Cogment: Open Source Framework For Distributed Multi-actor Human-AI Collaborative Environment

Sai Krishna Gottipati[1], Cloderic Mars[1], Gregory Szriftgiser[1], Sagar Kurandwad[1], Vincent Robert[1] and Francois Chabot[1]

***Abstract*— In this work, we introduce Cogment - an open-source framework that introduces an actor formalism to support a variety of humans / agents collaboration topologies and training approaches through an easily scalable distributed micro service architecture. Leveraging these human-agent collaboration typologies, we further demonstrate how social-intelligence based benchmarks can be developed using Cogment.**

I. INTRODUCTION

In order to understand the complex behaviors of humans, their mental states, and to develop AI agents able to engage in rich and complex interactions with humans, we argue that any number of human and AI agents should be perceived at the same level. In a typical reinforcement learning setup, an AI agent acts in the environment, receives feedback in the form of rewards from the environment and improves itself based on these rewards. In Figure-2, we describe several different ways (blue arrows) in which humans can interact with agents Humans can act in the environment on par with agents (e.g, man vs machine multi-player games), provide rewards to the agents, demonstrate tasks to them, or generate tasks for agents to achieve, and Cogment provides a unifying framework to fulfill all these human-in-the-loop cases. These capabilities of Cogment make it a desired framework to develop social intelligence benchmarks that are largely dependent on the human-AI collaborations.

Recently, there has been significant progress in the performance achieved by AI agents in cooperative multi-agent games. A prominent one among those is the game of Hanabi, which is a very challenging partial-information cooperative multi-agent game. In [1], the authors formulated the quantified notion of social intelligence as a life-long learning benchmark task in the context of reinforcement learning. The setup consisted of three phases. In phase-1, a pool of agents are trained through self-play. In phase-2, one of the agents is taken from this pool and trained sequentially with other partners (also sampled from the same pool) and periodically evaluated against all other agents. In phase-3, each of these agents is evaluated against a new set of agents that weren't used during the training process. While only AI agents were used, trained and played in the author's work, Cogment provides the means to go one step further with AI agents playing against humans. The use of these additional training samples allows agents to develop more socially

[1]AI Redefined, 400 McGill St 300, Montreal. <sai, cloderic, greg, sagar, vincent, francois> @ai-r.com

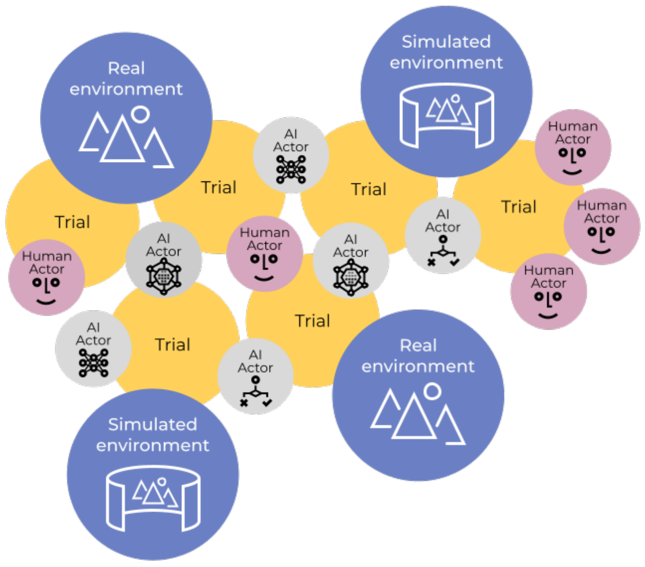

Fig. 1. Cogment orchestrates the running of *trials* involving AI and Human *actors* in simulated or "real-world" *environments*. Each trial involves one or multiple actors and runs in one environment. Multiple trials can run concurrently, and they can share actors or environments.

intelligent behaviours. The role of Hanabi in developing socially intelligent behaviour was also highlighted in several previous works including: [2], [3], [4], [5], [1], [6]. While all this current research has been focused on the centralized training of agents, with Cogment multiple agents can be deployed and train in a distributed and decentralized manner, thus increasing computational efficiency and enabling faster progress in multi-agent reinforcement learning and social intelligence research.

