Hypothesis:

IF

the mutation rate of a neural network is dynamically scaled inversely to its fitness score,

THEN

the population will reach a state of Policy Entropy stability (Nash Equilibrium) in fewer generations than a control group with a static mutation rate,

BECAUSE

this mechanism mimics biological 'stress-induced mutagenesis,' allowing poor-performing agents to explore the solution space rapidly while high-performing agents preserve their successful traits, balancing exploration and exploitation more efficiently.

Problem:

Deep Reinforcement Learning (DRL) is expensive to compute and hyper sensitive to many parameters. Simple evolutionary algorithms work well but often take longer to reach convergence because a fixed mutation rate is inefficient. Too high of a mutation rate disrupts good policies, and too low of a rate can cause it to stop altogether. Finding the optimal balance for the mutation rate is difficult and dependent on many factors.

Abstract:

This experiment investigates the efficiency of evolutionary algorithms in training AI. We compare a standard static mutation rate against an adaptive mutation strategy where the mutation rate scales inversely with an agent's fitness score. The hypothesis is that if an agent's mutation rate scales inversely with its fitness score it will result in faster convergence to a stable strategy (Nash Equilibrium).

Variables:

Independent Variable The factor that is intentionally changed Mutation Strategy: Control (Static) vs. Adaptive (Fitness-Scaled) ● Adaptive Generation Count Number of evolutionary generations ● Continuous	**Dependent Variables** The outcomes being measured Convergence Velocity: Generations to reach Nash ● Equilibrium (Policy Entropy < 0.001) Policy Entropy: ● Measure of strategy randomness (Shannon Entropy) Entropy Variance: ● Stability of population strategies Fitness Score: ● Relative skill level (Self-play performance)	**Controlled Variables** Factors kept constant to ensure validity Population Size: ● 1000 Fixed number of agents per generation Selection Pressure: ● Top 20% Tournament size / truncation ratio Network Architecture ● 24-64-4 Input/Hidden/Output layers Simulation Ticks: ● 750 Ticks per generation Random Seed Pairings: ● Matched Identical seeds for Control/Experimental pairs

Materials Used:

Hardware:

- Distributed Worker GPU Nodes (x4) w. nVidia Acceleration Cards (CUDA compatible GPUs)
  - nVidia 3080 (x1)
  - nVidia 3090 (x2)
  - nVidia 4090 (x1)
- Central Database Server which runs https://sf.defouw.ca/

Software Stack:

- Python 3.10
- PyTorch (Neural Networks framework)
- PostgreSQL (DB and Data Storage)
- Next.js Dashboard
- Python Pandas
- Linux Mint (Server)
- Windows 11 (Workers)

Procedure:

Step 1

Baseline Establishment

- Run Control experiments with static mutation rate ($\varepsilon=0.05$) to find convergence speed.

Step 2

Adaptive Implementation

- Implement adaptive mutation scaling. Mutation rate decreases as fitness increases.

Step 3

Data Collection

- Run at least 10 matched pairs of experiments (Control vs. Experimental) with identical seeds to ensure scientific rigor.
- Upload data to the central database.
- Monitor for failures.

Step 4

Statistical Analysis

- Perform Welch's t-test and Cohen's d test on convergence generations and analyze the results.