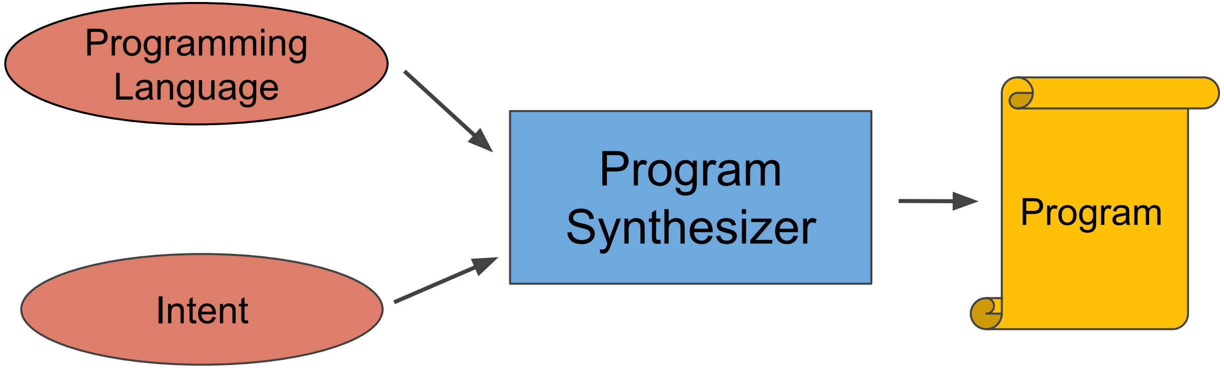
Making Neural Programming Architectures Generalize via Recursion

Jonathon Cai, Richard Shin, Dawn Song

BERKELEY ARTIFICIAL INTELLIGENCE RESEARCH

University of California, Berkeley

Program Synthesis



Example Applications:

- End-user programming
- Performance optimization of code
- Virtual assistant

Neural Program Synthesis

Training data	123 234	452 345	612 367	**Input**
	↓	↓	↓	
	357	797	979	**Output**

Neural Program Synthesis

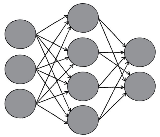
Training data

123 234	452 345	612 367	**Input**
357	797	979	**Output**

Neural Program Architecture

50
70

Test input

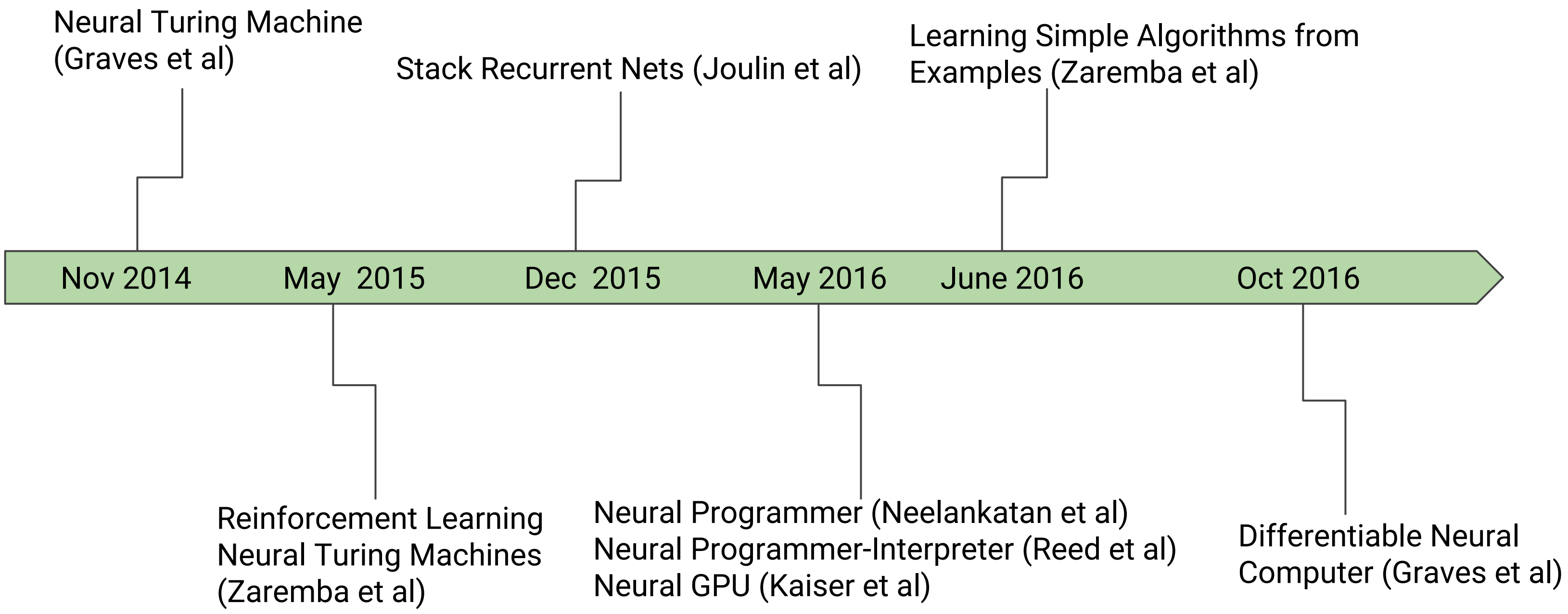


Learned neural program

120

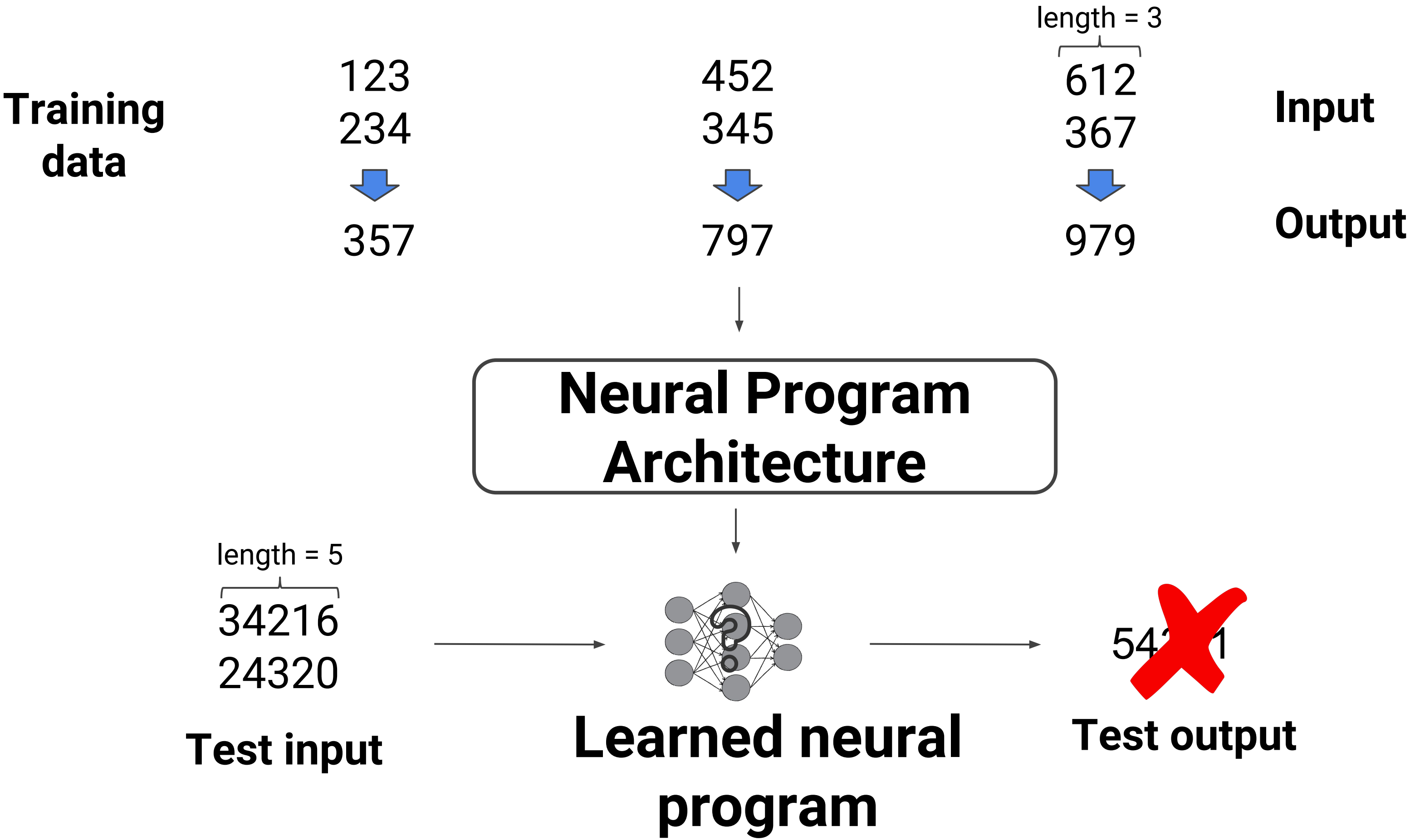
Test output

Neural Program Architectures

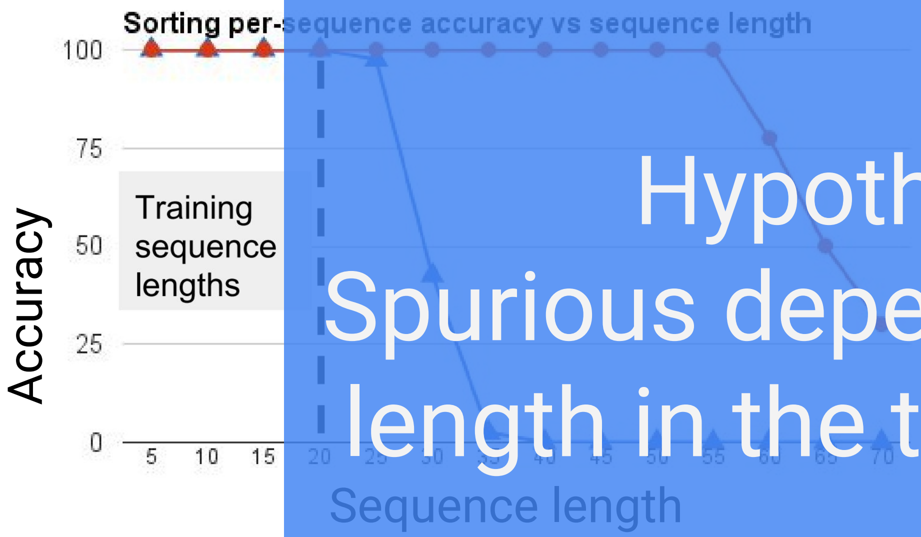


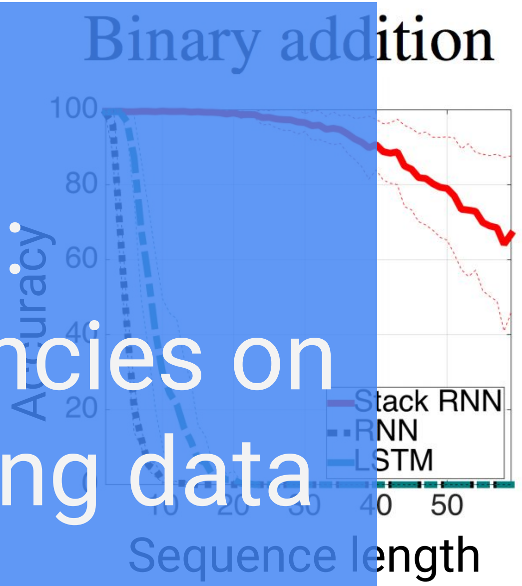
Neural Program Synthesis Tasks: Copy, Grade-school addition, Sorting, Shortest Path

Challenge 1: Generalization



Challenge 1: Existing Neural Program Architectures Do Not Generalize Well





Hypothesis:
Spurious dependencies on length in the training data

Trained up to: length 20
Fails from: length 60

NPI (Reed et al, 2016)

Trained up to: length 20
Fails from: length 20

Stack Recurrent Nets (Joulin et al, 2015)

Challenge 2: No Proof of Generalization

Training data

123 234	452 345	length = 3 612 367
357	797	979

Input

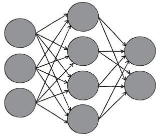
Output

Neural Program Architecture

length = 5

34216
24320

Test input



Learned neural program

58536

Test output

Problem Statement

For program synthesis tasks like addition and sorting:

- What challenges are we trying to address?
    - Generalization to more complex inputs
    - Proof of generalization

➔ Which approach will solve these challenges?

