УДК 004.8 DOI https://doi.org/10.32782/1995-0519.2025.1.16

A NEW APPROACH TO CODE GENERATION FOR EFFICIENT NEURAL COMPUTATION IN ACCESSIBLE ARTIFICIAL ECOSYSTEMS

Mykhailo Zachepylo
Postgraduate Student at the Department of Information Systems named by V. O. Kravets
National Technical University "Kharkiv Polytechnic Institute", 2 Kyrpychova str., Kharkiv, Ukraine, 61002, Mykhailo.Zachepylo@cit.khpi.edu.ua
ORCID: 0000-0001-6410-5934

Oleksandr Yushchenko
Candidate of Physical and Mathematical Sciences, Professor,
Professor at the Department of Information Systems named by V. O. Kravets,
National Technical University "Kharkiv Polytechnic Institute", 2 Kyrpychova str., Kharkiv, Ukraine, 61002, agyu@kpi.kharkov.ua
ORCID: 0000-0002-0078-3450

The purpose of this study is to address the critical computational challenges faced in Artificial Life (ALife) research, particularly in achieving scalable simulations of large populations of agents with evolving and heterogeneous neural networks. These simulations are essential for investigating complex phenomena such as adaptation, self-organization, and the emergence of intelligence (noogenesis) in artificial ecosystems. However, the computational overhead associated with evolving neural architectures, especially in open-ended environments with high variability, remains a significant barrier. The methodology proposed in this study employs a dynamic code generation approach that leverages the V8 JavaScript engine for Just-In-Time (JIT) compilation. This approach dynamically transforms neural network topologies into computational graphs, enabling the efficient generation of minimal instruction sets for activation functions, which are then compiled and optimized at runtime. By focusing on active subgraphs and pruning unnecessary computations, this method significantly reduces the overhead typically incurred by interpreting neural networks in each simulation step.

The findings demonstrate that the proposed approach outperforms traditional static interpretation methods, as exemplified by the Neataptic library. Benchmarking experiments conducted on an M1 Pro processor showed a 23 % reduction in timestep processing time, with the capability to handle up to thousands of agents efficiently. This scalability ensures the feasibility of running long-term ALife simulations, preserving high biological fidelity while reducing computational costs.

The originality of this work lies in its novel application of dynamic code generation to evolving neural networks in ALife models, an area traditionally dominated by frameworks optimized for static neural topologies or batch-based operations. By demonstrating that JavaScript, a non-traditional language for high-performance simulations, can achieve competitive efficiency through runtime optimization, this research broadens the accessibility of ALife simulations to a broader audience, including those focusing on educational or entertainment applications.

The practical value of this approach is its potential to enable more extensive and detailed investigations into emergent intelligence and behavior in artificial ecosystems. By overcoming computational limitations, the framework supports long-term studies of noogenesis, offering new insights into how adaptive intelligence and novel behaviors develop in evolving populations. In conclusion, this study presents a scalable, efficient, and flexible solution to the computational challenges of ALife simulations, forming a foundation for future research. Promising directions for further development include distributed simulation architectures and hybrid GPU-accelerated approaches to enhance scalability and performance.

Key words: artificial life, neural networks, neuroevolution, performance optimization, dynamic code generation.

Introduction. Artificial Life (ALife) research aims to deepen our understanding of biological and cognitive processes by synthesizing them in computational environments. Since its inception, ALife has spanned a range of models, from discrete cellular automata to sophisticated continuous-space simulations [1], with a key focus on exploring how complex phenomena such as adaptation, self-organization, and intelligence can emerge in virtual populations.

The principal scientific problem is developing a new approach for efficient neural computations to achieve large-scale ALife simulations that simultaneously accommodate numerous agents, each with a unique and evolving neural architecture, without incurring prohibitive computational overhead. This challenge becomes significantly pronounced when considering open-ended evolution and high-frequency data logging, both pivotal for studying emergent cognitive and behavioral

traits, making such Alife simulations accessible for education and entertainment without specific hardware requirements.

