CS395T: Foundations of Machine Learning for Systems Researchers

Fall 2025

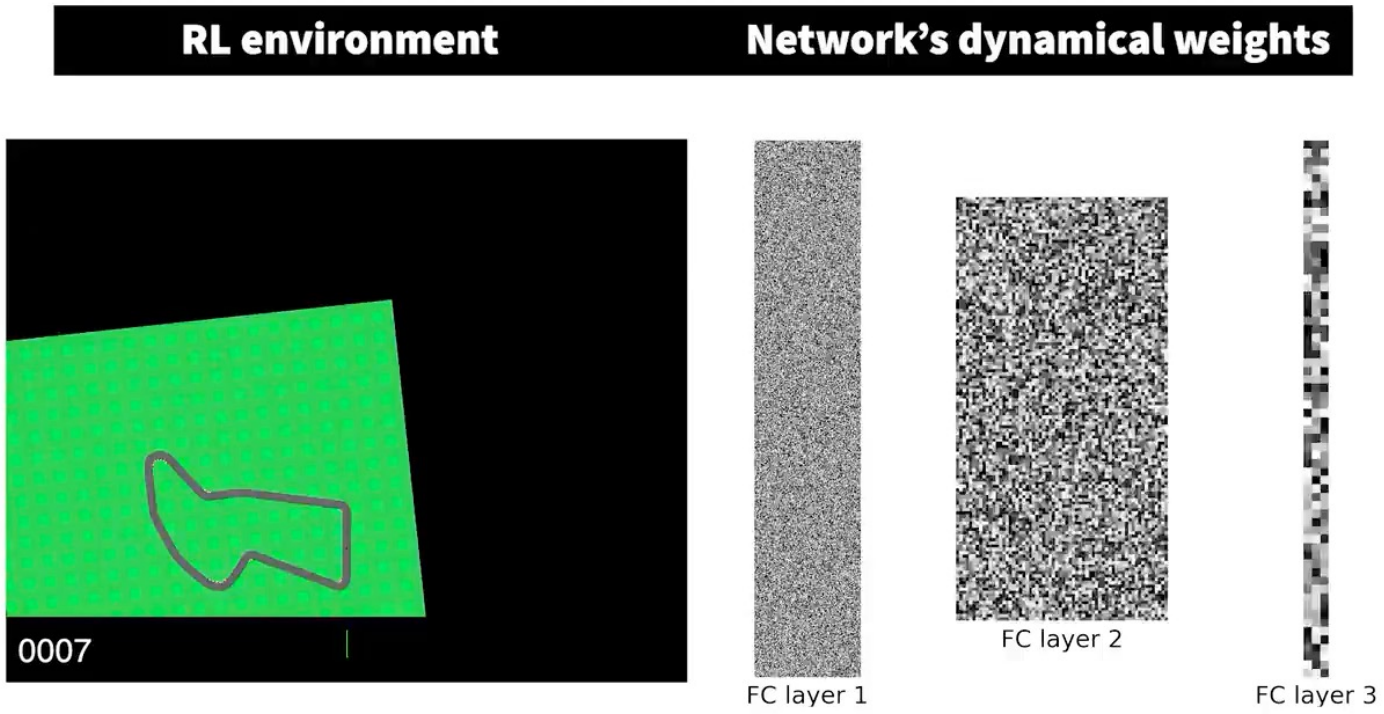
Lecture 12: Evolution

Lain (Zelal) Mustafaoglu

Overview

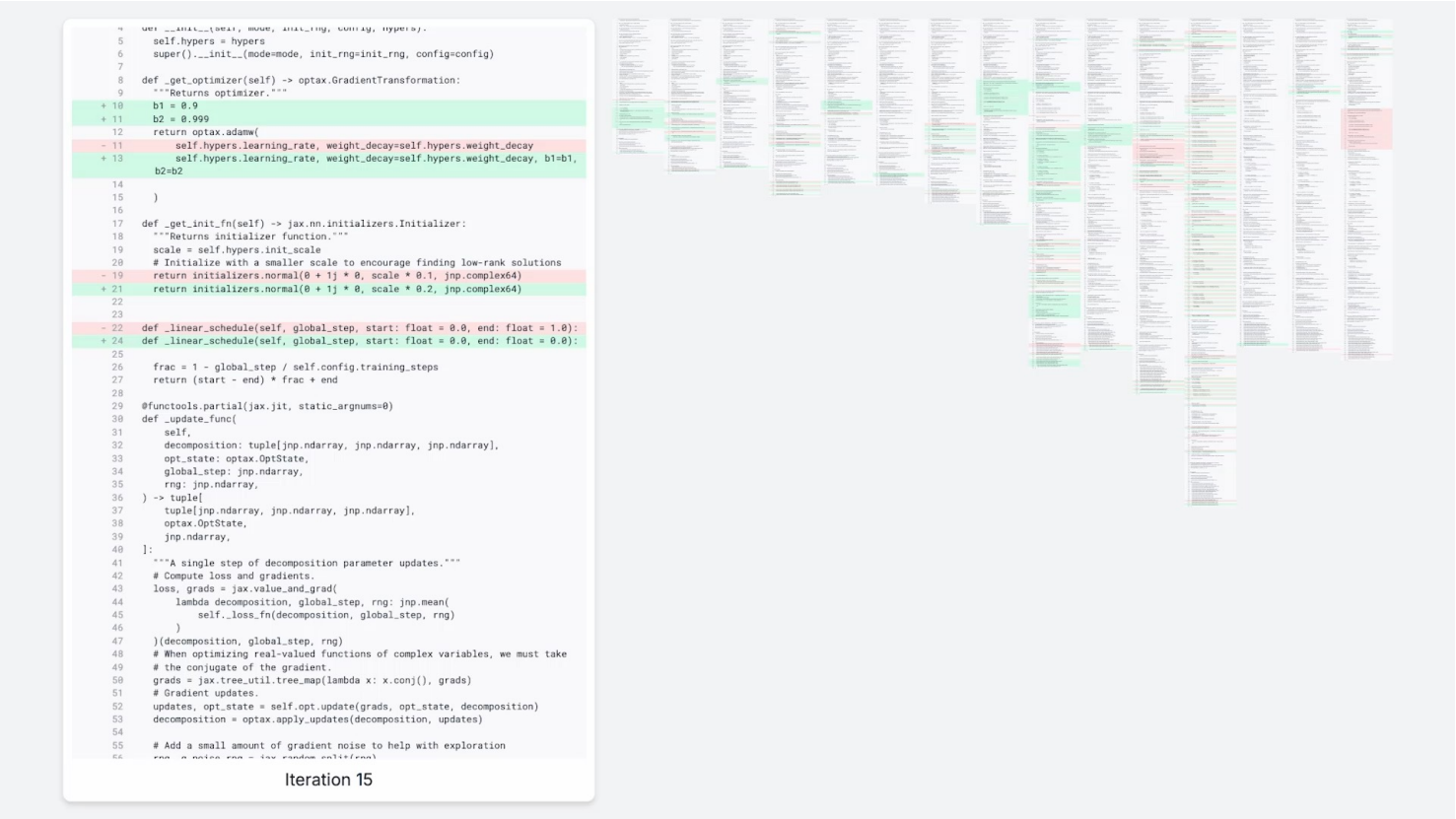
- Demos
- Mathematical foundations
  - Non-linear function optimization
  - Gradient-based vs. black-box optimization
  - Finite differences
  - ES / OpenAI-ES / NES
  - Natural gradient
- Key components of evolutionary algorithms
- Applications: AlphaEvolve, Training-Free GRPO
- Neuroevolution

