A Scalable Open-Source Simulation Framework for Neuroevolution and Multi-Agent Behavior

Jackson Baker, Jackson Salyards, Kai Sniadach, Adrian Quintero, Colton Underwood, M. Ilhan Akbas

Electrical Engineering and Computer Science, Embry-Riddle Aeronautical University

Daytona Beach, Florida

bakerj86@my.erau.edu , salyardj@my.erau.edu, sniadack@my.erau.edu, quinta11@my.erau.edu, underwc9@email.com, akbasm@erau.edu

ABSTRACT

Neuroevolution is a subfield of artificial intelligence (AI) and machine learning (ML) that leverages evolutionary algorithms to create, optimize, and train artificial neural networks. The study of neuroevolution and multi-agent behavior, a system of multiple autonomous entities, depends heavily on effective simulation frameworks to develop, test, and optimize complex interactions, such as drone swarm coordination and autonomous robot team strategies. Traditional custom-built simulation environments often exhibit limitations in scalability, flexibility, and reproducibility. To address these challenges, we present GENeSIS (Generalized Environment for Neuroevolution Simulation and Interactive Studies) an open-source simulation framework designed to overcome existing constraints. GENeSIS provides a highly modular and scalable system that enables rapid creation and deployment of diverse simulation environments, such as predator-prey dynamics or autonomous drone coordination, while supporting multiple neuroevolutionary models. We show our framework enhances experimental efficiency and adaptability by enabling simultaneous evaluation of distinct evolutionary models, including NEAT and HyperNEAT, with fully tunable parameters. We implement a predator-prey model, demonstrating the system's ability to facilitate complex agent interactions and evolutionary optimization. GENeSIS uniquely integrates remote execution capabilities, leveraging high-performance computing clusters for computationally intensive tasks. Additionally, a graphical user interface (GUI) with HTTP-based connectivity enables real-time monitoring and analysis of simulation outcomes from remote locations. The framework provides advanced data monitoring at multiple levels, from individual agent behaviors and group interactions to system-wide performance, enabling deep analysis of agent decision-making, emergent behaviors, and rapid optimization of neuroevolutionary algorithms. This comprehensive monitoring enhances interpretability and reproducibility in behavioral studies. Beyond its application in neuroevolution research, GENeSIS provides a user-friendly and scalable tool for advancing multi-agent system studies while reducing unnecessary development time and improving experiment reproducibility. The approach advances AI-driven simulation methodologies by addressing critical gaps, such as limited scalability and modularity in current tools, benefiting industries like aerospace, defense, and autonomous systems.

ABOUT THE AUTHORS

Jackson Baker is an undergraduate student at Embry-Riddle Aeronautical University's Computer Engineering program. They are studying under Professor Dr. Akbas.

Jackson Salyards is an undergraduate student at Embry-Riddle Aeronautical University's Computer Science program. They are studying under the guidance of Dr. Akbas. They are currently in the accelerated master's program and plan to complete it in December of 2026.

Kai Sniadach is an undergraduate Student at Embry-Riddle Aeronautical University on the Computer Science program's Cybersecurity track. They are studying under the guidance of Professor M. Ilhan Akbas.

Adrian Quintero is an undergraduate student at Embry-Riddle Aeronautical University's Computer Engineering program. They are studying under Professor Dr. Akbas.

Colton Underwood is an undergraduate student at Embry-Riddle Aeronautical University's Computer Engineering program. They are studying under Professor Dr. Akbas.

M. Ilhan Akbas is an associate professor at the Electrical Engineering and Computer Science Department, director of Flexible & Intelligent Complex Systems (FICS) research group, co-director of WIDE Lab, and a member of the leadership at the Center for Aerospace Resilient Systems (CARS) at Embry-Riddle Aeronautical University. He has over 15 years of experience in machine learning, complex systems, modeling and simulation. He received his Ph.D in Computer Engineering at the University of Central Florida (UCF), where he conducted research at the Artificial Intelligence (AI) Things Laboratory. His background also includes eight years of defense industry and enterprise level software development experience. His research has secured funding through grants from agencies such as National Science of Foundation, Federal Aviation Administration, US Air Force, Office of Naval Research, as well as industry. He is a senior member of IEEE, and a member of ACM, AIAA, SAE, International Alliance for Mobility Testing and Standardization, Complex Systems Society and Cyber Safety Commercial Aviation Team.

A Scalable Open-Source Simulation Framework for Neuroevolution and Multi-Agent Behavior

Jackson Baker, Jackson Salyards, Kai Sniadach, Adrian Quintero, Colton Underwood, M. Ilhan Akbas
Electrical Engineering and Computer Science, Embry-Riddle Aeronautical University
Daytona Beach, Florida
bakerj86@my.erau.edu , salyardj@my.erau.edu, sniadack@my.erau.edu, quinta11@my.erau.edu, underwc9@email.com, akbasm@erau.edu

PROBLEM STATEMENT & INTRODUCTION

Neuroevolution is a method for training artificial neural networks using evolutionary algorithms that has proven particularly effective for complex, dynamic domains such as swarm intelligence and multi-agent coordination. NEAT (Neuroevolution of Augmenting Topologies), introduced by Stanley & Miikkulainen, expands upon this by allowing for the growth and change of the networks structure through speciation and historical marking [1]. As an example, NEAT significantly outperforms fixed topology neuroevolution training in the double pole balancing task [2]. However, it is shown that NEAT can struggle with high dimensional problems. HyperNEAT, introduced by Stanley, D'Ambrosio, and Gauci, expands upon NEAT with indirect encoding techniques to build the network based on repeating spatial patterns generated by a hypercube [3]. While NEAT directly evolves the structure and weights of neural networks, HyperNEAT uses a generative approach via Compositional Pattern-Producing Networks (CPPNs), which allows it to represent and evolve large, complex networks more efficiently. This effectively moves the problem away from dimensionality and onto the problem structure itself.