Several computational paradigms have attempted to resolve these performance bottlenecks. Traditional statically compiled languages offer high raw speed but can introduce significant overhead when agent architectures change frequently. Conversely, dynamic and JIT-compiled environments can handle heterogeneous workloads more flexibly, yet they may suffer from throughput issues when agent populations scale. Additionally, the underlying spatial representation, ranging from simple grids to fully 3D continuous spaces, affects collision detection, movement rules, and perception modeling complexity. Identifying strategies that integrate large populations, evolving topologies, and complex spatial environments in a way that balances computational efficiency with biological fidelity remains nontrivial.

The **actuality** of addressing this issue is underscored by the growing emphasis on investigating noogenesis – intelligence emerging through long temporal horizons and large population sizes. Inadequate scalability can truncate simulations, omitting critical evolutionary transitions and interactions. By developing and refining computational strategies that allow for more extensive and prolonged yet accessible runs, the ALife community stands to make significant progress in uncovering how adaptive intelligence and novel behaviors materialize in artificial ecosystems.

Literature review. ALife research spans decades of efforts to create and explore artificial ecosystems where emergent phenomena, such as evolution, adaptation, and intelligence, can be studied under controlled conditions. Various models, methodologies, and computational tools have been proposed, each contributing to different facets of the field.

Genetic algorithms (GAs), which simulate the process of natural selection, provide a foundational approach to studying evolutionary dynamics in artificial systems[2]. GAs have proven effective for optimizing static problems and demonstrating basic evolutionary mechanisms. However, their focus on fixed, fitness-maximizing objectives limits their application to open-ended evolutionary systems where emergent complexity and adaptive intelligence are of interest. Expanding on this foundation, models of neural evolution, such as NEAT, introduced methods for evolving neural network architectures alongside weights, allowing agents to develop more complex behaviors over time [3].

Efforts to capture richer, biologically inspired dynamics have led to the development of spatially explicit ALife models. Early systems, such as cellular automata, were designed for simplicity and computational efficiency, relying on grid-based environments to facilitate discrete agent interactions. While these models help explore self-organization and adaptation, they inherently lack the complexity of continuous-space environments. PolyWorld, a seminal 3D ALife system, bridged this gap by incorporating a non-grid environment, agent vision, and biologically inspired rules governing neural processing and behavior [4]. PolyWorld demonstrated that higher-dimensional environments could foster richer emergent phenomena, such as predator-prey dynamics and cooperative behavior. However, the computational overhead associated with continuous 3D physics and collision detection has historically limited its scalability to smaller populations.

The computational bottlenecks associated with large-scale simulations have inspired researchers to explore hardware acceleration and optimized frameworks. TensorFlow [5] and JAX [6], both widely used in deep learning, provide GPU-accelerated tensor operations well-suited for training large, homogeneous neural networks. Recent adaptations of these frameworks, such as tensorized NEAT [7], attempt to extend their utility to evolving populations of neural networks. While these efforts show promise, their reliance on batch operations and static population makes it impractical for a dynamic population where networks could be added and removed in the population in each simulation step. GPU-based methods often struggle with the irregular memory access and branching logic inherent in these systems, leading to diminishing performance gains [8].

An alternative to tensor-based frameworks lies in the use of Just-In-Time (JIT) compilation, as exemplified by the V8 JavaScript engine [9], Python JAX. Unlike statically compiled languages, JIT-compiled environments can dynamically optimize function execution at runtime, making them particularly well-suited for workloads with frequent topology mutations and highly varied agent behaviors. JIT-compiled languages have shown potential for efficiently managing the heterogeneity of ALife models. Several studies advanced in JIT compilation for specific hardware [10, 11]. Still, they focused on layered architectures and large scales, making them inefficient for many neural networks built from evolved topologies.

Despite these advances, the field lacks a unified framework for efficiently simulating large-

scale populations of agents with evolving neural networks, especially in open-ended environments. The existing literature demonstrates that while higher-dimensional spaces and richer dynamics enable more nuanced behaviors, they impose prohibitive computational demands. Similarly, while frameworks like TensorFlow, JAX, and static languages provide strong baselines for performance, they fail to address the unique challenges posed by heterogeneity and dynamic evolution. This gap underscores the need for novel optimization strategies that balance computational efficiency, flexibility, and environmental richness, forming the basis of the present study

