LEARNING TO FIND PROOFS AND THEOREMS
BY LEARNING TO REFINE SEARCH STRATEGIES

THE CASE OF LOOP INVARIANT SYNTHESIS

NEURAL INFORMATION PROCESSING SYSTEMS

Jonathan Laurent, André Platzer

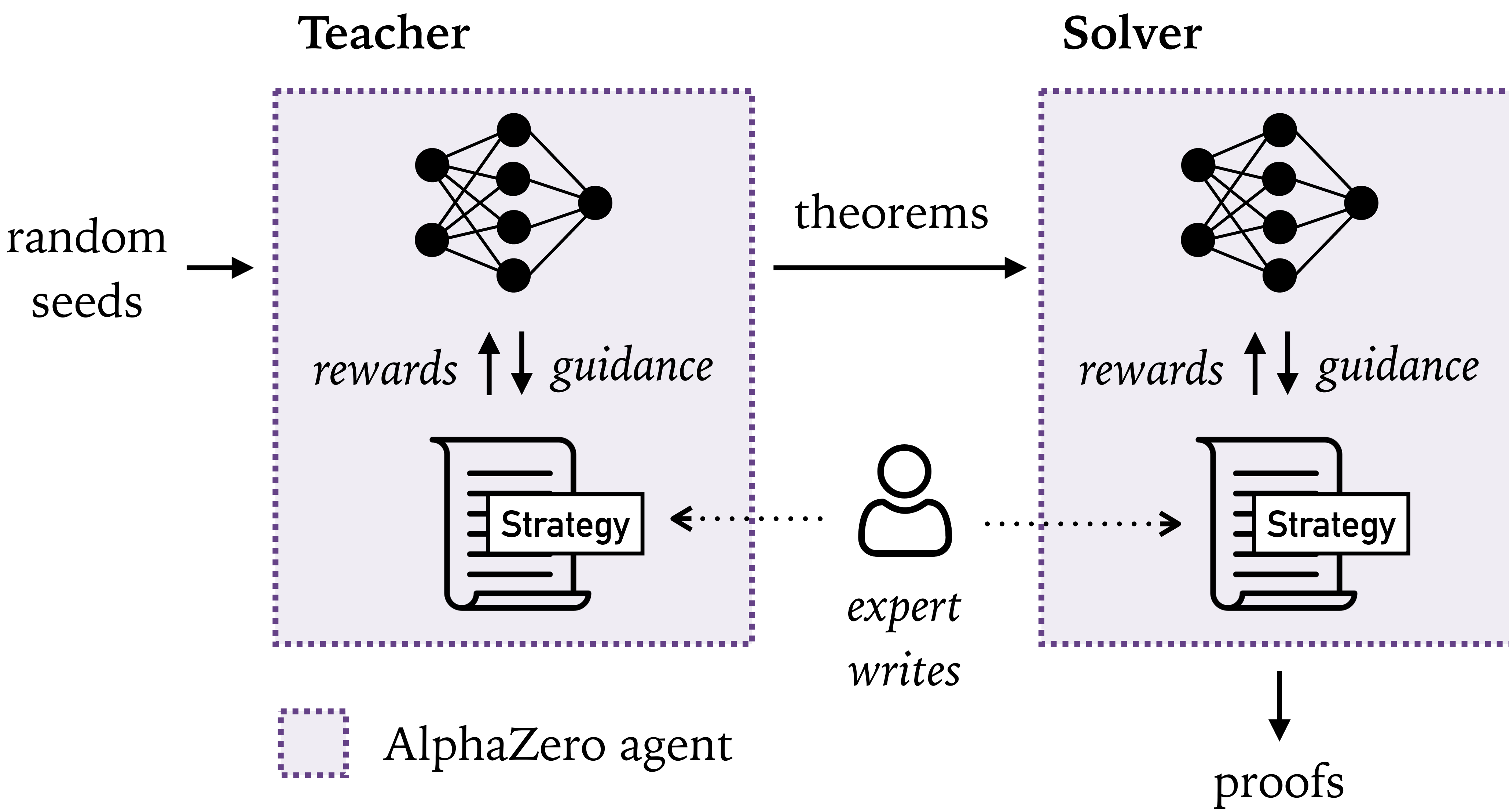
Carnegie Mellon University

Karlsruhe Institute of Technology

Can theorem proving be learned without a single example of a proof or theorem?