The neuroevolution-based methods, such as NEAT and HyperNEAT, have emerged as robust alternatives for gradient based training algorithms [2], excelling at tasks such as swarm behavior, and areas with a high degree of dimensionality. Simulation is crucial for studying swarm behavior, particularly in areas with a high degree of dimensionality since it allows researchers to explore complex interactions and emergent properties that would be impractical or impossible to observe in real-world experiments. By creating virtual environments, scientists can systematically vary parameters, test hypotheses, and understand the intricate dynamics of large numbers of interacting agents without the physical constraints, costs, or risks associated with live trials. No current publicly available simulator natively combines multi-agent physics with neuroevolution pipelines. Researchers must stitch together bespoke code, hampering cross-study comparison and repeatability. Consequently, researchers are often forced to create custom simulation environments for each new task (see TeamBots[4], SWARM[5], RePast[6], EpidemicSim[7], FPolyOS[8]). Currently existing tools like Open AI's GYM and MASON allow for the deployment of environments for the study of swarm & multi-agent behavior, but critically both tools lack the ability to implement Neuroevolution, NEAT, or HyperNEAT [9, 10].

To tackle the limitations of existing tools, we developed GENeSIS (Generalized Environment for Neuro-Evolution Simulation and Interactive Studies), a simulation framework specifically with neuroevolution and multi-agent systems in mind. The objective of GENeSIS is *not* to adjudicate whether NEAT, HyperNEAT, or any other optimizer is intrinsically superior. Instead designed for flexibility and scalability, the framework makes it straightforward to prototype custom simulation environments and experiment with a range of evolutionary algorithms, including NEAT and HyperNEAT. It includes a lightweight GUI that allows researchers to gain insights into ongoing simulations, allowing for analysis and observation. As proof of concept, we implemented a predator-prey simulation to demonstrate the capabilities of the system. Even though it is not a highly complex environment, it effectively illustrates the complex interactions between agents and highlights the strengths of neuroevolution in the promotion of emergent strategies. The predator-prey model is a well-established benchmark in artificial intelligence and computation literature, making it an ideal example for validating simulation platforms. The simulator features a modular codebase written in both Rust and Python, enabling execution and integration of new environments and agent types. The HTTP-based

communication supports visualization in real time, while logging mechanisms track performance of the system over time. Among the primary challenges were designing a system that could generalize across different neuroevolutionary strategies, support modular experimentation, and provide clear visibility into agent dynamics and performance metrics. As the simulator matured, its architecture was iteratively redesigned to support a growing range of experimental needs.

DESIGN

Language

The choice of programming language is paramount in simulation, as it directly impacts performance, development efficiency, and the ability to model complex systems. High-performance languages are crucial for computationally intensive simulations requiring rapid execution, while languages with robust libraries and frameworks can significantly accelerate the development of intricate models and facilitate data analysis.

Python was selected for the simulation core because of its mature AI and data-science ecosystem, and the experience in the research group [11, 12], which accelerates the development. The NumPy library was used to handle vectorized physics calculations, while Matplotlib streamlined visual inspection. Most critically, the project leverages neat-python [13], which provides a well-tested implementation of NEAT. Python's object-oriented features support modular architecture, letting environments, agents and evolutionary operators remain loosely coupled yet internally cohesive. To offset Python's slower interpreter speed, performance-sensitive sections were profiled and refactored. The starting conditions for the environment, hyperparameters, population data, and agents' networks are logged for full reproducibility.

For the graphical front-end, we adopted Rust. Rust's ownership model guarantees memory safety without a garbage collector and enables predictable, low-latency execution, which are qualities essential for a real-time GUI that must render thousands of agents while remaining responsive to user input. Additionally, the interface layer of Rust offers a contained opportunity to gain proficiency. The Iced library [14] was chosen since it offers a modern, cross-platform widget set, and an update loop that fits Rust's asynchronous ecosystem.

