimentation with limited resources.

2 XLand-MiniGrid

We present an initial release of **XLand-MiniGrid** (v0.0.1), a suit of tools and grid world environments for meta-RL research, inspired by the diversity and depth of XLand [Team et al., 2023, 2021] and by the simplicity and minimalism of MiniGrid [Chevalier-Boisvert et al., 2023].

Similar to XLand, we introduce a system of extendable rules and goals that can be combined in arbitrary ways to produce diverse distributions of dynamics and tasks (see Figure 1 for a demonstration). Similar to MiniGrid, we focus on a goal-oriented grid world environments and use a visual theme already well known in the community. However, despite the similarity, XLand-MiniGrid is written in the JAX framework from scratch and can therefore run directly on GPU or TPU accelerators, reaching millions of steps per second with a simple `jax.vmap` transformation. This makes it possible to use the Anakin architecture [Hessel et al., 2021] with the possibility to scale up to thousands of TPU cores.

This section provides a high-level overview of the library, describing the API, observation and action spaces, system of rules and goals. Examples of already implemented environments are also presented.

2.1 Rules and Goals

In XLand-MiniGrid, the system of rules and goals is the cornerstone of the emergent complexity and diversity. In the original MiniGrid [Chevalier-Boisvert et al., 2023] some environments have dynamic goals, but the dynamics are never changed. To train and evaluate highly adaptive agents, we need to be able to change the dynamics in non-trivial ways [Team et al., 2023].

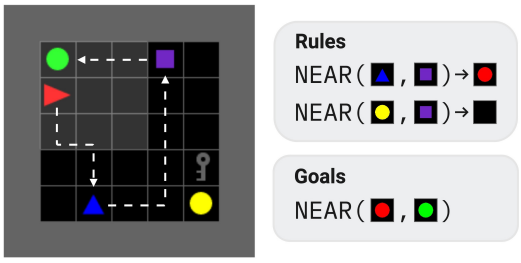


Figure 2: Visualization of a sampled `PutNear` environment rule set. This environment can generate at least 10^8 distinct tasks (see Appendix B.3). Additionaly, we highlight the optimal path to solve this particular environment. Agent should take blue pyramid and put it near the purple square to transform both objects into red circle. To complete the goal, red circle should be placed near green circle. See Figure 1 for additional details.

Rules. Rules are the functions that can change the environment state in some deterministic way according to the given conditions. As an example, `NEAR` rule (see Figure 1 for a visualization) accepts two objects `a`, `b` and transforms them to a new object `c` if `a` and `b` end up on neighbouring tiles. Rules can change between resets. For efficiency reasons, they are evaluated only after some actions or events (e.g. `NEAR` rule is checked only after `put_down` action).

Goals. Goals are similar to rules, except they do not change the state, they only test conditions. As an example, `NEAR` goal (see Figure 2 for a visualization) accepts two objects `a`, `b` and checks that they are on neighbouring tiles. Similar to the rules, goals are evaluated only after some actions and can change between resets.

To be able to change between resets and still be compatible with JAX both rules and goals are represented with an array encoding, where the first index states the rule or goal `ID` and the rest are arguments with optional padding to the same global length. Thus, every rule and goal should implement `encode` and `decode` methods. Environment state contains only these encodings, not the actual functions or classes. For the full list of rules and goals see Appendix B.2.

