Neural Programmer-Interpreters

Scott Reed and Nando de Freitas

Google DeepMind

Neural Programmer Interpreter (NPI) goals:

1. **Long-term prediction:** Model potentially long sequences of actions by exploiting compositional structure.

2. **Continual learning:** Learn new programs by composing previously-learned programs, rather than from scratch.

3. **Data efficiency:** Learn generalizable programs from a small number of example traces.

4. **Interpretability**: By looking at NPI's generated commands, we can understand what it is doing at multiple levels of temporal abstraction.

Model

NPI training data

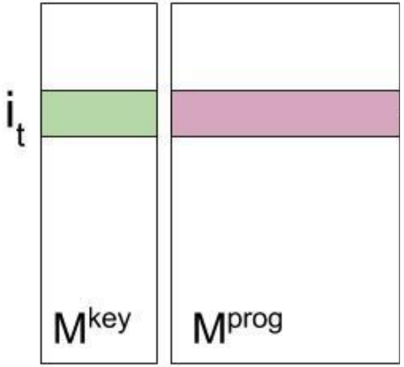
Traces: $\xi_t^{inp} : \{e_t, i_t, a_t\}$ $\xi_t^{out} : \{i_{t+1}, a_{t+1}, r_t\}$

environment observation

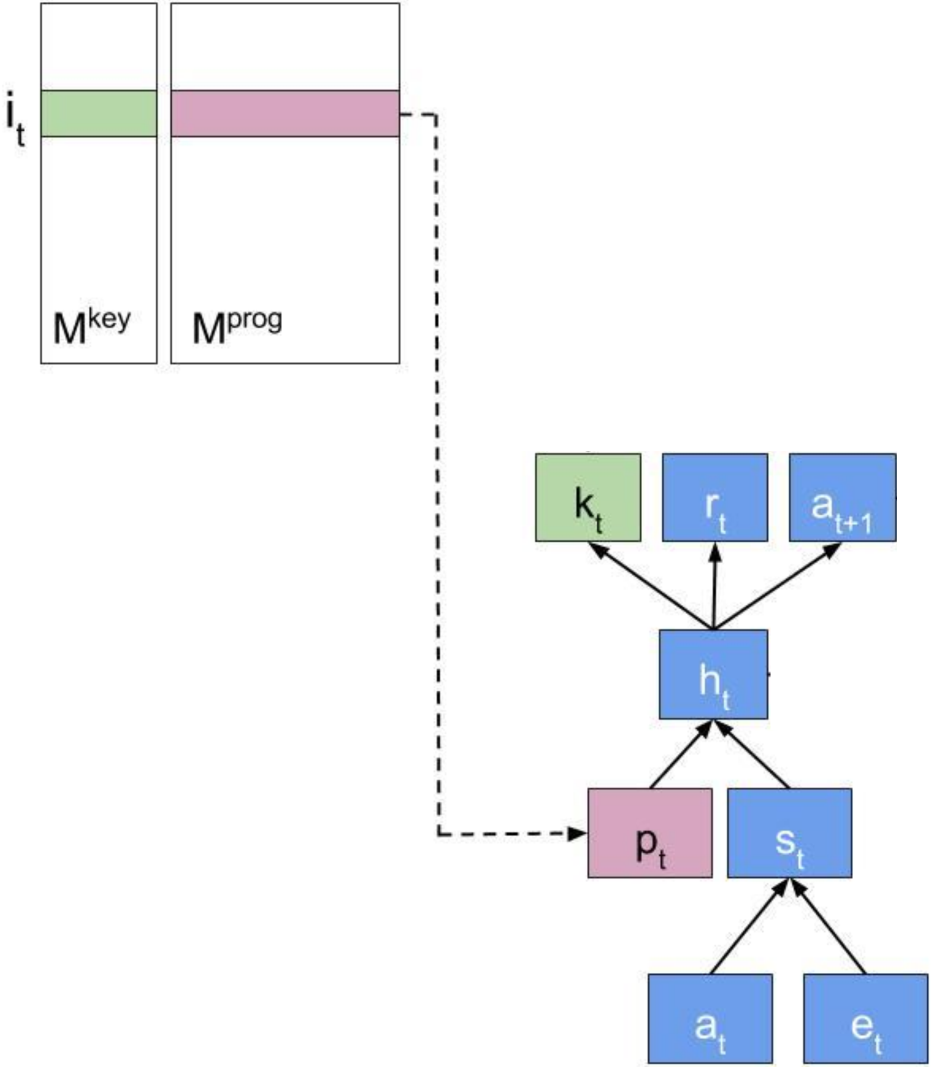
program indices

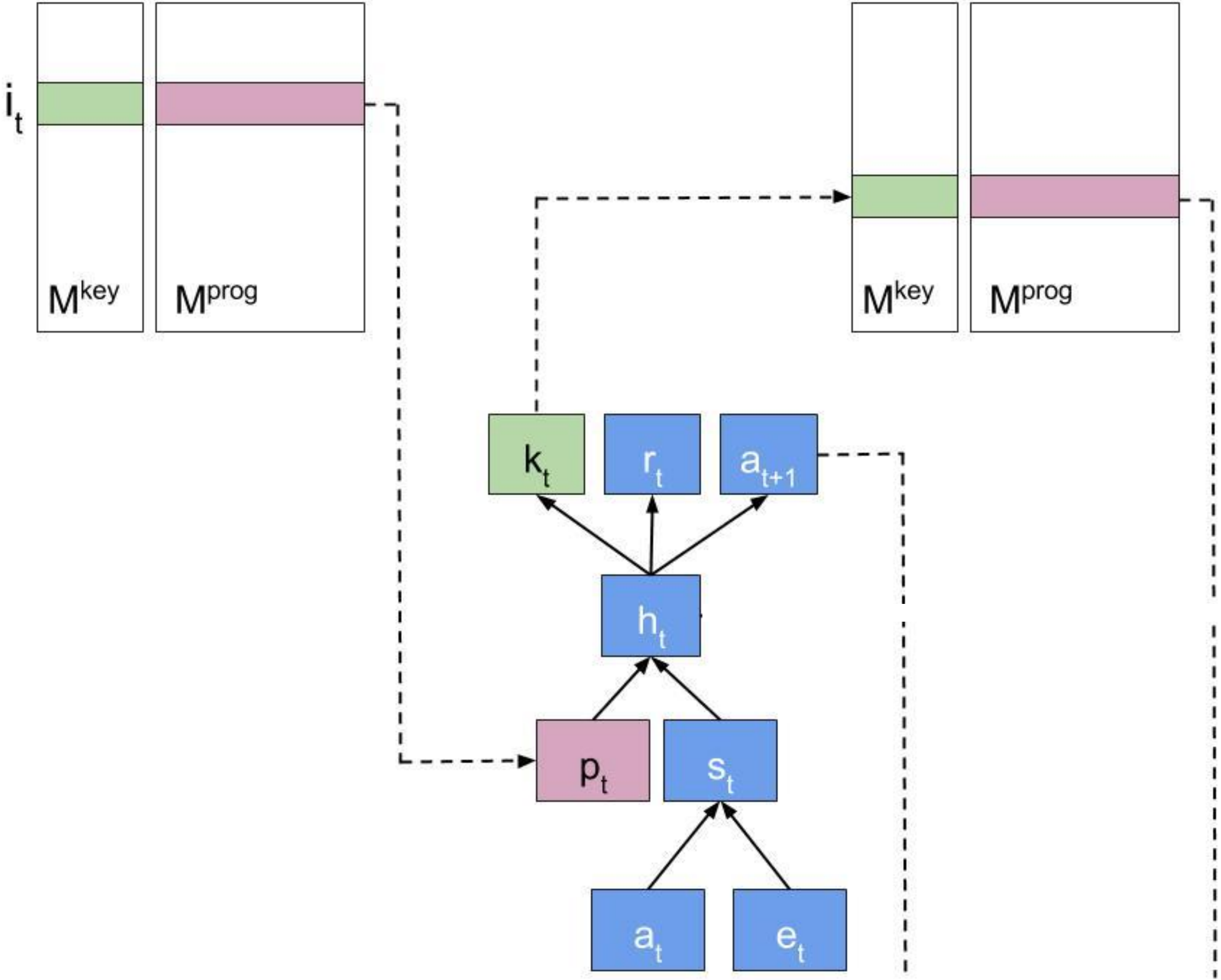
program args

return bit

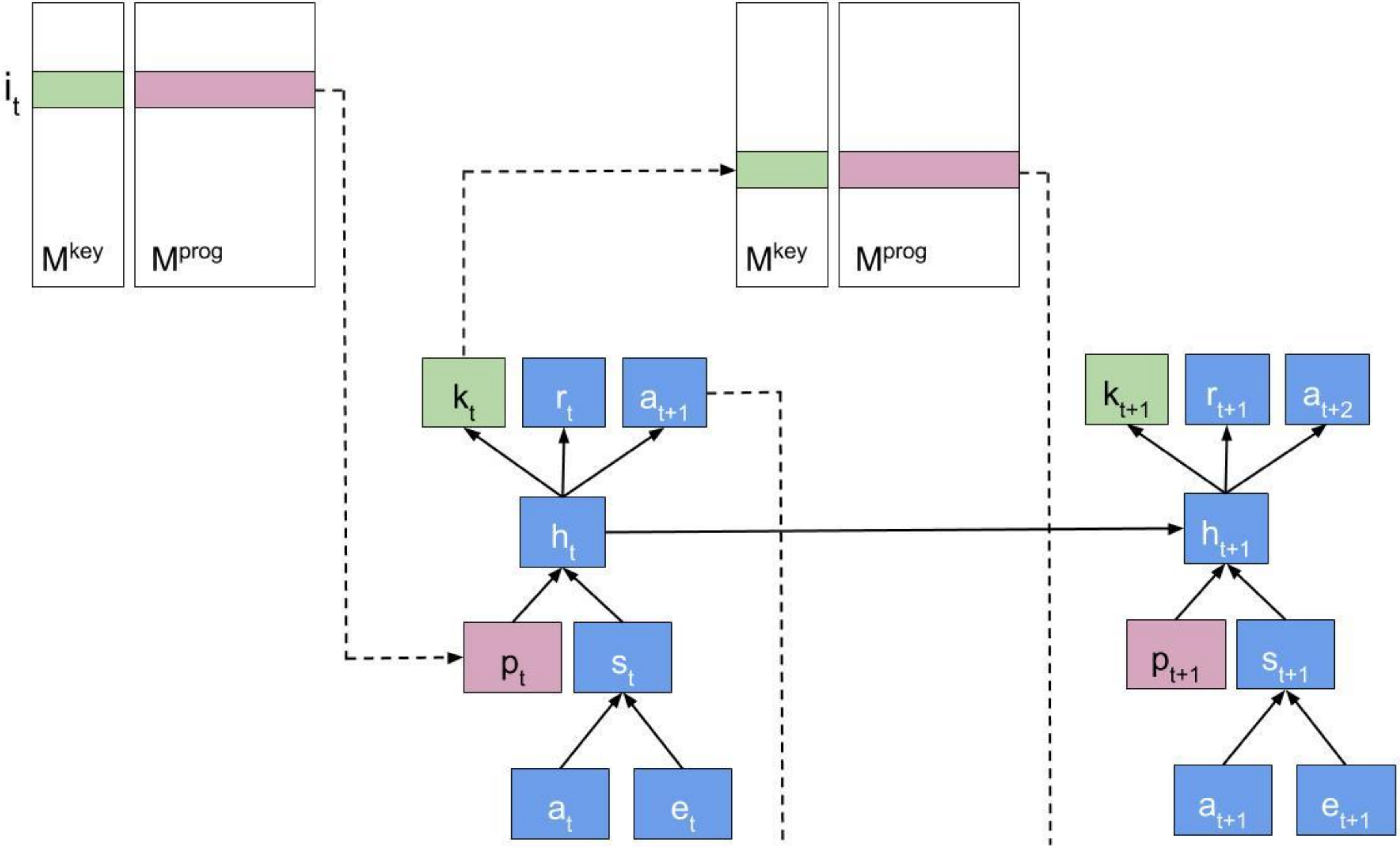


i_t
M^{key}
M^{prog}


i_t
M^{key}
M^{prog}
k_t
r_t
a_{t+1}
h_t
p_t
s_t
a_t
e_t



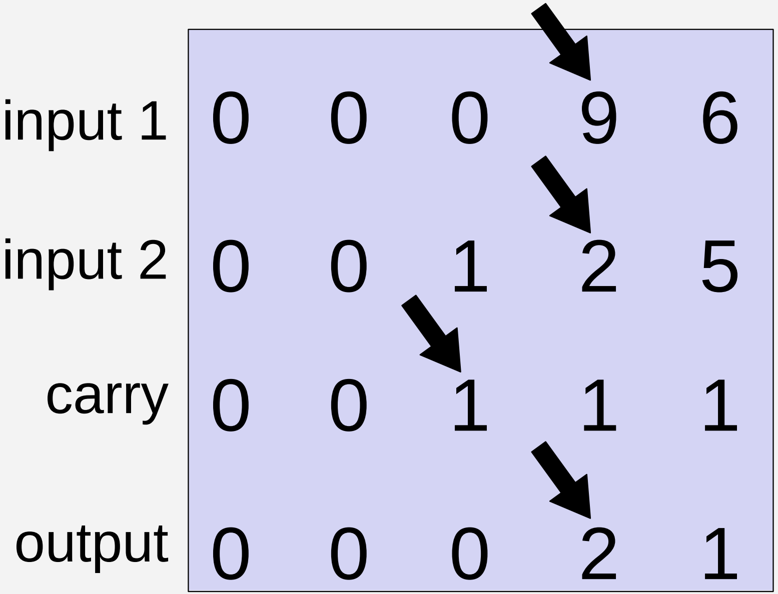
i_t
M^{key}
M^{prog}
M^{key}
M^{prog}
k_t
r_t
a_{t+1}
h_t
p_t
s_t
a_t
e_t



i_t
M^{key}
M^{prog}
M^{key}
M^{prog}
k_t
r_t
a_{t+1}
h_t
p_t
s_t
a_t
e_t