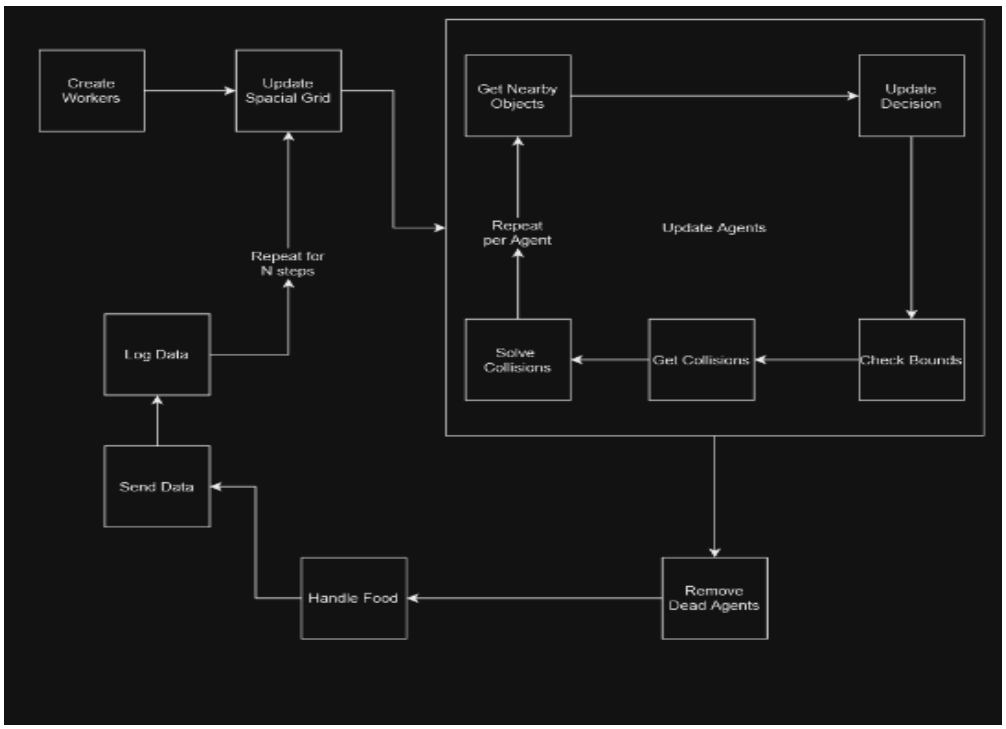

Figure 1. Control Flow of Simulation Step

Architecture

The presented software consists of two primary components: a simulation environment developed in Python, and a graphical user interface (GUI) implemented in Rust. These components operate independently, allowing them to be hosted on separate computational resources and providing the flexibility to execute the simulation without graphical interaction (headless operation).

The simulation itself comprises four main modules: server.py, environment.py, agent.py, and hyperneat.py. The server.py module serves as the central controller, managing initialization, execution flow, and the streaming of simulation data to the GUI. Upon execution, the server module instantiates the environment object defined in environment.py. This module encapsulates all core simulation logic, environment states, and logging processes. Within the environment, multiple agent instances are created as defined by the parameters of the experiment. These agents, encapsulated in the agent.py module, contain methods and functionalities necessary for interaction within the simulated environment. The hyperneat.py module specifically supports the evolutionary breeding stage, hosting the Compositional Pattern-Producing Networks (CPPNs) that generate neural networks for HyperNEAT agents.

The simulation follows a systematic execution loop as shown in Figure 1. First, initial user-defined parameters are read into memory, after which the server instantiates the population of agents and the environment. The environment then executes the primary simulation loop. Following each simulation epoch, the agent population undergoes an evolutionary breeding phase managed by the HyperNEAT module. This iterative process continues until a predefined number of epochs have been completed.

The GUI (see Figure 2) component receives, decodes, and visually represents simulation data in real-time. Data exchange between the simulation and GUI is structured using the JSON format, where individual simulation objects correspond to discrete JSON sections. Structurally, the GUI software comprises a main controller class, a web client class, and a view class. The main class provides entry and oversees the overall GUI operation. The web client class subscribes to the simulation data stream, receiving continuous updates, while the view class processes incoming data, updating the graphical representations accordingly.

The GUI is intentionally designed to function independently and remotely from the core simulation engine. This decoupled architecture enables headless simulation execution on high-performance computing (HPC) infrastructure, enhancing flexibility, scalability, and overall resilience in conducting computationally intensive experiments.

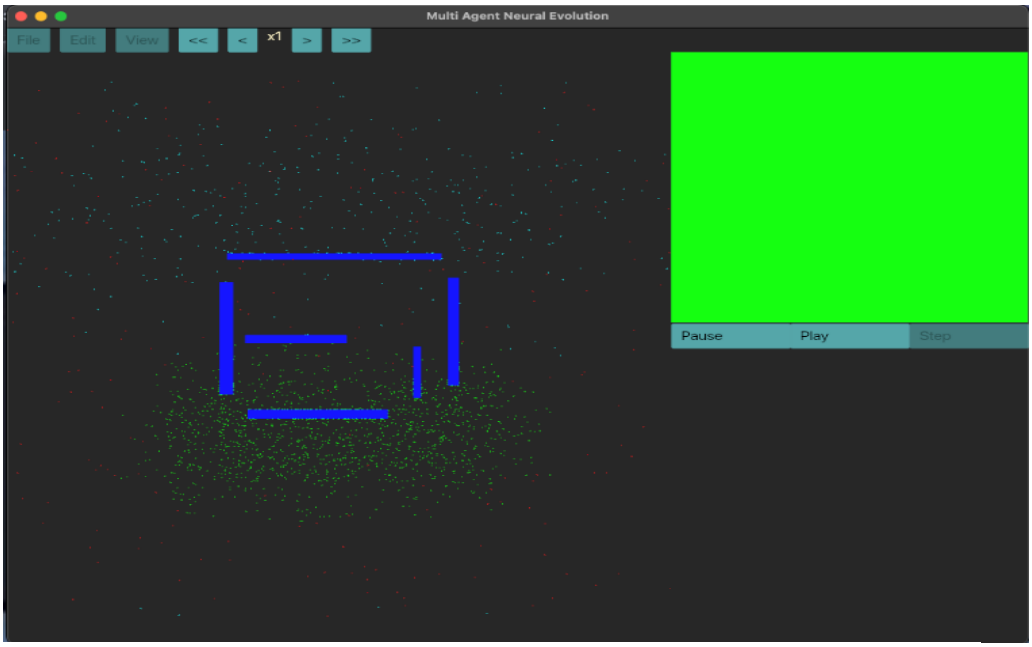

Figure 2. GUI (Green: Prey, Blue: Predators, Red: Food)

Remote Support

Communication between Python and Rust occurs over a lightweight HTTP and WebSocket bridge. The simulation core serializes agent states to JSON once per step; the GUI consumes the stream, reconstructs entity positions, and draws them. Control messages flow in the opposite direction, allowing pause–resume commands, and environment resets. This separation keeps the performance-critical visualization thread independent from the Python GIL and permits headless batch runs when visual output is unnecessary.

Logging

GENeSIS is designed to log extensively and in a way that is user configurable. By default, GENeSIS logs both population wide and individual agent metrics. The logging is designed to keep a balance between logging, speed, and memory footprint. Included in the logging is a viewer script that is used for analysis of individual agents' neural networks, an example of which is shown in Figure 3. This visualization feature is critical to observe the neural networks evolved over time.

