Sources of Error:

Simulation Randomization:

- how random number generators (RNGs) control the stochastic elements of each simulation run, and how those RNGs are deliberately seeded to ensure scientific rigor.
- Initial neural network weights, all 1,860 weights per agent are initialized randomly from a normal distribution
- Agent starting positions and angles, each of the 1,000 agents is placed at a random (x, y) coordinate and facing a random direction in the 1000×1000 toroidal Petri Dish
- Food pellet placement, 500 food pellets are spawned at random locations (and re-spawned periodically every 200 ticks)
- Crossover masks during reproduction, a random coin flip determines which parent contributes each of the 1,860 weights
- Mutation noise, random Gaussian noise is added to offspring weights
- Parent selection– top 20%
- At the start of every experiment, the code seeds all random number generators to Python's random, NumPy's np.random, PyTorch's CPU and GPU generators with this single seed value. This means that for

a given seed, the exact same sequence of "random" numbers is produced every time, making the initial population, food layout, and all stochastic decisions deterministic and reproducible.

- Why this matters for the experiment: This is a paired-seed design. For each seed, both a control (static mutation) and an experimental (adaptive mutation) run use the exact same initial population of neural networks and the exact same starting food placement. The only difference between the two runs is the mutation strategy. This isolates the independent variable (mutation mode) as the sole cause of any difference in outcomes, which strengthens the validity of the statistical comparison. Without identical seeds, differences in convergence speed could be attributed to one group randomly starting with a better initial population.

Network Initialization Variance

- the spread of the random initial weights assigned to each agent's neural network at the start of an experiment.
- In EvoNash, every agent's 1,860 neural network weights are initialized by drawing from a normal distribution with mean 0 and standard deviation 0.1

- This means at generation 0, each weight is a small random number centered around zero. The standard deviation of 0.1 controls how spread out those initial values are, this is the initialization variance Why 0.1?
    - Too small: All agents would start with weights near zero, producing nearly identical outputs. The population would have almost no behavioral diversity to begin with, giving the genetic algorithm very little variation to select from.
    - Too large: Agents would start with extreme, erratic behaviors (huge thrust, constant shooting, wild turning). Many would die immediately, and the useful "signal" in the weights would be drowned out by noise.
    - 0.1 is a sweet spot: Agents start with small but meaningfully different weights, producing varied but not chaotic initial behaviors. This gives the genetic algorithm a diverse starting population to work with while keeping outputs in a reasonable range through the ReLU activation and output clamping.
- Why it matters for the experiment: Since both the control and experimental groups use the same seed, they start with the exact same set of randomly initialized networks (same variance, same

values). This is a controlled variable, any difference in convergence speed cannot be attributed to one group getting "luckier" initial weights. The initialization variance is identical and fixed at $\sigma = 0.1$ across all 1,292 experiments.

Environmental Noise Across Runs

The random variation in the simulation environment that differs from one experiment run to another, and how the experiment controls for that. In EvoNash, the environment is the Petri Dish, a 1000×1000 toroidal world with food pellets, physics, and agent interactions. Several environmental factors introduce randomness each run:

Sources of environmental noise:

- Food placement, 500 food pellets are spawned at random (x, y) coordinates at the start of each generation, and re-spawned every 200 ticks. Different food layouts create different foraging landscapes, one run might cluster food in a corner, another might spread it evenly.

- Agent starting positions and angles, all 1,000 agents are placed at random positions and facing random directions. This means agents start near different food or different competitors each run.
- Interaction dynamics because agents move simultaneously, tiny differences in starting positions cascade into completely different encounter sequences. Agent A might collide with Agent B in one run but never meet in another. These chaotic interactions (projectile hits, energy theft, food competition) create unpredictable fitness outcomes.
- Food respawning, food respawns periodically at new random locations during the generation, creating a shifting resource landscape that differs across runs.

How EvoNash Controls for this Noise:

- Within a seed pair: Both the control and experimental runs for the same seed produce the exact same environmental noise, identical food layouts, identical starting positions, identical initial encounter dynamics. This is because all RNGs (NumPy, PyTorch, Python random) are seeded identically. So for seed 42, both the static and

adaptive run experience the same environment. Any difference in outcome is due to the mutation strategy alone.

- Across different seeds: Each seed creates a different random environment. Seed 42 has a completely different food layout and starting configuration than seed 43. This is intentional, it's what makes each experiment an independent sample. The environmental variation across seeds is the natural noise that the statistical analysis (t-test, confidence intervals) accounts for. By running 1,292 experiments across many different seeds, the effect of any single "lucky" or "unlucky" environment is averaged out.
- Fixed physics parameters: While random elements vary, the underlying physics rules are constant across every run same friction (0.0), same energy decay (1.0/tick), same food energy value (25), same thrust force (20), same world size (1000×1000), same ticks per generation (750). These controlled variables ensure the "rules of the game" are identical.