Lux AI Season 3: Multi-Agent Meta Learning at Scale

Stone Tao Akarsh Kumar Bovard Doerschuk-Tiberi Isabelle Pan
Addision Howard Hao Su

stao@ucsd.edu

Abstract

The proposed competition revolves around testing the limits of agents (e.g rule-based or Meta RL agents) when it comes to adapting to a game with changing dynamics. We propose a unique 1v1 competition format where both teams face off in a sequence of 5 games. The game mechanics, along with partial observability are designed to ensure that optimal gameplay requires agents to efficiently explore and discover the game dynamics. They ensure that the strongest agents may play "suboptimally" in game 1 to explore, but then win easily in games 2 to 5 by leveraging information gained through game 1 and adapting. This competition provides a GPU parallelized game environment via Jax to enable fast training/evaluation on a single GPU, lowering barriers of entry to typically industry-level scales of research. Participants can submit their agents to compete against other submitted agents on a online leaderboard hosted by Kaggle ranked by a Trueskill ranking system. The results of the competition will provide a dataset of top open-sourced rule-based agents as well as many game episodes that can lead to unique analysis (e.g. quantifying emergence/surprise) past competitions cannot usually provide thanks to the number of competitors the Lux AI Challenges often garner.

Keywords

Environment Design, Multi-Agent Reinforcement Learning, Meta Learning

1 Competition description

1.1 Background and impact

A cornerstone of human intelligence is the ability to adapt to new situations on the fly with little information. The machine learning equivalent is known as meta learning, which seeks to train agents capable of adapting to new environments/dynamics on the fly as it simultaneously explores and exploits the task at hand.

Meta-learning focused environments are of great interest in exploring adaptable AI agents, but most are often closed-source or slow to run. Another core issue of existing multi-agent environments is that they often do not have a set of strong diverse baselines. As a result it is difficult to know during training whether the agent is actually playing well or is just over optimizing against another agent. Furthermore, as far as the organizers know there has never been a large-scale AI competition on a meta-learning type environment before.

As thus, we believe it is of high interest to the NeurIPS and broader AI community in investigating the adapatibility of algorithms like Meta-RL when competing against the best rule-based solutions humans are able to write. By running this competition, we require the top prize winning solutions to

Preprint. Under review.

be open sourced which in tandem with our designed game mechanics, provide a diverse set of strong baselines to robustly test the abilities of learned agents.

The past iterations of the Lux AI Challenge (Season 1 and Season 2) have had a combined total of 1,800+ teams and 25,000+ submissions, underlining the success of our team in designing accessible and inclusive AI competitions. Based on this, we conservatively anticipate the proposed Season 3 will have at least 500+ competitors. Like past seasons, the competition will invite interest from not only researchers in reinforcement learning, but as well as top rule-based competitors who have built complex rule-based solutions in other AI programming competitions like StarCraft or Screeps. This is evidenced by the diversity of the top solutions in the most recent Lux AI Challenge Season 2 in the programming languages used as well as the strategies employed. The top 5 of the Season 2 competition consists of rule-based agents written in C++, Typescript, and Python, as well as an RL agent.

Like Season 2, we provide a Jax based game implementation, allowing algorithms to sample at the scale of millions of frames per second on a single 4090 GPU, allowing for much more accessible research into scalable RL without needing 1000s of CPUs / industry scale compute. Such a test bed enables researchers to focus less on engineering and more on novel algorithms. For a setting that is both multi-agent and requiring strong meta-learning, algorithms are likely in need of a ton of data which we enable via GPU parallelization of the environment. Moreover, past work on environments like XLand [1] have shown how emergent properties and open-endedness can arise from scaling up data collection which we supplement via the Jax implementation of the game.

1.2 Novelty

Lux AI Season 3 is the continuation of a series of competitions organized under the Lux AI Challenge name. Each season tackles a different problem in ML, most often around RL, and does not re-use old competition themes or data. Lux AI Season 1 was a standalone (for fun) competition while Season 2 was part of NeurIPS Competitions 2023. By constructing new games and themes each season, our competition is able to actively engage competitors and get large numbers of competitors and become one of the most popular RL competitions in recent years.

