KARIM'S AI LAB · DEMO 8

The auditor that rewrites itself

A machine that writes its own fraud-detection rule, improves it forty times, and then stops — and what that tells us about our own work.

This is a small program with a big claim attached to it. It is given a ledger and a budget, and no knowledge whatsoever of what fraud looks like. Over about forty seconds it writes a rule, tests it, rewrites it, and ends up finding seven times more fraud than random sampling — using a piece of professional judgement that nobody gave it. Then a single button turns the same machine into a perfect example of why performance measures destroy the thing they measure.

THE TASK

For anyone: imagine a company's accounts for the year — two thousand payments to suppliers. Sixty of them are fraudulent. An auditor cannot check all two thousand; there is time to examine a hundred. Which hundred? Pick at random and you will find about five of the sixty. The question the machine is set is simply: *which hundred should we look at?*

For auditors: the ledger is synthetic but built to behave. Legitimate spend is skewed — about seventy per cent under €6,200, a long tail out past €200,000 — with realistic noise on the features that matter: 18% of honest invoices are round numbers, 22% come from vendors less than four months old. The fraud is planted with three recognisable signatures and one deliberate blind spot:

- **Threshold gaming** — invoices priced just below €10,000, the limit above which a second signature is required, and approved at a single level. (24 of the 60.)
- **New-vendor risk** — mid-sized invoices to vendors less than seventy days old. (12 of the 60.)
- **Period-end pressure** — round-number amounts pushed through in the last five days of a quarter. (12 of the 60.)
- **Nothing at all** — twelve frauds carry no signature. They are undiscoverable by design, so that a perfect score is impossible and the ceiling you see is honest.

The machine is told none of this. It is not told that €10,000 matters, that approval limits exist, or that anyone games them. It sees six columns — amount, vendor age in days, day of the quarter, approver level, number of line items, spend category — and a single number at the end: how many frauds its selection caught.

WHAT IT ACTUALLY WRITES

The thing being improved is a scoring function in ordinary JavaScript. Every transaction is scored, the hundred highest scores are pulled for review. It begins deliberately stupid:

```
function risk(t) {
  let score = 0;
  if (t.amount > 1000) score += 1;
  return score;
}
```

Each generation it writes twenty variants of that function and scores every one against the whole ledger — two thousand rows, twenty times, forty generations: 1.6 million evaluations, which take less than a quarter of a second. The pacing you see on screen is deliberately slowed so the code changes are readable.

A variant is made by one of seven edits: nudge a threshold, flip a comparison, point a condition at a different column, change a weight, narrow a rule by adding an `&&` condition, add a whole new rule, or delete one. If the best variant beats the incumbent it becomes the new incumbent; otherwise nothing changes and the generation is wasted. There is also a tidying step: any condition whose removal leaves the result *identical* is stripped out, because this kind of search accumulates clutter that does nothing, and the point here is to be able to read the answer.

WHAT IT FOUND

After forty generations the entire program is one line:

```
function risk(t) {
  let score = 0;
  if (t.amount > 6750 && t.amount < 11450) score += 1;
  return score;
}
```

Review every invoice between roughly €6,750 and €11,450. That finds **38 of the 60** frauds — against five for random sampling — inside the same hundred-item budget. Every auditor recognises the rule immediately: it is the band bracketing the approval threshold, which is exactly where someone who wants to avoid a second signature has to operate. The machine inferred that a threshold exists, and where it sits, purely from the fact that guilty invoices cluster just underneath it.

The curve matters as much as the answer. It climbs slowly for twenty-nine generations, from eight caught to twenty-four. Then in a *single* generation it jumps to thirty-eight — the moment it happens to bracket the amount from both sides at once, which no smaller edit could have reached. Then it goes flat, and the last ten generations find nothing at all.

THE SECOND EXPERIMENT: THE MEASURE THAT EATS THE GOAL

There is a second button. It changes nothing about the machine, the ledger or the budget. It changes only what counts as better — from *frauds found* to *percentage of total spend reviewed*, a coverage figure of the kind that genuinely appears in audit reporting and internal scorecards.