Demo (I)



Neuroevolution: Harnessing Creativity in AI Agent Design, *An MIT Press Book by* Sebastian Risi, Yujin Tang, David Ha, *and* Risto Miikkulainen
https://neuroevolutionbook.com/demos/

Demo (II)



Source: [AlphaEvolve: A coding agent for scientific and algorithmic discovery](#) (Novikov et al., 2025)

Non-linear function optimization

Need search

Two key questions in all search methods:

1. At what point do you start the search?
2. What is the next search point, given some information from current search point?
   - Need to know two things: direction to move in, step size

Gradient-based optimization

- We've looked at gradient ascent/descent
  - Start with a random point
  - Obtain next search point by taking step along gradient:
    - Alpha → step size; gradient → direction

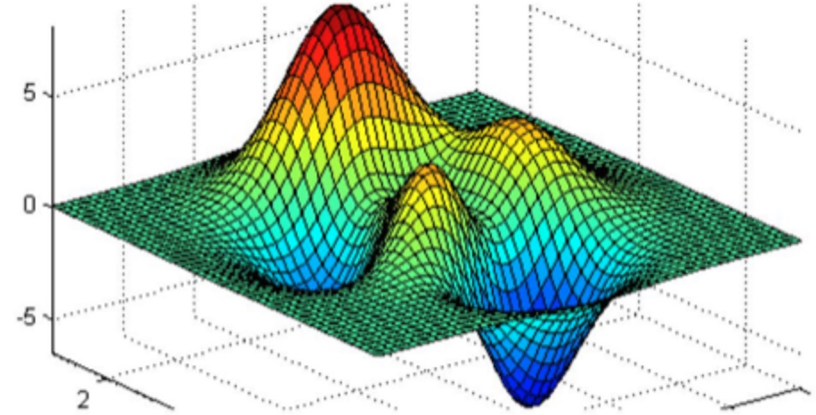
$$x_{t+1} = x_t + \alpha \nabla f(x_t)$$

Might be noisy or unavailable

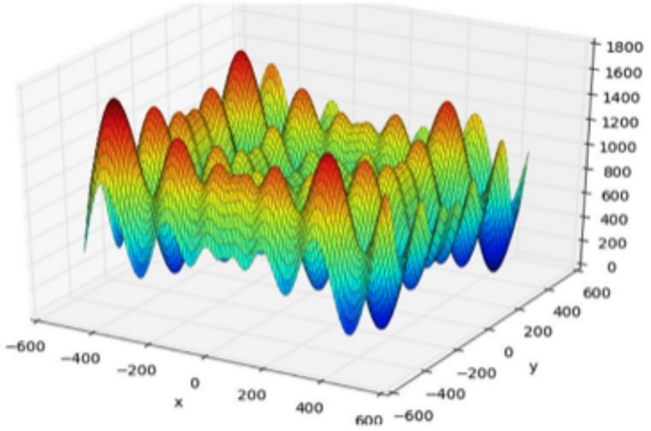
- Two issues with any gradient-based method:
  - Requires gradient information
  - Does *local* optimization (exploitation) unless exploration is injected explicitly (entropy bonus, etc.)

Black-box optimization

Optimization when no gradients are available, but function can be evaluated at different points



(a) Search Space Appropriate for Hill Climbing



(b) Search Space in a Creative Domain

Source: [Creative AI Through Evolutionary Computation: Principles and Examples]

Finite differences

Approximate derivatives with differences, which only require function evaluation

Forward differences:

$$\frac{\mathrm{d}f}{dx} \approx \frac{f(x+\Delta \mathrm{x}) - \mathrm{f(x)}}{\Delta \mathrm{x}}$$

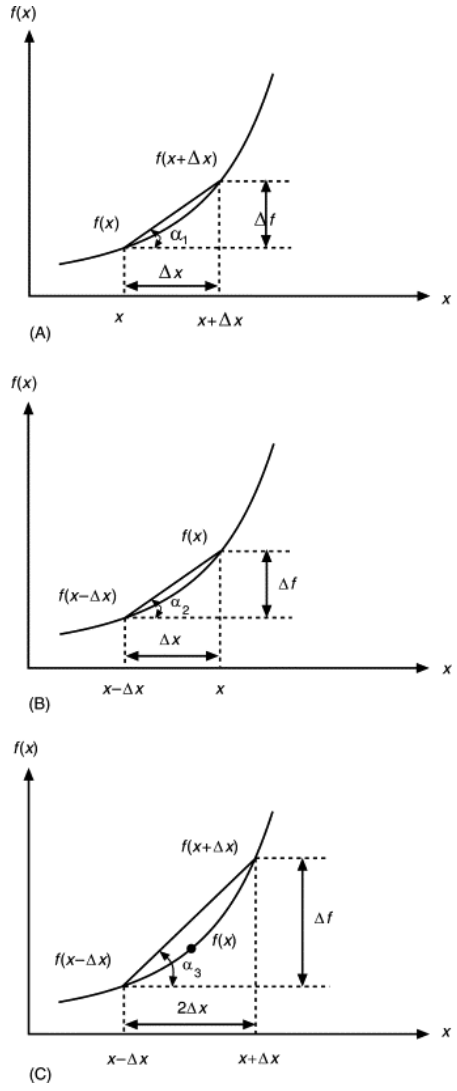
Backward differences:

$$\frac{\mathrm{d}f}{dx} \approx \frac{\mathrm{f(x)} - f(x-\Delta \mathrm{x})}{\Delta \mathrm{x}}$$

Centered differences:

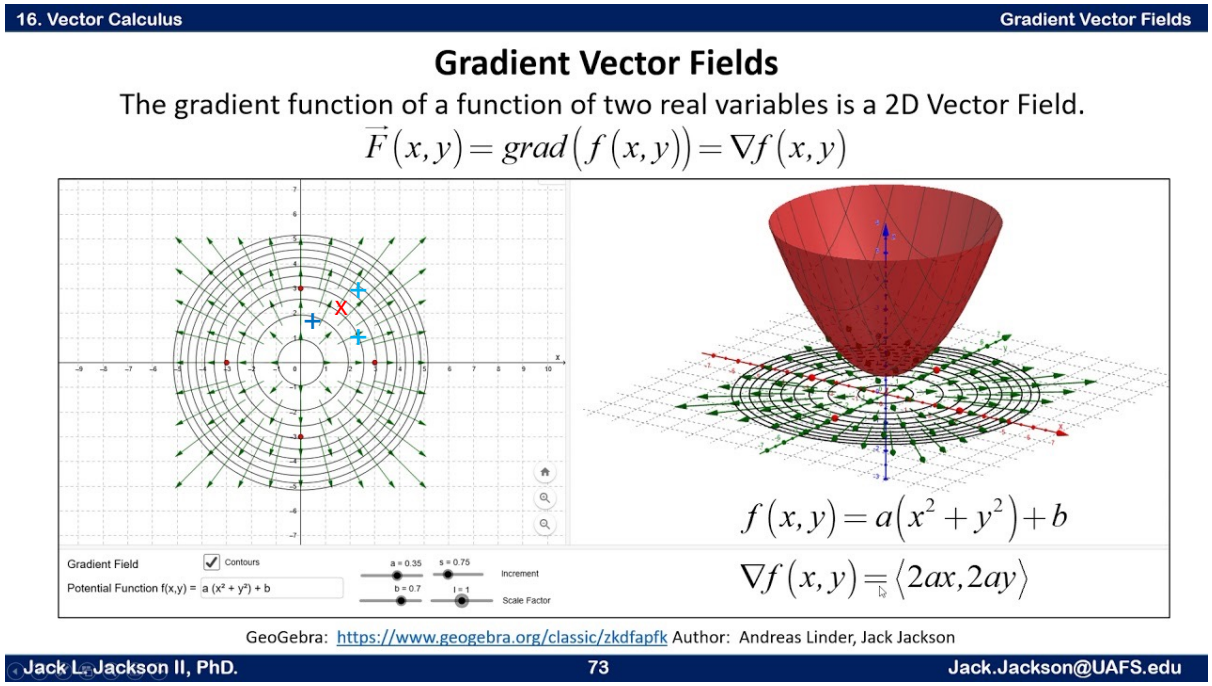
$$\frac{\mathrm{d}f}{dx} \approx \frac{f(x+\Delta \mathrm{x}) - f(x-\Delta \mathrm{x})}{2\Delta \mathrm{x}}$$

Higher dimensions: we can approximate partial derivatives in same way but we need *gradient*.



https://www.sciencedirect.com/topics/engineering/finite-difference-formula

One solution for higher dimensions



Use fact that gradient is direction of greatest increase in function value

- Sample a bunch of points around current search point
- Find maximal point
- Take step in direction of that point

Problem: greedy, so may get trapped in local optimum like gradient ascent/descent

Solution: don’t be greedy

Repeat:

- Sample points in neighborhood of current search point
- Pick *top-k* by function value (set of **elites** $\mathcal{E}$)
  - Referred to as **fitness score**
- Update search point to *elite average*

$$\bar{\mu}_{i+1} \leftarrow \frac{1}{k} \sum_{\bar{\theta}_n \in \mathcal{E}} \bar{\theta}_n$$

One step further: antithetic evolutionary strategies (ES)

- Sample random directions $\bar{\epsilon}_n$
- Evaluate two candidates for each direction: $\bar{\theta}_n^{\pm} = \bar{\mu}_i \pm \sigma\, \bar{\epsilon}_n$
- Get their scores: $f(\bar{\theta}_n^{+})$, $f(\bar{\theta}_n^{-})$
- Build the update direction:

$$\bar{g}_i = \frac{1}{N} \sum_{n=1}^{N} \frac{f(\bar{\theta}_n^{+}) - f(\bar{\theta}_n^{-})}{2\sigma}\, \bar{\epsilon}_n$$

- Update:

$$\bar{\mu}_{i+1} \leftarrow \bar{\mu}_i + \alpha\, \bar{g}_i$$

Also called *mirrored sampling*

Refer to: Mirrored sampling in evolution strategies with weighted recombination (Auger et al., GECCO 2011)

Two-point estimator = centered finite differences

- Two-point estimator in antithetic ES = centered finite differences

$$\bar{g}_i = \frac{1}{N} \sum_{n=1}^{N} \underbrace{\frac{f(\bar{\mu}_i + \sigma\bar{\epsilon}_n) - f(\bar{\mu}_i - \sigma\bar{\epsilon}_n)}{2\sigma}}_{\text{Centered finite diff along } \epsilon_i} \bar{\epsilon}_n$$

- **Intuitively:** Use fitness differences to build an update, average across pairs

Evolution Strategies as a Scalable Alternative to Reinforcement Learning (Salimans et al., 2017)

One sampling strategy: Isotropic Gaussian ES

Repeat:

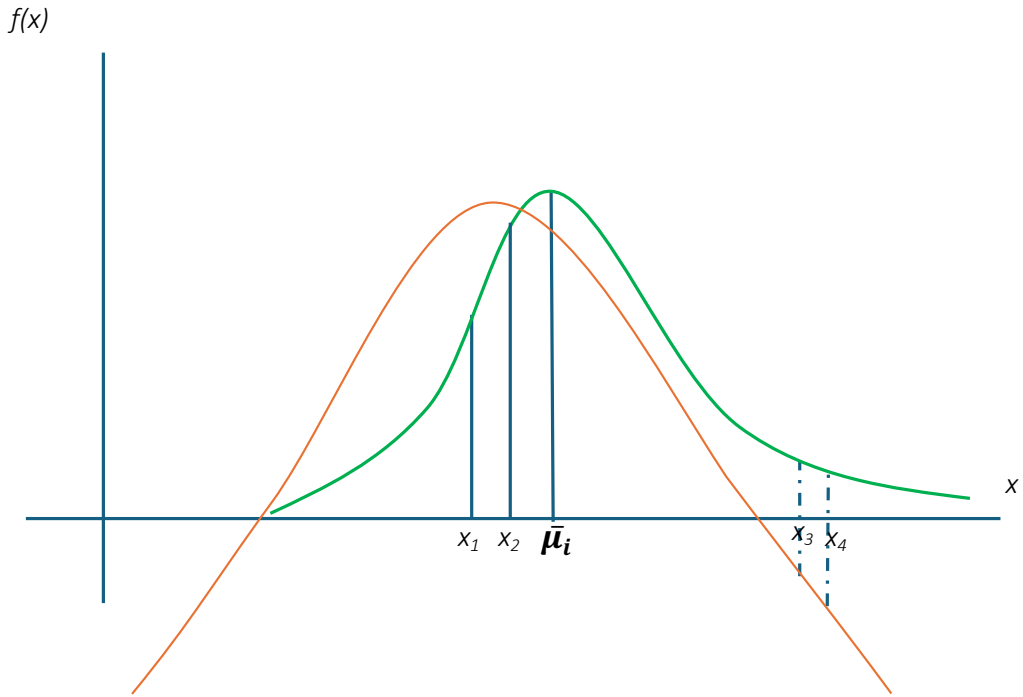
- Sample points in neighborhood of $\bar{\mu}_i$ to obtain population of size Λ

