JaxLife: An Open-Ended Agentic Simulator

Chris Lu*, Michael Beukman*, Michael Matthews and Jakob Foerster

FLAIR, University of Oxford, United Kingdom
*Equal Contribution
christopher.lu@exeter.ox.ac.uk, mbeukman@robots.ox.ac.uk

Abstract

Human intelligence emerged through the process of natural selection and evolution on Earth. We investigate what it would take to re-create this process *in silico*. While past work has often focused on low-level processes (such as simulating physics or chemistry), we instead take a more targeted approach, aiming to evolve *agents* that can accumulate open-ended culture and technologies across generations. Towards this, we present JaxLife: an artificial life simulator in which embodied agents, parameterized by deep neural networks, must learn to survive in an expressive world containing programmable systems. First, we describe the environment and show that it can facilitate meaningful turing-complete computation. We then analyze the evolved emergent agents' behavior, such as rudimentary communication protocols, agriculture, and tool-use. Finally, we investigate how complexity scales with the amount of compute used. We believe JaxLife takes a step towards studying evolved behavior in more open-ended simulations.[1]

Introduction

Human capabilities, culture and intelligence have emerged from open-ended evolution on Earth (Darwin, 1859). It follows that a multi-billion year full-fidelity physics simulation of Earth could produce similarly-capable beings. However, such an endeavor is clearly computationally infeasible. To reduce computational costs, one can reduce the fidelity of the simulation, raising the question of which components are necessary for the desired behavior. To answer this, we must specify what behavior or capabilities we would like to potentially emerge from the simulation.

One such objective is to evolve agents that are capable of advanced reasoning and tool-use (Parisi, 1997). After all, many of humanity's recent achievements involve mathematical reasoning and technical prowess. It may be the case that low-level control and perception—aspects that many simulations aim to reproduce (Dittrich et al., 2001; Hutton, 2002)—are not necessary to evolve these capabilities. Indeed, even evolving morphologies, as many simulations do (Sims, 1994; Silveira and Massad, 1998; Spector et al., 2007; Bessonov et al., 2015; Pathak et al., 2019; Heinemann, 2024), may not be necessary for the evolution of advanced reasoning.

For this reason, we focus on the evolutionary advancements that make humans *different* from other animals. Recent trends in anthropology focus on the idea of "cultural accumulation" (Henrich, 2015) as being the primary evolutionary origins of human intelligence. Cumulative culture is characterized by large amounts of social learning and the persistence and continuous advancement of shared knowledge. Muthukrishna et al. (2018) builds a simple computational model of the emergence of cultural accumulation and finds that it can be facilitated by a small bias towards social learning: If indeed this is humanity's key defining feature, it may not be difficult to replicate *in silico*.

Our work is not the first to investigate the emergence of intelligent behavior. Prior works have modeled the emergence of cumulative culture through agent-based models (Muthukrishna et al., 2018; Lu et al., 2022b, ABMs), which are high-level statistical models of agents. However, ABMs have not produced agents that are capable of advanced reasoning and instead model simplified high-level evolutionary dynamics. Similarly, other work in deep reinforcement learning has produced emergent social behaviors (Johanson et al., 2022), communication (Chaabouni et al., 2021), and tool-use (Baker et al., 2019) in games. Each of these settings suffers from the same failure mode: Their environments are not *expressive enough* to produce truly open-ended expression. For example, agents in these environments cannot reasonably communicate about mathematics or build advanced machines.

Our work aims to address this gap by allowing high-level agents to interact with and program composable robots that can express useful and meaningful Turing-complete behaviors. We present JaxLife, an artificial life simulator capable of expressing meaningful, Turing-complete behaviors. Our simulator is written entirely in JAX (Bradbury et al., 2018), meaning it can run on hardware accelerators and easily scale to multiple devices. Our contributions are as follows:

1. We design and implement an agentic simulator where evolved agents must survive, and are able to interact and program robots. We show that these robots can be pro-

[1]Our code is available at https://github.com/luchris429/JaxLife.