k_{t+1}
r_{t+1}
a_{t+2}
h_{t+1}
p_{t+1}
s_{t+1}
a_{t+1}
e_{t+1}

Demos

Adding numbers together - environment



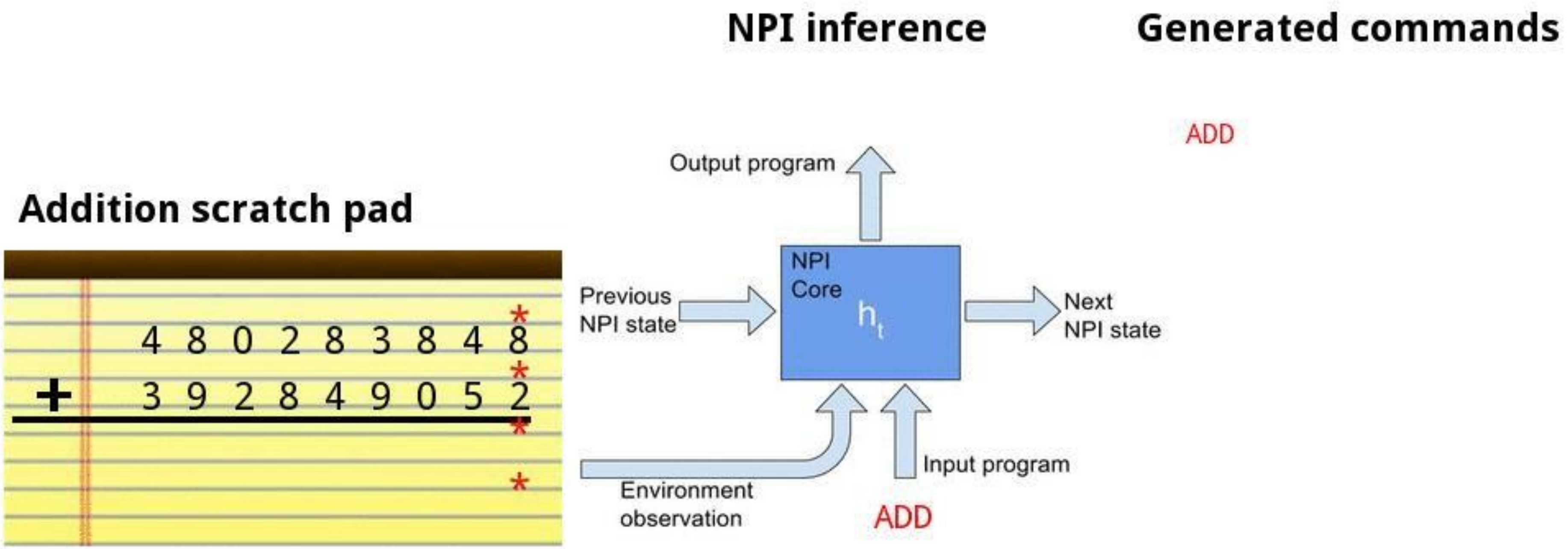
Addition environment interface:

- Scratch pad with the two numbers to be added, a carry row and output row.
- LEFT, RIGHT programs that can move a pointer left or right, respectively.
- WRITE program that writes a specified value to the location of a specified pointer.
- 4 read/write pointers; one per row.

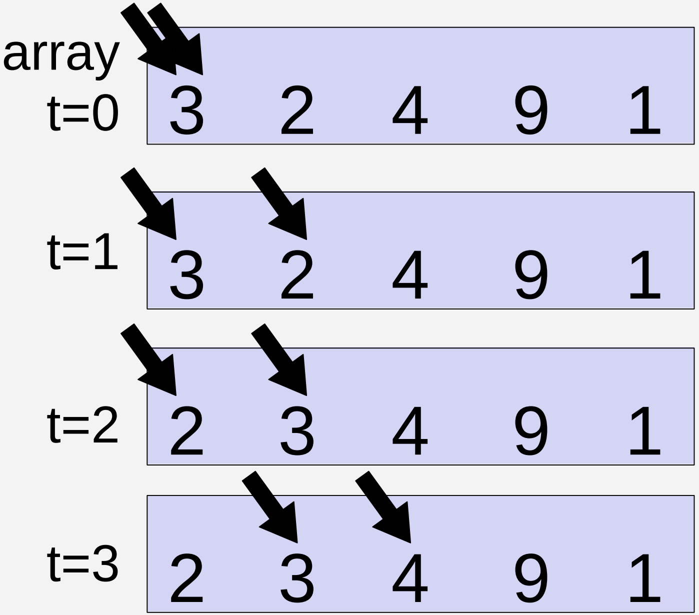
Adding numbers together – learned programs

Program	Description	Calls
ADD	Multi-digit addition	ADD1, LSHIFT
ADD1	Single-digit add	CARRY, ACT
CARRY	Mark a 1 in the carry row 1 step left.	ACT, LSHIFT, RSHIFT
LSHIFT	Shift specified pointer 1 step left.	ACT
RSHIFT	Shift specified pointer 1 step right.	ACT
ACT	Move pointer or write to the scratch pad	-

Adding numbers together



Bubble sort - environment



Sorting environment interface:

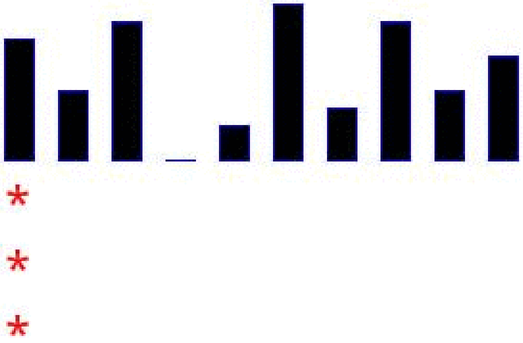
- Scratch pad with the array to be sorted.
- Read/write pointers.
- LEFT, RIGHT programs that can move a specified pointer left or right, respectively.
- SWAP program that swaps the values at two specified pointer locations.