In Section-II, we describe the Cogment framework, and in Section-III, we describe several use-cases and applications of Cogment involving human-in-the-loop learning. We then introduce a novel multi-agent reinforcement learning environment, "Death Match", and use it to develop another social intelligence benchmark task by formulating it in a life-long learning setup, similar to the setup used in [1].

II. KEY FEATURES

A. *Multi-actor framework*

Cogment provides a multi-actor framework where multiple heterogeneous actors interact with an environment during

trials. In Cogment, actors can be either humans or AI agents of any kind: learned or learning or heuristic or random, using machine learning models or not. Cogment thus enables AI agents and humans to interact and train together in shared environments, in any configuration; an actor can interact in one or several trials (see Figure 1).

One key capability of Cogment which supports a wide range of use cases is the training of AI agents based on different types of inputs from the humans present in the same environment. Compared to machines, humans have a limited work capacity and can't operate at the same speeds: this makes their presence scarce in such environments, in terms of data contribution. It is therefore desirable to leverage all the possible inputs that can be generated by them. This includes evaluative feedback, i.e. humans evaluating the performance of AIs at a given task, and demonstrations i.e. humans performing a task for the benefit of an AI [7]. Cogment supports both these types (among others) of human-in-the-loop training. When it comes to evaluative feedback, it can be desirable to mix several sources of such evaluation: multiple humans evaluating the same actions of an AI or mixing the subjective feedback of a human with an objective performance measurement. That's why Cogment supports **rewards from multiple sources** and aggregates them.

Another aspect of human-provided evaluative feedback is an inherent lag between a feedback-calling action, its perception by the human, and its subsequent evaluation. To facilitate that, Cogment supports **retroactive rewards** while maintaining online learning capabilities.

One foundational aspect of Cogment is the definition of contracts for the interactions between the actors and the environment, in the form of an action space and an observation space. These definitions makes it easy to decouple the development process of each actor and environment implementation and also enables a very interesting feature: **implementation swapping**.

While observation and action space define the interface and the way actors and environment can interact together, their *implementation* defines how they behave in accordance with this interface. Several implementations can share the same interface, and if they do, they can be swapped. This opens the door to several training and operating topologies that are especially relevant when involving humans.

Let's say we are training two AI agents in an environment with two humans. The problem with starting the training with this setup is that humans will interact with *very* dumb AI agents for a while before these start to do something useful, which is not an efficient use of Humans. Cogment's implementation swapping allows more interesting curriculum setups to work around this issue by pre-training with other heuristic or rule-based agents.

While the discussion so far has been focused on making the most out of human resources via efficient human-in-the-loop training setups especially for social intelligence, these capabilities can also be extended to situations where there's only different AI agents interacting with each other. Implementation swapping can be used to create complex training and evaluation scenarios. For example, training several AI agents in self play trials, before moving to mixing and matching AIs in another set of trials.

Cogment has been designed to address these issues for its users. From the get go, **Cogment applications are distributed**, every part being implemented as a micro service communicating with the others using a technology-agnostic protocol. With Cogment, a developer, a researcher or a data scientist is able to start working on their personal computer and, ultimately, have their work being deployed on a high performance cluster to support a larger load with **no integration discontinuity** [8]. In the same fashion, a Cogment application might begin with integrating a simulation before moving to its real-world counterpart, in order to facilitate the sim-to-real process and to continuously operate and train the AIs. Cogment **decouples the hosting and distribution concerns from the AI design, implementation and evaluation**.

III. Applications

A. Reinforcement Learning

Reinforcement learning has already seen lot of success in board games ([9], [10], [11]) and video games ([12], [13], [14]). More recently, RL is being used in real world applications like manipulation tasks ([15]), active localization ([16], [17]), autonomous navigation of stratospheric balloons ([18]), de novo drug design ([19], [20]), autonomous driving ([21]) and many others. For computationally efficient training of these RL algorithms, one needs to be able to launch multiple trials simultaneously and gather diverse experiences from the environment. Cogment supports this. It can launch multiple trials, synchronously or asynchronously, using multiple agents or multiple instances of the same agent. For sample efficient learning of algorithms, one has to learn from the old samples - Cogment supports this as well by storing all the relevant information in log files in addition to the standard usage of a replay buffer.