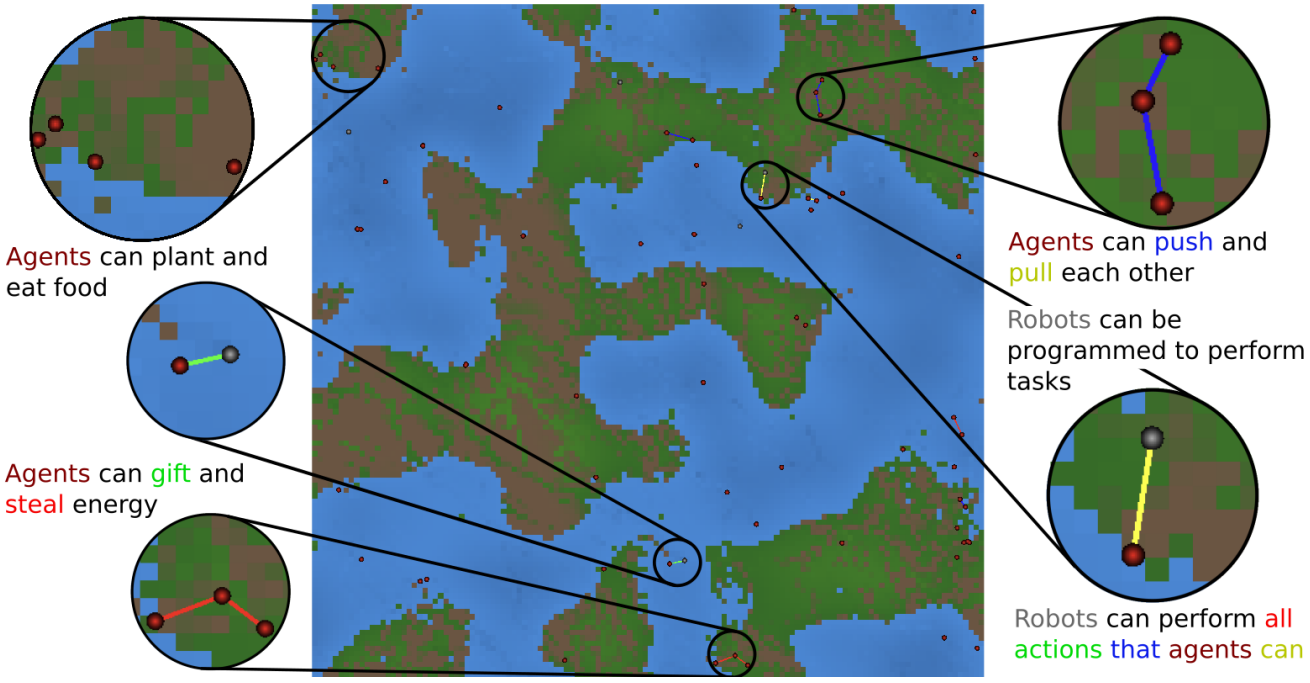


Figure 1: JaxLife is an ALife simulation containing agents that evolve through natural selection and programmable robots that can be requisitioned as tools. The color of the lines indicate the actions performed. Agents are red while bots are grey.

grammed as useful tools and express meaningful Turing-complete dynamics.

2. We demonstrate the emergence of rudimentary agriculture, tool-use, and communication.

3. We provide initial estimates for how important features of this simulation scale with the amount of compute provided, enabling rough estimates of expected future behaviors.

Simulation Description

Overview

Our simulation consists of three primary components, terrain, agents, and robots. At every step, all agents simultaneously observe the area around them and perform a set of actions. These actions may influence other agents, the robots, or the terrain. Robots are programmable systems with the same action space as agents. Agents evolve and change their behavior through the evolution of their controlling neural networks, while the terrain controls how difficult certain actions are and how much energy is available. Due to the limited amount of energy, there is selection pressure and agents that tend to eat and reproduce more will tend to pass on their genes more frequently. Agents can also control the terrain by terraforming it, thereby changing its properties. Finally, robots are systems that do not evolve, but can be programmed by agents. These robots possess a large amount of potential complexity and can execute useful behaviors to help agents survive.

Terrain

The terrain is divided into a grid of cells, each cell possessing several attributes. The primary attributes of each cell are how much energy it has and an *energy gain* amount, indicating how much energy each cell gains per timestep. Each terrain cell also has a cost associated with each agent action. Finally, each cell has an associated information bit that can be read from and written to by bots but not agents.

We implement a weather and climate-like system. At every timestep, the *base* terrain is slightly altered by adding a small number to the angles used to generate the Perlin noise map (Perlin, 1985). The attributes of each cell also slowly regress to this *base* state. The speed of this is proportional to the cell's maximum energy amount. This, for instance, can simulate long-term effects such as continental shifts.

Using this system, we can represent slowly changing landscapes, as well as the natural tendency of nature to return to its base state if not continually maintained. Finally, since the regression speed is different for different regions, the map contains high-maximum energy areas that quickly revert to their base state and lower-energy areas where changes the agents make have more permanence.

Agents

Agents require energy to survive and evolve through the process of natural selection. Agents can gain energy by performing the `EAT` action on terrain cells that have energy. Energy is consumed when performing any action (with the

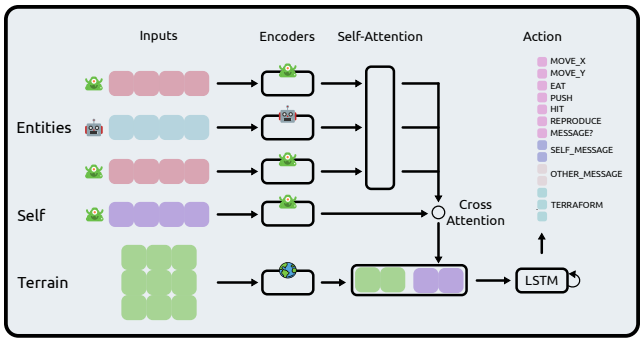


Figure 2: The agents' network architecture. Robots, agents and the terrain each have different encoders. Entity embeddings are processed using a self-attention block, followed by cross-attention using the agent's own embedding. This, concatenated with the terrain features, is the input to an LSTM that outputs an action vector. The terrain encoder is 1×1 convolution followed by a fully connected layer.

rate being determined by the current terrain cell). The agents also have a continuous (x, y)-position, which they can control by their MOVE_X and MOVE_Y actions. Agents have two types of messages, *self* and *other*, both of which are observed by the agent itself and other agents around it. The agent controls its own *self*-message, whereas the *other*-message can only be changed by other agents. If the agent chooses to send a message to other agents, it sends its self-message to the closest $N_{\text{agents}}^{\text{view}}$ agents, which updates these agents' *other* message property. This allows agents to communicate with themselves and others. As mentioned above, the agent can also spend energy to alter the properties of the terrain. Agents also age, and have to use more energy to survive as they become older. Every timestep, agents receive an observation consisting of the terrain's attributes in a region around it, the attributes of the closest $N_{\text{agents}}^{\text{view}}$ agents and $N_{\text{bots}}^{\text{view}}$ robots, as well as its own attributes.[2]

Network Architecture We parametrize agents using neural networks that process observations and output an action vector (see Fig. 2). All actions can be performed simultaneously, and the strength of the action's effect is determined by the magnitude of the corresponding entry. The network architecture consists of different encoders for agents, robots and the terrain. The entity embeddings are processed by a self-attention and multi-headed attention block (Vaswani et al., 2017). The result is concatenated with the terrain features and passed to an LSTM (Hochreiter and Schmidhuber, 1997) to incorporate memory. While the agent's networks do not change during their lifetimes, due to the use of a recurrent network, they are able to adapt their behavior when encountering the same observation multiple times.

[2] $N_{\text{agents}}^{\text{view}}$ and $N_{\text{bots}}^{\text{view}}$ are fixed hyperparameters of the simulation.

Reproduction Since crossover for fixed-topology neural networks is challenging (Haflidason and Neville, 2009; Pretorius and Pillay, 2024), our agents reproduce asexually, and must have a minimum amount of energy before they can execute the REPRODUCE action. Reproduction copies the agent's weights to a child agent, and random perturbations are also added to these weights. We reinitialize the population—with random networks—if all agents die.

