DESIGN PRINCIPLES OF MODERN LEARNING SYSTEMS

Closing the Loop

How priors, data, feedback, exploration, and reinforcement turn experience into specialized behavior.

ANDREI AFONIN

WHY THIS MATTERS

Industry examples

Generate, evaluate, improve cycle used in the industry

RESEARCH SYSTEMS

Google DeepMind

AlphaZero · AlphaTensor · AlphaProof

From games to discovery

Search and verifiers create stronger training targets.

FRONTIER PRODUCTS

OpenAI

ChatGPT - RLHF + Code/Math verifiers

Anthropic

Claude - RLHF + Code/Math verifiers

EMERGING ECOSYSTEM

Harmonic

Aristotle · Lean-verified mathematics

Black Forest Labs

Robot in the real world interaction

Luma AI

Text-to-image(video) model with a verifier on top

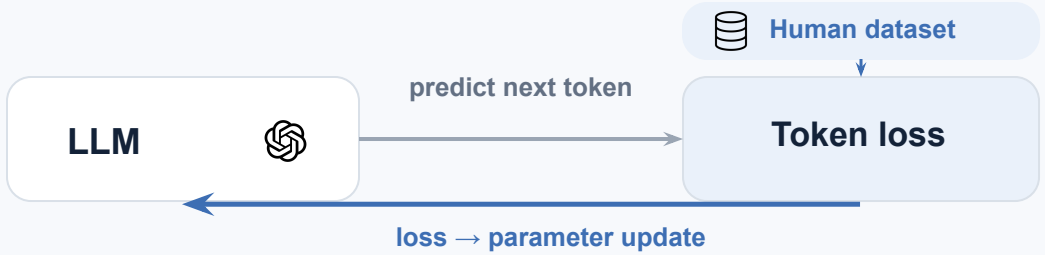
More: Black Forest Labs and Luma AI

A MODEL-LEVEL VIEW

Feedback loops for ChatGPT

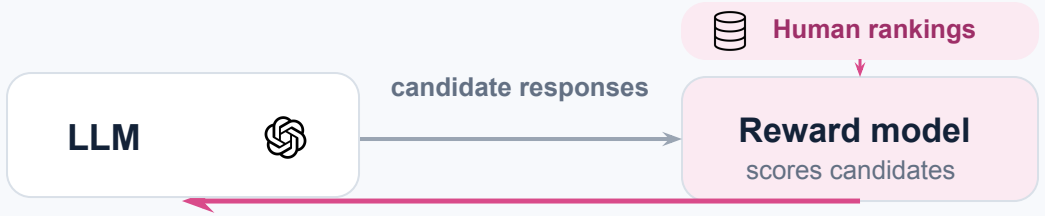
01 · TRAINING

Pretraining from human data



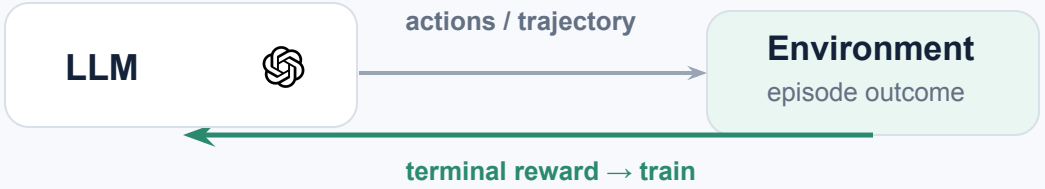
02 · TRAINING

RLHF through a reward model



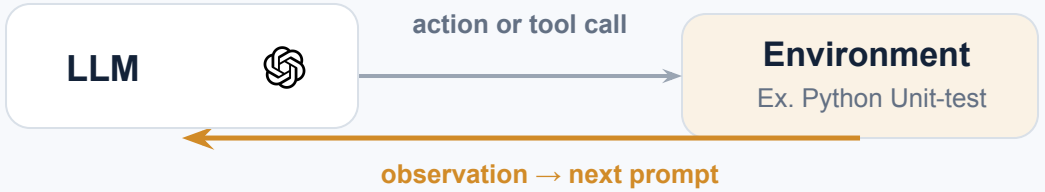
03 · TRAINING

Reinforcement from the environment



04 · INFERENCE

Inference-time interaction



Feedback changes the current context—not the model parameters.