In some cases, an agent performs better if it takes input from, or is bootstrapped by another agent. For example, in expert-iteration [10] based approaches like [11], a policy that imitates the MCTS agent is learnt. The MCTS agent in turn initializes its prior probabilities based on the policy network's output. More generally, RL can be combined with other learning or non-learning based algorithms. For example, genetic algorithms [22], [23] or heuristic search [24]. Cogment provides an efficient way to run such algorithms via the use of actor implementations. In offline RL, the behaviour policy that is used to collect experience could be completely different from the agent that is currently training based on these experiences [25], [26], [27]. Cogment handles this through its efficient use of its **activity logger**.

B. Human-in-the-loop Learning

Humans can interact with learning agents in multiple ways. Humans can act in the environment on par with agents (e.g, man vs machine multi-player games), provide rewards to the agents, demonstrate tasks to them, or generate tasks

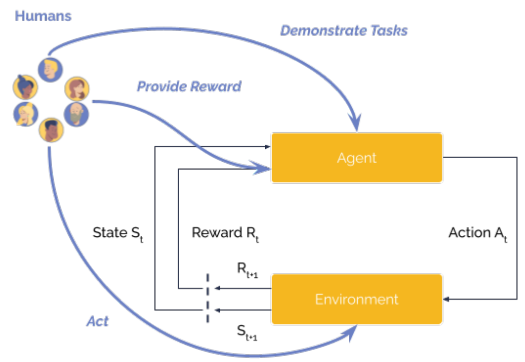

Fig. 2. An illustration of different ways in which humans can be involved in the training process of an AI agent. Humans can act in the environment, provide rewards to the agents, demonstrate tasks or generate tasks for the agents to achieve.

for agents to achieve, and Cogment provides a unifying framework to fulfill all these human-in-the-loop cases. For example, having humans acting in the environment is a way to ensure agents will take safe exploratory actions in sensitive contexts like autonomous driving [28], [29]. Humans can provide rewards for several learning algorithms, for example in the context of evaluating machine-generated dialogs [30], [31], [32], summaries [33], [34], semantic parsers [35], natural language [36], machine translation [37] and many others. In some cases, it is challenging to design a reward function or, the reward function could be sparse, thus making it hard for an RL agent to learn. In such cases, agents can learn from human demonstrations under the imitation learning (IL) paradigm [38], [39], [40]. If it's expensive to use human demonstrations, humans can be used to generate curriculum i.e, generate tasks with an increasing level of difficulty so that the AI agent can learn faster [41], [42], [43]. On the other hand, if no human input is possible, a different AI agent can be used to generate the curriculum [44], [45].

IV. Related Work

[46] is one of the first RL code bases in pytorch that implemented several algorithms like A2C, PPO, ACKTR, GAIL on a wide range of OpenAI gym environments and was successful in replicating the results of these algorithms. Concurrently, [47] introduced rllib (that was built on top of a distributed machine learning framework Ray [48]) that provides a lot of in-built algorithms for direct use. rlpyt [49] is another RL library that provides modular and optimized implementations of several deep RL algorithms in pytorch and is useful for small to medium scale research. Garage [50] also provides similar utilities and is often used for benchmarking different algorithms. ACME [51] (that uses reverb [52]) is a recent framework and is commonly used for quick prototyping. Platforms like [53], [54], [55] enable interaction of agents in real-world environments. While most of these frameworks are suitable for RL research, they do not explicitly account for how humans can actively participate in the learning process and thus are not suitable for prototyping socially intelligent behaviors.

V. Death Match

Death match was developed as a testbed and showcase for Multi Agent Reinforcement Learning (MARL), life long learning and social intelligence using Cogment. The game is a competitive and cooperative paintball-like shooter game, where teams of agents compete against each other in an arena for a trophy. Agents shoot paint balls at their opponents from other teams. An agent is eliminated if it is hit by an opponent; there is no friendly fire. Last team standing wins the game; each Cogment's trial consists of a game.

