Cogment Lab: A Practical Toolkit for Human-in-the-Loop RL Research

Ariel Kwiatkowski

Cogment Lab: A Practical Toolkit for Human-in-the-Loop RL Research

Ariel Kwiatkowski
AI Redefined
ariel@ai-r.com

Abstract

Human-in-the-loop learning is a key aspect of ensuring a positive future for the interactions between AI systems and humans. Despite that, the tooling for this line of research is often incomplete or inaccessible, creating a significant obstacle in this field. In this work, we introduce Cogment Lab, a researcher's toolkit for reinforcement learning experiments with the involvement of humans. It is a layer of abstraction on top of the already existing Cogment, which proves to be powerful, but difficult to use. In contrast, Cogment Lab preserves most of Cogment's flexibility, but making it significantly easier to use for practical research and development. We describe the design philosophy of Cogment Lab, some elements of its implementation, as well as research directions that it enables. We hope that this library will accelerate human-in-the-loop research by drastically reducing the barrier to entry of this field. Video: https://bit.ly/coglab-ijcai

1 Introduction

As AI systems become increasingly ubiquitous, the ability for AI to successfully work alongside humans is increasingly important. We need to ensure that we have both the infrastructure and the algorithms to enable smooth interactions between humans and AI. Despite the growing interest in human-in-the-loop AI, researchers and developers often find themselves constrained by the tools available. Many of these tools are designed for specific, one-time experiments and lack the flexibility needed for broader application.

Cogment was created as a solution to these challenges, offering a platform aimed at facilitating Human-in-the-Loop Learning (HILL), in particular Reinforcement Learning (RL), through an orchestrator that manages interactions between humans, AIs and their environments. However, its technical complexity and the steep learning curve made it difficult to use without a large time commitment.

To bridge this gap, we introduce Cogment Lab. Our goal with Cogment Lab is to make the development of human-interactive AI systems more approachable. We built it on top of Cogment to offer a more user-friendly toolkit that maintains the original's strengths but makes the process of integrating HILL elements into AI research and development simpler and more straightforward.

This paper will detail the design of Cogment Lab, its structure, and what makes it unique. We aim to show how Cogment Lab can be a valuable asset for researchers and developers looking to incorporate human feedback into AI systems without having to reinvent the entire interaction model. Through this, we hope to contribute to the field by making the development of collaborative human-AI systems more accessible to a wider audience.

2 Cogment Lab Outline

Cogment Lab is built on Cogment [Redefined *et al.*, 2021], which itself is a platform for HILL. This means that it inherits some of the built-in design choices and limitations.

Microservice architecture. Cogment Lab is based on microservice architecture, where various components interact with one another via the Cogment orchestrator. On the user side, these components are environments and actors, with Cogment itself providing additional auxiliary services like the trial datastore and the model registry. Each service is launched separately, with the orchestrator managing their interactions and sending messages between them.

Episodic interactions In Cogment Lab, the basic unit of interaction between actors and environments is an episode, also called a trial (following the Cogment terminology).

At its core, Cogment Lab is built for researchers and developers, and thus strives to be as unopinionated and extensible as possible, allowing its users to implement any research ideas they may deem interesting. This puts a natural limit on the scope of features we included in the core library. The

2.1 Cogment Lab vs Cogment

Cogment takes a relatively low-level approach in its design, opting for more fine-tuned control of messages exchanged between the relevant services. This comes at the expense of its ease of use – starting from scratch, implementing a new experiment using Cogment can be a significant effort. For example, the user must define and compile the desired protobuf specification for messages that will be exchanged between all the services, and describe them in a YAML to coordinate the launch via the Cogment orchestrator.

With Cogment Lab, we aim to improve specifically the ease of use. We eliminate most of the Cogment boilerplate, handling it internally within the library. We also provide precompiled message definitions that are designed for typical RL usage.

2.2 Cogment Lab vs Cogment-Verse

Cogment-Verse [Gottipati *et al.*, 2023] is the spiritual ancestor to Cogment Lab, initially also meant to be a research framework for HILL. The key difference between the two libraries is their design. Cogment-Verse functions largely as a modifiable stand-alone application. It is easy to customize it slightly, and then run it to get a standard experiment. However, it is significantly more difficult to use it as a component of a larger system, or for completely new use-cases. The usage of Cogment-Verse is largely config-driven – the user defines the desired experiment in a YAML file. This is passed to the predefined launch script of Cogment-Verse which will execute the experiment.

