

# **Program Induction: Bridging AI and program synthesis, by learning to write code that writes code**

---

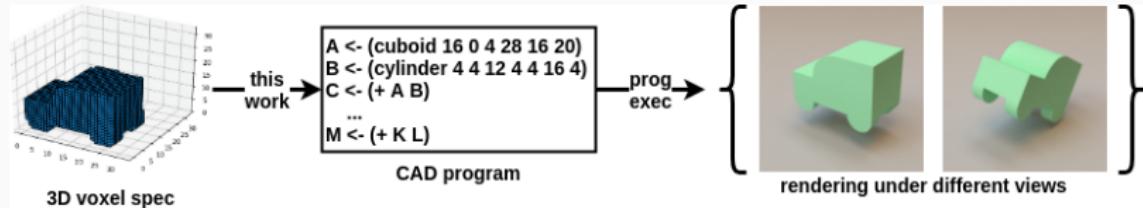
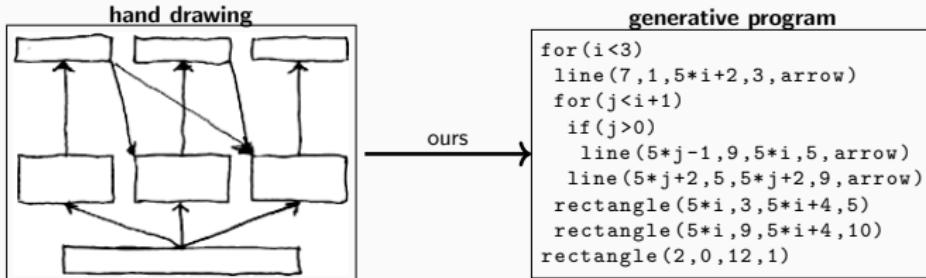
Kevin Ellis

2020

MIT

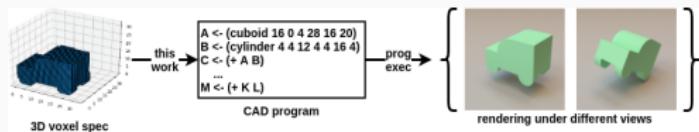
# Perception, Learning-to-Learn, Synthesizing models

Theme #1: high-level scene understanding, pixels→programs



# Perception, Learning-to-Learn, Synthesizing models

Theme #1: high-level scene understanding, pixels $\rightarrow$ programs



Theme #2: Learning to synthesize programs

## List Processing

### Sum List

$$[1 \ 2 \ 3] \rightarrow 6$$

$$[4 \ 6 \ 8 \ 1] \rightarrow 17$$

### Double

$$[1 \ 2 \ 3 \ 4] \rightarrow [2 \ 4 \ 6 \ 8]$$

$$[6 \ 5 \ 1] \rightarrow [12 \ 10 \ 2]$$

### Check Evens

$$[0 \ 2 \ 3] \rightarrow [T \ T \ F]$$

$$[2 \ 4 \ 9 \ 6] \rightarrow [T \ T \ F \ T]$$

## Text Editing

### Abbreviate

$$\text{Allen Newell} \rightarrow \text{A.N.}$$

$$\text{Herb Simon} \rightarrow \text{H.S.}$$

### Drop Last Characters

$$\text{jabberwocky} \rightarrow \text{jabberw}$$

$$\text{copycat} \rightarrow \text{cop}$$

## Regexes

### Phone Numbers

$$(555) \ 867-5309$$

$$(650) \ 555-2368$$

## LOGO Graphics

### Currency

$$\$100.25$$

$$\$4.50$$

### Dates

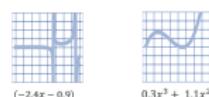
$$Y1775/0704$$

$$Y2000/0101$$

## Block Towers



## Symbolic Regression



## Recursive Programming

### Filter

$$[\textcolor{red}{\blacksquare} \textcolor{blue}{\blacksquare} \textcolor{red}{\blacksquare}] \rightarrow [\textcolor{blue}{\blacksquare} \textcolor{blue}{\blacksquare}]$$

$$[\textcolor{red}{\blacksquare} \textcolor{red}{\blacksquare} \textcolor{black}{\blacksquare}] \rightarrow [\textcolor{black}{\blacksquare} \textcolor{black}{\blacksquare}]$$

$$[\textcolor{red}{\blacksquare} \textcolor{black}{\blacksquare} \textcolor{red}{\blacksquare}] \rightarrow [\textcolor{black}{\blacksquare} \textcolor{black}{\blacksquare}]$$

### Index List

$$0, [\textcolor{red}{\blacksquare} \textcolor{blue}{\blacksquare} \textcolor{red}{\blacksquare}] \rightarrow \textcolor{red}{\blacksquare}$$

$$1, [\textcolor{red}{\blacksquare} \textcolor{blue}{\blacksquare} \textcolor{red}{\blacksquare}] \rightarrow \textcolor{blue}{\blacksquare}$$

$$1, [\textcolor{red}{\blacksquare} \textcolor{blue}{\blacksquare} \textcolor{red}{\blacksquare}] \rightarrow \textcolor{black}{\blacksquare}$$

## Physics

$$KE = \frac{1}{2} m |\vec{v}|^2$$

$$\vec{d} = \frac{1}{m} \sum_i \vec{F}_i$$

### Length

$$[\textcolor{red}{\blacksquare} \textcolor{blue}{\blacksquare} \textcolor{red}{\blacksquare}] \rightarrow 4$$

$$[\textcolor{red}{\blacksquare} \textcolor{black}{\blacksquare} \textcolor{red}{\blacksquare}] \rightarrow 6$$

$$[\textcolor{red}{\blacksquare} \textcolor{black}{\blacksquare}] \rightarrow 3$$

### Every Other

$$[\textcolor{red}{\blacksquare} \textcolor{blue}{\blacksquare} \textcolor{red}{\blacksquare}] \rightarrow [\textcolor{red}{\blacksquare}]$$

$$[\textcolor{red}{\blacksquare} \textcolor{black}{\blacksquare} \textcolor{red}{\blacksquare}] \rightarrow [\textcolor{black}{\blacksquare}]$$

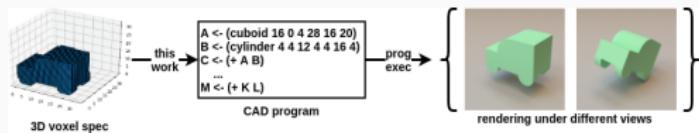
$$[\textcolor{red}{\blacksquare} \textcolor{black}{\blacksquare} \textcolor{red}{\blacksquare}] \rightarrow [\textcolor{black}{\blacksquare} \textcolor{black}{\blacksquare}]$$

$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}_1 - \vec{r}_2|^2} \hat{r}_1 \cdot \hat{r}_2$$

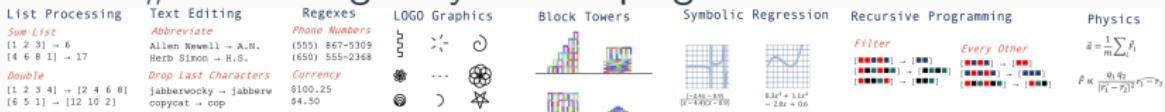
$$R_{total} = \left( \sum_i \frac{1}{R_i} \right)^{-1}$$

# Perception, Learning-to-Learn, Synthesizing models

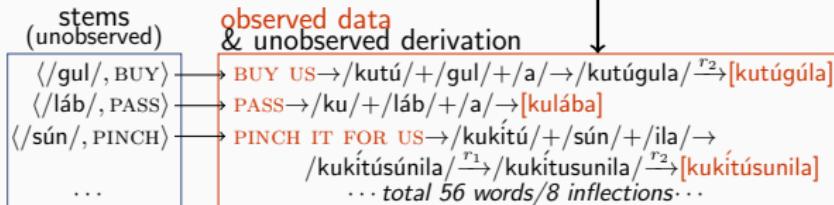
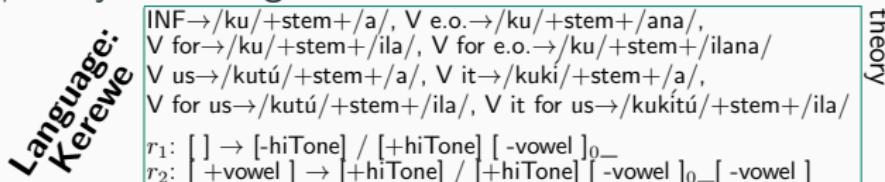
Theme #1: high-level scene understanding, pixels→programs



Theme #2: Learning to synthesize programs



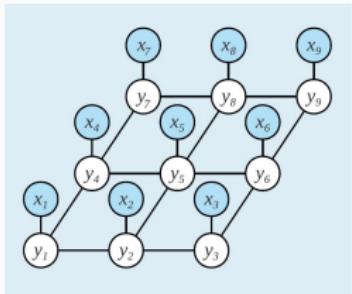
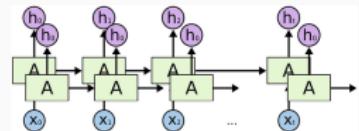
Theme #3: Synthesizing human-understandable models



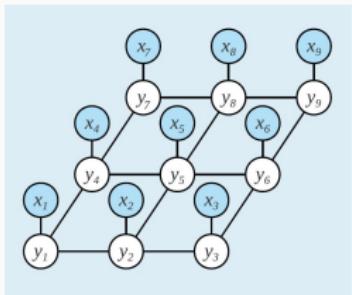
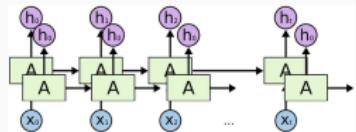
# High-level, abstract visual abilities



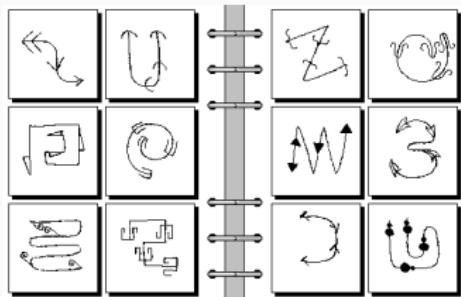
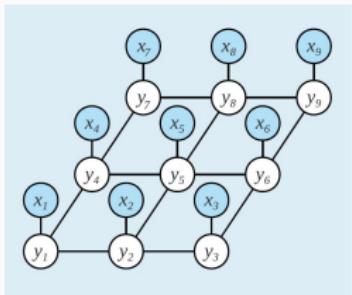
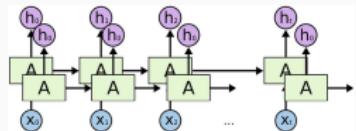
# High-level, abstract visual abilities



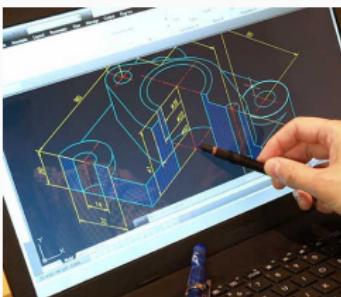
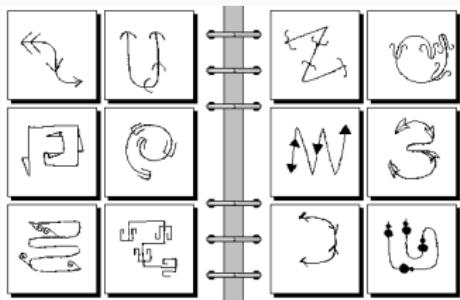
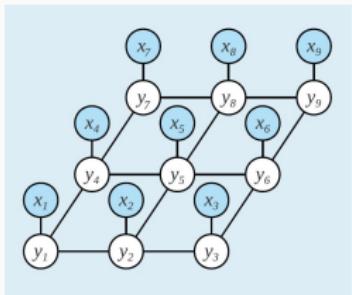
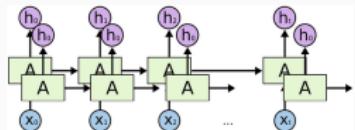
# High-level, abstract visual abilities



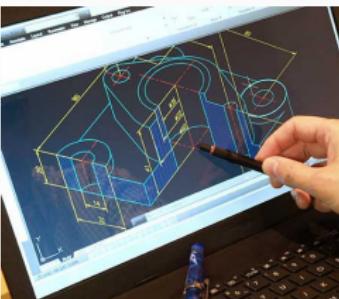
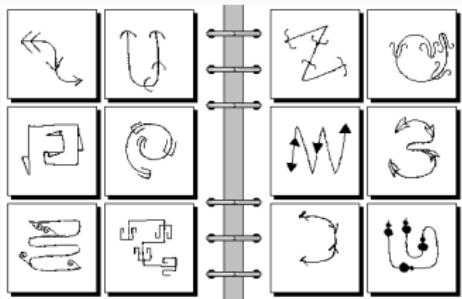
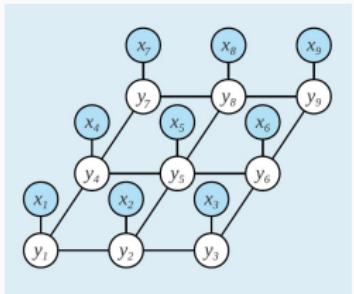
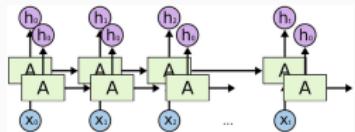
# High-level, abstract visual abilities



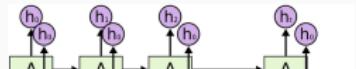
# High-level, abstract visual abilities



# High-level, abstract visual abilities



# High-level, abstract visual abilities

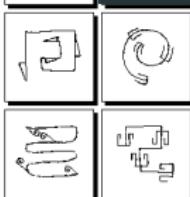


why?

impute missing objects, extrapolate percepts,  
learn visual concepts ('arch', 'spiral', HMM),  
assist graphic design, assist 3D modeling

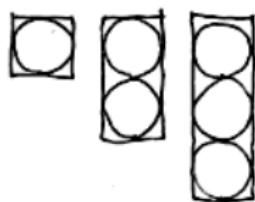
how?

Bayesian inference of graphics program conditioned on image,  
+program synthesis  
+learning



# Learning to infer graphics programs from hand-drawn images

model infers program from drawing



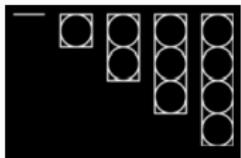
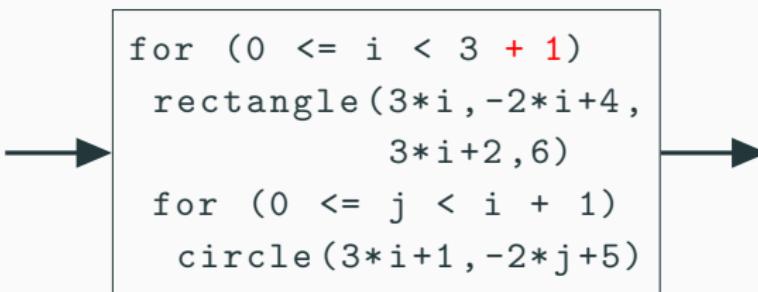
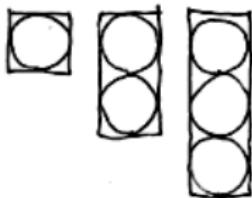
```
for (0 <= i < 3)
    rectangle(3*i, -2*i+4,
              3*i+2, 6)
    for (0 <= j < i + 1)
        circle(3*i+1, -2*j+5)
```

Ellis, Ritchie, Solar-Lezama, Tenenbaum. NeurIPS 2018.