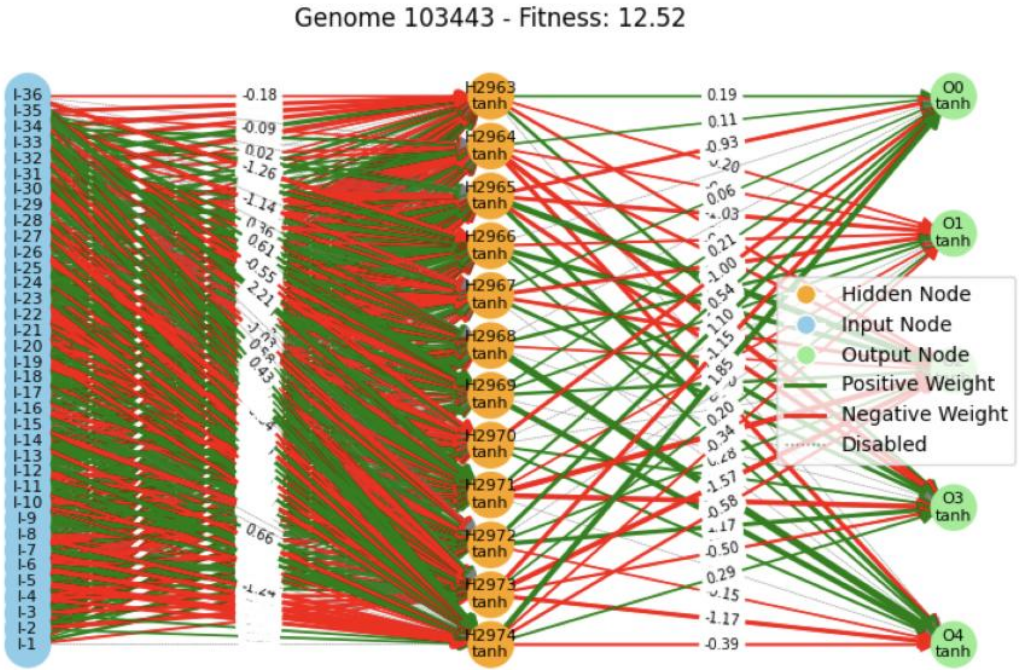

Figure 3. Visualization of an Agent Network

EXPERIMENTAL EVALUATION

GENeSIS was evaluated in a predator-prey simulation environment which was designed to assess its support for evolving multiple agent types using different neuroevolution strategies. This scenario was chosen because it allows for testing of multi-agent coordination, adaptation of behaviors in response to changes in the environment and accompanying agents, as well as allow for complex behaviors to emerge in agents such as swarming, herding, and cooperative hunting.

The use of a predator-prey simulation environment displays complex behaviors that can also be transferred to other more advanced environments in the future. While the current use case focused on 2D predator-prey dynamics, this framework generalizes well to other multi-agent environments, including simulated UAV coordination tasks, where navigation and obstacle avoidance offer similar challenges in a more complex spatial domain, allowing for these models to really prove advantageous.

To demonstrate GENeSIS's effectiveness, we evaluated three agent-generation strategies—genetic algorithms (referred to as STD), NEAT, and our HyperNEAT phenome generator—on a predator–prey task. All experiments used identical world bounds, population size, (850 total, 638 prey, 212 predators), fitness-functions, and sensory inputs (36-dimensional vectors). Each simulation ran between 500-650 epochs. To assess the flexibility of GENeSIS the simulation environment was changed in-between simulations. This included changing spawn location, adding/removing world obstacles, and changing the amount of food within the simulation. While this limits comparisons between simulation, it highlights how each training paradigm handles different environments.

Table 1. Showing Details About Each Simulation

Simulation	Models	Epochs	Time to complete
1	NEAT, STD	500	~5 hrs
2	NEAT, STD, HyperNEAT	650	~8 hrs

Table 1 provides more details on the two simulations run for this test. The first simulation was run without HyperNEAT to allow more direct comparison between NEAT and a normal genetic algorithm approach. In our testing we found that HyperNEAT would often outperform the other two models and make comparisons between them difficult. HyperNEAT would also increase (≈ 18%) the compute time for any given simulation.

Table 2. Showing Details About Fitness

Simulation	Models	Max Fitness	Epochs to 80%
1	NEAT Pred	8.7822994	283
1	NEAT Prey	735.99775	111
1	STD Pred	20.389595	384
1	STD Prey	782.41267	190
2	NEAT Pred	195.82886	502
2	NEAT Prey	23.451394	42
2	STD Pred	181.68922	494
2	STD Prey	23.701263	15
2	HyperNEAT Pred	391.67478	535
2	HyperNEAT Prey	38.025	25

In Table 2, we compare the maximum fitness achieved, and the time taken to reach 80% of that value. Our findings match the expected behavior of NEAT and HyperNEAT. HyperNEAT and NEAT performed better than their NEAT and baseline counter parts. However, they also took longer to achieve their maximum performance and required more compute time.

While the HyperNEAT-encoded predators achieved the highest peak fitness (≈391.7), HyperNEAT prey lagged both NEAT and the baseline, reaching only ≈38.0 versus ≈23.7 (baseline) and 23.45 (NEAT). This inverted trend suggests that the indirect CPPN encoding may be over-parameterized relative to the simpler prey foraging task: the richer topology gives predators more nuanced pursuit behaviors, but it injects unnecessary complexity for prey's narrower objective. We hypothesize that (a) the CPPN's mutation-only evolution without topology adjustment inhibits rapid adaptation on low-dimensional tasks, and (b) sensor-output mapping for prey may require a different substrate layout or activation regime.