Robots

We design the robots to have a similar action space to the agents; however, since robots do not reproduce, and we wish them to be programmable, how they obtain their actions must be different to the agents. To simulate the technological advantage of machines, robots update multiple times for every agent update cycle and do not use energy. We decide upon the following scheme and showcase its theoretical complexity and practical uses in the next section.

Each robot has a *program* and *memory*, each of the same size N_{prog}. At every step, each robot receives the messages from the two closest entities to it, breaking ties using x-position. This message is the SELF_MESSAGE of an agent or the *memory* of a bot; each of these is also of size N_{prog}.

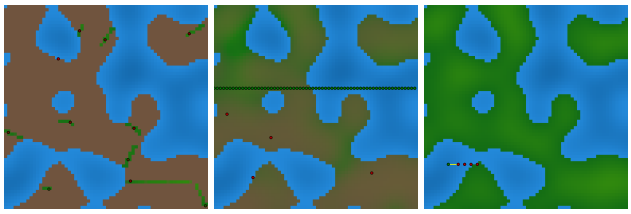
The program is a description of a function $f : \mathbb{R}^{N_{\text{prog}}} \times \mathbb{R}^{N_{\text{prog}}} \times \rightarrow \mathbb{R}^{N_{\text{prog}}}$, where the action $a = f(\text{mem}, m_1, m_2)$ is the action performed. This action, if the WRITE_SELF_MESSAGE entry is set, can also update the robot's memory. Whenever another robot or agent sends this robot a message, it is interpreted as changing the robot's program.

We note here that the dimension of these messages N_{prog} is always more than the action's dimensionality N_{act}, therefore, only the first part of the output is used as the action. The final entry, in particular, is interpreted as solely an information bit that allows robots to store and manipulate information.

Instructions We have the following instructions:

- COPY: $f(\text{mem}, m_1, m_2) = m_1$
- NOOP: $f(\text{mem}, m_1, m_2) = \text{mem}$
- PRODUCT: $f(\text{mem}, m_1, m_2) = \text{mem} \odot m_1$
- FMA: $f(\text{mem}, m_1, m_2) = \text{mem} \odot m_1 + m_2$
- XOR: $f(\text{mem}, m_1, m_2) = (2(\text{mem}^i \oplus m_1^i) - 1) \odot m_1 \odot \text{mem} + (1 - m_1)\text{mem}$, where $\oplus$ is logical XOR, and the superscript i indicates the information bit of the message.
- NAND: $f(\text{mem}, m_1, m_2) = 1 - (m_1 \odot m_2)$

The final operation, LOOKUP, uses the information bit from mem, m_1 and m_2 to construct a 3-bit number, from 0 to 7 inclusive, and uses this index to look up into a table as specified by the program. The result is written to the information bit of the action and replaces the bot's current memory if



(a) (b) (c)

Figure 3: Three snapshots of manually-designed useful bots. (a) shows patrolling bots, (b) shows mass terraforming and (c) shows the transport bot, which is pushing the agents.

the WRITE_SELF_MESSAGE entry of the action vector is set. This allows bots to compute complex functions, and to store the result in their information bit.

Analyzing Environment Complexity

Here we discuss the capabilities of the robots in JaxLife. We begin by illustrating useful and practical bots. We then go on to prove our simulation is Turing-complete (Turing, 1936), by showing that it can execute Rule 110 (Cook et al., 2004; Cook, 2009). Finally, we describe how robots can also compute arbitrary boolean functions.

Useful Machines

In practice, we expect the programs created by agents to be relatively simple at first. However, to illustrate what is practically possible, we manually design some useful robots that can easily be constructed, see Fig. 3.

Automated Terraforming These robots are programmed to terraform a large region of the map. This can be implemented by programming robots to move in a consistent direction (e.g. down the map), and perform the TERRAIN_ENERGY_GAIN action. This leads to the terrain becoming more fertile— leading to more food over time.

Patrolling & Oscillating Robots can also execute more complicated oscillatory behavior, by using the terrain's information bits as waypoints to oscillate over an arbitrary line. This is achieved by reading the current terrain bit, and using the XOR instruction, which inverts the action when paired with an appropriate m_1 sent by the closest agent.

Transportation Robots can also be used to transport agents in a more energy-efficient manner than walking. Suppose the robot's memory is zero, except for the entries associated with MOVE_X and PUSH. The robot's program is the one defined as $f(\text{mem}, m_1, m_2) = \text{mem} \odot m_1 + m_2$. This means that the robot will push and move nearby agents whenever they send messages with the move and push entries being nonempty.

Communicating Robots can also propagate information across space. We implement this proof-of-concept by using the lookup table instruction, with the lookup table simply copying the information bit of m_1. Arranging the robots in a chain such that the robot to the left of it is closer than the one to the right allows the information bit to pass from the left to the right across the map.

Turing-Complete Computation

We now move on to proving that JaxLife can facilitate universal computation by reducing it to Rule 110—a common technique that has been used in several prior settings; for instance, in Baba is you (Rodriguez, 2019; Su, 2023), the Micron Automata Processor (Wang and Skadron, 2015) and Petri Nets (Zaitsev, 2018).