# Learning to infer graphics programs from hand-drawn images

model infers program from drawing

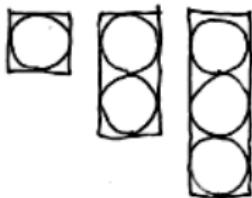
**zero-shot generalization / extrapolation**



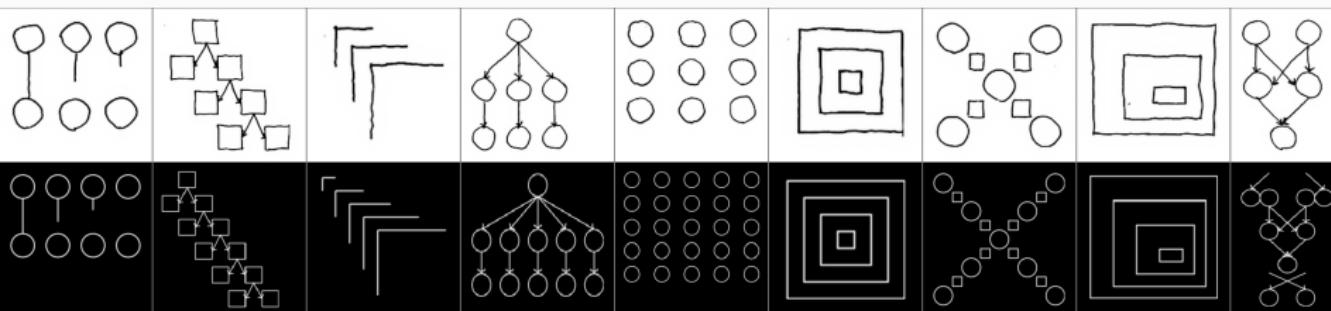
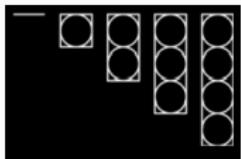
# Learning to infer graphics programs from hand-drawn images

model infers program from drawing

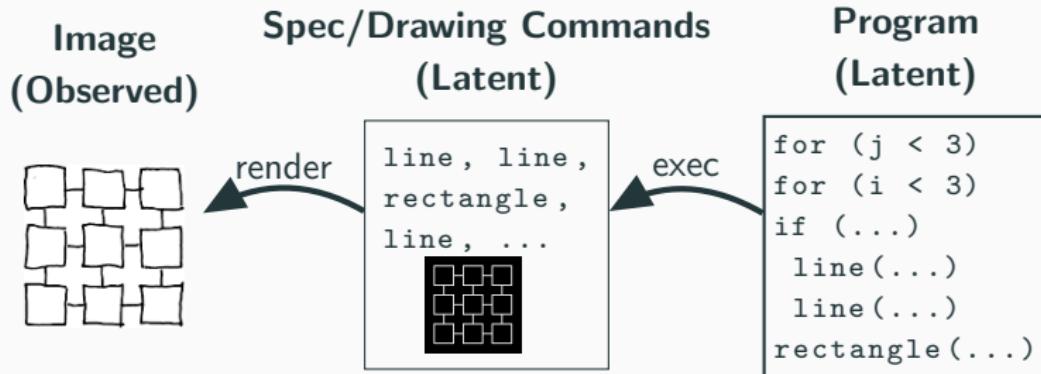
**zero-shot generalization / extrapolation**



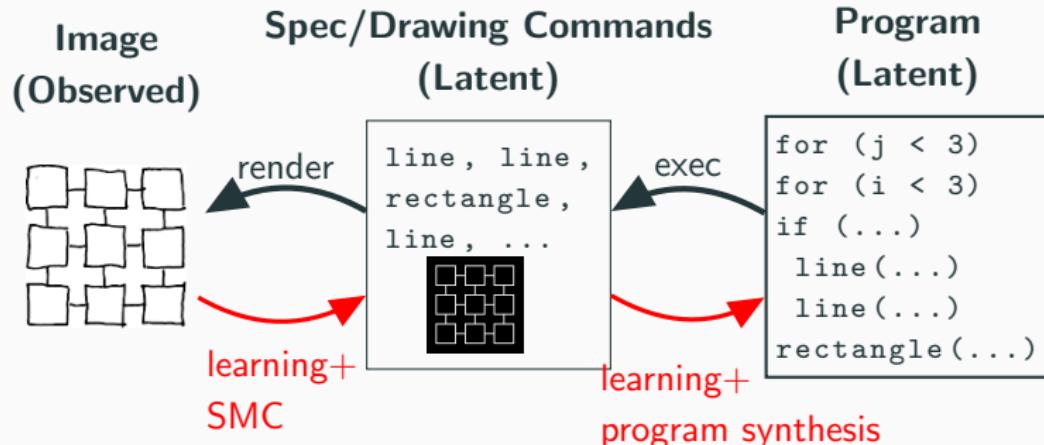
```
for (0 <= i < 3 + 1)
    rectangle(3*i, -2*i+4,
              3*i+2, 6)
    for (0 <= j < i + 1)
        circle(3*i+1, -2*j+5)
```



# How to infer graphics programs from hand-drawn images



# How to infer graphics programs from hand-drawn images

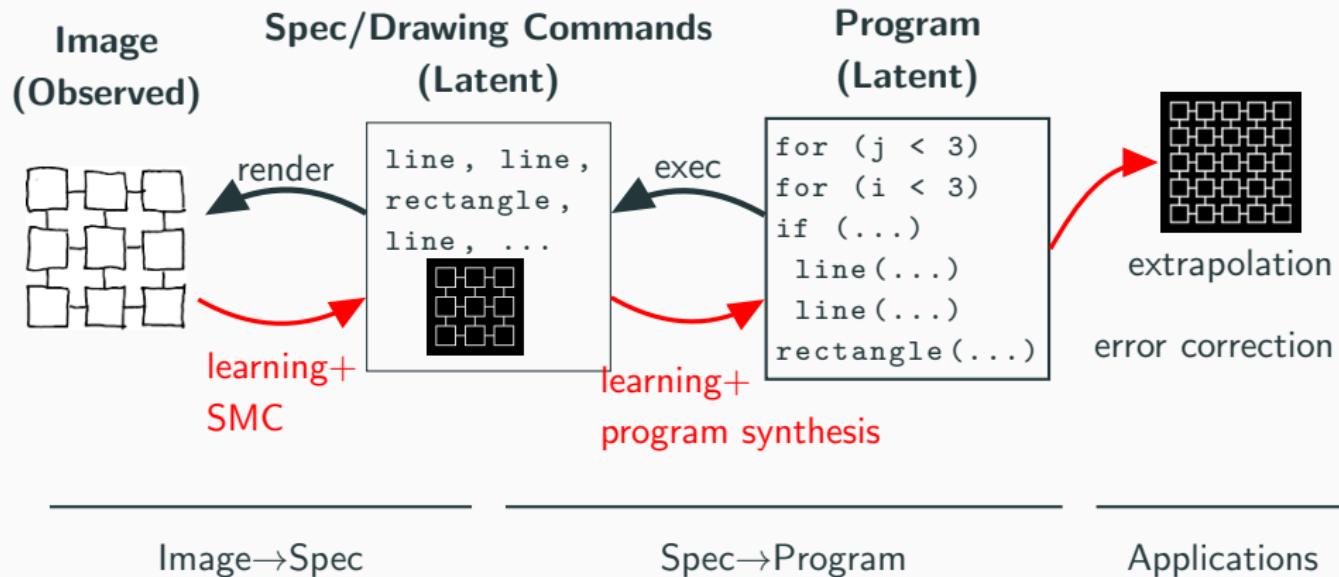


---

Image→Spec

Spec→Program

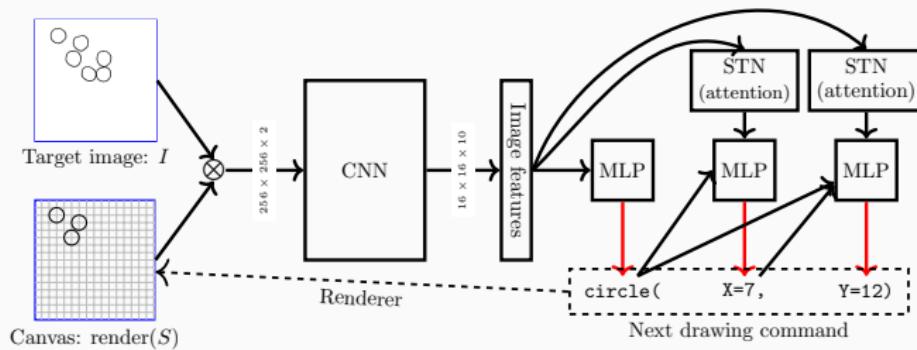
# How to infer graphics programs from hand-drawn images



# Parsing images into specs ( $\text{\LaTeX}$ TikZ commands)

Neurally Guided Procedural Modeling (Ritchie et al 2016) +

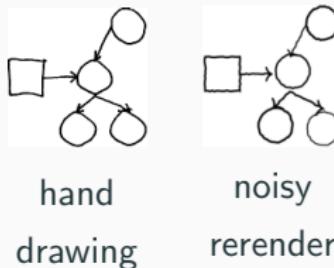
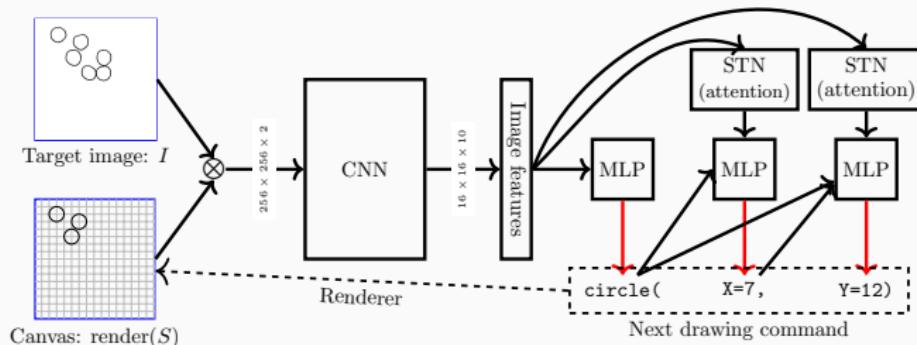
Attend, Infer, Repeat (Eslami et al 2016)



# Parsing images into specs ( $\text{\LaTeX}$ TikZ commands)

Neurally Guided Procedural Modeling (Ritchie et al 2016) +

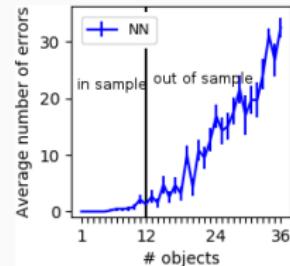
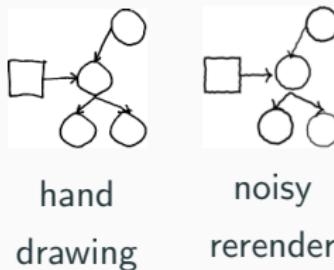
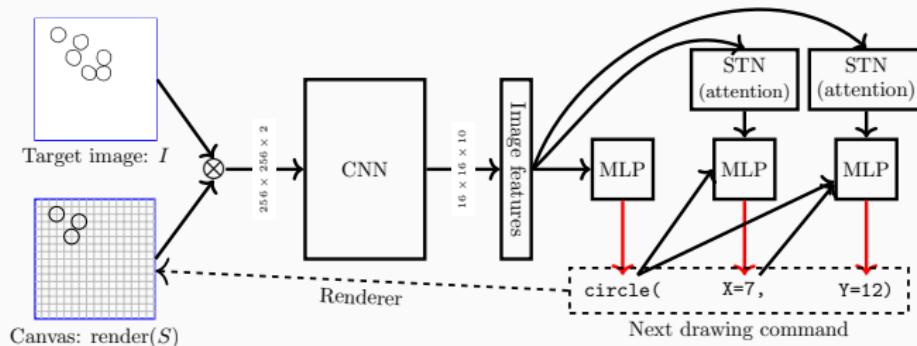
Attend, Infer, Repeat (Eslami et al 2016)



# Parsing images into specs ( $\text{\LaTeX}$ TikZ commands)

Neurally Guided Procedural Modeling (Ritchie et al 2016) +

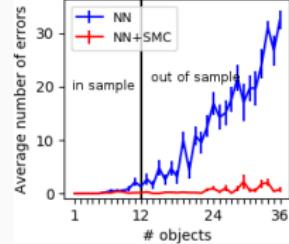
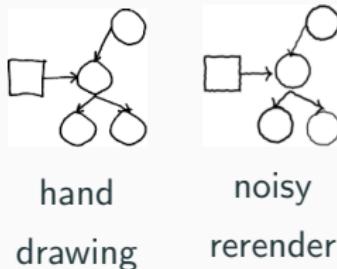
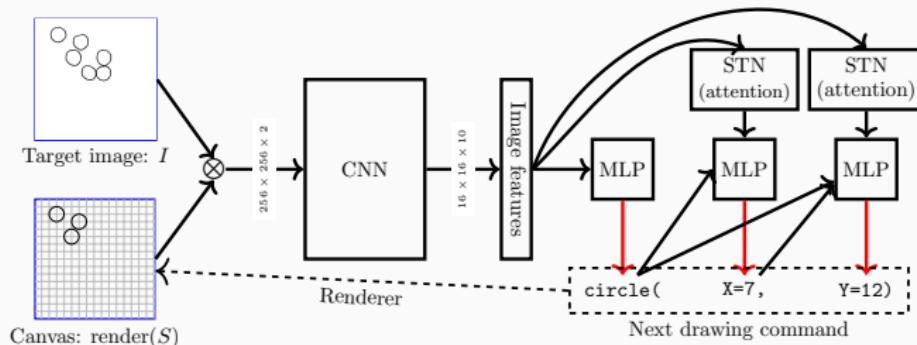
Attend, Infer, Repeat (Eslami et al 2016)



# Parsing images into specs ( $\text{\LaTeX}$ TikZ commands)

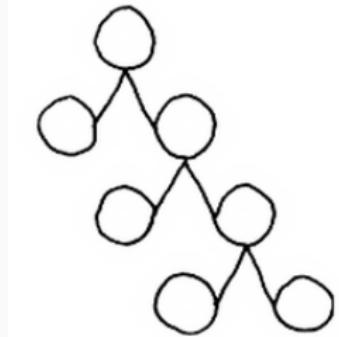
Neurally Guided Procedural Modeling (Ritchie et al 2016) +

Attend, Infer, Repeat (Eslami et al 2016)

