CORALAI: EMERGENT ECOSYSTEMS OF NEURAL CELLULAR AUTOMATA

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Aidan Barbieux

March 2024

COMMITTEE MEMBERSHIP

TITLE: Coralai: Emergent Ecosystems of Neural Cellular Automata

AUTHOR: Aidan Barbieux

DATE SUBMITTED: March 2024

COMMITTEE CHAIR: Rodrigo Canaan, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Franz Kurfess, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Zoë Wood, Ph.D.
Professor of Computer Science

ABSTRACT

Coralai: Emergent Ecosystems of Neural Cellular Automata

Aidan Barbieux

Artificial intelligence has traditionally been approached through centralized architectures and optimization of specific metrics on large datasets. However, the frontiers of fields spanning cognitive science, biology, physics, and computer science suggest that intelligence is better understood as a multi-scale, decentralized, emergent phenomenon. As such, scaling up approaches that mirror the natural world may be one of the next big advances in AI. This thesis presents Coralai, a framework for efficiently simulating the emergence of diverse artificial life ecosystems integrated with modular physics. The key innovations of Coralai include: 1) Hosting diverse Neural Cellular Automata organisms in the same simulation that can interact and evolve; 2) Allowing user-defined physics and weather that organisms adapt to and can utilize to enact environmental changes; 3) Hardware-acceleration using Taichi, PyTorch, and HyperNEAT, enabling interactive evolution of ecosystems with 500k evolved parameters on a grid of 1m+ 16-channel physics-governed cells, all in real-time on a laptop. Initial experiments with Coralai demonstrate the emergence of diverse ecosystems of organisms that employ a variety of strategies to compete for resources in dynamic environments. Key observations include competing mobile and sessile organisms, organisms that exploit environmental niches like dense energy sources, and cyclic dynamics of greedy dominance out-competed by resilience.

ACKNOWLEDGMENTS

Thanks to:

- Eavy Barbieux, Sarah Barbieux, and Arnaud Barbieux, for hosting and supporting me all the way through
- Adrian Conrad and Erik Luu, for hundreds of hours discussing slime, brains, and emergence
- Jonathan Stoller, for being a roll model and prime actor in my finishing of this work
- Michael Ullman, for our serendipitous connection and your fruitful mentorship

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 The Success and Limits of Artificial Intelligence

Intelligence has been traditionally associated with problem solving, planning, and learning, usually in reference to some centralized mechanism of control. This view has dominated work in artificial intelligence and machine learning. This framing of intelligence, however, breaks down on the frontiers of cognitive neuroscience, biology, physics, computer science, and economics. Here, intelligence, or competency, is viewed as a decentralized, emergent phenomena appearing at many scales. To progress in creating more robust and adaptive artificial intelligence we must shift from centralized architectures to diverse, multi-scale models that support emergence and adaptation [18, 43, 15, 47, 41, 32, 33].

The success of much of modern AI is a result of gradient-based optimization of specific metrics using larger and larger datasets and models. When a sufficiently large and powerful model is trained on a high-quality, diverse dataset, it can demonstrate what appears to be emergent abilities and generalization. However, these abilities are largely dependent on the quality and diversity of the training data, which raises the question: where does this data come from?

Datasets can be thought of as a sort of residue left behind from a living process. Whether it is language, video, or statistics, data usually represents something generated from a complex system of many parts interacting over time to survive in a shifting environment. It may be time to shift our efforts away from merely representing data, which is akin to making molds of what already exists, and instead focus on

the powerhouse under the hood: the processes that give rise to intelligent behavior in living systems.

1.2 The Transition to Artificial Life

Artificial Life (ALife) is a field that focuses on the creation of living systems. A living system is one that survives and evolves, exploring new abilities in a self-directed manner [31]. ALife can be hard, soft, or wet, meaning physical, digital, or chemical. Physical and chemical ALife tends to be limited by our ability to manipulate the real world and interact with physics or chemistry, making exploration difficult and slow, but benefits from the innate fidelity and complexity of our world. Digital ALife, however, is potentially only limited only by algorithms and scale.

If recent work in AI has told us anything, it is that size matters. Larger datasets and models produce significantly better results. Our models have only changed in architecture, not fundamental form, and it is modern computing infrastructure and better datasets that have truly made our progress for us. So, perhaps it is time to scale up ALife.

Scale, however, is not just about the number of parameters. It is also about diversity and representational capacity. Much of digital ALife research has been limited by computational resources and the diversity of the simulations—most simulations converge to a simple behavior. To avoid this we need larger simulations of systems that are more free and diverse [57, 60].

So, what might a recipe for a digital ALife simulation be that produces adaptive and robust behavior that is meaningful to us as life embodied in this specific world?

- High-performance simulations that can take advantage of large-scale parallel computing infrastructure
- Models that have the capacity to quickly generate and represent a wide range of diverse phenomena
- Integration of dynamics that map onto the real world so that results are understandable and useful

1.3 Coralai

Coralai is a response to these requirements. The name Coralai is inspired by the scale and diversity of coral reef ecosystems. Coralai is a framework for quickly and efficiently simulating the emergence of diverse artificial life ecosystems that interact with physics and weather. Coralai uses Neural Cellular Automata (NCA) with proven representational ability, limiting the emergence of behavior primarily to the set up of a simulation, as opposed to a fundamental limitation in architecture. Further, the integration of physics enables organisms to take advantage of the behavior of the environment to generate adaptive complexity with simple architectures, further expanding the space of possible ecosystems and aligning it with those that may be physically realizable.

The work of this thesis is primarily in providing tooling powerful enough to approach the pertinent questions of ALife. Coralai builds upon previous work with NCA [36, 49, 13], which have shown impressive results in tasks such as morphogenesis, maze solving, and pattern recognition. However, Coralai goes beyond previous systems by allowing users to define their own physics and weather patterns, enabling the creation of rich, diverse ecosystems that more closely resemble natural environments. The current implementation of Coralai is limited to 2D environments and a specific

set of physical constraints and environmental dynamics. Future work may explore the extension of Coralai to 3D environments and the incorporation of more sophisticated physics and weather models.

Previous work on Coralai has been previously published under the name EINCASM (Emergent Intelligence in Neural Cellular Automata Slime Molds) [5] with an emphasis on physiologies and physics conducive to the simulation of Physarum Polycephalum. Coralai expands this to work with arbitrary physics and diverse physiologies. All of Coralai's code is freely available via [3]

Chapter 2

BACKGROUND

To create Artificial Life (ALife) simulations that can benefit from the scale of modern computing, we must develop a principled understanding of what constitutes life-like intelligent behavior, how to detect it, and what systems may be capable of producing it. In this background chapter, we survey a variety of conceptual frameworks and tools from the field of ALife that motivate the design of Coralai or are used in its development. However, due to the scale and diversity of this highly interdisciplinary field, only a limited perspective can be provided.

2.1 Intelligent Behavior

A primary difficulty of creating ALife is determining what subset of systems display properties that we care about. Relatively simple computation can produce endless complexity with patterns that mirror natural systems. For example, the dragon curve fractal can produce fern-like patterns from simple rules, or the Mandelbrot and Julia sets seem to contain a huge variety of geometric patterns that we also see in nature [34]. However, though such systems may be useful for understanding and building ALife, they do not seem to display life-like intelligent behavior. And, further, if they did, we would need further tooling to explore them and asses which parts of their unending complexity might be worth looking at.

In the following section we explore properties and frameworks that enable us to quantify and detect intelligent behavior. However, intelligence is a much disputed topic and therefore a singular description of it is presently inaccessible. In this section we

move away from human-centered notions of intelligence and instead look at notions applicable to diverse systems, ranging from molecules to civilizations. This makes it possible to analyze digital systems that would otherwise be completely alien.

A traditional view of intelligence might provide a game of chess or a Turing test as a measure of intelligence. However, machines are now competent at both of these tasks are are still unable to replace humans or ecosystems in running economies or surviving harsh environments [11]. Instead, human games of pieces, boards, and words seem to demonstrate abilities whose utility is highly contingent on the specific social and natural environments in which we evolved and are not, fundamentally, representative of a system's general competency.

Instead of looking at the specific and contingent competencies of humans, it is useful to look at the processes that brought about their relevance. Some of human competency in chess, for example, may emerge from a competency in foraging in spatial environments, which itself is useful for more fundamental reasons.

2.1.1 Homeostasis, Allostasis, and Homeorhesis

Intelligence can be broadly defined as the capability to maintain a form in a dynamic environment. Though abstract, this can even be applied to chess, where likelihood of winning is maintained in the face of an opponents moves. This ability is often framed in the context of homeostasis, allostasis, and homeorhesis. Homeostasis refers to the ability to sustain a stable internal state despite external changes. Allostasis expands this behavior to longer time-scales, where success is in predicting, preparing for, or preventing change. Homeorhesis expands this even further to consider the maintenance of developmental trajectories, mapping sequences of changes that must be made to achieve a new form [41].

Daisy World, a model developed by James Lovelock within the Gaia hypothesis, demonstrates how complexity can emerge from a relatively simple self-regulating behaviors. In this model, a planet's temperature, influenced by the sun, is regulated by the albedo effect, which in turn is determined by the ratio of white to black daisies on its surface. Each daisy has a rate of reproduction tied to local temperature, determined by it and its neighbors. As daisies die on each time step and solar energy fluctuates this system develops complex and adaptive dynamics [67].

Daisy World demonstrates that complex homeostatic mechanisms can be exhibited by a system with no top-down control. Further, Daisy World shows that these are a necessary result of relatively simple feedback loops between organism and environment rather than highly evolved adaptations.

2.1.2 Morphogenesis

Morphogenesis represents a broader example of intelligence, characterized by a system's ability to not only maintain itself but to self-organize and form complex structures in changing environments. This process encompasses both cellular transformation into organs and organisms collaborating to form ecosystems, highlighting critical aspects of intelligence such as multiscale intelligence, emergence, and criticality.

One of the earliest investigations into such behavior was by Alan Turing's exploration of reaction-diffusion patterns. Turing's work demonstrated how chemical substances, called morphogens, interact to form patterns during embryonic development. He proposed a mathematical model where two chemicals spread through a medium at different rates, leading to the emergence of stable patterns. This model demonstrates how complex biological shapes and patterns can arise from simple chemical processes. Turing's theory laid the groundwork for understanding morphogenesis, the process

by which organisms develop shape and form, highlighting the role of chemical mechanisms in biological pattern formation [63].

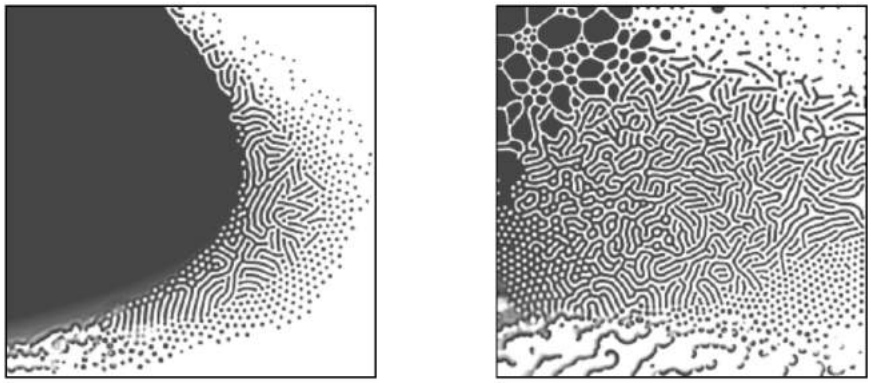

Figure 2.1: Phase spaces of a reaction diffusion system. Adapted from [54]

2.1.3 Criticality and Scale-Invariance

For complex patterns to emerge from a reaction-diffusion system it must avoid becoming either homogeneous or chaotic. This often occurs at a critical state of a system, which is the region of a state space on the cusp of a phase transition. Critical states are prevalent in many physical systems and often display another characteristic common to intelligent behavior: scale-invariance.

A pivotal example of scale-invariant criticality is the Ising model. Here, elementary units termed spins organize in response to their neighbors and the temperature of the system. At low temperatures, spins minimize their energy by aligning with their neighbors, resulting uniform order. Conversely, at high temperatures, spins are subject to random fluctuations, resulting in uniform chaos. However, at a critical temperature the system is transitioning between ordered and disordered states. At this phase transition spins have long range effects on other spins, sending information across different scales of the system. This results in scale-invariant regions of align-

ment, reflecting a universal distribution of sizes regardless of the observational scale [7].

Scale-invariance can be detected by analyzing the distribution of a system's properties across different scales. In the case of the Ising model, this could be the distribution of cluster sizes at the critical temperature. If the system is scale-invariant, the probability distribution of cluster sizes should follow a power law: $P(s) \propto s^{-\alpha}$, where $P(s)$ is the probability of observing a cluster of size s, and α is the scaling exponent. A power law distribution appears as a straight line on a log-log plot, with the slope of the line corresponding to the scaling exponent α. The presence of a power law distribution indicates that the system exhibits self-similarity across different scales, a hallmark of scale-invariant behavior [39]. Other methods for quantitatively measuring scale-invariance or multi-scale complexity are discussed below.

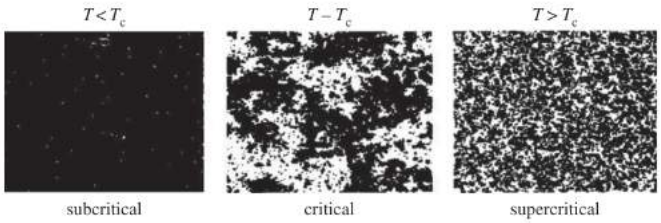

Figure 2.2: A two dimensional Ising model below, at, and above the critical temperature [26].

In the context of artificial life systems like Coralai, scale-invariance can be observed in the organization of cell types or the distribution of behavioral patterns across different scales (Figure 2.3). The presence of power law distributions in these properties would indicate that the system has self-organized to a critical state, enabling the emergence of complex, adaptive behaviors.