2.2 API

Environment Interface. Similar to Jumanji [Bonnet et al., 2023], XLand-MiniGrid `Environment` interface is inspired by the `dm_env` API [Muldal et al., 2019], which is particularly well suited for the meta-RL, as it separates episodes from trials by design (see Appendix D.1). Thus, each environment should provide jit-compatible `reset`, `reset_trial` and `step` methods. During the episode, `reset_trial` will be called several times when the trial terminates. For common use cases, this is handled automatically by the base class. For demonstration of how to instantiate and take actions in the multiple environments at once see code in Listing 1.

```python
import jax
import xland_minigrid

key = jax.random.PRNGKey(0)
reset_keys = jax.random.split(key, num=NUM_ENVS)
# initialize the environment
env = xland_minigrid.envs.meta.PutNear(num_trials=5, trial_max_steps=81)
# function to perform single step in the environment
def _step_fn(timestep, action):
    new_timestep = jax.vmap(env.step)(timestep, action)
    return new_timestep, None
# sample actions for the rollout
actions = jax.random.randint(key, shape=(TIMESTEPS, NUM_ENVS), maxval=6)
# reset NUM_ENVS environments
timestep = jax.vmap(env.reset)(key=reset_keys)
# vectorized rollout for TIMESTEPS steps
last_timestep, _ = jax.lax.scan(_step_fn, timestep, actions)
```

Listing 1: Sample code demonstrating how to instantiate and perform actions in the multiple environments at once using `jax.vmap` transformation. Note that this code can be freely combined with the `jax.jit` transformation, but in practice it should be applied to the entire training loop for maximum performance.

State. The `State` contains all the necessary information to describe the dynamics of the environment, such as rules, goals, grid and agent states. This is necessary for `step` and `reset` to be compatible with JAX transformations, since they must not have side effects. `State` can also contain arbitrary pytree[2] as a carry, to pass information between trials. This can be useful for procedural generation, for example, as objects sampled at the start of the episode can be stored and only their positions randomised during trial resets. For sampling, each state contains a pseudorandom number generator key that can be used during resets.

TimeStep. The `TimeStep` contains all the information available to the agent, such as `observation`, `reward`, `step_type` and `discount`, where the latter two being inherited from the `dm_env`[3] API. The step type will be FIRST at the beginning of the episode, LAST on the last step and MID on all others. Discount can be in the range $[0, 1]$ and we set it to 0.0 to indicate the end of the trial.

Observation and Action Space. We tried not to deviate too much from the original MiniGrid. Observations are four-dimensional arrays, where each position is encoded by tile and color ids. Thus, observations are not images and should not be treated as such by default. While naively treating them as images will work in most cases, the correct approach would be to pre-process them via embeddings. We also support the ability to prohibit an agent from seeing through walls. The actions are discrete, namely `move_forward`, `turn_left`, `turn_right`, `pick_up`, `put_down`, `toggle`. The agent can only pick up one item, and only if its pocket is empty.

[2] `https://jax.readthedocs.io/en/latest/pytrees.html`

[3] `https://github.com/google-deepmind/dm_env/blob/master/docs/index.md`

2.3 Supported environments

Single-task environments. Due to the generality of rules and goals most of the non-language based tasks from MiniGrid can be easily implemented in XLand-MiniGrid. To demonstrate this, we have ported several popular environments (see Figure 6) such as `Empty`, `FourRooms`, `UnlockPickUp`, `DoorKey`. For example, `Empty` and `FourRooms` are specified with `AgentOnEntityTile` goal, where target tile is green goal. `UnlockPickUp` is specified with `AgentHold` goal with the target object to pick up.

Meta-learning environments. In the initial release we implemented four meta-learning environments, namely `DarkRoom`, `FourGoals`, `PickUp`, `PutNear`. For brevity we will describe only PutNear here (see Figure 2). The goal of this environment is to put near two objects, one of which is not present in initial grid and must be produced using `NEAR` rule. However, there is another `NEAR` rule with this object, which will transition environment to the unsolvable state. Thus, agent should learn to avoid it during trials. Positions of objects are randomized between trials, while objects itself are randomized between episodes. This environment can generate at least 10^8 distinct tasks (see Appendix B.3). Such large number of tasks is important for studying scaling and generalization [Team et al., 2023]. For future planned meta-RL environments see Section 4.

3 Experiments

In this section we demonstrate the XLand-Minigrid ability to scale to thousands of parallel environments, dozens of rules and various grid sizes. All the measurements were done on a single A100 GPU. For PPO hyperparameters see Appendix C.