$$\bar{\theta}_n = \bar{\mu}_i + \sigma\,\bar{\epsilon}_n, \quad \bar{\epsilon}_n \sim \mathcal{N}(0, I), \quad n = 1, \ldots, \Lambda$$

- Pick top-k by fitness (elite set $\mathcal{E}$)
- Update mean toward elite average

$$\bar{\mu}_{i+1} \leftarrow \frac{1}{k} \sum_{\bar{\theta}_n \in \mathcal{E}} \bar{\theta}_n$$

Weighted averaging



Natural Evolution Strategies (NES)

We already sample parameters from a distribution p_ϕ

Key idea: Treat distribution parameters ϕ as optimization objective

$$J(\phi) = \mathbb{E}_{\bar{\theta} \sim p_\phi}[f(\bar{\theta})]$$

Use **log-derivative trick** to rewrite gradient of $J(\phi)$ without knowing ∇f:

$$\nabla_\phi J(\phi) = \mathbb{E}_{\bar{\theta} \sim p_\phi}\left[f(\bar{\theta}) \, \nabla_\phi \log p_\phi(\bar{\theta}) \right]$$

Update parameters: $\bar{\mu}_{i+1} = \bar{\mu}_i + \alpha \nabla J(\phi)$

Bound change in parameters: $|| \nabla \bar{\mu} || \leq \delta$

Source: Natural Evolution Strategies (Wiestra et al. 2014)

Constraining step sizes for safe updates

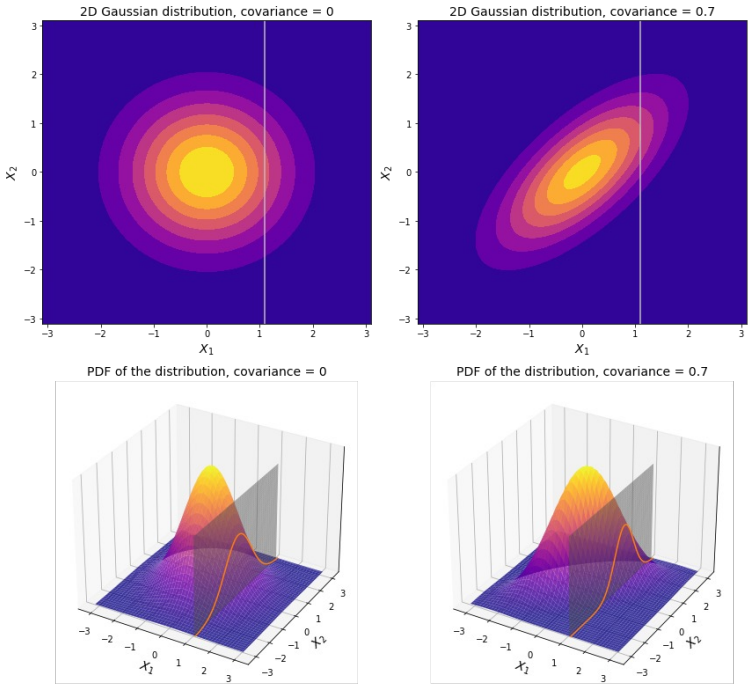
Solution: keep the next sampling distribution similar to current distribution and use steepest ascent in a KL trust-region:

$$\max_{\Delta\bar{\mu}} \nabla J(\bar{\mu}_i)^\top \Delta\bar{\mu} \quad \text{s.t.} \quad \mathrm{KL}\Big(p_{\bar{\mu}_i} \,\Big\|\, p_{\bar{\mu}_i+\Delta\bar{\mu}}\Big) \leq \delta$$

Closed form solution:

$$\Delta\bar{\mu} \propto \boxed{F(\bar{\mu}_i)}^{-1} \nabla J(\bar{\mu}_i)$$

Fisher information matrix F measures how **sensitive** distribution is to each parameter direction



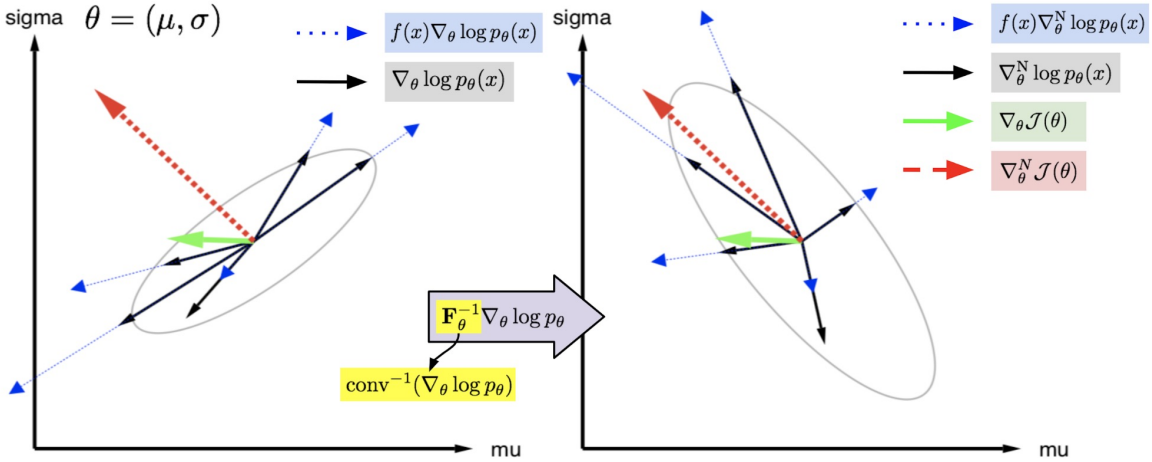
Source: Natural Evolution Strategies (Wiestra et al. 2014)