THREE ROUTES

We can close the loop with increasingly direct feedback

01

Collect better data

Supervised / self-supervised

Expose the learner to more informative examples.

High information density
The most sample efficient
Expensive and **"Human" Ceiling**

02

Model human judgment

RLHF / preference learning

Use a learned proxy for what people prefer.

Scalable alignment signal
Proxy can be misspecified
Harder to scale

03

Act in the environment

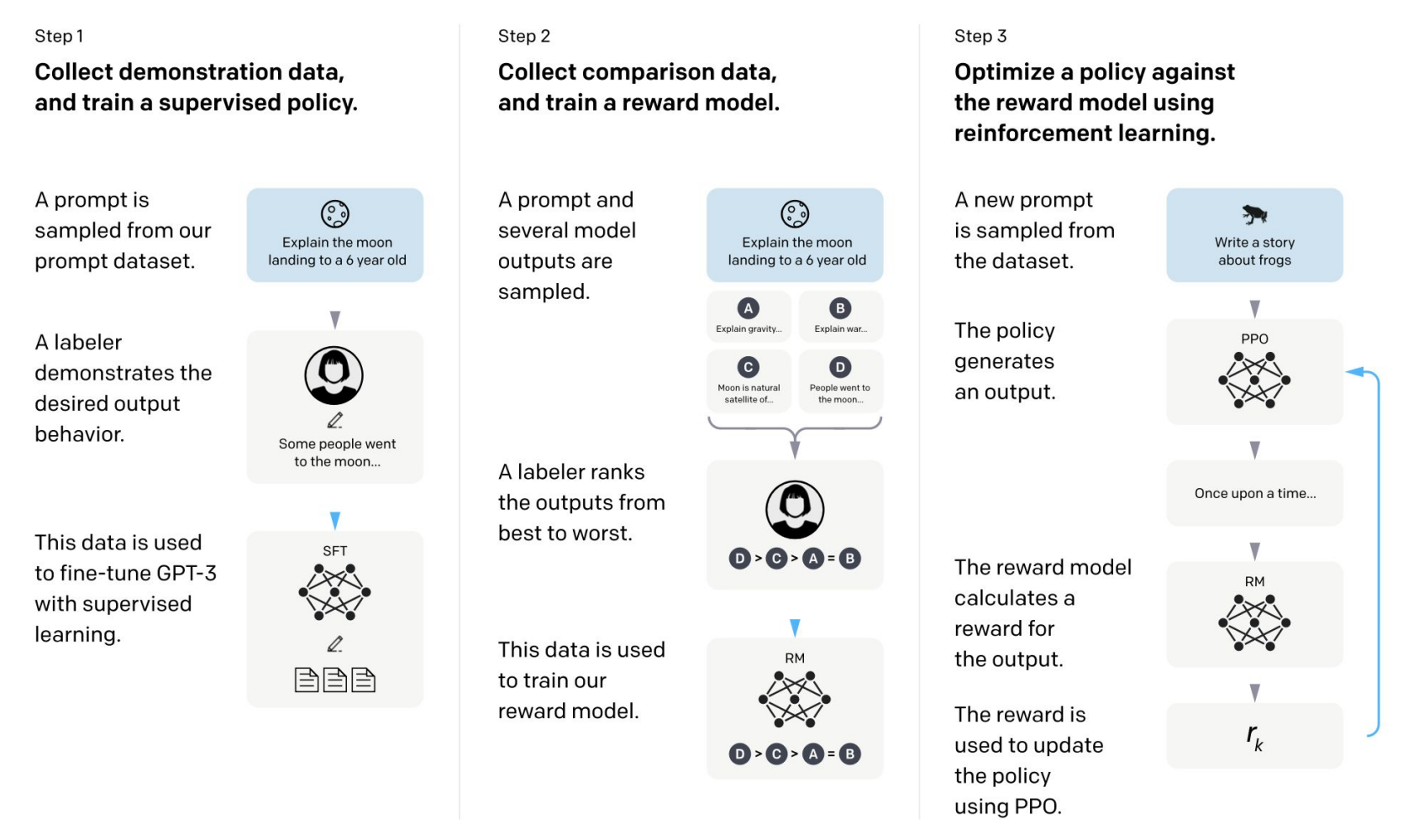
Direct reinforcement learning

Generate trajectories and learn from outcomes.