It optimises the new measure beautifully, and writes a rule that reviews the largest invoices:

OBJECTIVE GIVEN	SPEND REVIEWED	FRAUDS FOUND
Frauds found	2.7%	38 of 60
Percentage of spend reviewed	46.7%	0 of 60

Forty-seven per cent of all company spend examined — an excellent number to report — and not one of the sixty frauds found. The failure is not a bug and not stupidity. The machine did precisely what it was asked. The fraud is all hiding under €10,000, and euros are concentrated in the large clean invoices, so the two goals point in almost opposite directions. Any measure you can name, a sufficiently determined optimiser will satisfy without doing the thing you actually wanted — and it will look excellent while doing so.

WHAT ALPHAEVOLVE IS

This demo is a miniature of a real system. **AlphaEvolve** was unveiled by Google DeepMind in May 2025: an evolutionary coding agent that designs algorithms. The loop is the same three parts as here — propose, judge, keep — with one difference that matters: where this demo mutates code with seven blunt edits, AlphaEvolve has an ensemble of Gemini models *write* the candidate programs. Everything else is the same shape: automated metrics score each candidate, and a database of past programs decides which ones seed the next round.

WHAT IT HAS ACTUALLY DONE

· Multiplied two 4×4 complex matrices in **48** scalar multiplications, beating Strassen's 49 from 1969 — the first improvement in that case in 56 years.

· Written a scheduling heuristic for Borg, Google's cluster manager, that **continuously recovers 0.7% of Google's worldwide compute**. It has been running in production for over a year.

· Sped up a critical matrix-multiplication kernel in Gemini's own architecture by **23%**, cutting Gemini's total training time by **1%**. The system now helps train the models it is built from.

· Achieved up to a **32.5%** speedup on the FlashAttention kernel, and simplified a TPU arithmetic circuit.

· Across **50+ open mathematical problems**, matched the best known answer about **75%** of the time and **improved** on it about **20%** of the time — including a new lower bound for the kissing number in 11 dimensions, with a configuration of 593 spheres.

DeepMind is explicit about the one precondition, and it is the whole story: the problem's solutions must be **automatically verifiable**. Matrix multiplications can be checked. Compute recovered can be measured. Kernel speed can be timed. Wherever a machine can mark its own homework, this loop works — and, as the Borg result shows, works permanently and at industrial scale.

WHY THIS IS NOT RECURSIVE SELF-IMPROVEMENT

It is worth being precise, because the phrase is everywhere at the moment and this demo would be easy to over-sell. Three things get called self-improvement and they are in very different states.

Improving the output. The system searches for a better artefact — an algorithm, a rule, a heuristic — while remaining unchanged itself. That is AlphaEvolve, and that is this demo. It works, it is in production, and it is genuinely startling.

Improving the scaffolding. An agent rewrites its own code — its tools, its prompts, its loop — though not the model underneath. This also works, and recently: Sakana AI's Darwin Gödel Machine took itself from a 20% to a 50% success rate on a standard software-engineering benchmark by editing its own source.

Improving the ability to improve. Each round making the *next* round better, so that gains compound instead of accumulating. **This has not been achieved.** It is the thing people mean by an intelligence explosion, and it is why new laboratories are being founded specifically to attempt it.

Watch this demo's last ten generations to see why the distinction is not academic. The mutation operators at generation forty are precisely the ones from generation one. The improver never improved. It has exhausted what its own repertoire of seven edits can reach, and being run for a thousand more generations would not help, because nothing in the loop makes the loop better. That is the honest state of the art: today's self-improving systems plateau rather than accelerate — and they plateau at the point where the search can no longer find anything the measure will reward.

THE QUESTION WORTH TAKING AWAY

Not *can machines improve themselves?* — but: **can you say what better means, precisely enough that a machine could check it without you?**

Where the answer is yes, expect this loop, and expect it soon: the judge is cheap, the search is patient, and the result may be something no one in your profession has thought of. Where the answer is no, that work is out of reach for now — but not because the machine is not clever enough. It is out of reach because we cannot state the goal. And that ought to be uncomfortable rather than reassuring, because a great deal of work that cannot say what better means is nevertheless being measured by a proxy for it — and a proxy is exactly what the second button destroys.

Karim Benammar · karim-ai-lab.netlify.app/evolve · Run it yourself; it trains live in the browser, with no data leaving your machine. The demo is deterministic — the same seed produces the same forty generations every time, on any computer. Sources for the AlphaEvolve figures: Google DeepMind's AlphaEvolve announcement (May 2025) and Sakana AI's Darwin Gödel Machine paper (arXiv 2505.22954).