The coexistence of order and chaos at critical states enables the exploration and consolidation of new forms that persist across time. This state allows water molecules,

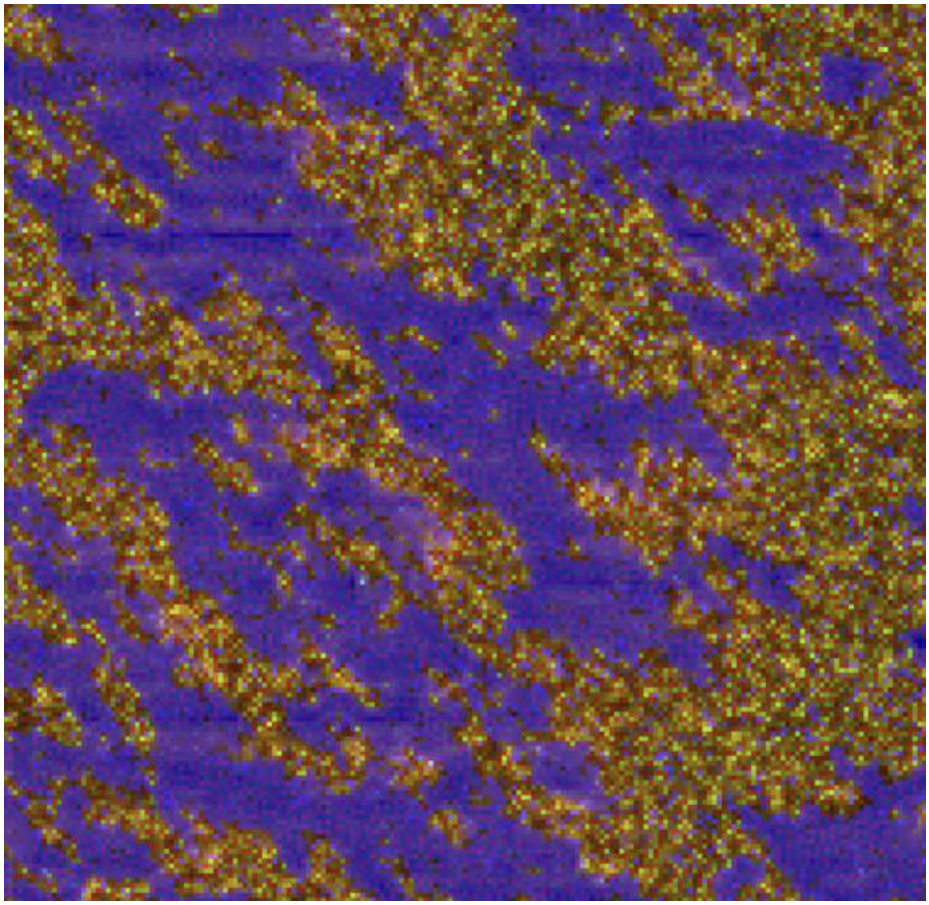

Figure 2.3: Scale-free organization of cell types in Coralai. Color represent combined energy and infrastructure levels and genome type

for example, to form symmetrical and repetitive patterns that develop into structures of increasing complexity — a snowflake. As such a system undergoes various stressors, existing forms either persist or dissolve. Those that persist have the opportunity to explore new states and configurations. These are the principal components of evolution.

2.1.4 Interaction Networks

Scale-invariance and criticality render systems more adaptable and robust, capable of transferring information across spaces to evolve in response to stressors. This applies to the maintenance of complex forms, which dissolve in chaos or conform in order,

through the process of evolution. This concept extends beyond physical systems and includes interaction networks, such as neural networks and social interactions [26].

In neural networks, criticality may be an optimal state for computational efficiency. The Critical Brain hypothesis suggests that the brain self-organizes to a critical state for effective information transfer. This criticality, while somewhat controversial, is proposed to facilitate communication across different scales within the system [20].

Interaction networks are also exemplified by linguistic evolution, as discussed in Hillis's "Intelligence as an Emergent Behavior; or, The Songs of Eden." Language evolves through the interactions of sounds, or "songs", with certain songs being propagated or lost as determined by the dynamics of the substrate through which they propagate: collections of agents. Just as in the other cases, propagation into the future is determined by adaptability and robustness to external influences such as lending information useful to the survival of the agents that propagate them or being memorable and unique. This process benefits from the ability of songs to travel across a wide range of scales of space, time, and contexts while maintaining distinct local niches, enabling diverse linguistic evolution [21].

2.1.5 Empowerment

Empowerment, as conceptualized in information theory, quantifies the control an agent exerts over the future states of its environment. This concept can be understood through the lens of mutual information, specifically, the mutual information between the current state of a system (or a component within the system) and future environmental states. Essentially, empowerment measures the capacity of a system's current state to predictably influence future environmental outcomes. Empowerment can even be used as a specific objective function when training artificial life simula-

tions, as demonstrated by Grasso and Bongard in training empowered neural cellular automata [16].

In the context of criticality and scale-invariance, empowerment refers to a system's ability to maintain or amplify its influence in competition or coordination with other influences in the environment. This aligns closely with the concepts of homeorhesis and allostasis, where an organism or system proactively interacts with its environment to preserve its integrity and trajectory amidst changing conditions. An empowered organism is one that is resilient enough to environmental changes to persist into the future and reproduce.

2.1.6 Emergence

Empowerment across scales enables small components to enact changes in larger components. Often, this appears as an emergent phenomenon. Emergence is observed when simple components interact to form complex wholes with properties not present in individual components, but composed of them. Often, emergence results in competency in a different, often larger and more abstract, space.

Emergence is evident in the various systems previously discussed: the coherent structures in Ising models; snowflakes; and literature out of sounds and songs. Similarly, in biological systems, a hierarchy of competencies spans from genetic networks to neural networks, tissues, organisms, and societies. Each level represents a leap in emergent behavior enabling a system to survive larger disruptions by utilizing a new embodiment. Organisms, for example, can lend motility to their cells and organs to navigate across large spaces while maintaining a local niche where the cells can flourish.

2.1.7 Multi-Scale Competency

Multi-scale competency architectures are a lens for viewing the interaction between these different scales. This active area of research suggests that cognitive abilities can be analyzed as competencies in navigating various spaces or state spaces. Michael Levin's work, for example, proposes analyzing cognition across different embodiments through the lens of competency in navigating arbitrary spaces [15]. This approach aligns with the concept of allostasis and homeorhesis, where a form must navigate its state space to predict and enact change in itself and the environment to continue to persist across time.

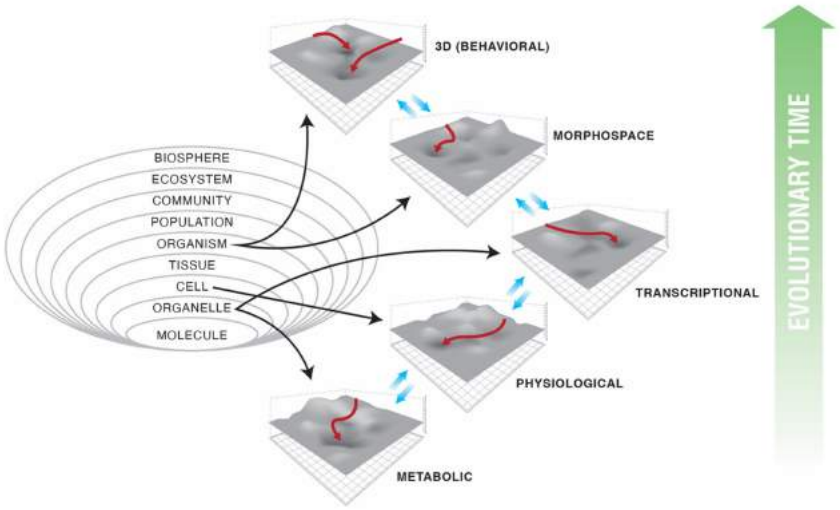

Figure 2.4: The navigation of different abstract spaces across different scales of a multi-scale competency architecture [15]

Characterizing intelligence as multi-level competency in navigating diverse spaces underscores the nested, emergent nature of intelligent systems. For a system to exhibit general intelligence and be adaptable and robust to environmental changes without being constrained by a specific objective function, it must embody multi-scale, self-organizing, and emergent intelligence. This framework highlights the importance of

understanding intelligence not just as a feature of individual entities but as a complex interplay of processes operating across multiple scales.

2.2 Open-Endedness

Open-endedness is a critical aspect of artificial life systems that aim to simulate the open-ended evolution and emergence of complex behaviors observed in natural systems. Open-ended systems are characterized by their ability to continually generate novel and increasingly complex adaptations, without being constrained by predefined objectives or external interventions [57].

2.2.1 Self-Organizing Intelligent Matter

One promising approach to achieving open-endedness in artificial life is the concept of self-organizing intelligent matter, as proposed by [17]. In this framework, the environment is composed of atomic elements that contain neural operations and interact through exchanges of information and physics-like rules. The key idea is that intelligent organisms can emerge from the interactions of these elements, without any explicit notion of an agent or objective function.

2.2.2 Evoloops

Other approaches to achieving open-endedness in artificial life include the evolution of self-reproducing structures, such as evoloops [52, 42] and autopoietic systems [35]. These systems demonstrate the emergence of complex behaviors and adaptations from simple, locally interacting elements, but are often limited in their ability to scale to more complex and diverse forms of life.

Finally, the Paired Open-Ended Trailblazer (POET) algorithm [66] represents another promising approach to open-endedness, by co-evolving environmental challenges and the agents that solve them. This dynamic generation of novel environments and selective pressures can drive the continuous emergence of increasingly complex and adaptive behaviors, in a way that is not limited by predefined objectives or constraints.

2.3 Cellular Automata

This section introduces cellular automata (CA) and neural cellular automata (NCA), which form the computational substrate of Coralai. CAs provide a simple yet powerful framework for simulating complex systems and emergent phenomena, as exemplified by the Game of Life and various physical simulations. NCAs extend this framework by incorporating neural networks as the update rules for the cells, enabling the evolution and learning of more sophisticated behaviors. Coralai builds upon recent advances in NCA research, such as the use of convolutional neural architectures and evolutionary techniques, to create a flexible and efficient platform for simulating artificial life.

2.3.1 Conway's Game of Life

The Game of Life, developed by John Conway, is a classic example of a cellular automaton that exhibits complex behaviors from simple rules [2]. In the Game of Life, cells on a grid evolve based on their local neighborhood, with rules for birth, survival, and death. Despite its simplicity, the Game of Life can give rise to intricate patterns and structures, such as gliders, oscillators, and spaceships. The Game of Life has been widely studied as a model of self-organization and emergent behavior, and has been shown to exhibit self-organized criticality, where the system naturally evolves to a critical state characterized by scale-invariant dynamics [2].

2.3.2 Physical Simulations with Cellular Automata

Cellular Automata (CA) can be used to simulate various physical phenomena, such as thermodynamics and hydrodynamics [51]. By defining local rules that capture the essential features of the physical system, CA models can reproduce complex macroscopic behaviors, such as phase transitions, fluid flow, and pattern formation. The discrete nature of CA makes them computationally efficient for large-scale simulations, while still capturing the key aspects of the underlying physical processes. CA models have been successfully applied to simulate a wide range of physical systems, from fluid dynamics to material science, demonstrating their versatility and effectiveness as a simulation tool.

2.3.3 Continuous Cellular Automata

Lenia is a continuous CA where the update rule is defined by a differentiable kernel function [8]. In Lenia, the state of each cell is a continuous value, and the update rule is applied using convolution with the kernel function. This allows for smooth transitions between states and the emergence of complex morphologies that can be studied like biological systems. Bert Chan and colleagues expanded the original Lenia framework to include a wider range of kernel functions, leading to the discovery of new morphologies and the development of a taxonomy of Lenia patterns [9]. Flow Lenia introduced mass conservation to Lenia, which reduces the set of possible morphologies and makes it easier to find interesting species [45]. Mass conservation also enables the simulation of multiple interacting Lenia species in the same environment, leading to a more evolutionary simulation. However, the evolutionary dynamics in Flow-Lenia are still limited compared to more powerful artificial life systems.

2.3.4 Neural Cellular Automata

Neural Cellular Automata (NCA) are a class of models that use neural networks as the update rule for a CA. This can be efficiently implemented using convolution operations, leveraging existing frameworks for training using gradient-based approaches. NCA can be trained to start form a single pixel and grow into a target image by applying a pixel loss function to train the network. These NCA produce images that are resilient to ablations (the disruption of a neighborhood of pixel values) and can dynamically react to modification [37].

By training NCA on images of solved mazes, Endo et. al [13] were able to produce NCA that could dynamically react to a changing maze and reliably find the shortest path, even when presented with unseen configurations. This ability of NCA to generalize was used by Randazzo et. al [49] to classify MNIST digits by learning to grow into a specific color based on the digit class, illustrating their potential for pattern recognition tasks.

The forms grown by NCA can also be trained to respond to specific cues, such as changing the tail direction within an image of a gecko upon a presented signal [62] or climbing gradients injected into the environment [28]. NCA, like other convolutional architectures, can be organized hierarchically to grow into different patterns at different scales, enabling the composition of smaller parts into larger patterns, relating to multi-scale competency [43].

NCAs have proven to be a powerful architecture with strong representational capabilities, capable of emulating morphogenesis and learning complex behaviors. However, the studies mentioned above are limited to specific objectives, often based on pixel-level loss functions, and do not approach open-ended evolution or exploration of a wide range of behaviors.

2.3.5 Evolved Neural Cellular Automata

Evolutionary optimization has been applied to NCAs to evolve models that exhibit desired behaviors or patterns. Nichele et. al [40] used compositional pattern producing networks (CPPNs) to control the update rule of cellular automata and evolved these CPPNs to generate target images.

Biomaker CA provides a broader evolutionary optimization of NCA by growing organisms that collect nutrients and solar energy to produce fruits and reproduce. The fruiting process enables the mutation of an organism and provides in-environment evolution. Meta-evolution of parameters and petri-dish simulations were also used to produce better organisms without a specific objective function [48]. However, this system is considerably hand-programmed and constrained in its physics and physiology, preventing it from producing open-ended complexification.

2.4 Neuroevolution

This section focuses on the developmental neural networks used in Coralai, specifically Compositional Pattern Producing Networks (CPPNs), NeuroEvolution of Augmenting Topologies (NEAT), and Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT). CPPNs serve as the main mechanism for generating the complex patterns and structures in Coralai, while NEAT and HyperNEAT are used to evolve these CPPNs and the larger neural networks they encode. By leveraging these powerful developmental techniques, Coralai is able to generate diverse and adaptable organisms that can exploit the regularities and symmetries present in the environment.

2.4.1 Compositional Pattern Producing Networks

Compositional Pattern Producing Networks (CPPNs) are a class of neural networks designed to generate complex patterns and structures. CPPNs were introduced by Stanley as an abstraction of the process of natural development, where genetic information is transformed into phenotypic patterns through a series of developmental steps [56].