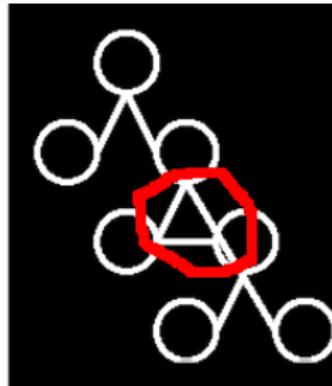


# Top-down influences on perception

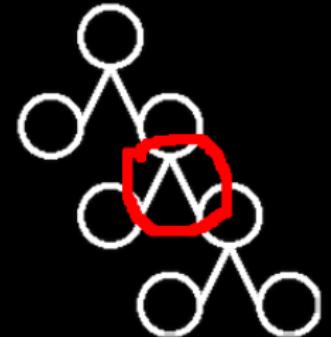
drawing



bottom-up neural net

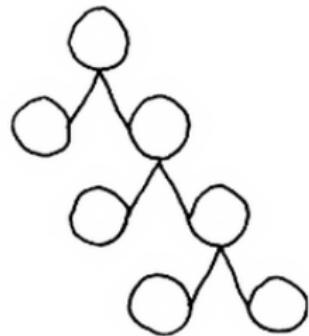


w/ top-down program bias

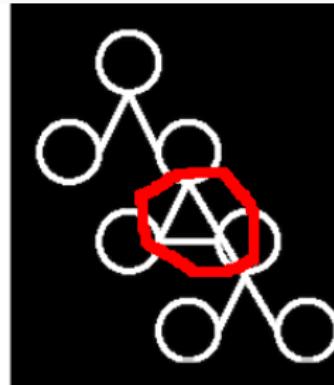


# Top-down influences on perception

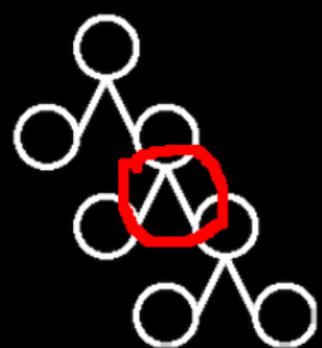
drawing



bottom-up neural net



w/ top-down program bias

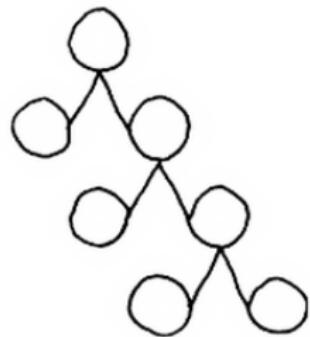


$$\hat{S}(I) = \arg \max_{S \in \mathcal{F}(I)} \mathbb{P}(I | \text{render}(S)) \times \mathbb{P}_\beta[\text{program}(S)]$$

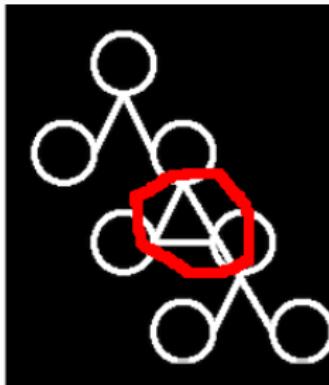
$$\beta^* = \arg \max_\beta \mathbb{E} \left[ \log \frac{\mathbb{P}_\beta[\text{program}(S)] \times \mathbb{P}(I | \text{render}(S))}{\sum_{S' \in \mathcal{F}(I)} \mathbb{P}_\beta[\text{program}(S')] \times \mathbb{P}(I | \text{render}(S'))} \right]$$

# Top-down influences on perception

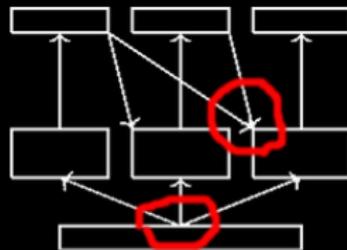
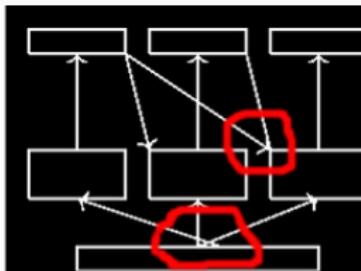
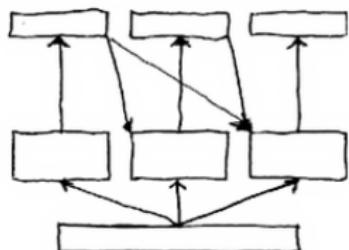
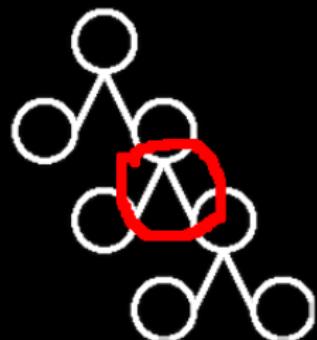
drawing



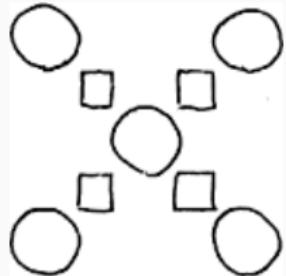
bottom-up neural net

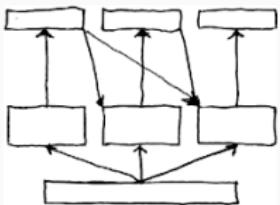


w/ top-down program bias

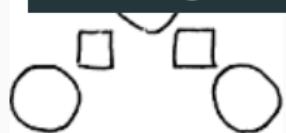
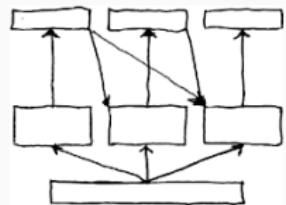


# Example programs

Drawing	Spec	Program
	<pre>Line(2,15, 4,15) Line(4,9, 4,13) Line(3,11, 3,14) Line(2,13, 2,15) Line(3,14, 6,14) Line(4,13, 8,13)</pre>	<pre>for(i&lt;3)     line(i,-1*i+6,           2*i+2,-1*i+6)     line(i,-2*i+4,i,-1*i+6)</pre>
	<pre>Circle(2,8) Rectangle(6,9, 7,10) Circle(8,8) Rectangle(6,12, 7,13) Rectangle(3,9, 4,10) ... etc. ....; 9 lines</pre>	<pre>reflect(y=8) for(i&lt;3)     if(i&gt;0)         rectangle(3*i-1,2,3*i,3)     circle(3*i+1,3*i+1)</pre>

	<pre>Line(3,10,3,14,arrow) Rectangle(11,8,15,10) Rectangle(11,14,15,15) Line(13,10,13,14,arrow) ... etc. ....; 16 lines</pre>	<pre>for(i&lt;3)     line(7,1,5*i+2,3,arrow)     for(j&lt;i+1)         if(j&gt;0)             line(5*j-1,9,5*i,5,arrow)             line(5*j+2,5,5*j+2,9,arrow)         rectangle(5*i,3,5*i+4,5)         rectangle(5*i,9,5*i+4,10)     rectangle(2,0,12,1)</pre>
--	---	--

# Example programs

Drawing	Spec	Program
	Line(2,15, 4,15) Line(4,9, 4,13) Line(3,11, 3,14) Line(2,13, 2,15) Line(3,14, 6,14) Line(4,13, 8,13)	for(i<3) line(i,-1*i+6, 2*i+2,-1*i+6) line(i,-2*i+4,i,-1*i+6)
Learning played a role... but much of this system is specific to 2-D graphics		
 Goal: a general algorithm for learning to synthesize programs		rectangle(8,12, 7,18) Rectangle(3,9, 4,10) ... etc. ....; 9 lines
 ... etc. ....; 16 lines		for(i<3) line(7,1,5*i+2,3,arrow) for(j<i+1) if(j>0) line(5*j-1,9,5*i,5,arrow) line(5*j+2,5,5*j+2,9,arrow) rectangle(5*i,3,5*i+4,5) rectangle(5*i,9,5*i+4,10) rectangle(2,0,12,1)

## Growing domain-specific knowledge

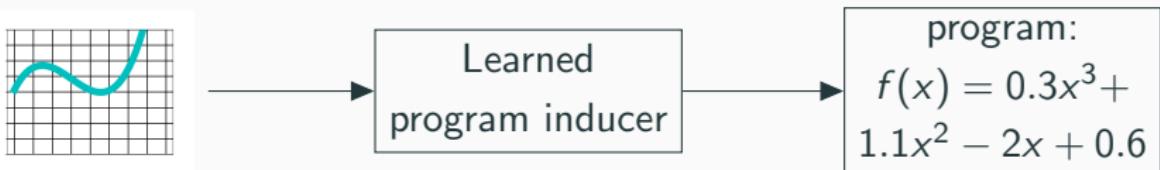
Goal: acquire domain-specific knowledge needed to induce a class of programs

- Library of concepts (declarative knowledge; domain specific language; generative model over programs)
- Inference strategy (procedural knowledge; synthesis algorithm)

# Growing domain-specific knowledge

Goal: acquire domain-specific knowledge needed to induce a class of programs

- Library of concepts (declarative knowledge; domain specific language; generative model over programs)
- Inference strategy (procedural knowledge; synthesis algorithm)



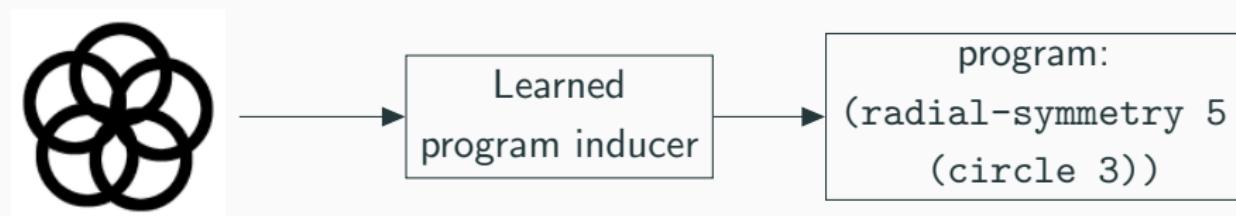
Concepts:  $x^3, \alpha x + \beta$ , etc

Inference strategy: neurosymbolic search for programs

# Growing domain-specific knowledge

Goal: acquire domain-specific knowledge needed to induce a class of programs

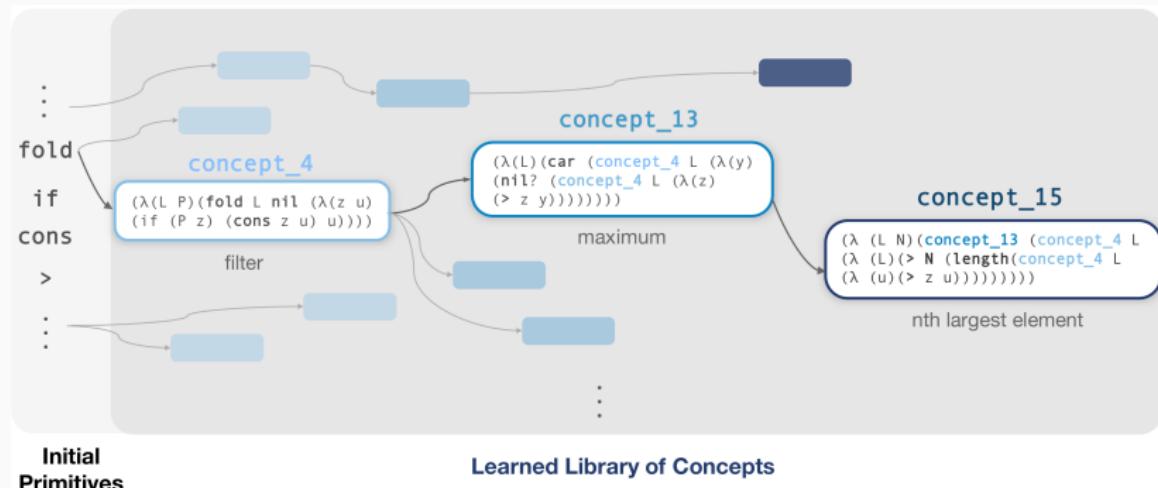
- Library of concepts (declarative knowledge; domain specific language; generative model over programs)
- Inference strategy (procedural knowledge; synthesis algorithm)



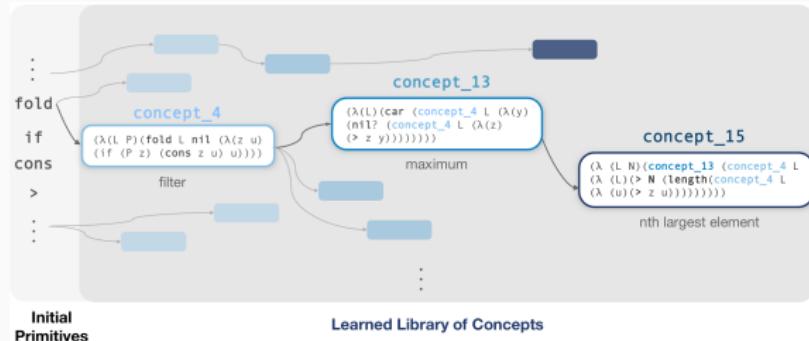
Concepts: circle, radial-symmetry, etc

Inference strategy: neurosymbolic search for programs

# Library learning



# Library learning



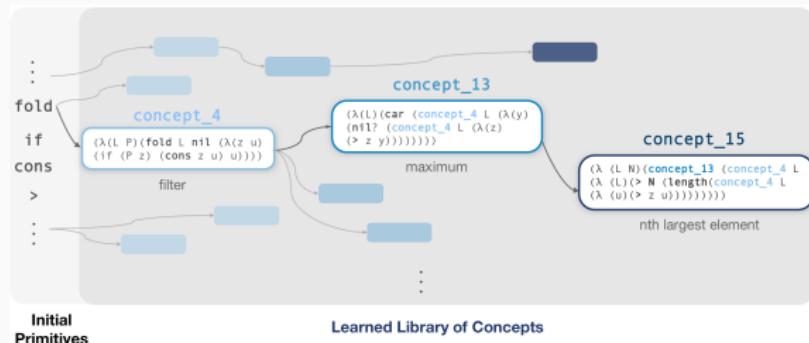
**Problem:** sort list

**Solution:**

```
(map (\n) (concept_15 L (+ 1 n))) (range (length L)))
```

get nth largest element      where n = 1, 2, 3 ....length of list

# Library learning



**Problem:** sort list

**Solution:**

$(map\ (\lambda(n)\ (concept_15\ L\ (+\ 1\ n)))\ (range\ (length\ L))))$   
get nth largest element      where n = 1, 2, 3 ... length of list

**Solution in initial primitives:**

```
(λ(x) (map (λ(y) (car (fold (fold x nil (λ(z u) (if (gt? (+ y 1) (length (fold x nil (λ(v w) (if (gt? z v) (cons v w)))))) (cons z u) u))) nil (λ(a b) (if (nil? (fold (fold x nil (λ(c d) (if (gt? (+ y 1) (length (fold x nil (λ(e f) (if (gt? c e) (cons e f) f)))))) (cons c d) d))) nil (λ(g h) (if (gt? g a) (cons g h) h)))) (cons a b) b)))) (range (length x))))
```

# DreamCoder

- **Wake:** Solve problems by writing programs
- **Sleep:** Improve DSL and neural recognition model:
  - **Abstraction sleep:** Improve library
  - **Dream sleep:** Improve neural inference model
- Combines ideas from Wake-Sleep & Exploration-Compression



# DreamCoder

- **Wake:** Solve problems by writing programs
- **Sleep:** Improve DSL and neural recognition model:
  - **Abstraction sleep:** Improve library
  - **Dream sleep:** Improve neural inference model
- Combines ideas from Wake-Sleep & Exploration-Compression

## List Processing

*Sum List*  
 $[1 \ 2 \ 3] \rightarrow 6$   
 $[4 \ 6 \ 8 \ 1] \rightarrow 17$

## Double

$[1 \ 2 \ 3 \ 4] \rightarrow [2 \ 4 \ 6 \ 8]$   
 $[6 \ 5 \ 1] \rightarrow [12 \ 10 \ 2]$

## Check Evens

$[0 \ 2 \ 3] \rightarrow [T \ T \ F]$   
 $[2 \ 4 \ 9 \ 6] \rightarrow [T \ T \ F \ T]$

## Text Editing

*Abbreviate*  
Allen Newell → A.N.  
Herb Simon → H.S.