➔ How do we implement the approach?

Our Approach: Introduce Recursion

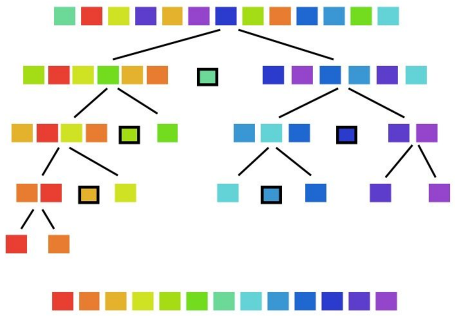
Learn recursive neural programs

Recursion

Fundamental concept in Computer Science and Math.

Solve whole problem by reducing it to smaller subproblems (*reduction rules*).

Base cases (smallest subproblems) are easier to reason about.



Quicksort

Our Contributions

For program synthesis tasks like addition and sorting:

- What challenges are we trying to address?
  - ✓ Generalization to more complex inputs
  - ✓ Proof of generalization

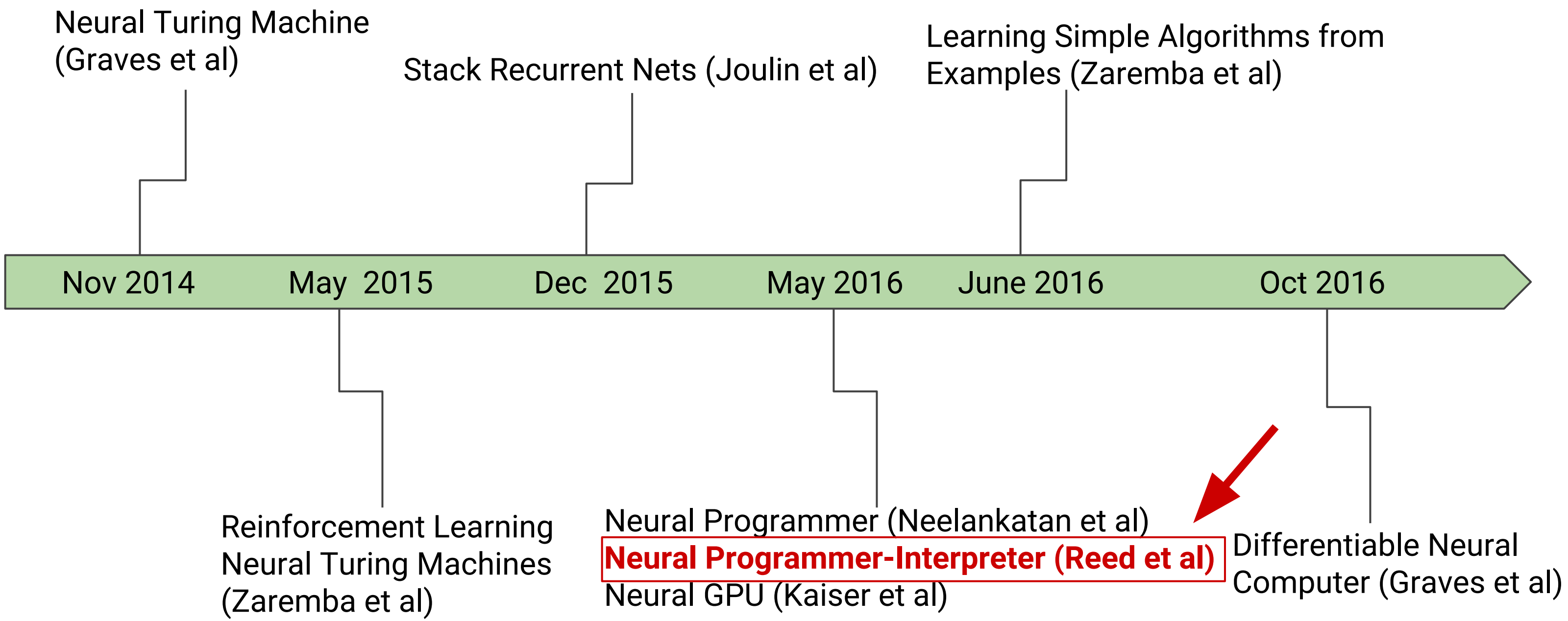
Main Contribution!

- Which approach will solve these challenges?
  - *Recursion* in neural programs
- How do we implement the approach?
  - Instantiation: Incorporate recursion into Neural Programmer-Interpreter
  - Training method: As a first step, strong supervision with *explicitly recursive execution traces* to learn a recursive neural program

Outline

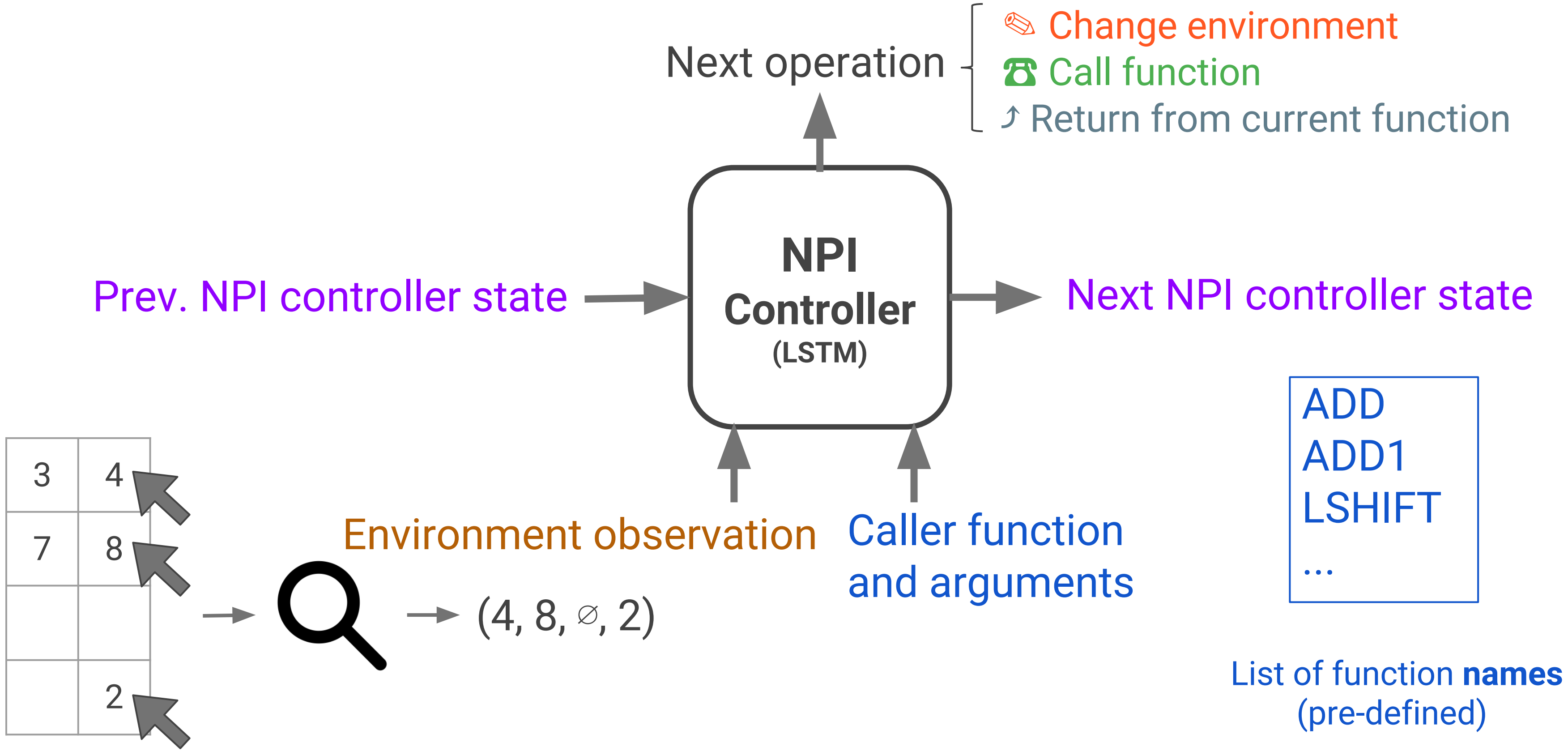
- Challenges in Neural Program Architectures
- Overview of Our Approach: Recursion
- ➔ Background: Neural-Programmer Interpreter
- Learning Recursive Neural Programs
- Provably Perfect Generalization
- Experimental Results
- Conclusion

Neural Program Architectures



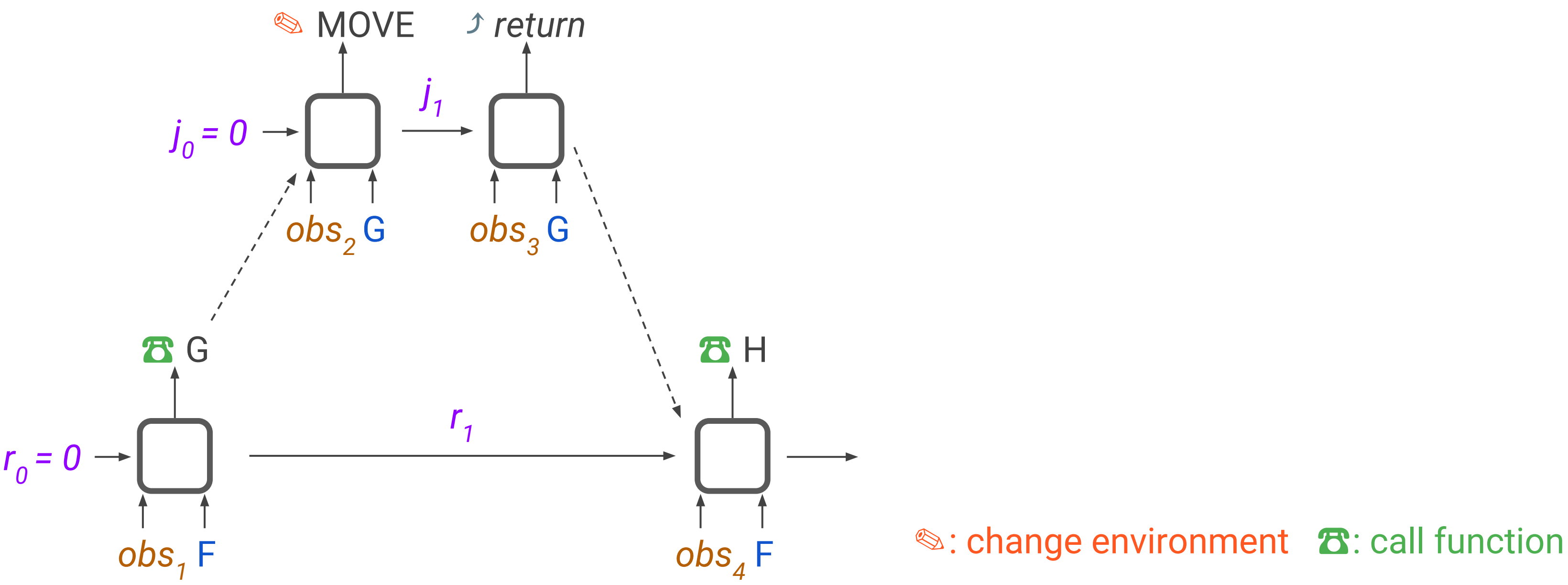
Neural Program Synthesis Tasks: Copy, Grade-school addition, Sorting, Shortest Path

Neural Programmer-Interpreter (NPI)



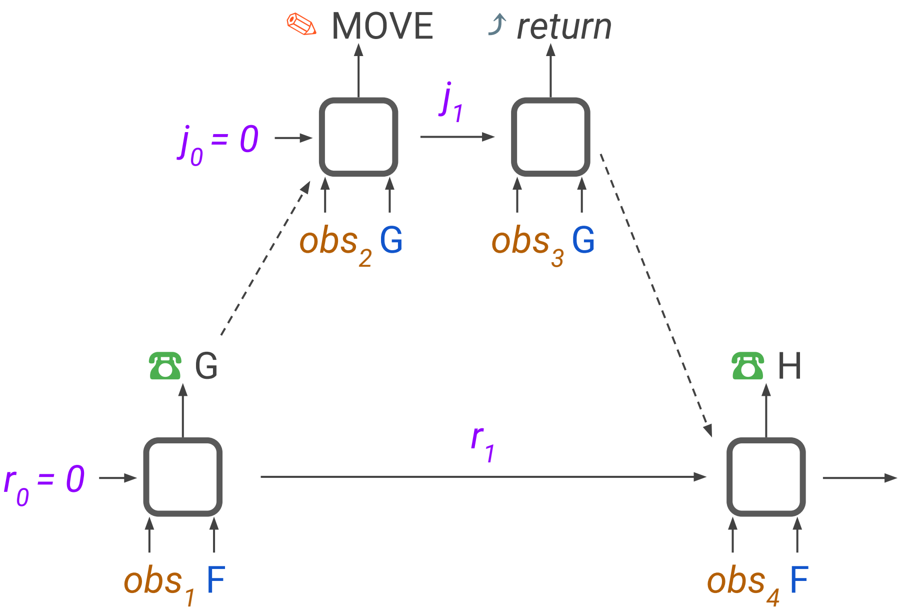
Execution of NPI

Calling a function creates a new NPI controller state (LSTM hidden state).



