Discovering Adaptable Symbolic Algorithms from Scratch

Google Research /

Stephen Kelly[1,4], Daniel S. Park[1], Xingyou Song[1,2], Mitchell McIntire[3], Pranav Nashikkar[3], Ritam Guha[5],
Wolfgang Banzhaf[5], Kalyanmoy Deb[5], Vishnu Naresh Boddeti[5], Jie Tan[1,2], Esteban Real[1,2]
[1]Google Research, [2]Google DeepMind, [3]Google, [4]McMaster University, [5]Michigan State University

MICHIGAN STATE UNIVERSITY

McMaster University

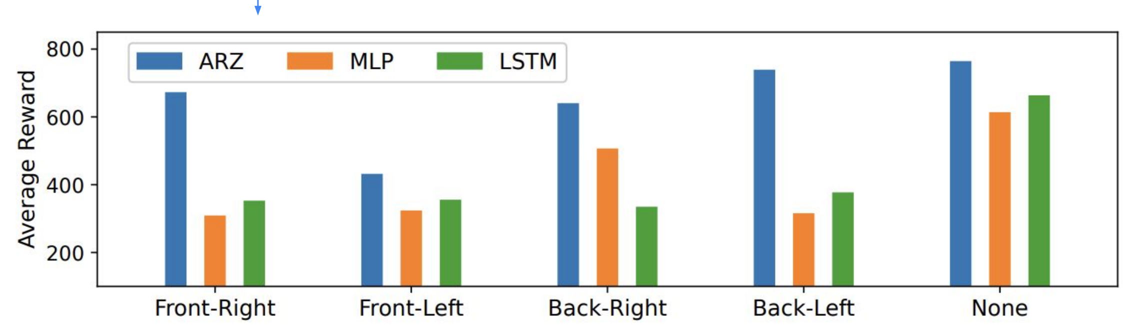
Autonomous robots in the real world must rapidly adapt to environmental changes.

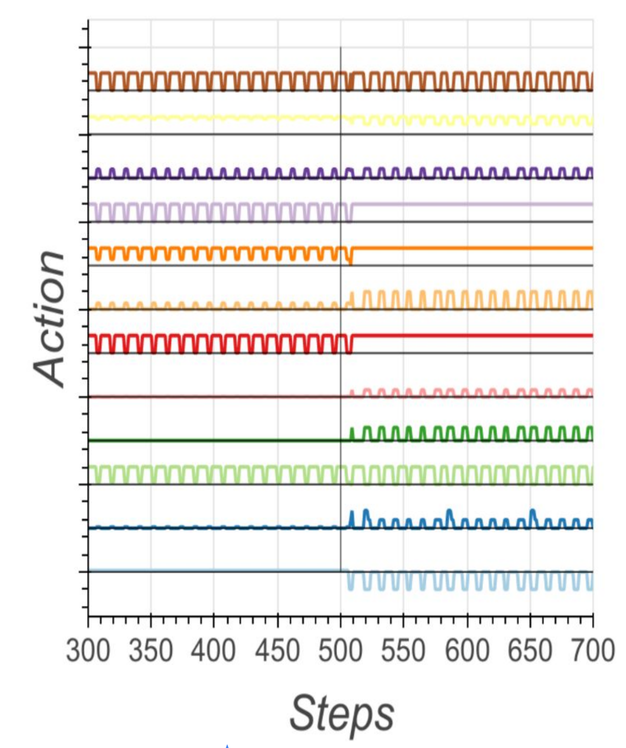
Starting from empty code and only using simple primitives, AutoRobotics-Zero (ARZ) searches for an algorithm to control a real robot *simulator*.

Laikago robot simulator.
Rendering credit: Peng 2020.
"Learning Agile Robotic..."