1.2.1 Novelty compared to past Lux AI Challenges

Season 1 was a simple game with no focus on particular topics in RL and did not provide a GPU parallelized environment.

Season 2 focused on letting competitors compete 1v1 against others at a large-scale with hundreds to thousands of controllable units on a game board, testing the limits of RL at scale. Scale was achieved via both the large game board, the large action-space and dynamics complexity, as well as a fast environment implemented in Jax [2] for GPU parallelized online training.

This proposal for Season 3 like Season 2 provides a fast Jax-based environment implementation but now focuses on testing the adaptability of algorithms (e.g. Meta RL) to partially observable environments that change dynamics in each 5-game sequence.

1.2.2 Novelty Compared to Past Competitions

Currently, there are very few open-ended, complex, meta learning environments for agents, and fewer (if any) competitions that require agents to adapt on the fly to changing game dynamics. The proposed challenge, as far as the organizers know, should be the first of its kind. Moreover, with a track record of often bringing hundreds of competitors, by running this competition we can gather significant amounts of gameplay data from top rule-based / learning based agents that can provide unique analysis into meta learning and compare learned vs human behaviors (described more in Sec. 1.6.3).

In terms of meta learning environments, there is XLand [1], however it is closed source and thus cannot be compared against. There is the recent open-sourced XLand Minigrid [3], however like XLand, they both suffer from limitations in terms of the kind of complex dynamics that are possible in those frameworks. Particularly, they are limited to fixed rulesets that are then combined together to achieve combinatorially high amounts of different game dynamics. Our proposed competition builds on a core engine we provide complete with Jax based utilities to enable GPU parallelization of a

wider variety of potential dynamics that are not limited by fixed rulesets, with more details in Sec. 1.4.

1.3 Data

There is no standard offline dataset associated with this competition and instead we will provide utilities and tools to scrape public match data that is generated as the competition progresses. The plan is to then open-source the data at the end of the competition for post-competition analysis and further research.

1.4 Tasks and application scenarios

The proposed Season 3 competition consists of a single stage of competition revolving around testing the adaptability of agents in a complex multi-agent environment. The base game teams compete on is composed of a fixed set of game mechanics (specifically defined in Sec. 1.6.2) and hyperparameters for each mechanic.

At test time on the leaderboard, unique to Lux AI Season 3 is that we run 5-matches one after another without resetting agent memory. At the start of the 5-match sequence random game mechanics are activated with randomized hyperparameters, none of which is information given to either team. As a result, optimal agents must be capable of uncovering and adapting to the changing game dynamics at test time and must be robust.

We believe this can result in a number of highly intriguing and different behaviors for both rules-based and RL Agents. Optimal agents may spend time on game 1 exploring and just finding out what mechanics are active and how they work, and then beat the opponent on the 4 remaining games because they have much more information. Some agents may even find ways to constrain the knowledge gathering of the opponent to ensure they do not figure out if certain mechanics are active or are efficient to use (e.g. blocking the path of opposition units). By running this competition we can test the limits of the adaptability of RL agents who compete against human written rule-based solutions and other learned agents.

The next few sections will cover some simple engineering details around how we are there are few limitations in the types of possible game dynamics, ease of use, as well as the specific game variations that are possible in the proposed competition. We further describe the potential unique post-competition analyses that can be done on the data that results from the competition (Data in the form of the code of top solutions as well as match data).

1.5 Core Engine

The core engine is effectively the template code that all game variations must follow. The objective is to provide tooling for writing Jax-based game dynamics as well as being templated well enough so that even LLMs can generate games themselves by filling in the blanks. Moreover, with a fixed core engine we ensure that all game variations have the same exact observation/action spaces, which should make large-scale training across different games easier to work with engineering wise.

We make a few simple assumptions to ensure the possiblity of many game dynamics while also being fast to run and simple to understand.

2D Grid: The environment itself is always in a 2D grid composed of rows of cells. Each cell at position i, j has a state $s^c_{i,j} \in \mathbb{R}^n$ associated with it that can be modified by the dynamics function. We fix the environment to be a 2D grid as this is easily visualizable and creating a render of the environment is fast and parallelizable as well (3D environements are very difficult to render in parallel and further require additional engineering time to build/maintain).