Execution traces in NPI

The sequence of operations forms an execution trace.



☎ F

obs_1, F → ☎ G

obs_2, G → ✎ MOVE

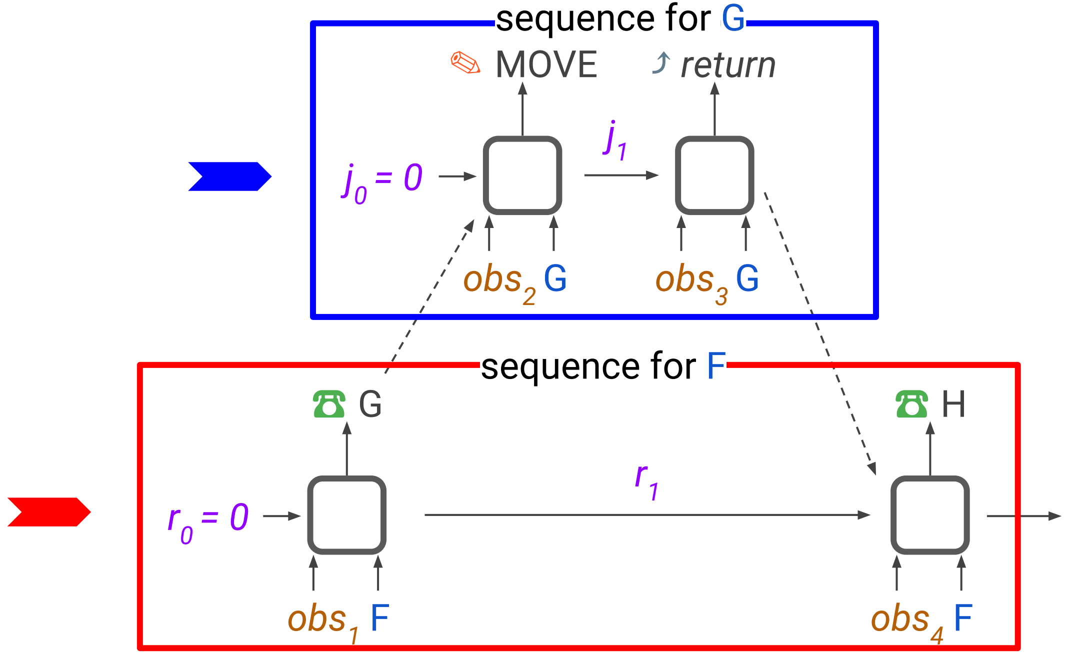
obs_3, G → ⤴ *return*

obs_4, F → ☎ H

✎: change environment ☎: call function

Training NPI with Execution Traces

Execution trace divided into training sequences, according to the caller function.



☎ F

obs_1, F → ☎ G

obs_2, G → ✎ MOVE

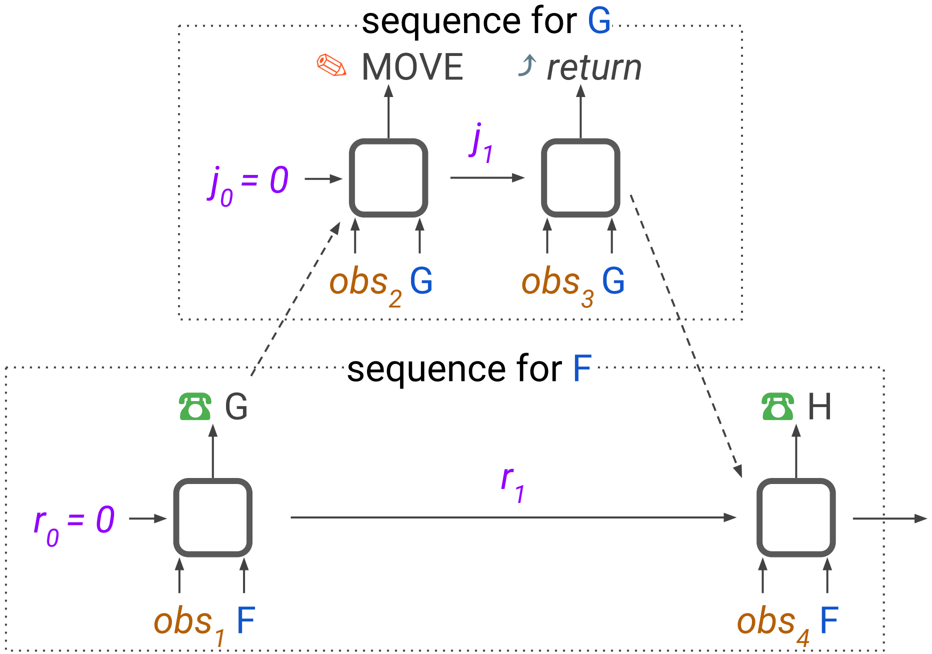
obs_3, G → ⤴ *return*

obs_4, F → ☎ H

✎: change environment ☎: call function

Simplified Execution Traces

For brevity, we omit details in the trace.



☎ F

☎ G

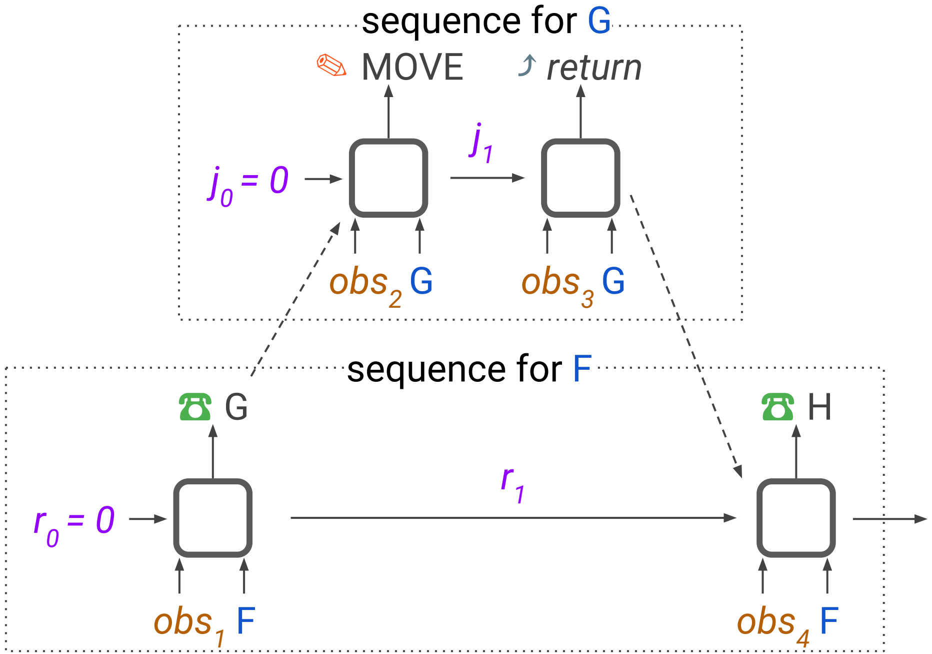
✎ MOVE

☎ H

✎: change environment ☎: call function

Simplified Execution Traces

For brevity, we omit details in the trace.



☎ F

☎ G

✎ MOVE

☎ H

✎: change environment ☎: call function

NPI Trains on Execution Traces, Not Input-Output Pairs

The training data for each architecture:

Input	123 234	452 345	612 367
Output	357	797	979

Neural Turing Machine
Neural GPU
Differentiable Neural Computer
etc.

☎ F
  ☎ G
    MOVE
  ☎ H

NPI

Outline

Challenges in Neural Program Architectures

Overview of Our Approach: Recursion

Background: Neural-Programmer Interpreter

➔ Learning Recursive Neural Programs

Provably Perfect Generalization

Experimental Results

Conclusion

Learn recursive neural programs

↓

Incorporate recursion into NPI

What is a Recursive NPI Program?

Trace from an example recursive NPI program: ☎ ADD calls itself

☎ **ADD**
- ☎ ADD1
- ☎ LSHIFT
- ☎ ADD1
- ☎ LSHIFT
- ☎ ADD1
- ☎ LSHIFT

…

Repeated inside **one** function call

Execution trace of
***non*-recursive program**
(previous work)

☎ **ADD**
- ☎ ADD1
- ☎ LSHIFT
- ☎ **ADD**
  - ☎ ADD1
  - ☎ LSHIFT
  - ☎ **ADD**

  …

Recursive calls

Execution trace of
recursive program
(our work)

Grade-School Addition

From right to left (smallest to largest position):

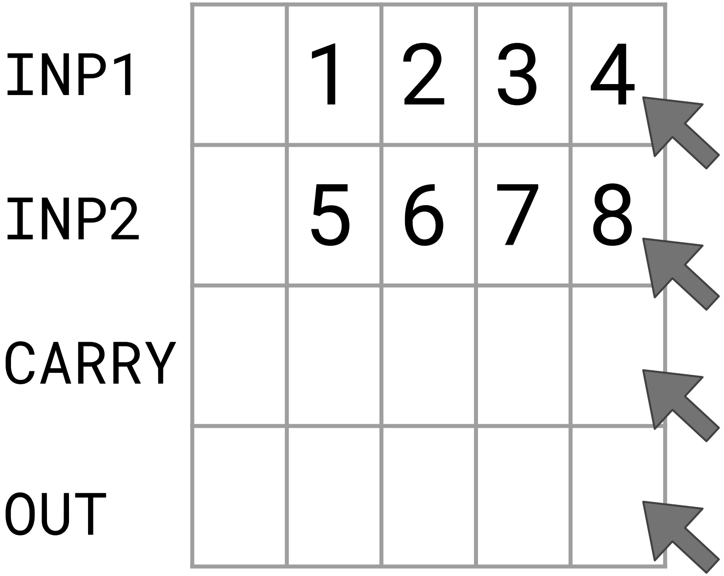
1. Add three values in the column.
2. If resulting sum exceeds 10, put a 1 in the next carry position.

