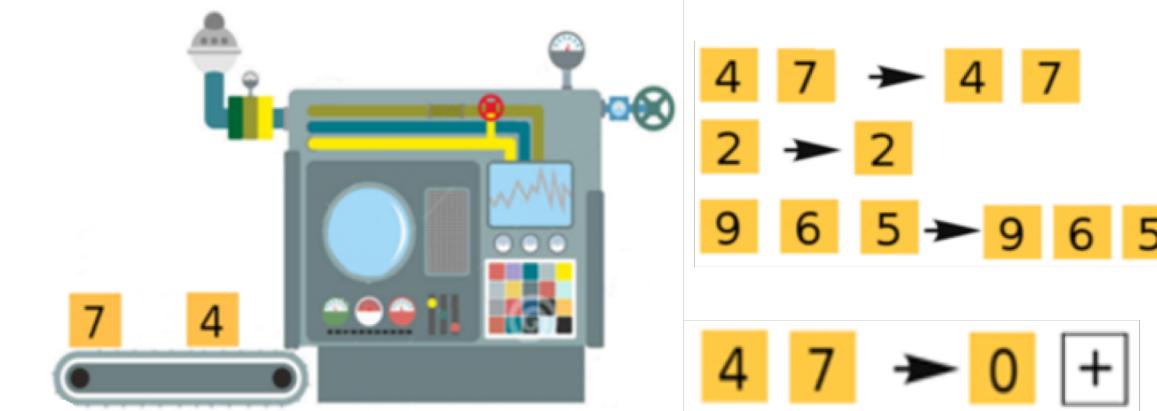


Concept Learning as Program Learning

Learning List Concepts through Program Induction

Problem: Model human learning of structured concepts in two online experiments using a game-based paradigm.



Approach: Search over rewrite systems (top) to learn concepts (bottom).

```
def search(data, h0, N=1500, n_top=10, n_steps=50, confidence=2/3):
    dataset = []
    h, score = h0, score(h0)
    hs = heap([(h, score)])
    for i, o in data:
        for _ in range(N):
            h_next = propose(h)
            score_next = score(h_next)
            h, score = metropolis(h, score, h_next, score_next)
            hs.insert((h, score))
    best_hs = hs.take_top(n_top)
    o_hat = most_likely_output(i, n_steps, best_hs)
    data.append((i, o))
    N *= (confidence if o_hat == o else 1/confidence)
    return hs
```