Agent State and Actions: The ith agent in the engine maintains an agent state $s^a_i \in \mathbb{R}^m$. The engine allows each agent i in the game to simultaneously submit a single action $a_i \in \mathbb{R}^k$. This is fairly standard to most implementations of multi-agent environments.

Local Dynamics Function: A local dynamics function ρ_l is used to process each tile of the map and adjust the cell states. It takes as input the cell states in a $k \times k$ area around the cell and computes an update to the cell state.

Agent Dynamics Function: A agent dynamics function ρ_a takes as input an agent state, as well as the cell states and agent states in a $k \times k$ area around the agent on the 2D grid and computes an update to the agent state.

Semantic Mapping Layer: For rule-based competitors and visualization, each dimension of cell state, agent state, as well as action space, are mapped to a human-readable string describing what it means. This allows for automatic visualization of most games (when coupled with an image generator to generate sprites/pictures of the cell state descriptions) making it easy to qualitatively watch games quickly without having to write additional visualization code.

1.6 Season 3 Game Variation

1.6.1 Game Setup

Game Map: The game is a 2D grid map of size 64×64. Each cell has the following possible states: 1. Empty Space, 2. Asteroid Chunk, 3. Planet. In each cell there can also be a unit (also known as a spaceship) from either team on top of it.

The map itself will generally generate cells with asteroid chunks to be grouped up into random clusters of size 1 - 16.

Planets are spawned randomly throughout the map on cells without asteroid chunks and are guaranteed to be at least 10 cells away (manhattan distance) from another planet. An equal number of planets will be home planets for one team or another team. Some planets are neutral and have no team affiliation.

The map is also not guaranteed to be symmetric. Moreover in the sequence of 5 games the game map is always fixed.

Game Initialization: The game is a 1v1 game between two teams, and both teams control the same number of spaceship units which is a random integer from 4 to 16. At the start of each game each unit holds 200 fuel and 0 ore. Each unit's spawn location is a unique Planet that only spawns that one unit (note there may be more Planets than units).

Game Objective: The objective of each of the 5 games is to have the most ore in cargo holds and deposited at friendly planets by the end of 200 steps. Whoever wins the majority of the 5 games is the final winner.

Observations: Each friendly unit provides up to date data on cell state and the state of units on those cells within a 7x7 square centered on the unit. The observed data by each team is the combination of each friendly unit's data.

With partial observability of game state, this ensures teams need to perform additional exploration, especially in the first game out of 5. Moreover, this further ensures that teams can test game mechanics via exploration (e.g. destroying units, running into asteroids etc. and seeing the result) without concern that the other team can simply take that information without risking their own units, meaning there are inherent advantages to exploration in the early games.

Units: Each unit is a spaceship that takes up a 1x1 space on the map. Each unit has a cargo hold indicating how much fuel and ore it is carrying. Each unit can hold a maximum of 1000 fuel and 1000 ore before being unable to pick up any more resources.

Unit Actions: Each unit on either team can do the following actions each timestep.

- Move: Unit moves one tile up, right, down, or left. Costs 10 fuel.
- Fire Mining Laser: Unit fires a laser that hits k tiles in a straight line in any direction. The behaviors of afflicted tiles (and units on those tiles) depend on the active mechanics. Costs $10 + 10k$ fuel. k can be any value from 1 to 5.
- Transfer Resource: Unit transfers any discrete amount of fuel and ore to an adjacent tile. If a different unit ends its turn on the tile with the resources just transferred to, that unit picks up the resources up to its cargo capacity. Otherwise the resources are left there on the tile.

Moreover, units always collect the resources underneath the unit at the end of a turn, whether it was resources transferred by another unit, or the resources dropped by destroyed units.

Asteroid Chunks: These chunks each take a 1x1 sized cell and are a source of ore that is critical for gaining points to win games. Each asteroid chunk starts with an integrity value of 100, which when decreased to 0, will cause the asteroid chunk to be destroyed and replaced by empty space. Each asteroid has a fixed amount of ore that gets dropped once destroyed, which ranges from 100 to 500 ore. More details covered in the mechanics section below.