Simulation throughput. We compare simulation throughput for single-task (see Figure 3) and meta-learning environments (see Figure 4a). For single-tasks environments we consider random policy and PPO. As can be seen, compared to the commonly used MiniGrid [Chevalier-Boisvert et al., 2023] environments with gymnasium [Towers et al., 2023] asynchronous vectorization, XLand-Minigrid achieves at least 10x faster throughput reaching tens of millions of steps per second. This result also holds for PPO training, showing that we can achieve a significant reduction in total training time for a fixed number of transitions. However, saturation is observed for PPO after 2048 environments, so further scaling is probably not beneficial due to the diminishing returns (at least on a single GPU). Testing on the TPU as well as on multiple GPUs is left for future work.

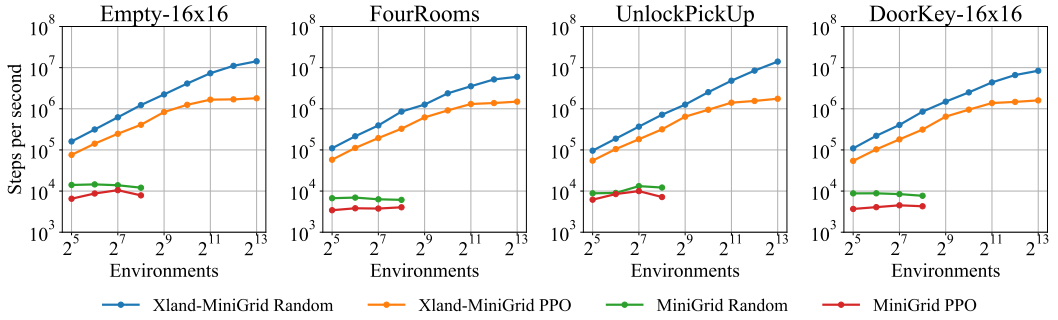
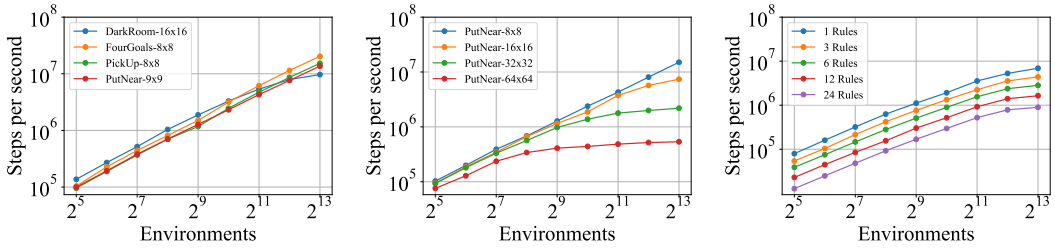


Figure 3: Simulation throughput for XLand-MiniGrid and MiniGrid.

Scaling grid size. While most of the grid world environments from MiniGrid Chevalier-Boisvert et al. [2023] use small grid sizes, it is still of interest to test XLand-MiniGrid scaling properties among this dimension. As can be seen on the Figure 4b, `PutNear` throughput can degrade significantly with the increased grid size, also showing earlier signs of the saturation. The explanation for this is that many game loop operations, such as conditional branching during action selection, do not fit well into the parallelism principles underlying JAX framework. Similar results we observed in the previous work on JAX-based environments [Bonnet et al., 2023, Koyamada et al., 2023]. Nevertheless, the throughput remains competitive even at large sizes. Furthermore, small grid sizes can still be a significant challenge for existing algorithms [Zhang et al., 2020].

Scaling number of rules. Full-scale XLand environment can use more than five rules according to the Team et al. [2023]. To test XLand-MiniGrid in similar conditions we report simulation throughput



(a) Meta-learning environments (b) Scaling grid size (c) Scaling number of rules

Figure 4: Simulation throughput for meta-learning environments, as well as for different grid sizes and different numbers of rules.

varying number of rules. For testing purposes we just replicated same `NEAR` rule multiple times in the `PutNear` environment. As can be seen in Figure 4c, throughput monotonically decreases. As a result, the number of parallel environments has to be increased in order to maintain the same level of throughput. However, in contrast to the increase in grid size, there is no apparent saturation even for 24 rules.