Bubble sort – learned programs

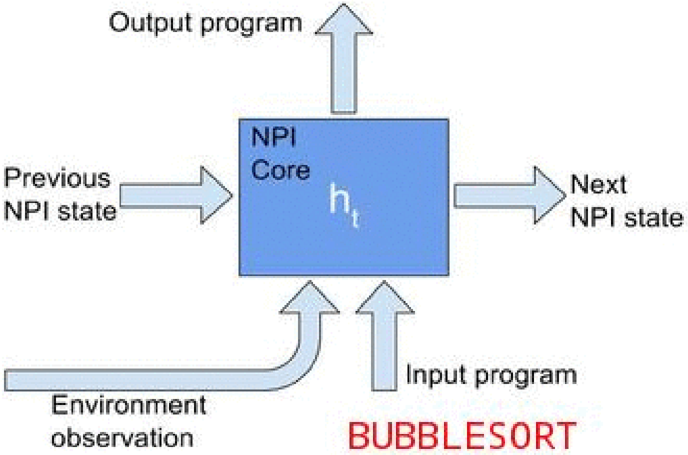
Program	Description	Calls
BUBBLESORT	Sort numbers in ascending order	BUBBLE, RESET
BUBBLE	Perform one sweep of bubble sort	BSTEP, ACT
RESET	Move pointers all back to the left	LSHIFT
BSTEP	Conditionally swap and advance pointers	COMPSWAP, RSHIFT
COMPSWAP	Conditionally swap two pointer values	ACT
LSHIFT	Shift specified pointer 1 step left.	ACT
RSHIFT	Shift specified pointer 1 step right.	ACT
ACT	Perform a swap or move a pointer.	-

Bubble sort

Input array



NPI inference



Generated commands

BUBBLESORT

3D car models - environment

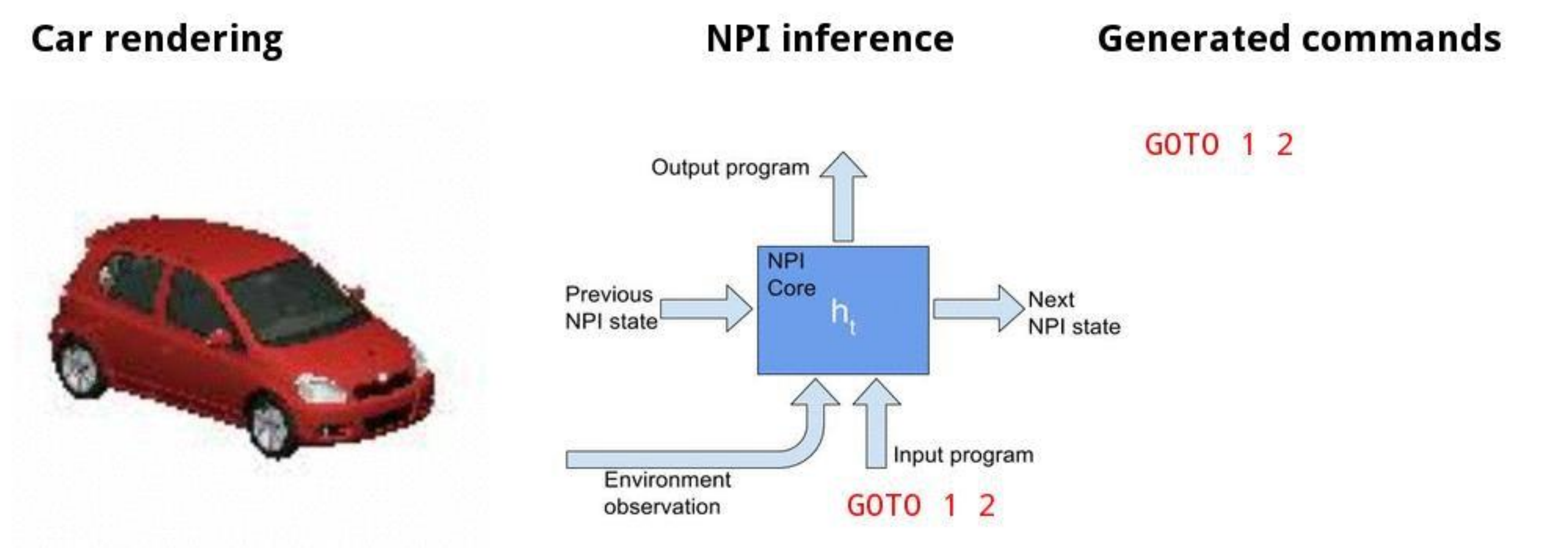
3D cars environment interface:

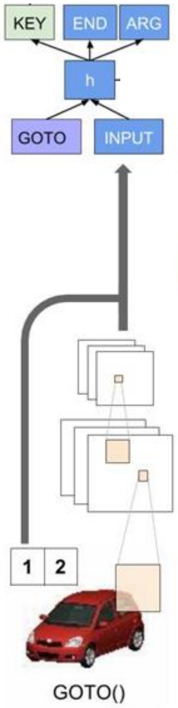
- Rendering of the car (pixels).
- Target angle and elevation coordinates.
- LEFT, RIGHT, UP, DOWN programs that can move the car 15 degrees at a time.
- **The current car pose is NOT provided**.

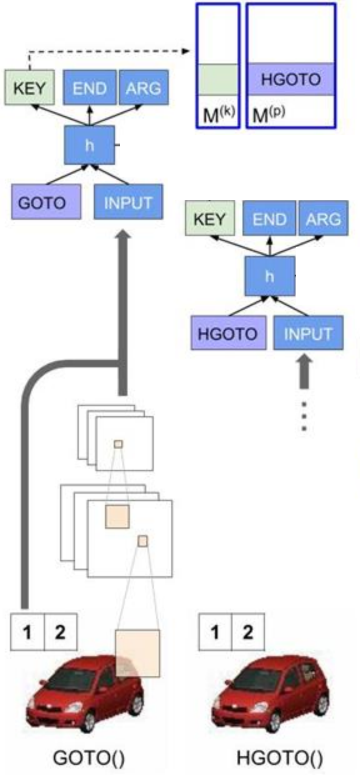
3D car models – learned programs

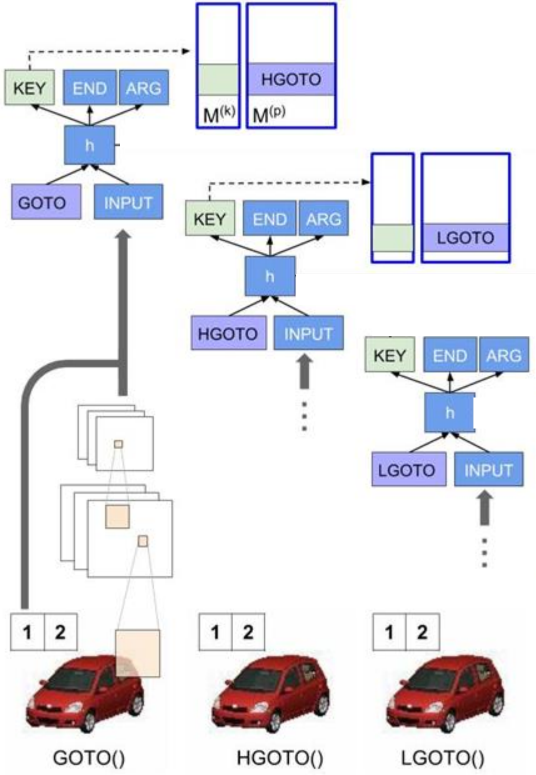
Program	Description	Calls
GOTO	Change 3D car pose to match target	HGOTO, VGOTO
HGOTO	Move horizontally to target angle	LGOTO, RGOTO
LGOTO	Move left to target	ACT
RGOTO	Move right to target	ACT
VGOTO	Move vertically to target elevation	UGOTO, DGOTO
UGOTO	Move up to target	ACT
DGOTO	Move down to target	ACT
ACT	Move 15 degrees up, down, left or right.	-

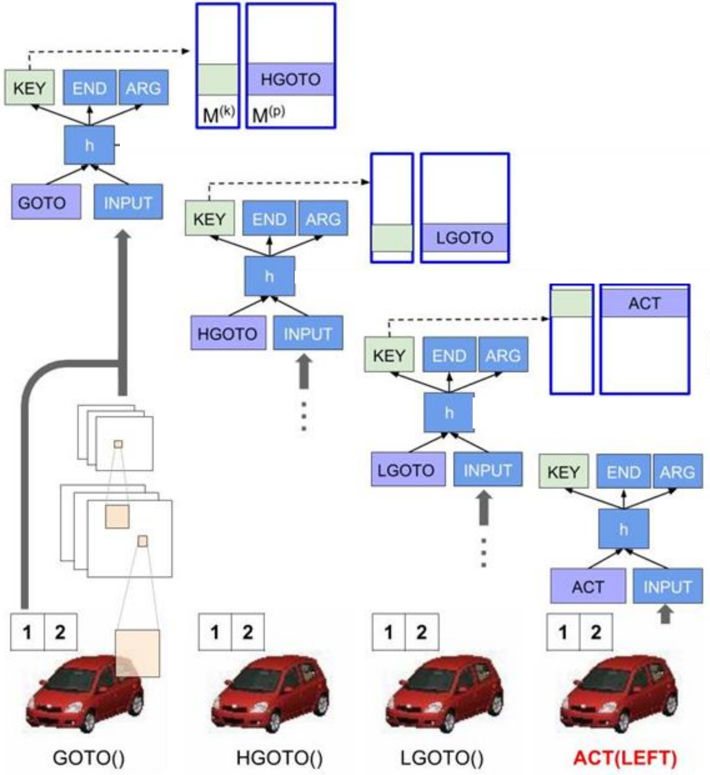
Canonicalizing the view of 3D car models