Methods and results. We evaluate the efficiency of neural network simulations using the ALife model as the base [1]. It uses modified NEAT to evolve network topology and weights with the physical body of creatures, and neural networks have sensory inputs, motor outputs, and processing units with possible recurrent connections. In this model, primitive creatures can evolve via splitting and mutation when energy is enough by consuming food resources, and obstacles are omitted in a 2D planar environment. The use of JavaScript as the primary programming language for this study warrants explanation. While JavaScript is not traditionally associated with high-performance computing or neural network simulations, it offers several advantages that align with the goals of this. JavaScript's widespread adoption, ease of deployment via web-based interfaces, and integration with modern JIT compilation engines like V8 make it a uniquely accessible choice for educational and entertainment-focused applications without requiring specific hardware. By leveraging the V8 engine's dynamic optimization capabilities, we demonstrate that JavaScript can achieve competitive performance for simulating evolving neural networks in large-scale ALife environments despite its perceived limitations.

We benchmarked two approaches: (1) a baseline approach using the Neataptic library [12], a JavaScript library for neural network computation with free topologies, and (2) a custom JIT compilation approach developed explicitly for dynamically evolving neural networks. Other JavaScript-based neural network libraries, such as Synaptic.js, Brain.js, and TensorFlow.js, were deemed unsuitable due to their reliance on fixed topologies or batch-based operations, making them incompatible with the mutable, heterogeneous networks central to this study. The baseline approach computes network activations using a fixed interpretation model provided by Neataptic.

To accommodate the rapid turnover of up to 1,000 neural networks in our simulation, we employ a just-in-time (JIT) compilation strategy using JavaScript's V8 JIT engine. This approach allows us to efficiently calculate network activations while minimizing the overhead of interpreting neural computations repeatedly. The steps for JIT-compiling a neural network are as follows: **1) Parsing the Genotype.** Each agent's neural network is encoded as a genotype representing a directed graph. Nodes correspond to neurons, while directed edges represent synaptic connections with associated weights. **2) Graph Traversal.** The genotype is parsed into a directed graph, and a reversed breadth-first search (R-BFS) is initiated from the output nodes. This traversal identifies the active subset of the network and prunes disconnected nodes that do not reach the output. **3) Instruction Set Generation.** An instruction set is dynamically generated during traversal to calculate each node's activation. Instructions are based on stack inputs, weights, and activation functions, ensuring that only necessary computations are included. **4) Topological Ordering.** The computational graph is ordered based on the farthest distance from the output nodes, ensuring that intermediate activations are computed in the correct sequence. **5) Dynamic Function Creation.** The instruction set is reversed to produce a computation sequence from inputs to outputs. A JavaScript function body is constructed dynamically using the "Function()" constructor, representing the minimal computation required for the network. **6) Compilation and Optimization.** The dynamically created function is executed repeatedly during the simulation, allowing the V8 JIT engine to optimize its performance over time.

This approach leverages JavaScript's ability to dynamically compile and optimize short-lived functions, making it well-suited to the high variability of neural networks in an evolutionary simulation. By focusing on active subgraphs and minimizing computation, the JIT-compiled approach reduces overhead compared to interpreting the network for every activation. The source code is available on GitHub [13]

We implemented a controlled benchmarking setup to evaluate the performance of our JIT-compiled code generation approach. The simulation used a snapshot of evolved 927 agents from simulation on $t = 4 \cdot 10^6$ steps, and tracked their evolution over Δt = 5,000 timesteps, repeating 5 times. Performance

metrics included Average computation time per simulation step standard deviation.

The performance of the two simulation approaches – JIT-compiled neural networks (proposed) and Neataptic-based baseline – was evaluated from a single simulation snapshot for 5,000 timesteps 5 times each. Snapshot contained 927 agents at the start with. Experiments were conducted on the M1 Pro processor. Table 1 summarizes the mean and standard deviation of the neural networks' processing time per timestep and load time (initialization).

Table 1

Performance Comparison of Simulation Approaches

Approach	Mean Timestep time (ms)	Mean load time (ms)
Neataptic (Baseline)	7.165 ± 3.39	0.22 ± 0.23
JIT-Compiled (Proposed)	5.506 ± 3.15	0.30 ± 0.18

The proposed JIT-compiled approach demonstrates significantly lower mean processing times by 23 % compared to the baseline implementation via Neataptic with the cost of neural network initiation (load). The efficiency gains are attributed to the dynamic compilation of neural network calculation instructions, which minimizes redundant computations. To validate efficiency outside the model simulation, we've randomly generated 1,000 feed-forward neural networks with custom topologies for each set of neurons count $N = \{10, 20, 30\}$ and connection density $DC = \{0.0, 0.005, \ldots, 0.4\}$. And measured the mean time of 1,000 samples for computing these 1,000 networks in each set combination. Fig. 1 depicts the results of this synthetic benchmark, where we can see the proposed approach is less effective on networks with high connectivity rates but wins on lower rates when measured in isolated synthetic conditions.

