EvoAtari: Benchmarking Bio-Inspired Agents across Atari Games

Tommaso Ballarini, Chiara Belli, Elisa Negrini
{tommaso.ballarini, chiara.belli, elisa.negrini}@studenti.unitn.it
University of Trento

***Abstract*—This study benchmarks evolutionary algorithms across three Atari 2600 games: Skiing, Freeway, and Space Invaders. High-dimensional inputs and sparse rewards challenges were addressed by leveraging OCAtari for object-centric RAM extraction and fitness shaping. Neuroevolution of Augmenting Topologies (NEAT) was compared against OpenEvolve, an LLM-driven framework evolving interpretable Python code. Results show NEAT excels in fine-grained temporal tasks (Freeway), while OpenEvolve offers a superior interpretability and generalization in complex scenarios (Skiing, Space Invaders). Findings highlight a strategic trade-off between "black-box" efficiency and "white-box" transparency, achievable even under constrained computational resources. The source code is available here.**

I. Introduction

THE Atari 2600 Arcade Learning Environment (ALE) remains the standard benchmark for evaluating general competency in AI research [1][2]. Although Deep Reinforcement Learning (DRL) agents have achieved human-level control, their reliance on high-dimensional pixel inputs often results in "black-box" models with opaque decision processes [1]. This study investigates bio-inspired alternatives aimed at enhancing computational efficiency and interpretability.

This research explores Neuroevolution of Augmenting Topologies (NEAT) for structural optimization alongside *OpenEvolve* [3], a framework inspired by automated algorithm discovery [4]. By utilizing Large Language Models (LLMs) as semantic mutation operators, OpenEvolve shifts the evolutionary search from weight matrices to executable Python code, producing "white-box" agents with verifiable logic.

Applying evolutionary methods to Atari environments using visual inputs involves significant challenges, notably the "curse of dimensionality" and the risk of goal misalignment or shortcut learning [5][6]. To mitigate these issues, high-dimensional visual inputs are bypassed in favor of RAM state extraction. The OCAtari library is leveraged to implement an object-centric approach, mapping RAM into compact vectors suitable for relational reasoning [5]. Furthermore, fitness shaping wrappers are employed to provide the dense rewards necessary to guide the evolutionary process [6].

Performance is evaluated across three distinct scenarios: *Skiing*, characterized by sparse rewards and delayed credit assignment; *Freeway*, which tests synchronization and local optima avoidance; and *Space Invaders*, a benchmark for dynamic complexity. These environments serve to compare classic Neuroevolution (FFNN and RNN architectures) against LLM-driven Code Evolution, analyzing performance in terms of raw score and generalization capabilities.

II. Methodologies

A. Bio-Inspired Algorithms

The study employs two distinct evolutionary paradigms to investigate the emergence of autonomous control in Atari games: NeuroEvolution of Augmenting Topologies (NEAT), representing the state-of-the-art in structural neuroevolution and OpenEvolve, an innovative framework leveraging Large Language Models for automated program discovery.

1) NEAT: NeuroEvolution of Augmenting Topologies (NEAT) is a genetic algorithm that simultaneously optimizes connection weights and network topology, starting from minimal structures to ensure computational efficiency through complexification. This methodology was selected specifically for its ability to automatically discover compact, task-specific architectures, avoiding the computational overhead of fixed, fully-connected networks. By employing historical markings to solve the competing conventions problem and speciation to protect structural innovations, NEAT allows for the reliable evolution of increasingly sophisticated behaviors [7].

2) OPEN EVOLVE: OpenEvolve [3] is adopted as an open-source implementation of Google DeepMind's methodology for algorithm discovery [4], to benchmark the potential of Large Language Models in evolutionary tasks. Algorithmically, OpenEvolve can be viewed as a modernization of Genetic Programming (GP). Unlike traditional GP, which relies on stochastic perturbations, it leverages the semantic knowledge of LLMs to perform "intelligent mutations", shifting the search space from abstract neural weights to executable, interpretable Python code.

The implementation centers on a prompt sampler that aggregates system instructions, high-performing parent programs and execution errors to refine logic based on past failures. A local instance of `Qwen2.5-Coder-7B` (via Ollama) serves as the mutation operator, while the population is managed through a MAP-Elites database. This approach indexes solutions by score and algorithmic complexity, preventing premature convergence and preserving structural diversity.