4 Limitations and Future Work

There are several notable limitations to our work, some of which we expect to address in future releases of our library.

Compared to the full-scale XLand [Team et al., 2021, 2023], we do not currently support multi-agent simulations, procedural generation of complex worlds, rules with multiple output entities, or goal composition. Our initial environments use simple multi-room grid worlds and small sets of predefined rules (see Section 2.3). However, we plan to release a benchmark that approximates the procedural generation of XLand-v2.0 rule sets (see Appendix A.2 in Team et al. [2023]) as well as the procedural generation of different maze layouts.

Compared to the MiniGrid [Chevalier-Boisvert et al., 2023], we do not yet support all tiles such as lava or moving obstacles. Also, as our focus is on meta-RL, we do not provide explicit natural language encoding of rules and goals, although this can easily be done. For a language focused learning environment, we refer the reader to the recent HomeGrid [Lin et al., 2023] environment. Although we do not support all existing MiniGrid environments in the initial release, we provide enough tools to make it easy to implement them if needed (and have demonstrated this in the Section 2.3).

In general, while JAX is much more high-level than CUDA or Triton [Tillet et al., 2019], it is still much more restrictive than PyTorch [Paszke et al., 2019], can be difficult to debug, and is poorly suited to the heterogeneous computations or conditional branching that are common when implementing environments. However, as we show in our work, when used correctly it can provide excellent scalability opportunities.

References

I. Adamski, R. Adamski, T. Grel, A. Jędrych, K. Kaczmarek, and H. Michalewski. Distributed deep reinforcement learning: Learn how to play atari games in 21 minutes. In *High Performance Computing: 33rd International Conference, ISC High Performance 2018, Frankfurt, Germany, June 24-28, 2018, Proceedings 33*, pages 370–388. Springer, 2018.

C. Bamford, S. Huang, and S. Lucas. Griddly: A platform for ai research in games, 2020.

C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

C. Bonnet, D. Luo, D. Byrne, S. Surana, V. Coyette, P. Duckworth, L. I. Midgley, T. Kalloniatis, S. Abramowitz, C. N. Waters, et al. Jumanji: a diverse suite of scalable reinforcement learning environments in jax. *arXiv preprint arXiv:2306.09884*, 2023.

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

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

Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR, 2018.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

A. W. Hanjie, V. Y. Zhong, and K. Narasimhan. Grounding language to entities and dynamics for generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 4051–4062. PMLR, 2021.

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

M. Hessel, M. Kroiss, A. Clark, I. Kemaev, J. Quan, T. Keck, F. Viola, and H. van Hasselt. Podracer architectures for scalable reinforcement learning. *arXiv preprint arXiv:2104.06272*, 2021.

D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2018. URL `https://api.semanticscholar.org/CorpusID:59345798`.

S. Koyamada, S. Okano, S. Nishimori, Y. Murata, K. Habara, H. Kita, and S. Ishii. Pgx: Hardware-accelerated parallel game simulation for reinforcement learning. *arXiv preprint arXiv:2303.17503*, 2023.

M. Laskin, L. Wang, J. Oh, E. Parisotto, S. Spencer, R. Steigerwald, D. Strouse, S. Hansen, A. Filos, E. Brooks, et al. In-context reinforcement learning with algorithm distillation. *arXiv preprint arXiv:2210.14215*, 2022.

J. N. Lee, A. Xie, A. Pacchiano, Y. Chandak, C. Finn, O. Nachum, and E. Brunskill. Supervised pretraining can learn in-context reinforcement learning. *arXiv preprint arXiv:2306.14892*, 2023.

Z. Li, T. Chen, Z.-W. Hong, A. Ajay, and P. Agrawal. Parallel q-learning: Scaling off-policy reinforcement learning under massively parallel simulation. 2023.

J. Lin, Y. Du, O. Watkins, D. Hafner, P. Abbeel, D. Klein, and A. Dragan. Learning to model the world with language. *arXiv preprint arXiv:2308.01399*, 2023.