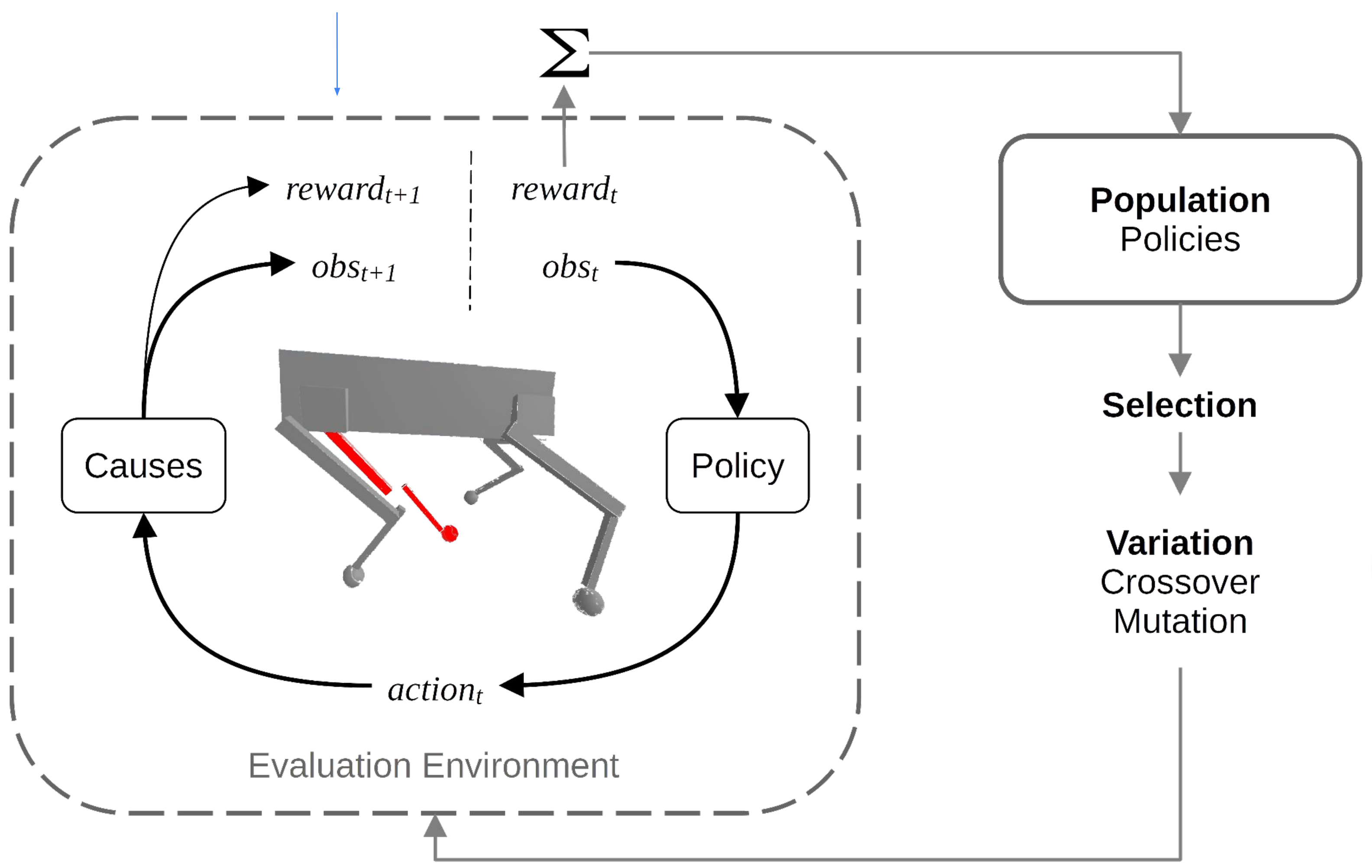
```
# wX: vector memory at address X.
def f(x, v, i):
  w0 = copy(v)
  w0[i] = 0
  w1 = abs(v)
  w1[0] = -0.858343 * norm(w2)
  w2 = w0 * w0
  return log(x), w1

# sX: scalar memory at address X.
# vX: vector memory at address X.
# obs, action: observation and action vectors.
def GetAction(obs, action):
  if s13 < s15: s5 = -0.920261 * s15
  if s15 < s12: s8, v14, i13 = 0, min(v8, sqrt(min(0, v3))), -1
  if s1 < s7: s7, action = f(s12, v0, i8)
  action = heaviside(v12)
  if s13 < s2: s15, v3 = f(s10, v7, i2)
  if s2 < s0: s11, v9, i13 = 0, 0, -1
  s7 = arcsin(s15)
  if s1 < s13: s3 = -0.920261 * s13
  s12 = dot(v3, obs)
  s1, s3, s15 = maximum(s3, s5), cos(s3), 0.947679 * s2
  if s2 < s8: s5, v13, i5 = 0, min(v3, sqrt(min(0, v13))), -1
  if s6 < s0: s15, v9, i11 = 0, 0, -1
  if s2 < s3: s2, v7 = f3(s8, v12, i1)
  if s1 < s6: s13, v14, i3 = 0, min(v8, sqrt(min(0, v0))), -1
  if s13 < s2: s7 = -0.920261 * s2
  if s0 < s1: s3 = -0.920261 * s1
  if s7 < s1: s8, action = f(s5, v15, i3)
  if s0 < s13: s5, v7 = f(s15, v7, i15)
  s2 = s10 + s3
  if s7 < s12: s11, v13 = f(s9, v15, i5)
  if s4 < s11: s0, v9, i13 = 0, 0, -1
  s10, action[i5] = sqrt(s7), s6
  if s7 < s9: s15 = 0
  if s14 < s11: s3 = -0.920261 * s11
  if s8 < s5: s10, v15, i1 = 0, min(v13, sqrt(min(0, v0))), -1
  return action
```