Death match includes a lightweight web based front-end where users can configure and watch a trial. Figure 3 is a screenshot of this Web Client featuring a duel between two agents trained using self-play. The larger circles are the players, the smaller circles are the paint balls, and the lines represent the region of visibility for each player. Since the velocity of the paint balls is low, the physics of the simulation makes their trajectory nonlinear with the movements of the players.

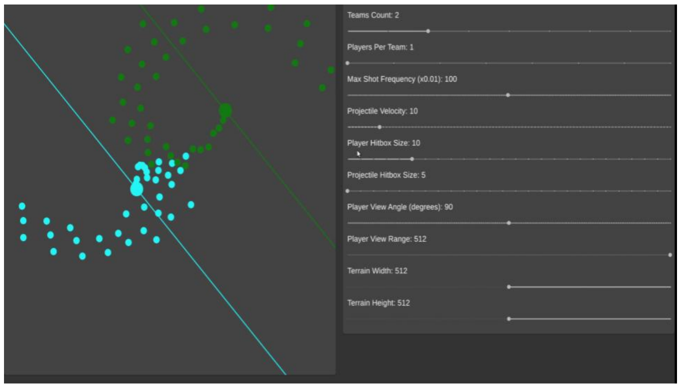
Teams Count: 2
Players Per Team: 1
Max Shot Frequency (x0.01): 100
Projectile Velocity: 10
Player Hitbox Size: 10
Projectile Hitbox Size: 5
Player View Angle (degrees): 90
Player View Range: 512
Terrain Width: 512
Terrain Height: 512

Fig. 3. Screenshot of the Death Match Web Client. The left pane of the window shows the trial in progress, while the right pane displays the configurations of the trial such as the number of teams in the trial, number of players in each team, shot frequency, shot velocity, arena size etc.

Death match also allows to run in a "headless" mode, i.e. without any visualization, to gather data and train multiple agents over campaigns of multiple trials executed in parallel and at scale.

A. Modeling

Similar to [1], our social-intelligence benchmarking task on Death Match involves three phases. In the first phase, different agents (varying algorithms, architectures and seeds) were trained using self-play. Their performance in this phase was judged based solely on the rewards achieved at the end of their training. In phase-2, agents were sampled from this trained pool of agents and paired against each other. The agents are expected to develop social intelligence behaviours in this phase. They are scored based on several different metrics including total survival time of each of the agents, number of paint balls fired by each agent, collaborating with other agents to fight against a common enemy etc.. However, this is only a training phase, and hence the behaviours can be biased towards playing against other agents present only in this pool. To avoid this, we introduce phase-3 where in, each agent is made to play against agents that it hasn't played before. This could include humans as well. Death Match is a simpler game than Hanabi (as the rewards are more dense and the action space is fixed) and hence the focus would be on developing algorithms than could master social intelligence behaviours rather than on multi-agent reinforcement algorithms for training agents using self-play.