Ability to improve beyond human perf.
Scale
Often lowest sample efficiency
Limited set of tasks

SuperHuman performance!

ChatGPT Training Loops



Read more: InstructGPT

INFORMATION EFFICIENCY

Reinforcement learning can spend most of its compute on noise

SUPERVISED / PRETRAINING

one example → dense signal

A labeled example reveals its target in a single pass. More diversity usually means collecting or curating more examples.

REINFORCEMENT LEARNING

one trajectory → narrow evidence

A random interaction exposes only one path. Most paths may say little about the behavior we want.

THE CORE PATTERN

Learning setup

1 PRIOR

What the learner already believes or can generate

→

2 EXPERIENCE

Data, actions, trajectories, or candidate solutions

→

3 FEEDBACK

Labels, preferences, rewards, tests, or outcomes

→

4 UPDATE

Reinforce useful behavior; suppress harmful behavior

The update becomes the next prior—and the cycle compounds.

EXPLORATION

Exploration decides what data the learner will see

Exploration is not merely randomness.
It is the policy for selecting experience.

DIVERSITY

Which states, problems, and trajectories enter the training distribution?

INFORMATION

How much useful signal does each unit of compute reveal?

EXPLORATION BUDGETS

Learning must balance new worlds with deeper paths

MORE WORLDS ↑

Broad but shallow

Many environments; little useful experience extracted from each.

Broad and deep

Coverage plus informative search—the generalization target.

Narrow and shallow

Low cost, low signal, and little chance to discover rare behavior.

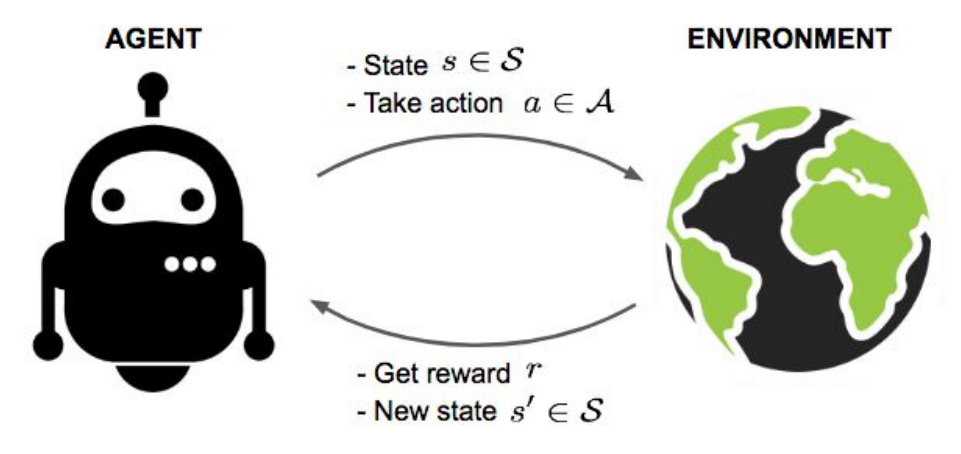
Deep but narrow

Strong local competence with a high risk of overfitting.

MORE SEARCH WITHIN EACH WORLD →

Reinforcement amplifies behavior that the model can already sample

sample → verify → reinforce → sample more often



The prior determines what can be explored

- Good trajectories must already have non-trivial probability.
- Reinforcement redistributes probability mass toward rewarded behavior.
- Moving beyond the prior requires new search, new data, or a new representation.

TEST-TIME COMPUTE

Search pays when verification is easier than generation

GENERATE

Explore many candidate answers or action sequences

→

VERIFY

Select the best candidate cheaply

STRONG FIT

Chess, Go, proofs, code with tests: outcomes are comparatively easy to check.

WEAKER FIT

Open-ended perception or retrieval: evaluation can be as hard as producing the answer.