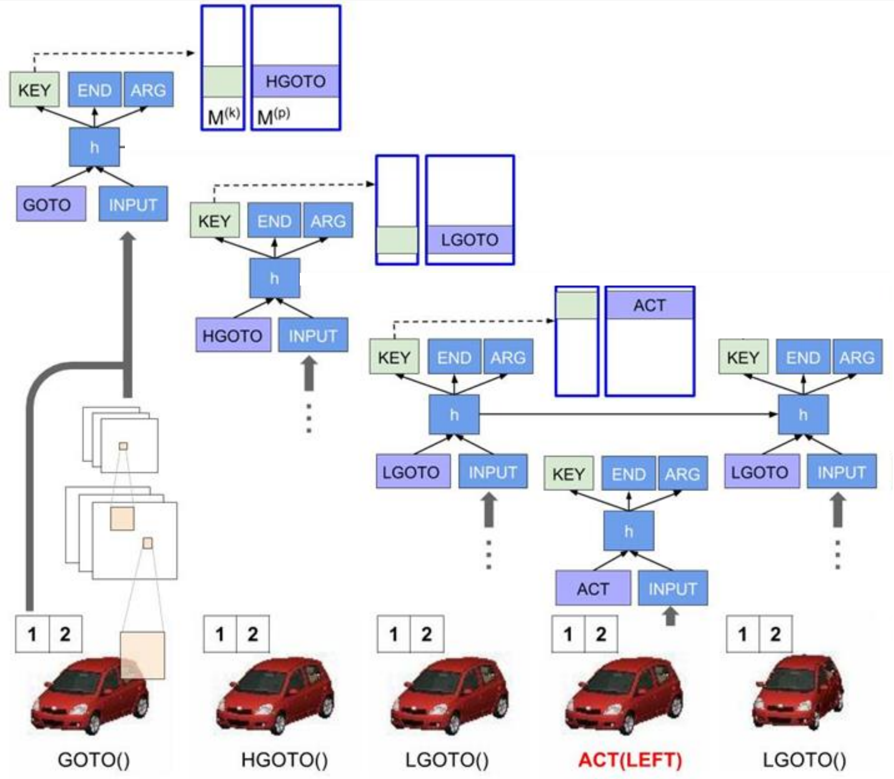



KEY
END
ARG
h
GOTO
INPUT
1
2
GOTO()

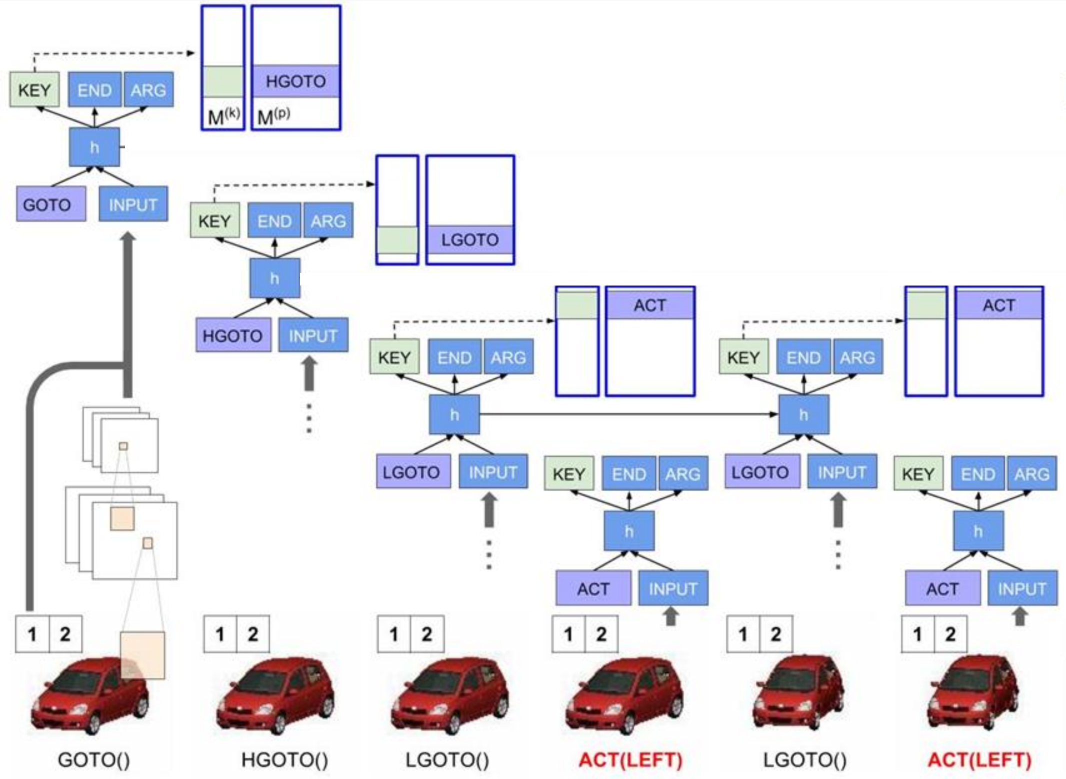

KEY
END
ARG
M(k)
HGOTO
M(p)
h
GOTO
INPUT
KEY
END
ARG
h
HGOTO
INPUT
1
2
1
2
GOTO()
HGOTO()

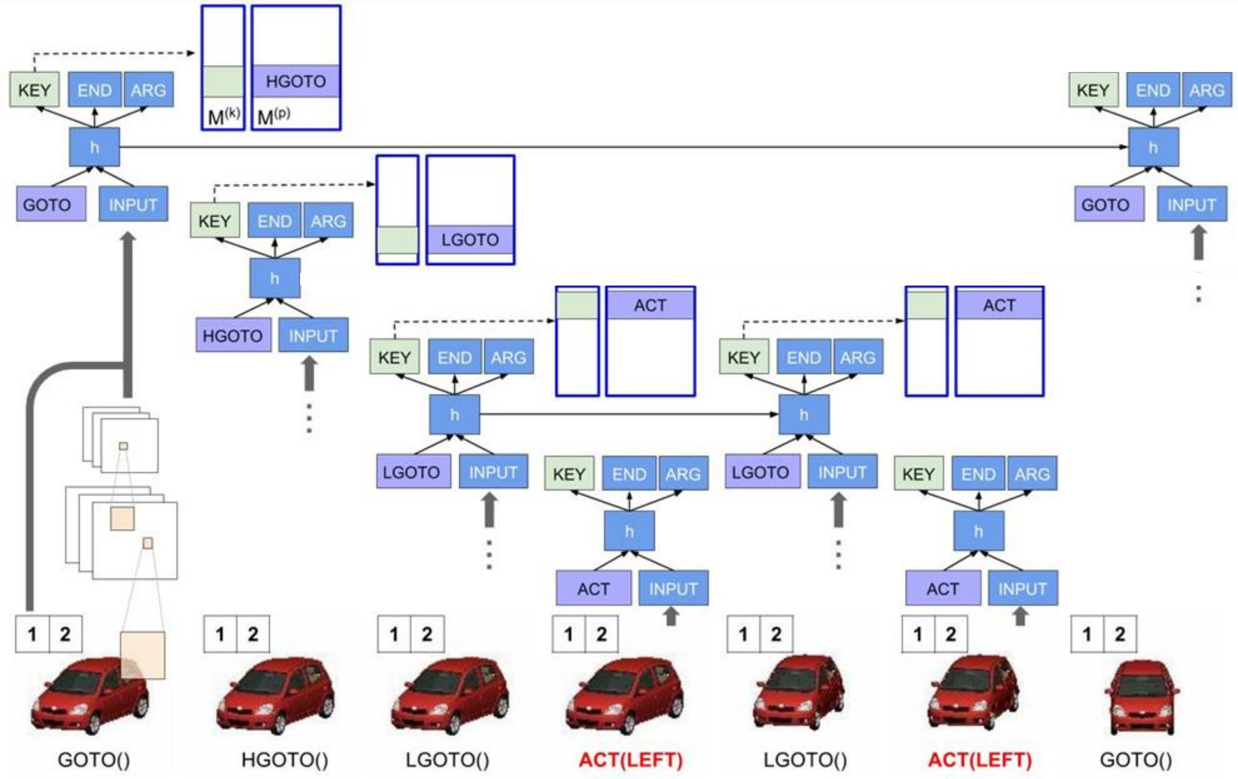

KEY
END
ARG
HGOTO
M(k)
M(p)
h
GOTO
INPUT
KEY
END
ARG
LGOTO
h
HGOTO
INPUT
KEY
END
ARG
h
LGOTO
INPUT
1
2
1
2
1
2
GOTO()
HGOTO()
LGOTO()