The key idea behind CPPNs is to use a neural network as a function that maps spatial coordinates to output patterns. The network takes as input the coordinates of a point in a multi-dimensional space (e.g., 2D or 3D) and produces an output value for that point. By querying the network at different coordinates, a complete pattern or structure can be generated.

CPPNs are typically composed of a set of activation functions that are commonly found in natural systems, such as Gaussian, sigmoid, and periodic functions. These activation functions are chosen to mimic the types of patterns and regularities observed in biological development, such as symmetry, repetition, and variation [59].

One of the main advantages of CPPNs is their ability to generate complex patterns with a relatively small number of network parameters. This is achieved through the composition of activation functions, which can create a wide range of patterns by combining and transforming simpler patterns. Additionally, CPPNs can be evolved using techniques such as neuroevolution, allowing for the discovery of novel and interesting patterns through evolutionary search [53].

CPPNs have been successfully applied to a variety of domains, including the generation of 2D images, 3D structures, and robot morphologies. In the context of artificial life and emergent behavior, CPPNs have been used to evolve the morphology and con-

trol systems of virtual creatures, leading to the emergence of complex and adaptive behaviors [10].

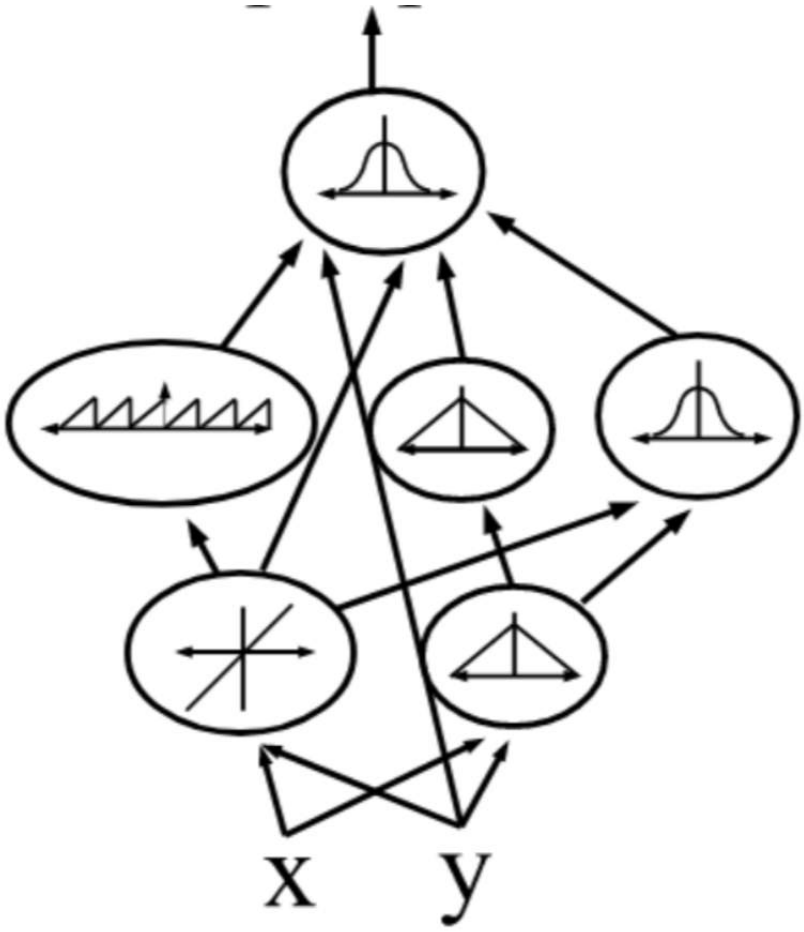

Figure 2.5: Illustration of a Compositional Pattern Producing Network (CPPN) where each node contains an activation function. Adapted from [40, 56, 53]

Recent work has also explored the integration of CPPNs with other developmental models, such as cellular automata, to generate more complex and biologically plausible structures [40]. Additionally, the principles of CPPNs have been extended to other domains, such as the evolution of neural network architectures, where they can be used to generate connectivity patterns and modularity in large-scale networks [59, 50].

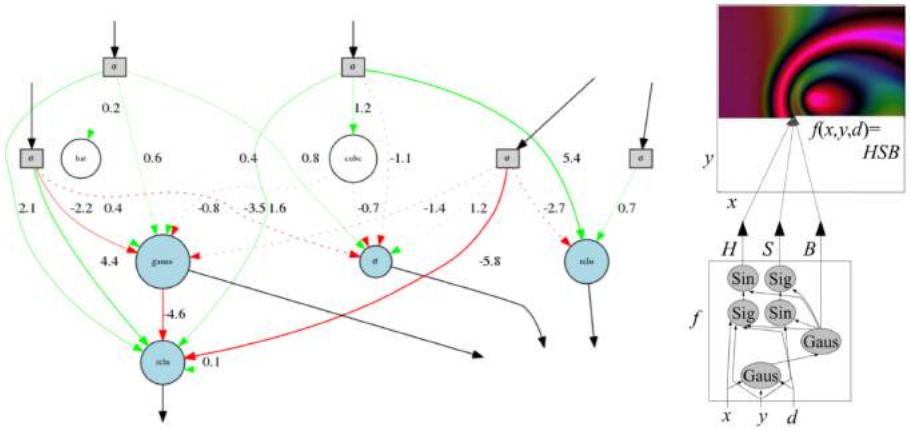

Figure 2.6: Illustration of a CPPN generating a 2D image pattern. The network takes as input the coordinates of each pixel and outputs the corresponding pixel value. Adapted from [56, 53]

2.4.2 NEAT

NeuroEvolution of Augmenting Topologies (NEAT) is an evolutionary algorithm designed to evolve artificial neural networks (ANNs) with increasingly complex topologies [61, 55]. NEAT addresses key challenges in evolving ANNs, such as the competing conventions problem and the preservation of structural innovation.

One of the main features of NEAT is its ability to start with a minimal network topology and gradually add complexity over generations through mutations. This is achieved by introducing two types of structural mutations: adding a new node and adding a new connection between existing nodes. By starting with simple networks and incrementally adding complexity, NEAT can efficiently explore the space of possible network topologies.

To protect structural innovation, NEAT employs a mechanism called speciation, which groups genomes into species based on their compatibility. Genomes are considered compatible if they have similar topologies, as measured by a compatibility distance metric. Speciation allows novel topological innovations to survive and com-

pete primarily within their own niches, rather than being immediately outcompeted by more optimized networks.

NEAT also introduces historical markings to address the competing conventions problem, which arises when multiple genomes represent the same solution but with different encodings. Historical markings uniquely identify each gene in the genome, allowing NEAT to align and crossover genomes with differing topologies effectively.

2.4.3 HyperNEAT

HyperNEAT (Hypercube-based NeuroEvolution of Augmenting Topologies) is an extension of the NEAT (NeuroEvolution of Augmenting Topologies) algorithm that uses Compositional Pattern Producing Networks (CPPNs) to evolve large-scale neural networks with complex connectivity patterns [59].

The main idea behind HyperNEAT is to use a CPPN to generate the connectivity patterns of a larger neural network, rather than evolving the connections directly. The CPPN takes as input the spatial coordinates of two neurons in the substrate network and outputs the weight of the connection between them. By querying the CPPN for all pairs of neurons in the substrate, the complete connectivity pattern of the network can be generated.

HyperNEAT exploits the geometric regularities and symmetries that are often present in the problem domain to generate neural networks with appropriate connectivity patterns. For example, in a vision-based task, the CPPN can be queried with the coordinates of neurons in a 2D grid, allowing it to generate convolutional-like connectivity patterns that are well-suited for image processing.

One of the main benefits of HyperNEAT is its ability to evolve neural networks with millions of connections using relatively small CPPNs. This allows for the evolution of large-scale networks that can tackle complex problems in domains such as computer vision, robotics, and artificial life [29, 50].

In the context of artificial life and emergent behavior, HyperNEAT has been used to evolve the morphology and control systems of virtual creatures, leading to the emergence of complex and adaptive behaviors. By evolving both the physical structure and the neural connectivity patterns of the creatures, HyperNEAT can generate organisms that are well-adapted to their environment and can exhibit lifelike behaviors [10].

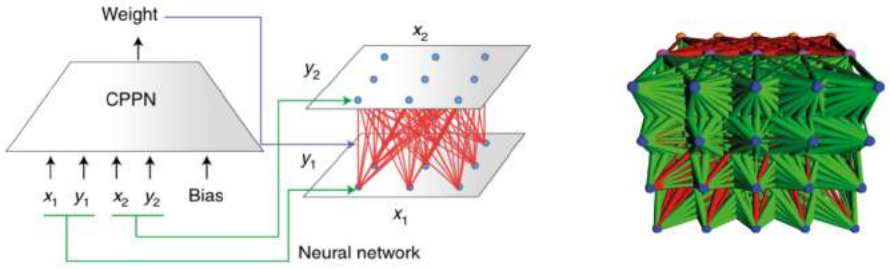

Figure 2.7: Illustration of a compositional pattern producing network generating the weights of a densely connected network encoded on a hypercube. Adapted from [58]

Chapter 3

IMPLEMENTATION

3.1 Languages and Tools

Coralai is implemented using a combination of Python 3.11 [64], Taichi Language 1.6, PyTorch 2.2, and PyTorch NEAT. These tools were chosen for their efficiency, flexibility, and compatibility with parallel computing on GPUs. The code for Coralai and a modified PyTorch NEAT can be found at [3, 4]

3.1.1 Taichi Language

Taichi is a language embedded in Python that enables high-speed parallel computing [22, 24, 23]. It excels at efficient and complex physical simulation with simple code. Kernels can be run directly on PyTorch tensors stored on the GPU, enabling easy interfacing with Coralai's neural networks.

Taichi's memory management capabilities are leveraged in the implementation of the substrate of the simulation. The substrate is represented as a Taichi field containing nested channels, enabling simple indexing into complex subsections of memory while on the GPU.

Taichi also facilitates the efficient application of diverse neural networks to different areas of the substrate according to a map of genome keys. This allows hundreds of different organisms to sense and act on the environment in parallel.

Real-time visualization is another key feature of Taichi utilized in Coralai. The visualization allows for interaction with the simulation, such as getting cell statistics, painting, mutating specific areas, resetting the simulation, and selecting channels to view.

Taichi's kernel system enables the implementation of easy-to-read and efficient physics simulations. Physics can be dynamically applied based on cell types, weather parameters, and world state.

Coralai takes advantage of Taichi's deployability on various platforms, including mobile and web, and its compatibility with different GPU computing frameworks. In this experiment, Taichi's compatibility with Apple's Metal Performance Shaders was used to run the simulation on an M1 Metal chip.

3.1.2 PyTorch

PyTorch [44] is used in Coralai to store the weights of the organisms' neural networks. While not strictly necessary due to the weights being applied with Taichi and generated with CPPNs, PyTorch provides a convenient interface for integration with PyTorch NEAT.

PyTorch is also utilized for various tensor operations, such as applying activation functions to the entire environment, clamping values, obtaining tensor statistics, and performing normalization.

3.1.3 NEAT Python

NEAT (NeuroEvolution of Augmenting Topologies) [61, 58] is used for the neuroevolution of CPPNs in Coralai. However, most of NEAT's functionality is not directly

utilized. It serves primarily as a shell to generate, mutate, and merge CPPNs based on a configuration file. Experiments with a custom implementation of NEAT in Taichi were conducted, but the original NEAT implementation excels at simply creating diverse CPPNs with heterogeneous architectures.

One limitation of using NEAT Python is that mutating, merging, and generating genomes must be done on the CPU, which introduces a performance bottleneck in the simulation. To mitigate this, the rate of these operations is limited to set intervals, analogized as "radiation events." However, this is not ideal, as merging genomes should ideally happen on the fly as cells interact. In the future, the plan is to operate entirely in Taichi to enable dynamic merging of genomes. Currently, sexual reproduction (merging) and mutation are only performed at set intervals.

3.1.4 PyTorch NEAT

PyTorch NEAT [30] is used to apply the CPPNs generated by NEAT to the weights of a larger, fully connected neural network contained in PyTorch, following the HyperNEAT algorithm. This speeds up the computation of weights and enables easy transfer of the weights to a Taichi kernel, as they are generated on the GPU.

Coralai utilizes a custom implementation of HyperNEAT, built upon the CPPNs provided by PyTorch NEAT. This custom implementation generates biases alongside weights, takes a 3D input and output coordinate space, and creates weights compatible with the batch processing of the Taichi kernels. The code for this custom implementation can be found at [4].

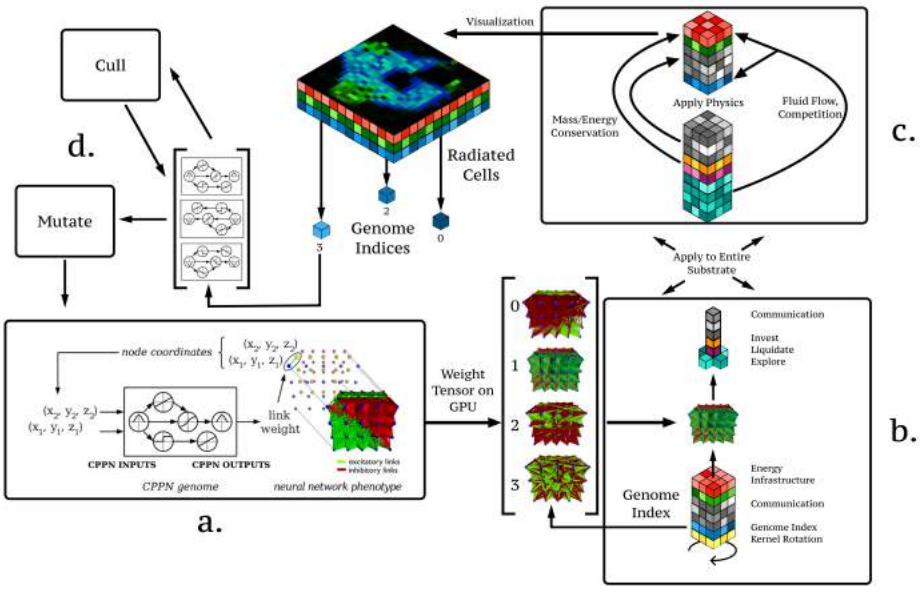

Figure 3.1: (a.) Generation of dense network from the NEAT genome. Adapted from [19] (b.) Application of dense network to a single cell. (c.) Application of physics. (d.) Mutation an culling of radiated cells.