B. Implementation

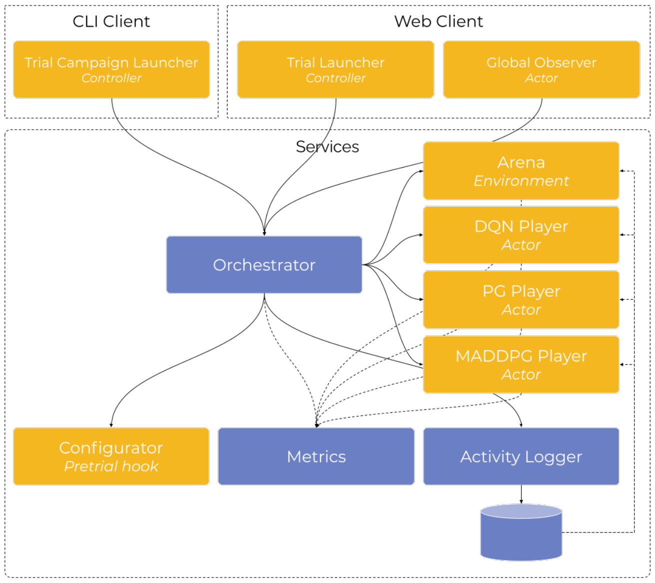

Fig. 4. Death Match instantiates Cogment architecture implementing a command line interface (CLI) client including a trial campaign launcher, a web client including a trial launcher as well as an observer actor, and finally a number of services, the arena environment, a trial configurator and a number of Actors. All of the Death Match specific modules are depicted in orange and are implemented using Cogment SDKs, Cogment out-of-the-box modules are depicted in blue.

Death Match architecture is depicted in Figure 4. Several modules were developed specifically for Death Match using the Cogment SDKs.

- The **arena** is the environment service, it implements the logic for the Death match game: terrain, physics simulation, players elimination etc..
- The **web client**, shown in Figure 3, allows to configure trial parameters through a controller and and visualize the trial in progress thanks to a global observer actor.
- The **command line interface (CLI) client** triggers a number of parallel trials continuously for training purposes.
- Several **player agents** are implemented as actor services. Several RL algorithms are available, some able to leverage the continuous nature of the action space, such as MADDPG [56], others needing to discretize such as PG [57], [58] or DQN [59]. Other non-learning agents were implemented using scripted heuristic behaviors, which can be used during training or as an evaluation baseline.
- Finally the **configurator** is a pre trial hook used for service discovery and load balancing. Upon startup, actor services register with the configurator. When a trial starts, the configurator can then enrich the trial's configuration with a random network endpoint of the desired actor implementation to enable the orchestrator to reach them and spreading the load between several services.

To understand how all the pieces work together, let's look at how a trial plays out. First, one of the controllers defines a basic trial configuration, this includes the number of teams, which player agent should be part of each team as well as configuration parameters for the arena such as the speed of the paint balls. When the trial is started by the web client's controller, it specifies that the global observer actor should be part of the trial too. Once the configuration is complete the start of the trial is requested to the orchestrator.

As soon as the orchestrator receives the trial start requests it calls the configurator pre trial hook which extend the trial configuration with the network endpoints of the actor implementations involved in the trial. If the global observer client actor is involved the orchestrator wait for it to join and the proper trial can start.

Every trial begins with the arena environment sending initial observations to every actor in the trial through the orchestrator. After receiving observations from the environment, the actor implementation chooses an action according to their respective model (e.g, a neural network). Actions are then sent to the environment through the orchestrator. Once received, it updates its internal state accordingly (e.g. it moves players that decided on a movement, eliminate players hit by a paint ball), it then computes the reward for each player and then sends new observations and rewards to their respective actors. At the end of each trial, the orchestrator dumps tuples of observations, actions, and rewards of every actor to the activity logger, which the actors sample randomly from to update their respective deep learning models and learn from their experiences so far. The total rewards the actors receive across trials can be monitored through the metrics module's dashboard, the actors' models are considered trained if the moving average of the total rewards they accumulated across ten consecutive trials is more than a predefined threshold.

C. Results

While the different agents trained using self-play using different algorithms ([56], [57], [58], [59]) with different seeds achieved a local maximum behaviour, they performed abysmally when paired against each other. When there are more than two teams involved, they failed to implicitly cooperate with other teams to defeat a common enemy. This demonstrated the need to train agents in a socially intelligent way by letting the agents play against each other where they learn to act based on other agent's behaviour and intents. Further in phase-3, we test these socially intelligent agents by making them play against new agents (learned, random and humans) that were not used in phase-2.

References

[1] H. Nekoei, A. Badrinaaraayanan, A. Courville, and S. Chandar, "Continuous coordination as a realistic scenario for lifelong learning," 2021.

[2] H. Hu and J. N. Foerster, "Simplified action decoder for deep multi-agent reinforcement learning," 2019.

[3] J. N. Foerster, F. Song, E. Hughes, N. Burch, I. Dunning, S. Whiteson, M. Botvinick, and M. Bowling, "Bayesian action decoder for deep multi-agent reinforcement learning," 2019.