Planets: These Planets take up 1x1 sized cells and are both spawn locations for units, as well as sources of fuel. Each Planet Tile has a specified discrete yield rate ranging from 10 to 20, which is then multiplied by a game parameter γ described in the next section to give the amount of fuel to the unit each timestep. Note that fuel gathering from planets can be disabled as a mechanic. Planets also allow units to drop off the ore in their cargo hold which is then counted permanently towards the team's score. Units drop off ore automatically upon moving on top of a planet of the same team. See the mechanics section for the different behaviors when a unit moves on top of a planet of a different team.

1.6.2 The Mechanics

Note that we always run beta versions of competitions for a few weeks to beta test the designed mechanics and game themes and revise accordingly. The revisions made are often done to ensure the game is simple enough (low skill floor) for new/less experienced competitors to ensure good inclusivity in the competition, as well as complex enough to support emergent open-ended interactions/behaviors in learned RL agents and also be of interest to the strongest rule-based competitors.

The mechanics listed below are the initial proposed ideas for the Season 3 game and are annotated with whether they can be deactivated or not and the conditions of activation.

1. Destroyed Unit Behavior. When a unit is destroyed for any reason, any ore held by the units are dropped on to the tile with $\alpha\%$ removed from the game entirely. Fuel is lost entirely. The unit itself is then respawn at its original spawn location with 200 fuel after a 20 timestep delay. α is randomized from the following set of values $\{0, 0.25, 0.5, 0.75, 1\}$. Always active.
2. Unit Spawning Behavior. There are two mechanics, of which only one can be active at a time.
   (a) When a unit is to be spawned on a Planet cell after the 20 timestep respawn timeout, it can be blocked from being spawned in by a friendly or opposition unit that ends its turn on the same Planet cell. This resets the unit's respawn timer so it has to wait another 20 timesteps before trying to spawn again.
   (b) The second behavior is instead of blocking spawning, the unit that blocks the spawn is destroyed and the spawned unit is actually spawned in.
3. Unit to Unit Collisions. Units moving onto the same tile will collide and both be destroyed. Always active.
4. Unit to Asteroid Collisions. Units moving onto a asteroid chunk tile will collide and both the unit and asteroid chunk will be destroyed. The asteroid chunk is replaced by empty space. Can be deactivated.
5. Laser hits Asteroid Chunk. Any laser fired by any unit that hits an asteroid chunk will decrease the chunk's integrity by β. Once the integrity reaches or goes below 0 the asteroid chunks resources are dropped and the asteroid chunk is replaced by empty space. Always active.
6. Laser hits unit of the same team. A laser fired by one unit that hits a unit on the same team will destroy that unit. Can be deactivated.
7. Laser hits unit of the opposition team. A laser fired by one unit that hits a unit on the different team will destroy that unit. Can be deactivated.
8. Passive Fuel Gathering from Planet Tiles. When a unit ends the turn on top of a Planet Tile, it will gain the Planet Tile's fuel yield rate multiplied by γ. γ is randomized from the following set of values $\{0.5, 1, 1.5, 2\}$. Can be deactivated. (Note if deactivated, agents must devise unique strategies to get more fuel via transferring fuel and destroying units to respawn them in with the starting fuel).

The mechanics above are designed such that different strategies are required to play optimally when certain mechanics are activated/deactivated. Some mechanics will also interact directly with other mechanics which can create unique combinations that increase the complexity of the game despite having the same fixed ruleset.

1.6.3 Applications / Post Competition Analysis

Once the competition is over, we will have accumulated many game episodes between agents of all skill levels, including agents written by humans, as well as agents trained via RL or IL. Additionally, with the requirement that prize winning solutions must open-source their code, there will be a dataset of high-quality and high-performing rule-based agents that can then be run to generate infinitely more gameplay data at will.

Quantifying Aspects of Surprise/Emergence: Unique to the Lux AI Challenge, due to the massive number of rule-based agents our competition often gets, we can conduct at scale unique analysis of the behavior of agents trained with RL. With data from rule-based and RL agents, we can quantify some properties of surprise where agents learn strategies that humans could not come up with during the competition timeframe. Normally in other competitions/environments authors often rely on qualitative observation to determine the amount of surprise. Concretely, we propose to construct a strategy embedding space based on data and game episodes/trajectories of human written rule-based agents and RL-agents and show quantitatively how novel a learned agent is behaving compared to human written rule-based agents.