3.2 Substrate

3.2.1 Taichi and PyTorch Tensors

The substrate of Coralai is represented as a single large tensor that can be stored as either a Taichi field or a PyTorch tensor. The memory for the substrate always resides on the GPU, ensuring efficient computation and minimizing data transfer overhead.

3.2.2 Channels and Indexing

The substrate of Coralai is organized into channels, which can be nested to form a hierarchical structure. A channel can contain multiple subchannels, enabling the representation of complex data structures within each cell of the substrate. This nested

channel structure is easily implemented using Taichi's flexible datatype definitions, where each set of channels can be defined as a Taichi struct with named subchannels.

The channel hierarchy is then converted into an index tree, which allows for simple queries to return the set of indices into the tensor representing those channels. This indexing system makes the programms easier to write, more readable, and more reliable, while also being fast due to its execution on the GPU.

The channels and indexing of a substrate can be easily saved and loaded for checkpointing experiments and ensuring reproducibility. This allows for the preservation of the entire state of the simulation, including the configuration of the substrate and the state of the organisms within it.

It is important to note that the substrate contains all the memory for the simulation, except for the weights of the organisms. This includes the actuators of the organisms, and even parameters for physics determining the local "weather" or difficulty of the environment. By consolidating all this data into a single substrate tensor, memory management is simplified and the efficiency of the simulation is improved.

3.3 Organisms

3.3.1 Organism Definition and Representation

Organisms in Coralai are defined by their sensors, actuators, and the kernel that determines their local neighborhood. The sensors and actuators are specified as a set of channels in the substrate, allowing the organisms to perceive and interact with their environment. The kernel is represented as an array of offsets from the central cell, defining the local neighborhood that each organism can sense and influence.

The actuators of the organisms are stored directly in the substrate, alongside all other data. This consolidation of memory enables the efficient sensing of previous actuator outputs by the organisms, as they can simply access the relevant channels in their local neighborhood.

The genomes of the organisms are represented using NEAT (NeuroEvolution of Augmenting Topologies) [61, 58]. Each organism's genome is used by NEAT, in conjunction with a configuration file, to generate a Compositional Pattern Producing Network (CPPN) [56]. The CPPN, in turn, is used to generate the weights of the organism's neural network. This process is highly flexible and can be customized using the parameters of NEAT.

The NEAT configuration file allows users to control various aspects of the evolution process, such as mutation rates for adding and removing nodes and connections, modifying connection weights, adding biases, and specifying aggregation and activation functions for the CPPN. This flexibility enables the exploration of different evolutionary strategies and the adaptation of the organisms to their environment.

One potential future direction is the dynamic modification of these parameters according to the "weather" of the system. For example, different regions of the substrate could have different mutation rates or availabilities of activation functions, creating niches where different organism morphologies can flourish independently from the rest of the population. This would allow for a more diverse and adaptive ecosystem, as organisms with different evolutionary strategies could coexist and compete in the same environment.

3.3.2 HyperNEAT Neural Architecture

The neural architectures of the organisms in Coralai are represented as dense, single-layer feedforward neural networks. Each network is essentially a tensor of weights and biases, with the weights generated by applying the HyperNEAT algorithm [59] to the CPPN encoded by the organism's genome.

HyperNEAT is used to map the input and output spaces of the organism's neural network to a higher-dimensional coordinate space, allowing the CPPN to generate weights that reflect the geometric relationships between the input and output neurons. This is achieved by defining a set of input and output coordinates for each neuron, based on their positions in the substrate and the topology of the organism's sensors and actuators.

The input to the organism's neural network is a flattened 1D tensor, with a length equal to the number of sensed channels multiplied by the number of neighboring cells in the organism's kernel. For example, with a Von Neumann neighborhood kernel (5 cells, including the central cell) and 5 sensed channels, the input tensor would have a length of 25.

Similarly, the output of the network is a 1D tensor with a length equal to the number of actuators. The actuator values are then reshaped and applied to the corresponding channels in the substrate.

Despite the flattening of the input and output tensors, the HyperNEAT algorithm allows the organisms to learn and exploit the geometric relationships between their sensors and actuators. By querying the CPPN with the coordinates of each input-output pair, HyperNEAT generates a weight matrix that reflects the spatial structure of the organism's neural architecture. This allows for the evolution of complex and

modular neural networks, with weights that are adapted to the specific sensory and motor layout of each organism.

One of the key advantages of this approach is that it allows for the evolution of neural architectures that are well-suited to the specific challenges faced by the organisms in their environment. By evolving the topology and geometry of the neural networks, along with their weights, HyperNEAT can discover solutions that exploit the spatial and temporal structure of the task at hand. This is particularly important in the context of Coralai, where organisms must learn to navigate and survive in a complex and dynamic world.

3.3.3 Diverse Weight Application

A crucial feature of Coralai is its ability to simulate a diverse population of organisms within the same environment. Each organism can have its own unique neural architecture, defined by its genome and CPPN. This allows for the coexistence and interaction of organisms with different cognitive capabilities, strategies, and behaviors.

To enable this diversity, Coralai maintains a map of genome keys for each cell in the substrate. These keys correspond to indices in a tensor that stores the weights of every organism's neural network. During the simulation, a Taichi kernel uses this map to dynamically apply the appropriate weights and biases to each cell, based on the genome currently occupying it.

This dynamic weight application mechanism can be thought of as a single large neural network, with a spatial encoding that activates different subnetworks based on the state of a recurrent internal memory structure—the substrate itself. The dynamics of this memory structure are determined by the performance of the large network on a generalized task, giving rise to a form of emergent, self-organizing computation.

The ability to simulate diverse populations of organisms is a key advantage of Coralai over other neural cellular automata frameworks. To the best of our knowledge, the only other implementation with similar capabilities is BioMakerCA [48] , which also supports the coexistence of multiple "species" within the same grid. However, BiomakerCA is constrained to a specific set of physiologies and interactions primarily centered around a single ecosystem model—specifically, one that mimics the growth and behavior of plants. These plants grow from soil towards light, utilizing nutrients and energy to grow and produce fruits, which then enable reproduction. This focus on a singular type of organism and its interactions within a specified environment limits BioMakerCA's versatility and representational capacity compared to Coralai. Coralai, on the other hand, is designed with a broader scope, allowing for the simulation of a wide range of organisms and their complex interactions within diverse ecosystems. This makes Coralai a more flexible and powerful tool for exploring the dynamics of life-like systems across various scales and contexts.

The diversity of organisms in Coralai allows for the study of complex ecological interactions, such as competition, cooperation, and symbiosis. By observing the dynamics of these interactions over time, researchers can gain insights into the emergence of higher-level behaviors and the evolution of complex systems. This is particularly relevant to the study of artificial life, where the goal is to understand the fundamental principles that give rise to life-like phenomena in computational substrates.

In summary, the diverse weight application mechanism is a key feature of Coralai that enables the simulation of complex, evolving ecosystems. By allowing each cell in the substrate to be occupied by a different organism, with its own unique neural architecture, Coralai opens up new possibilities for the study of emergent behavior, self-organization, and the evolution of complexity.

3.4 Physics

3.4.1 Actuator Application

The physics module in Coralai plays a crucial role in transforming the abstract outputs of the organisms' neural networks into concrete changes in the environment. This is achieved through a series of functions that map the actuator values stored in the substrate onto the relevant channels, modifying their values according to user-defined rules and constraints.

One of the main challenges in designing the physics module is to ensure that the actuator outputs are compatible with the intended physical dynamics of the system. This often requires the use of output activation functions, which scale and shift the raw actuator values to match the expected range and distribution of the target channels.

The specific activation functions and mapping rules used in the physics module are highly dependent on the nature of the simulation and the desired physical dynamics. In some cases, it may be sufficient to use simple linear transformations, while in others, more complex nonlinear functions may be required to capture the relevant physical phenomena.

Once the actuator values have been appropriately scaled and transformed, the physics module applies them to the substrate using a set of predefined functions. These functions can be thought of as the "laws of physics" that govern the behavior of the simulated environment, specifying how different quantities (such as energy, mass, or velocity) evolve over time based on the actions of the organisms and the state of the system.

The specific form of these functions is highly flexible and can be customized by the user to model a wide range of physical phenomena. Some common choices include:

- Conservation of mass [45]: ensuring that the total amount of mass in the system remains constant, even as it is moved around by the organisms.
- Energy gradients [28]: allowing organisms to sense and respond to variations in the distribution of energy across the substrate.
- Hydrodynamics and thermodynamics [51]: modeling the flow of fluids and the transfer of heat within the environment.
- Granular dynamics [48]: simulating the behavior of particulate materials, such as sand or soil.

By defining these functions in a modular and composable way, Coralai allows users to build complex physical models from simple building blocks, while still maintaining the efficiency and parallelism afforded by the Taichi language.

One of the key advantages of Coralai's physics module is that it allows for the offloading of many computational tasks from the organisms themselves onto the environment. For example, instead of each organism having to learn how to navigate energy gradients or fluid flows from scratch, these behaviors can be built into the physics of the system, allowing the organisms to focus on higher-level tasks such as foraging, reproduction, or communication.

This separation of concerns between the organisms and the environment is reminiscent of the concept of Markov blankets in computational neuroscience [47]. A Markov blanket defines the boundary between an agent (such as an organism) and its environment, separating the internal states of the agent from the external states of the

world. By encapsulating the physical dynamics of the environment within the physics module, Coralai provides a natural way to implement this separation, allowing the organisms to interact with the world through a well-defined interface (the actuators and sensors) while still being subject to the constraints and affordances of the simulated physics.

In summary, the actuator application mechanism is a crucial component of Coralai's physics module, enabling the translation of abstract neural outputs into concrete physical actions. By providing a flexible and modular framework for defining the laws of physics that govern the simulated environment, Coralai allows users to explore a wide range of emergent behaviors and evolutionary dynamics, while still maintaining the efficiency and scalability of the underlying computational substrate.

3.4.2 Weather

In addition to the physical dynamics governed by the actuator application mechanism, Coralai also includes a weather system that allows for the simulation of external environmental factors. These factors can vary over space and time, creating a dynamic and heterogeneous landscape that the organisms must adapt to in order to survive.

The weather system in Coralai is implemented using a set of functions that modify the values of specific channels in the substrate, according to predefined rules and patterns. These modifications can represent a wide range of environmental phenomena, such as changes in temperature, humidity, nutrient availability, or predation risk.

One of the key benefits of the weather system is that it allows for the creation of localized niches within the larger environment. By applying different weather patterns to different regions of the substrate, Coralai can simulate the presence of distinct

habitats or microenvironments, each with its own unique set of challenges and opportunities for the organisms.

For example, a simple weather function might involve adding and removing energy from the environment in a periodic fashion, mimicking the diurnal cycle of solar radiation. Organisms in regions with higher energy influx might be able to sustain faster growth and reproduction, while those in energy-poor regions might need to develop more efficient foraging strategies or enter a dormant state to conserve resources.

More complex weather patterns could involve modulating the mutation rates or energetic costs of different behaviors across the substrate. For example, organisms in certain regions might experience higher mutation rates, allowing them to explore a larger space of possible adaptations but also exposing them to the risk of deleterious mutations. Conversely, organisms in other regions might face higher costs for certain actions (such as movement or reproduction), forcing them to adopt more conservative strategies to maintain their energy balance.

The use of spatially heterogeneous weather patterns in Coralai draws inspiration from techniques such as Map Elites [38], which seek to maintain diversity in evolving populations by explicitly preserving individuals that excel in different niches. By creating a range of distinct environmental conditions within the same simulation, Coralai allows for the emergence of a wide variety of organism morphologies and behaviors, each adapted to a specific set of local challenges.

In addition to its role in creating spatial niches, the weather system in Coralai can also be used to simulate dynamic, temporal changes in environmental conditions. For example, the intensity or frequency of energy influx might vary over time according to a predefined schedule or a stochastic process, representing seasonal changes or unpredictable fluctuations in resource availability. Organisms would need to evolve

strategies for coping with these temporal variations, such as storing excess energy during periods of abundance to survive periods of scarcity.

The ability to simulate both spatial and temporal heterogeneity in environmental conditions is a key strength of Coralai's weather system. By exposing organisms to a range of challenges and opportunities that vary over space and time, Coralai can promote the evolution of robust, adaptive behaviors that are capable of generalizing to novel situations. This is particularly important for the study of open-ended evolution, where the goal is to create autonomous agents that can continuously adapt and innovate in response to changing circumstances.

3.5 Evolution

3.5.1 Population

In Coralai, the population of evolving organisms is represented as a tensor of neural network weights, with each index corresponding to a different genome. The NEAT genome objects themselves are stored in a separate array, in the same order as the weight tensor, allowing for easy mapping between genomes and their corresponding neural networks.

Alongside the genome array, Coralai maintains an array of ages for each genome, tracking how long each organism type has existed in the simulation. This information can be used to study the evolutionary dynamics of the population, such as the rate of turnover of different lineages or the average lifespan of successful organisms.

The use of a tensor-based representation for the population allows for efficient computation on GPUs, as the weights can be easily manipulated and updated using parallelized operations.

As the number of organisms in an environment grows, the storage of their networks can present a memory problem. To work around this limitation, Coralai implements a culling mechanism that periodically removes networks from the population to make room for new ones. However, the genomes that generate the networks can still be saved and later introduced into the environment by simply generating the weights of the network again.

The culling mechanism is controlled by several user-defined parameters, including the maximum population size, the culling interval (how often culling occurs), and the metric used to select which genomes to remove. The default metric simply removes genomes based on the number of cells they occupy in the substrate, with genomes occupying fewer cells being removed first. This ensures that genomes that are not able to reproduce and spread in the environment are gradually eliminated, making room for more successful organisms.

However, this default culling metric introduces a bias towards genomes that are able to quickly proliferate and occupy a large number of cells. In some cases, this may lead to a kind of "quantity over quality" selection pressure, where organisms that prioritize rapid growth and expansion are favored over those that have more sophisticated strategies for survival and adaptation.