In our Alife simulation model, creatures evolved to have mean connections density is $D_C = 0.05$, calculated as:

$$D_C = \frac{C}{N^2}, \tag{1}$$

where N– number of neurons in the network;

C – number of neural connections.

Conclusions. This study addresses a critical challenge in Artificial Life (ALife) research: achieving scalable, large-scale simulations incorporating numerous agents with evolving, heterogeneous neural networks without prohibitive computational overhead. This problem is central to advancing our understanding of noogenesis, the emergence, and development of intelligence within virtual ecosystems, as it requires simulations capable

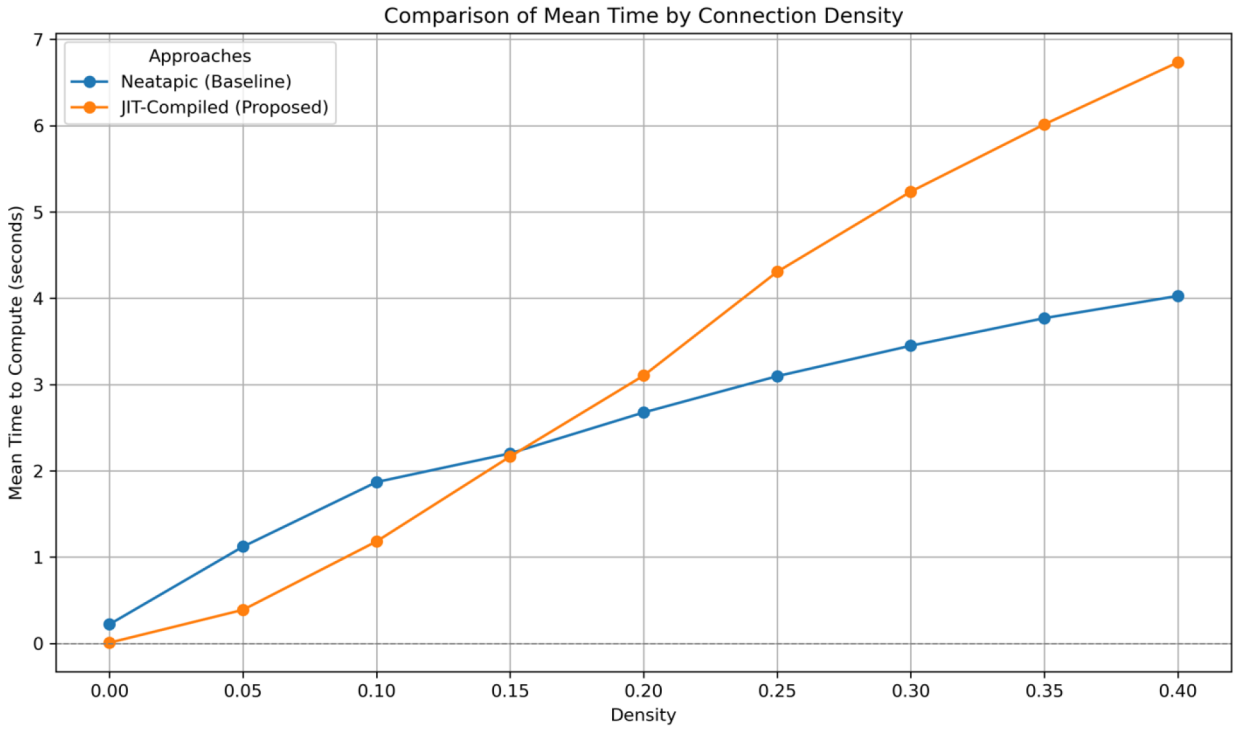

Fig. 1. Mean Time vs. Connection Density for Neataptic (Baseline) and JITCompiled (Proposed)

of running over long timescales with complex and dynamic agent populations. The inefficiencies of traditional methods and frameworks, which often fail to balance performance and flexibility, underscore the need for innovative approaches to ALife simulation.

