Jumanji: a Diverse Suite of Scalable Reinforcement Learning Environments in JAX

Clément Bonnet et al. [1],

[1]InstaDeep Ltd. [2]University of Cambridge. [3]Imperial College London.

Abstract

Modern Reinforcement Learning (RL) research requires simulated environments that are performant, scalable, and modular to be applicable to a wide range of potential real-world applications. Therefor, we present Jumanji – a suite of diverse RL environments specifically designed to be fast, flexible, and scalable that focuses on combinatorial problems frequently encountered in industry, as well as challenging general decision-making tasks. By leveraging the efficiency of JAX and hardware accelerators like GPUs and TPUs, Jumanji enables rapid iteration of research ideas and large-scale experimentation, ultimately empowering more capable agents. Unlike existing RL environment suites, Jumanji is highly customizable, allowing users to tailor the initial state distribution and problem complexity to their needs. Furthermore, we provide actor-critic baselines for each environment, accompanied by preliminary findings on scaling and generalization scenarios.

Jumanji

# 1 Current release

Jumanji current release (v0.3.0), consist of 18 JAX based environments visualised in Figure 1. The environments consist of NP-hard combinatorial and decision making problems. These diverse problems rely on a variety of geometries, including grids, graphs, and sets, are broadly divided into 3 categories namely *logic*, *routing* and *packing* problems.