B. Object-Centric Representation via OCAtari

Directly utilizing the raw RAM of the Atari 2600 console (128 bytes) for training bio-inspired agents presents significant challenges. Although it encapsulates the entire game state, the raw information often is inaccessible or misleading for learning agents: many bytes remain unused, others dynamically shift function depending on the game context. To overcome these intrinsic limitations, this work adopts an approach based on environment-specific wrappers obtained through the OCAtari library, which acts as semantic filters, extracting and normalizing only the features relevant to the specific task, thereby drastically reducing input dimensionality.

C. Game-Specific Environments & Reward Shaping

The custom wrappers designed to address the unique challenges of each environment are detailed, focusing on the specific state representations and reward shaping strategies adopted for both NEAT and OpenEvolve configurations.

1) Skiing: The Skiing environment (`ALE/Skiing-v5`) represents a challenging scenario due to its sparse reward structure and delayed credit assignment. The objective requires navigating a downhill slalom course using three discrete actions (NOOP, LEFT, RIGHT). The agent must avoid obstacles and pass through gates defined by pairs of flags while minimizing completion time. The native scoring system depends exclusively on the total elapsed time which is awarded only upon the conclusion of the run, creating a sparse and delayed reward signal that makes it difficult for an evolutionary agent to associate specific maneuvers with success. This limitation necessitates the use of a specialized wrapper to extract semantic features directly from the RAM, which are subsequently utilized to formulate a customized fitness function capable of guiding the evolutionary process.

The wrapper transforms raw object properties into a compact state vector of nine normalized features, organized into three semantic groups:

Player Dynamics (3 inputs): Encodes the agent's physical state through normalized horizontal position, ski orientation and horizontal velocity ($x - x_{prev}$).

Target Navigation (3 inputs): Provides guidance towards the next gate via lateral offset to the gate center, vertical proximity and a binary gate detection flag.

Obstacle Awareness (3 inputs): Facilitates collision avoidance by tracking the relative (x, y) coordinates of the nearest threat and a categorical indicator distinguishing between trees and flags.

To overcome the sparsity of native rewards, a custom fitness function provides dense, frame-by-frame guidance. The total fitness F accumulated over an episode is formalized as:

$$F = \sum_{t=0}^{T}(R_{gate} + R_{magnet} - P_{collision} - P_{boundary} - P_{time}) \tag{1}$$

where R_{gate} provides a reward for successful gate passages, R_{magnet} offers dense gradients for horizontal alignment to the target center, $P_{collision}$ penalizes collisions, $P_{boundary}$ penalizes track boundary violations and P_{time} discourages stagnation while minimizing descent time to ensure a fast and valid trajectory.

2) Freeway: The Freeway environment (`ALE/Freeway-v5`) serves as a benchmark for reactive obstacle avoidance and rhythmic timing. The agent, represented as a chicken, must navigate across ten lanes of traffic to reach the opposite side of the highway, by using 3 actions (NOOP, UP, DOWN). Freeway presents a unique challenge: collisions are not terminal but result in a significant temporal penalty, as the agent is pushed back and stunned for several frames, making time management the primary optimization objective.

To provide the agent with a semantically rich representation, a custom wrapper was implemented. The state is encoded into a 22-dimensional normalized vector:

Agent State (2 inputs): Normalized vertical position $y_{norm} \in [0, 1]$ of the chicken and a binary collision flag.

Traffic State (10 inputs): Horizontal positions (x) of the cars in each of the ten lanes, normalized relative to the screen width.

Temporal Dynamics (10 inputs): Computed velocities (Δx) of each car, allowing networks to perceive motion without frame-stacking and reducing topological complexity.

Similar to the other environments, the sparse, integer-based Atari score was transformed into a continuous landscape using a shaped fitness, providing a denser gradient more suitable for evolutionary optimization. The total fitness F is defined as:

$$F = \sum_{t=0}^{T}(R_{crossing} + Ry_{max} - P_{collision} - P_{timepenalty}) \tag{2}$$

Where $R_{crossing}$ is reward for the number of successful crossings, Ry_{max} is the maximum normalized vertical progress achieved in the current attempt (providing a dense reward even if no crossing is completed), $P_{collision}$ is the penalty inferred on the total count of collisions and $P_{timepenalty}$ is the penalty given for the total number of elapsed frames. The inclusion of a constant time penalty ($T \cdot 0.01$, in our case) is fundamental to discourage "camping" behavior, forcing the agent to take calculated risks to maximize the number of crossings within the episode limit.

