CS395T: Foundations of Machine Learning for Systems Researchers Fall 2025

**Lecture 12: Evolution** 

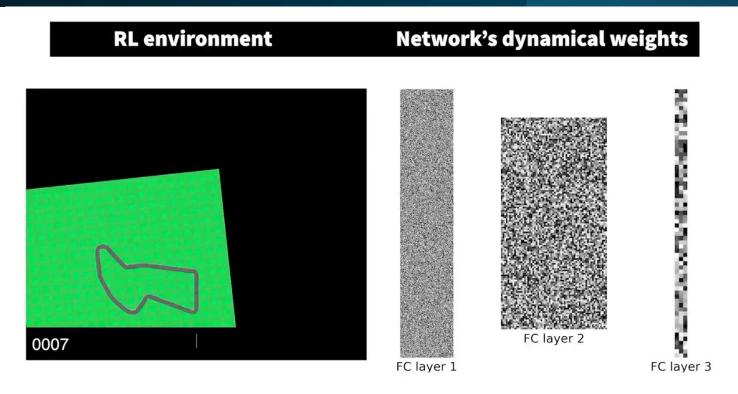
Lain (Zelal) Mustafaoglu



#### 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 Al Agent Design, *An MIT Press Book by* <u>Sebastian Risi, Yujin Tang, David Ha, and Risto Miikkulainen https://neuroevolutionbook.com/demos/</u>

#### Demo (II)

```
def __init__(self, mode, init_ing, coming, mypers).
self.hypers = hypers
                  super().__init__(mode=mode, init_rng=init_rng, config=config)
   8  def _get_optimizer(self) -> optax.GradientTransformation:
9  ""Returns optimizer."""
+ 10  b1 = 0.9
                  b2 = 0.999
                   self.hypers.learning_rate, weight_decay=self.hypers.weight_decay
self.hypers.learning_rate, weight_decay=self.hypers.weight_decay, b1=b1,
        77 def _get_init_fn(self) -> jax.nn.initializers.Initializer:
88 """Returns initializer function."""
                  scale = self.hypers.init_scale
                 # Initialize with a smaller scale to encourage finding low-rank solutions return initializers.normal(\theta + 1j * \theta, scale * \theta.1, jnp.complex64)
+ 21 return initializers.normal(0 + 1j * 0, scale * 0.2, jnp.complex64)
- 22 def _linear_schedule(self, global_step, start: float = 0.0, end: float = 1.0):
+ 24 def _linear_schedule(self, global_step, start: float = 0.0, end: float = 0.0):
               frac = 1 - global_step / self.config.training_steps
return (start - end) * frac + end
               @functools.partial(jax.jit, static_argnums=0)
               def _update_func(
                  self,
                   decomposition: tuple[jnp.ndarray, jnp.ndarray, jnp.ndarray],
                     opt_state: optax.OptState,
      global_step: jnp.n
35 rng: jnp.ndarray,
36 ) -> tuple[
37
                     global_step: jnp.ndarray,
                   tuple[jnp.ndarray, jnp.ndarray, jnp.ndarray],
optax.OptState,
                    jnp.ndarray,
        40 l:
""A single step of decomposition parameter updates.""
                 # Compute loss and gradients.
loss, grads = jax.value_and_grad(
                       lambda decomposition, global_step, rng: jnp.mean(
self._loss_fn(decomposition, global_step, rng)
                 ) | (decomposition, global_step, rng) | # When optimizing real-valued functions of complex variables, we must take # the conjugate of the gradient.
grads = jax.tree_util.tree_map(lambda x: x.conj(), grads)
                  # Gradient updates.
updates, opt_state = self.opt.update(grads, opt_state, decomposition)
                  decomposition = optax.apply_updates(decomposition, updates)
                  # Add a small amount of gradient noise to help with exploration
                                                   Iteration 15
```