		1	1		carry
	1	2	3	4	1st number
+	5	6	7	8	2nd number
	6	9	1	2	output

NPI (Reed et al, 2016)

Grade-School Addition

Scratchpad (environment):



Observation: value at each pointer; in this example, (4, 8, ∅, ∅)

Three functions

- ADD1: adds 1 column
- LSHIFT: move to next column
- CARRY: write carry digit if needed

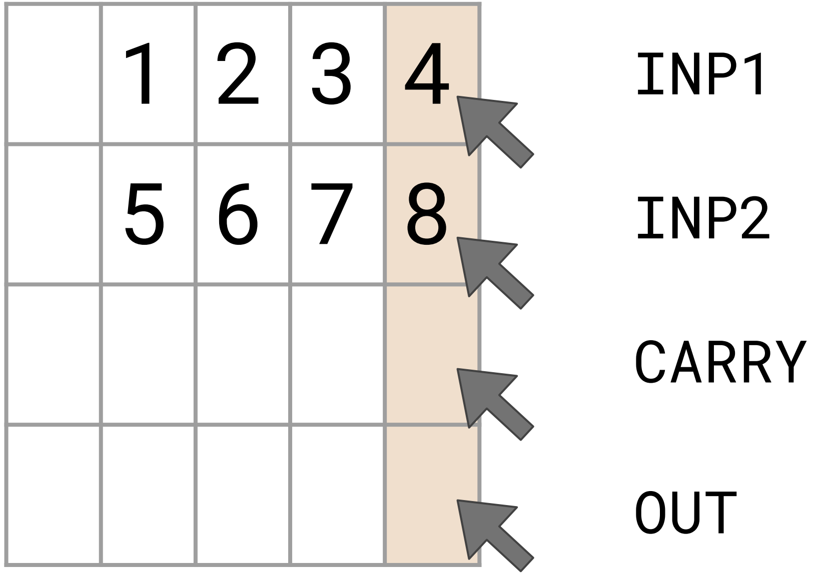
NPI (Reed et al, 2016)

Non-Recursive

Grade-School Addition

✎: change environment ☎: call function

☎ ADD
  ☎ ADD1



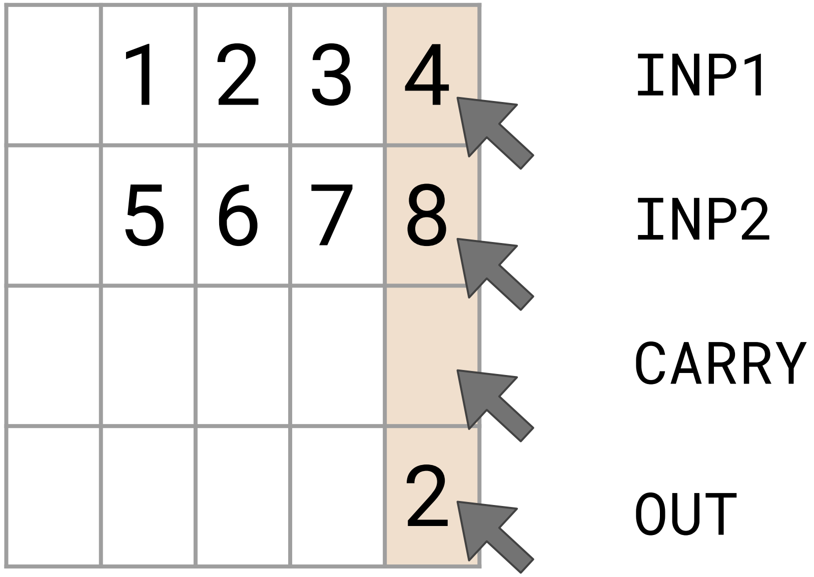
NPI (Reed et al, 2016)

Non-Recursive

Grade-School Addition

✎: change environment ☎: call function

☎ ADD
 ☎ ADD1
  ✎ WRITE OUT 2



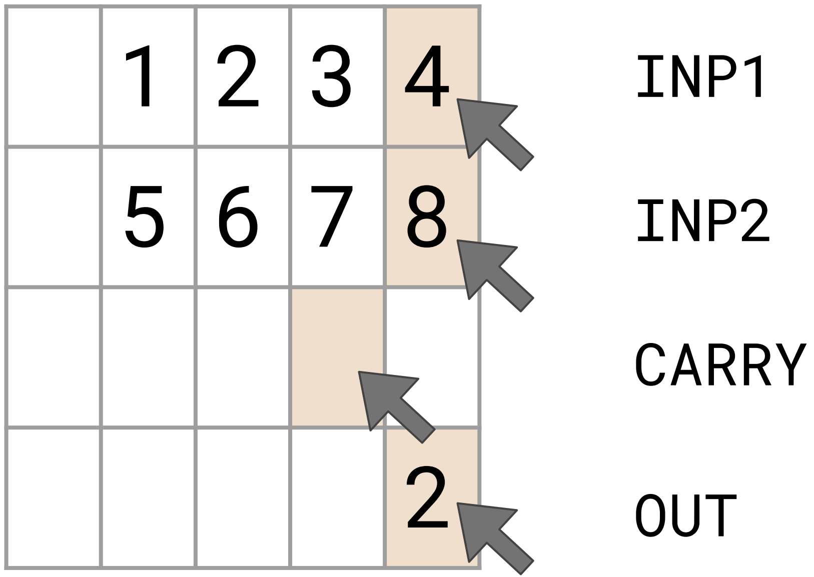
NPI (Reed et al, 2016)

Non-Recursive

Grade-School Addition

✎: change environment ☎: call function

- ☎ ADD
  - ☎ ADD1
    - ✎ WRITE OUT 2
    - ☎ CARRY
      - ✎ PTR CARRY LEFT



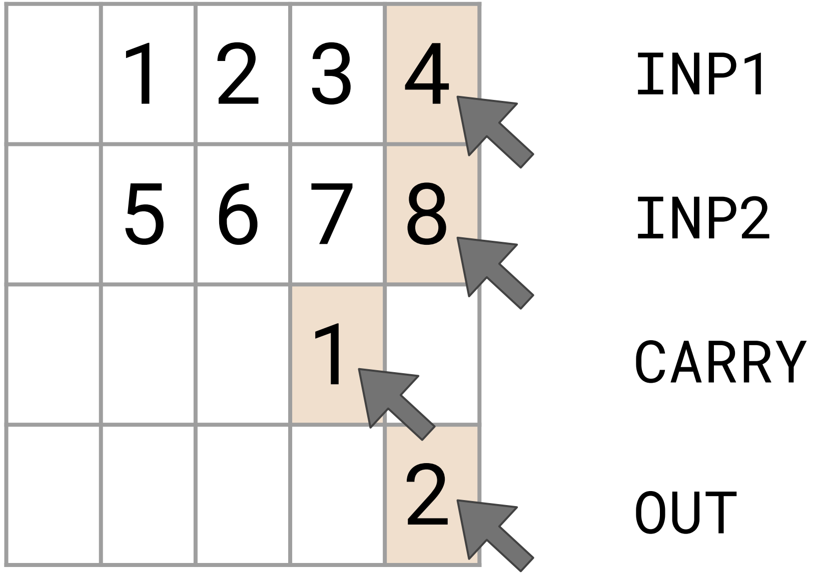
NPI (Reed et al, 2016)

Non-Recursive

Grade-School Addition

✎: change environment ☎: call function

- ☎ ADD
  - ☎ ADD1
    - ✎ WRITE OUT 2
    - ☎ CARRY
      - ✎ PTR CARRY LEFT
      - ✎ WRITE CARRY 1



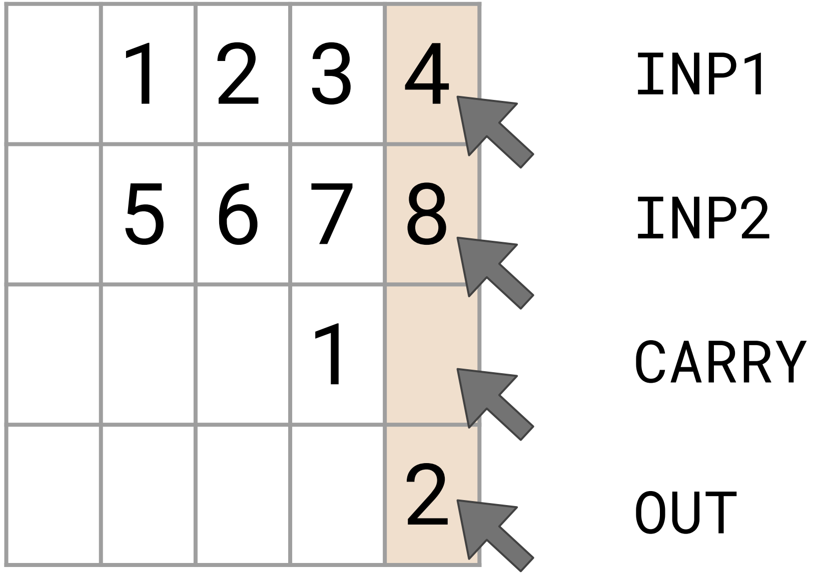
NPI (Reed et al, 2016)

Non-Recursive

Grade-School Addition

✎: change environment ☎: call function

- ☎ ADD
  - ☎ ADD1
    - ✎ WRITE OUT 2
    - ☎ CARRY
      - ✎ PTR CARRY LEFT
      - ✎ WRITE CARRY 1
      - ✎ PTR CARRY RIGHT



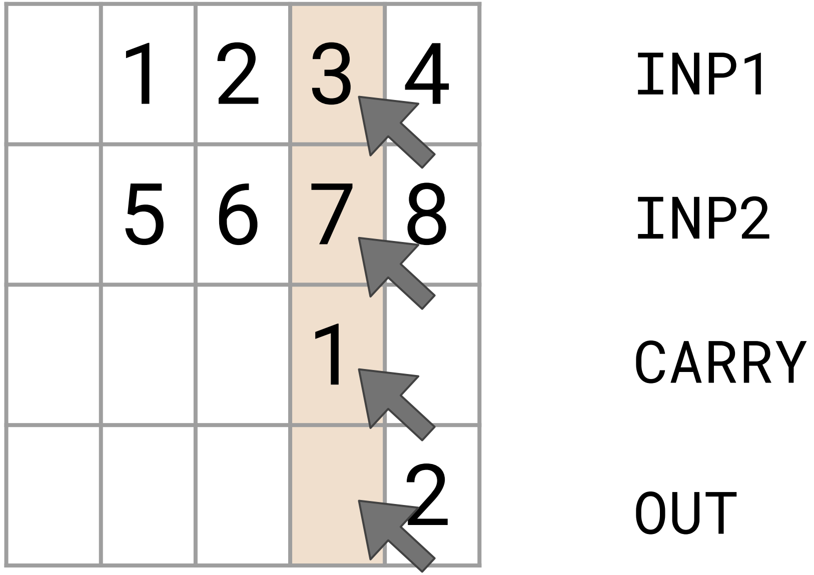
NPI (Reed et al, 2016)

Non-Recursive

Grade-School Addition

✎: change environment ☎: call function

- ☎ ADD
  - ☎ ADD1
  - ☎ LSHIFT
    - ✎ PTR INP1 LEFT
    - ✎ PTR INP2 LEFT
    - ✎ PTR CARRY LEFT
    - ✎ PTR OUT LEFT



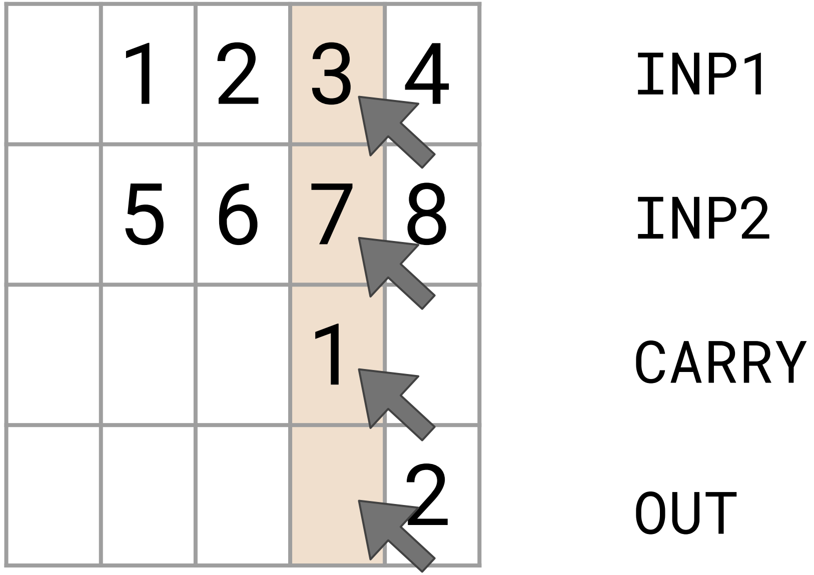
NPI (Reed et al, 2016)

Non-Recursive

Grade-School Addition

✎: change environment ☎: call function

- ☎ ADD
  - ☎ ADD1
  - ☎ LSHIFT



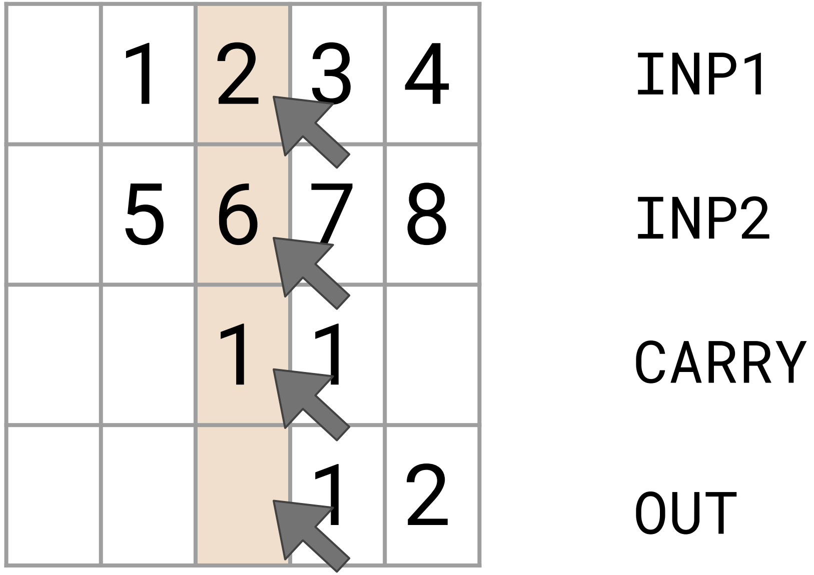
NPI (Reed et al, 2016)