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

C. Lu, Y. Schroecker, A. Gu, E. Parisotto, J. Foerster, S. Singh, and F. Behbahani. Structured state space models for in-context reinforcement learning. *arXiv preprint arXiv:2303.03982*, 2023.

V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

A. Muldal, Y. Doron, J. Aslanides, T. Harley, T. Ward, and S. Liu. dm_env: A python interface for reinforcement learning environments, 2019. URL `http://github.com/deepmind/dm_env`.

A. Nikulin, V. Kurenkov, D. Tarasov, D. Akimov, and S. Kolesnikov. Q-ensemble for offline rl: Don't scale the ensemble, scale the batch size. *arXiv preprint arXiv:2211.11092*, 2022.

B. Norman and J. Clune. First-explore, then exploit: Meta-learning intelligent exploration. *arXiv preprint arXiv:2307.02276*, 2023.

F. Paischer, T. Adler, V. Patil, A. Bitto-Nemling, M. Holzleitner, S. Lehner, H. Eghbal-Zadeh, and S. Hochreiter. History compression via language models in reinforcement learning. In *International Conference on Machine Learning*, pages 17156–17185. PMLR, 2022.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

A. Petrenko, Z. Huang, T. Kumar, G. Sukhatme, and V. Koltun. Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning. In *International Conference on Machine Learning*, pages 7652–7662. PMLR, 2020.

A. Petrenko, E. Wijmans, B. Shacklett, and V. Koltun. Megaverse: Simulating embodied agents at one million experiences per second. In *International Conference on Machine Learning*, pages 8556–8566. PMLR, 2021.

A. Rajeswaran, K. Lowrey, E. V. Todorov, and S. M. Kakade. Towards generalization and simplicity in continuous control. *Advances in Neural Information Processing Systems*, 30, 2017.

B. Shacklett, E. Wijmans, A. Petrenko, M. Savva, D. Batra, V. Koltun, and K. Fatahalian. Large batch simulation for deep reinforcement learning. *arXiv preprint arXiv:2103.07013*, 2021.

B. Shacklett, L. G. Rosenzweig, Z. Xie, B. Sarkar, A. Szot, E. Wijmans, V. Koltun, D. Batra, and K. Fatahalian. An extensible, data-oriented architecture for high-performance, many-world simulation. *ACM Trans. Graph.*, 42(4), 2023.

B. C. Stadie, G. Yang, R. Houthooft, X. Chen, Y. Duan, Y. Wu, P. Abbeel, and I. Sutskever. Some considerations on learning to explore via meta-reinforcement learning. *arXiv preprint arXiv:1803.01118*, 2018.

A. Stooke and P. Abbeel. Accelerated methods for deep reinforcement learning. *arXiv preprint arXiv:1803.02811*, 2018.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

D. Tarasov, V. Kurenkov, A. Nikulin, and S. Kolesnikov. Revisiting the minimalist approach to offline reinforcement learning. *arXiv preprint arXiv:2305.09836*, 2023.

A. A. Team, J. Bauer, K. Baumli, S. Baveja, F. Behbahani, A. Bhoopchand, N. Bradley-Schmieg, M. Chang, N. Clay, A. Collister, et al. Human-timescale adaptation in an open-ended task space. *arXiv preprint arXiv:2301.07608*, 2023.

O. E. L. Team, A. Stooke, A. Mahajan, C. Barros, C. Deck, J. Bauer, J. Sygnowski, M. Trebacz, M. Jaderberg, M. Mathieu, et al. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 2021.

P. Tillet, H. T. Kung, and D. Cox. Triton: An intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2019, page 10–19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367196. doi: 10.1145/3315508.3329973. URL `https://doi.org/10.1145/3315508.3329973`.

M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis. Gymnasium, Mar. 2023. URL `https://zenodo.org/record/8127025`.