The last two seasons of the Lux AI season have shown our designed games have always resulted in fairly different top rule-based agents, especially after revising based on beta testing results / competitor feedback. Thus, we can be fairly confident in expecting sufficient strategical diversity at the top of the leaderboard.

Strong Baselines and Testbed for Meta-Learning Generalization in Multi-Agent Games: The core engine described in Sec. 1.5 is kept simple but highly expressive in order to foster potentially emergent agent behaviors via simple sets of mechanics. It is also built like a template to enable the potential for LLMs to be prompted to generate infinite different games with highly different game mechanics, free of the limited rulesets past environments like XLand are bound to. The result is there is potential future research to test general purpose "game playing" agents (trained on a training set of games) on an unseen multi-agent environment that already has strong agents that play well. Normally mutli-agent environments often do not come with strong baselines meaning it is difficult to say if during testing, the agent really is playing at a high-level apart from defining some custom metrics by hand. With the released data of the proposed competition, high-level agents will exist (of varying skill-levels as well) to then test agents exhaustively without needing custom metrics or subjective qualitative judgement.

1.7 Metrics

As the Lux AI Challenge is a multi-agent competition, we use a Trueskill [4] ranking system to rank agents, which allows ranking players in 1v1 matches. At a high-level, players are each given an initial ranking point estimate μ and confidence score σ, with their ranking score being computed as $\mu - 3\sigma$. After each match, the Trueskill algorithm updates each players μ, σ values according to the match result. Trueskill will generally increase the winning player's μ values and decrease the losing player's μ values, and decrease all players' σ values. To determine awards and final rankings, once submissions are closed we run a 14 day evaluation period where we run as many matches between submissions as possible to converge the rankings.

1.8 Baselines, code, and material provided

For Season 3, we will provide rule-based and RL baselines. We will further provide useful utilities to scrape public match data / process the Jax-environment data for Imitation Learning solutions to use. Below we detail the baselines that are provided.

1.8.1 Rule Based Baseline

For competitors seeking to write a purely rule based agent to compete in the challenge, we have provided a simple baseline as well as resources regarding the top open-sourced solutions from the Season 2 competition. The simple baseline is a barebones kit that enables controlling each unit on ones team to move to the closest asteroid, fire the mining laser at it until resources are dropped, and then move to pick up the resources.

Moreover, the starter kit comes in several languages: Python, C++, and Javascript/Typescript; supporting one of our goals with respect to inclusivity as we try to make the competition open to as many competitors of different backgrounds as possible. This has shown in the past to invite strong competitors to compete, as in particular in Season 2 out of the top 5 on the leaderboard, one team uses C++ while another team uses Typescript, with the rest being in Python. We further expect the competition community to help write starter kits in other languages as this has happened in each of the past iterations of the challenge (Rust, Java, and Go kits were provided via open-source contributions).

1.8.2 RL Baseline

The RL Baseline is a simple adaption of the CleanRL [5] Proximal Policy Optimization (PPO) code which is easy an straight to read. The reward function provide simply encourages units to get close to asteroid chunks, and then encourages units to have as much ore in their cargo as possible. The resulting trained agent simply learns to move units close to asteroids (without crashing into them as depending on active mechanics that can cause the unit to be destroyed) and fire the mining lasers at the asteroid and gather resources.

Note that the baseline keeps it simple by training the agent to play only one game at a time instead of optimizing over the full 5-match sequence.

1.9 Website, Tutorial, and Documentation

The main competition page will look similar to the Lux AI Challenge Season 2 page, but with the contents updated to reflect the this Season 3 proposal. Prior to the competition page being launched the Lux AI home page will be updated upon paper acceptance to reflect the new competition details. These pages will host information about the current competition from simple Colab/Notebook tutorials to game specifications.

Concretely the following tutorials will be provided:

- Tutorial on how to use the Jax based environment
- Tutorial for how to build a rules-based agent
- Tutorial for how to build and train an RL agent

2 Organizational aspects

2.1 Protocol

Competition Platform: The competition will be hosted on the Kaggle platform under the simulations category of competitions. For a participant to compete, they must first create an Kaggle account and then register for the competition on the competition page. The Kaggle platform has hosted hundreds of competitions previously and provides state-of-the-art technology and management for large-scale competitions capable of supporting of 1000s of teams and many more submissions. Moreover Kaggle has already hosted several past iterations of the Lux AI Challenge successfully already.