*Drop Last Characters*  
jabberwocky → jabberw  
copycat → cop

## Extract

see spot(run) → run  
a (bee) see → bee

## Regexes

*Phone Numbers*  
(555) 867-5309  
(650) 555-2368

## Currency

\$100.25  
\$4.50

## Dates

Y1775/07/04  
Y2000/01/01

## LOGO Graphics



Physics

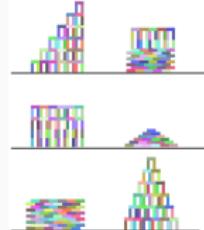
$$KE = \frac{1}{2}m|\vec{v}|^2$$

$$\bar{d} = \frac{1}{m} \sum_i \vec{F}_i$$

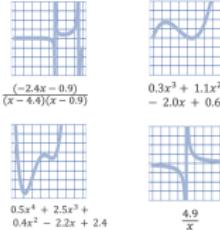
$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}_1 - \vec{r}_2|^2} \hat{r}_1 - \hat{r}_2$$

$$R_{total} = \left( \sum_i \frac{1}{R_i} \right)^{-1}$$

## Block Towers



## Symbolic Regression



## Recursive Programming

### Filter

$[■■■■■] \rightarrow [■■■]$   
 $[■■■■■■■■] \rightarrow [■■■■■■]$   
 $[■■■■■■■] \rightarrow [■■■■■]$

### Length

$[■■■■■] \rightarrow 4$   
 $[■■■■■■■■] \rightarrow 6$   
 $[■■■■■■■] \rightarrow 3$

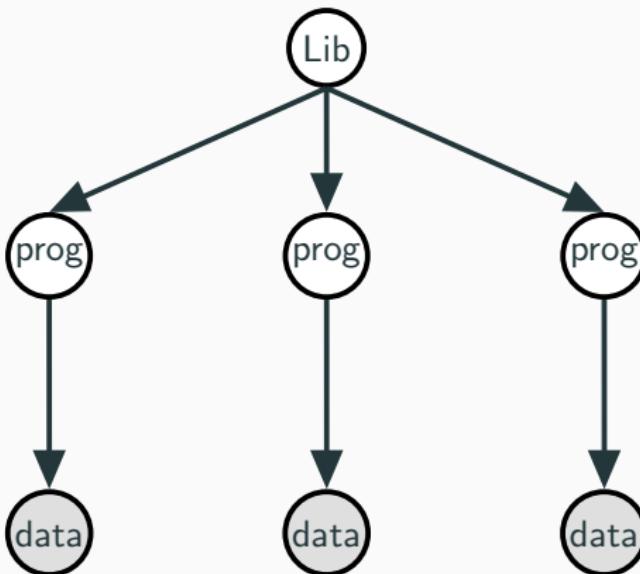
### Index List

$0, [■■■■■■■■] \rightarrow ■$   
 $1, [■■■■■■■■] \rightarrow ■■$   
 $1, [■■■■■■■■] \rightarrow ■■■$

### Every Other

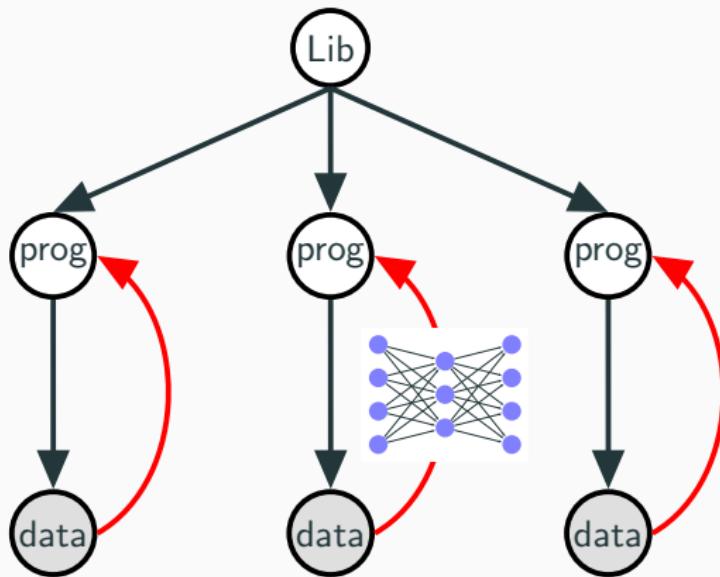
$[■■■■■■■■] \rightarrow [■■■■]$   
 $[■■■■■■■■] \rightarrow [■■■■■■]$   
 $[■■■■■■■■] \rightarrow [■■■■■■■]$

# Library learning as Bayesian inference

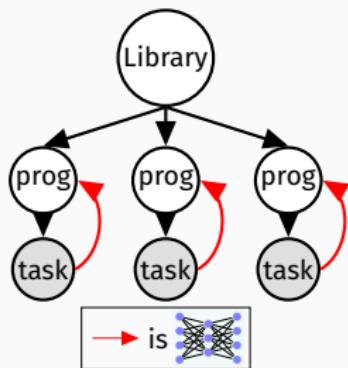


[Dechter et al., 2013] [Liang et al, 2010]; [Lake et al, 2015]

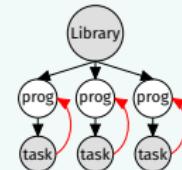
# Library learning as amortized Bayesian inference



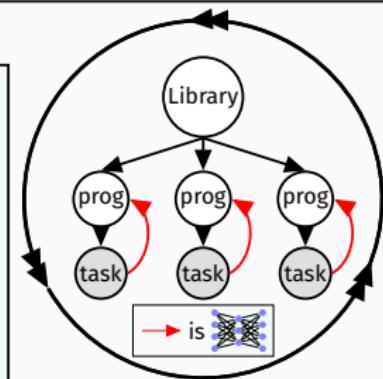
amortized inference +  
better program representation (Lisp) +  
library learning via program analysis +  
new neural inference network for program synthesis



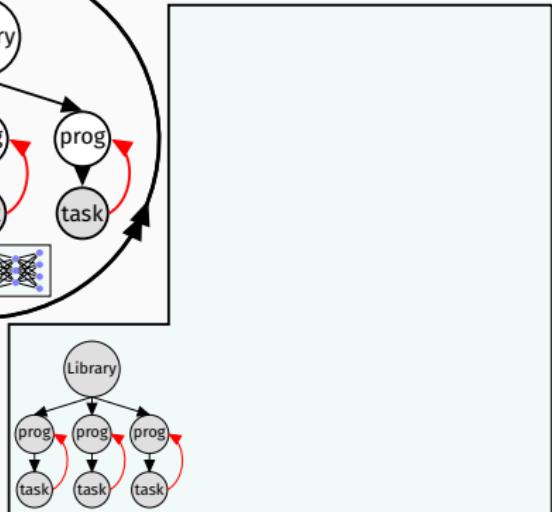
## WAKE

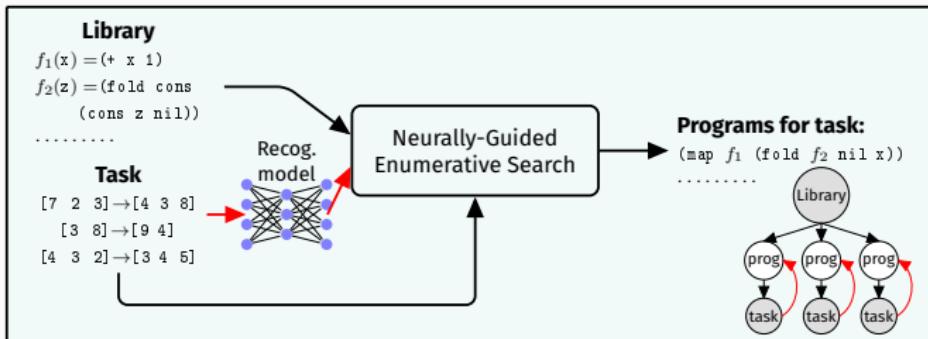
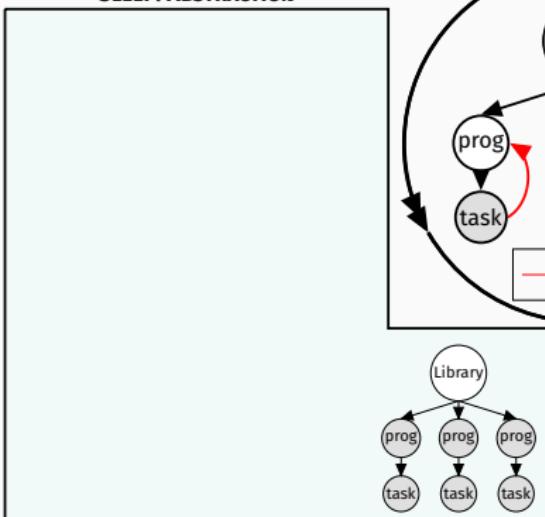
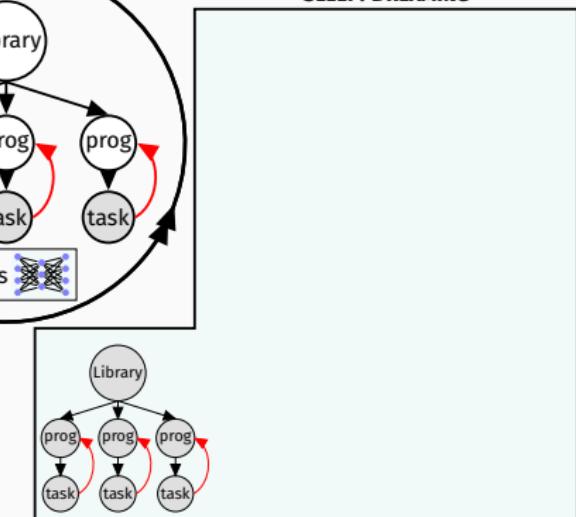


