Background Research:

I used Google Antigravity to work on this project with me. For fun, I asked Antigravity "*Can you tell me how you and I worked on this project from my initial idea to the completed science experiment we built together?*"

Here is what Antigravity replied:

How We Built EvoNash: A Human-AI Collaboration

You (Joel) were the scientist, project lead, and operator. Antigravity (me) was your AI developer and devops engineer. Here's how we worked together to take this from

an idea to a completed experiment with over 1,200 runs and a live dashboard.

The Roles

You — The Human Operator:

- Came up with the scientific question: "Can smart mutation rates help AI agents find Nash Equilibrium faster?"
- Made every design decision — what the experiment should test, how agents should behave, what statistics to use
- Approved or rejected every code change before it went live
- Ran the GPU workers on your hardware to execute the actual experiments

- Interpreted the results and decided what to put on your science fair poster
- Caught bugs and told me when something looked wrong ("the data just got wiped," "the p-value is showing wrong")

Antigravity — The AI Developer & DevOps:

- Wrote the actual code — Python for the simulation, TypeScript/React for the dashboard, SQL for the database
- Set up the server infrastructure (Linux server, PostgreSQL database, domain name, HTTPS)
- Debugged issues when things broke (500 errors, memory crashes, incorrect displays)

- Optimized performance (GPU acceleration, batched neural network inference, SQL query tuning)
- Explained concepts back to you when needed (neural networks, statistics, game theory)

How a Typical Work Session Went

- You'd describe what you wanted — in plain English, like "I need the dashboard to show how many experiments converged" or "the fitness graph looks wrong."
- I'd research the codebase — reading through files, understanding how things connected, figuring out where the problem was or where the new feature should go.

- I'd propose a plan — explaining what I'd change and why, so you could check that my approach was scientifically sound.
- You'd approve or adjust — sometimes you'd say "go ahead," other times you'd push back: "that won't maintain data integrity" or "make sure it matches the original logic exactly."
- I'd write the code — making the actual changes to files on your computer.
- We'd deploy together — I'd push the code to GitHub, then run commands on your server to pull, build, and restart the application.
- You'd verify it worked — checking the live dashboard, confirming the numbers made sense, and flagging anything that looked off.

The Journey, Start to Finish

Phase 1 — Building the Foundation: We built the simulation engine in Python with PyTorch — 1,000 neural network agents living in a virtual Petri Dish, competing for food, evolving through a genetic algorithm. We set up two modes: static mutation (control) and adaptive mutation (experimental).

Phase 2 — The Dashboard: We built a Next.js web application (the dashboard at sf.defouw.ca) so you could monitor experiments in real-time — watching fitness curves, convergence detection, and statistical tests update live as workers completed runs.

Phase 3 — Running Experiments: You ran GPU workers that executed hundreds of experiments, each with a

unique random seed. The workers automatically uploaded results to the PostgreSQL database on your server.

Phase 4 — Scaling Problems: As the dataset grew past 1,000 experiments and 1.5 million generation records, the dashboard crashed with memory errors. I refactored the heavy computation from JavaScript into optimized SQL queries using window functions — keeping the exact same scientific logic but letting the database handle the math instead of loading everything into memory.

Phase 5 — Analysis & Poster: With the dashboard working again, we pulled the final statistics — convergence generations, t-test results, effect sizes — and I helped you write the observations text for your science fair poster board.

What Made This Work

The key to this collaboration was that you didn't need to know how to code to drive the project. You understood the science — the hypothesis, the experimental design, what the numbers should look like — and I handled the engineering. When I made a mistake (like a query that could have broken data integrity), you caught it because you understood what the correct output should be. And when you saw something confusing on the dashboard, you described it to me and I tracked down the bug.

It's the same idea as a scientist working with a lab technician: the scientist designs the experiment and

interprets the results, while the technician builds and operates the equipment. In this case, the "equipment" was software, and the "technician" was an AI.