In the end, evolution discovers an *algorithm* that walks and *adapts* to the breakage of a random limb at a random time on-the-fly. The discovered algorithm has very few parameters and *outperforms* baselines.

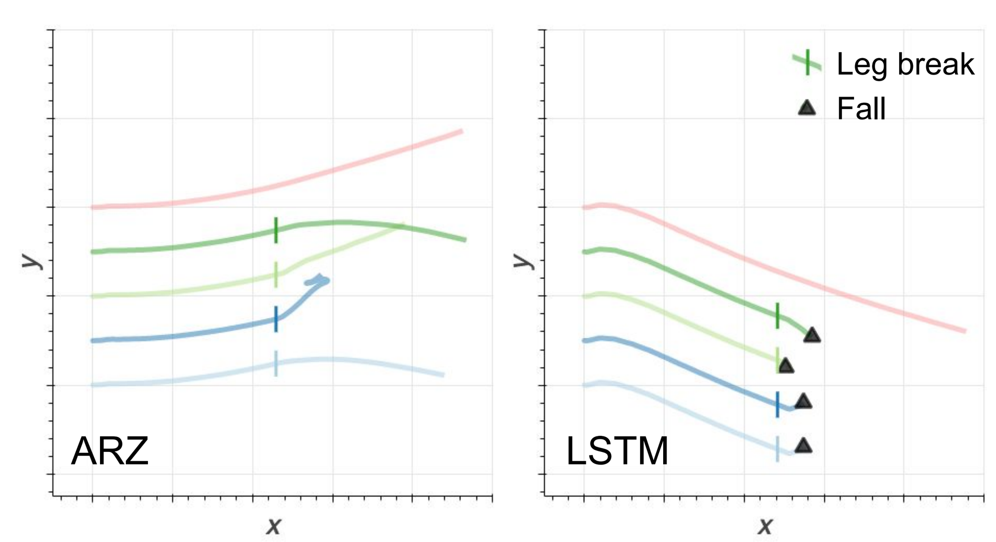




To achieve this, an outer search loop acts on a "genome" of code and an inner loop evaluates its ability to predict actions and adapt to radical change, simulating *evolution and lifetime learning.*