J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

J. X. Wang, M. King, N. Porcel, Z. Kurth-Nelson, T. Zhu, C. Deck, P. Choy, M. Cassin, M. Reynolds, F. Song, et al. Alchemy: A benchmark and analysis toolkit for meta-reinforcement learning agents. *arXiv preprint arXiv:2102.02926*, 2021.

J. Weng, M. Lin, S. Huang, B. Liu, D. Makoviichuk, V. Makoviychuk, Z. Liu, Y. Song, T. Luo, Y. Jiang, et al. Envpool: A highly parallel reinforcement learning environment execution engine. *Advances in Neural Information Processing Systems*, 35:22409–22421, 2022.

Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pages 1094–1100. PMLR, 2020.

A. Zhang, N. Ballas, and J. Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018.

T. Zhang, H. Xu, X. Wang, Y. Wu, K. Keutzer, J. E. Gonzalez, and Y. Tian. Bebold: Exploration beyond the boundary of explored regions. *arXiv preprint arXiv:2012.08621*, 2020.

A. Zholus, A. Skrynnik, S. Mohanty, Z. Volovikova, J. Kiseleva, A. Szlam, M.-A. Coté, and A. I. Panov. Iglu gridworld: Simple and fast environment for embodied dialog agents. *arXiv preprint arXiv:2206.00142*, 2022.

L. Zintgraf, K. Shiarlis, M. Igl, S. Schulze, Y. Gal, K. Hofmann, and S. Whiteson. Varibad: A very good method for bayes-adaptive deep rl via meta-learning. *arXiv preprint arXiv:1910.08348*, 2019.

A Related Work

Meta-learning environments. Historically, benchmarks for meta-reinforcement learning have focused on tasks with simple distributions, such as bandits, 2D navigation [Wang et al., 2016, Duan et al., 2016, Finn et al., 2017], or few-shot adaptation of control policies (e.g. MuJoCo [Zintgraf et al., 2019] or MetaWorld [Yu et al., 2020]), where the latent component is reduced to a few parameters that control goal location or changes in robot morphology. The recent wave of research on in-context reinforcement learning [Laskin et al., 2022, Lee et al., 2023] also uses simple environments for evaluation, such as the navigational DarkRoom, KeyToDoor, or MuJoCo with random projections of observations [Lu et al., 2023]. Some notable exceptions are XLand [Team et al., 2021, 2023] and Alchemy [Wang et al., 2021], but XLand is not open source and Alchemy is built on top of Unity (www.unity.com) and runs at 30 FPS, which is not ideal.

We hypothesize that the popularity of such simple benchmarks is attributed to their affordability, as meta-training can require an order of magnitude more environmental transitions than single-task RL. However, this prevents researchers from uncovering the limits and scaling properties of the proposed methods. We believe that the solution is an environment that does not compromise interestingness and task complexity for the sake of affordability. This is why we designed XLand-MiniGrid to take the best of the XLand and Alchemy environments without sacrificing speed and scalability thanks to the JAX ecosystem.

Grid world environments. Grid world environments have a long history in RL research Sutton and Barto [2018] as they possess a number of attractive properties. They are typically easy to implement, do not require large computational resources, have simple observation spaces, and yet pose a significant challenge even to modern RL methods, allowing the testing of exploration [Zhang et al., 2020], language understanding and generalization [Zholus et al., 2022, Lin et al., 2023, Chevalier-Boisvert et al., 2023, Hanjie et al., 2021], or memory [Paischer et al., 2022].

Despite the greater benefits variety of the existing grid world benchmarks, to our knowledge only the KrazyWorld [Stadie et al., 2018] focuses on meta-learning, and unfortunately it is no longer maintained. Other libraries, such as the popular MiniGrid [Chevalier-Boisvert et al., 2023] and Griddly [Bamford et al., 2020], are not scalable and extensible enough to cover meta-learning needs. In this work, we have attempted to address these needs with new grid world environments in the minimalist style of MiniGrid. However, XLand-MiniGrid is completely written in JAX, which allows it to scale to millions of steps per second on the single GPU.