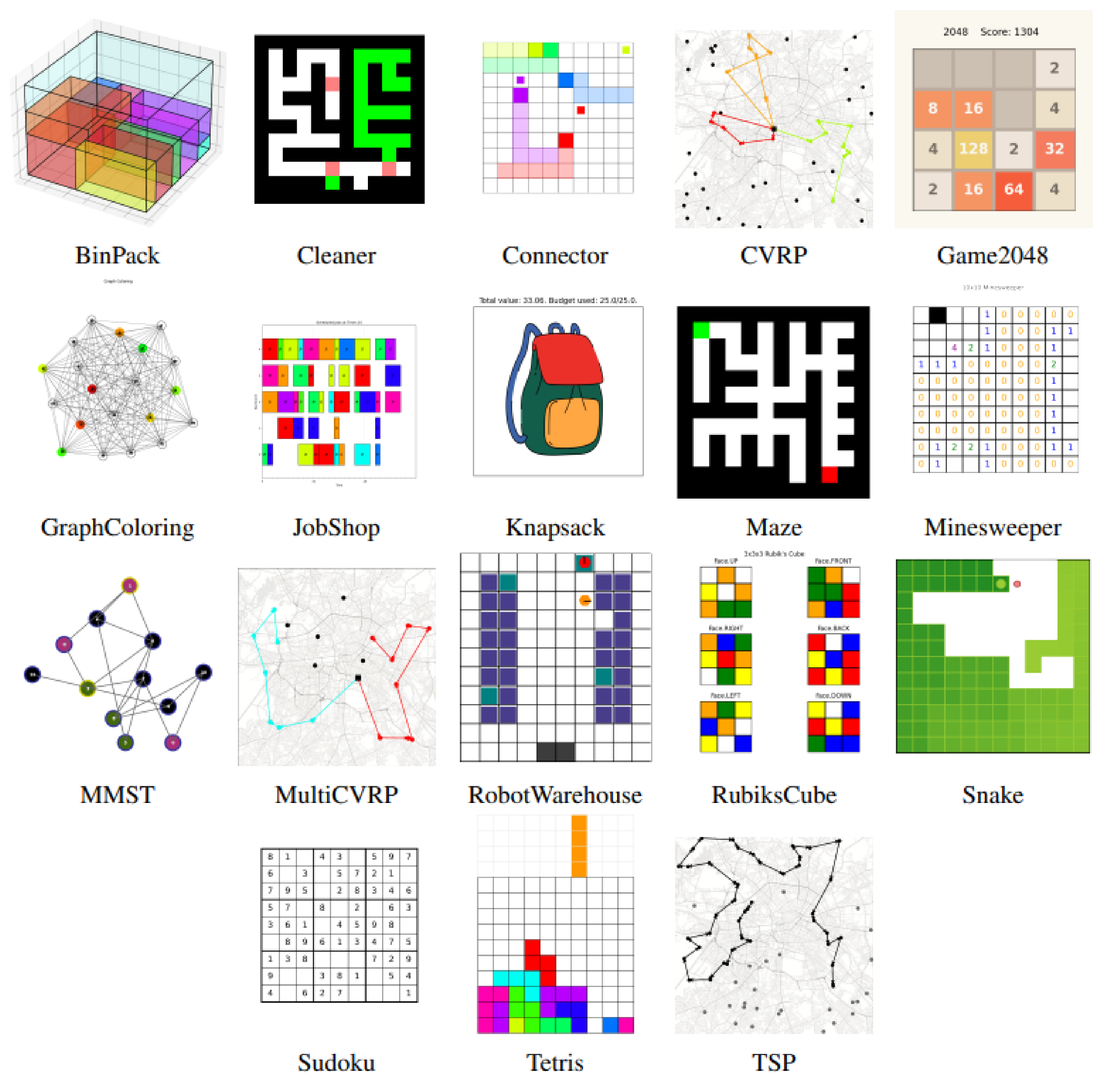


Figure 1. All 18 environments implemented in Jumanji.

# 2 API

Each environment must contain the following methods: reset, step, observation_spec, and action_spec. The API allows for optional render and animate methods to visualise a state or a sequence of states. To fully take advantage of JAX acceleration routines, the reset and step functions in Jumanji are pure and stateless. Figure 2 is snippet code of how to instantiated a Jumanji environment.

```python
import jax
import jumanji

# Instantiate a Jumanji environment from the registry
env = jumanji.make('Snake-v1')

# Reset the environment
key = jax.random.PRNGKey(0)
state, timestep = jax.jit(env.reset)(key)

# Sample an action and take an environment step
action = env.action_spec().generate_value()
state, timestep = jax.jit(env.step)(state, action)

# (Optional) Render the environment state
env.render(state)
```

Figure 2. Jumanji environment interface.

# 3 Environment flexibility

Generators are used to define the initial state distribution of environments. Jumanji allows the use of custom generators, enabling users to define an initial state distribution specific to their problem. The user can choose from a set of preexisting generators or implement their own generator.

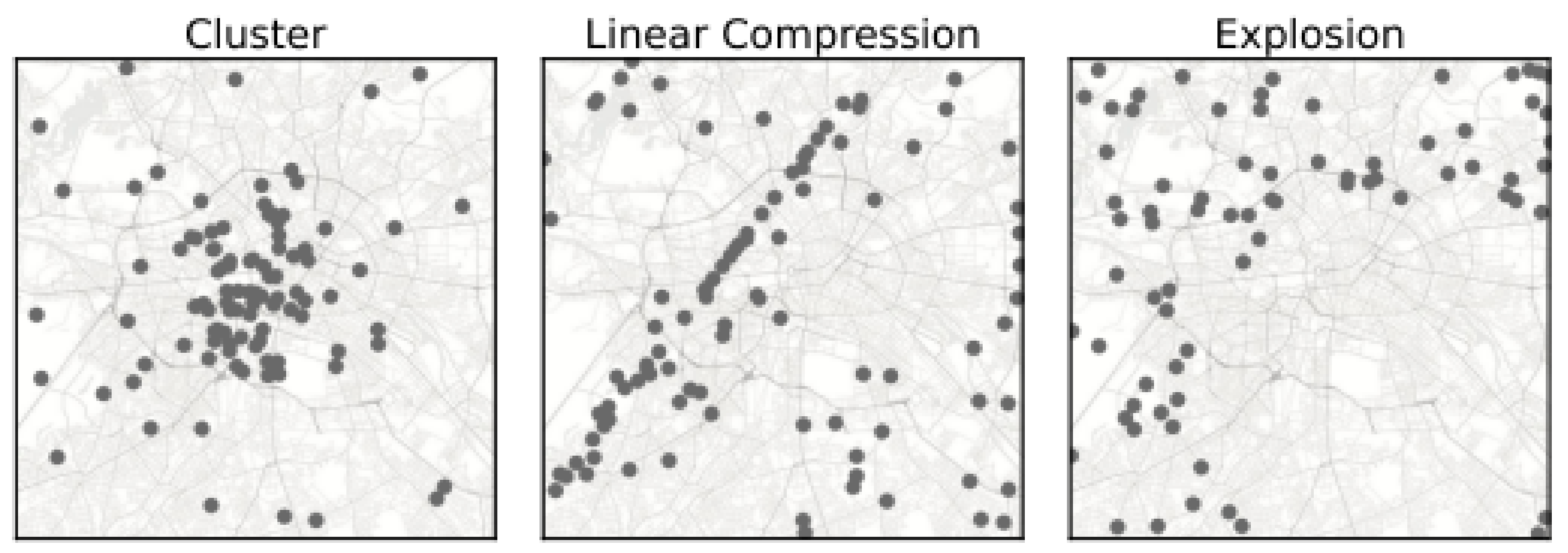


Figure 3. Different generators for the Travelling Salesman Problem (TSP)