Our proposed code generation approach leverages Just-In-Time (JIT) compilation using the V8 JavaScript engine to optimize the execution of neural network computations dynamically. By generating minimal instruction sets for each network and dynamically compiling them at runtime, we demonstrated significant performance gains over conventional approaches, such as the Neataptic library. Our results show that the JIT-compiled approach reduces timestep processing times by 23 % on a given benchmark compared to the baseline while maintaining scalability agent count and accommodating complex network topologies.

The limitations are found in connection density. In complex topologies with higher connection density, the proposed method displayed higher compute time, but in the Alife simulations with evolving neural networks, it is not common for creatures to evolve a high connectivity rate

The findings highlight several important insights. First, JIT compilation is particularly effective for simulations with frequent mutations to network topologies, where static computation models incur substantial overhead. The ability to dynamically compile and optimize short-lived neural activation functions allows the simulation to scale efficiently with population size and network complexity.

By enabling the efficient simulation of larger populations with more complex and dynamic neural networks, our approach lays the groundwork for deeper explorations of emergent intelligence in artificial ecosystems. Furthermore, the accessibility of JavaScript as a programming platform makes this approach particularly well-suited for educational and entertainment-focused applications, bridging the gap between high-performance research tools and broader usability.

Future research should focus on extending this approach in several directions. Solving the downgrade on higher connectivity rates by utilizing sub-function patterns for more efficient lining. Additionally, distributed simulation architectures, where parallel instances run across multiple machines, could enhance scalability. Finally, hybrid approaches incorporating GPU acceleration for batching subcomponents like sensory data processing while retaining JIT-based optimization for heterogeneous neural networks could provide an optimal balance of speed and flexibility.

In conclusion, the proposed JIT-compiled code generation approach demonstrates a practical and effective solution to the computational challenges of modeling noogenesis in virtual biocenoses. By bridging the gap between performance and flexibility, this approach enables more extensive and detailed ALife simulations, advancing our ability to study the emergence of intelligence in artificial ecosystems.

REFERENCES

1. Zachepylo, M., & Yushchenko, O. (2023). The scientific basis, some results, and perspectives of modeling evolutionarily conditioned noogenesis of artificial creatures in virtual biocenoses. *Bulletin of National Technical University "KhPI". Series: System Anal ysis, Control and Information Technologies*, 2 (10), 85–94. https://doi.org/10.20998/2079-0023.2023.02.13

2. Melanie, M. (1999). *An Introduction to Genetic Algorithm*. https://doi.org/https://doi.org/10.7551/mitpress/3927.001.0001

3. Stanley, K.O., & Miikkulainen, R. (2002). Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*, 10(2), 99–127. https://doi.org/10.1162/106365602320169811

4. Yaeger, L., (1994) Computational Genetics, Physiology, Metabolism, Neural Systems, Learning, Vision, and Behavior or PolyWorld: Life in a New Context. *In Santa Fe Institute Studies in the Sciences of Complexity-Proceedings*, vol. 17, pp. 263–288,

5. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., … Brain, G. (2016). TensorFlow: A System for Large-Scale Machine Learning. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI' 16).,* pp. 265–283. https://doi.org/10.5555/3026877.3026899

6. Bradbury, J., Frostig, R., Hawkins, P., Johnson, M.J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). *JAX: composable transformations of Python+NumPy programs*. Retrieved from: http://github.com/jax-ml/jax

7. Wang, L., Zhao, M., Liu, E., Sun, K., & Cheng, R. (2024). *Tensorized NeuroEvolution of Augmenting Topologies for GPU Acceleration. 16*, pp. 1156–1164. https://doi.org/10.1145/3638529.3654210

8. Che, S., Beckmann, B.M., Reinhardt S.K., and Skadron K. (2013) Pannotia: Understanding irregular GPGPU graph applications. *Proceedings – IEEE International Symposium on Workload Characterization, IISWC 2013*, pp.185–195, doi: https://doi.org/10.1109/IISWC.2013.6704684