## SLEEP: ABSTRACTION

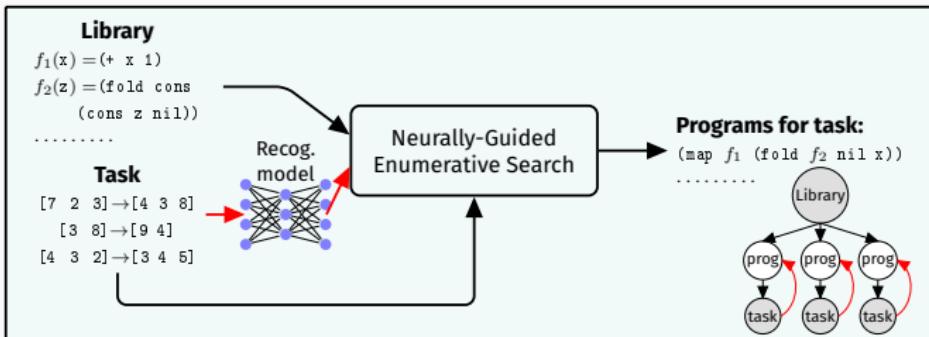


## SLEEP: DREAMING

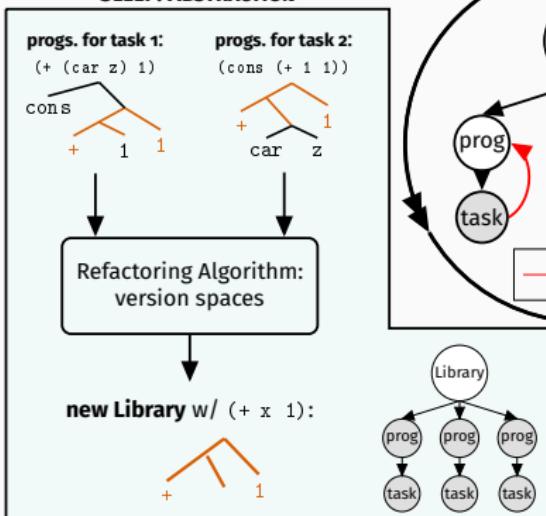


**WAKE****SLEEP: ABSTRACTION****SLEEP: DREAMING**

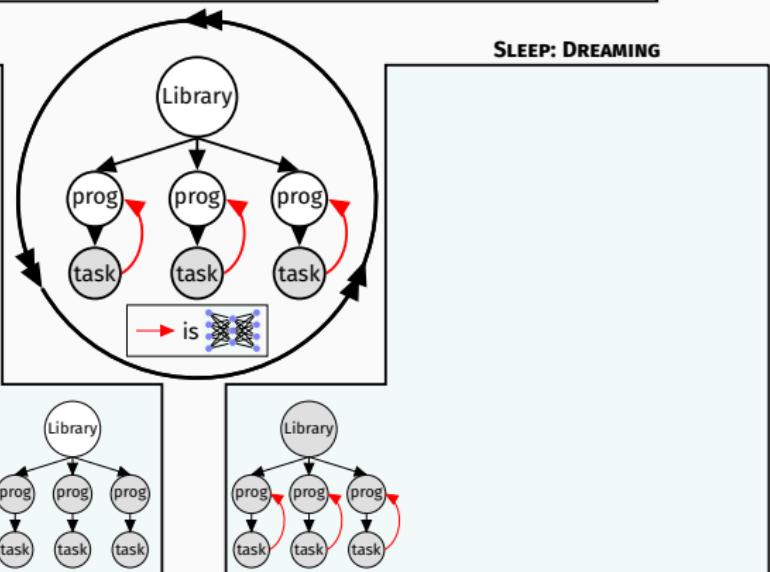
# WAKE



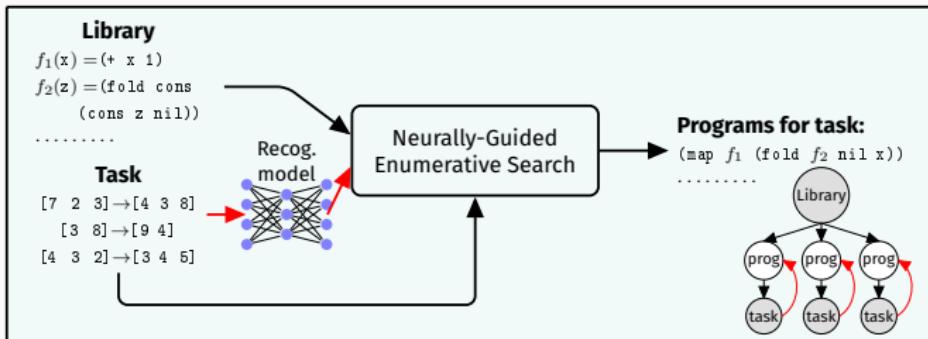
# SLEEP: ABSTRACTION



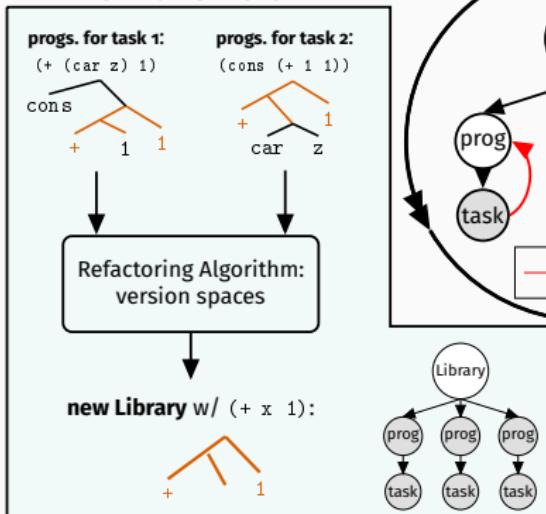
# SLEEP: DREAMING



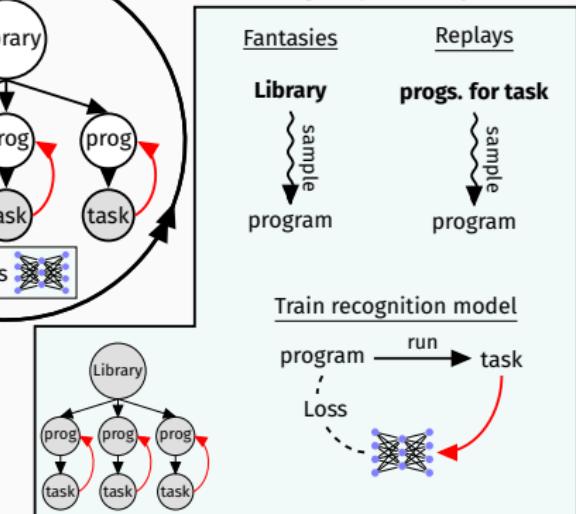
# WAKE



# SLEEP: ABSTRACTION



# SLEEP: DREAMING



# Abstraction Sleep: Growing the library via refactoring

**Task:**  $(1\ 2\ 3) \rightarrow (2\ 4\ 6)$   
 $(4\ 3\ 2) \rightarrow (8\ 6\ 8)$

Wake: program search

```
(Y (\lambda (r l) (if (nil? l) nil  
                      (cons (+ (car l) (car l))  
                            (r (cdr l)))))))
```

**Task:**  $(1\ 2\ 3) \rightarrow (0\ 1\ 2)$   
 $(4\ 3\ 2) \rightarrow (3\ 2\ 3)$

Wake: program search

```
(Y (\lambda (r l) (if (nil? l) nil  
                      (cons (- (car l) 1)  
                            (r (cdr l)))))))
```

# Abstraction Sleep: Growing the library via refactoring

Task:  $(1\ 2\ 3) \rightarrow (2\ 4\ 6)$   
 $(4\ 3\ 2) \rightarrow (8\ 6\ 8)$

Wake: program search

```
(Y (λ (r 1) (if (nil? 1) nil  
           (cons (+ (car 1) (car 1))  
                  (r (cdr 1)))))))
```

Task:  $(1\ 2\ 3) \rightarrow (0\ 1\ 2)$   
 $(4\ 3\ 2) \rightarrow (3\ 2\ 3)$

Wake: program search

```
(Y (λ (r 1) (if (nil? 1) nil  
           (cons (- (car 1) 1)  
                  (r (cdr 1)))))))
```

refactor  
 $(10^{14}$  refactorings)

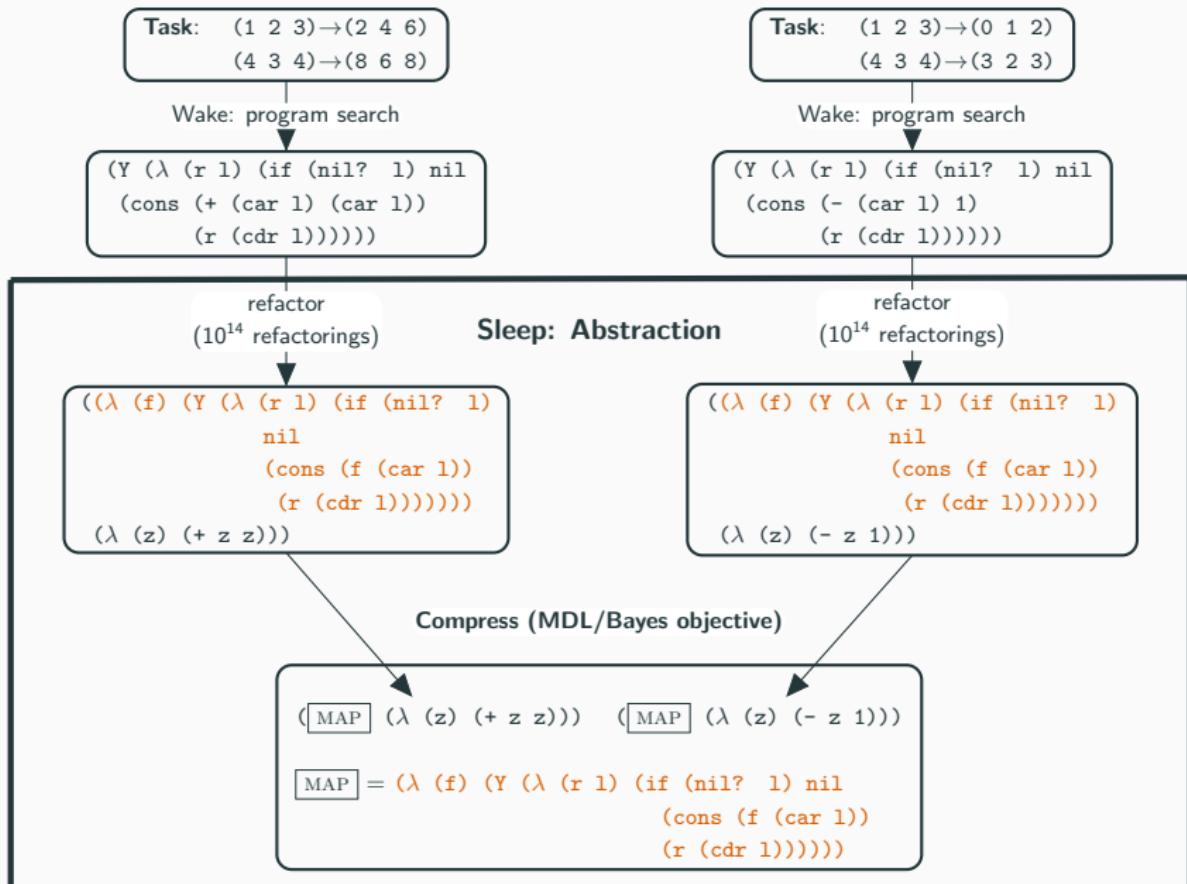
## Sleep: Abstraction

refactor  
 $(10^{14}$  refactorings)

```
((λ (f) (Y (λ (r 1) (if (nil? 1)  
                           nil  
                           (cons (f (car 1))  
                                 (r (cdr 1)))))))  
  (λ (z) (+ z z)))
```

```
((λ (f) (Y (λ (r 1) (if (nil? 1)  
                           nil  
                           (cons (f (car 1))  
                                 (r (cdr 1)))))))  
  (λ (z) (- z 1)))
```

# Abstraction Sleep: Growing the library via refactoring



# Version space algebra for refactoring

$\text{Expr} \rightarrow \text{var}$

- |  $\lambda\text{var}.\text{Expr}$
- |  $(\text{Expr } \text{Expr})$
- | primitive

## Version space algebra for refactoring

$\text{Expr} \rightarrow \text{var}$

|  $\lambda\text{var}.\text{Expr}$   
| ( $\text{Expr } \text{Expr}$ )  
| primitive

$\text{VS} \rightarrow \text{var}$

|  $\lambda\text{var}.\text{VS}$   
| ( $\text{VS } \text{VS}$ )  
| primitive  
|  $\text{VS} \uplus \text{VS}$       *nondeterministic choice*  
|  $\Lambda$                   *choose any expression*  
|  $\emptyset$                   *choose no expression*

# Version space algebra for refactoring

$\text{Expr} \rightarrow \text{var}$

|  $\lambda \text{var}. \text{Expr}$   
| ( $\text{Expr} \ \text{Expr}$ )  
| primitive

$\text{VS} \rightarrow \text{var}$

|  $\lambda \text{var}. \text{VS}$   
| ( $\text{VS} \ \text{VS}$ )  
| primitive  
|  $\text{VS} \uplus \text{VS}$       *nondeterministic choice*  
|  $\Lambda$                   *choose any expression*  
|  $\emptyset$                   *choose no expression*

what version spaces mean:

$$[\![\text{var}]\!] = \{\text{var}\}$$

$$[\![v_1 \uplus v_2]\!] = \{e : v \in \{v_1, v_2\}, e \in [\![v]\!]\}$$

$$[\![\lambda x. v]\!] = \{\lambda x. e : e \in [\![v]\!]\}$$

$$[\![(v_1 v_2)]\!] = \{(e_1 e_2) : e_1 \in [\![v_1]\!], e_2 \in [\![v_2]\!]\}$$

$$[\![\emptyset]\!] = \emptyset$$

$$[\![\Lambda]\!] = \Lambda$$

## Using version spaces

$VS \rightarrow var \mid \lambda var.VS \mid (VS\ VS) \mid primitive$

|  $VS \cup VS$       *nondeterministic choice*

|  $\Lambda$             *choose any expression*

|  $\emptyset$             *choose no expression*

## Using version spaces

$$\begin{aligned} \text{VS} \rightarrow & \text{ var } | \lambda \text{var}. \text{VS} | (\text{VS } \text{VS}) | \text{ primitive} \\ & | \text{VS} \sqcup \text{VS} \quad \textit{nondeterministic choice} \\ & | \Lambda \quad \textit{choose any expression} \\ & | \emptyset \quad \textit{choose no expression} \end{aligned}$$

exploit the fact that  $e \in [\![v]\!]$  can be efficiently computed:

$$\text{REFACTOR}(v|\text{Lib}) = \begin{cases} e, & \text{if } e \in \text{Lib and } e \in [\![v]\!]} \\ \text{REFACTOR}'(v|\text{Lib}), & \text{otherwise.} \end{cases}$$

$$\text{REFACTOR}'(v|\text{Lib}) = v, \text{ if } v \text{ is a primitive or variable}$$

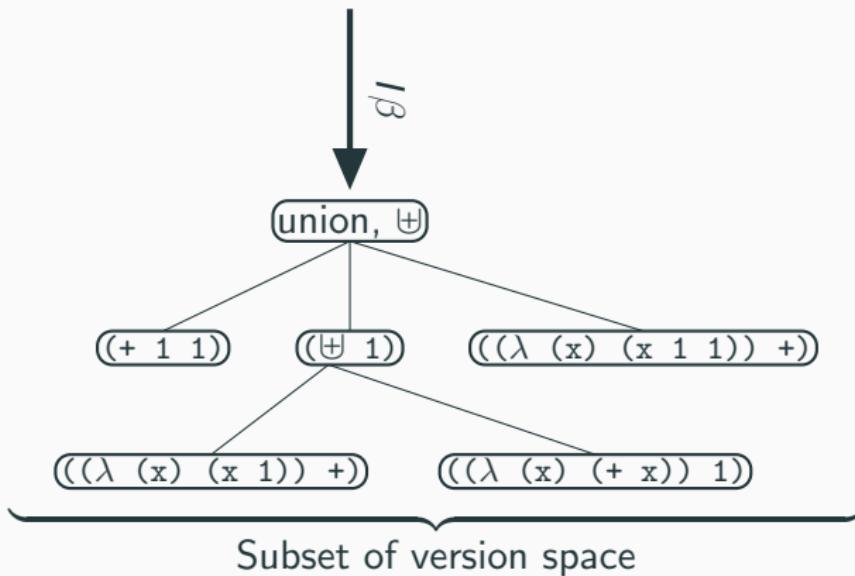
$$\text{REFACTOR}'(\lambda x. b|\text{Lib}) = \lambda x. \text{REFACTOR}(b|\text{Lib})$$

$$\text{REFACTOR}'(v_1 v_2|\text{Lib}) = (\text{REFACTOR}(v_1|\text{Lib}) \text{ REFACTOR}(v_2|\text{Lib}))$$

## Using version spaces

$VS \rightarrow var \mid \lambda var.VS \mid (VS \vee VS) \mid primitive$   
|  $VS \oplus VS$       *nondeterministic choice*  
|  $\Lambda$                 *choose any expression*  
|  $\emptyset$                 *choose no expression*

Program:  $(+ 1 1)$



## Using version spaces

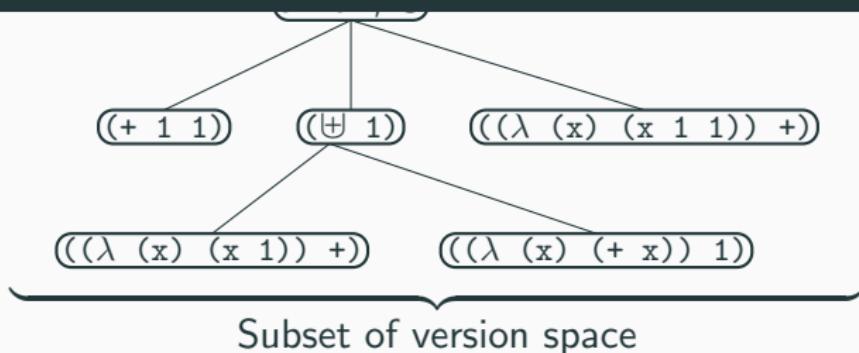
$VS \rightarrow var \mid \lambda var.VS \mid (VS \; VS) \mid primitive$   
|  $VS \sqcup VS$       *nondeterministic choice*  
|  $\Lambda$                 *choose any expression*  
|  $\emptyset$                *choose no expression*

**completeness:**  $I\beta$  gets all the refactorings

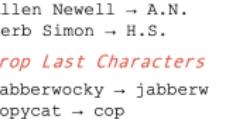
let  $v_2 = I\beta(v_1)$  and  $e_1 \in \llbracket v_1 \rrbracket$ . for any  $e_2 \rightarrow e_1$  then  $e_2 \in \llbracket v_2 \rrbracket$