Large-batch RL. Large batches are known to be beneficial in deep learning [You et al., 2019] and deep reinforcement learning is no exception. It is known to be extremely sample inefficient and large batches can increase experience throughput accelerating convergence and improving training stability. For example, many of the early breakthroughs on the Atari benchmark were driven by more efficient distributed experience collection [Horgan et al., 2018, Espeholt et al., 2018, Kapturowski et al., 2018], eventually decreasing training time to only a few minutes [Stooke and Abbeel, 2018, Adamski et al., 2018] per game. Increasing the mini-batch size can also be beneficial in the offline RL [Nikulin et al., 2022, Tarasov et al., 2023].

However, not all algorithms scale equally well, and off-policy methods have been lagging behind until recently [Li et al., 2023], whereas on-policy methods, while generally less sample efficient, can scale to enormous batch sizes of millions [Berner et al., 2019, Petrenko et al., 2020], completing training much faster [Stooke and Abbeel, 2018, Shacklett et al., 2021] in wall clock time. While we do not introduce novel algorithmic improvements in our work, we hope that the proposed highly scalable XLand-MiniGrid environments will help practitioners conduct meta-reinforcement learning experiments at scale faster and with fewer resources.

Hardware-accelerated environments. There are several approaches to increasing the throughput of environment experience. The most general approach would be to write environment logic in low-level languages (to bypass Python GIL) for asynchronous collection, as EnvPool Weng et al. [2022] does. However, it does not eliminate the bottleneck of data transfer between CPU and GPU/TPU on every iteration, and debugging asynchronous systems is difficult. Porting the entire environment to the GPU (like Isaac Gym [Makoviychuk et al., 2021], Megaverse [Petrenko et al., 2021] or Madrona [Shacklett et al., 2023]) can drastically increase training time by removing the bottleneck, but has the disadvantage of being GPU-only.

Recently, new environments written entirely in JAX Bradbury et al. [2018] have appeared, taking advantage of the GPU or TPU and the ability to just-in-time compile the entire training loop, further reducing the total training time. However, most of them focus on single-task environments for robotics [Freeman et al., 2021], board games [Koyamada et al., 2023] or combinatorial optimization [Bonnet et al., 2023]. The proposed XLand-MiniGrid continues this path by adding the JAX-based environments for meta-reinforcement learning, which are currently missing.

B Environments Details

B.1 Tiles and Colors

(a) Supported Tiles.

Tile	ID
END_OF_MAP	0
UNSEEN	1
EMPTY	2
FLOOR	3
WALL	4
BALL	5
SQUARE	6
PYRAMID	7
GOAL	8
KEY	9
DOOR_LOCKED	10
DOOR_CLOSED	11
DOOR_OPEN	12

(b) Supported Colors.

Color	ID
END_OF_MAP	0
UNSEEN	1
EMPTY	2
RED	3
GREEN	4
BLUE	5
PURPLE	6
YELLOW	7
GREY	8
BLACK	9
ORANGE	10
WHITE	11

B.2 Rules and Goals

For efficiency, we separate rules and goals with the same meaning that apply only to entities or to the agent, since internally the agent is not considered a valid entity. For example, the rules `AgentNear` and `EntityNear` are separated.

Table 2: Supported Goals.

Rule	Meaning	ID
`Empty`	Placeholder goal, returns False	0
`AgentHold(a)`	Whether agent holds `a`	1
`AgentOnTile(a)`	Whether agent is on tile `a`	2
`AgentNear(a)`	Whether agent and `a` are on neighboring tiles	3
`EntityNear(a, b)`	Whether `a` and `b` are on neighboring tiles	4
`AgentOnPosition(x, y)`	Whether agent is on `(x, y)` position	5
`EntityOnPosition(x, y)`	Whether entity is on `(x, y)` position	6

Table 3: Supported Rules.