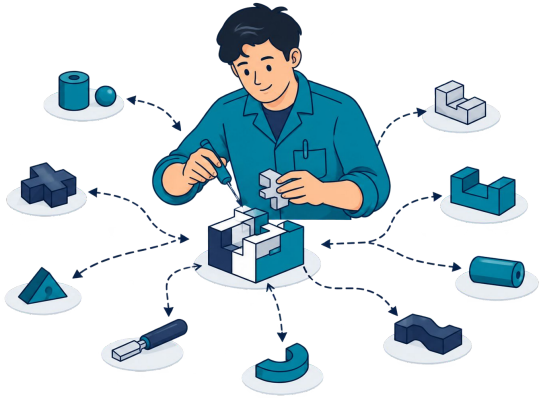
STARTING POINTS

“Zero” and pretrained systems trade bias for efficiency

START FROM ZERO

Less inherited bias. More expensive discovery.

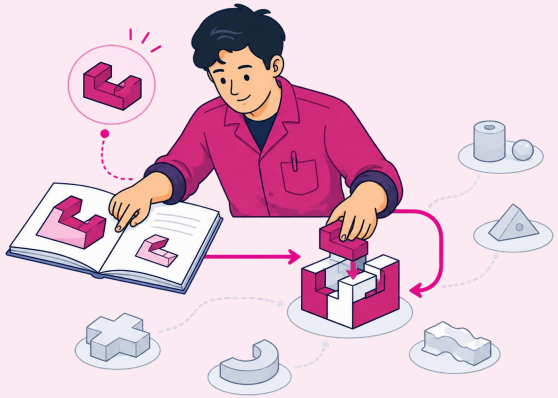
- The loop can discover non-human strategies.
- Sparse feedback makes early learning fragile.



START FROM PRETRAINING

Faster exploration. A narrower frontier.

- Useful behavior is sampled much sooner.
- Human-generated data shapes what looks likely.



The same prior that makes exploration possible can also make unfamiliar solutions hard to sample.

From Zero to Banned

Key Milestones Achieved:

1. Beating **MCTS with random rollouts**.
2. Beating **MCTS guided by a supervised policy** trained on data from top human players.
3. **Small model** defeating many top players on Othello Quest, achieving **2250 ELO** and **49th place out of ~45k**.
4. **Large model** beating **Egaroucid** at **level 10** (SuperHuman level).

Read more: From Zero to Banned: Make Othello AI at home

How AlphaZero improves through self-play

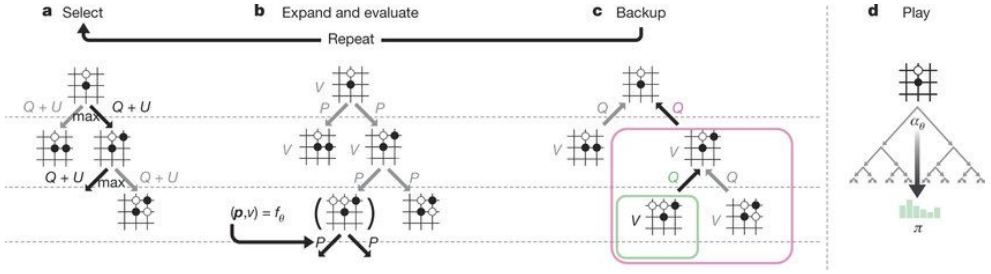
Neural predictions guide search; search produces stronger learning targets.

AlphaZero Training Loop

1. Start with a neural network that outputs a policy vector and a value scalar for any game state.
2. Use it inside MCTS. Search explores with the network outputs, returns an improved policy vector, selects a move, and stores that policy in a replay buffer.
3. Repeat until the game ends, then record the final reward (win, loss, or draw).
4. Train the model to distill the improved policy and predict the true final reward for every recorded state.
5. Return to step 1 with the updated network.

The loop alternates policy improvement (MCTS) with policy evaluation (learning).