3) Space Invaders: Space Invaders is a classic fixed-shooter where the player controls a laser cannon to defeat waves of descending aliens. The (`ALE/SpaceInvaders-v5`) environment presents a highly dynamic environment characterized by a variable number of destructible entities using 4 actions (NOOP, FIRE, RIGHT, LEFT), requiring the agent to manage firing cooldowns while dodging projectile trajectories.

As in the previous games, this complex state was mapped into fixed inputs and different wrapper philosophies were evaluated. The final selected approach compresses the RAM data into 19 normalized features:

Player Position (1 input): The recalibrated horizontal coordinate (P_x) within the playable area.

Threat Sensors (10 inputs): 5 vertical ray-casting sensors detecting the nearest projectile in each sector, plus 5 temporal deltas to infer velocity.

Targeting and Strategic Awareness (5 inputs): Combines the relative horizontal distance to the nearest bottom-row alien ($x_{alien} - P_x$) for immediate aiming with the enemy count across four quadrants to guide strategic positioning.

Game State (3 inputs): Features tracking the total alien fraction (game progression), UFO position and UFO availability.

As native scoring fails to penalize risky behavior, we guide the learning process via this fitness function:

$$F = \sum_{t=0}^{T}(R_{kill} + R_{aim} - P_{danger} - P_{spam}) \tag{3}$$

This composite metric sums weighted bonuses for alien elimination (R_{kill}) and target alignment (R_{aim}), while subtracting penalties for projectile proximity (P_{danger}) and firing cooldown abuse (P_{spam}), explicitly conditioning the agent's reward landscape to prioritize survival when threatened.

D. Open Evolve Setup

To ensure a rigorous comparison, OpenEvolve agents interact with the same OCAtari wrappers and reward shaping strategies described in Section II-C. However, the pipeline required specific adaptations.

System Prompt and Code Generation: a specialized System Prompt that functions as a "meta-genome" was designed, setting operational constraints (e.g., "no external libraries") and optimization priorities without enforcing a specific strategy. The LLM is tasked with generating valid Python code via a "Full Rewrite" strategy, as the chosen model (`Qwen2.5-Coder-7B`) proved unstable when attempting "Diff-Based" approach.

Cumulative Evolution: Finally, a cumulative evolutionary strategy was implemented, where the highest-performing code from generation N serves as a few-shot example for generation $N + 1$. The underlying hypothesis is that providing the LLM with its previous best attempt allows for iterative refinement of complex logic, enabling the system to "debug" and optimize strategies incrementally over generations.

III. Experimental Results

All evolutionary configurations were trained on random seeds (excluding the first 100) and evaluated on the remaining 100 seeds to assess generalization performance. Additionally, each game environment includes a comparison with a human benchmark to contextualize evolved agents' capabilities relative to intermediate-level human players.

A. Skiing

In the Skiing environment, four distinct evolutionary configurations were evaluated to assess the impact of architecture, state representation and reward structure (population: 100, generations: 150).

Since the native score represents negative elapsed time, early termination can paradoxically yield higher values than slow completion. To ensure meaningful comparisons, Table I reports raw scores conditioned on successful gate completion (fitness > 9700), with the exception of the baseline, whose results (*) reflect unconditional metrics and the reported 'Best Score' corresponds to the episode achieving the highest shaped fitness value (n refers to the successful runs).

The baseline configuration, trained on raw RAM with raw score and FFNN, failed to evolve gate-seeking behavior, converging instead to a degenerate strategy of vertical descent to minimize time. In contrast, all wrapper-based configurations successfully navigated gates, with performance distinctions driven by architecture and fitness shaping. Among these, the `Wr.+FFNN+Fit.dyn.` achieved the highest performance (peak score -4886) leveraging adaptive time penalties, surpassing the conservative `Wr.+FFNN+Fit.` (-5248). The

TABLE I
Comparative Results in Skiing Environment

Configuration	Best Score	Avg Score
Baseline	-6528.0^{*}	$-6401.6(\pm 718.5*)$
Wr. + FFNN + Fit. (n=100)	-5248.0	$-5581.1(\pm 136.5)$
Wr. + RNN + Fit. (n=36)	-7152.0	$-7844.2(\pm 522.8)$
Wr. + FFNN + Fit. dyn. (n=35)	$\mathbf{-4886.0}$	$\mathbf{-5391.6(\pm 334.8)}$
Human Benchmark (N=46)	-3341.0	$-3628.5(\pm 234.9)$

RNN variant yielded the poorest results and highest variance among successful agents, suggesting that the wrapper's explicit velocity features rendered recurrent memory redundant. Notably, initializing the FFNN architectures with 14 hidden nodes proved essential, preventing the stagnation often observed in minimal topologies. Although the best agents mastered gate precision, they have not yet matched human-level path planning.