**consistency:**  $I\beta$  only gets valid refactorings

let  $v_2 = I\beta(v_1)$  and  $e_2 \in \llbracket v_2 \rrbracket$ . then there is a  $e_1 \in \llbracket v_1 \rrbracket$  where  $e_2 \rightarrow e_1$



# DreamCoder Domains

List Processing	Text Editing	Regexes	LOGO Graphics
<b>Sum List</b> [1 2 3] → 6 [4 6 8 1] → 17	<b>Abbreviate</b> Allen Newell → A.N. Herb Simon → H.S.	<b>Phone Numbers</b> (555) 867-5309 (650) 555-2368	↶ ↷ ↸ ↹
<b>Double</b> [1 2 3 4] → [2 4 6 8] [6 5 1] → [12 10 2]	<b>Drop Last Characters</b> jabberwocky → jabberw copycat → cop	<b>Currency</b> \$100.25 \$4.50	✿ ⋯ ⋮ ⋆
<b>Check Evens</b> [0 2 3] → [T T F] [2 4 9 6] → [T T F T]	<b>Extract</b> see spot(run) → run a (bee) see → bee	<b>Dates</b> Y1775/0704 Y2000/0101	⊗ ⊖ ⊚ ⊛
<b>Block Towers</b> 	<b>Symbolic Regression</b>  $(-2.4x - 0.9) / (x - 4.4)(x - 0.9)$ $0.3x^3 + 1.1x^2 - 2.0x + 0.6$	<b>Recursive Programming</b> <b>Filter</b> [■■■■■] → [■■] [■■■■■■] → [■■■■] [■■■■■■■] → [■■■■■] <b>Length</b> [■■■■■] → 4 [■■■■■■] → 6 [■■■■] → 3	<b>Physics</b> $KE = \frac{1}{2} m  \vec{v} ^2$ $\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$ $\vec{F} \propto \frac{q_1 q_2}{ \vec{r}'_1 - \vec{r}'_2 ^2} \hat{\vec{r}} = \hat{\vec{r}}$ $R_{total} = \left( \sum_i \frac{1}{R_i} \right)^{-1}$

# DreamCoder Domains

## List Processing

### Sum List

[1 2 3] → 6

[4 6 8 1] → 17

### Double

[1 2 3 4] → [2 4 6 8]

[6 5 1] → [12 10 2]

### Check Evens

[0 2 3] → [T T F]

[2 4 9 6] → [T T F T]

## Text Editing

### Abbreviate

Allen Newell → A.N.

Herb Simon → H.S.

### Drop Last Characters

jabberwocky → jabberw

copycat → cop

### Extract

see spot(run) → run

a (bee) see → bee

## Regexes

### Phone Numbers

(555) 867-5309

(650) 555-2368

### Currency

\$100.25

\$4.50

### Dates

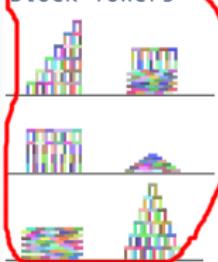
Y1775/0704

Y2000/0101

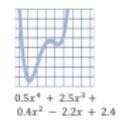
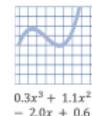
## LOGO Graphics



## Block Towers



## Symbolic Regression



## Recursive Programming

### Filter

[■■■■] → [■■■]  
[■■■■■] → [■■■■]  
[■■■■] → [■■■]

### Length

[■■■■] → 4  
[■■■■■] → 6  
[■■■] → 3

### Index List

0, [■■■■■■] → ■  
1, [■■■■■■] → ■■  
1, [■■■■■■] → ■■■

### Every Other

[■■■■■■] → [■■]  
[■■■■■■] → [■■■■]  
[■■■■■■] → [■■■■■]

## Physics

$$KE = \frac{1}{2} m |\vec{v}|^2$$

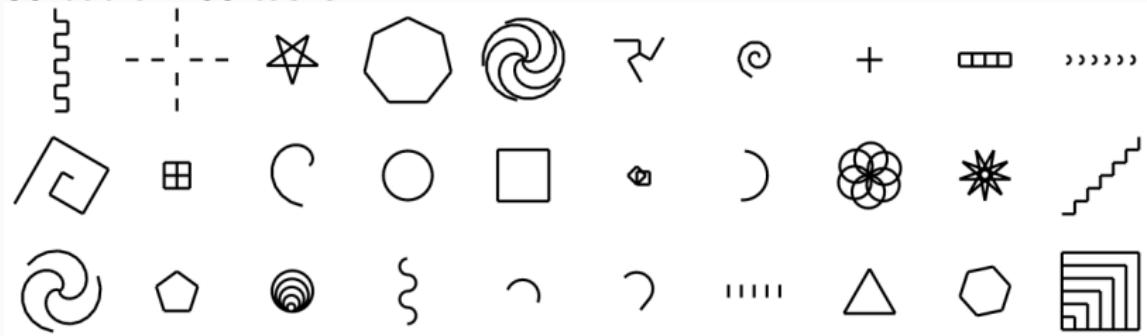
$$\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$$

$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}_1 - \vec{r}_2|^2} \vec{r}_1 - \vec{r}_2$$

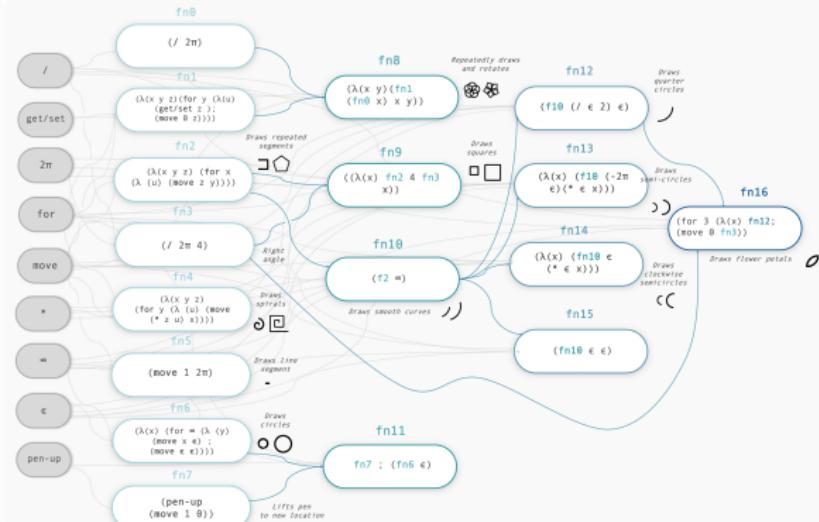
$$V_{total} = \left( \sum_i \frac{1}{R_i} \right)^{-1}$$

# LOGO Graphics

30 out of 160 tasks



# LOGO Graphics – learning interpretable library of concepts



$\{ \text{fn8 } 5 \cdot (\text{fn4} \cdot (* \cdot 2 \cdot = \cdot e)) \}$

$\{ \text{fn10 } 5 \cdot (\lambda(x) (\text{fn14 } 2); (\text{fn13 } 2)) \}$

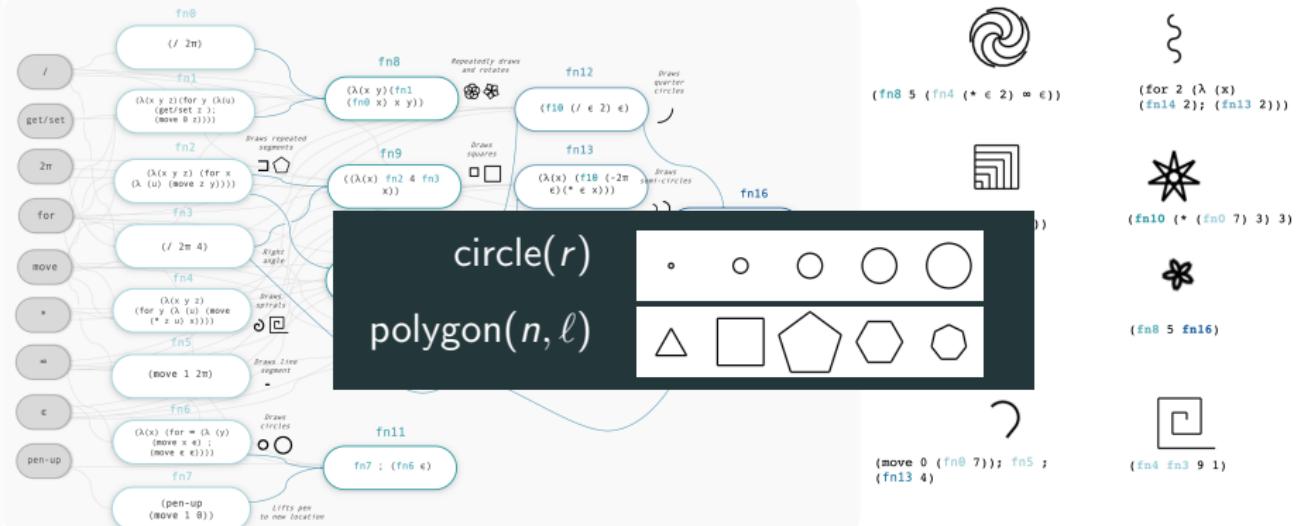
$\{ \text{fn10 } (* \cdot (\text{fn0 } 7) \cdot 3) \cdot 3 \}$

$\{ \text{fn8 } 6 \cdot (\text{fn7} \cdot ; \text{fn5} \cdot ; \text{fn7} \cdot ; \text{fn5} \cdot ) \}$

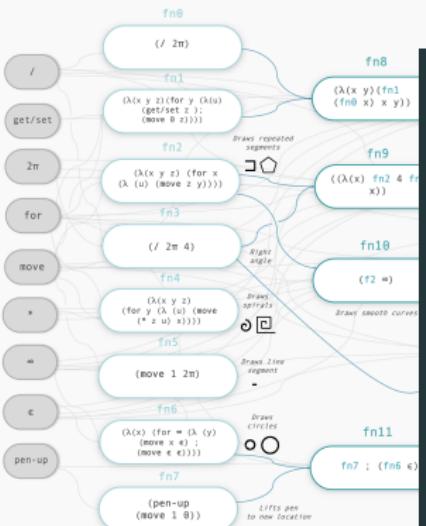
$\{ \text{fn0 } 5 \cdot \text{fn16} \}$

$\{ \text{move } 0 \cdot (\text{fn0 } 7); \text{fn5} \cdot ; \text{fn13 } 4 \}$

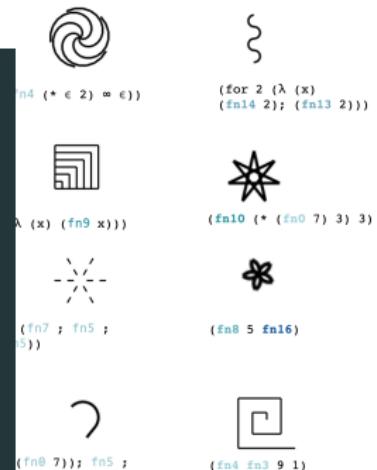
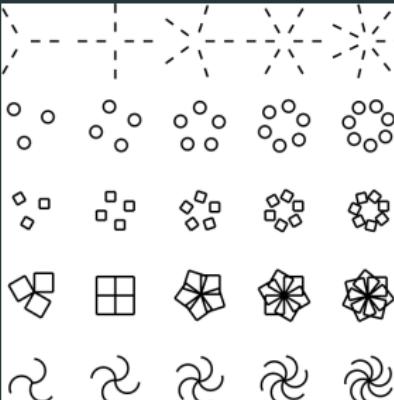
# LOGO Graphics – learning interpretable library of concepts



# LOGO Graphics – learning interpretable library of concepts



## radial symmetry( $n$ , body)

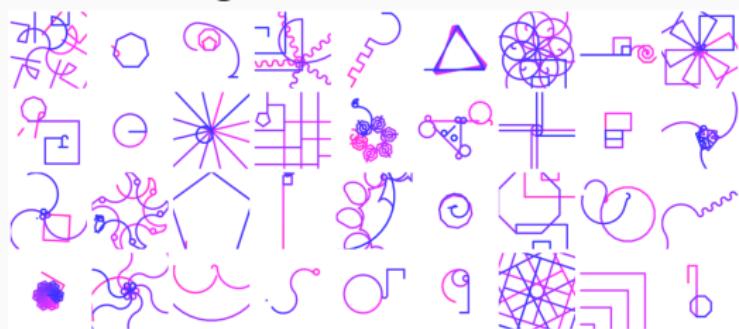


# what does DreamCoder dream of?

before learning

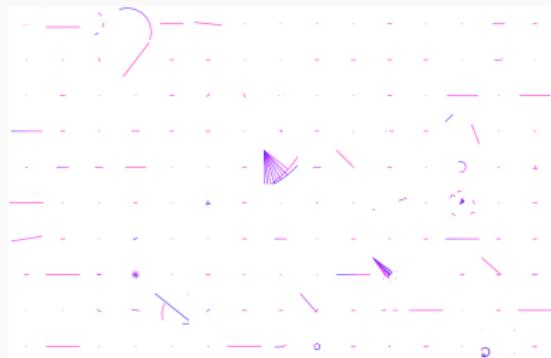


after learning

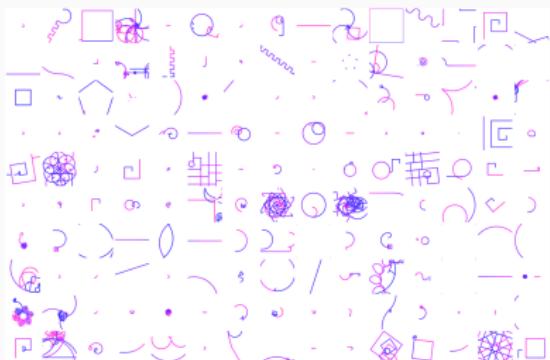


# what does DreamCoder dream of?

before learning

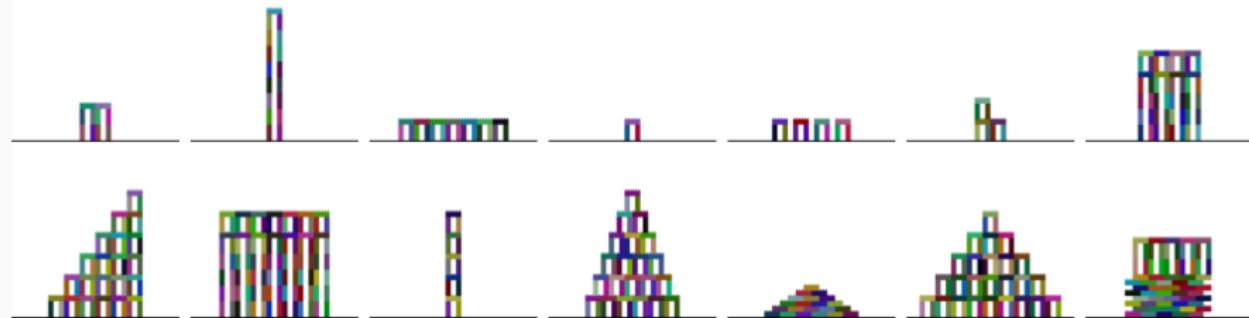


after learning



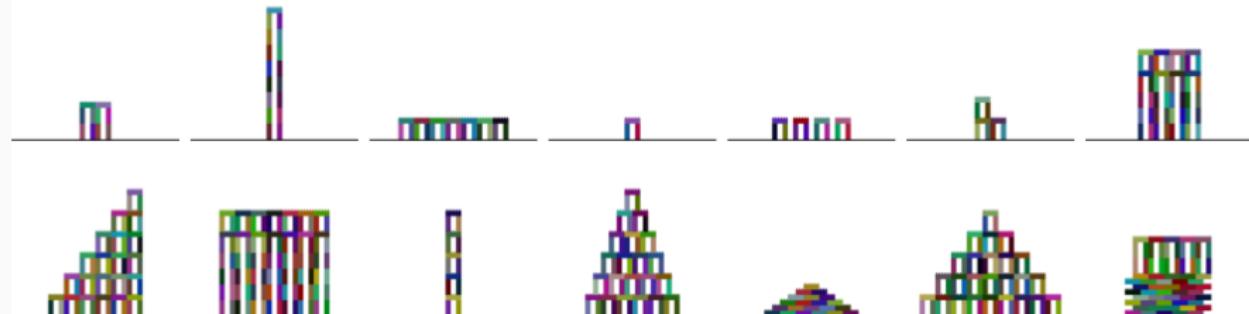
# Planning to build towers

example tasks (112 total)