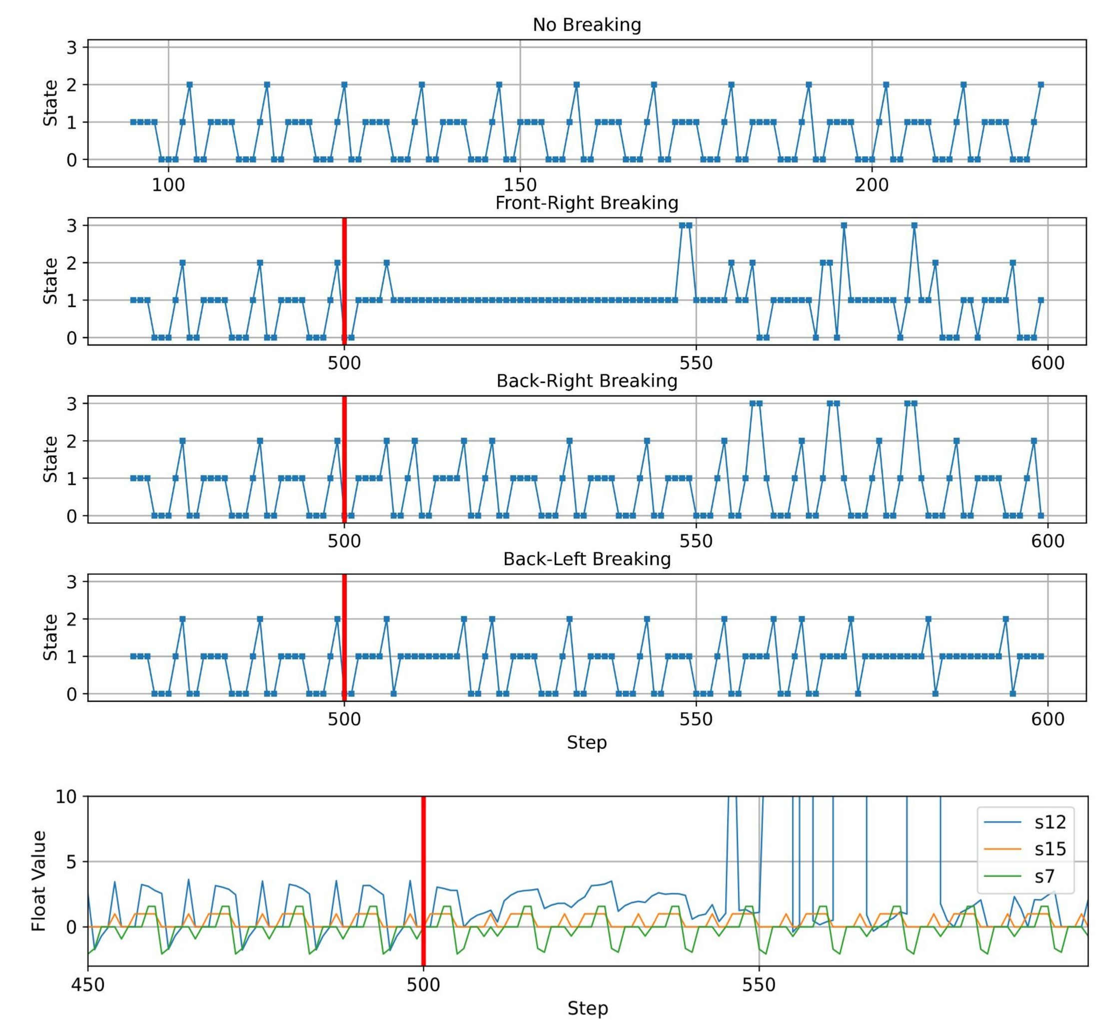


Evolved *walking behavior* changes rapidly when a leg breaks, and the robot reliably *avoids falling.*



Sample robot trajectories in each leg breaking task.

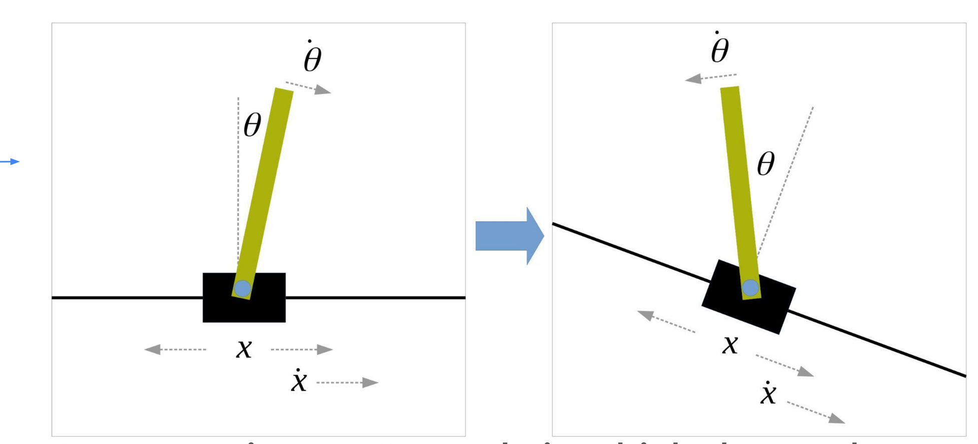
Code evolution builds simple, interpretable algorithms with minimal inductive bias.



The algorithm descretizes its observations into *4 memory states over time.* A random leg breaking disrupts the temporal pattern, signalling a change in the environment.

The behavior of other memory scalars are unaffected by the leg break. Intriguingly, their *periodicity resembles central pattern generators in biological circuits* responsible for generating rhythmic movements (Marder 2021. "Central pattern generators…").