[4] N. Bard, J. N. Foerster, S. Chandar, N. Burch, M. Lanctot, H. F. Song, E. Parisotto, V. Dumoulin, S. Moitra, E. Hughes, and et al., "The hanabi challenge: A new frontier for ai research," *Artificial Intelligence*, vol. 280, p. 103216, 2020. [Online]. Available: http://dx.doi.org/10.1016/j.artint.2019.103216

[5] J. Walton-Rivers, P. R. Williams, R. Bartle, D. Perez-Liebana, and S. M. Lucas, "Evaluating and modelling hanabi-playing agents," 2017.

[6] A. Fuchs, M. Walton, T. Chadwick, and D. Lange, "Theory of mind for deep reinforcement learning in hanabi," 2021.

[7] R. Zhang, F. Torabi, L. Guan, D. H. Ballard, and P. Stone, "Leveraging Human Guidance for Deep Reinforcement Learning Tasks," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. Macao, China: International Joint Conferences on Artificial Intelligence Organization, Aug. 2019, pp. 6339–6346. [Online]. Available: https://www.ijcai.org/proceedings/2019/884

[8] C. Muratori, "Designing and evaluating reusable components," 2014. [Online]. Available: https://caseymuratori.com/blog_0024

[9] G. Tesauro, "Td-gammon, a self-teaching backgammon program, achieves master-level play," *Neural Computation*, vol. 6, no. 2, pp. 215–219, 1994.

[10] T. Anthony, Z. Tian, and D. Barber, "Thinking fast and slow with deep learning and tree search," *CoRR*, vol. abs/1705.08439, 2017. [Online]. Available: http://arxiv.org/abs/1705.08439

[11] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," 2017.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[13] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, T. Ewalds, D. Horgan, M. Kroiss, I. Danihelka, J. Agapiou, J. Oh, V. Dalibard, D. Choi, L. Sifre, Y. Sulsky, S. Vezhnevets, J. Molloy, T. Cai, D. Budden, T. Paine, C. Gulcehre, Z. Wang, T. Pfaff, T. Pohlen, D. Yogatama, J. Cohen, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, C. Apps, K. Kavukcuoglu, D. Hassabis, and D. Silver, "AlphaStar: Mastering the Real-Time Strategy Game StarCraft II," https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/, 2019.

[14] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with large scale deep reinforcement learning," *CoRR*, vol. abs/1912.06680, 2019. [Online]. Available: http://arxiv.org/abs/1912.06680

[15] O. OpenAI, M. Plappert, R. Sampedro, T. Xu, I. Akkaya, V. Kosaraju, P. Welinder, R. D'Sa, A. Petron, H. P. de Oliveira Pinto, A. Paino, H. Noh, L. Weng, Q. Yuan, C. Chu, and W. Zaremba, "Asymmetric self-play for automatic goal discovery in robotic manipulation," 2021.

[16] S. K. Gottipati, K. Seo, D. Bhatt, V. Mai, K. Murthy, and L. Paull, "Deep active localization," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4394–4401, 2019.

[17] D. S. Chaplot, E. Parisotto, and R. Salakhutdinov, "Active neural localization," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=ry6-G_66b

[18] M. G. Bellemare, S. Candido, P. S. Castro, J. Gong, M. C. Machado, S. Moitra, S. S. Ponda, and Z. Wang, "Autonomous navigation of stratospheric balloons using reinforcement learning," *Nature*, 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2939-8

[19] S. K. Gottipati, B. Sattarov, S. Niu, Y. Pathak, H. Wei, S. Liu, S. Liu, S. Blackburn, K. Thomas, C. Coley, J. Tang, S. Chandar, and Y. Bengio, "Learning to navigate the synthetically accessible chemical space using reinforcement learning," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 2020, pp. 3668–3679. [Online]. Available: http://proceedings.mlr.press/v119/gottipati20a.html

[20] S. K. Gottipati, Y. Pathak, R. Nuttall, Sahir, R. Chunduru, A. Touati, S. G. Subramanian, M. E. Taylor, and S. Chandar, "Maximum reward formulation in reinforcement learning," 2020.