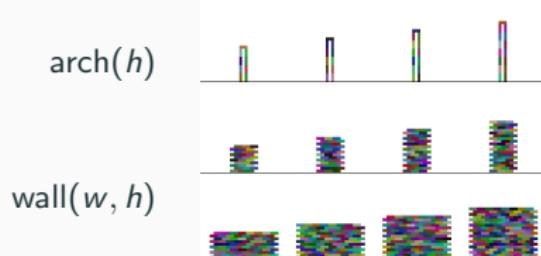


# Planning to build towers

example tasks (112 total)

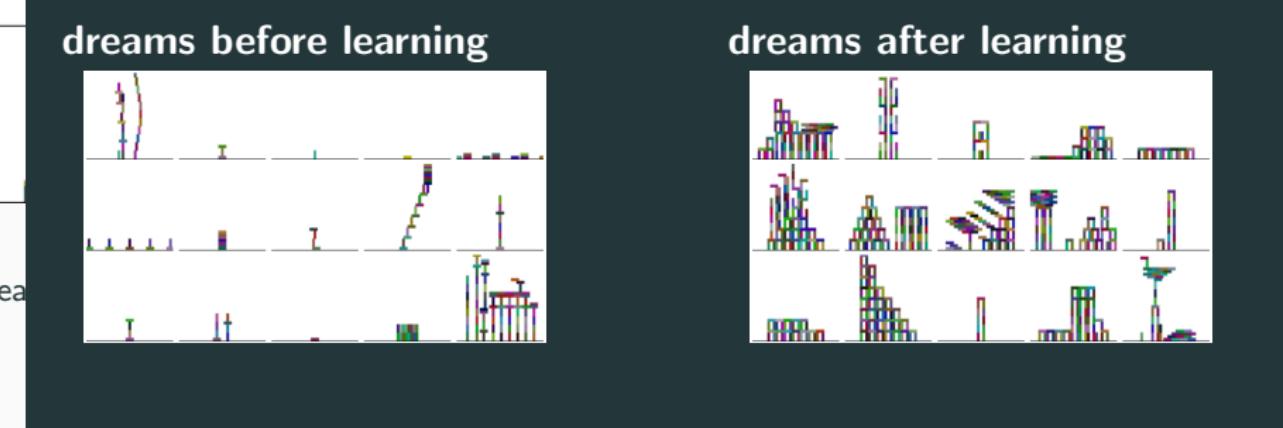


learned library routines ( $\approx 20$  total)

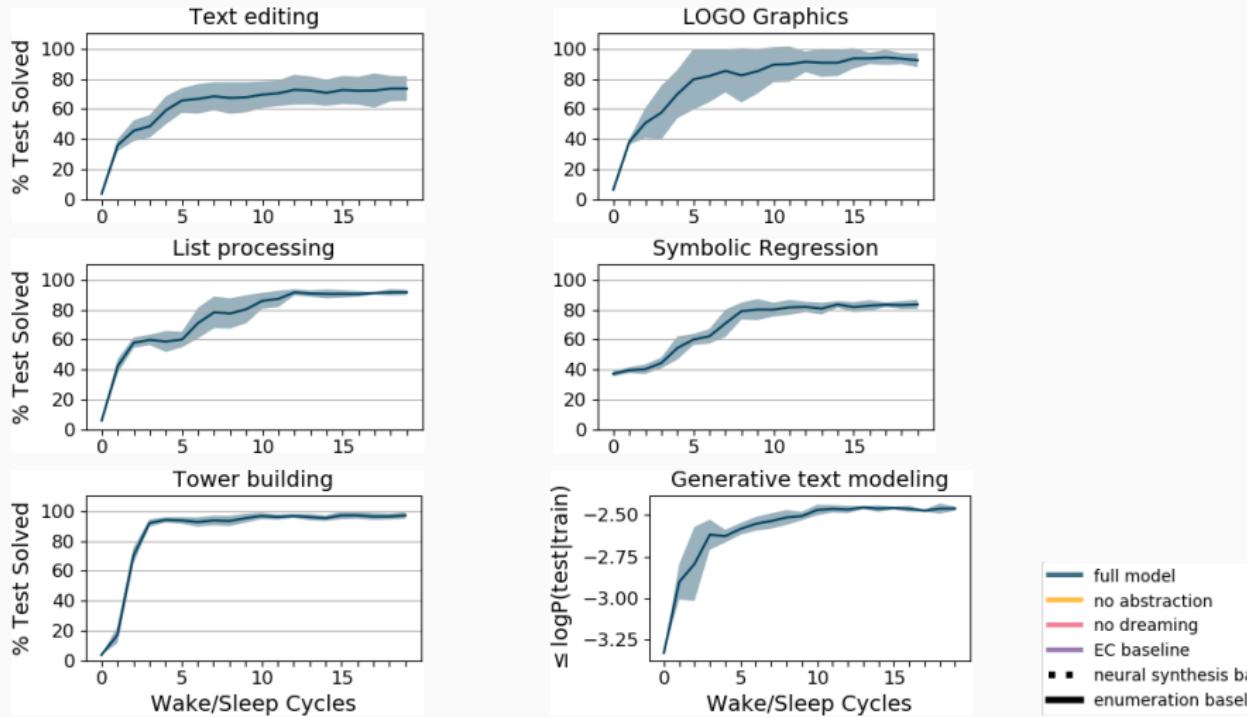


# Planning to build towers

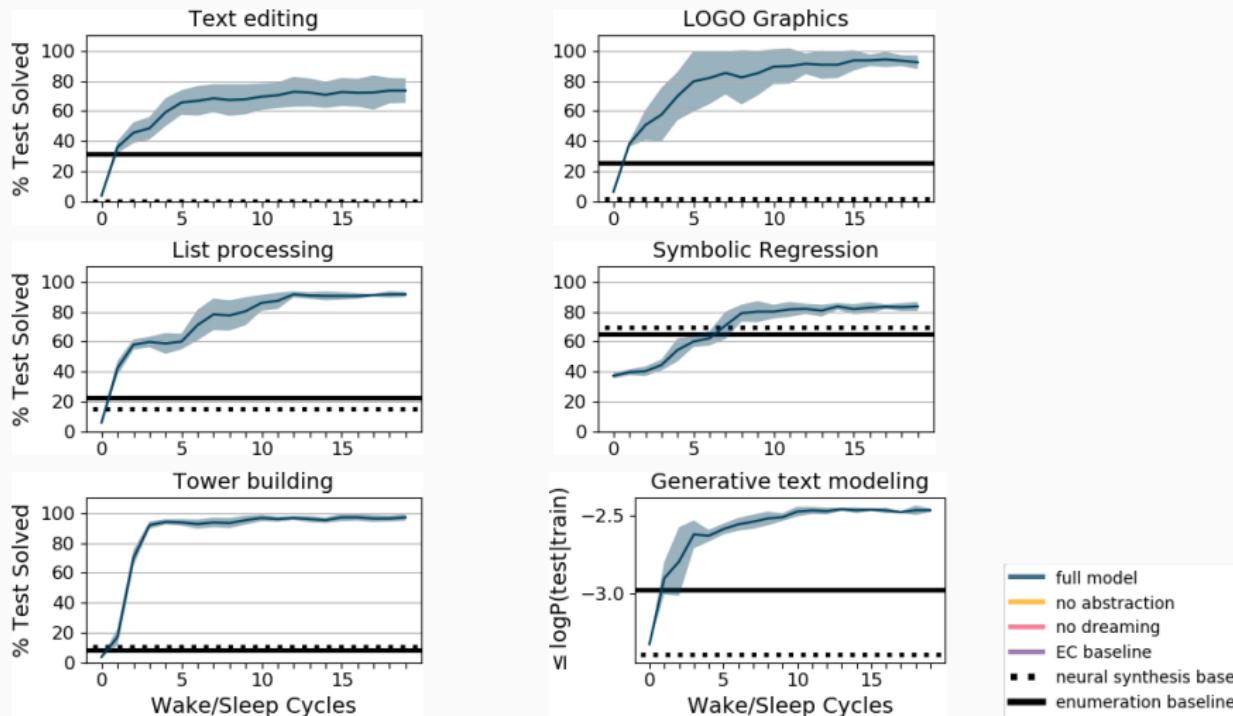
example tasks (112 total)



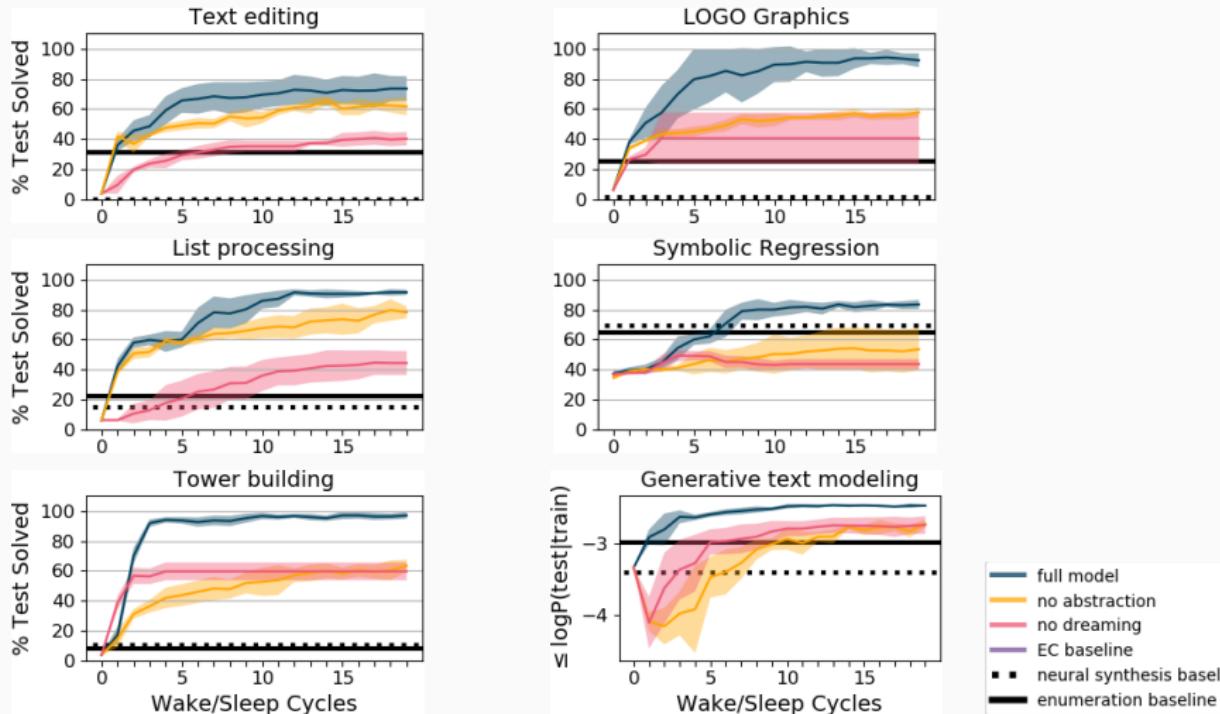
# synergy between dreaming and library learning



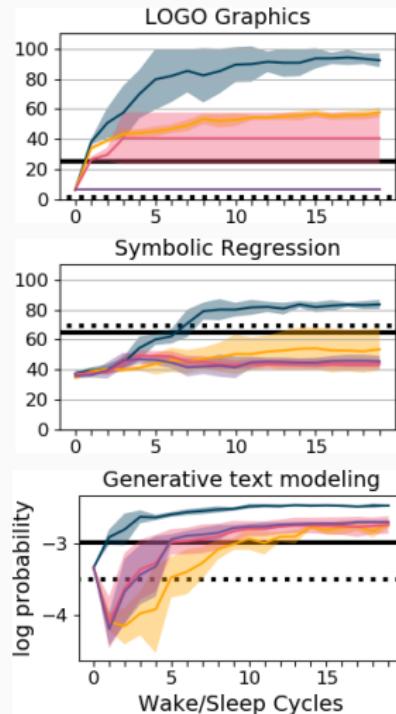
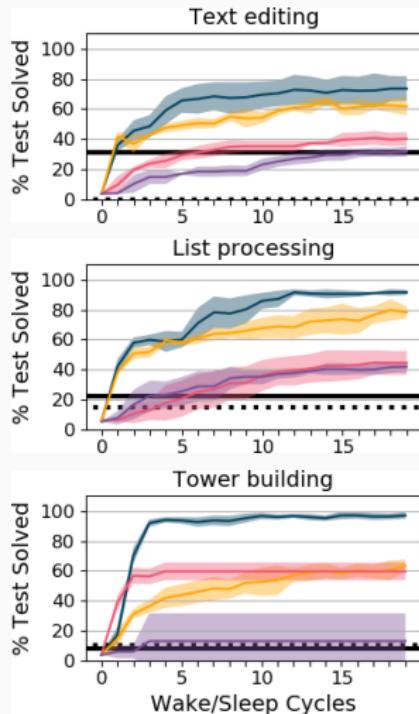
# synergy between dreaming and library learning