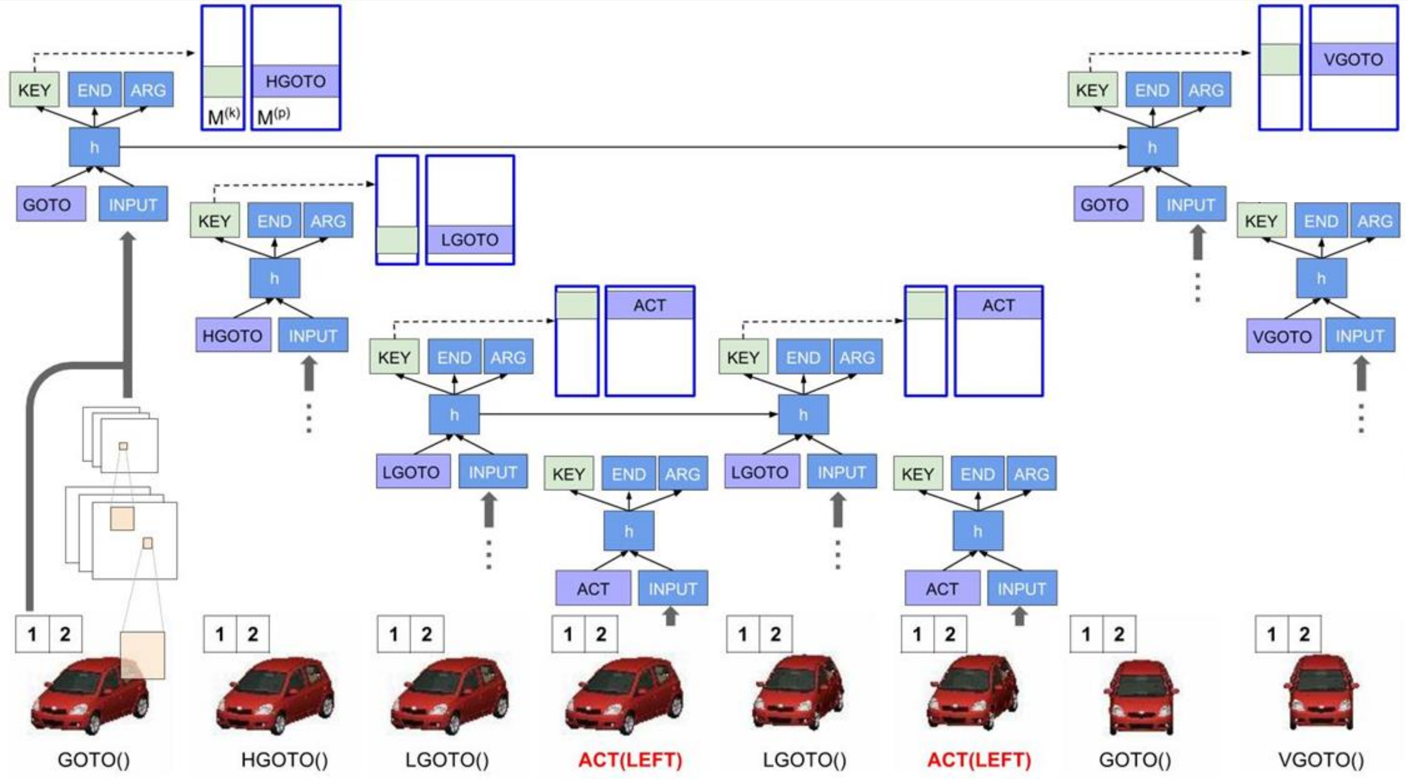

KEY
END
ARG
HGOTO
$M^{(k)}$
$M^{(p)}$
h
GOTO
INPUT
KEY
END
ARG
LGOTO
h
HGOTO
INPUT
ACT
KEY
END
ARG
h
LGOTO
INPUT
KEY
END
ARG
h
ACT
INPUT
1
2
1
2
1
2
1
2
GOTO()
HGOTO()
LGOTO()
ACT(LEFT)

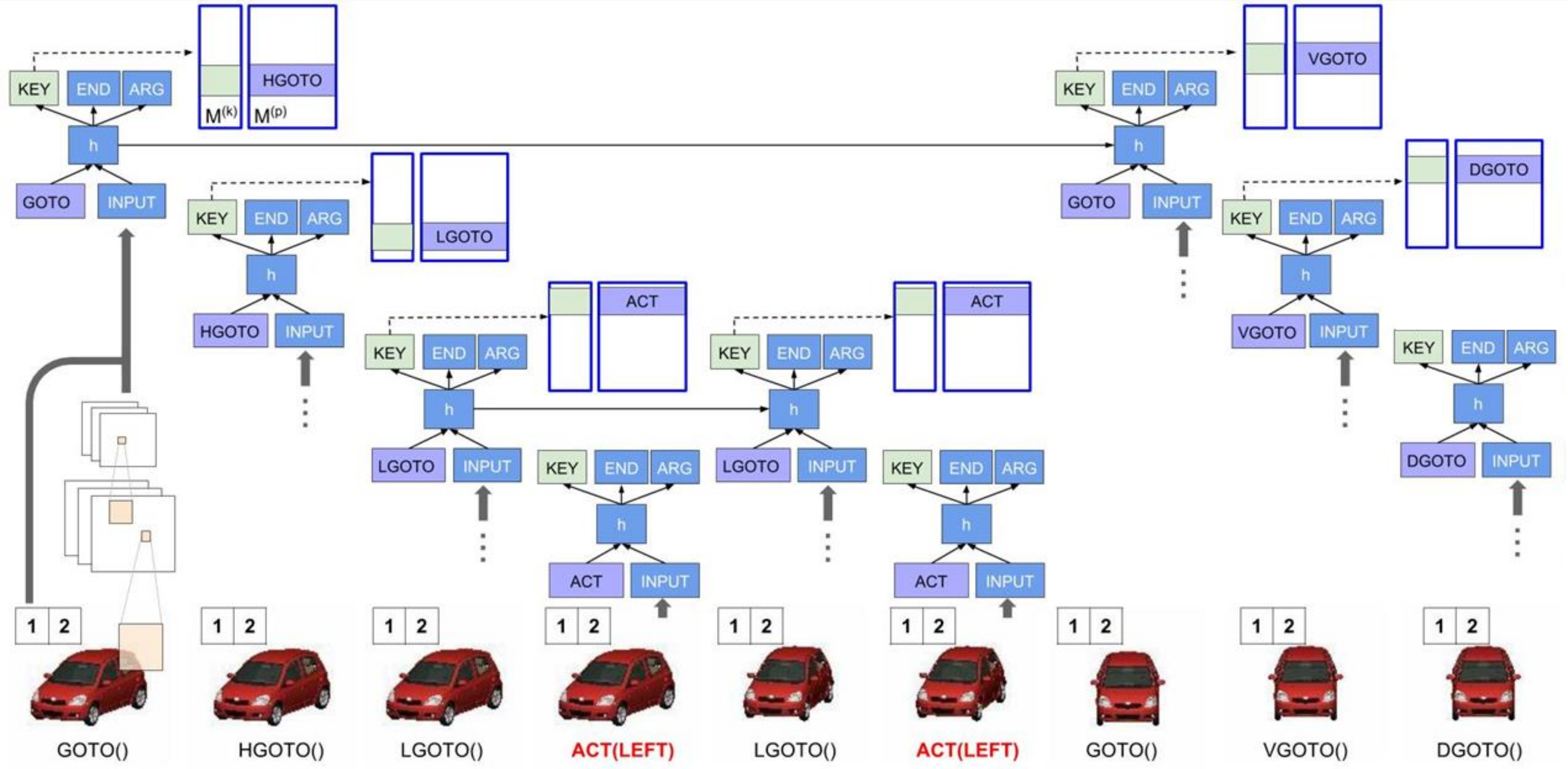


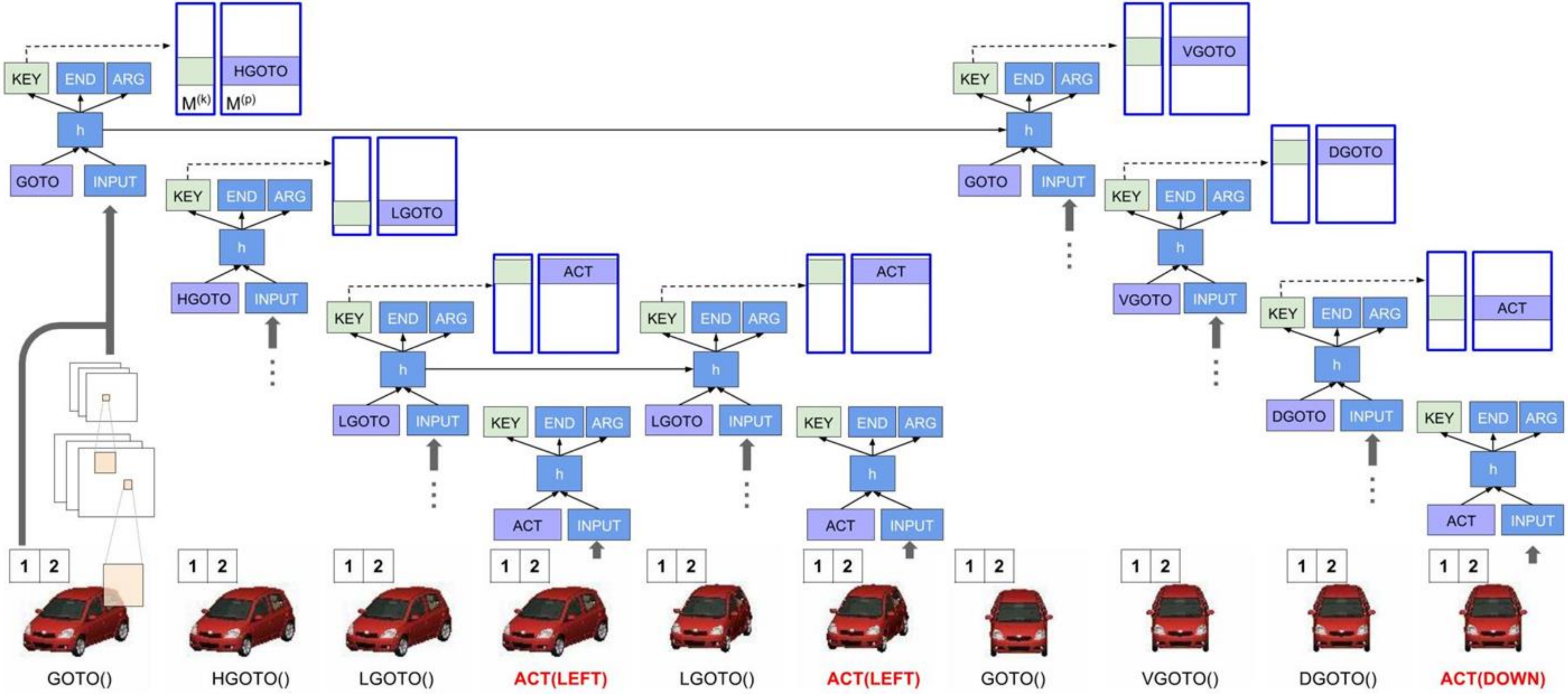
KEY
END
ARG
h
GOTO
INPUT
HGOTO
M(k)
M(p)
KEY
END
ARG
h
HGOTO
INPUT
LGOTO
KEY
END
ARG
h
LGOTO
INPUT
ACT
KEY
END
ARG
h
LGOTO
INPUT
KEY
END
ARG
h
ACT
INPUT
1 2
1 2
1 2
1 2
1 2
GOTO()
HGOTO()
LGOTO()
ACT(LEFT)
LGOTO()