Natural gradient

Rescale gradients using how sensitive the distribution is to its parameters

$$\tilde{\nabla}_\phi J = F(\phi)^{-1} \nabla_\phi J$$

Natural gradient step



Plain Gradient **Natural Gradient**

Source: Natural Evolution Strategies (Wiestra et al. 2014)

TRPO – same idea, applied to policies

- Same “safe step” idea, but the distribution is **actions** from the policy

- TRPO constraint:

$$\max_{\Delta\bar{\theta}} \nabla_{\bar{\theta}} J(\bar{\theta}_i)^\top \Delta\bar{\theta} \;\text{ s.t. }\; \mathbb{E}\Big[\mathrm{KL}\big(\pi_{\bar{\theta}_i}(\cdot|s) \,\|\, \pi_{\bar{\theta}_i+\Delta\bar{\theta}}(\cdot|s)\big)\Big] \leq \delta$$

$$\Rightarrow \quad \Delta\bar{\theta} \;\propto\; \boxed{F_\pi(\bar{\theta}_i)^{-1} \nabla_{\bar{\theta}} J(\bar{\theta}_i)}$$

Uses natural gradient updates like NES

Example: ES as an alternative to RL (OpenAI-ES)

- Treat policy parameters as a vector $\bar{\mu}$ –**Representation**
- Sample perturbations $\bar{\mu} \pm \sigma\epsilon$ – **Population**
- Evaluate episode return = fitness – **Selection**
- Use the **two-point estimator** to update the mean $\bar{\mu}$ – **Evolutionary Operators**

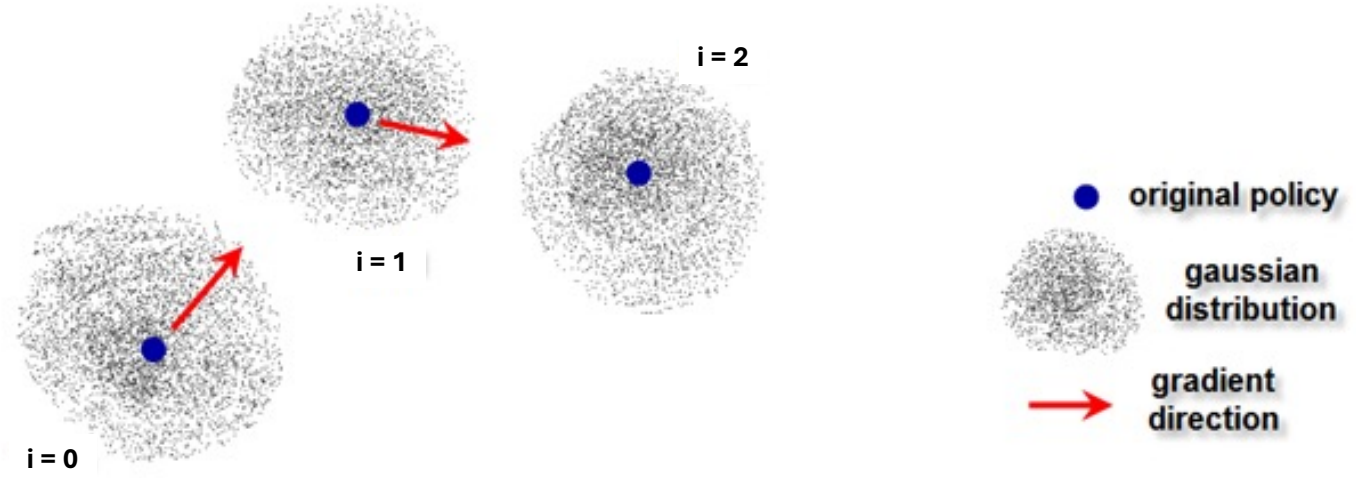
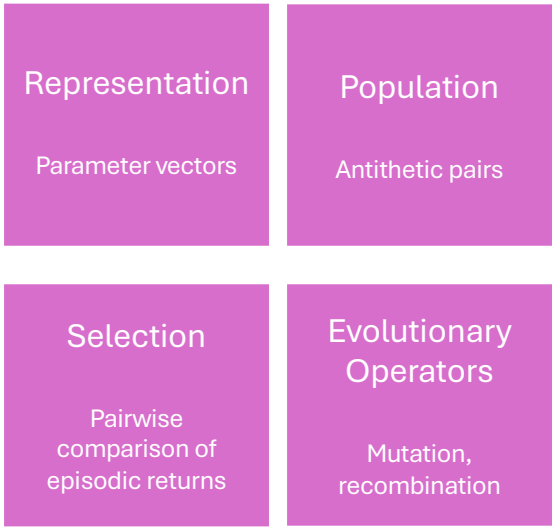


Image: https://medium.com/swlh/evolution-strategies-844e2694e632

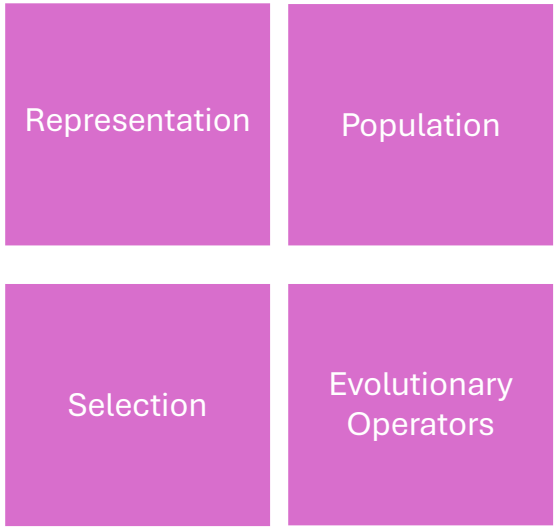
OpenAI-ES from an EA perspective

- **Representation**: parameter vector $\bar{\mu}$
- **Population**: $\{\bar{\mu}_i \pm \sigma\bar{\epsilon}_n\}_{n=1}^{N}$
- **Selection**: comparison of episodic returns $f^+,\ -f^-$
  - No elite set
- **Evolutionary Operators:**
  - Mutation: $\pm\sigma\bar{\epsilon}_n$
  - Recombination: average contributions to $\bar{g}_i$



Evolutionary algorithms: 4 key parts

- **Population** of "candidates" (search points)
- **Representation** of each candidate
- **Selection** of best candidates
  - Elitism
  - Methods such as MAP-Elites
- **Evolutionary operators** to generate new candidates