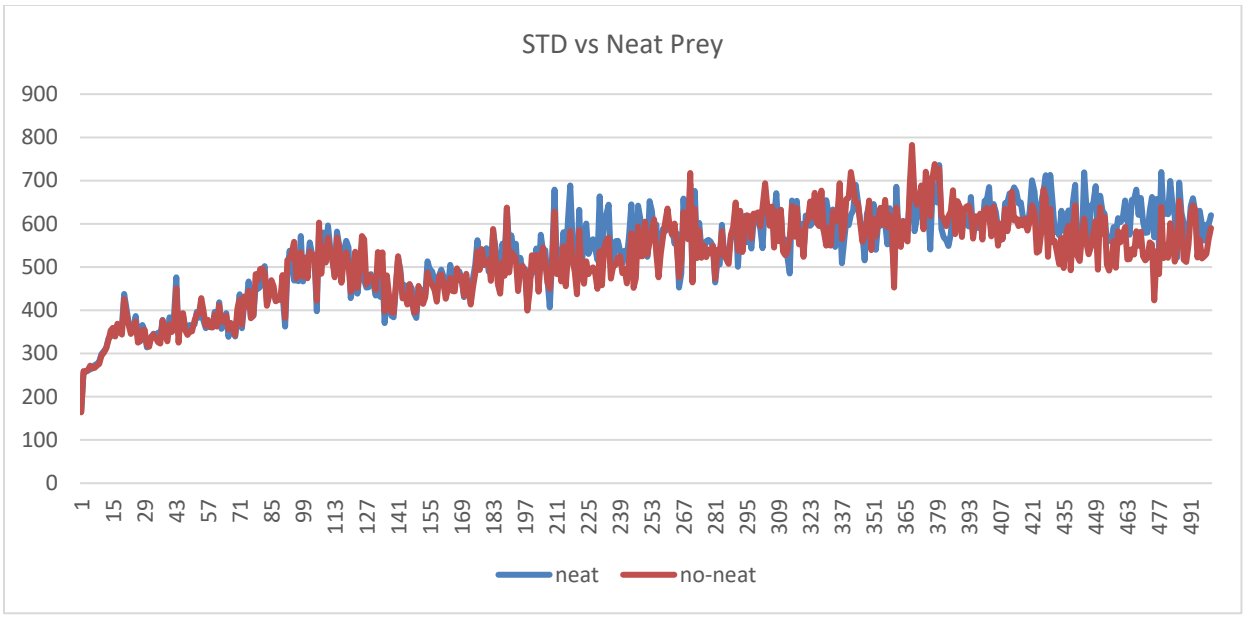

Figure 4. Average Fitness for Each Generation for Prey

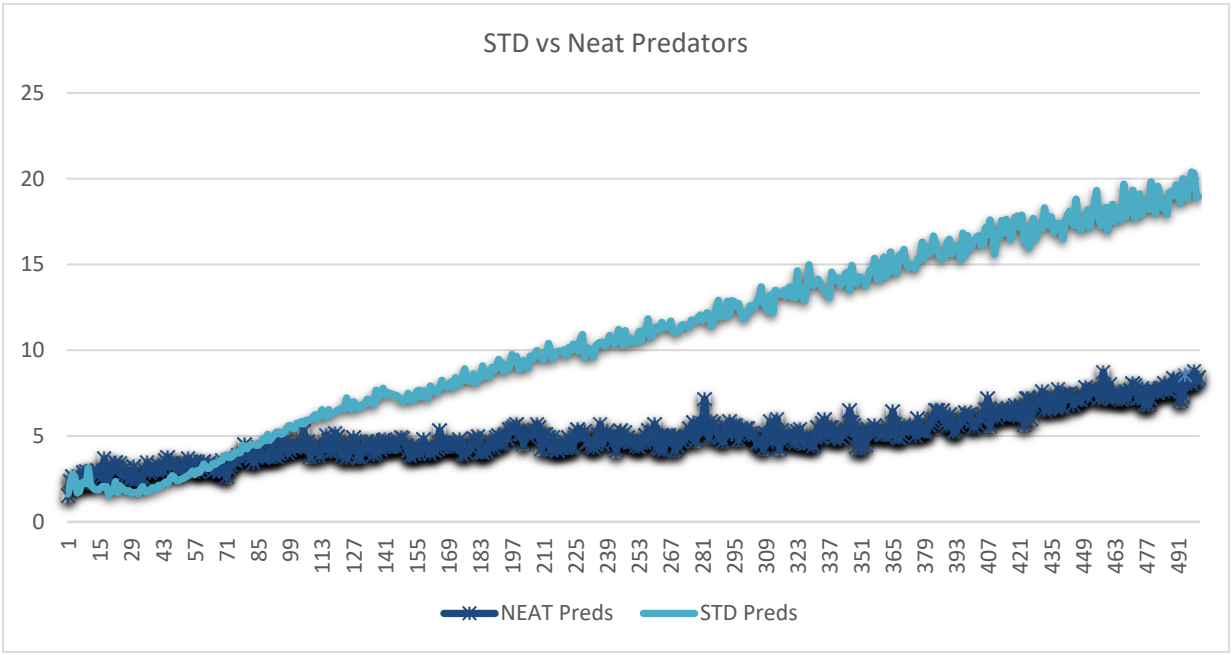

Figure 5. Average Fitness for Each Generation

In Figures 4 and 5 we compare NEAT vs baseline for predators and prey. In this simulation, the environment was mostly open, and food was plentiful. This effectively made it easier for both predators and prey to achieve high fitness scores. We can see that for both predators and prey the baseline performs better than the NEAT algorithm. This is particularly apparent for predators where the baseline is significantly ahead of NEAT by epoch 490 (≈20 vs ≈9). We attribute this to the relatively simple environmental conditions that favors the simpler training approach in the baseline algorithm. We believe that both prey algorithms effectively reached near the max possible fitness score for this simulation. (Fitness score is gained over time, so there is a max possible for each simulation). The NEAT algorithm produced significantly smaller models for both predator and prey. Therefore, the NEAT prey algorithm created a more computationally efficient model.