Experiments

In this section, we presents some of the benchmark results of the experiments performed on all the environments. For training we use the Actor-Critic algorithm. Each environment's architecture depends on its observation geometry (CNN for images and Transformers for permutation invariant observations).

# 1 Actor-Critic Baseline

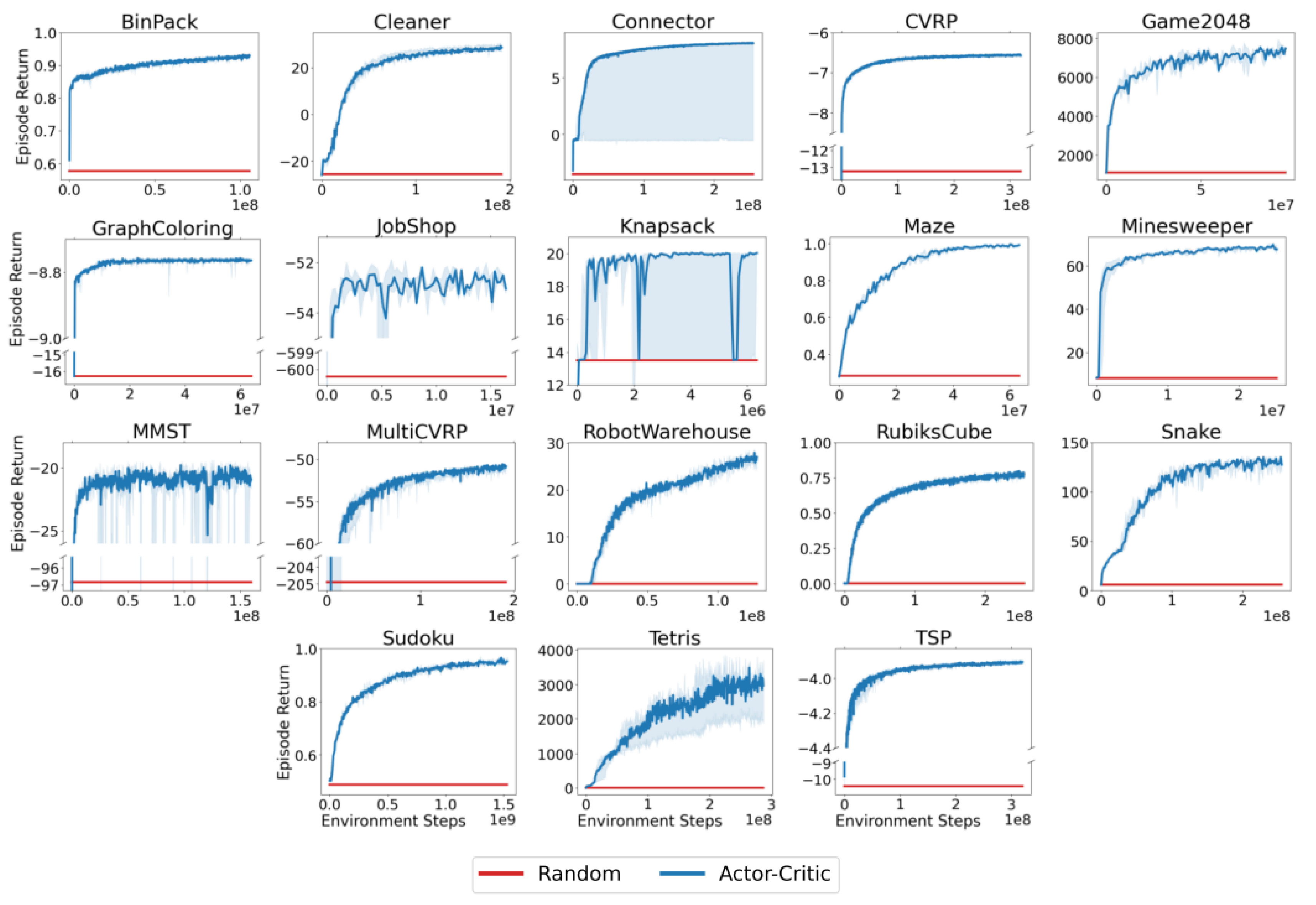