[21] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J. Allen, V. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," *CoRR*, vol. abs/1807.00412, 2018. [Online]. Available: http://arxiv.org/abs/1807.00412

[22] S. Chang, J. Yang, J. Choi, and N. Kwak, "Genetic-gated networks for deep reinforcement," 2018.

[23] M. E. Taylor, S. Whiteson, and P. Stone, "Temporal difference and policy search methods for reinforcement learning: An empirical comparison," in *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI)*, 2007, pp. 1675–1678.

[24] L. Orseau and L. H. S. Lelis, "Policy-guided heuristic search with guarantees," 2021.

[25] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," 2020.

[26] A. Shrestha, S. Lee, P. Tadepalli, and A. Fern, "Deepaveragers: Offline reinforcement learning by solving derived non-parametric mdps," 2020.

[27] I. Kostrikov, J. Tompson, R. Fergus, and O. Nachum, "Offline reinforcement learning with fisher divergence critic regularization," 2021.

[28] M. Palan, N. C. Landolfi, G. Shevchuk, and D. Sadigh, "Learning reward functions by integrating human demonstrations and preferences," *CoRR*, vol. abs/1906.08928, 2019. [Online]. Available: http://arxiv.org/abs/1906.08928

[29] J. Leike, D. Krueger, T. Everitt, M. Martic, V. Maini, and S. Legg, "Scalable agent alignment via reward modeling: a research direction," *CoRR*, vol. abs/1811.07871, 2018. [Online]. Available: http://arxiv.org/abs/1811.07871

[30] N. Jaques, A. Ghandeharioun, J. H. Shen, C. Ferguson, A. Lapedriza, N. Jones, S. Gu, and R. Picard, "Way off-policy batch deep reinforcement learning of implicit human preferences in dialog," 2019.

[31] S. Yi, R. Goel, C. Khatri, A. Cervone, T. Chung, B. Hedayatnia, A. Venkatesh, R. Gabriel, and D. Hakkani-Tur, "Towards coherent and engaging spoken dialog response generation using automatic conversation evaluators," 2019.

[32] B. Hancock, A. Bordes, P.-E. Mazaré, and J. Weston, "Learning from dialogue after deployment: Feed yourself, chatbot!" 2019.

[33] N. Stiennon, L. Ouyang, J. Wu, D. M. Ziegler, R. Lowe, C. Voss, A. Radford, D. Amodei, and P. Christiano, "Learning to summarize from human feedback," 2020.

[34] F. Böhm, Y. Gao, C. M. Meyer, O. Shapira, I. Dagan, and I. Gurevych, "Better rewards yield better summaries: Learning to summarise without references," 2019.

[35] C. Lawrence and S. Riezler, "Improving a neural semantic parser by counterfactual learning from human bandit feedback," 2018.

[36] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving, "Fine-tuning language models from human preferences," 2020.

[37] J. Kreutzer, S. Khadivi, E. Matusov, and S. Riezler, "Can neural machine translation be improved with user feedback?" *CoRR*, vol. abs/1804.05958, 2018. [Online]. Available: http://arxiv.org/abs/1804.05958

[38] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters, "An algorithmic perspective on imitation learning," *CoRR*, vol. abs/1811.06711, 2018. [Online]. Available: http://arxiv.org/abs/1811.06711

[39] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/cc7e2b878868cbae992d1fb743995d8f-Paper.pdf

[40] R. Dadashi, L. Hussenot, M. Geist, and O. Pietquin, "Primal wasserstein imitation learning," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=TtYSU29zgR

[41] R. Wang, J. Lehman, J. Clune, and K. O. Stanley, "Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions," 2019.

[42] B. Ibarz, J. Leike, T. Pohlen, G. Irving, S. Legg, and D. Amodei, "Reward learning from human preferences and demonstrations in atari," 2018.

[43] P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," 2017.