Submission: Participants can then submit their submission code in the form of a .tar.gz file which packs all their code into a single file (participants can follow the submission instructions provided in the starter kits and documentation to learn how to generate a working submission file). They can then simply upload their file to the submission portal for the competition. Participants are limited to 5 submissions per day and we only actively evaluate their latest 3 submissions in each competition stage. Submissions are evaluated by selecting another submission, running a match using the agents in the pair of submissions, and changing the ranking points of both submissions according to the Trueskill ranking system. See Sec. **??** for specifics on evaluation and metrics in all stages.

Testing: A beta test will be run using the initially proposed game mechanics and theme three weeks prior to the official competition start. Note that as we have collaborated with Kaggle multiple times successfully in the past, we already know most of the system and hardware works, so the beta test primarily will be used to test and revise the game mechanics.

2.2 Rules and Engagement

2.2.1 Rules

- Maximum of 5 submissions are allowed per day.
- Teams may not share code with each other privately. Any shared code must be publicly shared to all other teams. Kaggle's team actively monitors teams and verifies submissions when cleaning up final leaderboard rankings for us.
- Reproducible training and inference code is required for teams to be eligible for prizes.

A full version of the rules above that will be similar to what we will write for the proposed competition can be found on the Season 2 competition page here. We plan to actively monitor submissions and ensure teams are competing fairly under the rules as done in past seasons. The rules work towards supporting fair competition between competitors as well as ensuring they can't easily overload the competition servers so that all competitors get equal match time for determining submission rankings. Moreover, the rules work towards encouraging the open-sourcing and reproducibility of results, which will be critical for various post-competition analysis.

2.2.2 Engagement

A Discord server for the Lux AI Challenge is maintained, and will be moderated for the competition by Organizers. Kaggle also provides a discussion forum for all competitors for the competition. Moreover, participants may also discuss/communicate via Github issues and Github discussions. Finally, we provide the following email for those who wish to communicate via email: support@lux-ai.org. We will communicate important updates and announcements via Discord and the competition's discussion forums.

2.3 Schedule and readiness

- April 1st - July 1st: Build out the proposed game mechanics and play test them internally amongst the organizers.
- July 1st: Begin the beta test of the competition. Revise game mechanics based on user feedback and fix any bugs that show up.
- July 24th: Beta version concludes. Official competition starts with finalized game rules that will not be changed.
- October 18: Final submissions are due. Final evaluation period begins.
- October 31: Final evaluation period ends and awards are given out to top eligible teams.

2.4 Competition promotion and incentives

We plan to promote participation through twitter and emails to potential competitors and researchers. Concretely we will leverage the following mediums

- Lux AI Challenge Discord - 3000+ users
- Kaggle Mailing List - 10,000,000+ users
- Twitter/X Accounts of Kaggle / Organizers - Combined 300k Followers

2.4.1 Incentives

For the Season 3 challenge there will be a $50,000 prize pool distributed amongst the top 8 teams on the final leaderboard, graciously provided by Kaggle.

To ensure more active participation throughput the 3 month official competition period, we adopt sprint awards. At the end of each month we give the top two teams on the leaderboard the sprint award that have yet to win it (which comes with custom merchandise from the Lux AI Challenge organization).

We also will consider giving co-authorships for the subsequent NeurIPS 2025 Datasets and Benchmark track Submission for top solutions if those solutions become a core part of the submission.

3 Resources

3.1 Organizing team

The organizing team consists of a diverse group of professionals from different backgrounds and with different experiences relevant for the smooth running of a high research-value, accessible, and inclusive AI competition.