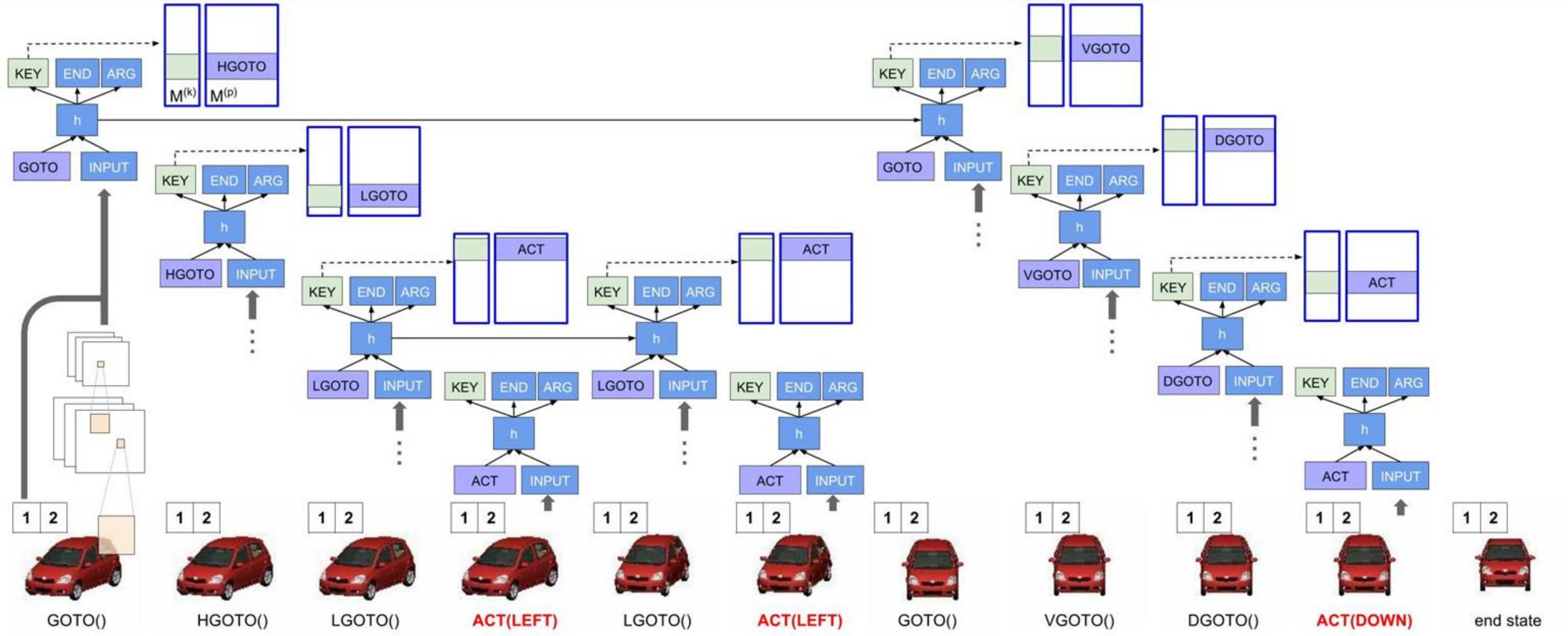

KEY
END
ARG
HGOTO
M(k)
M(p)
h
GOTO
INPUT
KEY
END
ARG
LGOTO
h
HGOTO
INPUT
ACT
KEY
END
ARG
ACT
KEY
END
ARG
h
h
LGOTO
INPUT
KEY
END
ARG
LGOTO
INPUT
KEY
END
ARG
h
h
ACT
INPUT
ACT
INPUT
1 2
GOTO()
HGOTO()
LGOTO()
ACT(LEFT)
LGOTO()
ACT(LEFT)


KEY
END
ARG
HGOTO
M(k)
M(p)
h
GOTO
INPUT
LGOTO
ACT
1
2
GOTO()
HGOTO()
LGOTO()
ACT(LEFT)
LGOTO()
ACT(LEFT)
GOTO()


KEY
END
ARG
h
GOTO
INPUT
HGOTO
LGOTO
ACT
VGOTO
M(k)
M(p)
1
2
GOTO()
HGOTO()
LGOTO()
ACT(LEFT)
LGOTO()
ACT(LEFT)
GOTO()
VGOTO()


KEY
END
ARG
HGOTO
$M^{(k)}$
$M^{(p)}$
h
GOTO
INPUT
LGOTO
ACT
VGOTO
DGOTO
1
2
GOTO()
HGOTO()
LGOTO()
ACT(LEFT)
LGOTO()
ACT(LEFT)
GOTO()
VGOTO()
DGOTO()


KEY
END
ARG
M(k)
M(p)
HGOTO
h
GOTO
INPUT
LGOTO
ACT
VGOTO
DGOTO
1
2
GOTO()
HGOTO()
LGOTO()
ACT(LEFT)
LGOTO()
ACT(LEFT)
GOTO()
VGOTO()
DGOTO()
ACT(DOWN)


KEY
END
ARG
h
GOTO
INPUT
HGOTO
$M^{(k)}$
$M^{(p)}$
LGOTO
ACT
VGOTO
DGOTO
1
2
GOTO()
HGOTO()
LGOTO()
ACT(LEFT)
LGOTO()
ACT(LEFT)
GOTO()
VGOTO()
DGOTO()
ACT(DOWN)
end state

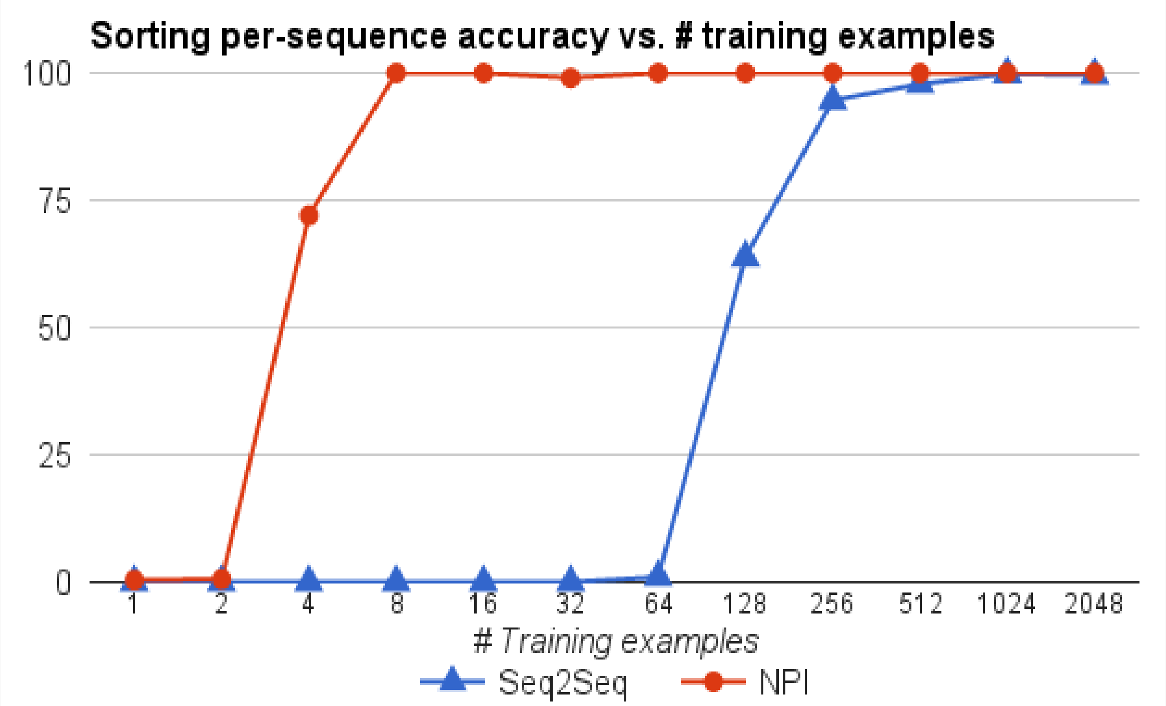
Experiments

Data Efficiency – Sorting

Seq2Seq LSTM and NPI used the same number of layers
and hidden units.

Trained on length 20 arrays of single-digit numbers.