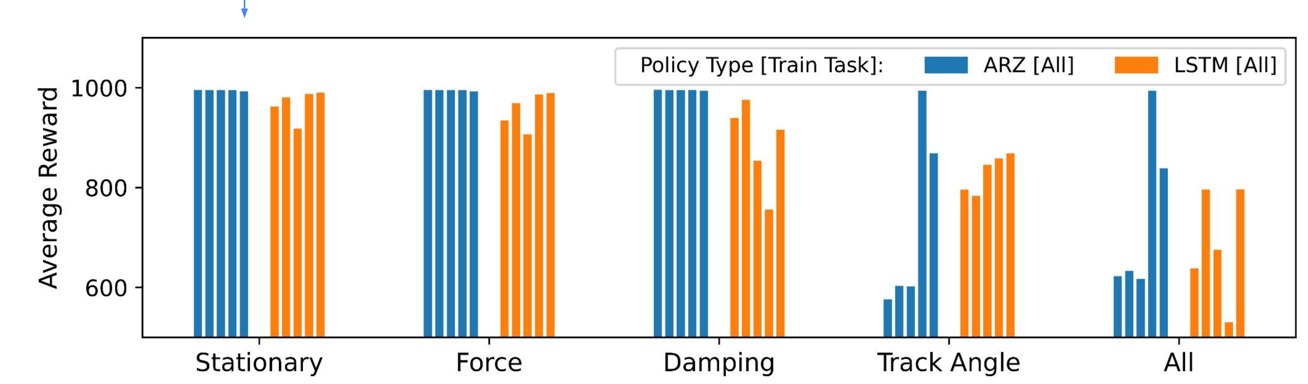
In order to run hundreds of experiments for analysis, we introduce a *Cataclysmic Cartpole* task that requires less compute than the quadruped.

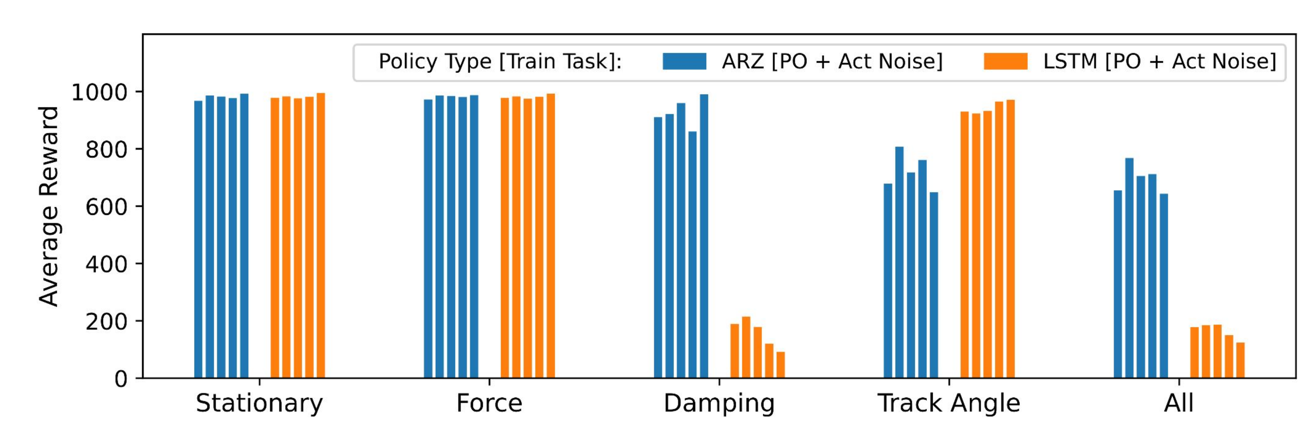


A non-stationary Cartpole in which the track angle and other parameters change at random times.

In Cataclysmic Cartpole, the *evolved controller* can be interpreted as a linear RNN or a variant of a PID. Even when multiple physics parameters suddenly change, this policy *maintains near optimal control.*

```
# sX: scalar memory at address X.
# obs: vector [x, theta, x_dot, theta_dot].
# a, b, c: fixed scalar parameters.
# V, W: 4-dimensional vector parameters.
def GetAction(obs, action):
  s0 = a * s2 + action
  s1 = s0 + s1 + b * action + dot(V, obs)
  s2 = s0 + c * s1
```





How can we build adaptive control policies without any prior knowledge of what type of change will occur?

Future work may build on preliminary findings that adding partial-observability and actuator noise to the standard Cartpole task allows ARZ to evolve algorithms that *adapt to multiple unseen changes* in Cataclysmic Cartpole.