- **Imitation learning** is limited by the scarcity of human proofs
- **Reinforcement learning** presents challenges:
  - Infinite action spaces are hardly amenable to exploration
  - Theorems are still needed as training tasks

PROPOSED APPROACH



LOOP INVARIANT SYNTHESIS

- Training data unavailable and hard to generate!
- No pre-existing deep-learning agent capable of generalizing across instances.

```
assume x >= 1
y = 0
while y < 1000 {
   x = x + y
   y = y + 1
}
assert x >= y
```

To prove the final assertion, one must find a <u>loop invariant</u> that:

1. is true before the loop
2. is preserved by the loop body (when the loop guard holds)
3. implies the final assertion (when the loop guard does not hold)

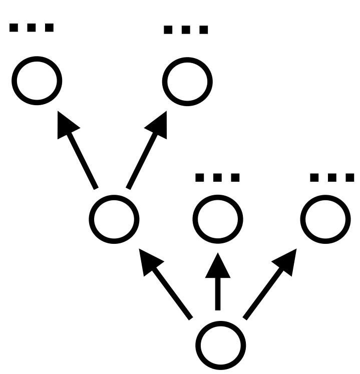
Invariant: $x \geq y \land x \geq 1 \land y \geq 0$

A LANGUAGE FOR EXPRESSING STRATEGIES

We define a strategy language based on **choose** and **event** operators.

Expert strategy

compiled into



MDP amenable to RL and neural-guided search

```
def solver(
    init: Formula, guard: Formula,
    body: Program, post: Formula) -> Formula:
    def prove_inv(inv: Formula) -> List[Formula]:
        assert valid(Implies(init, inv))
        ind = Implies(And(guard, inv), wlp(body, inv))
        event(PROVE_INV_EVENT)
        match abduct(ind):
        case Valid:
            return [inv]
        case [*suggestions]:
            aux = choose(suggestions)
            return [inv] + prove_inv(aux)
inv_cand = choose(abduct(Implies(Not(guard), post)))
inv_conjuncts = prove_inv(inv_cand)
return And(*inv_conjuncts)
```

▲ A **solver strategy** for invariant synthesis

GENERATING TRAINING PROBLEMS

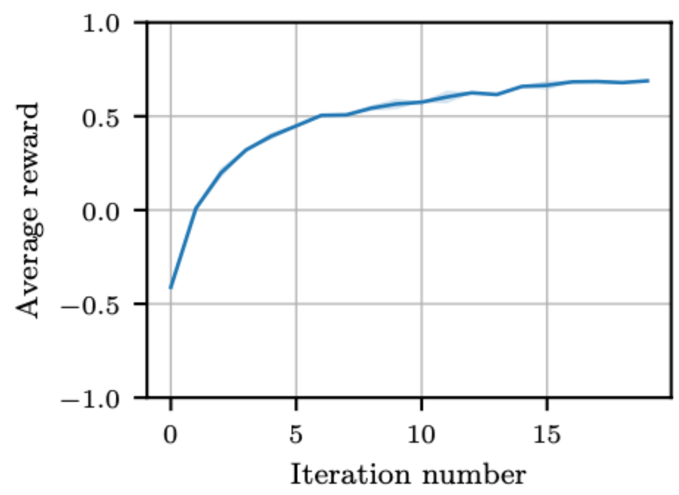
- Generating interesting theorems is harder than proving those!
- **Our approach:** refining **conditional generative strategies** using RL.

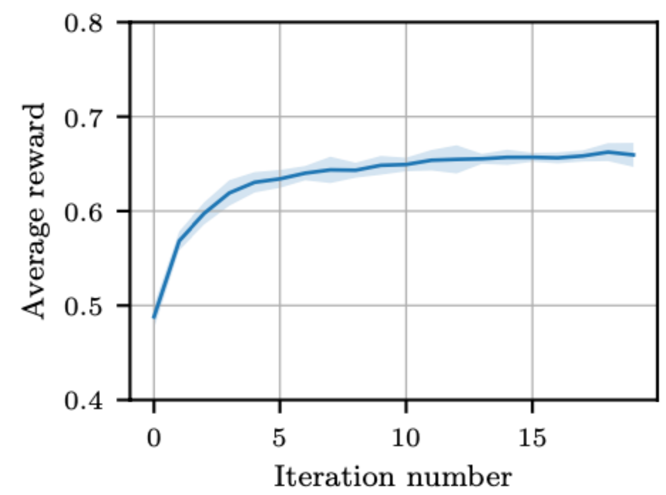
```
def teacher(rng: RandGen) -> Prog:
    cs = sample_constrs(rng)
    p = generate_prog(cs)
    p = transform(p, rng)
    p = hide_invariants(p)
    return p

def generate_prog(cs: Constrs):
    p = Prog("
      assume init;
      while (guard) {
        invariant inv_lin;
        invariant inv_aux;
        invariant inv_main;
        body; }
      assert post;")
    p = refine_guard(p, cs)
    p = refine_inv(p, cs)
    p = refine_body(p, cs)
    assert valid(inv_preserved(p))
    p = refine_post(p, cs)
    assert valid(inv_post(p))
    p = refine_init(p, cs)
    assert valid(inv_init(p))
    penalize_violations(p, cs)
    return p

def transform(p: Prog, rng: RandGen):
    p = shuffle_formulas(p, rng)
    p = add_useless_init(p, rng)
   ...
    return p
```