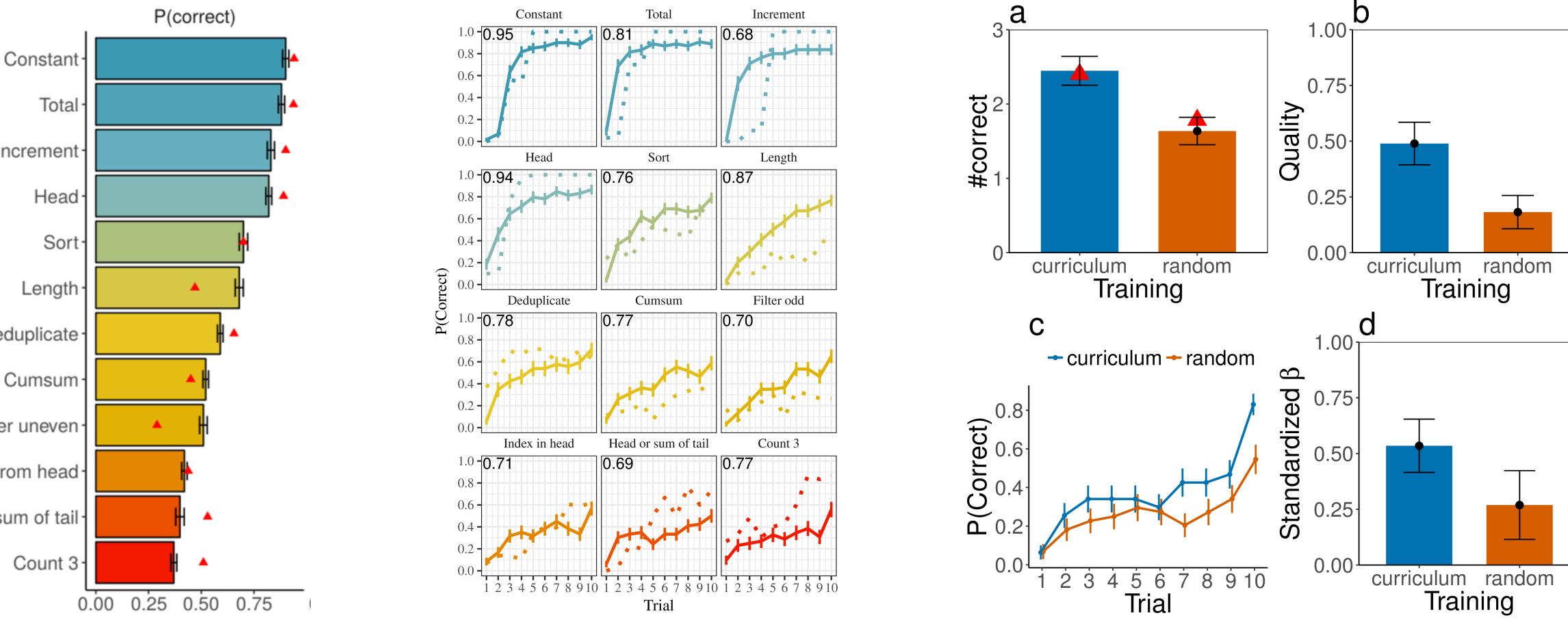
```
# const xs: return 3
# Example: const([1,2,4]) = [3]
const(x_) = 3;

# total xs: sum all the elements of xs
# Example: total([1,2,3]) = [6]
total(x_) = sum(x_);
```

```
# head-or-tail: the larger of head or summed tail
# Example: head_or_tail([2,3,1]) = [4]
head_or_tail([]) = [];
head_or_tail(cons(x_, y_)) =
    if(greater(x_ sum(y_)) x_ sum(y_));

# count3 xs: how often does 3 appear in xs?
# Example: count3([2,3,3]) = [2]
count3(x_) = count(succ(succ(succ(0))) x_);
```

Results: This approach accurately models overall (left) and per-trial performance (middle) in Exp. 1, and predicts curriculum training outperforming random training in Exp. 2.



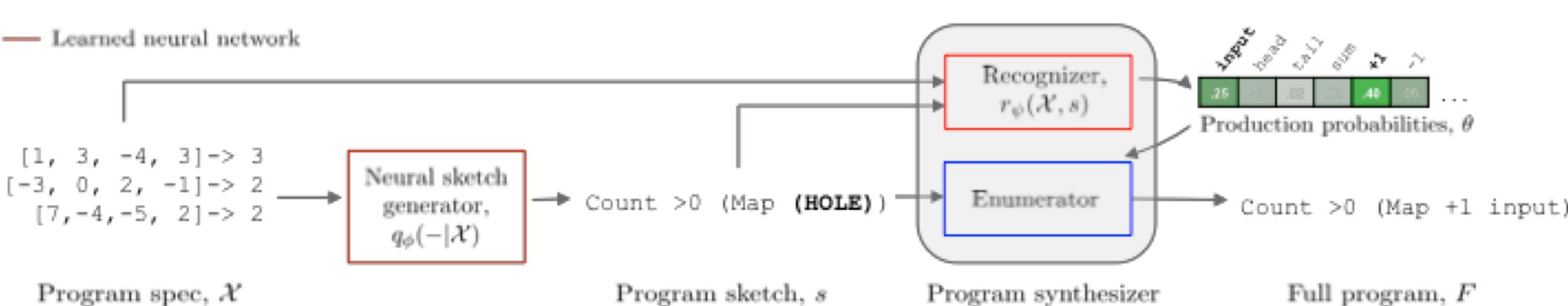
Learning to Infer Program Sketches

Problem: Build systems which write code automatically from the kinds of specifications humans can easily provide, such as examples and natural language instruction.