[44] S. Sukhbaatar, I. Kostrikov, A. Szlam, and R. Fergus, "Intrinsic motivation and automatic curricula via asymmetric self-play," *CoRR*, vol. abs/1703.05407, 2017. [Online]. Available: http://arxiv.org/abs/1703.05407

[45] O. OpenAI, M. Plappert, R. Sampedro, T. Xu, I. Akkaya, V. Kosaraju, P. Welinder, R. D'Sa, A. Petron, H. P. d. O. Pinto, A. Paino, H. Noh, L. Weng, Q. Yuan, C. Chu, and W. Zaremba, "Asymmetric self-play for automatic goal discovery in robotic manipulation," 2021.

[46] I. Kostrikov, "Pytorch implementations of reinforcement learning algorithms," https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail, 2018.

[47] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, J. Gonzalez, K. Goldberg, and I. Stoica, "Ray rllib: A composable and scalable reinforcement learning library," *CoRR*, vol. abs/1712.09381, 2017. [Online]. Available: http://arxiv.org/abs/1712.09381

[48] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, W. Paul, M. I. Jordan, and I. Stoica, "Ray: A distributed framework for emerging AI applications," *CoRR*, vol. abs/1712.05889, 2017. [Online]. Available: http://arxiv.org/abs/1712.05889

[49] A. Stooke and P. Abbeel, "rlpyt: A research code base for deep reinforcement learning in pytorch," *CoRR*, vol. abs/1909.01500, 2019. [Online]. Available: http://arxiv.org/abs/1909.01500

[50] T. garage contributors, "Garage: A toolkit for reproducible reinforcement learning research," https://github.com/rlworkgroup/garage, 2019.

[51] M. Hoffman, B. Shahriari, J. Aslanides, G. Barth-Maron, F. Behbahani, T. Norman, A. Abdolmaleki, A. Cassirer, F. Yang, K. Baumli, S. Henderson, A. Novikov, S. G. Colmenarejo, S. Cabi, C. Gulcehre, T. L. Paine, A. Cowie, Z. Wang, B. Piot, and N. de Freitas, "Acme: A research framework for distributed reinforcement learning," *arXiv preprint arXiv:2006.00979*, 2020. [Online]. Available: https://arxiv.org/abs/2006.00979

[52] A. Cassirer, G. Barth-Maron, E. Brevdo, S. Ramos, T. Boyd, T. Sottiaux, and M. Kroiss, "Reverb: A framework for experience replay," 2021.

[53] A. Mandlekar, Y. Zhu, A. Garg, J. Booher, M. Spero, A. Tung, J. Gao, J. Emmons, A. Gupta, E. Orbay, S. Savarese, and L. Fei-Fei, "Roboturk: A crowdsourcing platform for robotic skill learning through imitation," *CoRR*, vol. abs/1811.02790, 2018. [Online]. Available: http://arxiv.org/abs/1811.02790

[54] C. Gan, J. Schwartz, S. Alter, M. Schrimpf, J. Traer, J. D. Freitas, J. Kubilius, A. Bhandwaldar, N. Haber, M. Sano, K. Kim, E. Wang, D. Mrowca, M. Lingelbach, A. Curtis, K. T. Feigelis, D. M. Bear, D. Gutfreund, D. D. Cox, J. J. DiCarlo, J. H. McDermott, J. B. Tenenbaum, and D. L. K. Yamins, "Threedworld: A platform for interactive multi-modal physical simulation," *CoRR*, vol. abs/2007.04954, 2020. [Online]. Available: https://arxiv.org/abs/2007.04954

[55] X. Gao, R. Gong, T. Shu, X. Xie, S. Wang, and S. Zhu, "Vrkitchen: an interactive 3d virtual environment for task-oriented learning," *CoRR*, vol. abs/1903.05757, 2019. [Online]. Available: http://arxiv.org/abs/1903.05757

[56] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *CoRR*, vol. abs/1706.02275, 2017. [Online]. Available: http://arxiv.org/abs/1706.02275

[57] R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems 12*, vol. 12. MIT Press, 2000, pp. 1057–1063.

[58] J. Zhang, J. Kim, B. O'Donoghue, and S. Boyd, "Sample efficient reinforcement learning with reinforce," 2020.

[59] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602