In contrast, Cogment Lab is meant to be a proper library, and not a framework or an application. It provides components that can be imported and combined in whatever way the user desires, without enforcing any particular way of doing things. Programs built with Cogment Lab has a single, obvious entry point – the main Python script, or a Jupyter notebook, based on the user's preference. Within that, the user can dynamically start and stop any services according to their needs, launch trials to collect data, and use that data using any other libraries and algorithms.

3 Library Design

Because Cogment is inherently asynchronous, the same is true in Cogment Lab. All interactions with Cogment Lab must be wrapped in an asyncio event loops. In return, this makes it possible to easily develop asynchronous algorithms enabling better hardware utilization.

The entry point to any Cogment Lab project is the `Cogment` object, which manages the connection to the orchestrator, and holds all of the key functionalities of the library. By convention, it is created as:

```
cog = Cogment(log_dir=...)
```

This initializes the connection to the Cogment services, and starts keeping track of any services we add.

3.1 Running Services

With the `Cogment` object created, we can run services (environments and actors) either in a separate process, or in some cases, in the same one. Typically, this involves creating an inactive instance of the service in the main process, and then creating a copy of it in the subprocess. While this may initialize some resources redundantly, it is typically useful to obtain metadata about the environment (primarily the observation and action spaces), and it can be disabled by initializing the service appropriately.

3.2 Environments

Cogment Lab supports both Gymnasium [Towers *et al.*, 2023] and PettingZoo [Terry *et al.*, 2021] environments by default. They can be initialized using the environment ID in the Gymnasium registry, and the path to the PettingZoo environment, respectively. For example:

```
cenv = GymEnvironment(
    env_id="CartPole-v1",
    render=True
)
await cog.run_env(env=cenv,
    env_name="cartpole",
    port=9011,
)

pz_path = "pettingzoo.sisl.pursuit.env"
pz_cenv = AECEnvironment(
  env_path=pz_path,
  render=True
)
await cog.run_env(env=pz_cenv,
                  env_name="pursuit",
                  port=9012
)
```

With this, we have two environments running in their own subprocesses, interacting with Cogment on ports 9011 and 9012.

3.3 Actors

Similarly to environments, we can run arbitrary actor implementation. A minimal actor is one that subclasses 'CogmentActor', and implements the `act(obs, rendered_frame)` method which returns the action. An actor can be run in a subprocess:

```
actor = ConstantActor(0)
await cog.run_actor(actor,
  actor_name="constant",
  port=9021
)
```

or in the same process:

```
cog.run_local_actor(actor,
    actor_name="dqn",
    port=9012
)
```

The latter option is particularly useful for neural network-based actors, which may need to be frequently updated during the training. This makes it possible to update the model without having to share the memory, or send messages between different processes.

Human Actors

Finally, in any environment, we can replace algorithmic actors with a human actor. Cogment Lab starts a simple web app that can be accessed from the browser. The user must specify an action map that converts key presses in the web UI to in-environment actions. They may also customize the interface further by providing a custom HTML file that connects with Cogment Lab via a web socket.

```
await cog.run_web_ui(
    actions=["no-op", "ArrowRight"]
)
```

4 Use Cases

4.1 Environment Compatibility

By default, Cogment Lab is compatible with any Gymnasium and PettingZoo environments. These libraries are the de facto standard for open-source RL environments, in the single- and multi-agent context respectively. Any properly defined Gymnasium or PettingZoo environment can be used as an environment service. If it also supports the `rgb_array` render mode, it can be used with the web UI to support a human actor taking the place of one of the agents.

It is also possible to implement a custom environment without going through either interface. To that end, the user can implement an instance of the `BaseEnv` abstract class, which directly follows the design of the Cogment environment API. Alternatively, the `CogmentEnv` base class offers a more modular approach to building a `BaseEnv`, using abstractions that are more similar to typical RL code.

4.2 Environment Conversions

A key concept for non-standard workflows is that of environment conversions. This refers to the process of taking an environment and changing something about its actors. This way, we can add human involvement in environments which, on the surface, do not have any room for that. Cogment Lab currently supports two types of environment conversions: **Observer** and **Teacher**.

In the **Observer** conversions, we take in a Gymnasium environment and produce a PettingZoo environment, using either the AEC or Parallel API. This environment will have two agents: `"gym"` representing the original environment's agent, and `"observer"`, typically a human. The observer receives all the same observations as the agent, including the environment renders, but its actions are ignored. This makes it possible for a human operator to watch the agent's actions in real time through the web UI.