Rule	Meaning	ID
`Empty`	Placeholder rule, does not change anything	0
`AgentHold(a)` $\rightarrow$ `c`	If agent hold `a` replaces it with `c`	1
`AgentNear(a)` $\rightarrow$ `c`	If agent is on neighboring tile with `a` replaces it with `c`	2
`EntityNear(a, b)` $\rightarrow$ `c`	If `a` and `b` are on neighboring tiles, replaces one with `c` and removes the other	3

B.3 Number of PutNear Possible Tasks

Here we describe our reasoning for estimating the total number of unique tasks possible in the `PutNear` environment (see Figure 2 and Section 2.3). For this environment there are 7 valid colors and 4 entities, for a total of 28 valid objects. We sample 6 out of 28 without repetition (5 to place on the grid, 1 for the goal as it is not initially present on the grid). Next, we have to sample 2 out of 6 to determine the target (the order is important here). Then we sample 2 more from the remaining 4 to

determine the first `NEAR` rule, and 1 from the last 2 objects to determine the last `NEAR` rule. In total we get

$$\binom{28}{6}\binom{6}{2}2!\binom{4}{2}\binom{2}{1} = 376740 \cdot 15 \cdot 2 \cdot 6 \cdot 2 = 135,626,400$$

which is order of 10^8 and without taking into account starting positions randomization.

C Hyperparameters

Table 4: PPO hyperparameters used in the Section 3 experiments. We adapted our PPO implementation from PureJaxRL library [Lu et al., 2022].

Parameter	**Value**
num envs	See Section 3
num steps	64
num epochs	4
num minibatches	4
total timesteps	20000000
learning rate	2.5e-4
clip eps	0.2
gamma	0.99
gae lambda	0.95
entropy coef	0.001
value function coef	0.5
max grad norm	0.5
optimizer	Adam

D Additional Figures

D.1 Meta-RL Process

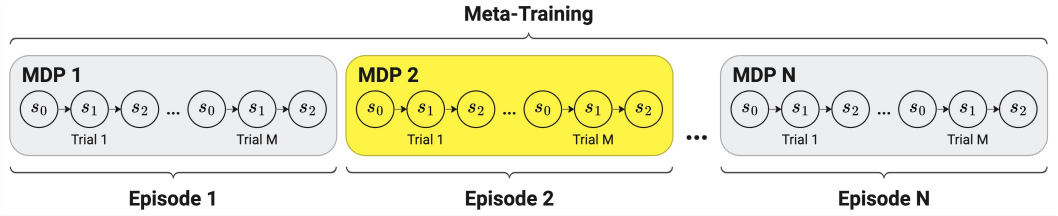
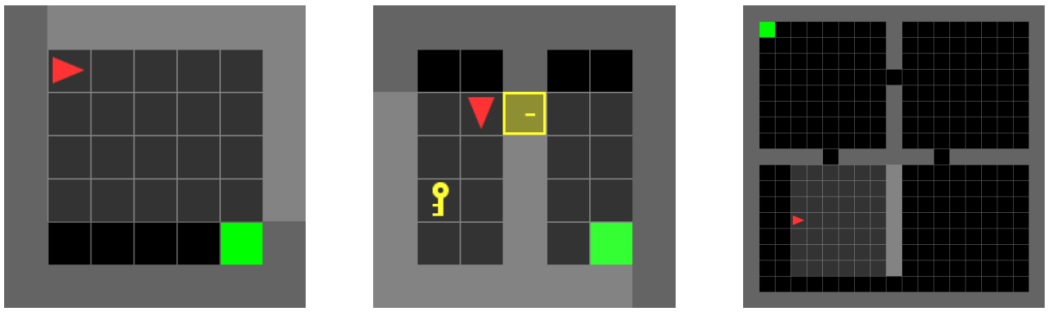


Figure 5: Agent-environment interaction during meta-reinforcement learning. At the beginning of each episode new task is sampled and the agent is given M trials to solve it. Notably, for memory-based agents [Wang et al., 2016, Duan et al., 2016], memory is reset only at the end of the episodes.

D.2 MiniGrid Environments



(a) Empty (b) DoorKey (c) FourRooms

Figure 6: Visualisation of several environments ported from MiniGrid.