Non-Recursive

Grade-School Addition

✎: change environment ☎: call function

- ☎ ADD
  - ☎ ADD1
  - ☎ LSHIFT
  - ☎ ADD1
  - ☎ LSHIFT



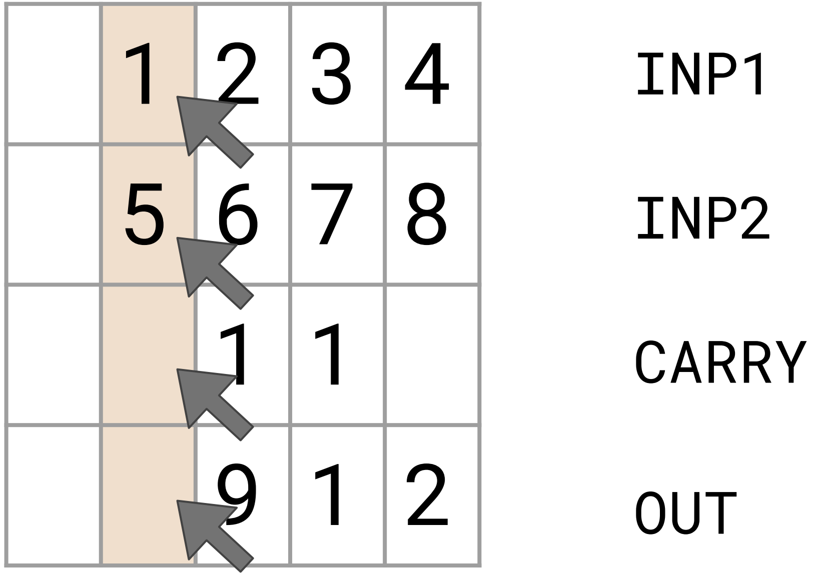
NPI (Reed et al, 2016)

Non-Recursive

Grade-School Addition

✎: change environment ☎: call function

- ☎ ADD
  - ☎ ADD1
  - ☎ LSHIFT
  - ☎ ADD1
  - ☎ LSHIFT
  - ☎ ADD1
  - ☎ LSHIFT



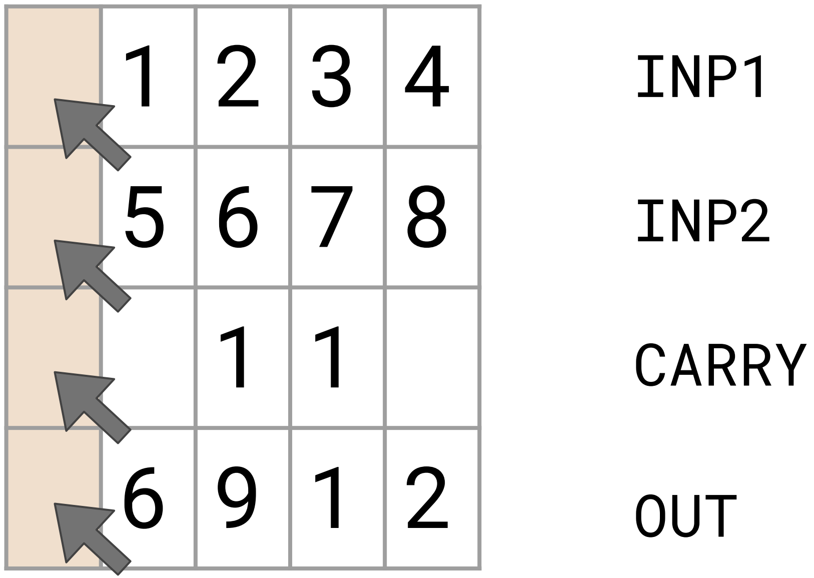
NPI (Reed et al, 2016)

Non-Recursive

Grade-School Addition

✎: change environment ☎: call function

☎ ADD
- ☎ ADD1
- ☎ LSHIFT
- ☎ ADD1
- ☎ LSHIFT
- ☎ ADD1
- ☎ LSHIFT
- ☎ ADD1
- ☎ LSHIFT



NPI (Reed et al, 2016)

Non-Recursive

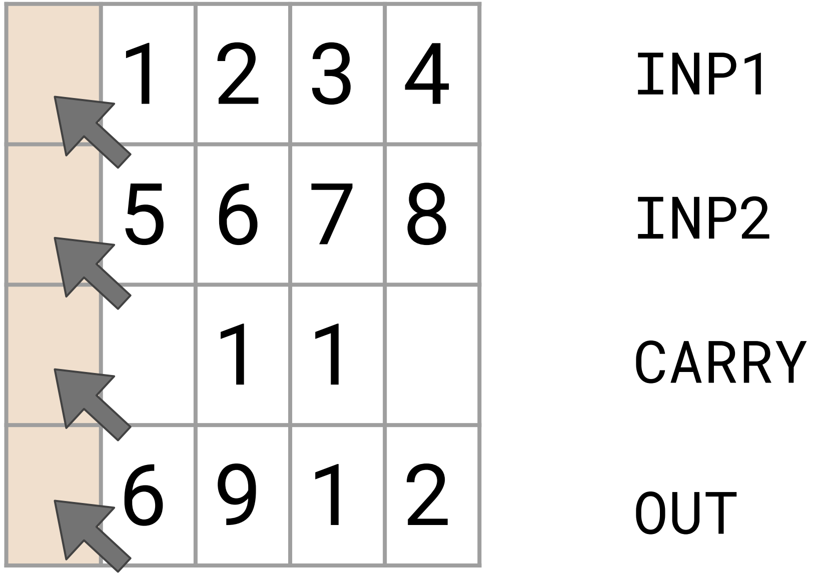
Grade-School Addition

✎: change environment ☎: call function

☎ ADD
- ☎ ADD1
- ☎ LSHIFT

Repeated x4 in **one** call

- ☎ ADD1
- ☎ LSHIFT
- ☎ ADD1
- ☎ LSHIFT
- ☎ ADD1
- ☎ LSHIFT

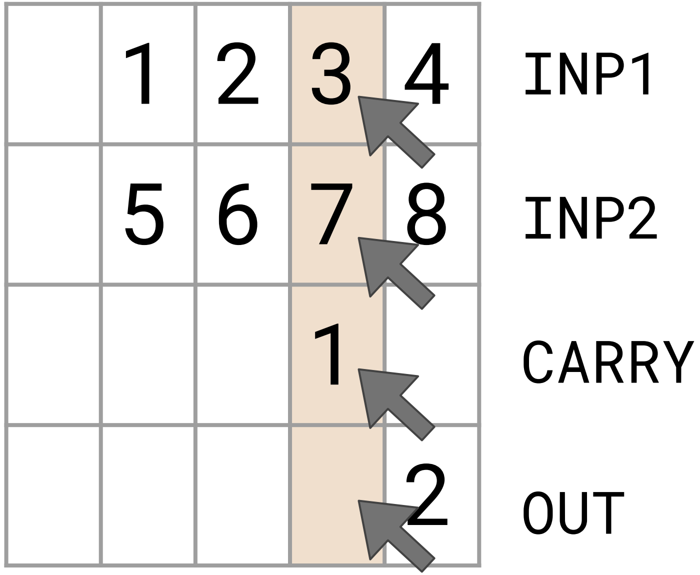


NPI (Reed et al, 2016)

Non-Recursive vs Recursive
Grade-School Addition

✎: change environment ☎: call function

***Non*-recursive (previous work)**	**Recursive (our work)**
☎ ADD	☎ **ADD**
☎ ADD1	☎ ADD1
☎ LSHIFT	☎ LSHIFT



Non-Recursive vs Recursive
Grade-School Addition

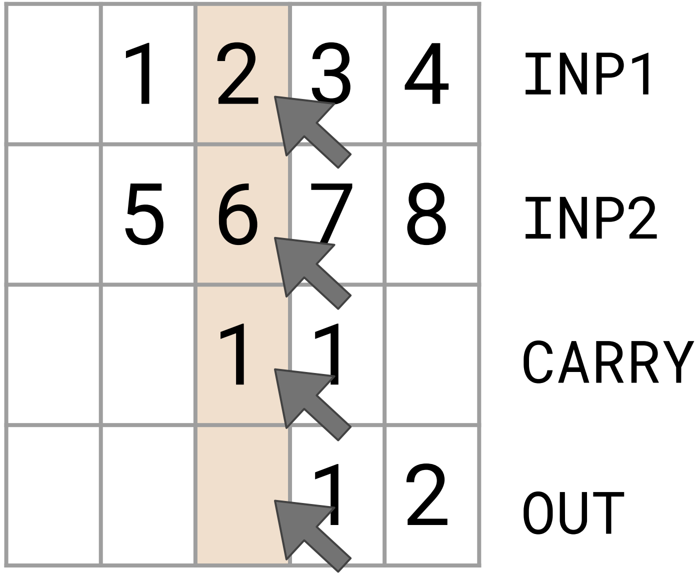
✎: change environment ☎: call function

Non-recursive (previous work)

- ☎ ADD
  - ☎ ADD1
  - ☎ LSHIFT
  - ☎ ADD1
  - ☎ LSHIFT

Recursive (our work)

- ☎ **ADD**
  - ☎ ADD1
  - ☎ LSHIFT
  - ☎ **ADD**
    - ☎ ADD1
    - ☎ LSHIFT



Non-Recursive vs Recursive
Grade-School Addition

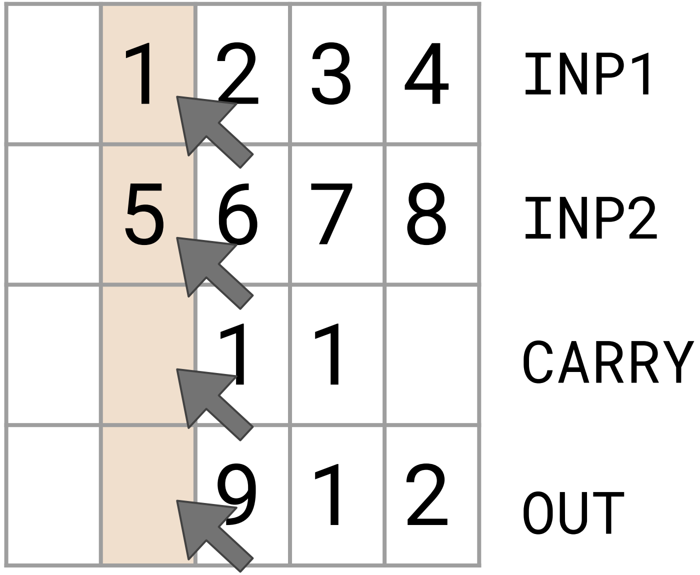
✎: change environment ☎: call function

Non-recursive (previous work)

- ☎ ADD
  - ☎ ADD1
  - ☎ LSHIFT
  - ☎ ADD1
  - ☎ LSHIFT
  - ☎ ADD1
  - ☎ LSHIFT

Recursive (our work)

- ☎ **ADD**
  - ☎ ADD1
  - ☎ LSHIFT
  - ☎ **ADD**
    - ☎ ADD1
    - ☎ LSHIFT
    - ☎ **ADD**
      - ☎ ADD1
      - ☎ LSHIFT