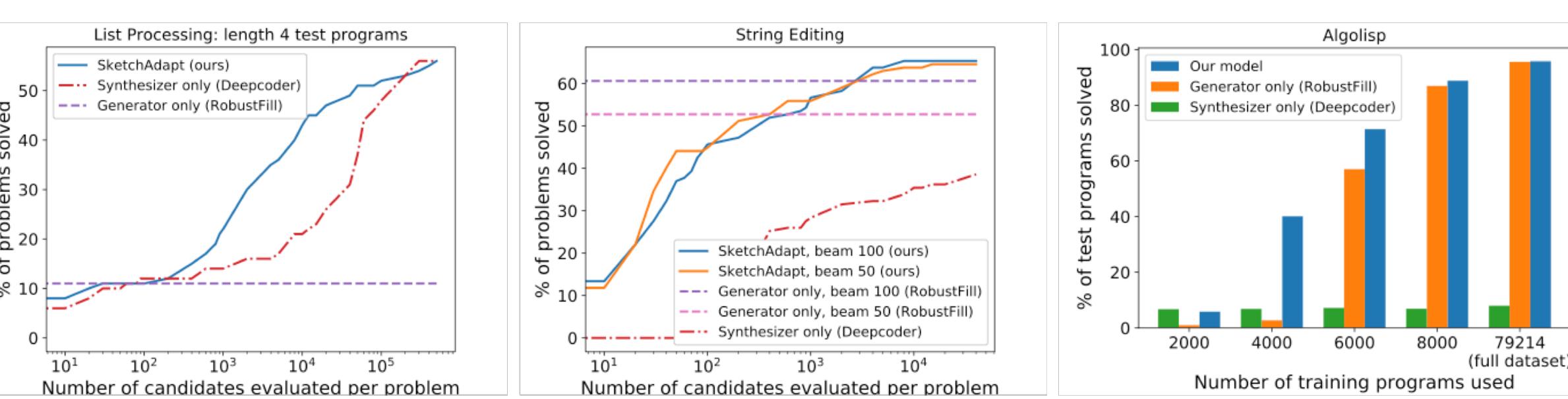
Spec	Program
$[2,3,4,5,6] \rightarrow [2,4,6]$	filter(input, x: x%2==0)
$[5,8,3,2,1,12] \rightarrow [8,2,12]$	(reduce(reverse(digits(de ref (sort a) (/ (len a) 2))) 0 (lambda2 (+* arg1 10) arg2)))

Given an array of numbers, your task is to compute the median of the given array with its digits reversed.

Approach: Learn *Program sketches*, which serve as an intermediate between pattern recognition and explicit search approaches.



Results: This approach can synthesize programs more accurately than baselines in a variety of domains: list processing from examples (left), text editing from examples (middle), and character and list manipulation from language description (right).