INSIDE EACH MCTS SEARCH

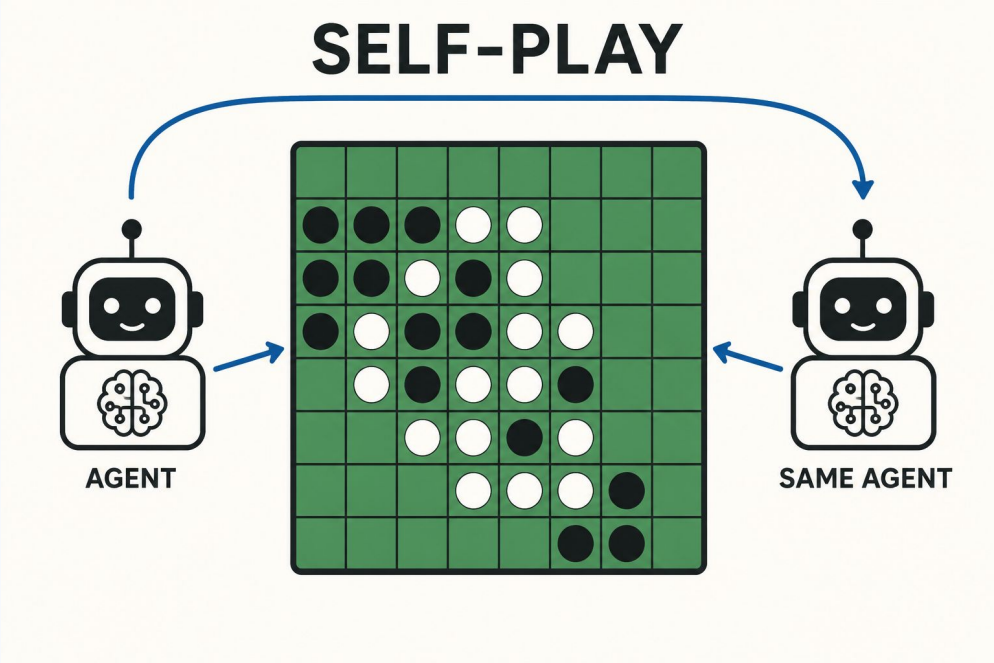


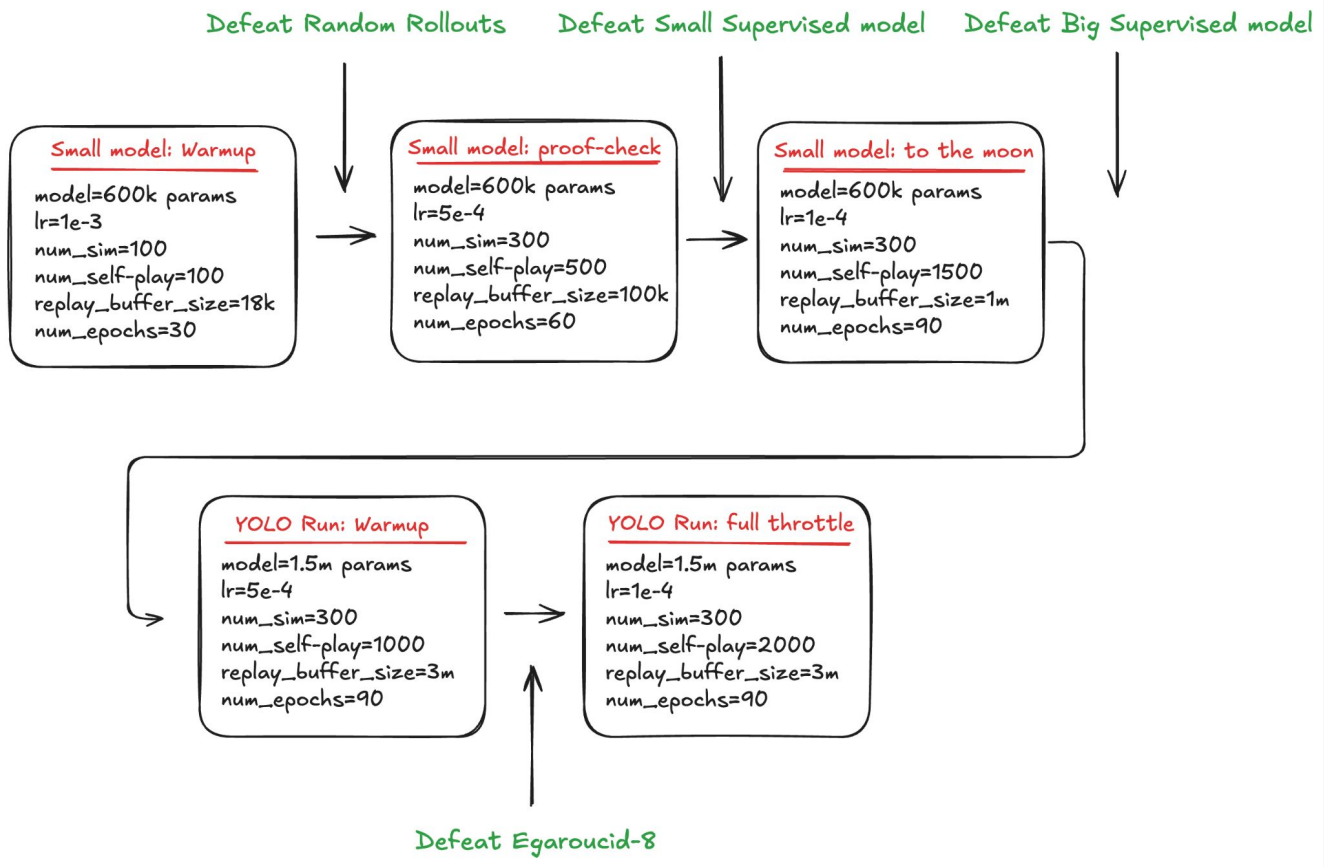
Select → expand & evaluate → back up values → play from the improved visit-count policy