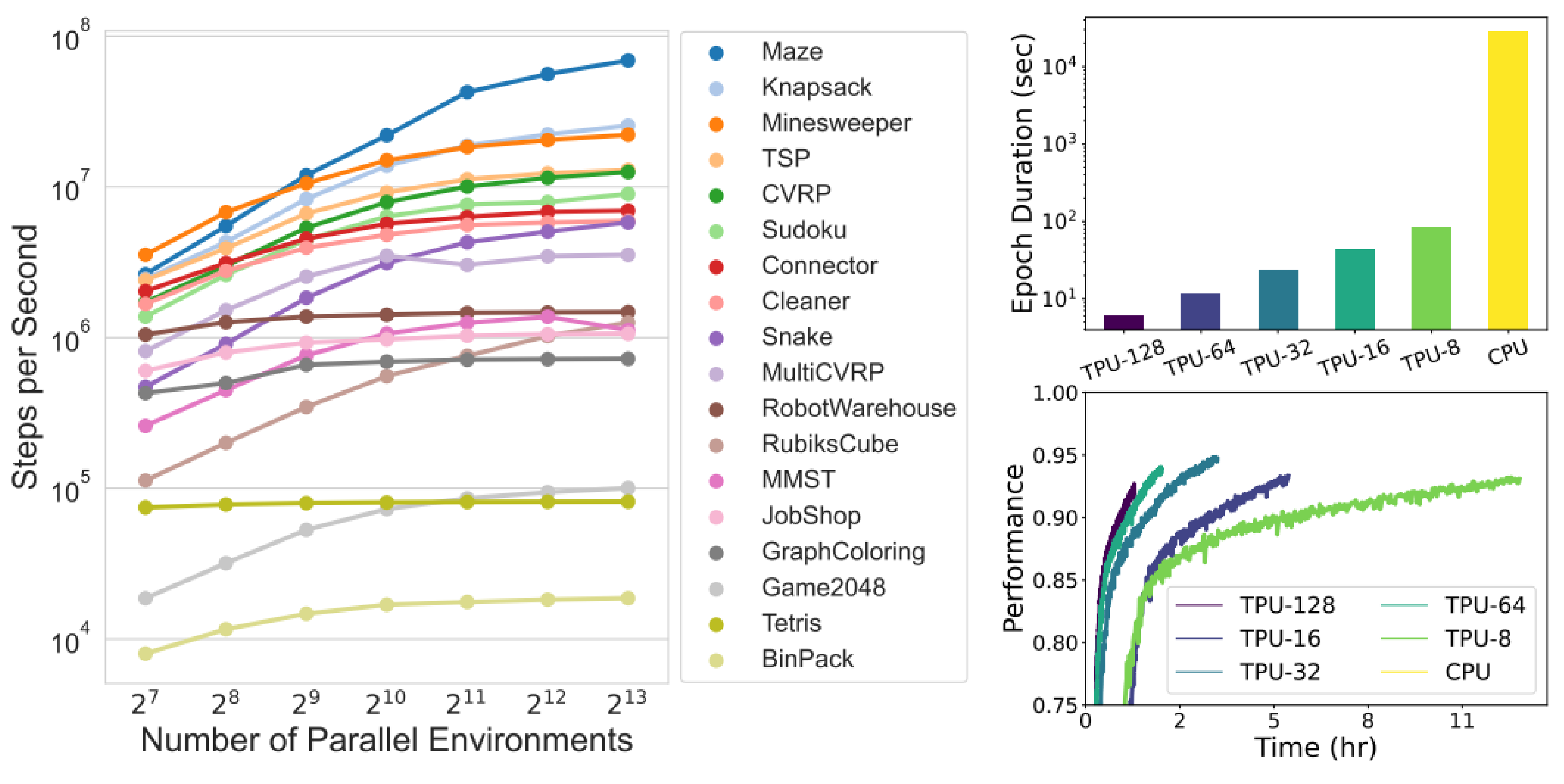


Figure 4. Learning curves from training an actor-critic agent (A2C) in blue compared to a masked random policy in red on all 18 Jumanji environments. Experiments were run with three different seeds, with the median represented as a blue curve and the min/max as the shaded region.