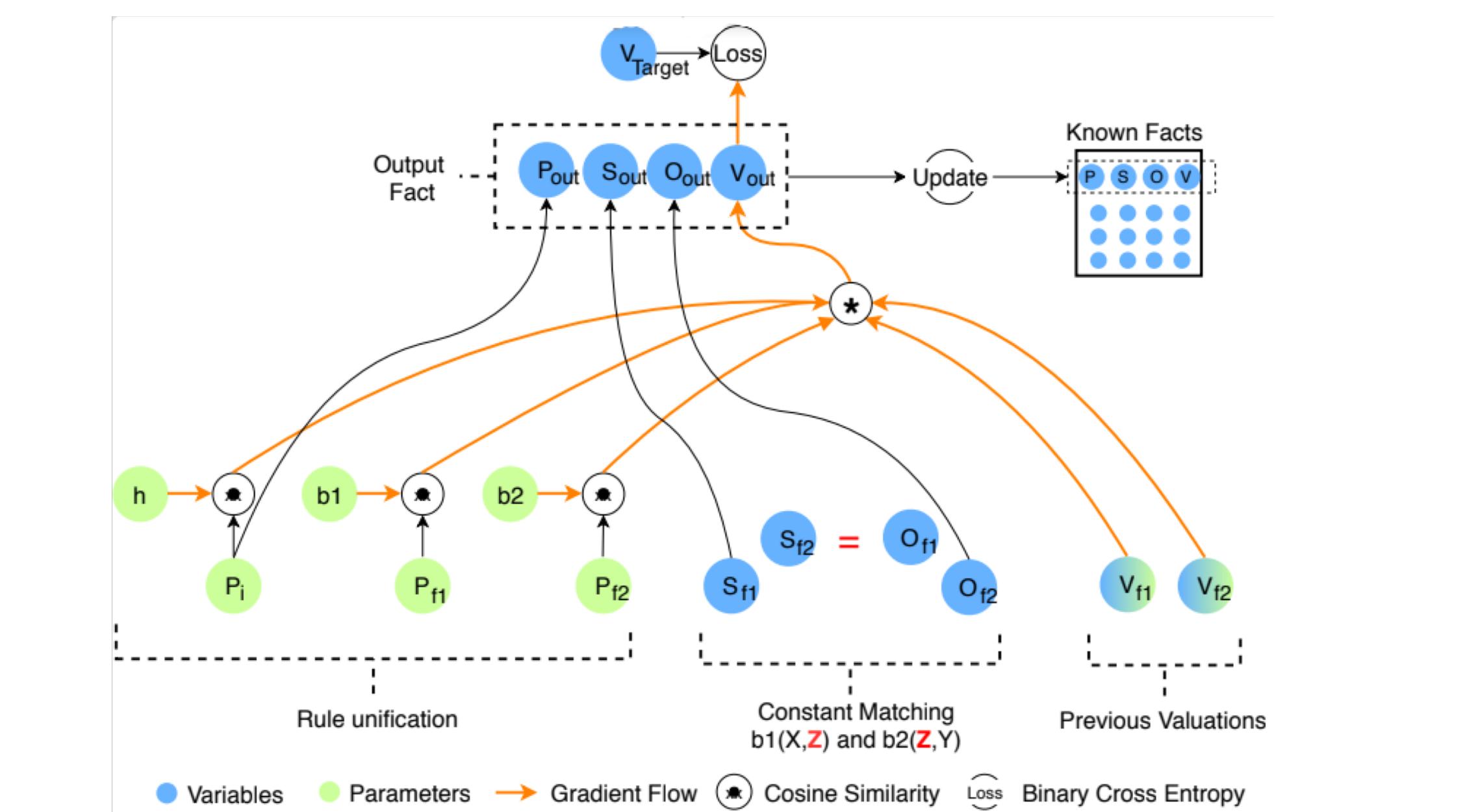
Logical Rule Induction via Neural Theorem Proving

Problem: Transform a knowledge base of observations (left) into a logical theory containing generalized predicates and core facts (right).

```
[grandfatherOf, ABE, BART].  
[fatherOf, ABE, HOMER].  
[fatherOf, HOMER, BART].  
[parentOf, ABE, HOMER].  
[parentOf, HOMER, BART].  
[fatherOf, JAMES, HARRY].  
[parentOf, LILY, HARRY].  
...  
...
```

```
[grandfatherOf, X, Y] :-  
  [[fatherOf, X, Z], [parentOf, Z, Y]].  
[parentOf, X, Y] :- [fatherOf, X, Y].  
[parentOf, X, Y] :- [motherOf, X, Y].  
[fatherOf, ABE, HOMER].  
[fatherOf, HOMER, BART].  
[motherOf, LILY, HARRY].  
...
```

Approach: Learn programs using neuro-symbolic induction. This overview shows one reasoning step.



Results: This approach recovers taxonomic relations (left) and outperforms previous state-of-the-art on a variety of problems (right; score is percentage of random initializations leading to solution. δILP is previous state-of-the-art (Evans, Grefenstette, 2018)).

Task	Recursive	δILP	Ours
Even	Yes	48.5	100
Fizz	Yes	10	10
Buzz	Yes	35	70
Grandparent	No	96.5	100
Cyclic	Yes	100	100