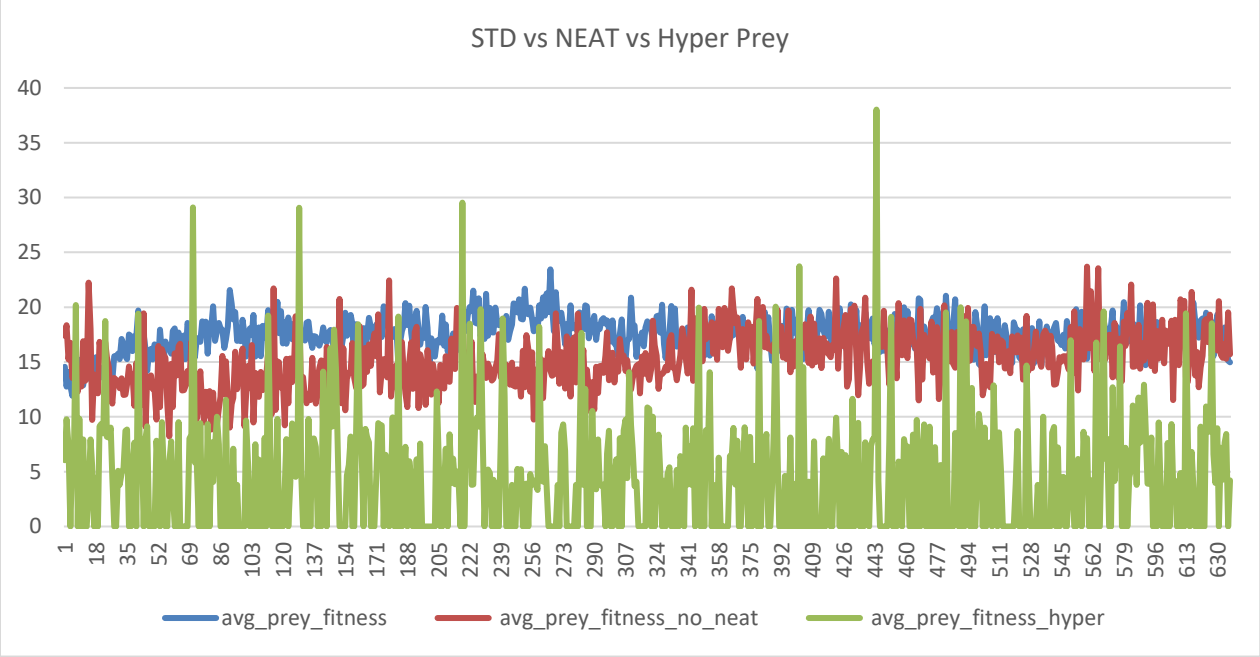

Figure 6. Average Fitness for Each Generation with HyperNEAT

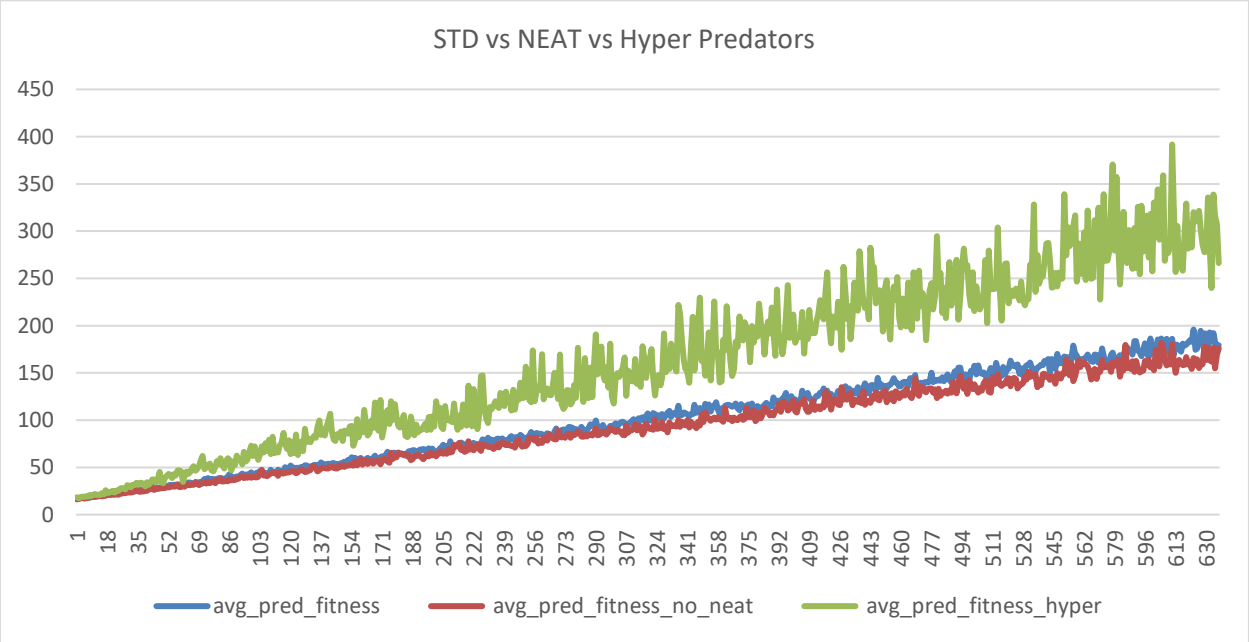

Figure 7. Average Fitness for Each Generation with HyperNEAT

In Figures 6 and 7 we compare NEAT vs HyperNEAT vs baseline for both predators and prey. In this simulation environment food supply was reduced, additional obstacles were added, and the length of each epoch was slightly increased. All other parameters remained unchanged from the previous simulation. Immediately apparent is the significant reduction in prey fitness across all models. We attribute this to the introduction of HyperNEAT predators, and the increase in environmental difficulty. On the predator side HyperNEAT quickly jumps out as the dominant model, reaching over double the score of the next best model (≈391 vs ≈195). This aligns with the findings from the

earlier research where it was shown that HyperNEAT outperforms its NEAT counterpart [3]. However, HyperNEAT generally struggled with creating good prey models. This behavior was repeated over multiple runs of the same simulation. We attribute this behavior to the simplistic nature of the problem combined with the pressure from the predator agents. This likely overwhelmed the training algorithm.