Constructing Rule 110 Elementary cellular automata (Wolfram, 2002, ECA) are simple, one-dimensional rules that can lead to complex patterns. An ECA is defined on a grid, with an initial state in the top row, and a local transition rule that transforms the row into the next one; every time step this is applied and a new row is appended. In ECAs, every cell is binary-valued, and the local transition rule of a particular cell depends on it, as well as its left and right neighbours. Using these three binary digits a, b and c, we can construct a binary number abc_2 such that $0 = 000_2 \leq abc_2 \leq 111_2 = 7$. Each ECA is then an 8-dimensional lookup table, giving the next value of the center cell for each possible three-digit binary number. Rule 110 has the pattern described in Table 1:

Table 1: Rule 110

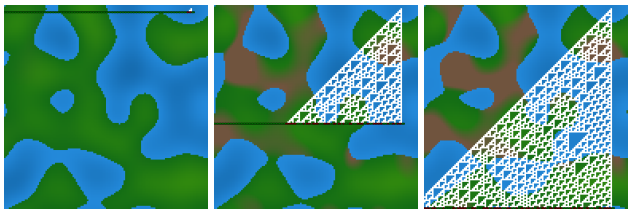
Pattern (lcr)	111	110	101	100	011	010	001	000
Next Value	0	1	1	0	1	1	1	0

Cook et al. (2004) proved that Rule 110 is capable of universal computation, i.e., that it is Turing complete. Here we describe a way in which a particular configuration of JaxLife results in an implementation of Rule 110—showing that it, too, is capable of universal computation. The core idea is to use the terrain's information bit to store the previously computed rows, and to have a single row of robots act as the latest state while moving down the map.

Program The program has the LOOKUP_TABLE instruction bit set as 1, all other instruction bits set to zero, and the lookup table is defined as in Table 1.

Memory The memory is empty, except for three locations:

1. MOVE_Y $= 1$: In order to move in the y-direction.
2. WRITE_TERRAIN $= -1$: To write the previously-computed bit to the terrain (note that $+1$ indicates reading and -1 indicates writing).
3. UPDATE_MEMORY $= 1$: To save the updated cell state to the robot's memory.



(a) (b) (c)

Figure 4: Three snapshots of the computation of Rule 110. In this figure, terrain bits are explicitly rendered.

Result The result of this construction is that at each step, each bot performs the following in order:

1. Compute Rule 110 using the information bits of the robot immediately to its left and right, as well as its own bit.
2. Write the updated state to its memory's information entry.
3. Write this information bit to the terrain
4. Move one position forward in the x-direction

Therefore, the robots collectively implement Rule 110, with the previous iterations' results being saved on the terrain bits of the world. The initial information bit in each robot's memory defines the initial state of the simulation, and can be set arbitrarily. Under the assumption of an infinitely large simulation, this system of bots is Turing-complete, due to it being able to implement Rule 110 with an arbitrary initial pattern (Cook et al., 2004). An illustration of this process is presented in Fig. 4.

Functional Completeness

Functional completeness of a set of logical operators states that all possible boolean functions can be realized by composing elements from this. Functionally complete of complete operators include $\{\texttt{NAND}\}$, $\{\texttt{NOR}\}$, $\{\texttt{OR}, \texttt{AND}, \texttt{NOT}\}$ (Enderton, 2001).

NAND We note that one of our instructions described above—the lookup table—can implement any three-bit truth table, which includes NAND between two inputs, which we refer to as the NAND instruction. We now describe a way to compose logical operators, resulting in functional completeness.

Composition Suppose we have a vertical line of n input bots $A_1, \ldots, A_n$. We can compute any possible function by having at most n columns of n bots each, that process these input bits, such that the final column has one bot which represents the output. This final bot could then write to the terrain. We describe now how to perform the two necessary operations, NAND and passthrough (i.e., identity).

Consider two robots A and B in row i that contain the input bits in their information slot. Suppose robot R_j^{i+1}—in row $i+1$—is positioned such that its closest two agents are A and B. Then, by using the NAND program, it can compute A NAND B and store the result in its memory.

To pass bit A unchanged to the next row, suppose that bot A is closest to bot R_j^{i+1}; it can then use the COPY action to copy the bit unchanged (this could also be implemented using the lookup table to copy the closest robot's bit).

Temporal Order Due to the temporal nature of our simulation, we cannot compute an arbitrary function in one step. We note that in our above construction, initially, only the input column is correct. Every step ensures that the subsequent column computes the correct function. At the end, the final bot's memory would be correct. Using these two operations, and the temporal structure, the robots in JaxLife can compute any n-variable binary-valued function for arbitrary n.

Results

In this section we present the empirical results obtained when running JaxLife. We run the simulation for 2^{16} timesteps using a single NVIDIA A100 GPU. We first describe behaviors that we observed, with 128 agents and 32 bots, and then examine quantitative metrics of the complexity of the simulation. Finally, we observe how these results change with scaling the number of agents in the environment.

Emergent Behaviors

Agriculture and Terraforming: Fig. 5 shows that the agents manage to substantially modify their environment, creating structured regions in which food is produced. We find that this corresponds to a period in which the agents construct striped diagonal regions over which they travel.

Tool-Use: In Fig. 6 we visualize the behavior of the programmed bots. In particular, we find that the bots play a significant role in the stable terraforming behavior we found in Fig. 5, as they move at the same diagonal stripes as the agents while also helping with food production.

Communication In Fig. 8f we use "saliency" (i.e., the derivative of the agent's action with respect to the input (Simonyan et al., 2013)) as a measure of the use of communication. We find that, over time, agent behaviors tend to be more sensitive to their communication channels, suggesting that they can use the channel to influence each other.

Scaling Results

To quantitatively measure the complexity of the simulation, we consider several metrics in Figure 8.

1. *Energy Consumption*: The total amount of energy available to a civilization is often considered a marker of technological advancement (Kardashev, 1964; Gray, 2020). Inspired by this, we measure the amount of energy per agent consumed by the population of JaxLife.