Search makes the policy target stronger. Learning makes the next search stronger.

Read more: Notes on AlphaZero

Training: High-level





Training: Code time

```python
for _ in range(num_iterations):
    # 1. Self-play
    data, _, _ = play(
        best, best, num_self_play_games, collect=True
    )
    replay_buffer.extend(data)

    # 2. Train candidate
    network.train()

    for _ in range(num_training_steps):
        state, target_p, target_v = sample_batch(
            replay_buffer
        )

        logits, value = network(state)
        log_p = F.log_softmax(logits, dim=-1)

        policy_loss = -(target_p * log_p).sum(-1).mean()
        value_loss = F.mse_loss(value, target_v)
        loss = policy_loss + value_loss

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    # 3. Compare models
    _, best_win, network_win = play(
        best, network, 50
    )

    if network_win > best_win + 10:
        best = snapshot(network)
```

```python
def play(model_a, model_b, games, collect=False):
    data = []
    wins = {"A": 0, "B": 0}
    models = [(model_a, "A"), (model_b, "B")]

    for i in range(games):
        game = OthelloGame()
        history = []

        swap = i % 2
        players = {BLACK: models[swap], WHITE: models[1 - swap]}

        while not game.is_finished():
            player = game.current_player
            model, _ = players[player]

            policy = run_mcts(model, game.state)
            history.append((game.state, policy, player))

            action = sample(policy) if collect else argmax(policy)
            game.play(action)

        winner = game.winner()
        if winner != DRAW:
            wins[players[winner][1]] += 1

        if collect:
            for state, policy, player in history:
                data.append((state, policy, game.result(player)))

    win_a = 100 * wins["A"] / games
    win_b = 100 * wins["B"] / games
    return data, win_a, win_b
```