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

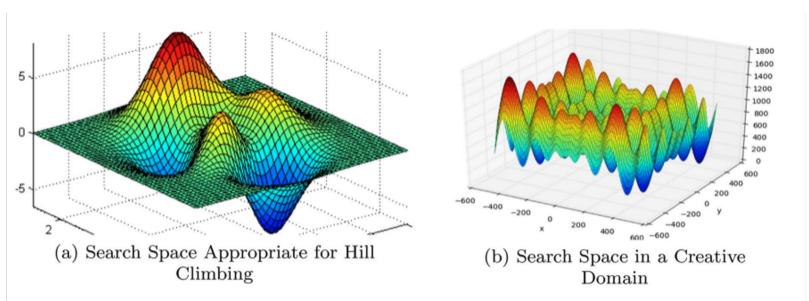
$$x_{t+1} = x_t + \alpha \nabla f(x_t)$$

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



Source: Creative Al Through Evolutionary Computation: Principles and Examples



Approximate derivatives with differences, which only require function evaluation

Forward differences:

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

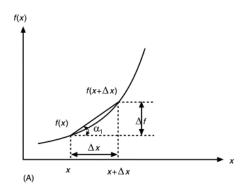
Backward differences:

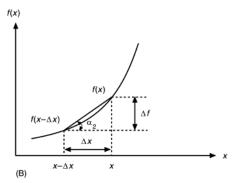
$$\frac{\mathrm{d}f}{\mathrm{d}x} \approx \frac{\mathrm{f}(\mathrm{x}) - f(x - \Delta\mathrm{x})}{\Delta\mathrm{x}}$$

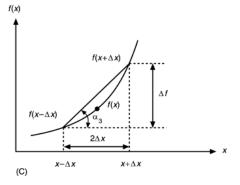
Centered differences:

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

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







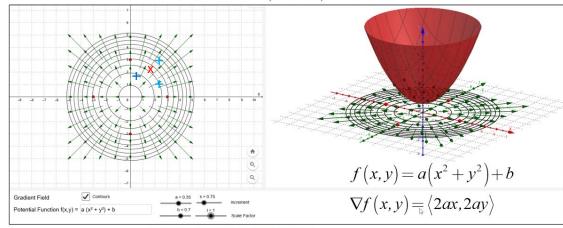
## One solution for higher dimensions

16. Vector Calculus Gradient Vector Fields

#### **Gradient Vector Fields**

The gradient function of a function of two real variables is a 2D Vector Field.

$$\overrightarrow{F}(x,y) = grad(f(x,y)) = \nabla f(x,y)$$



GeoGebra: https://www.geogebra.org/classic/zkdfapfk Author: Andreas Linder, Jack Jackson

Jack Le Jackson II, PhD.

73

Jack.Jackson@UAFS.edu

#### 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 E)
  - · Referred to as fitness score
- Update search point to elite average

$$ar{\mu}_{i+1} \; \leftarrow \; rac{1}{k} \; \sum_{ar{ heta}_n \in \mathcal{E}} ar{ heta}_n$$

#### One step further: antithetic evolutionary strategies (ES)

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

$$ar{g}_i = rac{1}{N} \sum_{n=1}^N rac{f(ar{ heta}_n^+) - f(ar{ heta}_n^-)}{2\sigma} \, ar{\epsilon}_n$$

Update:

$$ar{\mu}_{i+1} \leftarrow ar{\mu}_i + lpha \, ar{g}_i$$

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

$$ar{g}_i = rac{1}{N} \sum_{n=1}^N rac{f(ar{\mu}_i + \sigma ar{\epsilon}_n) - f(ar{\mu}_i - \sigma ar{\epsilon}_n)}{2\sigma} ar{\epsilon}_n$$