# 2 Environment parallelization

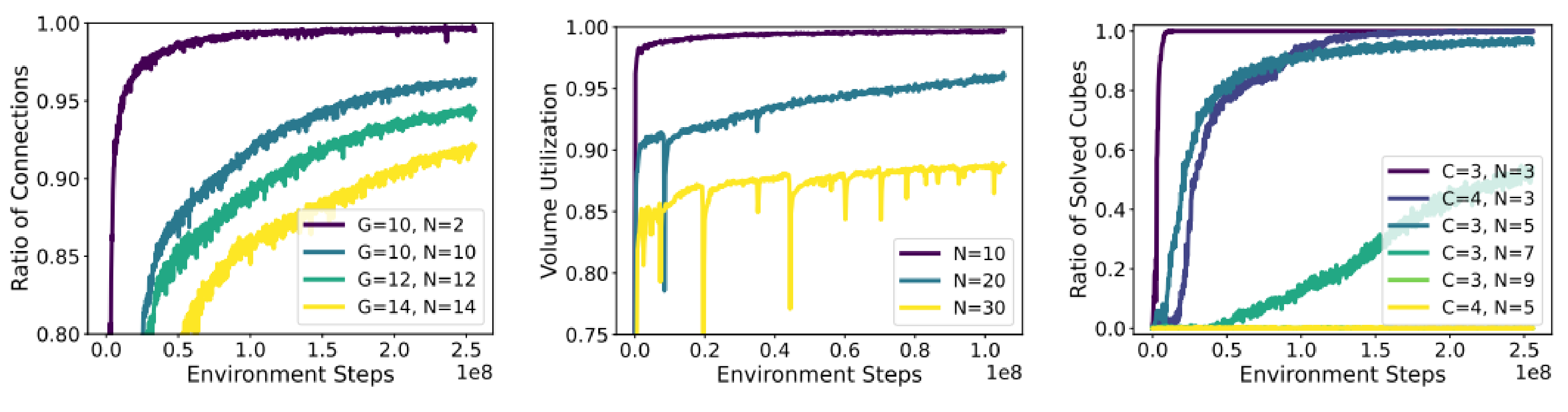


(a) Effective environment steps per second

(b) A2C training on Connector

Figure 5. Analysis of environment parallelization. (a): Scaling of the effective environment steps per second for each registered environment as the number of parallel environments increases, on an 8-core TPU-v4. The legend is ordered by the throughput of the rightmost data point. (b): Training of an A2C agent on the Connector environment on a CPU and TPU-v4 with a number of cores varying from 8 to 128. Each training is run for 255M steps. Performance denotes the proportion of wires connected (an optimal policy would reach 1.0).

# 3 Environment scalability



(a) Connector

(b) BinPack

(c) RubiksCube

Figure 6. The learning curves of the A2C agent on varying difficulty levels in three different Jumanji environments. In Connector, the size of the grid (G) and the number of node pairs to be connected (N) are varied. In BinPack, the number of items (N) is varied. In RubiksCube, the size of the cube (C) and the number of scrambles made from a solved Rubiks Cube (N) are varied.

Conclusion

We introduce Jumanji, an open-source and diverse suite of industry-inspired RL environments that are fast, flexible, and scalable. Written in JAX, Jumanji environments can be parallelized and seamlessly scale with hardware. Furthermore, Jumanji provides for flexibility with the use of custom environment generators for different types of problems with various levels for difficulties. Currently Jumanji covers different NP-hard industry inspired combinatorial problems.

Full authors list

Clement Bonnet, Daniel Luo, Donal Byrne, Shikha Surana, Vincent Coyette, Paul Duckworth, Laurence I. Midgley (2), Tristan Kalloniatis, Sasha Abramowitz, Cemlyn N. Waters, Andries P. Smith, Nathan Grinsztajn, Ulrich A. Mbou Sob, Omayma Mahjoub, Elshadai Tegegn, Mohamed A. Mimouni, Raphael Boige, Ruan de Kock, Daneil Furelos-Blanco (3), Victo Le, Arnu Pretorius, and, Alexandre Laterre.