B. Freeway

A similar experimental campaign was designed for Freeway (population: 100, generations: 50). A baseline configuration was first tested, which was proved insufficient for achieving competitive performance: while the agent managed to complete some crossings, it lacked a systematic understanding of traffic patterns, often resulting in "deadlock" situations where the chicken remained trapped in collision-recovery cycles.

To address these limitations, as already mentioned, multiple configurations were evaluated, involving both Feed-Forward (FFNN) and Recurrent (RNN) architectures, coupled with the 22-inputs wrapper (Table II). While the RNN models achieved quite high scores, probably by effectively "memorizing" the traffic rhythm through internal state transitions, the most robust and efficient performance was delivered by the `Wr.+FFNN+Fit.` (which achieves an average score of 25.06 and a best score of 29.0). The success of this model can be attributed to the synergy between explicit velocity features and continuous shaped fitness: by rewarding vertical progress and penalizing collisions frame-by-frame, the agent adjusts its speed to exploit narrow temporal windows rather than merely reacting to traffic. Still, evolved controllers exhibit limitations in global trajectory optimization compared to human-level benchmarks.

TABLE II
Comparative Results in Freeway Environment

Configuration	Best Score	Avg Score
Baseline	20.0	$17.07(\pm 1.12)$
Wr. + FFNN	21.0	$19.51(\pm 0.94)$
Wr. + FFNN + Fit.	**29.0**	$\mathbf{25.06(\pm 1.57)}$
Wr. + RRN	28.0	$24.90(\pm 1.52)$
Wr. + RNN + Fit.	28.0	$23.92(\pm 1.73)$
Human Benchmark (N=33)	31.0	$25.8(\pm 4.1)$

C. Space Invaders

As shown in Table III, initial configurations using Raw RAM and Column-based wrappers plateaued below 850 points, likely due to excessive input dimensionality. Despite this, the `Wr.Col+RNN` consistently outperformed FFNN runs, confirming that, unlike Skiing, here temporal memory is essential for tracking projectile trajectories.

To overcome the dimensionality bottleneck, the Egocentric wrapper was included: this yielded an immediate improvement in average performance (304 ± 149). Recognizing the complexity of the task, the evolutionary budget was scaled from a minimal configuration (population: 150, generations: 50) to more extended runs (population: 500, generations: 300). Under these expanded conditions, the impact of the Customized Fitness became decisive. While `Wr.Ego.+RNN` reached a respectable best score of 1880, the configuration guided by the custom fitness function achieved the global best of 2,380 points (421 ± 120 avg). This confirms that for dynamic multi-agent environments, semantic compression must be supported by both temporal memory (RNN) and dense reward guidance.

Qualitative analysis of the best agent revealed sophisticated emergent behaviors, such as learning to "lead" shots against moving targets and prioritizing the high-value "Mystery Ship" (UFO), a strategic nuance that emerged naturally despite not being explicitly encoded in the fitness formula.

TABLE III
COMPARATIVE RESULTS IN SPACE INVADERS ENVIRONMENT

Configuration	**Best Score**	**Avg Score**
Baseline	760.0	221(±156)
Wr. Col + FFNN	760.0	233(±154)
Wr. Col + RNN	830.0	249(±162)
Wr. Ego + RNN	1050.0	304(±149)
Wr. Ego + RNN (ext. run)	1880.0	366(±113)
Wr. Ego + RNN + Fit (ext. run)	**2380.0**	**421(±120)**
Human Benchmark (N=54)	970.0	541.3(±194)

D. OpenEvolve

OpenEvolve was evaluated across all three environments using the optimal wrappers and fitness functions identified in prior experiments. The evolutionary budget involved a population of 2,500 candidates and 1,500 generations; despite the higher evaluation count, the total wall-clock time on the same hardware was comparable to the extended NEAT runs.

In *Skiing*, the LLM-generated agent achieved a remarkable peak score of -3856, effectively leveraging the cumulative evolutionary strategy to refine trajectory planning. In *Freeway*, the evolved code exhibited a safety-prioritizing behavior, resulting in a best score of 18.0. Finally, in *Space Invaders*, the approach demonstrated strong generalization across varying initial conditions, attaining an average score of 517. The performance metrics for the best evolved code agents are summarized in Table IV.