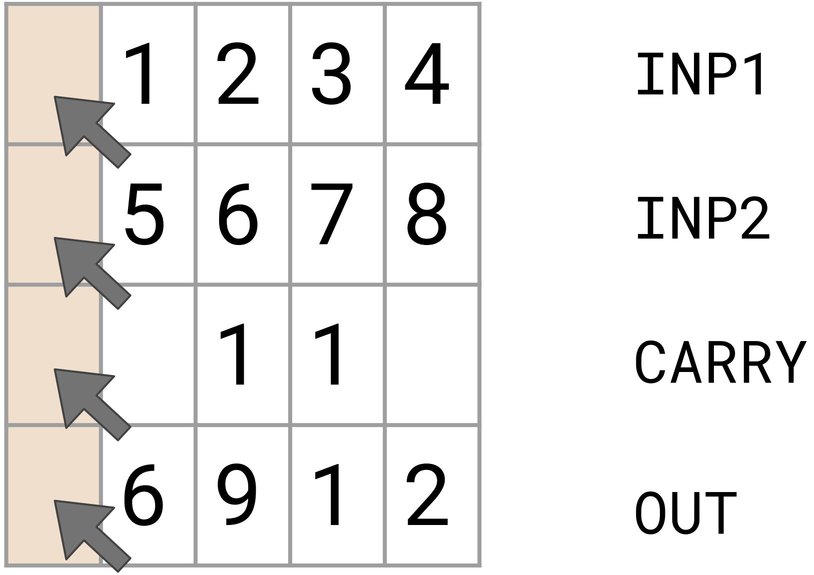
Recursive

Grade-School Addition

✎: change environment ☎: call function

- ☎ **ADD**
  - ☎ ADD1
  - ☎ LSHIFT
  - ☎ **ADD**
    - ☎ ADD1
    - ☎ LSHIFT
    - ☎ **ADD**
      - ☎ ADD1
      - ☎ LSHIFT
      - ☎ **ADD**
      - ...

recursive calls



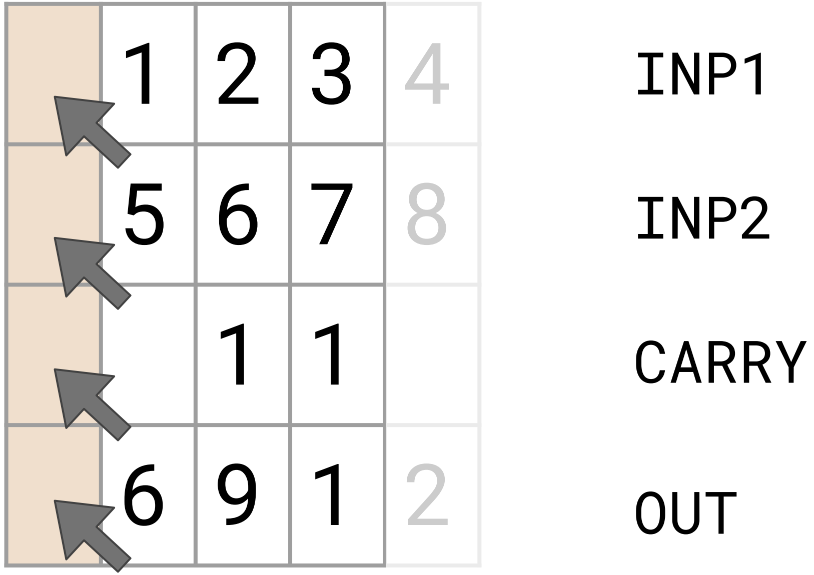
Recursive

Grade-School Addition

✎: change environment ☎: call function

☎ ADD
  ☎ ADD1
  ☎ LSHIFT
  ☎ **ADD**
    ☎ ADD1
    ☎ LSHIFT
    ☎ **ADD**
      ☎ ADD1
      ☎ LSHIFT
      ☎ **ADD**
        ...

recursive calls



Non-Recursive vs Recursive Addition

☎ **ADD**
- ☎ ADD1
- ☎ LSHIFT

Repeated inside **one** function call

- ☎ ADD1
- ☎ LSHIFT
- ☎ ADD1
- ☎ LSHIFT

…

***Non*-recursive execution trace (previous work)**

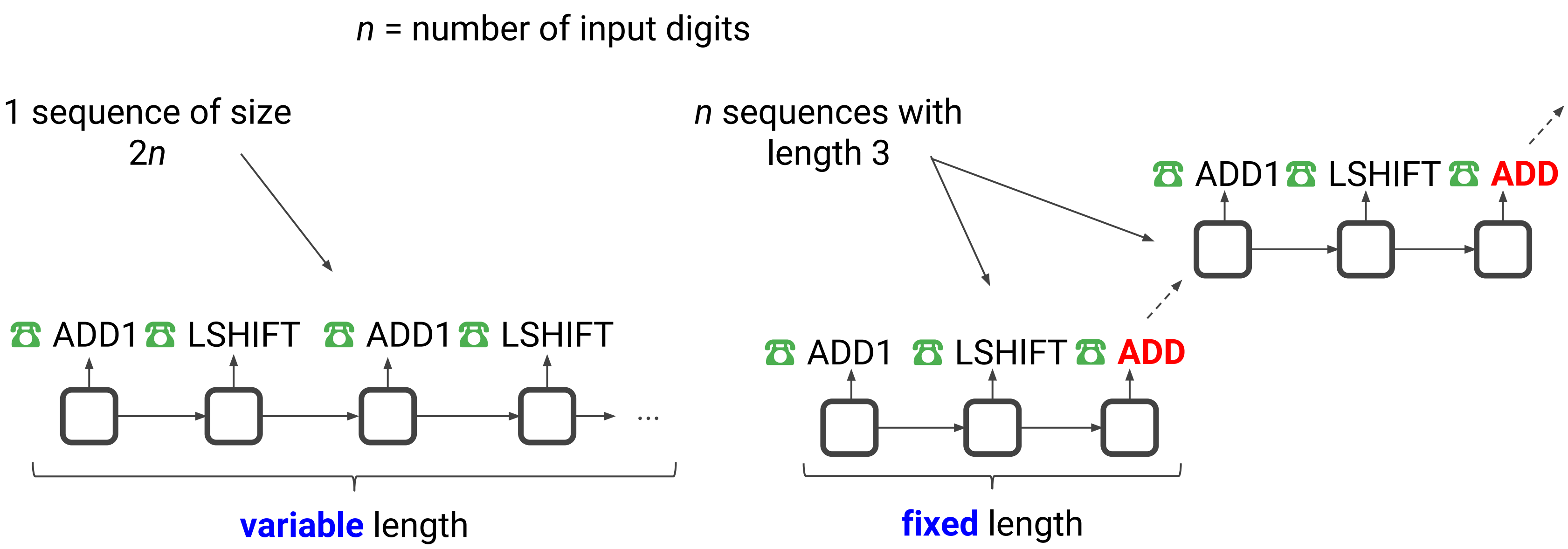
☎ **ADD**
- ☎ ADD1
- ☎ LSHIFT
- ☎ **ADD**
  - ☎ ADD1
  - ☎ LSHIFT
  - ☎ **ADD**

    …

recursive calls

Recursive execution trace (our work)

Non-Recursive vs Recursive Addition



- ✓ Generalization to more complex inputs
- ✓ Proof of generalization

How to Learn a Recursive NPI Program

- In NPI, any function can call any function, including itself
  (but original NPI didn’t explicitly make use of recursive calls)

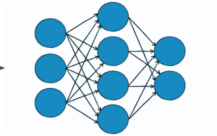
- To learn a recursive NPI program:
  - No architecture change
  - Only change the training data, instead of the architecture

How to Learn a Recursive NPI Program

ADD
- ADD1
- LSHIFT
- ADD1
- LSHIFT
- ADD1
- LSHIFT
- …

Non-recursive training trace

NPI Architecture

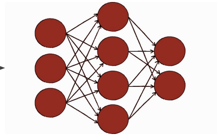


Non-recursive NPI program

ADD
- ADD1
- LSHIFT
- **ADD**
  - ADD1
  - LSHIFT
  - **ADD**
  - …

Recursive training trace

NPI Architecture



Recursive NPI program

Outline

Challenges in Neural Program Architectures

Overview of Our Approach: Recursion

Background: Neural-Programmer Interpreter

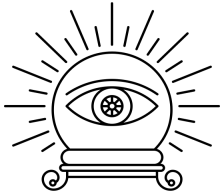
Learning Recursive Neural Programs

➔ Provably Perfect Generalization

Experimental Results

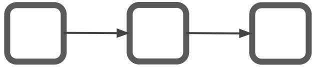
Conclusion

Verifying Perfect Generalization



Oracle
(correct program behavior)