▲ Outline of a **teacher strategy** for invariant synthesis

RESULTS ON INVARIANT SYNTHESIS

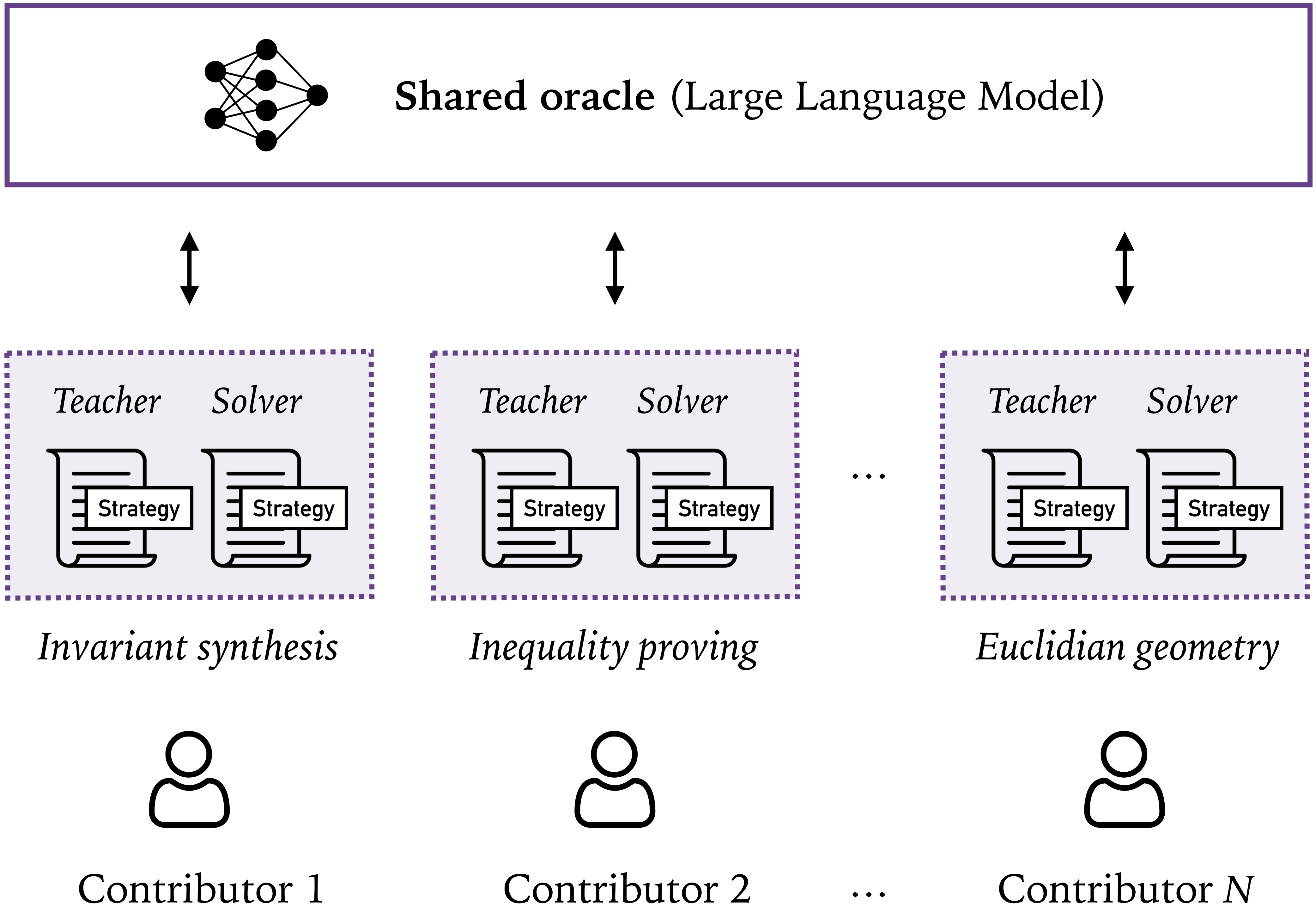
- Training curves for the teacher and the solver (respectively):





- Experimental results on Code2Inv (no backtracking search):

Policy	% Problems solved
Random	18.4 ± 0.0
Network (untrained teacher)	39.7 ± 1.6
Network (trained teacher)	$\mathbf{61.5} \pm \mathbf{0.4}$



Shared oracle (Large Language Model)
Teacher
Solver
Strategy
Strategy
Invariant synthesis
Teacher
Solver
Strategy
Strategy
Inequality proving
...
Teacher
Solver
Strategy
Strategy
Euclidian geometry
Contributor 1
Contributor 2
...
Contributor N