9. Google. *Google V8*. Retrieved January 21, 2025, from https://v8.dev/

10. Thielke, F., & Hasselbring, A. (2019). A JIT Compiler for Neural Network Inference. *RoboCup 2019: Robot World Cup XXIII, 11531 LNAI*, 448–456. https://doi.org/10.1007/978-3-030-35699-6_36/TABLES/1

11. Hu, S. M., Liang, D., Yang, G. Y., Yang, G. W., & Zhou, W. Y. (2020). Jittor: a novel deep learning framework with meta-operators and unified graph execution. *Science China Information Sciences, 63*(12), pp. 1–21. https://doi.org/10.1007/S11432-020-3097-4/METRICS

12. *Neat;aptic: Blazing fast neuro-evolution & backpropagation for the browser and Node.js*. Retrieved January 25, 2025, from https://github.com/wagenaartje/neataptic

13. Zachepylo, M. (2025). *Neural Network computational graph generation for fast and scalable computation in Alife models in web*. Retrieved fromhttps://github.com/WorldThirteen/alife_neural_jit

НОВИЙ ПІДХІД ГЕНЕРАЦІЇ КОДУ ДЛЯ ЕФЕКТИВНОГО ОБРАХУВАННЯ НЕЙРОМЕРЕЖ У ДОСТУПНИХ ШТУЧНИХ

Михайло Зачепило
аспірант кафедри системи інформації імені В. О. Кравця
Національний технічний університет «Харківський політехнічний інститут»,
Кирпичова, 2, Харків, Україна, 61002, Mykhailo.Zachepylo@cit.khpi.edu.ua
ORCID: 0000-0001-6410-5934

Олександр Ющенко
кандидат фізико-математичних наук, професор, професор кафедри системи інформації імені В. О. Кравця
Національний технічний університет «Харківський політехнічний інститут»,
Кирпичова, 2, Харків, Україна, 61002, agyu@kpi.kharkov.ua
ORCID: 0000-0002-0078-3450

Метою цього дослідження є подолання критичних обчислювальних викликів, з якими стикається дослідження штучного життя (ALife), зокрема досягнення масштабованих симуляцій великих популяцій агентів із еволюціонувальними та гетерогенними нейромережами. Такі симуляції необхідні для вивчення складних феноменів, як-от адаптація, самоорганізація та виникнення інтелекту (ноогенез) в штучних екосистемах. Проте обчислювальне навантаження, пов'язане з еволюцією нейронних архітектур, особливо в умовах відкритих середовищ із високою варіативністю, залишається суттєвою перешкодою.

Методологія, запропонована в цьому дослідженні, використовує підхід динамічної генерації коду, що базується на використанні JavaScript-рушія V8 для Just-In-Time (JIT) компіляції. Цей підхід динамічно перетворює топології нейронних мереж в обчислювальні графи, що дає змогу ефективно генерувати мінімальні набори інструкцій для функцій активації, які потім компілюються та оптимізуються під час виконання. Зосереджуючись на активних підграфах й усуваючи непотрібні обчислення, цей метод значно знижує навантаження, яке зазвичай виникає при інтерпретації нейронних мереж на кожному кроці симуляції.

Результати показують, що запропонований підхід перевершує традиційні методи статичної інтерпретації, зокрема ті, які використовує бібліотека Neataptic. Тестові експерименти, проведені на процесорі M1 Pro, показали зменшення часу обробки одного кроку на 23 % з можливістю ефективно працювати з тисячами агентів. Ця масштабованість забезпечує можливість тривалого запуску симуляцій ALife, зберігаючи високу біологічну точність за одночасного зниження обчислювальних витрат.

Оригінальність цієї роботи полягає в її новаторському застосуванні динамічної генерації коду до еволюціонувальних нейромереж у моделях ALife, що традиційно домінувалися фреймворками, оптимізованими для статичних нейронних топологій або групових операцій. Демонструючи, що JavaScript, мова, яка традиційно не асоціюється з високопродуктивними симуляціями, може досягати конкурентної ефективності завдяки оптимізації в реальному часі, це дослідження розширює доступність симуляцій ALife для ширшої аудиторії, включно з тими, хто зосереджується на освітніх чи розважальних застосуваннях.