In the **Teacher** conversions, just like before, we turn a Gymnasium environment into a PettingZoo environment, this time with the agents `"gym"` and `"teacher"`. The role of the teacher is observing, and potentially overriding the main student's actions. At each step, the teacher's action indicates whether they wish to override the main agent's action, and if so, what action should actually be executed. The teacher can base this decision on the main agent's observation, and in the case of the AEC API, the agent's originally chosen action. All of this information (both agent's actions, and whether an override happened) is automatically recorded via Cogment, enabling its use during the training process.

4.3 Regular RL

The first, relatively simple use for Cogment Lab is for regular RL training loops, without human involvement. After defining the code for the environment and sampling the actions of a model, Cogment Lab provides a simple abstraction for collecting episodes of data from the environment:

```
trial_id = await cog.start_trial(
    env_name="<env_name>",
    actor_impls={
        "gym": "<actor_name>",
    },
)
data = await cog.get_trial_data(trial_id)
```

Note that this can be done asynchronously – we can start the trial, and then perform some other computation while the trial is running in the background.

This component can be easily integrated in most standard DRL algorithms. Simply add the gathered data to a replay or rollout buffer, and optimize the model using the chosen algorithm. In the repository, we provide examples for both DQN [Mnih *et al.*, 2013] and PPO [Schulman *et al.*, 2017] implementations using this approach.

4.4 Imitation Learning

Cogment Lab can be easily used for all parts of imitation learning [Pomerleau, 1988]. The typical workflow involves collecting the data for imitation beforehand, and then using it as a static dataset for the training loop. In some cases, it is also possible to interactively obtain actions from a pretrained algorithmic agent. This approach is clearly limiting, as often it might be valuable for a human demonstrator to adapt their demonstrations based on the learner's progress.

With Cogment Lab, it is possible to concurrently collect new data for imitation from a human operator, and train the agent on the data it already has access to. We provide an implementation of the simple Behavioural Cloning algorithm in the repository as a proof of concept.

4.5 Active Teaching

With the Teacher conversion described in Section 4.2 we can implement new, creative ways of training RL agents with human involvement. One such method involves combining a learner agent with a human instructor. They act in the environment together, with the human overriding the agent's actions whenever they deem necessary. The actions that were actually executed in the environment are then added to the replay buffer, and used in the learning process. This results in an effective hybrid team, using both the human and the AI together to achieve the goal. We provide an example of this workflow on the Lunar Lander environment in the repository.

5 Summary

In this work, we demonstrate Cogment Lab, an evolution of interacting with Cogment for HILL RL research. We describe its core philosophy, and how it differs from similar projects. We provide an outline of how it can be used in terms of code, but also what algorithms it enables – both existing and new. We aim to extend Cogment Lab with new functionalities, and use it for both commercial and research projects. We release Cogment Lab with a permissive Apache 2 license, allowing the wider community to use it for their own work. With this, we hope to contribute to the growing field of human-in-the-loop learning, to ensure a more collaborative future between humans and AI.

References

[Gottipati *et al.*, 2023] Sai Krishna Gottipati, Luong-Ha Nguyen, Clodéric Mars, and Matthew E. Taylor. Hiking up that HILL with Cogment-Verse: Train & Operate Multi-agent Systems Learning from Humans. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '23, pages 3065–3067, Richland, SC, May 2023. International Foundation for Autonomous Agents and Multiagent Systems.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602 [cs]*, December 2013.

[Pomerleau, 1988] Dean A. Pomerleau. ALVINN: an autonomous land vehicle in a neural network. In *Proceedings of the 1st International Conference on Neural Information Processing Systems*, NIPS'88, pages 305–313, Cambridge, MA, USA, January 1988. MIT Press.

[Redefined *et al.*, 2021] AI Redefined, Sai Krishna Gottipati, Sagar Kurandwad, Clodéric Mars, Gregory Szriftgiser, and François Chabot. Cogment: Open Source Framework For Distributed Multi-actor Training, Deployment & Operations, June 2021. arXiv:2106.11345 [cs].

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*, August 2017.

[Terry *et al.*, 2021] Justin K. Terry, Benjamin Black, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis Santos, Clemens Dieffendahl, Niall L. Williams, Yashas Lokesh, Caroline Horsch, and Praveen Ravi. PettingZoo: Gym for Multi-Agent Reinforcement Learning. *arXiv:2009.14471 [cs, stat]*, February 2021. arXiv: 2009.14471.

[Towers *et al.*, 2023] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023. Language: eng.