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

#### One sampling strategy: Isotropic Gaussian ES

#### Repeat:

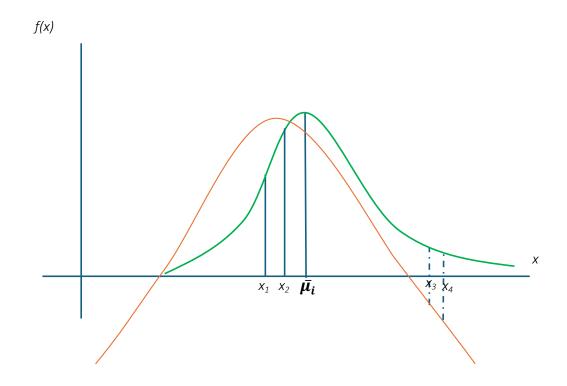
• Sample points in neighborhood of  $\bar{\mu_i}$  to obtain population of size  $\Lambda$ 

$$ar{ heta}_n \; = \; ar{\mu}_i \; + \; \sigma \, ar{\epsilon}_n, \; \; ar{\epsilon}_n \sim \mathcal{N}(0,I), \; n = 1,\ldots,\Lambda$$

- Pick top-k by fitness (elite set E)
- Update mean toward elite average

$$ar{\mu}_{i+1} \leftarrow rac{1}{k} \sum_{ar{ heta}_n \in \mathcal{E}} ar{ heta}_n$$

## Weighted averaging



### Natural Evolution Strategies (NES)

We already sample parameters from a distribution  $p_{oldsymbol{\phi}}$ 

**Key idea:** Treat distribution parameters  $\phi$  as optimization objective

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

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

$$abla_\phi J(\phi) \ = \ \mathbb{E}_{ar{ heta}\sim p_\phi} \Big[ f(ar{ heta}) \ 
abla_\phi \log p_\phi(ar{ heta}) \Big]$$

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

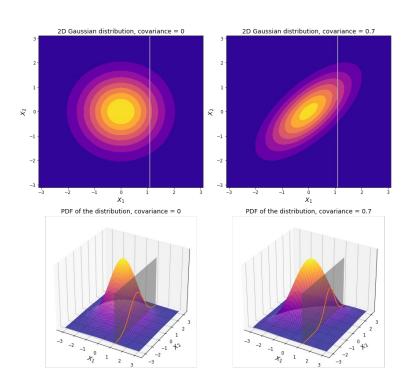
**Solution:** keep the next sampling distribution similar to current distribution and use steepest ascent in a KL trust-region:

$$\max_{\Deltaar{\mu}} \ 
abla J(ar{\mu}_i)^{\! op} \Deltaar{\mu} \quad ext{s.t.} \quad ext{KL} \Big( p_{ar{\mu}_i} \ ig| \ p_{ar{\mu}_i + \Deltaar{\mu}} \Big) \ \leq \ \delta$$

#### **Closed form solution:**

$$\Deltaar{\mu} \, \propto \, ar{F(ar{\mu}_i)}^{-1} \, 
abla J(ar{\mu}_i)$$

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



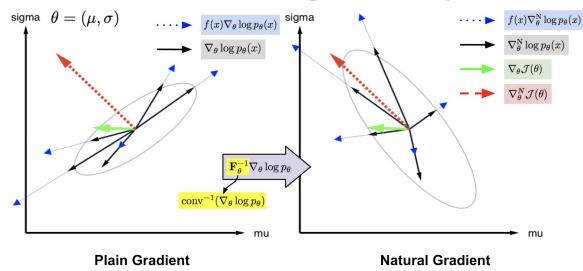
Source: Natural Evolution Strategies (Wiestra et al. 2014)