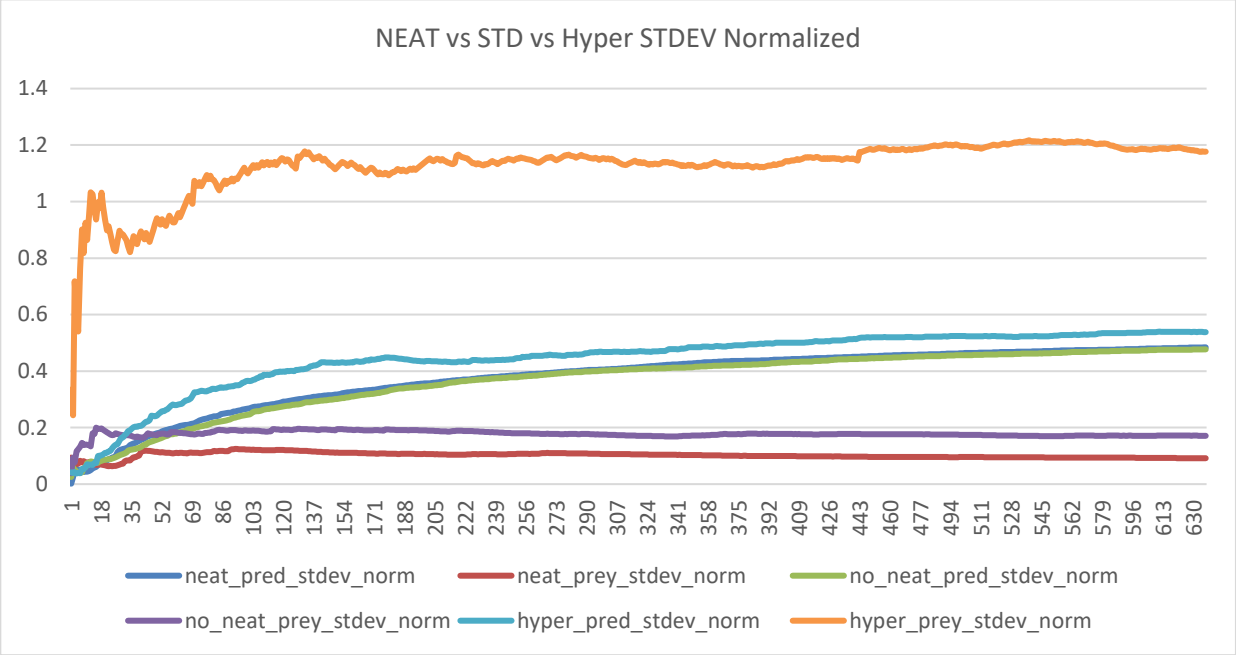

Figure 8: Standard Deviation normalized over time.

Figure 8 compares the normalized standard deviation for all models from simulation two. Standard deviation was calculated based on all previous data, so for epoch 100 it includes 1-99, then this was normalized by dividing it by the average fitness at that point. Generally, we expect an ideal curve to follow a logarithmic pattern, where the deviation rises quickly as the model learns, the levels off at some fixed point as the model approaches its peak. We can see how the HyperNEAT prey line is a large outlier here. This coincides with the poor performance within its class. On the other hand, we can see how all the predator models stay close to each other throughout the simulation.

Findings

We successfully instantiated fixed-topology GA agents, NEAT-evolved agents, and HyperNEAT agents within a single simulation run, each governed by the same world model, spawn mechanics, and sensing pipeline. HyperNEAT encoded predators achieved the highest peak fitness but required more time and computation to converge compared to NEAT and fixed-topology baselines. This shows the advantages of HyperNEAT in spatially structured tasks but also revealed some drawbacks when applied to simpler roles like prey. Environmental difficulty also seemed to amplify the differences in the model. With increased pressure from features like reduced food and increased obstacles, predator fitness became more discriminative of model quality. Predators using HyperNEAT excelled under this increased pressure, while prey models declined, demonstrating that prey roles may be more sensitive to environmental complexity and agent pressure. For these environmental conditions, NEAT evolved smaller networks than the baseline or HyperNEAT, particularly in prey. While baseline models sometimes achieved higher scores for simple conditions, NEAT offered better computational efficiency and maintainability. This seems to scale upwards with higher complexity environments, which NEAT and HyperNEAT prove advantageous for. These findings highlight the strengths and limitations of indirect encodings like HyperNEAT in multi-agent environments. GENeSIS enabled side-by-side evaluation of different evolutionary strategies and provided insights through visual and quantitative feedback.

LIMITATIONS

GENeSIS has several constraints that serve as the directions for future work.