Read more: Github

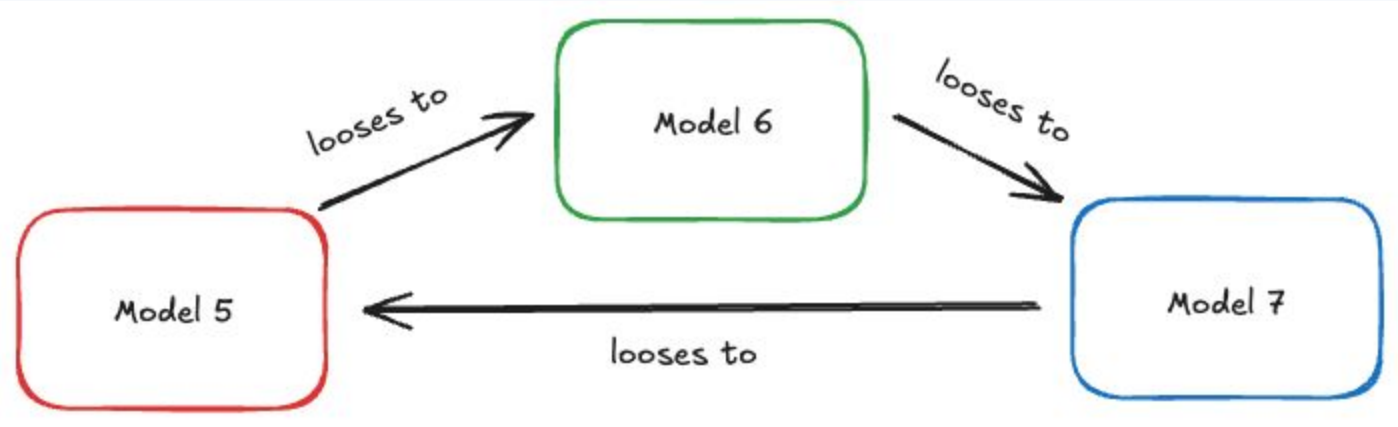
Learnings

- **More simulations vs More games**
  More games to ensure the training pipeline was exposed to a wider range of states

- **Simulation budget vs Variance trade-off**
  More simulation - less variance

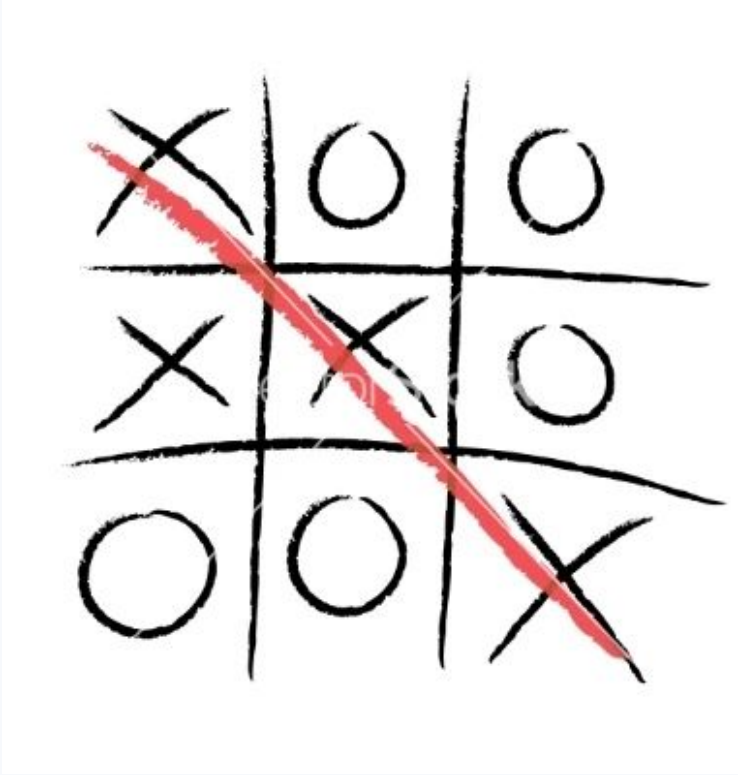
- **Exploration vs Variance trade-off**
  More exploration - higher the noise, but also more state exposure. Keep exploration relatively high throughout training and address the resulting noise in the training pipeline

- **Rock-paper-scissors Trap**
  Entropy regularisation helps to fight the policy being to specialised



Practical Tips

- **Unit Test in Smaller Environments**
- **Decide on the model size**
- **Plot stuff**
- **Play with your agent**



Final note: It’s easier to observe meaningful progress when exploration is sufficient. Yes, the model may not be as strong initially, but it’s better to get somewhere first—and only then make it more greedy.

SYNTHESIS

Five design principles for closing the loop

01	**Start from a prior with enough support**	The learner must be able to sample useful behavior before reinforcement can amplify it.
02	**Match feedback to the real objective**	Proxies scale, but direct outcomes reveal proxy gaps.
03	**Treat exploration as data design**	Choose which worlds and paths deserve compute.
04	**Exploit verifier gaps deliberately**	Spend test-time compute where evaluation is cheaper than generation.
05	**Measure generalization beyond the reward**	Check whether the system learned a procedure or merely a task-shaped heuristic.