TABLE IV
OPENEVOLVE PERFORMANCE BENCHMARKS

Environment	**Best Score**	**Avg Score**
Skiing (n=57)	−3856.0	−4004.9(±85.8)
Freeway	18.0	15.28(±1.43)
Space Invaders	1495.0	517.0(±212)

IV. DISCUSSION AND LESSONS LEARNED

The experimental results reveal complementary strengths between the two evolutionary paradigms. In Skiing, OpenEvolve achieved the best overall performance (3856.0), significantly outperforming NEAT's top configuration (4886.0). Conversely, in Freeway, NEAT agents exploited frame-perfect timing to achieve near-optimal throughput (29.0), while OpenEvolve adopted a conservative strategy, resulting in lower peak scores (18.0). In Space Invaders, although NEAT achieved a higher peak score (2,380) by capitalizing on stochastic high-value targets, OpenEvolve demonstrated superior generalization with a higher average score on unseen seeds (517.0 vs 421.0), suggesting more robust policy learning.

Beyond performance metrics, the fundamental difference lies in interpretability: while NEAT produces effective but opaque "black-box" controllers that are difficult to inspect, OpenEvolve generates "white-box" agents in the form of executable Python code. This transparency allows for direct analysis, facilitating debugging and human understanding of emerging decision-making strategies.

Despite the performance gains, several technical challenges were encountered. First, baseline experiments confirmed that evolutionary search becomes effectively blind without dense rewards, often converging to degenerate strategies. Second, the design of wrappers and fitness functions remains a non-trivial trade-off: abstract representations risk information loss, while detailed ones suffer from dimensionality issues. Furthermore, misaligned fitness shaping can inadvertently reward unintended behaviors, requiring careful calibration to ensure objectives match game goals.

Regarding OpenEvolve, the exclusive reliance on `Qwen2.5-Coder-7B` represents a key limitation, likely hindering the cumulative evolutionary strategy and leading to inconsistent results across environments. Future work with more capable models (e.g., GPT, Claude) could overcome this through superior context handling and stable "Diff-Based" generation.

Finally, another limitation concerns the computational infrastructure. The deliberate restriction to consumer-grade hardware, while verifying accessibility, imposed some logistical constraints.

V. CONCLUSION

This work indicates that, especially on consumer-grade hardware, bio-inspired learning without semantic preprocessing is insufficient for complex environments. The curse of dimensionality makes object-centric wrappers and fitness shaping fundamental architectural requirements rather than simple optimizations. Additionally, the study highlights a performance-interpretability trade-off: NEAT provides efficiency, while OpenEvolve ensures transparency at the cost of higher complexity. Future research should prioritize overcoming current hardware constraints by deploying these frameworks on high-performance clusters and leveraging more capable LLMs to fully assess their scalability.

Authors' Contributions

- **T. Ballarini**: Space Invaders wrapper/fitness design, OpenEvolve implementation and evaluation.
- **C. Belli**: OpenEvolve setup and prompt engineering, Freeway wrapper/fitness design
- **E. Negrini**: Skiing wrapper/fitness design, experimental protocol coordination, report preparation, project documentation

All authors conducted exploratory experiments on additional Atari environments not included in this work, which informed the design principles for wrappers and fitness shaping strategies presented herein. All approved the final report.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of artificial intelligence research*, vol. 47, pp. 253–279, 2013.

[3] Algorithmic Superintelligence, "OpenEvolve: Github repository," https://github.com/algorithmicsuperintelligence/openevolve, 2024, accessed: 2025-11-30.

[4] A. Novikov *et al.*, "Alphaevolve: A coding agent for scientific and algorithmic discovery," *arXiv preprint arXiv:2506.13131*, 2025.

[5] Q. Delfosse, J. Blüml, B. Gregori, S. Sztwiertnia, and K. Kersting, "Ocatari: Object-centric atari 2600 reinforcement learning environments," *arXiv preprint arXiv:2306.08649*, 2023.

[6] Q. Delfosse, S. Sztwiertnia, M. Rothermel, W. Stammer, and K. Kersting, "Interpretable concept bottlenecks to align reinforcement learning agents," *Advances in Neural Information Processing Systems*, vol. 37, pp. 66 826–66 855, 2024.

[7] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.