To mitigate this bias, users of Coralai can define their own custom culling metrics that take into account other factors beyond just cell count. For example, a metric might consider the age of the genome, its overall fitness (based on some user-defined criteria), or its role in maintaining the diversity and stability of the ecosystem as a whole. By carefully designing the culling metric, users can steer the evolutionary process towards desired outcomes and ensure that the population remains healthy and adaptable over time.

3.5.2 Mutation

Mutation is a crucial component of the evolutionary process in Coralai, as it introduces new variation into the population and allows organisms to explore new regions of the fitness landscape. Like other aspects of the simulation, mutation in Coralai is designed to be flexible and customizable, allowing users to define their own mutation operators and rates.

The default mutation mechanism in Coralai is modeled after a process of random "radiation," where mutations are applied to the genome at fixed intervals and with a fixed probability. When a mutation event occurs, a number of random points in the substrate are selected, and if a genome is present at one of those points, it undergoes mutation.

The mutation operator itself is defined using the NEAT algorithm, which specifies a set of rules for adding, removing, or modifying nodes and connections in the genome. The specifics of these rules (e.g., the probability of adding a new node versus modifying an existing one) can be adjusted by the user to control the pace and scope of mutation.

Once a genome has been mutated, the resulting offspring is placed back into the substrate at the same location as the parent. Optionally, the mutation can also be accompanied by a "radiation radius," where all other genomes within a certain distance of the mutated one are removed, and the offspring is allowed to spread into the vacated space. This can help the mutated genome establish a foothold in the environment and increases its chances of survival.

If a mutation event occurs at a point in the substrate where no genome is present, Coralai can optionally insert a random genome from the population into that location.

This can help to maintain diversity in the population and prevent it from getting stuck in suboptimal regions of the fitness landscape. However, this feature is often disabled by default, as it can also lead to the rapid spread of successful genomes across the entire environment, which may not be desirable in all cases.

3.6 Simulation

3.6.1 Checkpointing

In Coralai, checkpointing is implemented by periodically saving the entire state of the simulation to disk, including the contents of the substrate, the configuration of the NEAT algorithm, and the current population of genomes. These checkpoints are saved in a structured directory hierarchy that includes metadata about the simulation, such as the current time step, the random seed used for initialization, and any user-defined parameters or settings.

The substrate itself is saved as a binary file containing the raw tensor data, along with a separate JSON file that describes the structure and layout of the tensor, including the number and arrangement of channels, the spatial dimensions of the grid, and any other relevant metadata. This allows the substrate to be easily loaded and restored on any machine with a compatible version of PyTorch, without the need for any additional preprocessing or conversion.

The NEAT configuration file is also saved as part of each checkpoint, allowing the evolutionary algorithm to be resumed from the same point and with the same settings as when the checkpoint was created. This is important for ensuring the reproducibility and consistency of the evolutionary process, especially when running long simulations or comparing results across different runs.

Finally, the population of genomes is saved using Python's built-in "pickle" serialization format, which allows complex Python objects (such as NEAT genomes) to be efficiently stored and restored from disk. Importantly, the genomes are saved separately from their corresponding phenotypes (i.e., the actual neural network weights), which are re-generated on-the-fly from the genome specifications when the checkpoint is loaded. This helps to reduce the size of the checkpoint files and improve the efficiency of the saving and loading process.

3.6.2 Visualization

Visualization is an essential component of Coralai, as it allows users to observe and interact with the simulated ecosystem in real-time. Coralai leverages the powerful visualization capabilities of the Taichi programming language, which enables high-performance rendering of complex 3D scenes directly on the GPU.

The basic visualization pipeline in Coralai involves mapping the contents of the substrate tensor to a set of visual properties, such as color, opacity, or texture, and then rendering these properties using Taichi's built-in graphics primitives. For example, the values of a particular channel in the substrate (e.g., the energy level of each cell) might be mapped to a color gradient, with high values corresponding to bright colors and low values corresponding to dark colors. This mapping can be customized by the user to highlight different aspects of the simulation, such as the distribution of specific resource types, the presence of particular organism traits, or the overall health and diversity of the ecosystem.

One of the key advantages of using Taichi for visualization in Coralai is that it allows for smooth, real-time rendering of the simulation, even for large and complex environments. This is because Taichi's runtime is optimized for parallel execution on

modern GPUs, which can process and display large amounts of data with minimal overhead. Additionally, Taichi provides a range of built-in tools and utilities for common visualization tasks, such as camera control, lighting, and post-processing effects, which can be easily integrated into the Coralai visualization pipeline.

Another important aspect of visualization in Coralai is interactivity. Users can interact with the simulation in real-time using a variety of input devices, such as the mouse and keyboard, to manipulate the camera view, adjust simulation parameters, or introduce new stimuli into the environment. For example, users can "paint" new resource patches or obstacles directly into the substrate using the mouse, or adjust the global temperature or nutrient availability using keyboard shortcuts. These interactive features allow users to explore different scenarios and test hypotheses about the behavior and evolution of the simulated ecosystem.

Chapter 4

EXPERIMENT

4.1 Chosen Dynamics

To test Coralai we tested a simple substrate, physics, and weather designed to minimally demonstrate the potential emergent dynamics. This initial experiment is primarily limited by the simplistic weather system and overly generous physical constraints. This enabled relatively simple organisms to propagate across the entire environment and out-compete budding, novel organisms. This can be analogized to a nutrient rich medium being colonized by a homogeneous mold as opposed to a diverse ecosystem of various flora, fauna, and decomposers which can accommodate a range of weather conditions.

Any experiment in Coralai must consist of a few definitions: the channels, sensors, actuators, physics, and weather. The environment of our experiment contains three main channels: energy, infrastructure, and communication. Energy is freely available in the environment and is added and removed via weather. Infrastructure is generated by organisms by "investing" energy available on its cell. Infrastructure can then be "liquidated" to return to energy. These are the first two actuators of the organism. Additionally, organisms can explore cells according to the provided kernel, in this case a 9 cell Moore's neighborhood with a toroidal boundary condition. Exploration occurs via four actuators: none, forward, left, and right. These are all relative to a rotation parameter stored on each cell, which is randomly initialized then updated to match incoming explorations that update or reinforce a cell's genome. The maximal output is used to transfer a portion of a cell's infrastructure to a neighbor, carrying

its genome with it. This is subject to competitive rules, whereby the quantity of infrastructure given to a neighbor must exceed the neighbor's infrastructure for the target cell to receive the exploring cell's genome. Otherwise, the target cell retains its original genome and simply receives the new infrastructure as an investment. Finally, each cell outputs values for the communication channels, which are unaffected by physics and simply normalized to be centered at 0. Thus, the sensors of the organism are energy, infrastructure, and communication and the actuators are invest, liquidate, and explore.

The communication actuators are a common addition to NCA [37, 36, 49] and provide cells with the means of coordinating actions and storing additional state information over time. In NCA optimized to grow into a target image, communication (referred to as "hidden") channels enable a cell to differentiate itself as, for example, an eye cell, instead of a tail cell.

The physics of our simulation takes these actuators and ensures that energy and mass are conserved and competitive dynamics are observed. Liquidation and investment can simply be proportionally balanced to ensure no energy or infrastructure is added or removed. Exploration, similarly, ensures no energy is added or removed and genomes are only adopted with sufficient exploratory investment. Physics also controls a few dynamics independent of the organisms' actions. These include dissipating dense areas of infrastructure and energy to neighboring cells, preventing runaway accumulation. Additionally, the physics governs the flow of energy towards areas of high infrastructure. This enables organisms to passively accumulate energy by developing gradients of increasing infrastructure.

Weather is implemented as a simple periodic addition and subtraction of energy from the environment. This is accomplished by adding the sine of the time step of the simulation to a normal distribution of values scaled via a parameter. This is overly

simple and produces a somewhat homogeneous and predictable pattern of energy that can be exploited by homogeneous organisms. However, even with this severe limitation, diverse organisms were found to coexist and compete.

4.2 Tests and Observations

The primary analysis of our experiment was exploratory—modifying parameters around weather, scale, and initial state and observing the resultant organisms. Due to the complex nature of diverse artificial life simulations, quantitative analysis can be difficult and was out of scope given the time constraints of the project. However, simple analysis of speciation, energy distributions over time, and parameter-organism compatibility can be implemented relatively simply and will soon be explored as future work.

Additional to exploratory analysis we observed the performance characteristics of the simulations, identifying bottlenecks and bounds of scale.

4.2.1 Quantitative Measures

To quantitatively assess the behavior and dynamics of the Coralai system, we can apply several of the metrics discussed in the background section. These measures can help us understand the emergence of complex patterns, the adaptability of organisms, and the overall health and diversity of the ecosystem. In this section, we describe how these metrics can be calculated and applied to the Coralai framework.

4.2.1.1 Measuring Complexity in Coralai

Complexity is a crucial aspect of adaptive and evolving systems, and quantifying it can provide valuable insights into the dynamics and health of the Coralai ecosystem. In the background section, we discussed how the coexistence of order and chaos at critical states enables the exploration and consolidation of new forms that persist across time. To assess whether the Coralai system exhibits such critical behavior, we need to measure its complexity and analyze how it changes over time.

In this section, we present several entropy-based approaches to measuring the complexity of the Coralai system based on the distribution of cell types across the grid. We focus on characterizing cell types using a combination of energy, infrastructure, and genome key values, as these properties capture essential aspects of the system's state and evolutionary dynamics. Energy and infrastructure levels reflect the cells' ability to survive, grow, and interact with their environment, while genome keys represent the genetic diversity and evolutionary history of the organisms.

It is important to note that other cell properties, such as communication channel values, could also be incorporated into the complexity measure. Communication channels can provide additional information about the cells' behavior and interactions, and their inclusion may lead to a more comprehensive assessment of complexity. However, for the purpose of this demonstration, we limit our analysis to energy, infrastructure, and genome keys.

4.2.1.2 Shannon Entropy

The Shannon entropy is a widely used measure of complexity and diversity in various fields, including ecology [25], physics [14], and artificial life [68]. It quantifies

the amount of information or uncertainty in a probability distribution, with higher entropy values indicating a more complex and diverse system.

To calculate the Shannon entropy, we first need to define cell types based on the relevant properties of each cell. Let p_i be the proportion of cells belonging to type i, where i ranges from 1 to n, the total number of distinct cell types. The Shannon entropy of the cell type distribution is given by:

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i. \tag{4.1}$$

To apply this measure to Coralai, we would follow these steps:

At each time step, create a histogram of cell types based on the energy, infrastructure, and genome key values of each cell. Normalize the histogram to obtain the probabilities p_i. Compute the entropy using the formula above. By tracking the entropy over time, we can observe how the complexity of the system evolves as organisms interact and adapt to their environment. A high entropy value would indicate a diverse and complex ecosystem, while a low entropy value would suggest a more homogeneous or ordered state. Analyzing the topology of the entropy curve over time can provide insights into the system's behavior, such as identifying critical points or phase transitions between order and chaos [46].

4.2.1.3 Discretized Entropy

When dealing with continuous variables like energy and infrastructure, the Shannon entropy may overestimate complexity due to cells with slightly different values being considered different types. To address this, we can discretize the continuous variables into bins or ranges before calculating the entropy.

The choice of binning strategy can affect the resulting complexity measure. One approach is to use fixed, equally spaced bins based on the range of values observed in the system. For example, if the energy values range from 0 to 1, we can divide this range into b equal-width bins, such as [0, 0.1), [0.1, 0.2), ..., [0.9, 1]. The number of bins, b, determines the granularity of the discretization, with larger values of b resulting in a finer-grained representation of the continuous variables.

Alternatively, we can use adaptive binning strategies that adjust the bin boundaries based on the distribution of the data. For example, the bin boundaries could be chosen such that each bin contains an equal number of cells (equiprobable binning) or based on the quantiles of the data (quantile binning) [12]. These adaptive strategies can better capture the underlying structure of the data and provide a more informative discretization.

Once the binning strategy is chosen, let p_{ijk} be the proportion of cells belonging to energy bin i, infrastructure bin j, and genome key k. We can then calculate the discretized entropy as:

$$H_d = -\sum_{i=1}^{b}\sum_{j=1}^{b}\sum_{k=1}^{g} p_{ijk} \log_2 p_{ijk}, \tag{4.2}$$

where b is the number of energy and infrastructure bins, and g is the total number of distinct genome keys.

To visualize the distribution of cell types, we can create a 3D histogram, where each axis represents one of the discretized variables (energy, infrastructure, and genome key). Each bin in the 3D histogram corresponds to a unique combination of energy, infrastructure, and genome key values, and the height of each bin represents the number of cells belonging to that cell type.

The discretized entropy measure provides a more robust estimate of complexity by grouping similar cell types together based on their energy and infrastructure ranges. By comparing the discretized entropy values for different binning strategies or numbers of bins, we can assess the sensitivity of the complexity measure to these choices and select an appropriate level of granularity for the analysis.

4.2.1.4 Continuous Entropy Estimators

If a more precise measure of complexity is required, we can use continuous entropy estimators that directly estimate the entropy from the continuous energy and infrastructure values without the need for discretization. These estimators are based on the idea of approximating the probability density function of the continuous variables using the observed data points.

One such estimator is the nearest-neighbor entropy estimator [27], which estimates the entropy based on the distances between each point and its nearest neighbors in the feature space. Let $\mathbf{x}_i = (e_i, f_i, g_i)$ be the feature vector of cell i, where e_i, f_i, and g_i are the energy, infrastructure, and genome key values, respectively. The nearest-neighbor entropy estimator is given by:

$$H_{nn} = \frac{1}{N} \sum_{i=1}^{N} \log_2 \left(N \cdot \frac{\pi^{d/2}}{\Gamma(d/2+1)} \cdot r_i^d \right) + \log_2(2) + \gamma, \tag{4.3}$$

where N is the total number of cells, d is the dimensionality of the feature space (in this case, $d = 3$), r_i is the distance between cell i and its nearest neighbor, Γ is the gamma function, and $\gamma \approx 0.5772$ is the Euler-Mascheroni constant. The gamma function and Euler-Mascheroni constant are mathematical constants that arise in the

derivation of the nearest-neighbor entropy estimator and are necessary for its accurate computation.

Another continuous entropy estimator is the kernel density entropy estimator [6], which estimates the entropy based on a kernel density estimate of the probability density function. The kernel density entropy estimator is given by:

$$H_{kde} = -\int p(\mathbf{x}) \log_2 p(\mathbf{x}) d\mathbf{x}, \tag{4.4}$$

where $p(\mathbf{x})$ is the kernel density estimate of the probability density function, given by:

$$p(\mathbf{x}) = \frac{1}{Nh^d} \sum_{i=1}^{N} K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right), \tag{4.5}$$

where K is the kernel function (e.g., Gaussian kernel) and h is the bandwidth parameter that controls the smoothness of the density estimate.