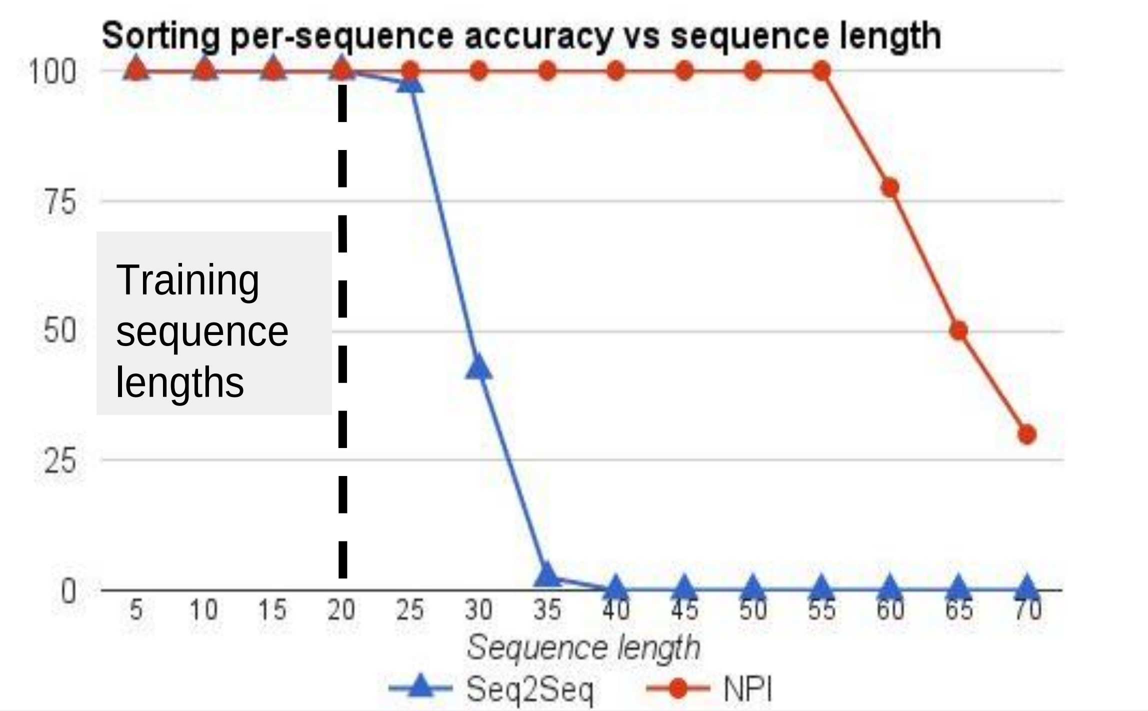
NPI benefits from mining multiple subprogram examples per sorting instance, and additional paramete rs of the program memory.



Generalization – Sorting

For each length, we provided 64 example bubble sort traces, for a total of 1,216 examples.

Then, we evaluated whether the network can learn to sort arrays beyond length 20



Generalization – Addition

Example problem: 90 + 160 = 250, we could represent the sequence as:

90X160X250

and solve addition via sequence prediction, e.g. “Learning to Execute” paper.

Generalization – Addition problems

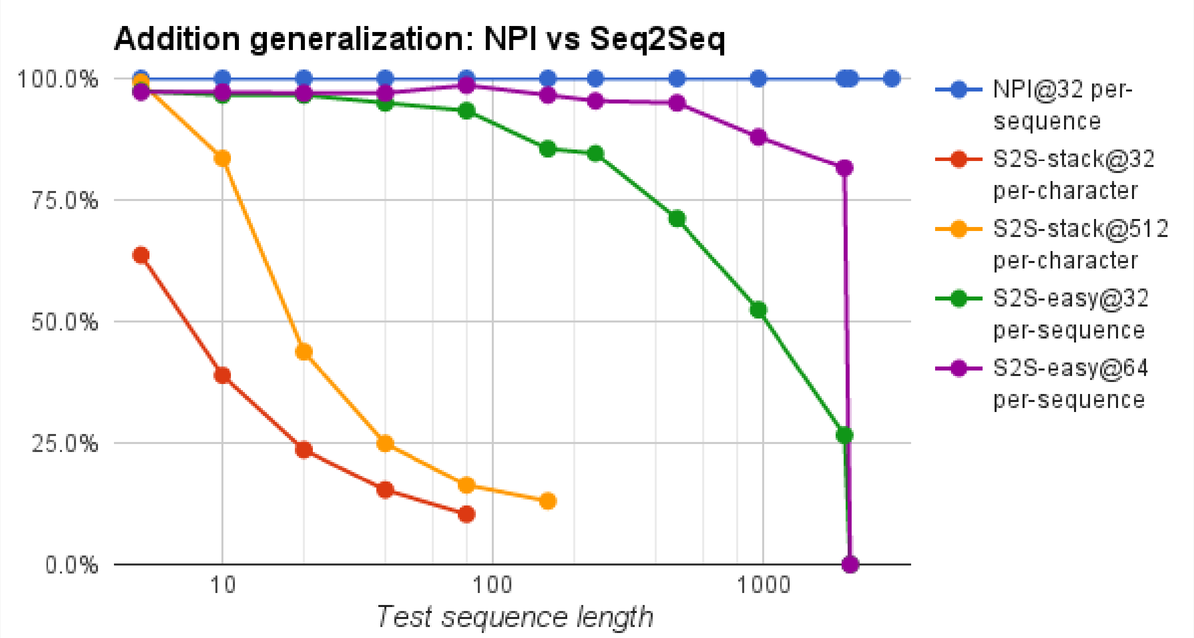
To make it easier, we can reverse and stack the inputs. (s2s-stack)

output: XXXX250
input 1: 090XXXX
input 2: 061XXXX

Even easier version: computation is entirely local. (s2s-easy)

output: 052
input 1: 090
input 2: 061

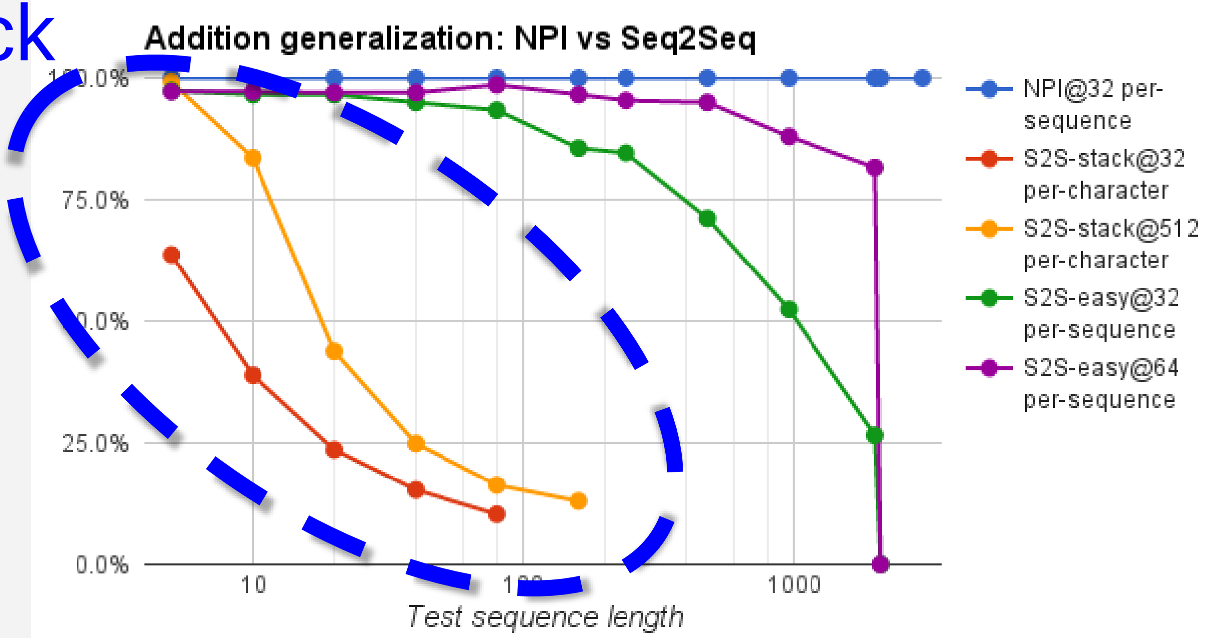
Generalization – Addition problems



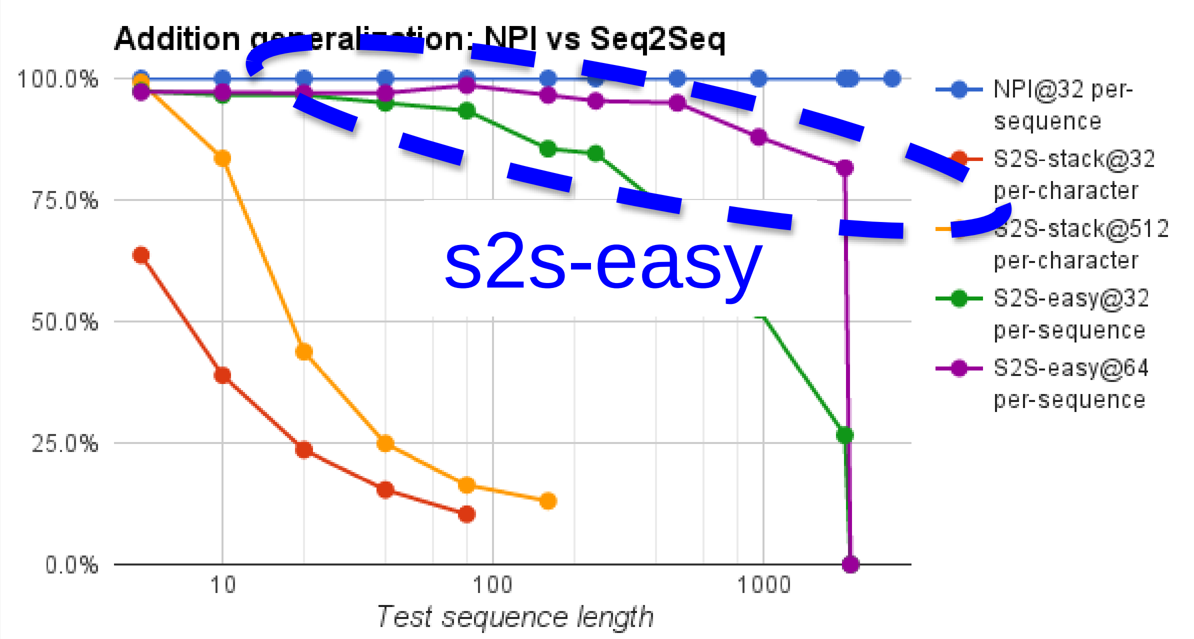
Trained on problem sizes 1,..,20 digits.