- **Stone Tao** is a Ph.D. candidate at UC San Diego, advised by Professor Hao Su. His research focuses on scalable embodied AI, RL, as well as building foundational benchmarks for advancing research in multi-agent RL and robotics. He is also the co-founder of the Lux AI Challenge and is the lead researcher and engineer for the Lux AI Challenges. His research is funded by the NSF Graduate Research Fellowship Program.
- **Akarsh Kumar** is a Ph.D. candidate at MIT, advised by Professor Philip Isola. His research broadly revolves around the question of how intelligence arises and the emergent properties of learning algorithms in increasing complex environments. To that end he employs AI Generating Algorithms (AI-GAs) (meta-learning architectures, learning algorithms, and environments) and Open-Ended quality-diverse evolutionary algorithms. His research is funded by the NSF Graduate Research Fellowship Program.
- **Bovard Doerschuk-Tiberi** is a full stack software engineer on Kaggle's Competitions Team. Bovard studied computer science and mathematics at Montana State University before getting a Masters in Data Science from UC Berkeley. In between he served as a Peace Corps Volunteer in Burkina Faso teaching computer literacy. Before Kaggle, Bovard worked at Google Nest. Bovard enjoys AI programming competitions, chess, reading, and getting outside with his family.
- **Isabelle Pan** is a HCI researcher at UC San Diego and one of the co-founders of the Lux AI Challenge. She leads the design of all user-facing applications from the game visualizer to community management. She previously led the design of all previous Lux AI Challenge seasons, and has conducted HCI research at the Design Lab at UC San Diego.
- **Addison Howard** is the Head of Competitions Program Management at Kaggle. He holds Bachelors degrees in Mathematics, Economics, and Accounting from Furman University, and a Masters degree in Accounting from Wake Forest University. He's helped launch over 100 machine learning competitions on Kaggle.
- **Hao Su** is an Associate Professor of Computer Science at the University of California, San Diego. He is the Director of the Embodied AI Lab at UCSD, a founding member of the Data Science Institute, and a member of the Center for Visual Computing and the Contextural Robotics Institute. He works on algorithms to model, undertand, and interact with the physical world. His interests span computer vision, machine learning, computer graphics, and robotics – all areas in which he has published and lectured extensively.

  Hao Su obtained his Ph.D. in Computer Science from Stanford. Prior to that, he obtained a Ph.D. in Mathematics from Beihang University, China. At Stanford and UCSD he developed widely used datasets and softwares such as ImageNet, ShapeNet, PointNet, PartNet, SAPIEN, and more recently, ManiSkill. He also developed new courses to promote machine learning methods for 3D geometry and embodied AI. He served as the Program Chair of CVPR, and Area Chair or Associate Editor for top conferences and journals in computer vision (ICCV/ECCV/CVPR), computer graphics (SIGGRAPH/ToG), robotics (IROS/ICRA), and machine learning (NeurIPS/ICLR). He received the SIGGRAPH Best Ph.D. Thesis Award Honorable Mention and the NSF CAREER Award.

3.2 Resources provided by organizers

As the competition runs on Kaggle, all competitors are provided free GPU/TPU resources through Kaggle Notebooks which are similar to Colab Notebooks. The resources are limited to 30 GPU hours per week.

The organizing team will provide technical support on all parts of the competition, including maintaining the Jax based environment code, maintaining the competition servers, and maintaining baselines.

3.3 Support requested

It would be great if NeurIPS could help provide financial assistance to top teams to attend the conference in-person so they can present their work (travel-costs, lodging costs).

References

[1] Open Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michaël Mathieu, Nat McAleese, Nathalie Bradley-Schmieg, Nathaniel Wong, Nicolas Porcel, Roberta Raileanu, Steph Hughes-Fitt, Valentin Dalibard, and Wojciech Marian Czarnecki. Open-ended learning leads to generally capable agents. CoRR, abs/2107.12808, 2021.

[2] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[3] Alexander Nikulin, Vladislav Kurenkov, Ilya Zisman, Viacheslav Sinii, Artem Agarkov, and Sergey Kolesnikov. XLand-minigrid: Scalable meta-reinforcement learning environments in JAX. In Intrinsically-Motivated and Open-Ended Learning Workshop, NeurIPS2023, 2023.

[4] Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill(tm): A bayesian skill rating system. In Advances in Neural Information Processing Systems 20, pages 569–576. MIT Press, January 2007.

[5] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G. M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. J. Mach. Learn. Res., 23:274:1–274:18, 2022.