Evolutionary operators

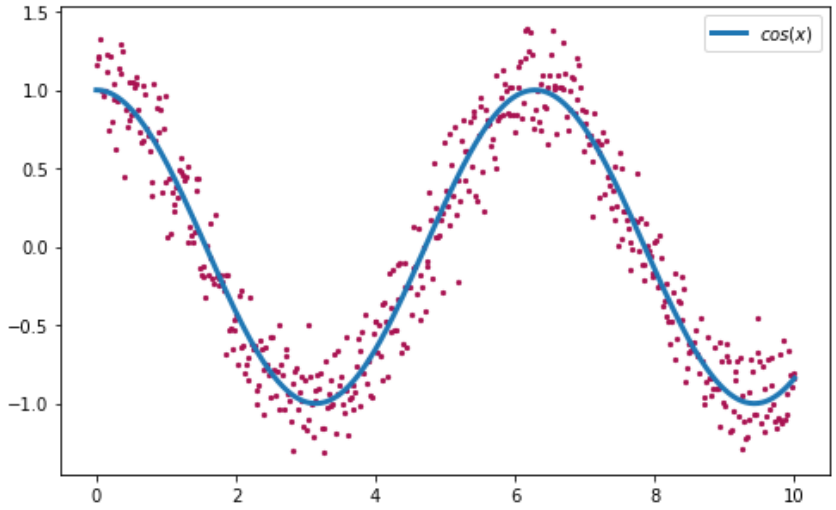
Many ways of carrying over information from a candidate or combining information from multiple candidates

Two common operations:

- **Mutation:** explores region *near candidate*
- **Recombination:** explores region *between multiple candidates* such as by using convex combination of vectors
  - **Crossover:** explores region *between two candidates*

Example: polynomial curve fitting

- Given a set of points (x, y) in dataset, find the polynomial that best fits given points by minimizing the sum of distance between each point and curve
- Example: Generate data by adding Gaussian noise to $f(x) = cos(x)$
- We can use:
  - Polynomial regression
  - Differential evolution (DE)
- Objective: minimize MSE



$$f_{model}(\mathbf{w}, x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + w_5 x^5$$

Differential Evolution (DE)

- **Population** = set of vectors of coefficients $\boldsymbol{w}$
- **In each generation:**
  - **Mutation:** pick three individuals a, b, c and create mutant vector v

$$v = a + f(b - c)$$

  - **Crossover:** mix v with the current target $\mathbf{w}_i$ to obtain u

$$u_j = \begin{cases} v_j, & \text{if } \text{rand}() < \text{CR} \\ w_{i,j}, & \text{otherwise} \end{cases}$$

  - **Selection:** evaluate both u and $\mathbf{w}_i$ with MSE, keep the better one

$$\mathbf{w}_i \leftarrow \begin{cases} u, & \text{if } \text{MSE}(u) < \text{MSE}(\mathbf{w}_i) \\ \mathbf{w}_i, & \text{otherwise} \end{cases}$$

A tutorial on Differential Evolution with Python

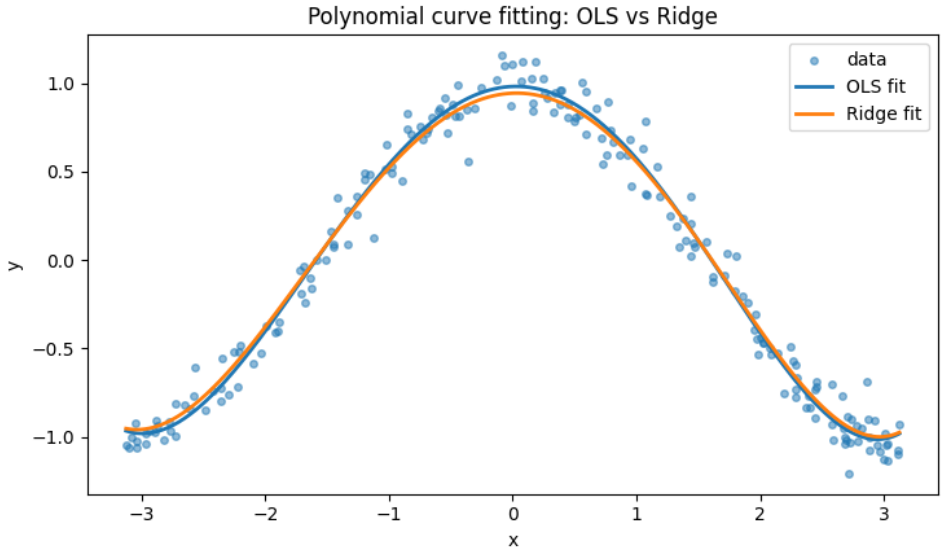
Baseline demo: Polynomial regression

- Ordinary least squares (OLS):

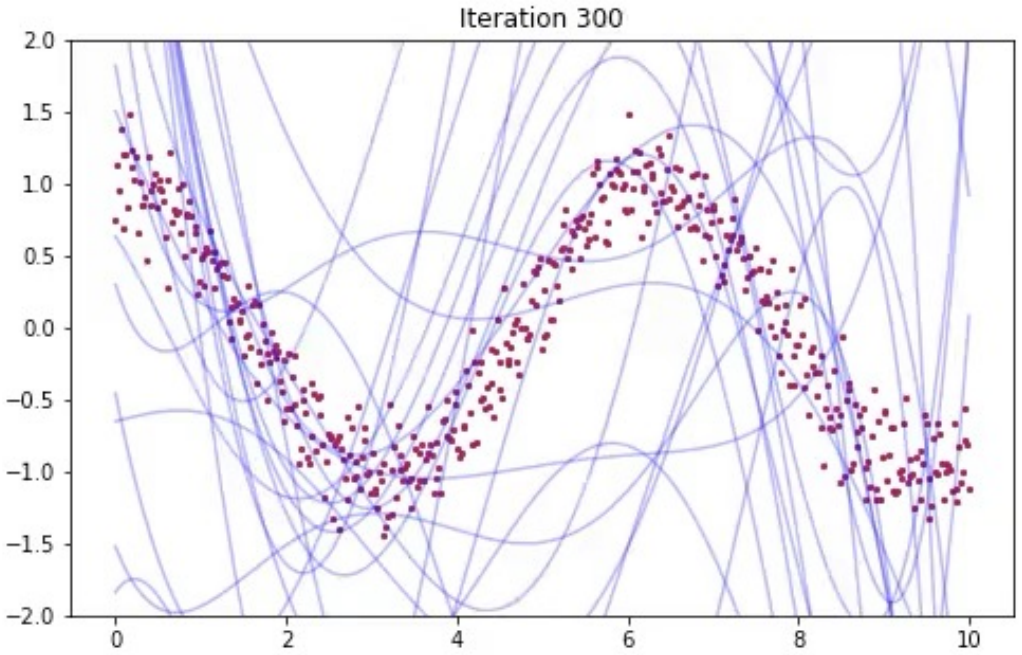
$$\hat{\mathbf{w}} = (X^\top X)^{-1} X^\top \mathbf{y}$$