Практична цінність цього підходу полягає в його потенціалі дозволити більш масштабні та детальні дослідження інтелекту та поведінки, що виникає в штучних екосистемах. Подолання обчислювальних обмежень дає змогу використовувати цей фреймворк для тривалих досліджень ноогенезу, пропонуючи нові уявлення про те, як адаптивний інтелект і нові поведінкові моделі розвиваються в еволюціонувальних популяціях. У висновку це дослідження пропонує масштабоване, ефективне та гнучке рішення для обчислювальних викликів симуляцій ALife, що створює основу для майбутніх досліджень. Перспективні напрями подальшого розвитку передбачають розподілені архітектури симуляцій та гібридні GPU-прискорені підходи для покращення масштабованості й продуктивності.

Ключові слова: штучне життя, нейромережі, нейроеволюція, оптимізація продуктивності, динамічна генерація коду.

BIBLIOGRAPHY

1. Zachepylo M., Yushchenko O. The scientific basis, some results, and perspectives of modeling evolutionarily conditioned noogenesis of artificial creatures in virtual biocenoses. *Bulletin of National Technical University "KhPI". Series: System Analysis, Control and Information Technologies*. 2023. №2 (10). P. 85–94. DOI: https://doi.org/10.20998/2079-0023.2023.02.13

2. Melanie M. An Introduction to Genetic Algorithm. 1999. DOI: https://doi.org/10.7551/mitpress/3927.001.0001

3. Stanley K. O., Miikkulainen R. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*. 2002. Vol. 10, №2. P. 99–127. DOI: https://doi.org/10.1162/106365602320169811

4. Yaeger L. Computational Genetics, Physiology, Metabolism, Neural Systems, Learning, Vision, and Behavior or PolyWorld: Life in a New Context. *In Santa Fe Institute Studies in the Sciences of Complexity-Proceedings*. 1994. Vol. 17, P. 263–288.

5. Abadi M., Barham P., Chen J., Chen Z., Davis A., Dean J., Devin M., Ghemawat S., Irving G., Isard M., Kudlur M., Levenberg J., Monga R., Moore S., Murray D. G., Steiner B., Tucker P., Vasudevan V., Warden P., Brain G. TensorFlow: A System for Large-Scale Machine Learning. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI' 16.)*. 2016. P. 265–283. DOI: https://doi.org/10.5555/3026877.3026899

6. Bradbury J., Frostig R., Hawkins P., Johnson M. J., Leary C., Maclaurin D., Necula G., Paszke A., VanderPlas J., Wanderman-Milne S., Zhang Q. JAX: Composable transformations of Python+NumPy programs. 2018. URL: http://github.com/jax-ml/jax (Accessed at 21.01.2025).

7. Wang L., Zhao M. Liu E., Sun K., Cheng R. Tensorized NeuroEvolution of Augmenting Topologies for GPU Acceleration. *Proceedings of the 16th International Symposium*. 2024. P. 1156–1164. DOI: https://doi.org/10.1145/3638529.3654210

8. Che S. Beckmann B. M., Reinhardt S. K., and Skadron K. Pannotia: Understanding irregular GPGPU graph applications. *Proceedings – IEEE International Symposium on Workload Characterization, IISWC 2013*. 2013. P. 185–195. DOI: https://doi.org/10.1109/IISWC.2013.6704684

9. Google. *Google V8*. URL: https://v8.dev/ (Accessed at 21.01.2025)

10. Thielke F., Hasselbring A. A JIT Compiler for Neural Network Inference. *RoboCup 2019: Robot World Cup XXIII, 11531 LNAI*, 448–456. DOI: https://doi.org/10.1007/978-3-030-35699-6_36/TABLES/1

11. Hu S. M., Liang D., Yang G. Y., Yang G. W., Zhou W. Y. Jittor: a novel deep learning framework with meta-operators and unified graph execution. *Science China Information Sciences*. 2020. No. 63 (12), P. 1–21. DOI: https://doi.org/10.1007/S11432-020-3097-4/METRICS

12. Neataptic: Blazing fast neuro-evolution & backpropagation for the browser and Node.js. URL: https://github.com/wagenaartje/neataptic (Accessed at 21.01.2025).

13. Zachepylo M. Neural Network computational graph generation for fast and scalable computation in Alife models in web. 2025. URL: https://github.com/WorldThirteen/alife_neural_jit (Accessed at 27.01.2025).

Стаття надійшла 27.01.2025