$\stackrel{?}{=}$

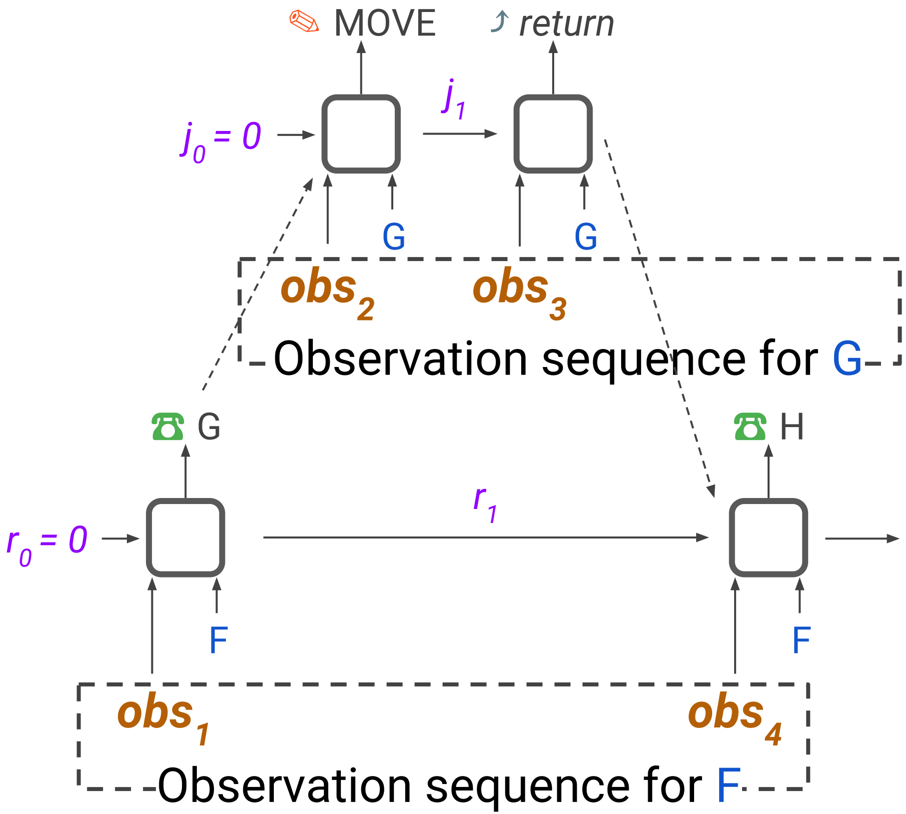


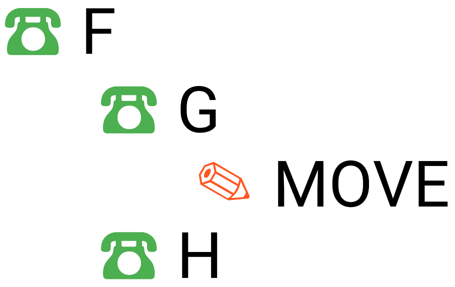
Learned neural program

Image by Daniel Hammer from the Noun Project

Verifying Perfect Generalization

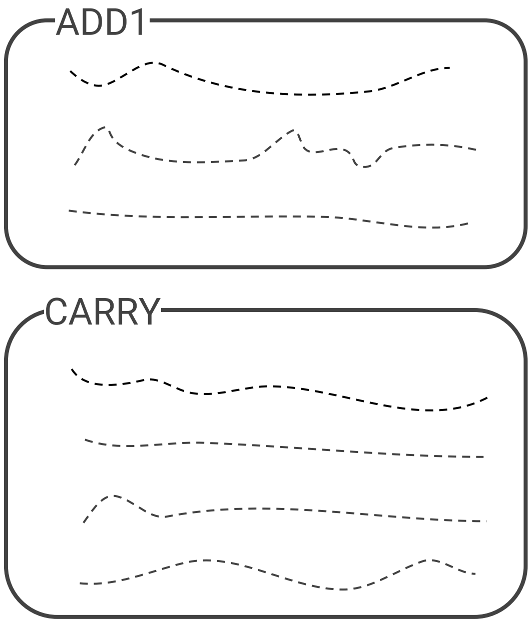
Observation Sequences



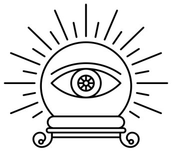


Verifying Perfect Generalization

Creating the Verification Set



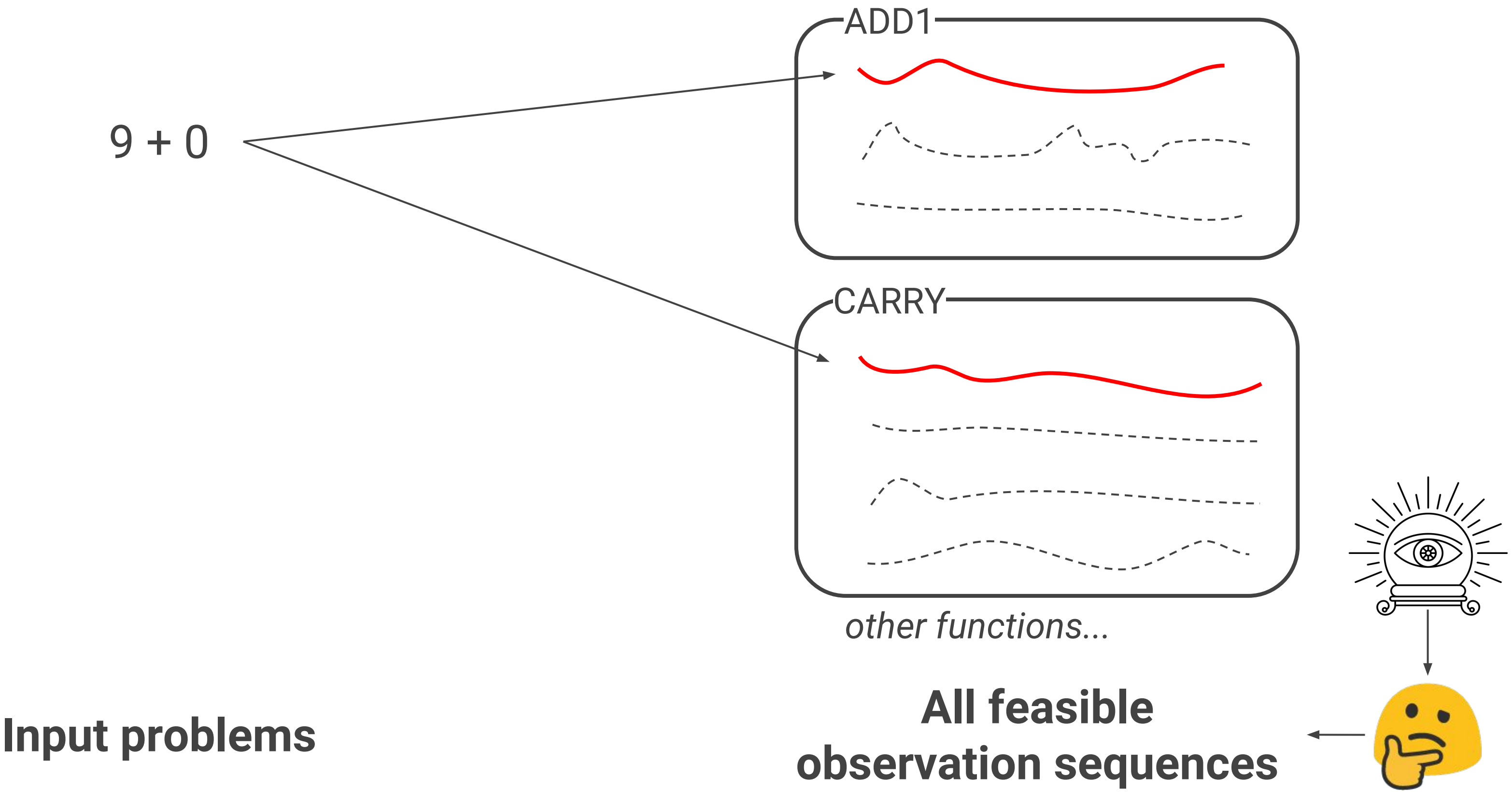
other functions...



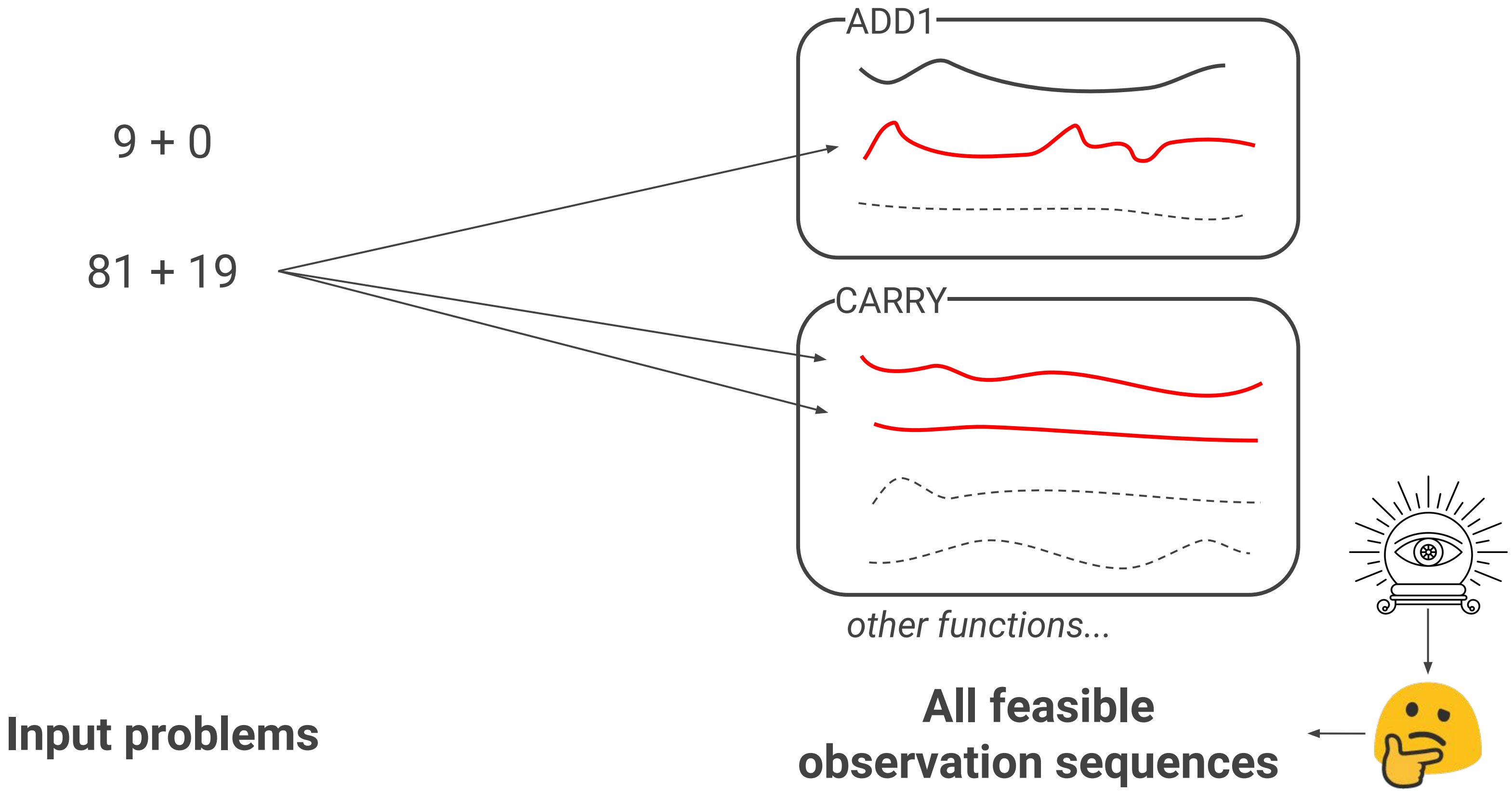
All feasible observation sequences

Verifying Perfect Generalization

Creating the Verification Set



Creating the Verification Set



Verifying Perfect Generalization

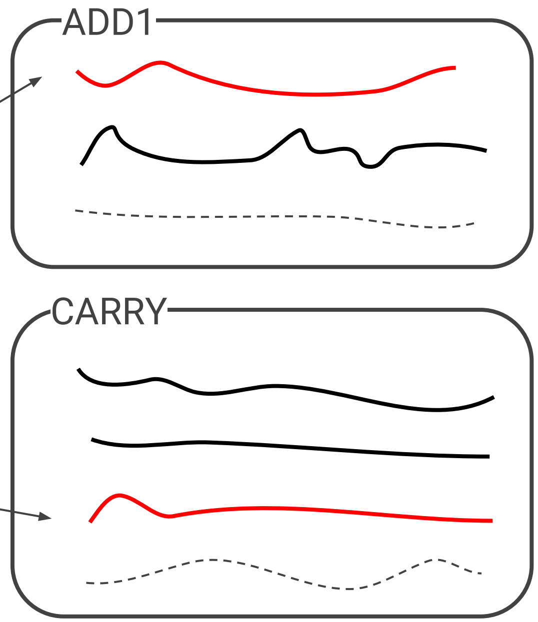
Creating the Verification Set

9 + 0

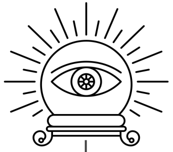
81 + 19

61 + 79

Input problems



other functions...



All feasible observation sequences

Creating the Verification Set

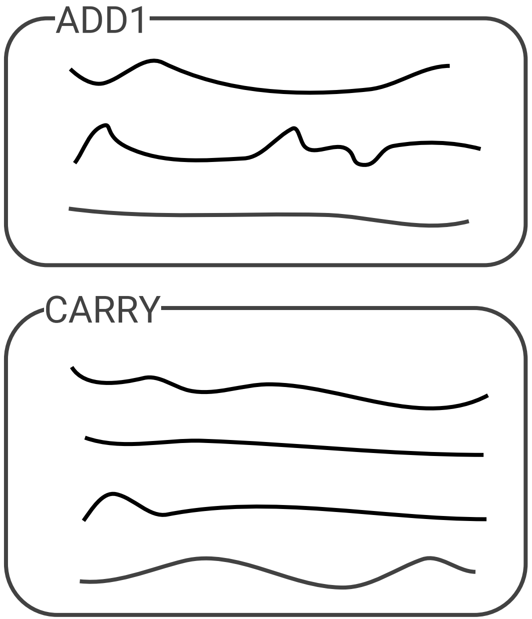
9 + 0

81 + 19

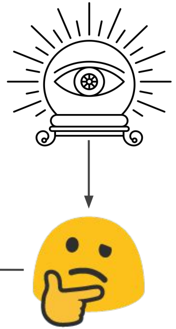
61 + 79

...

Input problems



other functions...



All feasible observation sequences

Creating the Verification Set

9 + 0

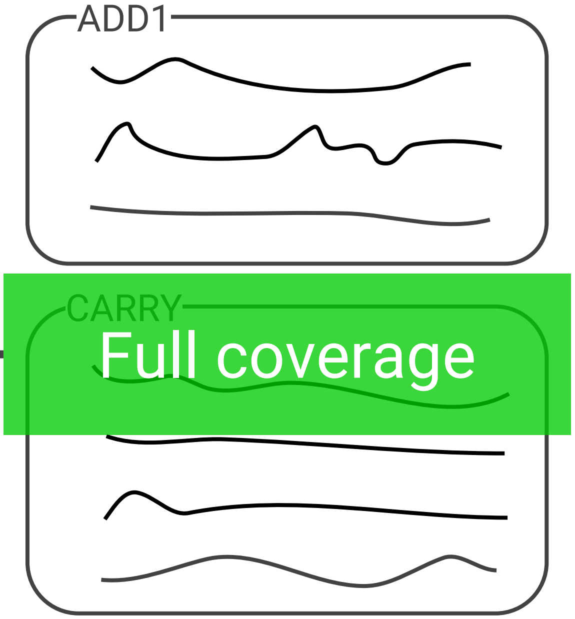
81 + 19

61 + 79

...