Continuous entropy estimators provide a more accurate estimate of complexity by considering the continuous nature of the energy and infrastructure values. However, they are more computationally expensive and sensitive to the choice of parameters, such as the number of neighbors or the kernel bandwidth.

4.2.1.5 Interpreting Complexity Measures

Once we have calculated the complexity measures for the Coralai system, we can interpret the results in the context of the background section's discussion on criticality and the balance between order and chaos.

A very high entropy value indicates a diverse and complex ecosystem, with a wide variety of cell types and interactions. This could suggest a system that is overly chaotic and free. On the other hand, a very low entropy value implies a more homogeneous or ordered state, which may limit the system's ability to evolve and respond to perturbations.

By analyzing the topology of the entropy curve over time, we can identify phase transitions or critical points in the system's dynamics. A sudden increase in entropy may indicate a transition from an ordered to a chaotic state, while a plateau or decrease in entropy may suggest the emergence of new, stable configurations or the consolidation of existing forms.

Additionally, we can compare the complexity measures for different experimental conditions or parameter settings to assess their impact on the system's behavior. For example, we can investigate how changes in the environment, such as variations in resource availability or mutation rates, affect the complexity and adaptability of the ecosystem.

It is important to note that due to time constraints, we were unable to apply these complexity measures to the Coralai system in this study. However, the framework presented here provides a foundation for future research on quantifying and analyzing complexity in artificial life systems.

Measuring complexity is a crucial step towards understanding the dynamics and evolution of the Coralai ecosystem. By using entropy-based measures, such as Shannon entropy, discretized entropy, and continuous entropy estimators, we can quantify the diversity and complexity of the system and relate it to the concepts of criticality and the balance between order and chaos. Future work should focus on applying these

measures to the Coralai system and interpreting the results in the context of the system's behavior and evolutionary dynamics.

4.2.2 Scale-invariance

Scale-invariance can be assessed by measuring the distribution of organism sizes and checking if it follows a power law. Let $p(s)$ be the probability of observing an organism of size s, where size can be defined as the number of cells belonging to the organism. If the system exhibits scale-invariance, the size distribution should follow a power law:

$$p(s) \propto s^{-\alpha}, \tag{4.6}$$

where α is the scaling exponent. To check if the size distribution follows a power law, we can plot $\log p(s)$ against $\log s$ and look for a linear relationship. The slope of the line would give an estimate of the scaling exponent α.

In Coralai, we can calculate the size distribution by creating a histogram of organism sizes at each time step, where the size of an organism is the number of cells that share its genome key. We can then normalize the histogram to obtain the probabilities $p(s)$, and plot $\log p(s)$ against $\log s$ to check for a power law relationship.

4.2.3 Measuring Multi-Scale Complexity in Coralai

Inspired by the work of Bagrov et al. [1], we propose to measure the multi-scale structural complexity of patterns emerging in the Coralai system. This approach is based on the idea that a system is more scale-invariant if it exhibits distinctive features

at different characteristic scales. This is similar to measures of fractal dimension such as box-counting, but can be more readily applied to domains with multiple channels of continuous values. By applying a coarse-graining procedure to the Coralai substrate and comparing the resulting patterns at different scales, we can quantify the system's complexity and gain insights into its behavior and dynamics.

To measure the multi-scale complexity of the Coralai substrate, we first need to define a coarse-graining procedure. Following the method outlined in [1], we can use a simple discrete decimation scheme. At each iteration of the coarse-graining procedure, the substrate is divided into blocks of $\Lambda \times \Lambda$ cells, and each block is replaced by a single cell. The state of the new cell is calculated as the average of the states of the cells in the corresponding block:

$$s_{ij}(k) = \frac{1}{\Lambda^2} \sum_{l=0}^{\Lambda-1} \sum_{m=0}^{\Lambda-1} s_{\Lambda i+m, \Lambda j+l}(k-1), \tag{4.7}$$

where $s_{ij}(k)$ represents the state of the cell at position (i, j) after k iterations of the coarse-graining procedure, and $s_{ij}(0)$ corresponds to the original substrate.

After applying the coarse-graining procedure for a desired number of iterations, we obtain a stack of renormalized substrates at different scales. To compute the multi-scale complexity, we compare the patterns in neighboring scales by calculating their overlap. The overlap between two patterns at scales k and $k-1$ is defined as:

$$O_{k,k-1} = \frac{1}{L_k^2} \sum_{i=1}^{L_k} \sum_{j=1}^{L_k} s_{ij}(k) \cdot \sum_{m=0}^{\Lambda-1} \sum_{l=0}^{\Lambda-1} s_{\Lambda i+m, \Lambda j+l}(k-1), \tag{4.8}$$

where L_k is the linear size of the substrate after k iterations of the coarse-graining procedure.

The multi-scale structural complexity C is then defined as the sum of the differences between the overlaps of neighboring scales:

$$C = \sum_{k=1}^{N} |O_{k,k-1} - \frac{1}{2}(O_{k-1,k-1} + O_{k,k})|, \tag{4.9}$$

where N is the total number of coarse-graining iterations.

By applying this measure to the Coralai substrate at different points in the parameter space or at different moments in time, we can study how the system's complexity evolves and relates to its behavior. For example, we can investigate:

- Phase transitions: By computing the multi-scale complexity across the parameter space (e.g., varying energy and infrastructure levels, mutation rates, or environmental conditions), we can detect phase transitions and identify critical points where the system's complexity is maximized.
- Evolutionary dynamics: By tracking the multi-scale complexity over time, we can study how the system's complexity evolves as organisms interact, adapt, and evolve. This can provide insights into the emergence of complex behaviors and the interplay between order and disorder in the system.
- Robustness and adaptability: By comparing the multi-scale complexity of the system under different perturbations or environmental changes, we can assess its robustness and adaptability. A system that maintains a relatively high complexity despite disturbances may be considered more resilient and capable of generating novel adaptive behaviors as opposed to one where multi-scale complexity collapses to order or chaos on perturbation.

Implementing the multi-scale complexity measure in the Coralai framework is straightforward, as it only requires applying the coarse-graining procedure to the substrate and calculating the overlaps between neighboring scales. This can be done efficiently using the existing data structures and parallel computing capabilities of the framework. For example, the coarse-graining steps can be done via repeated application of a Gaussian kernel (blurring), referred to as a Gaussian pyramid. However, this produces coarse images each half the size of the previous, limiting samples. This can be counteracted by convolving the original image with Gaussian kernels of various sizes and with various step sizes to get diverse samples.

Chapter 5

RESULTS

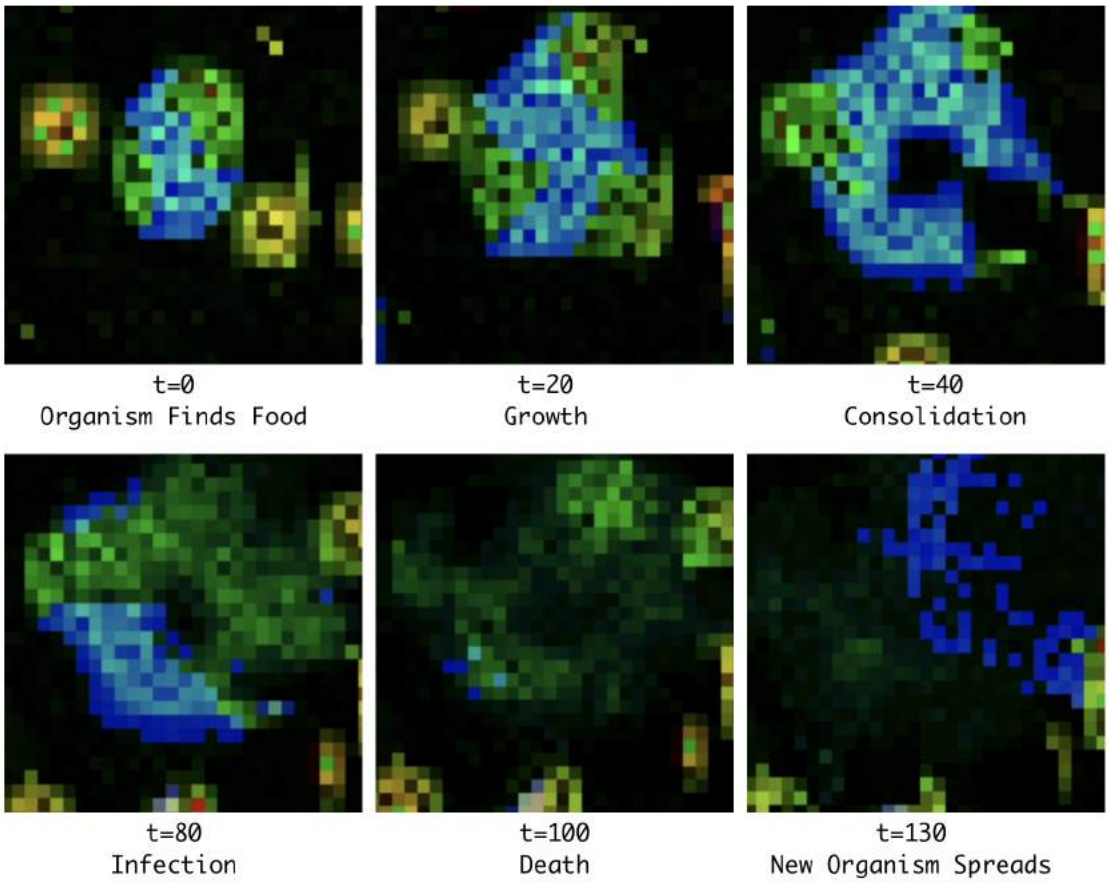

Figure 5.1: The growth and death of an organism via competition. This visualization uses red to represent energy, green for infrastructure, and blue for genome key. This simple method unfortunately results in the invisibility of some genomes, as seen at t=80 where the infecting organism appears simply as darkness until it adopts a larger genome key upon removing the old organism at t=130

5.1 Performance

The primary performance bottleneck was found to be the mutation of genomes using NEAT. This is simply because it is done on the CPU and requires pausing the

simulation until the new genomes are created and used to create the PyTorch neural networks, which itself is quite fast.

The largest simulation contained 640,000 individual cells each with 16 channels. These were all updating in parallel with 500 unique networks (species) each containing 1200 parameters (6 sense channels with a kernel of 9 cells and 11 actuators). This included all physics for mass and energy conservation as well as exploration and competition. This was able to run at 20fps on an Apple M1 chip using metal performance shaders. This frame rate would dip down upon mutation, creating an average frame rate of 15 frames per second.

Separate from NEAT, this experiment contained a number of performance bottlenecks around taking weight information on and off the GPU as genomes were updated and testing the state of the system to ensure the successful application of physics. Without these constraints, an identical simulation with static organisms was able to run at 75 fps with real-time interaction and visualization, which was constrained by the refresh rate of the monitor.

Previous experiments with simpler dynamics demonstrated that large, static, networks, containing as much as 1m parameters, and physics of similar complexity, could run at up to 1200 fps on 640,000 16-channel cells without visualization.

5.2 Observed Behavior

Figure 5.1 shows a small sample of a larger ecosystem where two organisms compete for resources. Over the course of 130 time steps an organism grows to consume three sources of energy and begins to consolidate its form when a competing organism infects it. The competing organism adopts a more mobile strategy difficult to convey in

static images. The competing organism is able to maintain cells with higher infrastructure by consolidating into smaller points. This organism quickly out-competes the more static original one and devours the energy, turning it into infrastructure that it caries with it through exploratory moves where all of its infrastructure is spread to neighboring cells, effectively moving across space.

Figure 5.2 shows the introduction and progression of a very dominant organism that relies on homogeneous and reliable energy availability to spread in a fire-like manner. This organism takes little care to consolidate resources and barely survives the "night" cycle. Due to this weakness, other organisms are able to maintain their infrastructure and accumulate resources to slowly grow resilient to the dominant organism.

Of special interest is an organism found in the top left at time step 500 which seemingly seeks out localized, dense energy rather than relying on weather-generated energy. A less interesting organism, found in the center of the simulation, simply consolidates infrastructure into dust and conserves itself, acting as a food source for the dominant wave-like organism.

Figure 5.3 shows the random addition of dense energy sources and their counterpart of ablations, where energy, infrastructure, and genomes are removed. Here dense energy is consumed by slime mold-like organisms with both a branching and consolidated morphology. These organisms maintain their boundaries between each other and often out-compete novel mutations. This is with the exception of blue areas where mutated cells propagate "downstream" of an organism to dominate a subsection of its infrastructure. There are also a number of mutations that remain local, similar to tumors or parasites, that survive on the energy accumulation of their host organisms.

Figure 5.4 Demonstrates the initial conditions of a simulation where energy and genomes are removed from the central area. The boundary is seeded with random

genomes and dense energy, providing a fertile and competitive environment that is stable during night cycles where energy is removed across the environment. Here we see the emergence of an organism that carelessly explores the central area in an energy intensive process that eventually fails during a night cycle, causing its extinction. However, the energy left over after its death enables the exploration of the barren area by other organisms hosted on the boundary. This demonstrates the potential power of heterogeneous energy landscapes for creating niches that support diverse species. It also demonstrates a limitation in the current simplistic visualization system where genome keys rapidly change on a culling, resulting in difficulty in tracking organisms.

Figure 5.5 demonstrates a more stable ecosystem where diverse organisms co-exist. Here we see a branching organism that exists on its own as well as in a colony supported by more stable organisms. This potentially displays a level of symbiosis where branching organisms find resources to be stored by the more stable, congealed organisms. We also notice a similar species of organism that, instead of branching, loops back on itself, demonstrating subtle mutations in exploratory strategies.