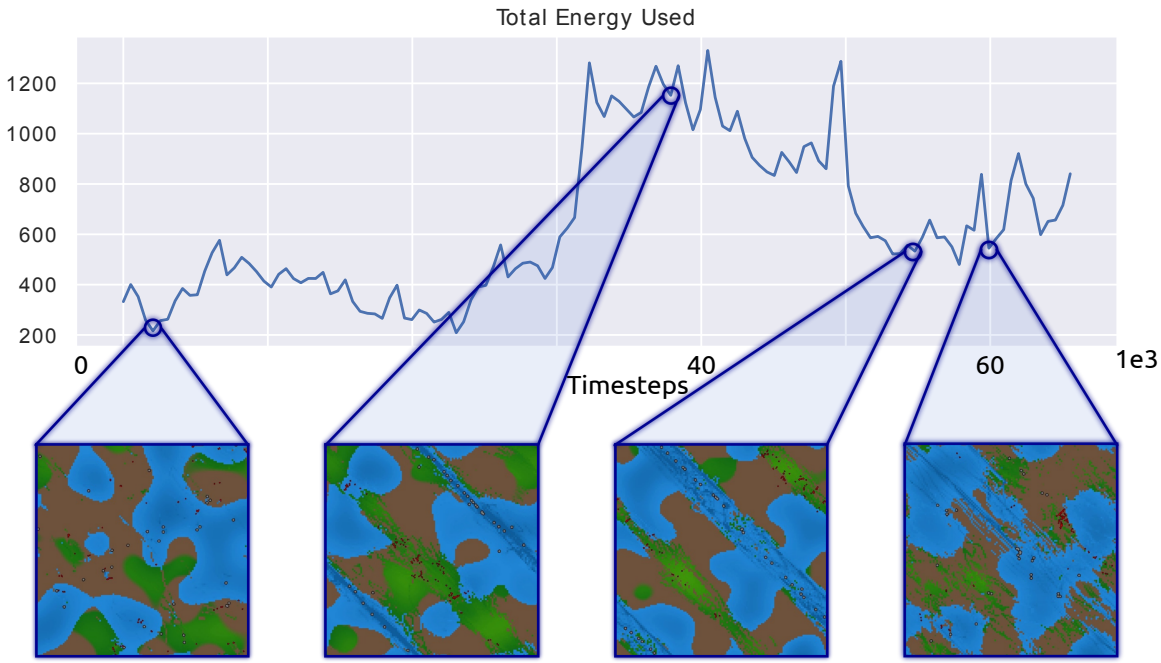


Figure 5: Snapshots of the simulation at various points in time. At first, the world state is effectively random. At some point, however, agents start forming a diagonal bridge, and travel along it. This coincides with a large spike in energy usage. Later on the bridge collapses and the total energy usage declines.

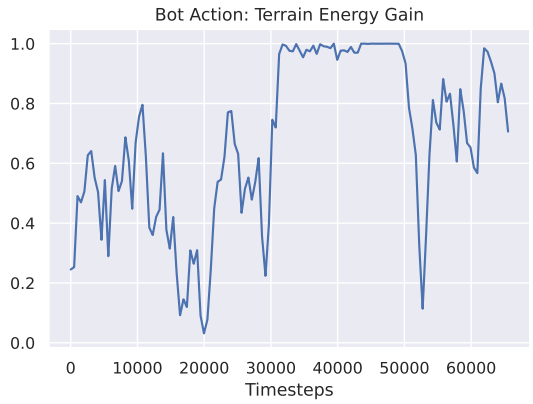


Figure 6: The average value of all of the bots' "Terrain Energy Gain" action, which increases the amount of energy a given position produces. We observe that the time period in which this is high corresponds to peaked region in Fig. 5 where the bots have constructed a bridge.

2. *Terrain*: Since agents have the ability to transform the terrain, we measure how much the terrain changes over time, focusing on the total amount of energy produced.

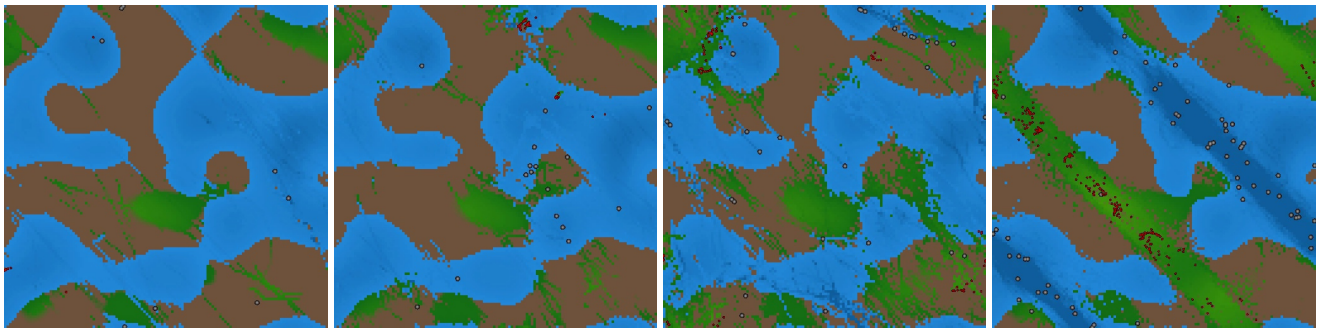
3. *Action Selection*: We analyze the distribution of agent actions, i.e., how agents are spending their energy and how this changes over time. More specifically, we show how often agents are using the EAT action, as changes in this value would be evidence for selection.

4. *Input Saliency* How much the agents use the messages received from other agents in deciding their actions, computed as the partial derivative of the action vector with respect to the observations. In particular, we investigate *communication* saliency (e.g. the impact of the messages on the action) and *bot* observation saliency.

We present our primary results in Fig. 8, indicating the value of each metric across time. We also vary the number of agents to provide an indication of how the results vary with the amount of compute we provide. First, in Figs. 8a and 8d, we see that the amount of energy used (both overall and per agent) remains relatively consistent if we have less than 256 agents. If we have 256, there are more pronounced peaks and troughs, indicating that the agents go through phases of using a large amount of energy, followed by sharp declines. We find this is caused by the primary food source becoming exhausted.

In most cases, except if there are only 32 agents, the agents quickly learn to perform the eat action almost constantly (Fig. 8b). We find that the terrain's average energy gain amount—indicating how fertile the land is—- has a decreasing trend with 256 agents, but remains relatively constant if we have fewer agents.

In Fig. 8c, we further find that most settings result in



(a) 32 Agents (b) 64 Agents (c) 128 Agents (d) 256 Agents

Figure 7: Visualization of the results of scaling the number of agents after 18000 steps. At too few agents, we do not observe any meaningful emergent patterns. However, as we increase the number of agents, large-scale patterns emerge, such as the diagonal stripes in (d).

the maximum number of agents being alive, but that there are some spikes, and a notable drop in the number of alive agents, coinciding with the large decrease in total energy used. Notably, this shows that the population does not entirely die out, and the automatic reinitialization does not occur.