#### Natural gradient

Rescale gradients using how sensitive the distribution is to its parameters

$$ilde{
abla}_{\phi} J \ = igg| F(\phi)^{-1} \, 
abla_{\phi} J$$

#### Natural gradient step



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:

$$egin{array}{l} \max_{\Deltaar{ heta}} \ 
abla_{ar{ heta}} J(ar{ heta}_i)^{\! op} \Deltaar{ heta} \ ext{ s.t. } \mathbb{E}ig[ ext{KL}ig(\pi_{ar{ heta}_i}(\cdot|s) \, \| \, \pi_{ar{ heta}_i+\Deltaar{ heta}}(\cdot|s)ig)ig] \leq \delta \ \ \Rightarrow \ \Deltaar{ heta} \propto ig[F_\pi(ar{ heta}_i)^{-1} \, 
abla_{ar{ heta}} J(ar{ heta}_i) ig] \end{array}$$

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**

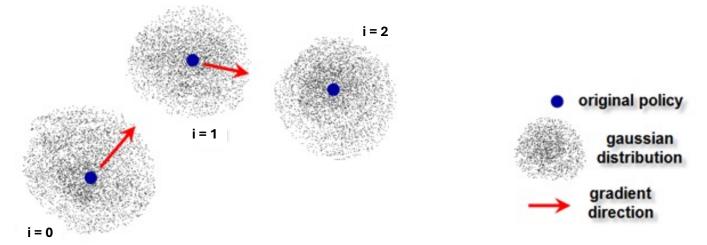


Image: https://medium.com/swlh/evolution-strategies-844e2694e632

## 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  $\bar{q}_i$

Representation

Parameter vectors

Population

Antithetic pairs

Selection

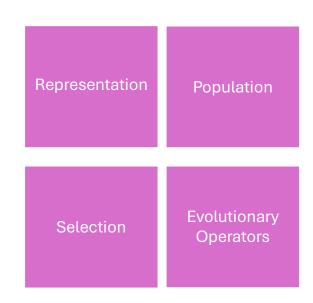
Pairwise comparison of episodic returns

Evolutionary Operators

Mutation, recombination

#### 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**

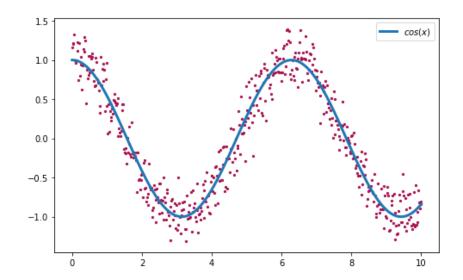
Many ways of carrying over information from a candidate or combining information from multiple candidates

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 **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

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

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

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

A tutorial on Differential Evolution with Python

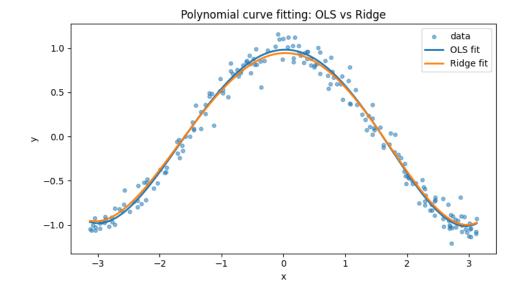
## Baseline demo: Polynomial regression

• Ordinary least squares (OLS):

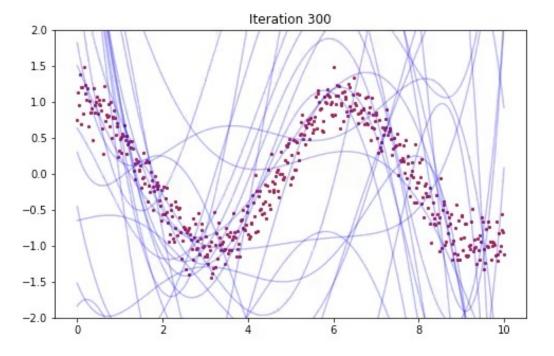
$$\hat{\mathbf{w}} = (X^\top X)^{-1} X^\top \mathbf{y}$$

• Ridge regularization:

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



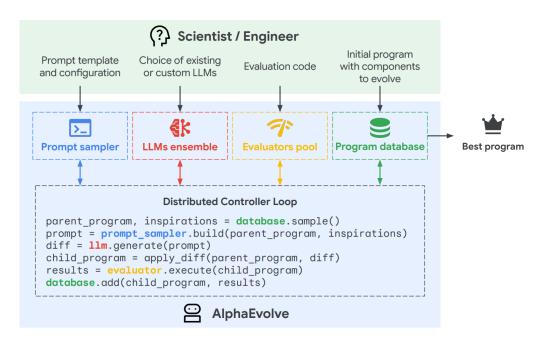
# Demo: Differential **Evolution**



#### **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)

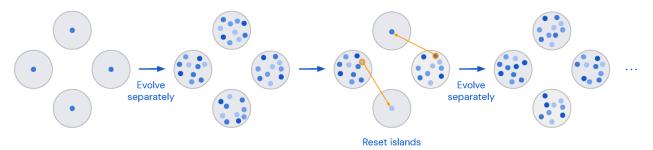


- Evolve entire files in any language
- **Database:** Pool of candidate programs
  - Every candidate is a full source file with versioned metadata and vector of fitness scores
- LLM **samplers** = mutation/crossover engine
  - Sampler prompt = top-k elites + diff context + task spec
- **Evaluators:**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

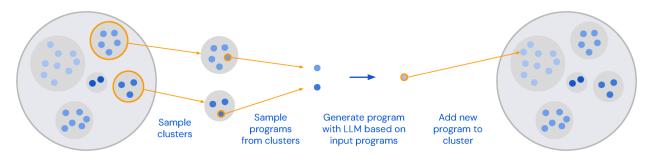
Source: AlphaEvolve: A coding agent for scientific and algorithmic discovery (Novikov et al., 2025)

## Example: AlphaEvolve (II)

• *Islands* of programs evolved separately:

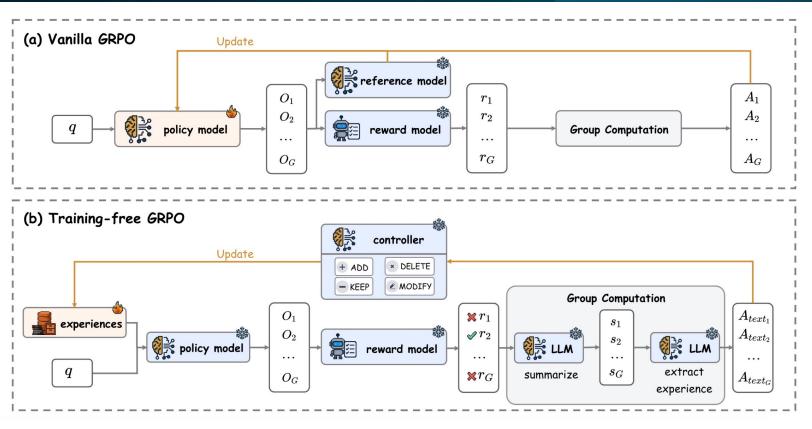


• Generating new offspring:



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**

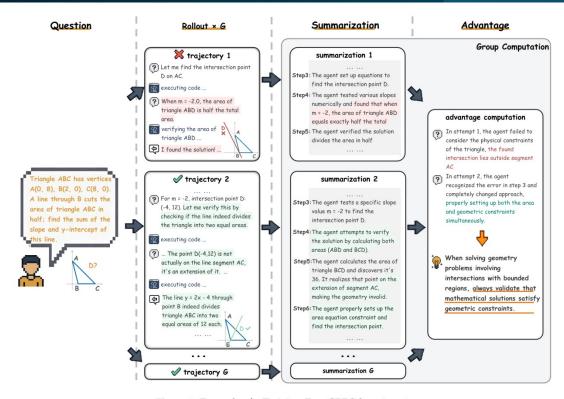


Figure 3. Example of a Training-Free GRPO learning step.

## Takeaways