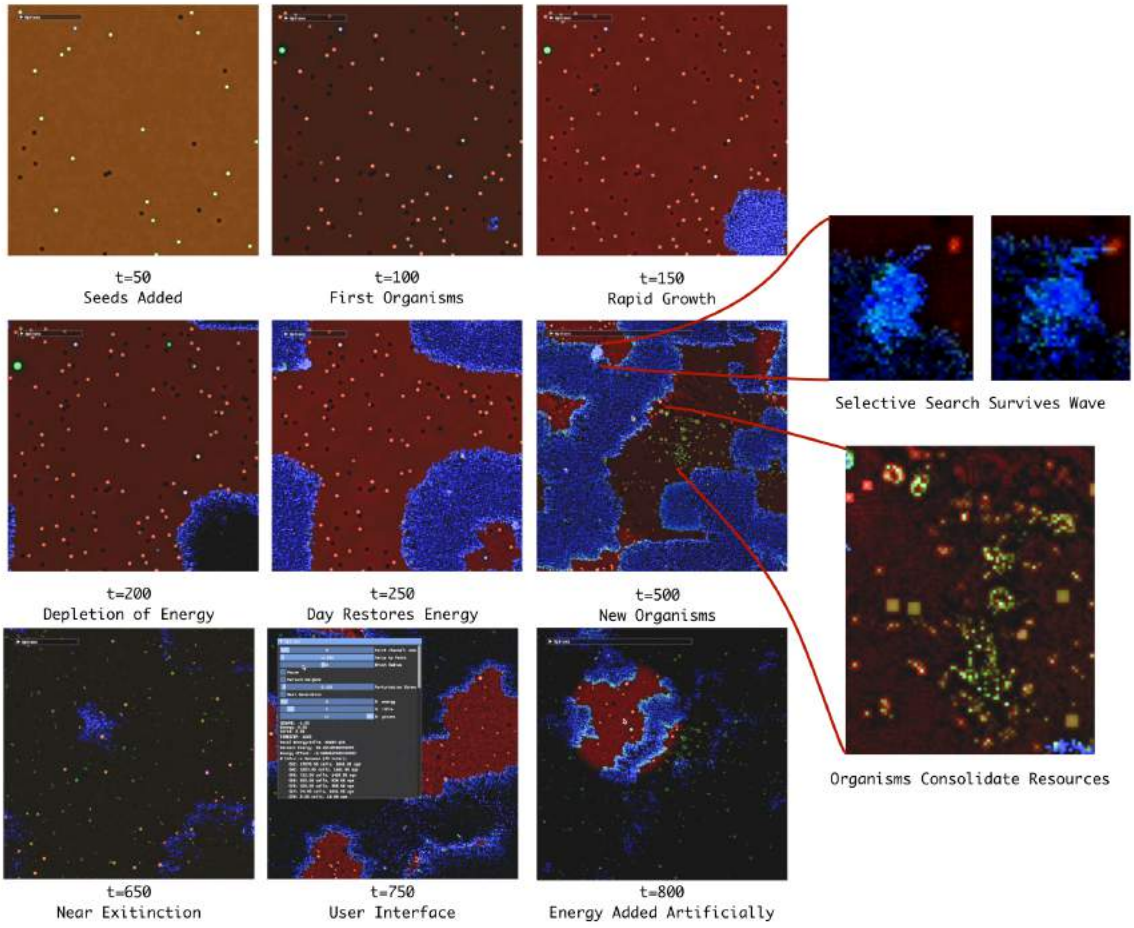

Figure 5.2: The progression of a simulation containing a dominant organism that relies on the regularity of the weather. This additionally shows the simple user interface that enables the artificial addition of energy and observation o cell states. Like other figures, red represents energy, green represents infrastructure, and blue represents genomes

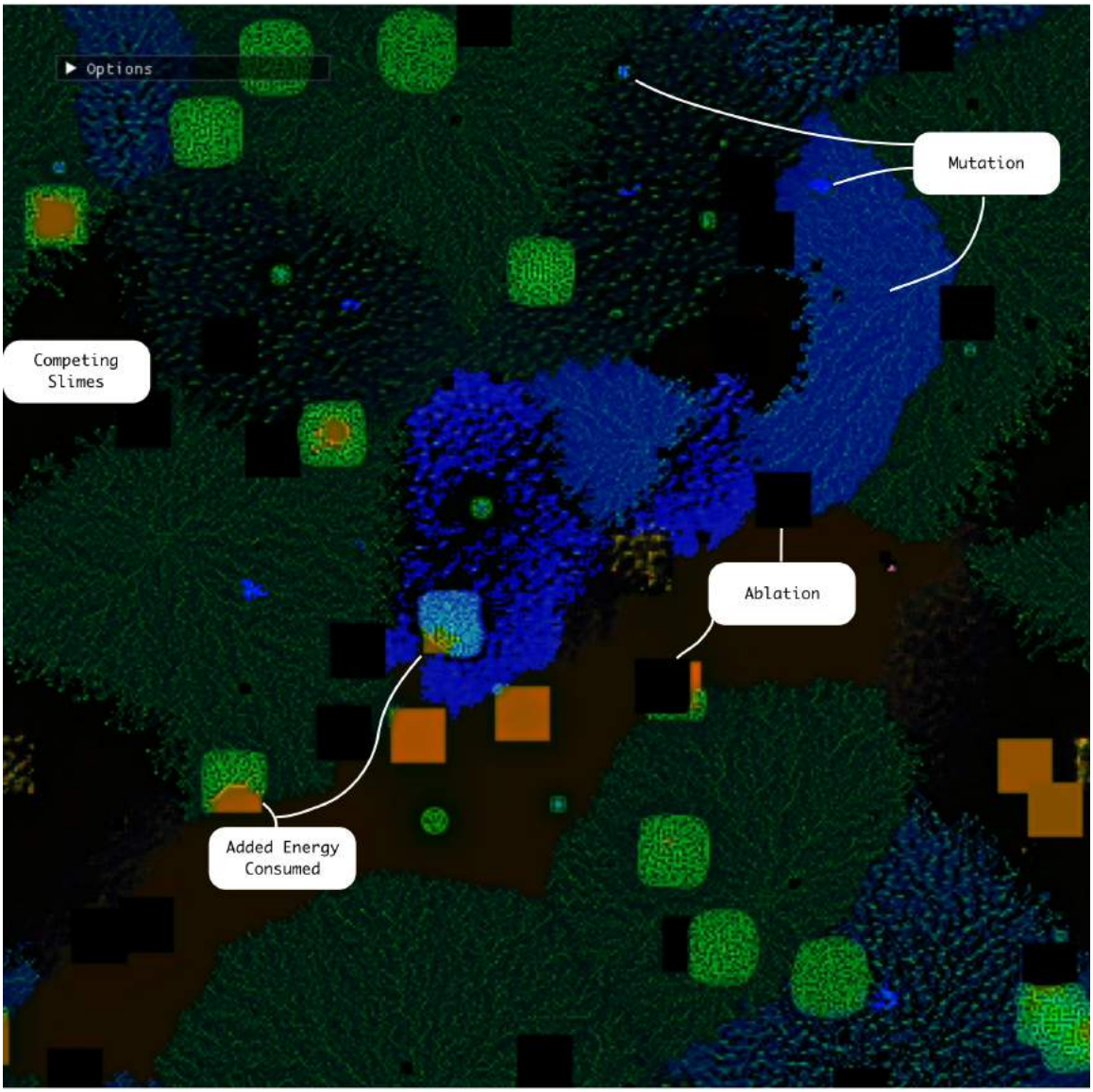

Figure 5.3: This figure demonstrates the random addition of dense energy sources and ablation of areas. The variety of slime-mold like organism coexist by competing on boundaries. Blue areas represent genomes with larger keys, meaning recent mutations. It is possible to observe areas of mutated organisms that propagate or remain local.

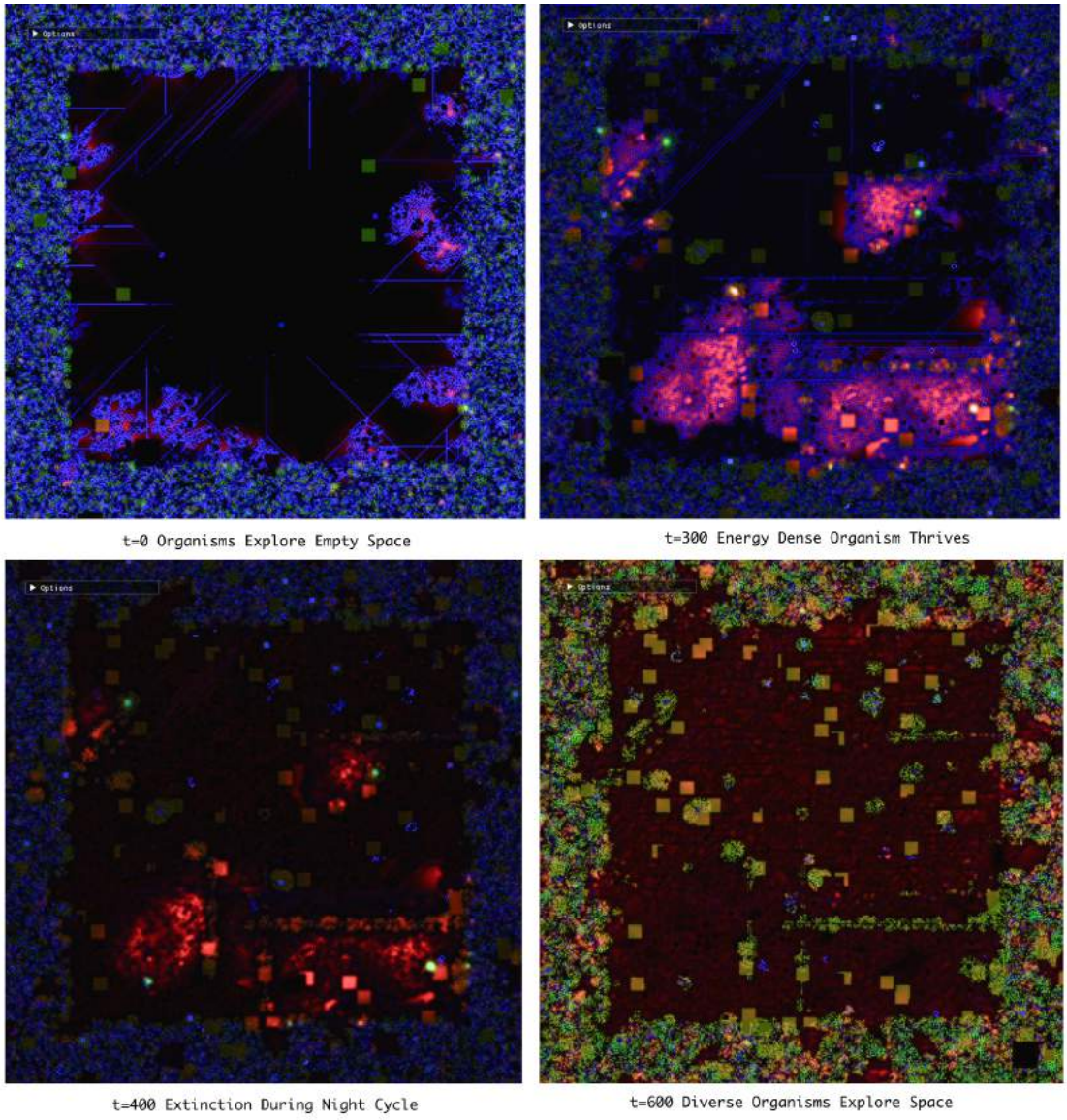

Figure 5.4: Here a simulation with random genomes and energy on the edges generates an energy dense exploratory organism whose extinction enables the flourishing of new diversity in the barren space. The rapid change from blue to green represents re-ordering of genome-keys after culling

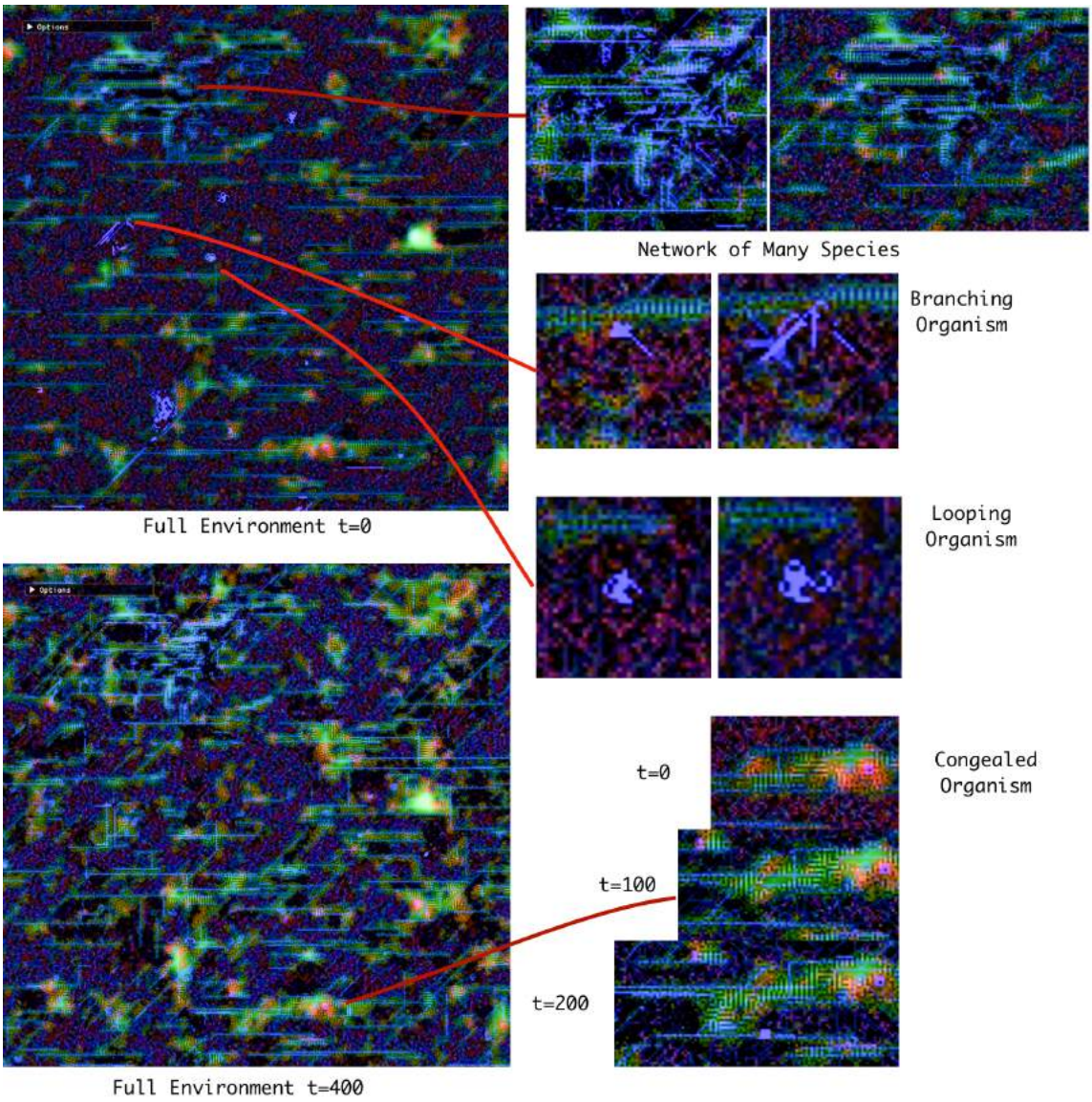

Figure 5.5: A diverse ecosystem of organisms coexisting.