Finally, we note that communication saliency (Fig. 8f) increase gradually over the course of the simulation. Interestingly, with 256 agents, the communication saliency remains relatively constant at near zero, possibly indicating that the agents have learnt to not use communication if there are too many potentially random agents.

In summary, we note that there are differences in behavior and metrics as we increase the number of agents in the simulation. We further see non-trivial progression over time, suggesting that running the simulation for longer and with more agents would be interesting.

Related Work

Early instantiations of the ALife concept include Tierra (Ray, 1996) and Avida (Ofria and Wilke, 2004), which evolved computer programs that compete for finite resources. Geb (Channon, 2006) was the first simulation said to pass the activity statistics test (Bedau et al., 1998) and to be a truly open-ended system. However, all these simulations are relatively simple. Despite their simplicity, however, these simulations can also be used to study the emergence of behavior—such as associative learning—and to perform controlled experiments to test various evolutionary theories (Pontes et al., 2020; Moreno and Ofria, 2022; Ferguson and Ofria, 2023).

Recent work, often stemming from the reinforcement learning (RL) community (Johnson et al., 2016; Charity et al., 2023; Rutherford et al., 2023; Matthews et al., 2024), contains some elements of ALife simulations, and open-endedness (Stanley et al., 2017; Leibo et al., 2019; Clune, 2019) in general. One example is Neural MMO (Suarez et al., 2019), which defines a world of agents competing for a finite amount of resources. These agents are generally trained with RL, instead of competing via natural selection. XLand (Team et al., 2021) similarly defines an expressive collection of multi-agent environments, although much of the interesting variation comes in the form of tasks and worlds rather than agents. However, these environments are generally posed as relatively static learning environments for RL agents instead of environments to study evolutionary and open-ended behavior.

Other works that are more in the spirit of traditional ALife simulations include MineLand (Yu et al., 2024), which defines a cooperative multi-agent world in Minecraft and the work by Park et al. (2023), which simulates a human-like community using pre-trained language models to interact between agents. Lenia (Chan, 2018) is a cellular automata that can result in the evolution of a diverse range of creatures with lifelike properties from low-level building blocks.

Recent hardware-accelerated ALife works include Alien (Heinemann, 2024), which simulates agents that can evolve different morphologies in a particle-based system and Biomaker CA (Randazzo and Mordvintsev, 2023) which evolves artificial plants in a grid-based world. While both are powerful simulations, their focus is more on the evolution of lower-level dynamics as opposed to higher-level agentic *behaviors*.

Conclusion and Future Work

We introduce JaxLife, where agents must survive and evolve within a changing world, with programmable robots affording unbounded complexity. We believe JaxLife could be used to explore alternative routes to technological and cultural advancements, such as the evolution of mathematics. Using JaxLife to perform hypothesis-driven study of important questions in evolution would be a fruitful avenue for future work. For instance, we could change various hyperparameters of the simulation, or disable components such as communication, to see what effect these changes have on the final outcomes. We also note that the simulation results can vary

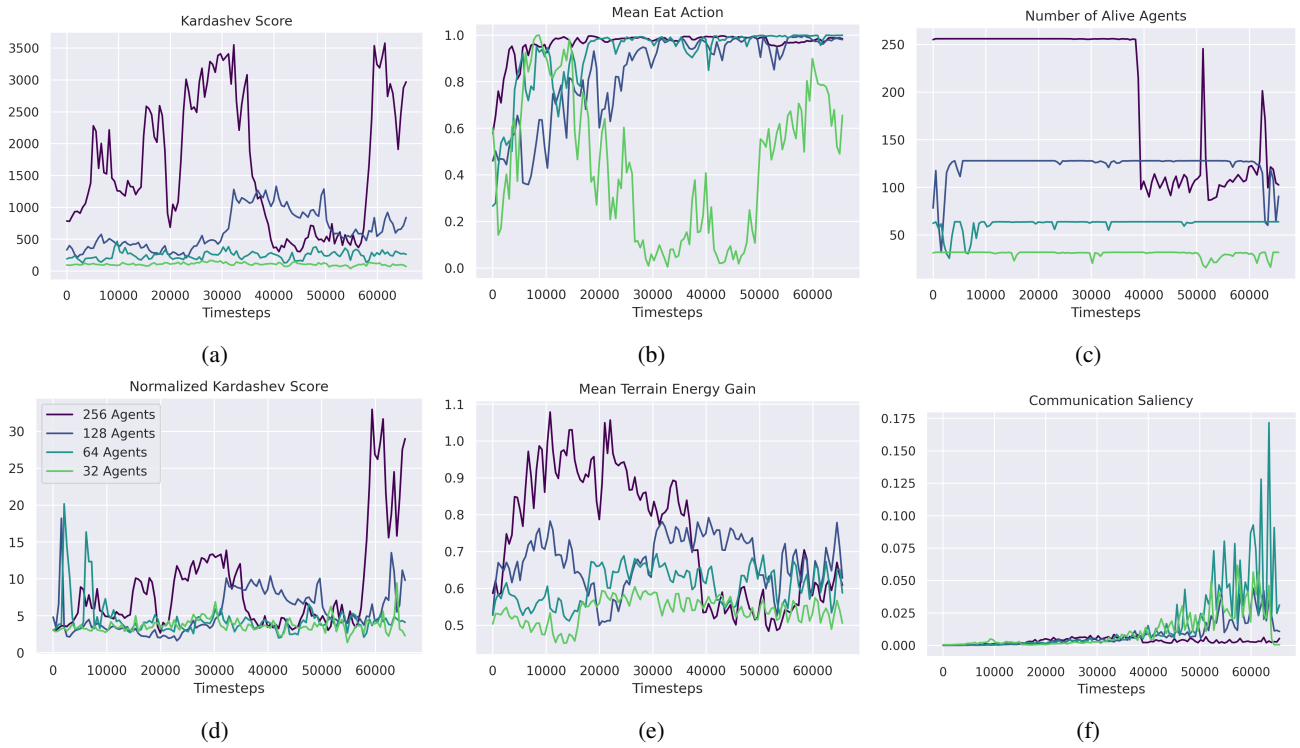


Figure 8: Quantitative metrics over time for different numbers of agents. Kardashev score in (a) is a measure of the total amount of energy used; this is normalised by the number of alive agents in (d).

wildly when changing the initial conditions, so investigating the effects of different initial settings would also be valuable.