- Ridge regularization:

$$\hat{\mathbf{w}}_\lambda = (X^\top X + \lambda I)^{-1} X^\top \mathbf{y}$$



Demo: Differential Evolution

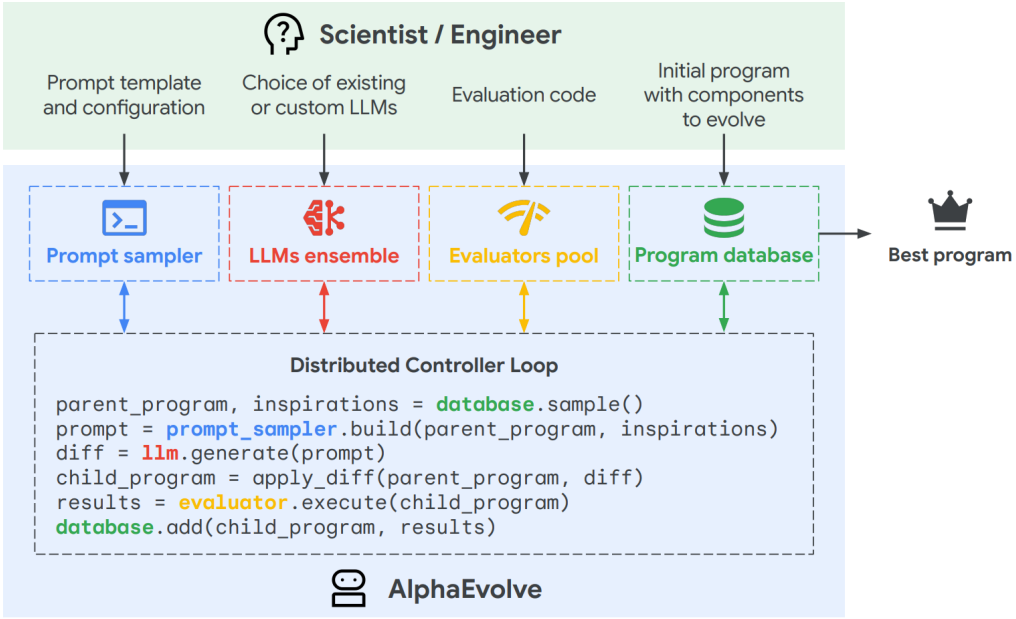


A tutorial on Differential Evolution with Python

Evolving neural networks

- **Neuroevolution:** Optimize neural networks without backprop
- Two modes:
  - **Evolve only weights** for a fixed architecture
    - Works when the environment is non-differentiable (physics engine jitter, human feedback, etc.)
  - **Evolve both topology and weights**
    - The network's structure itself changes over time
- Examples: NEAT, etc.

Example: AlphaEvolve (I)

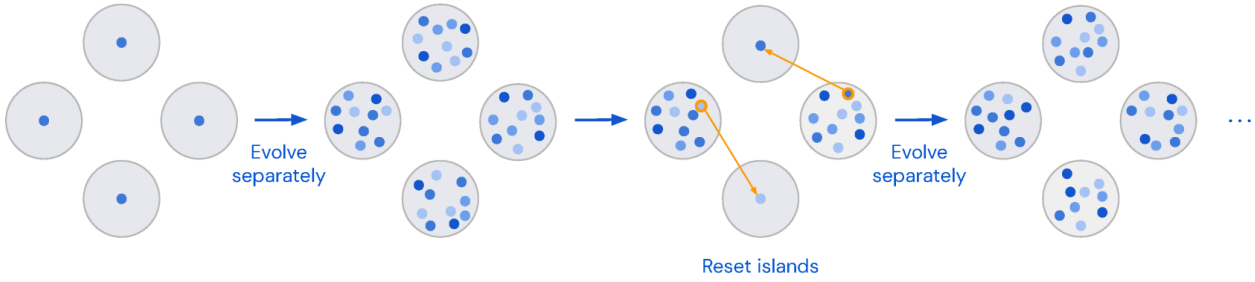


- Evolve entire files in *any* language
- **Database:** Pool of candidate programs
  - Every candidate is a *full source file* with versioned metadata and vector of fitness scores
- LLM **samplers** = mutation/crossover engine
  - Sampler prompt = top-k elites + diff context + task spec
- **Evaluators:** distributed evaluation of candidates
  - Supports *a vector of fitness metrics*: ⟨performance, resources, provability…⟩

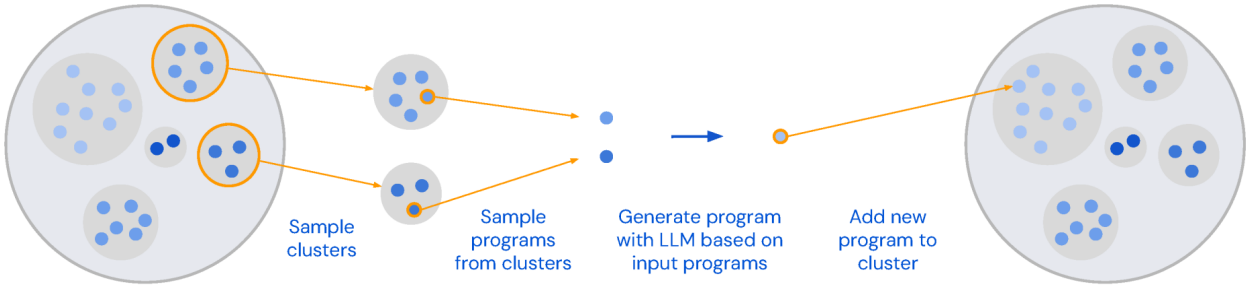
Source: AlphaEvolve: A coding agent for scientific and algorithmic discovery (Novikov et al., 2025)

Example: AlphaEvolve (II)

- *Islands* of programs evolved separately:

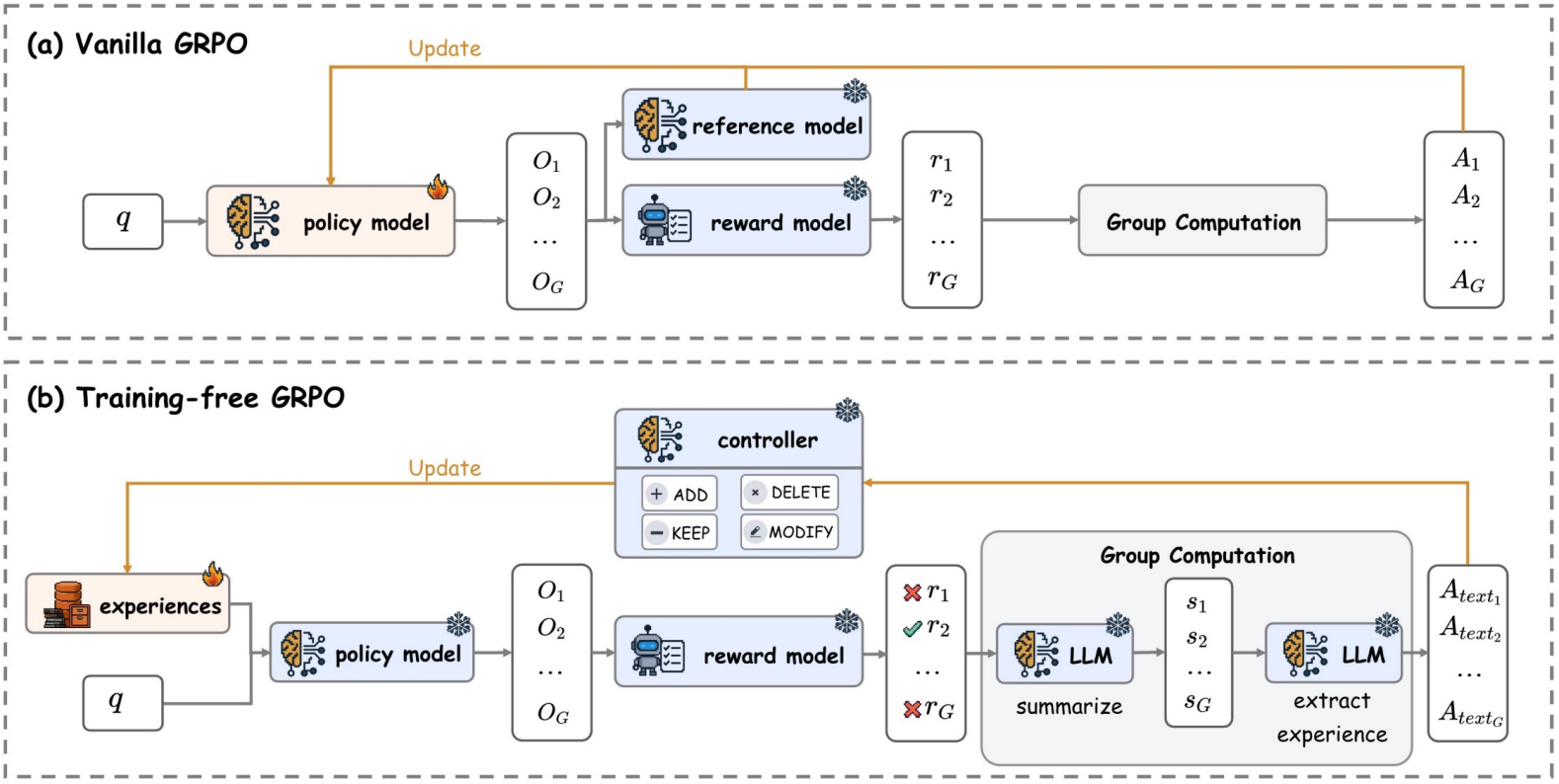


- Generating new offspring:



Mathematical discoveries from program search with large language models (Romera-Parades et al. 2023)

Example: Training Free GRPO



Training-Free Group Relative Policy Optimization (Cai et al., 2025)

Example: Training Free GRPO

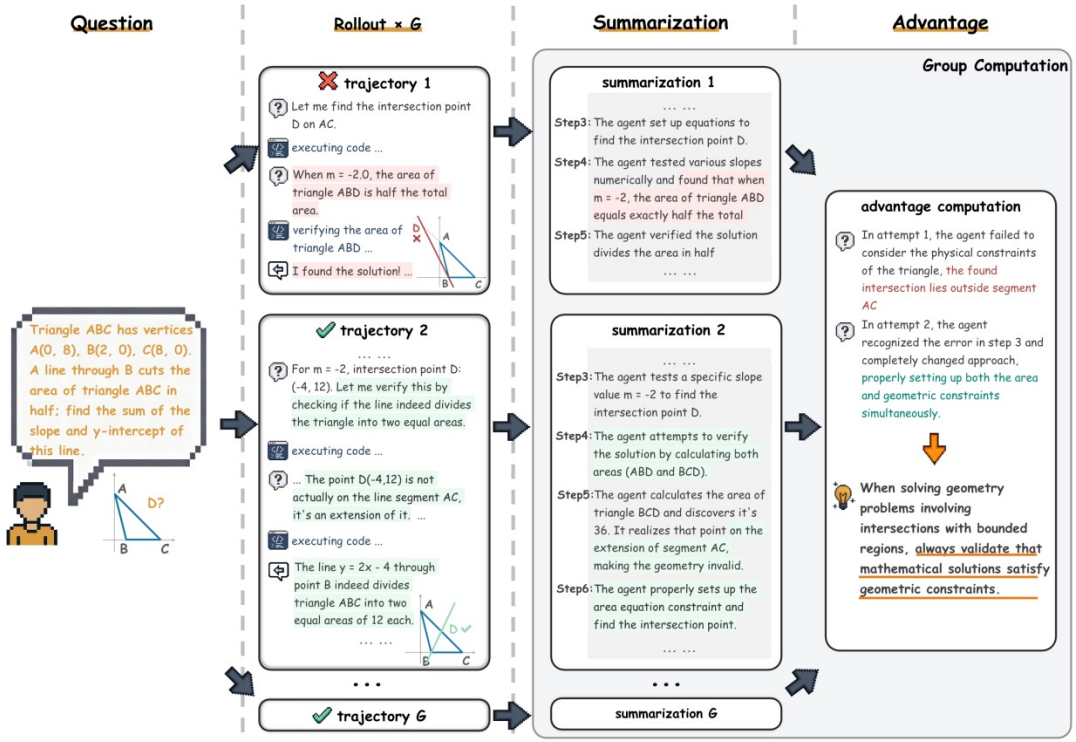


Figure 3. Example of a Training-Free GRPO learning step.

Training-Free Group Relative Policy Optimization (Cai et al., 2025)

Takeaways