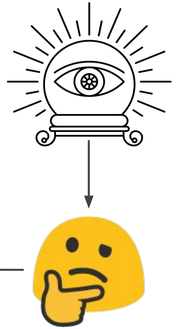
Verification set

Input problems



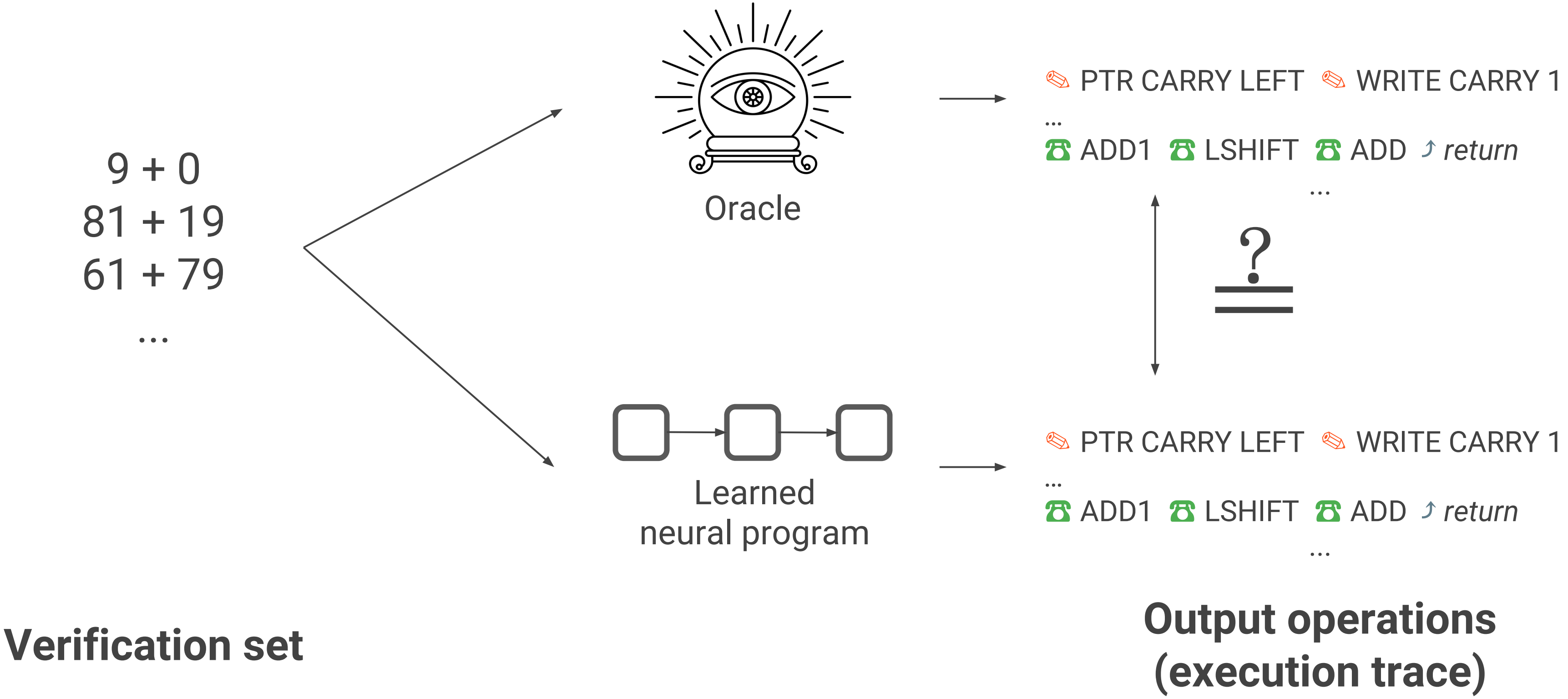
other functions...

All feasible observation sequences



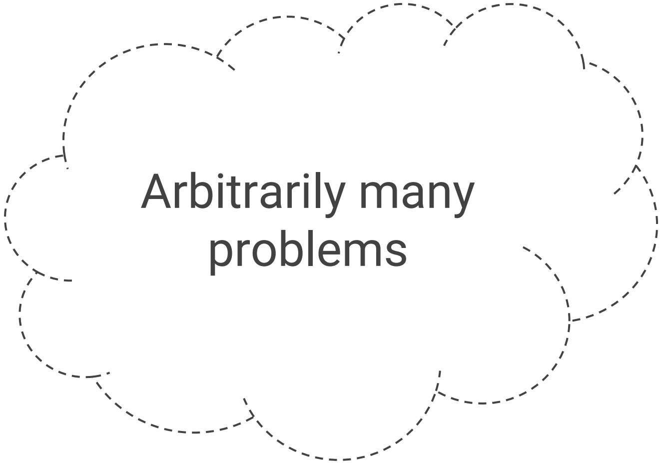
Verifying Perfect Generalization

Oracle Matching

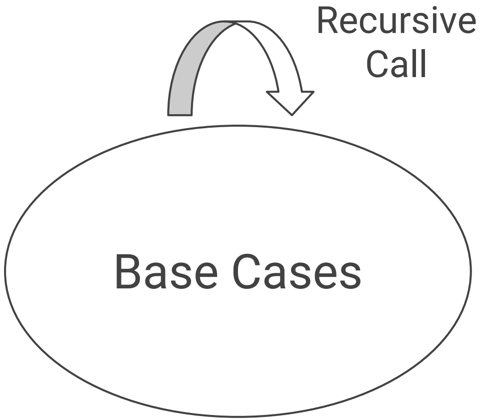


Recursion Induces Boundedness

Neural network needs to solve:



Without recursion (previous work)



With recursion (our work)

Recursion Enables Verification

Recursion allows for a *finite* (and therefore tractable) verification set, for certain domains.

Verification sets for addition:

1+1 = 2 ✔
99 + 99 = 198 ✔
99…99 + 99…99 = … ✔
⋮

∞ cases

Without recursion (previous work)

1+1 = 2 ✔
⋮
99 + 99 = 198 ✔

~20000 cases

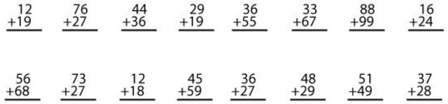
With recursion (our work)

Outline

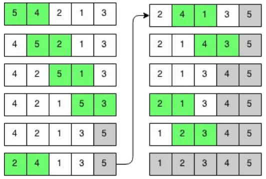
- Challenges in Neural Program Architectures
- Overview of Our Approach: Recursion
- Background: Neural-Programmer Interpreter
- Learning Recursive Neural Programs
- Provably Perfect Generalization
- ➔ Experimental Results
- Conclusion

Tasks in Experiments

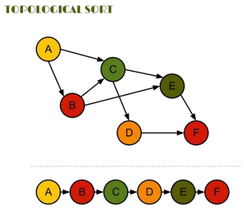
Grade-School Addition



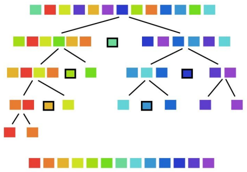
Bubble Sort



Topological Sort NEW!



Quicksort NEW!



Experimental Results

- Experimental setup:
  - Recursive and non-recursive NPI program learned for each task using the same training problems.
  - Both evaluated on same (randomly generated) test problems.

- Empirical results:
  - Learned recursive programs are **100% accurate** on the test problems.
  - Non-recursive program accuracy often degrades on the test problems.

Empirical Accuracy: Quicksort

Length of Array	Non-Recursive	Recursive
3	**100%**	**100%**
5	**100%**	**100%**
7	**100%**	**100%**
11	73.3%	**100%**
15	60%	**100%**
20	30%	**100%**
30	3.33%	**100%**
70	0%	**100%**

Training set: 4 length-5 arrays

Empirical Accuracy: Other Tasks

Bubble Sort

Length	Non-Recursive	Recursive
2	**100%**	**100%**
4	10%	**100%**
20	0%	**100%**
90	0%	**100%**

Training set: 100 length-2 arrays

Topological Sort

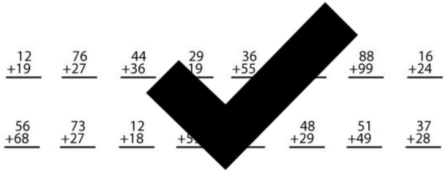
Vertices	Non-Recursive	Recursive
5	6.7%	**100%**
7	3.3%	**100%**
8	0%	**100%**
70	0%	**100%**

Training set: a graph with 5 vertices

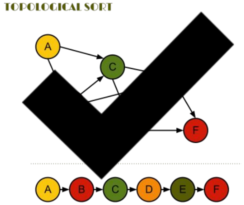
On grade-school addition, both non-recursive and recursive show 100% empirical accuracy (non-recursive matches Reed et al 2016).

Verification of Perfect Generalization

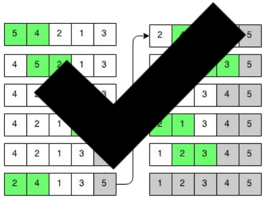
We successfully verified a learned recursive program for each task via the *oracle matching* procedure.



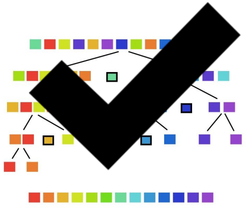
Grade-School Addition



Topological Sort



Bubble Sort



Quicksort

Outline

- Challenges in Neural Program Architectures
- Overview of Our Approach: Recursion
- Background: Neural-Programmer Interpreter
- Learning Recursive Neural Programs
- Provably Perfect Generalization
- Experimental Results
- ➔ Conclusion

Importance of Recursion in Neural Program Architectures

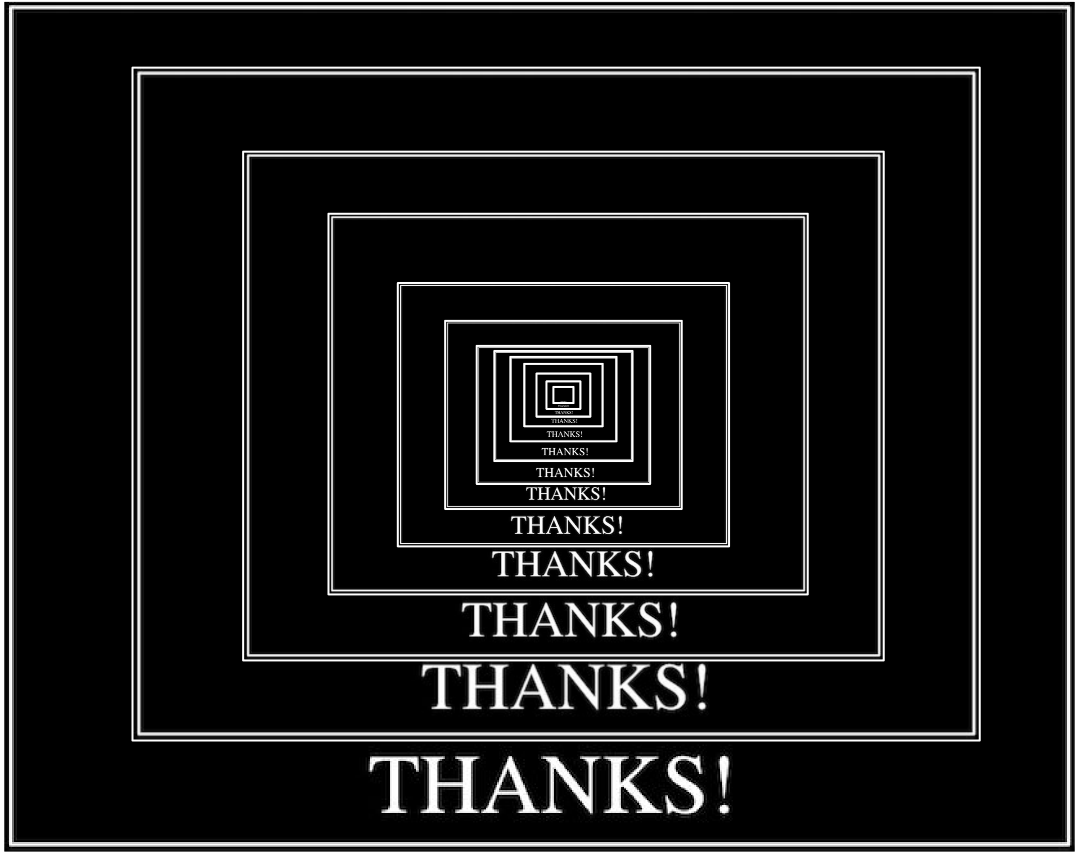
- We introduce recursion, for the first time, into neural program architectures, and learn recursive neural programs

Main Contribution!

- We address two main challenges using recursion:
  - Generalization to more complex inputs
  - Proof of generalization

Learning Recursive Neural Programs

- Our first step instantiation:
  - Architecture: Learn recursive programs in NPI
  - Training method: With explicitly recursive execution traces
- Future work and open questions:
  - Extend to other architectures beyond NPI
  - Learn recursive programs with less supervision
    - Without requiring explicitly recursive training traces
    - Input-output examples instead of execution traces
  - Explore other domains such as perception and control


THANKS!
THANKS!
THANKS!
THANKS!
THANKS!
THANKS!
THANKS!
THANKS!
THANKS!