JaxLife was designed at a higher level of abstraction to many comparable ALife simulations, completely ignoring the low-level aspects of physics, chemistry and control. This comes from our basic assumption that agents do not need to first evolve basic control and perception to be able to develop higher-level reasoning. In particular, we take an *anthropocentric* view of evolution, completely overlooking early evolutionary advancements in control, perception, and morphology and instead focusing entirely on human evolution. Furthermore, we are potentially limited by having each agent be physically identical, where the only difference between agents is their behavior. Therefore, JaxLife will not be able to examine physical speciation and different morphology-based niches; however, this is not our goal, and there are several prior simulations that are more well-suited to this (Sims, 1994; Silveira and Massad, 1998; Spector et al., 2007; Bessonov et al., 2015; Heinemann, 2024).

There are several promising avenues for future work, such as using other parameterizations to control the agents' behavior, e.g., VSML (Kirsch and Schmidhuber, 2021), where a small number of parameters can define an entire *learning algorithm*, potentially being better suited to evolutionary methods. Another alternative would be to evolve a reinforcement learning objective (Lu et al., 2022a; Jackson et al., 2024) or reward (Sapora et al., 2023) function. Another direction would be to investigate which features of the environment could encourage agents to interact more with the robots; for instance, adding near-extinction events that could encourage agents to store information to be used by future generations. Developing improved quantitative metrics to measure the interaction with the robots would also be promising; for instance, computing the Kolmogorov complexity of the system as a whole (Kolmogorov, 1965). Moreover, investigating the altruistic or selfish nature of agents' behavior within a resource-constrained world would also be interesting (Perolat et al., 2017; Lupu and Precup, 2020).

Ultimately, we believe JaxLife's large amount of potential complexity provides a good testbed for investigating the emergence of higher-level features, such as culture, language and mathematics. Being able to interact with complex computational tools could lead to particularly interesting results when the simulation is run for long enough.

References

Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. (2019). Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*. 1

Bedau, M. A., Snyder, E., and Packard[1], N. H. (1998). A classification of long-term evolutionary dynamics. *Artificial Life: The Proceedings...*, page 228. 7

Bessonov, N., Reinberg, N., and Volpert, V. (2015). How morphology of artificial organisms influences their evolution. *Ecological complexity*, 24:57–68. 1, 8

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs. 1

Chaabouni, R., Strub, F., Altché, F., Tarassov, E., Tallec, C., Davoodi, E., Mathewson, K. W., Tieleman, O., Lazaridou, A., and Piot, B. (2021). Emergent communication at scale. In *International conference on learning representations*. 1

Chan, B. W.-C. (2018). Lenia-biology of artificial life. *arXiv preprint arXiv:1812.05433*. 7

Channon, A. (2006). Unbounded evolutionary dynamics in a system of agents that actively process and transform their environment. *Genetic Programming and Evolvable Machines*, 7:253–281. 7

Charity, M., Rajesh, D., Earle, S., and Togelius, J. (2023). Amorphous fortress: Observing emergent behavior in multi-agent fsms. *arXiv preprint arXiv:2306.13169*. 7

Clune, J. (2019). Ai-gas: Ai-generating algorithms, an alternate paradigm for producing general artificial intelligence. *arXiv preprint arXiv:1905.10985*. 7

Cook, M. (2009). A concrete view of rule 110 computation. *arXiv preprint arXiv:0906.3248*. 4

Cook, M. et al. (2004). Universality in elementary cellular automata. *Complex systems*, 15(1):1–40. 4, 5

Darwin, C. (1859). Origin of the species. In *British Politics and the Environment in the Long Nineteenth Century*, pages 47–55. Routledge. 1

Dittrich, P., Ziegler, J., and Banzhaf, W. (2001). Artificial chemistries—a review. *Artificial life*, 7(3):225–275. 1

Enderton, H. B. (2001). *A mathematical introduction to logic*. Elsevier. 5

Ferguson, A. J. and Ofria, C. (2023). Potentiating Mutations Facilitate the Evolution of Associative Learning in Digital Organisms. volume ALIFE 2023: Ghost in the Machine: Proceedings of the 2023 Artificial Life Conference of *Artificial Life Conference Proceedings*, page 71. 7

Gray, R. H. (2020). The extended kardashev scale. *The Astronomical Journal*, 159(5):228. 5

Haflidason, S. and Neville, R. (2009). On the significance of the permutation problem in neuroevolution. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, page 787–794, New York, NY, USA. Association for Computing Machinery. 3

Heinemann, C. (2024). Alien. https://github.com/chrxh/alien. 1, 7, 8

Henrich, J. (2015). The secret of our success: How culture is driving human evolution, domesticating our species, and making us smarter. In *The secret of our success*. princeton University press. 1

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780. 3

Hutton, T. J. (2002). Evolvable self-replicating molecules in an artificial chemistry. *Artificial life*, 8(4):341–356. 1

Jackson, M. T., Lu, C., Kirsch, L., Lange, R. T., Whiteson, S., and Foerster, J. N. (2024). Discovering temporally-aware reinforcement learning algorithms. *arXiv preprint arXiv:2402.05828*. 8

Johanson, M. B., Hughes, E., Timbers, F., and Leibo, J. Z. (2022). Emergent bartering behaviour in multi-agent reinforcement learning. *arXiv preprint arXiv:2205.06760*. 1

Johnson, M., Hofmann, K., Hutton, T., and Bignell, D. (2016). The malmo platform for artificial intelligence experimentation. In *Ijcai*, volume 16, pages 4246–4247. 7

Kardashev, N. S. (1964). Transmission of Information by Extraterrestrial Civilizations. *sovast*, 8:217. 5