- At present, the evolutionary back end supports only canonical genetic algorithms, NEAT, and HyperNEAT. Techniques such as novelty search, quality-diversity, or reinforcement-learning hybrids are absent. Consequently, performance claims cannot be generalized to methods that rely on gradient information, surrogate modeling, or dynamic reward shaping.
- Runs cannot be paused and resumed because the environment state, random seeds, and agent genomes are not serialized mid-simulation. Long jobs therefore require uninterrupted compute availability; any hardware failure forces full reruns and may bias aggregate statistics toward shorter episodes that happened to be completed.
- Validation is confined to a predator–prey benchmark and a limited hyperparameter sweep. No cross-validation has been performed against external benchmarks such as OpenAI Gym Retro or the MASON multi-agent suite. Effect sizes reported here should be viewed as indicative rather than definitive until broader replication is performed.
- The project relies on Neat-Python with a modified implementation. That library restricts genome size to 64 k nodes and expects fully connected initial topologies. These constraints may hinder scaling to very large sensory inputs or highly sparse starting graphs.
- Runtime data are broadcast as HTTP payloads over WebSockets without encryption or compression. Although this choice simplifies debugging and works well on a trusted LAN, it is unsuitable for sensitive experiments across untrusted networks. Metric streams can also saturate slow network links, introducing latency in the visualizer and potential packet loss in wide-area deployments.

The planned development activities will address several of these issues by adding checkpointing, TLS support, alternative evolutionary operators, and continuous integration across multiple platforms.

CONCLUSION

This paper presented an advanced simulation framework designed to facilitate research in artificial intelligence, particularly within neuroevolution and multi-agent systems. The proposed system addresses key limitations of existing simulation tools by offering enhanced flexibility, scalability, and reproducibility, enabling researchers to efficiently explore complex behaviors and interactions among multiple autonomous agents. Through practical demonstrations, such as the predator-prey scenario, the framework's effectiveness in rapidly implementing diverse models and optimizing agent behaviors was confirmed. Additionally, features such as remote high-performance computing capabilities, real-time monitoring, and multi-level data analysis significantly improve researchers' ability to evaluate performance and interpret experimental results. Ultimately, this work contributes valuable tools and methods to the broader scientific community, supporting continued advances in artificial intelligence research across industries such as aerospace, defense, robotics, and autonomous systems.

FUTURE WORK

GENeSIS is under active development and features and functionality will continuously be added in the future. Future development will focus on expanding robustness, algorithmic breadth, and external interoperability. Check-pointing, increased fault tolerance, GPU acceleration, and gradient based learning are all either planned or currently in progress features. The development team welcomes external input on bug fixes, new agent modules, physics plug-ins, alternative evolutionary operators, and forked projects. Community discussions in the repository’s forum channel will help priorities features such as checkpointing, GPU acceleration, ensuring that the framework evolves in step with the broader needs of the neuroevolution and swarm-autonomy research communities.

APPENDIX

All requirements, usage, and documentation can be found in the open-source repository: https://github.com/Multi-Agent-Neuroevolution/NeuroEvolutionTestbench.

REFERENCES

[1] Akinci, T.C., Topsakal, O., Akbas, M.I. (2024). Machine Learning Methods from Shallow Learning to Deep Learning. In: Ertuğrul, Ö.F., Guerrero, J.M., Yilmaz, M. (eds) *Shallow Learning vs. Deep Learning. The Springer Series in Applied Machine Learning*. Springer, Cham. https://doi.org/10.1007/978-3-031-69499-8_1

[2] Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, *10*(2), 99–127. https://doi.org/10.1162/106365602320169811

[3] Stanley, K. O., D'Ambrosio, D. B., & Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2), 185-212.

[4] Tucker Balch. TeamBots simulation and real robot execution environment, 1997. https://www.cs.cmu.edu/~trb/TeamBots/.

[5] Nelson Minar, Roger Burkhart, Christopher Langton, and Manar Askenazi. The swarm simulation system., 1996. http://www.swarm.org.

[6] North, Michael J, Nicholson T Collier, Jonathan Ozik, Eric R Tatara, Charles M Macal, Mark Bragen, and Pam Sydelko. 2013. “Complex Adaptive Systems Modeling with Repast Simphony.” Complex Adaptive Systems Modeling 1 (1): 3. https://doi.org/10.1186/2194-3206-1-3.

[7] Kopman, S., Akbas, M. I. & Turgut, D. "EpidemicSim: Epidemic simulation system with realistic mobility," In IEEE Conference on Local Computer Networks - Workshops, Clearwater, FL, USA, 2012, pp. 659-665, doi: 10.1109/LCNW.2012.6424047.

[8] Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., & Balan, G. (2005). MASON: A multiagent simulation environment. *Simulation*, 81, 517–527.

[9] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. *ArXiv, abs/1606.01540*. https://arxiv.org/abs/1606.01540#

[10] Akbas, M. I., Long, C. A., Hanumanthu, S. K., Anderson, E., & Razdan, R. (2019, April). “FPolyOS: A simulation platform to explore breakthrough concepts in intelligent transportation.” In 2019 SoutheastCon, Huntsville, AL, USA, 2019, pp. 1-6, doi: 10.1109/SoutheastCon42311.2019.9020382.

[11] Goss, Q and Akbas, M. I., "Integration of Formal Specification and Traffic Simulation for Scenario-Based Validation ." In the IEEE International Conference on Mobility: Operations, Services, and Technologies (IEEE MOST), Detroit, MI, USA, pp. 213-222, doi: 10.1109/MOST57249.2023.00030, May 2023.

[12] Gonzalez Nunez, J. A. and Akbas. M. I., "Demo: Agent-Based Crowdsensing Simulation for Urban Meteorological Data Collection and Hybrid Aerial-Terrestrial Route Determination," In the IEEE Conference on Local Computer Networks (IEEE LCN), Daytona Beach, Florida, doi: 10.1109/LCN58197.2023.10223330, 2023.

[13] McIntyre, M. & Collier, M. (2025). NEAT-Python (v0.92) [Computer software]. Available from https://github.com/CodeReclaim/neat-python

[14] Iced-rs Contributors. (2025). *iced: A cross-platform GUI library for Rust* (v0.12.1) [Computer software]. Available from https://github.com/iced-rs/iced