Generalization – Addition problems

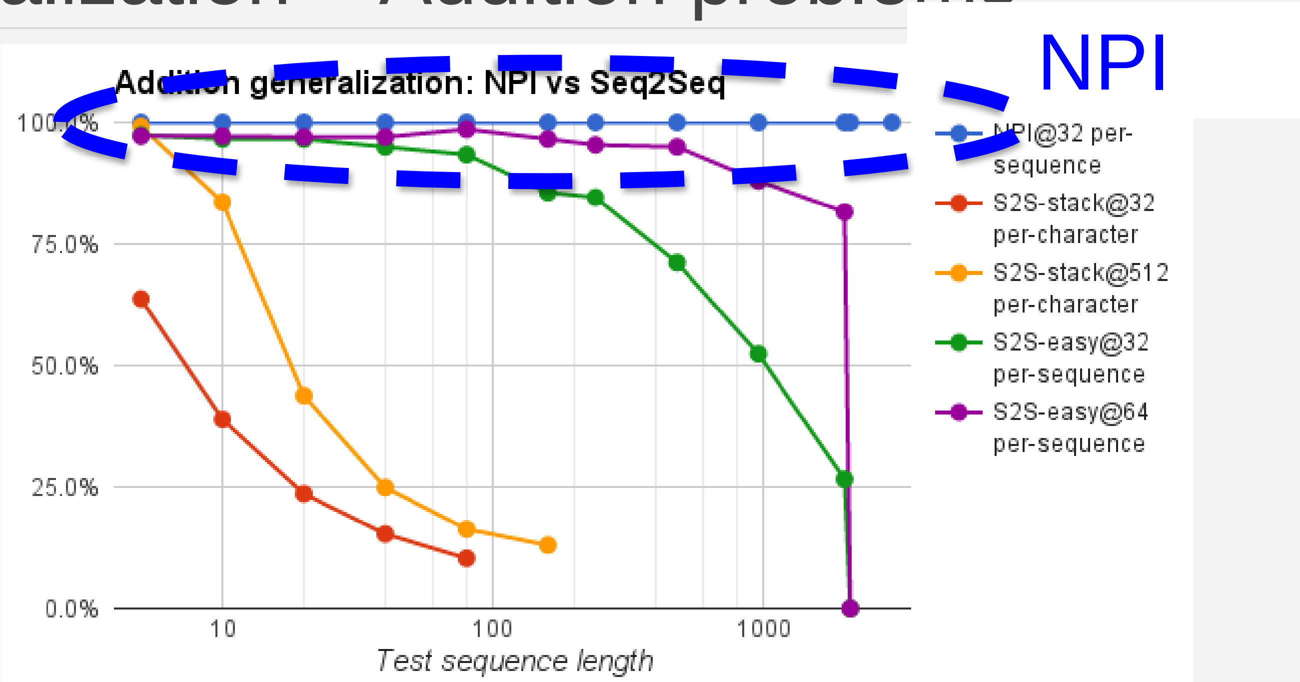
s2s-stack



Generalization – Addition problems



Generalization – Addition problems



Multi-task NPI – Core is shared across all programs

Program	Descriptions	Calls
ADD	Perform multi-digit addition	ADD1, LSHIFT
ADD1	Perform single-digit addition	ACT, CARRY
CARRY	Mark a 1 in the carry row one unit left	ACT
LSHIFT	Shift a specified pointer one step left	ACT
RSHIFT	Shift a specified pointer one step right	ACT
ACT	Move a pointer or write to the scratch pad	-
BUBBLESORT	Perform bubble sort (ascending order)	BUBBLE, RESET
BUBBLE	Perform one sweep of pointers left to right	ACT, BSTEP
RESET	Move both pointers all the way left	LSHIFT
BSTEP	Conditionally swap and advance pointers	COMPSWAP, RSHIFT
COMPSWAP	Conditionally swap two elements	ACT
LSHIFT	Shift a specified pointer one step left	ACT
RSHIFT	Shift a specified pointer one step right	ACT
ACT	Swap two values at pointer locations or move a pointer	-
GOTO	Change 3D car pose to match the target	HGOTO, VGOTO
HGOTO	Move horizontally to the target angle	LGOTO, RGOTO
LGOTO	Move left to match the target angle	ACT
RGOTO	Move right to match the target angle	ACT
VGOTO	Move vertically to the target elevation	UGOTO, DGOTO
UGOTO	Move up to match the target elevation	ACT
DGOTO	Move down to match the target elevation	ACT
ACT	Move camera 15° up, down, left or right	-
RJMP	Move all pointers to the rightmost posiiton	RSHIFT
MAX	Find maximum element of an array	BUBBLESORT,RJMP

Learning new programs with a fixed NPI core

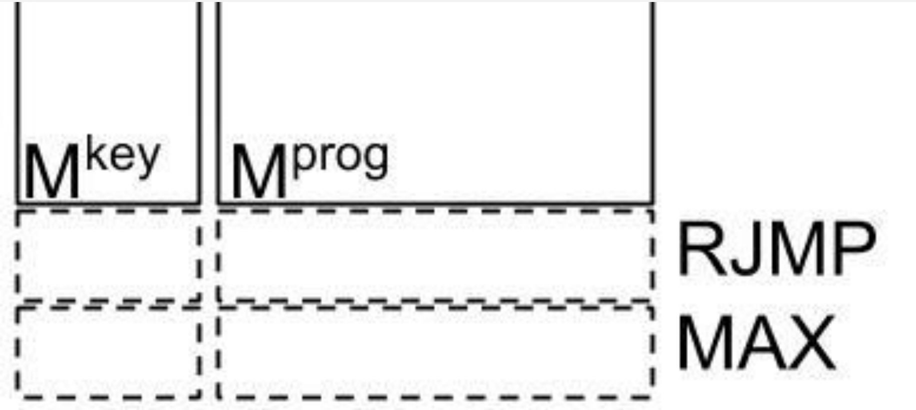
Toy example: Maximum-finding in an array.

Simple (not optimal) way: Call BUBBLESORT and then take the right-most element of the array. Two new programs:

RJMP: Move all pointers to the rightmost position in the array by repeatedly calling RSHIFT program.
MAX: Call BUBBLESORT and then RJMP

Expand program memory by adding 2 slots. Randomly initialize, then learn by backpropagation with the NPI core and all other parameters fixed.

Learning new programs with a fixed NPI core



Protocol:

1. Randomly initialize new program vectors in memory
2. Freeze core and other program vectors
3. Backpropagate gradients to new program vectors

Quantitative Results

Task	Single	Multi	+ Max
Addition	100.0	97.0	97.0
Sorting	100.0	100.0	100.0
Canon. seen car	89.5	91.4	91.4
Canon. unseen	88.7	89.9	89.9
Maximum	-	-	100.0

- Per-sequence % accuracy.
- + Max indicates performance after addition of MAX program to memory.
- “unseen” uses a test set with disjoint car models from the training set.

Conclusions & Next Steps

- A single NPI can learn multiple programs in dissimilar environments with different affordances.
- NPI sorting and addition programs exhibit strong generalization compared to baseline Seq2Seq models.
- A trained NPI with a fixed core can continue to learn new programs without forgetting already learned programs.

- **Next steps**: reduce supervision, scale up #programs, integrate new perception modules and affordances.

Related work

Too much to cover in 20 minutes!

- Sigma-Pi Units (Rumelhart, 1986): activations of one network become th e weights of a second network. Slowly changing network learns to contro l rapidly-changing network (Schmidhuber, 1992).
- Hierarchical RL (Sutton 1999, Dietterich 2000, Andre and Sutton 2001).
- Recent extensions of Seq2Seq: NTM, Pointer Networks, Memory Networks, Stack/Queue/Dequeue-augmented recurrent networks.
- Several other ICLR'16 papers on neural program induction. Main differen ce is that NPI explicitly incorporates compositional program structure.
- Recent models of prefrontal cognitive control (Donnarumma 2015).
- Learning to Execute (Zaremba 2014)

Thanks!

NPI single time step computation

Traces: $\xi_t^{inp} : \{e_t, i_t, a_t\}$ $\xi_t^{out} : \{i_{t+1}, a_{t+1}, r_t\}$

LSTM core input	$s_t = f_{enc}(e_t, a_t)$
LSTM output	$h_t = f_{lstm}(s_t, p_t, h_{t-1})$
pred. return prob	$r_t = f_{end}(h_t)$
next program key	$k_t = f_{prog}(h_t)$
next program args	$a_{t+1} = f_{arg}(h_t)$

NPI single time step computation

Traces: $\xi_t^{inp} : \{e_t, i_t, a_t\}$ $\xi_t^{out} : \{i_{t+1}, a_{t+1}, r_t\}$

$$i^* = \arg\max_{i=1..N} (M_{i,:}^{\texttt{key}})^T k_t \quad , \quad p_{t+1} = M_{i^*,:}^{\texttt{prog}}$$

selected prog inde x

program key memory

program memory

NPI single time step computation

Traces: $\xi_t^{inp} : \{e_t, i_t, a_t\}$ $\xi_t^{out} : \{i_{t+1}, a_{t+1}, r_t\}$

$$e_{t+1} \sim f_{env}(e_t, p_t, a_t)$$

Next environment observation; depends on selected program and arguments.
(Not controlled by NPI parameters)

NPI learning

Traces: $\xi_t^{inp} : \{e_t, i_t, a_t\}$ $\xi_t^{out} : \{i_{t+1}, a_{t+1}, r_t\}$

Objective:

$$\theta^* = \arg\max_\theta \sum_{(\xi^{inp},\xi^{out})} \log P(\xi^{out}|\xi^{inp};\theta)$$

$$\log P(\xi_{out}|\xi_{inp};\theta) = \sum_{t=1}^{T} \log P(\xi_t^{out}|\xi_1^{inp}, ..., \xi_t^{inp};\theta)$$