Kirsch, L. and Schmidhuber, J. (2021). Meta learning backpropagation and improving it. *Advances in Neural Information Processing Systems*, 34:14122–14134. 8

Kolmogorov, A. N. (1965). Three approaches to the quantitative definition ofinformation'. *Problems of information transmission*, 1(1):1–7. 8

Leibo, J. Z., Hughes, E., Lanctot, M., and Graepel, T. (2019). Autocurricula and the emergence of innovation from social interaction: A manifesto for multi-agent intelligence research. *arXiv preprint arXiv:1903.00742*. 7

Lu, C., Kuba, J., Letcher, A., Metz, L., Schroeder de Witt, C., and Foerster, J. (2022a). Discovered policy optimisation. *Advances in Neural Information Processing Systems*, 35:16455–16468. 8

Lu, C., Willi, T., De Witt, C. A. S., and Foerster, J. (2022b). Model-free opponent shaping. In *International Conference on Machine Learning*, pages 14398–14411. PMLR. 1

Lupu, A. and Precup, D. (2020). Gifting in multi-agent reinforcement learning. In *Proceedings of the 19th International Conference on autonomous agents and multiagent systems*, pages 789–797. 8

Matthews, M., Beukman, M., Ellis, B., Samvelyan, M., Jackson, M., Coward, S., and Foerster, J. (2024). Craftax: A lightning-fast benchmark for open-ended reinforcement learning. *arXiv preprint arXiv:2402.16801*. 7

Moreno, M. A. and Ofria, C. (2022). Exploring evolved multicellular life histories in a open-ended digital evolution system. *Frontiers in Ecology and Evolution*, 10:750837. 7

Muthukrishna, M., Doebeli, M., Chudek, M., and Henrich, J. (2018). The cultural brain hypothesis: How culture drives brain expansion, sociality, and life history. *PLoS computational biology*, 14(11):e1006504. 1

Ofria, C. and Wilke, C. O. (2004). Avida: A software platform for research in computational evolutionary biology. *Artificial life*, 10(2):191–229. 7

Parisi, D. (1997). Artificial life and higher level cognition. *Brain and Cognition*, 34(1):160–184. 1

Park, J. S., O'Brien, J., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. (2023). Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22. 7

Pathak, D., Lu, C., Darrell, T., Isola, P., and Efros, A. A. (2019). Learning to control self-assembling morphologies: a study of generalization via modularity. *Advances in Neural Information Processing Systems*, 32. 1

Perlin, K. (1985). An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3):287–296. 2

Perolat, J., Leibo, J. Z., Zambaldi, V., Beattie, C., Tuyls, K., and Graepel, T. (2017). A multi-agent reinforcement learning model of common-pool resource appropriation. *Advances in neural information processing systems*, 30. 8

Pontes, A. C., Mobley, R. B., Ofria, C., Adami, C., and Dyer, F. C. (2020). The evolutionary origin of associative learning. *The American Naturalist*, 195(1):E1–E19. 7

Pretorius, K. and Pillay, N. (2024). Neural network crossover in genetic algorithms using genetic programming. *Genetic Programming and Evolvable Machines*, 25(1):1–30. 3

Randazzo, E. and Mordvintsev, A. (2023). Biomaker ca: a biome maker project using cellular automata. *arXiv preprint arXiv:2307.09320*. 7

Ray, T. S. (1996). An approach to the synthesis of life. *The philosophy of artificial life*, pages 111–145. 7

Rodriguez, M. (2019). Baba is turing complete: A sketch of a proof. https://www.twitlonger.com/show/n_1sqrh1m. 4

Rutherford, A., Ellis, B., Gallici, M., Cook, J., Lupu, A., Ingvarsson, G., Willi, T., Khan, A., de Witt, C. S., Souly, A., et al. (2023). Jaxmarl: Multi-agent rl environments in jax. *arXiv preprint arXiv:2311.10090*. 7

Sapora, S., Lu, C., Swamy, G., Teh, Y. W., and Foerster, J. N. (2023). Evil: Evolution strategies for generalisable imitation learning. 8

Silveira, P. S. P. and Massad, E. (1998). Modeling and simulating morphological evolution in an artificial life environment. *Computers and biomedical research*, 31(1):1–17. 1, 8

Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*. 5

Sims, K. (1994). Evolving 3d morphology and behavior by competition. *Artificial life*, 1(4):353–372. 1, 8

Spector, L., Klein, J., and Feinstein, M. (2007). Division blocks and the open-ended evolution of development, form, and behavior. In *Proceedings of the 9th annual conference on genetic and evolutionary computation*, pages 316–323. 1, 8

Stanley, K. O., Lehman, J., and Soros, L. (2017). Open-endedness: The last grand challenge you've never heard of. *While open-endedness could be a force for discovering intelligence, it could also be a component of AI itself*. 7

Su, C. J. (2023). A new approach to turing completeness in baba is you. *International Journal of High School Research*, 5(7):140–146. 4

Suarez, J., Du, Y., Isola, P., and Mordatch, I. (2019). Neural mmo: A massively multiagent game environment for training and evaluating intelligent agents. *arXiv preprint arXiv:1903.00784*. 7

Team, O. E. L., Stooke, A., Mahajan, A., Barros, C., Deck, C., Bauer, J., Sygnowski, J., Trebacz, M., Jaderberg, M., Mathieu, M., et al. (2021). Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*. 7

Turing, A. M. (1936). On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363):5. 4

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, pages 5998–6008. 3

Wang, K. and Skadron, K. (2015). Cellular automata on the micron automata processor. *University of Virginia Departement of Computer Science*. 4

Wolfram, S. (2002). *A New Kind of Science*. Wolfram Media Inc. 4

Yu, X., Fu, J., Deng, R., and Han, W. (2024). Mineland: Simulating large-scale multi-agent interactions with limited multimodal senses and physical needs. *arXiv preprint arXiv:2403.19267*. 7

Zaitsev, D. A. (2018). Simulating cellular automata by infinite petri nets. *J. Cell. Autom.*, 13(1-2):121–144. 4