Chapter 6

DISCUSSION

The recognition of competency as a multi-scale, decentralized, emergent phenomenon is a unifying theme across various fields. From the growing interest in gut microbiomes and embodied cognition to the application of cognitive neuroscience and psychology in studying deep learning models, there is a clear shift away from reductionist, mind-body dualistic perspectives towards a more holistic, enactive understanding of intelligence [65, 33, 15]. The recent advancements in large language models have further highlighted the necessity of interdisciplinary approaches, drawing from ecology, economics, and psychology, to study and develop artificial intelligence and its interaction with society. Central to this convergence is the concept of collective intelligence and multi-level competency [18, 15], which emphasizes the capacity for emergent intelligence in complex systems, whether composed of cells, individuals, AI agents, or even abstract entities like stories.

The insights gained from the study of artificial life (ALife) and emergent intelligence have significant implications for the nascent field of AI alignment. As AI systems become increasingly embedded in complex socio-technical systems, such as the economy, social networks, and the internet, they have the potential to exhibit emergent behaviors and competencies that may have unintended or difficult-to-detect consequences. Attempting to align such systems through brute force methods may prove futile without compromising their adaptive capabilities. Moreover, the effects of these systems may not be immediately apparent, as exemplified by the delayed recognition of social media's impact on mental health. A more promising approach may be to view these systems as ecosystems, where AI and human agents strive to establish

symbiotic relationships to address shared challenges. By providing a platform for studying the dynamics of emergent behavior in complex systems, ALife research can offer valuable guidance for the development and governance of AI-human collectives.

Chapter 7

CONCLUSION

The development of Coralai as a framework for simulating open-ended evolution and emergent complexity in artificial life represents a meaningful addition in the field. By leveraging the power of modern computational tools, such as Taichi for efficient parallel computing, PyTorch for deep learning, and NEAT for neuroevolution, Coralai enables the simulation of large-scale, interactive environments with hundreds of unique organisms. The modular and flexible design of Coralai, exemplified by its support for user-defined physics and weather systems, as well as the ability to customize organism sensors, actuators, and neural architectures, makes it adaptable to a wide range of research questions and experimental setups.

The initial experiments conducted with Coralai demonstrate its capacity to generate diverse and complex ecosystems, even with relatively simple environmental conditions. The emergence of organisms exhibiting various strategies, such as resource exploitation, niche specialization, and cyclic dominance, highlights the potential for Coralai to serve as a valuable tool for studying the dynamics of evolution and adaptation in complex systems. Moreover, the observed scale-invariant properties of the simulated environments, as evidenced by the coexistence of organisms at different spatial and temporal scales, suggest that Coralai captures important characteristics of real-world ecosystems and may offer insights into the fundamental principles governing the emergence of life and intelligence.

The performance analysis of Coralai reveals both its strengths and limitations. The ability to simulate environments with over 500 organisms on a grid of 640,000 cells at

interactive frame rates demonstrates the efficiency and scalability of the framework. However, the reliance on CPU-based mutation operations and the need for frequent data transfer between the GPU and CPU present significant bottlenecks that limit the overall performance. Addressing these limitations through the implementation of a custom neuroevolution algorithm and the optimization of data transfer operations will be crucial for enabling larger-scale and more complex simulations in the future.

The discussion of the broader implications of ALife research and its relevance to other fields, such as embodied cognition, collective intelligence, and AI alignment, underscores the importance of frameworks like Coralai. As AI systems become increasingly integrated into complex socio-technical systems, understanding the dynamics of emergent behavior and developing strategies for managing and aligning these systems will be essential. By providing a platform for studying the evolution of intelligence and adaptation in complex environments, Coralai can contribute to the development of more robust, adaptive, and collaborative AI systems.

In conclusion, Coralai represents a significant step forward in the field of artificial life and provides a powerful tool for exploring fundamental questions about the nature of life, cognition, and emergence. While there are limitations to the current implementation, the modular and flexible design of the framework allows for future improvements and extensions. By continuing to develop and refine tools like Coralai, researchers can deepen our understanding of the principles governing the emergence of complexity in natural and artificial systems, paving the way for the creation of more advanced and adaptive AI systems that can effectively collaborate with humans to address the challenges of the future.

BIBLIOGRAPHY

[1] A. A. Bagrov, I. A. Iakovlev, A. A. Iliasov, M. I. Katsnelson, and V. V. Mazurenko. Multiscale structural complexity of natural patterns. 117(48):30241–30251. Publisher: Proceedings of the National Academy of Sciences.

[2] P. Bak, K. Chen, and M. Creutz. Self-organized criticality in the 'game of life. 342(6251):780–782. Publisher: Nature Publishing Group.

[3] A. Barbieux. github.com/aidanbx/coralai: Spatial evolution of NCA ecosystems with physics. highly parallelized via taichi and PyTorch HyperNEAT.

[4] A. Barbieux. github.com/aidanbx/PyTorch-NEAT. original-date: 2024-02-14T14:23:10Z.

[5] A. Barbieux and R. Canaan. EINCASM: Emergent intelligence in neural cellular automaton slime molds. MIT Press.

[6] J. Beirlant, E. Dudewicz, L. Gyor, and E. Meulen. Nonparametric entropy estimation: An overview. 6.

[7] S. G. BRUSH. History of the lenz-ising model. 39(4):883–893. Publisher: American Physical Society.

[8] B. W.-C. Chan. Lenia - biology of artificial life. 28(3):251–286.

[9] B. W.-C. Chan. Lenia and expanded universe. In *The 2020 Conference on Artificial Life*, pages 221–229. MIT Press.

[10] N. Cheney, R. MacCurdy, J. Clune, and H. Lipson. *Unshackling Evolution: Evolving Soft Robots with Multiple Materials and a Powerful Generative*

Encoding. Journal Abbreviation: GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference Publication Title: GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference.

[11] G. Dodig-Crnkovic. How GPT realizes leibniz's dream and passes the turing test without being conscious. 8(1):66. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute.

[12] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. 1995.

[13] K. Endo and K. Yasuoka. Neural cellular maze solver.

[14] D. P. Feldman and J. P. Crutchfield. Measures of statistical complexity: Why? 238(4):244–252.

[15] C. Fields and M. Levin. Competency in navigating arbitrary spaces as an invariant for analyzing cognition in diverse embodiments. 24(6):819. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

[16] C. Grasso and J. Bongard. Empowered neural cellular automata. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 108–111.

[17] K. Gregor and F. Besse. Self-organizing intelligent matter: A blueprint for an AI generating algorithm.

[18] D. Ha and Y. Tang. Collective intelligence for deep learning: A survey of recent developments. 1(1):26339137221114874. Publisher: SAGE Publications.

[19] L. Helms and J. Clune. Improving HybrID: How to best combine indirect and direct encoding in evolutionary algorithms. 12(3):e0174635. Publisher: Public Library of Science.

[20] J. Hesse and T. Gross. Self-organized criticality as a fundamental property of neural systems. 8. Publisher: Frontiers.

[21] W. D. Hillis. Intelligence as an emergent behavior; or, the songs of eden. 117(1):175–189. Publisher: The MIT Press.

[22] Y. Hu, L. Anderson, T.-M. Li, Q. Sun, N. Carr, J. Ragan-Kelley, and F. Durand. DiffTaichi: Differentiable programming for physical simulation.

[23] Y. Hu, T.-M. Li, L. Anderson, J. Ragan-Kelley, and F. Durand. Taichi: a language for high-performance computation on spatially sparse data structures. 38(6):1–16.

[24] Y. Hu, J. Liu, X. Yang, M. Xu, Y. Kuang, W. Xu, Q. Dai, W. T. Freeman, and F. Durand. QuanTaichi: a compiler for quantized simulations. 40(4):1–16.

[25] L. Jost. Entropy and diversity. 113(2):363–375. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.2006.0030-1299.14714.x.

[26] Y. Khaluf, E. Ferrante, P. Simoens, and C. Huepe. Scale invariance in natural and artificial collective systems: a review. 14(136):20170662. Publisher: Royal Society.

[27] A. Kraskov, H. Stoegbauer, and P. Grassberger. Estimating mutual information. 69(6):066138.

[28] S. Kuriyama, W. Noguchi, H. Iizuka, K. Suzuki, and M. Yamamoto. Gradient climbing neural cellular automata. MIT Press.

[29] S. Lee, J. Yosinski, K. Glette, H. Lipson, and J. Clune. Evolving gaits for physical robots with the HyperNEAT generative encoding: The benefits of simulation. In A. I. Esparcia-Alcázar, editor, *Applications of Evolutionary Computation*, Lecture Notes in Computer Science, pages 540–549. Springer.

[30] J. Lehman. github.com/uber-research/PyTorch-NEAT. original-date: 2018-08-28T23:53:20Z.

[31] J. Z. Leibo, E. Hughes, M. Lanctot, and T. Graepel. Autocurricula and the emergence of innovation from social interaction: A manifesto for multi-agent intelligence research.

[32] M. Levin. Technological approach to mind everywhere: An experimentally-grounded framework for understanding diverse bodies and minds. 16.

[33] P. Lyon, F. Keijzer, D. Arendt, and M. Levin. Reframing cognition: getting down to biological basics. 376(1820):20190750. Publisher: Royal Society.

[34] B. B. Mandelbrot and J. A. Wheeler. The fractal geometry of nature. 51(3):286–287.

[35] H. R. Maturana and F. J. Varela. *Autopoiesis and Cognition: The Realization of the Living*. Springer Science & Business Media. Google-Books-ID: nVmcN9Ja68kC.

[36] A. Mordvintsev, E. Randazzo, and C. Fouts. Growing isotropic neural cellular automata. In *The 2022 Conference on Artificial Life*. MIT Press.

[37] A. Mordvintsev, E. Randazzo, E. Niklasson, and M. Levin. Growing neural cellular automata. 5(2):e23.

[38] J.-B. Mouret and J. Clune. Illuminating search spaces by mapping elites.

[39] M. E. J. Newman. Power laws, pareto distributions and zipf's law. 46(5):323–351.

[40] S. Nichele, M. B. Ose, S. Risi, and G. Tufte. CA-NEAT: Evolved compositional pattern producing networks for cellular automata morphogenesis and replication. 10(3):687–700.

[41] A. Ororbia and K. Friston. Mortal computation: A foundation for biomimetic intelligence.

[42] N. Oros and C. L. Nehaniv. Sexyloop: Self-reproduction, evolution and sex in cellular automata. In *2007 IEEE Symposium on Artificial Life*, pages 130–138. IEEE.

[43] R. Pande and D. Grattarola. Hierarchical neural cellular automata. MIT Press.

[44] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library.

[45] E. Plantec, G. Hamon, M. Etcheverry, P.-Y. Oudeyer, C. Moulin-Frier, and B. W.-C. Chan. Flow-lenia: Towards open-ended evolution in cellular automata through mass conservation and parameter localization.

[46] M. Prokopenko, F. Boschetti, and A. Ryan. An information-theoretic primer on complexity, self-organisation and emergence. 15.

[47] M. J. Ramstead, A. Constant, P. B. Badcock, and K. J. Friston. Variational ecology and the physics of sentient systems. 31:188–205.

[48] E. Randazzo and A. Mordvintsev. Biomaker CA: a biome maker project using cellular automata.

[49] E. Randazzo, A. Mordvintsev, E. Niklasson, M. Levin, and S. Greydanus. Self-classifying MNIST digits. 5(8):e00027.002.

[50] S. Risi and K. O. Stanley. An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. 18(4):331–363.

[51] J. B. Salem and S. Wolfram. Thermodynamics and hydrodynamics with cellular automata.

[52] H. Sayama. A new structurally dissolvable self-reproducing loop evolving in a simple cellular automata space. 5(4):343–365. Conference Name: Artificial Life.

[53] J. Secretan, N. Beato, D. B. D'Ambrosio, A. Rodriguez, A. Campbell, J. T. Folsom-Kovarik, and K. O. Stanley. Picbreeder: A case study in collaborative evolutionary exploration of design space. 19(3):373–403.

[54] K. Sims. Reaction-diffusion tutorial.

[55] K. Stanley and R. Miikkulainen. Efficient evolution of neural network topologies. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volume 2, pages 1757–1762. IEEE.

[56] K. O. Stanley. Compositional pattern producing networks: A novel abstraction of development. 8(2):131–162.

[57] K. O. Stanley. Why open-endedness matters. 25(3):232–235.

[58] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen. Designing neural networks through neuroevolution. 1(1):24–35.

[59] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. 15(2):185–212. Conference Name: Artificial Life.

[60] K. O. Stanley and J. Lehman. *Why Greatness Cannot Be Planned: The Myth of the Objective*. Springer International Publishing.

[61] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. 10(2):99–127.

[62] J. Stovold. Neural cellular automata can respond to signals. In *The 2023 Conference on Artificial Life*.

[63] A. M. Turing. The chemical basis of morphogenesis. 52(1):153–197.

[64] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. CreateSpace.

[65] F. J. Varela, E. Thompson, E. Rosch, and J. Kabat-Zinn. The embodied mind. Publisher: The MIT Press.

[66] 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.

[67] A. J. Watson and J. E. Lovelock. Biological homeostasis of the global environment: the parable of daisyworld. 35(4):284–289. Publisher: Taylor & Francis _eprint: https://doi.org/10.3402/tellusb.v35i4.14616.

[68] J. R. Woodward and A. Farjudian. Artificial life, the second law of thermodynamics, and kolmogorov complexity. In *2010 IEEE International Conference on Progress in Informatics and Computing*, volume 2, pages 1266–1269.