# synergy between dreaming and library learning

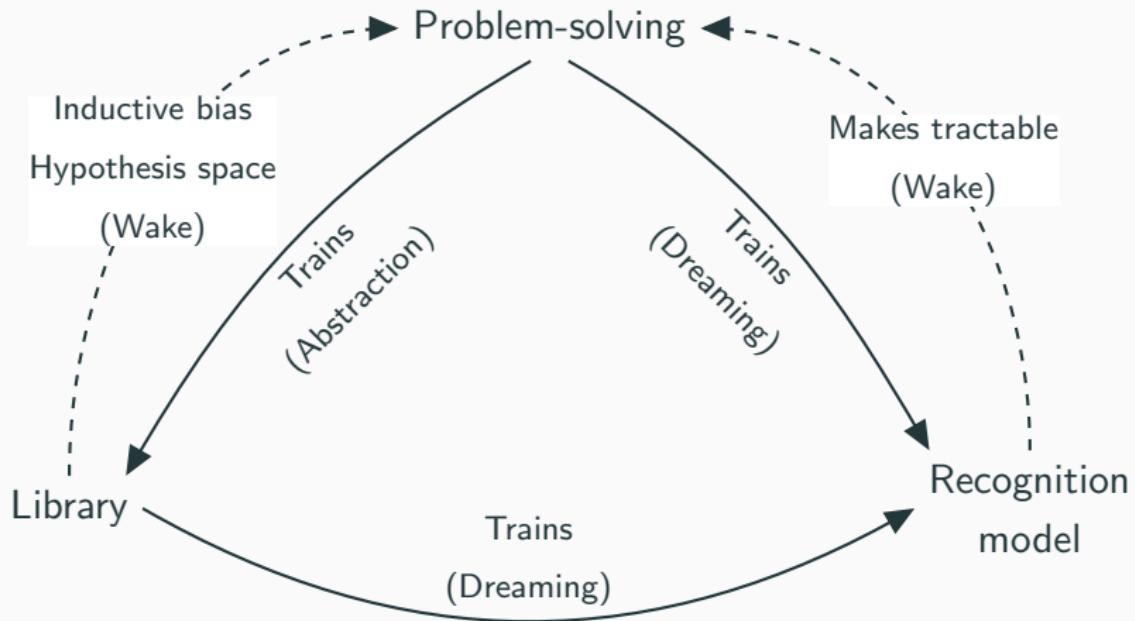


# synergy between dreaming and library learning

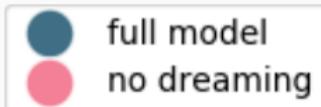
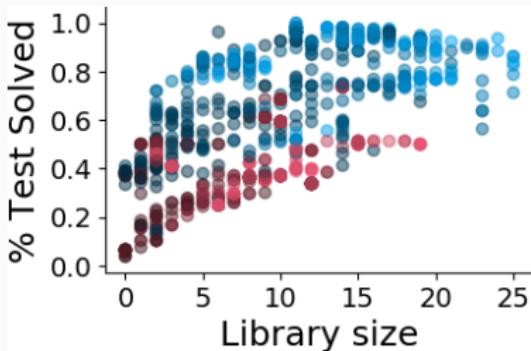
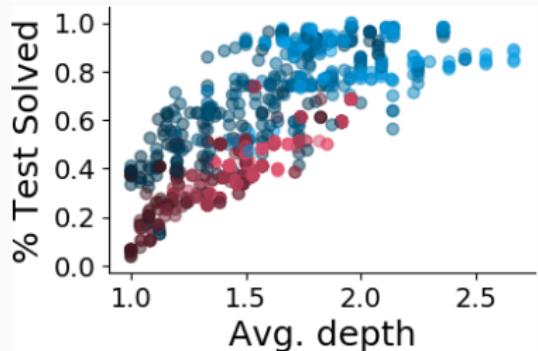


- full model
- no abstraction
- no dreaming
- EC baseline
- neural synthesis baseline
- enumeration baseline

# synergy between dreaming and library learning



# Evidence for dreaming bootstrapping better libraries



Dark→Light: Early in learning→Later in learning

## DSL

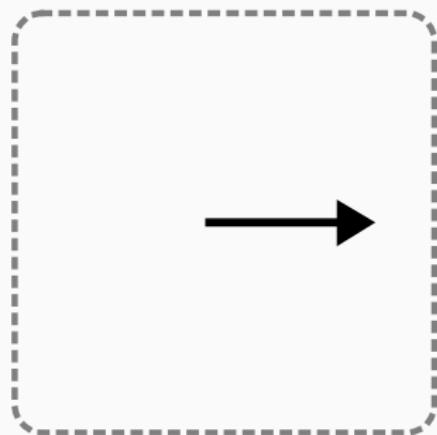
OP ::= FW x | RT x | UP | DOWN | SET state

## Tasks

task : image



FW 1



## DSL

OP ::= FW x | RT x | UP | DOWN | SET state

## Tasks

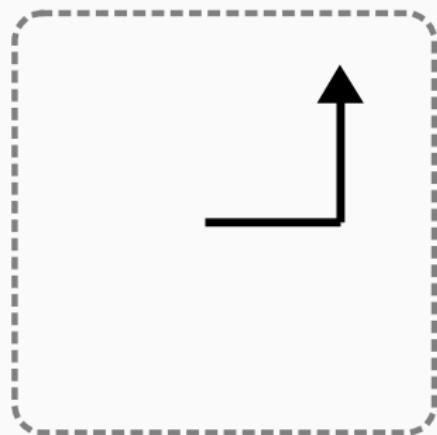
task : image



FW 1

RT  $\frac{\pi}{2}$

FW 1



## DSL

OP ::= FW x | RT x | UP | DOWN | SET state

## Tasks

task : image



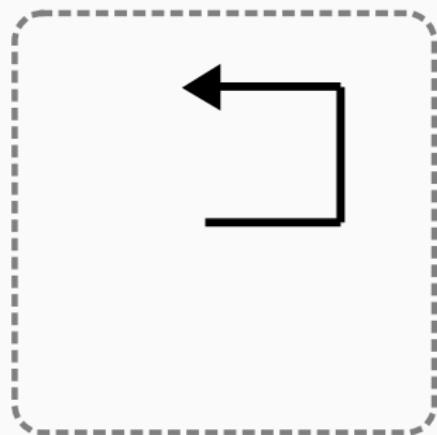
FW 1

RT  $\frac{\pi}{2}$

FW 1

RT  $\frac{\pi}{2}$

FW 1



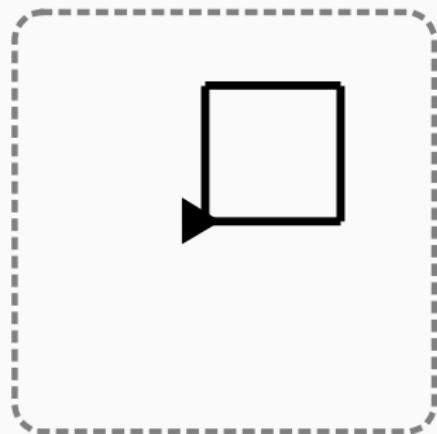
## DSL

OP ::= FW x | RT x | UP | DOWN | SET state

## Tasks

task : image

```
for i in range(4)
> FW 1
> RT π/2
```



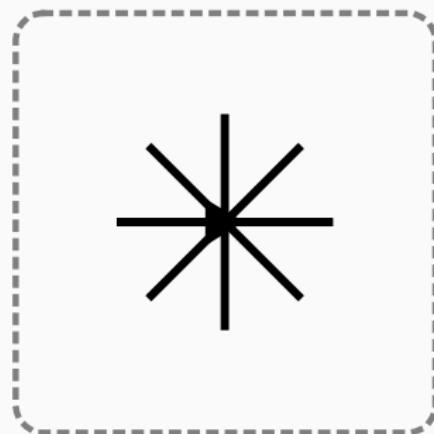
## DSL

OP ::= FW x | RT x | UP | DOWN | SET state

## Tasks

task : image

```
for i in range(8)
> FW 1
> SET origin
> RT  $\frac{2\pi}{8}$ 
```



## DSL

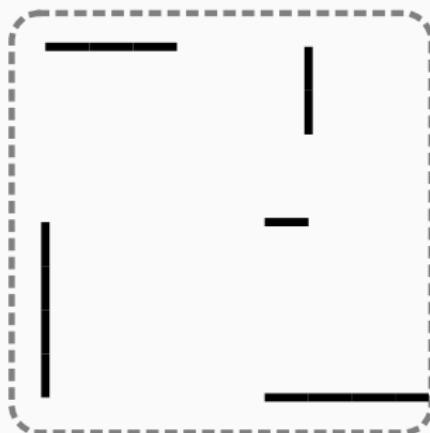
OP ::= FW x | RT x | UP | DOWN | SET state



## Tasks

task : image

```
for i in range(8)
> PU
> FW  $\frac{i}{2}$ 
> PD
> FW  $\frac{i}{2}$ 
> RT  $\frac{\pi}{2}$ 
```



## DSL

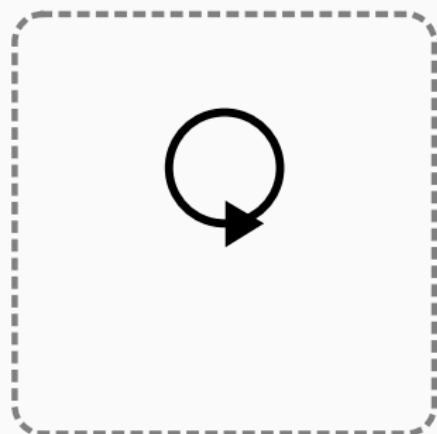
OP ::= FW x | RT x | UP | DOWN | SET state

## Tasks

task : image

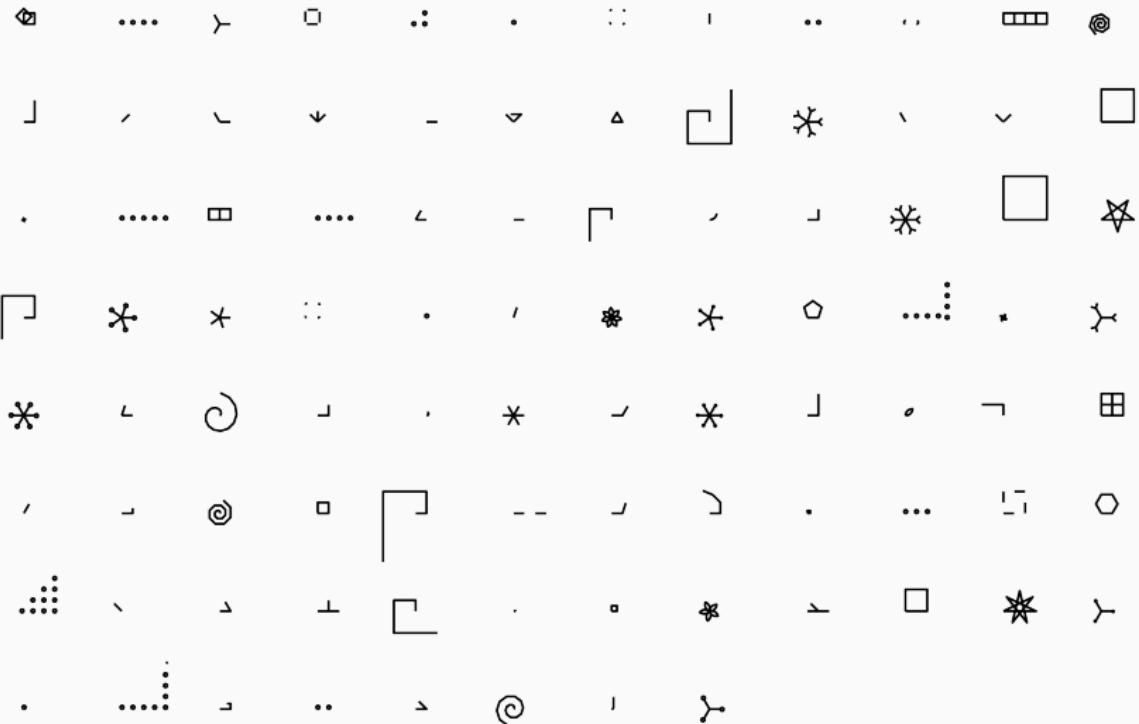


```
for i in range( $\infty$ )
> FW ε
> RT ε
```



NUM ::= 1 |  $\pi$  |  $\infty$  |  $\varepsilon$  | + | - | \* | /

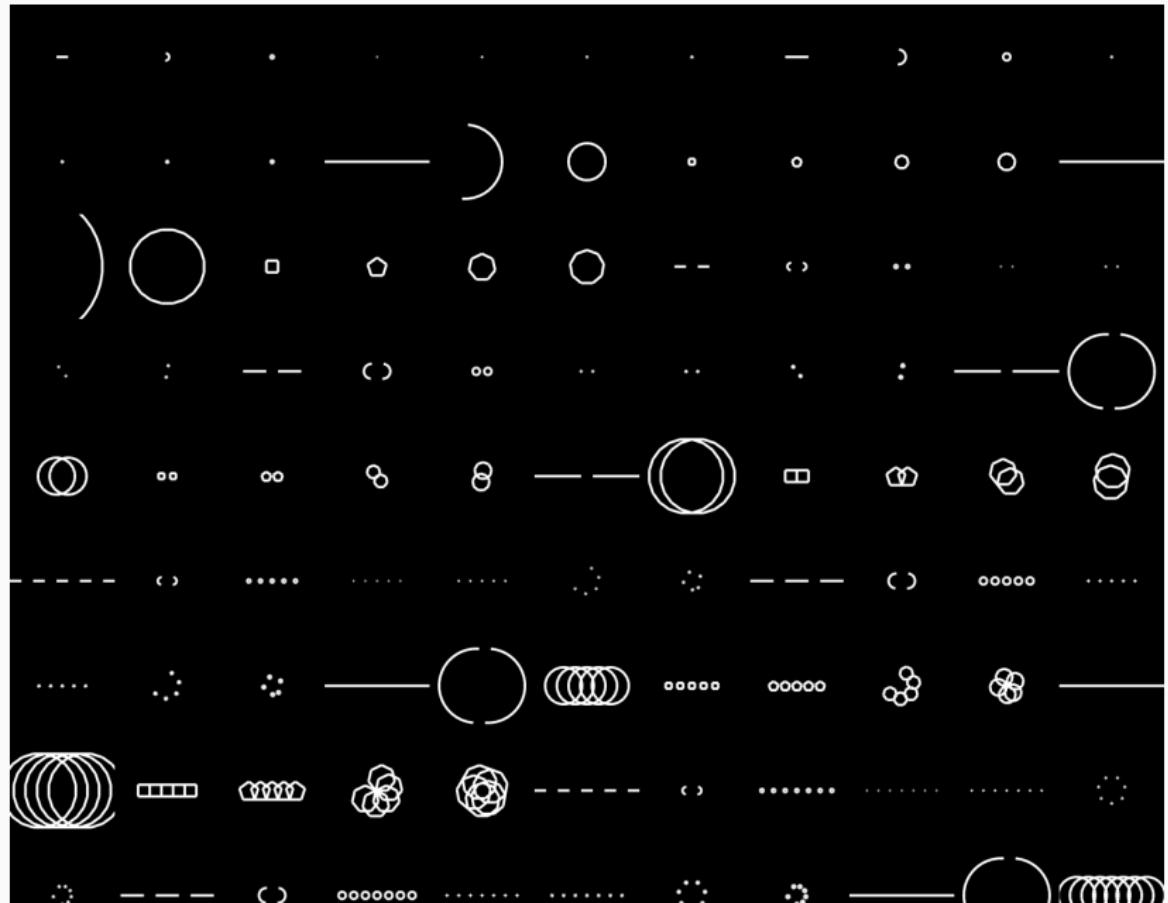
# Turtle graphics — Training tasks



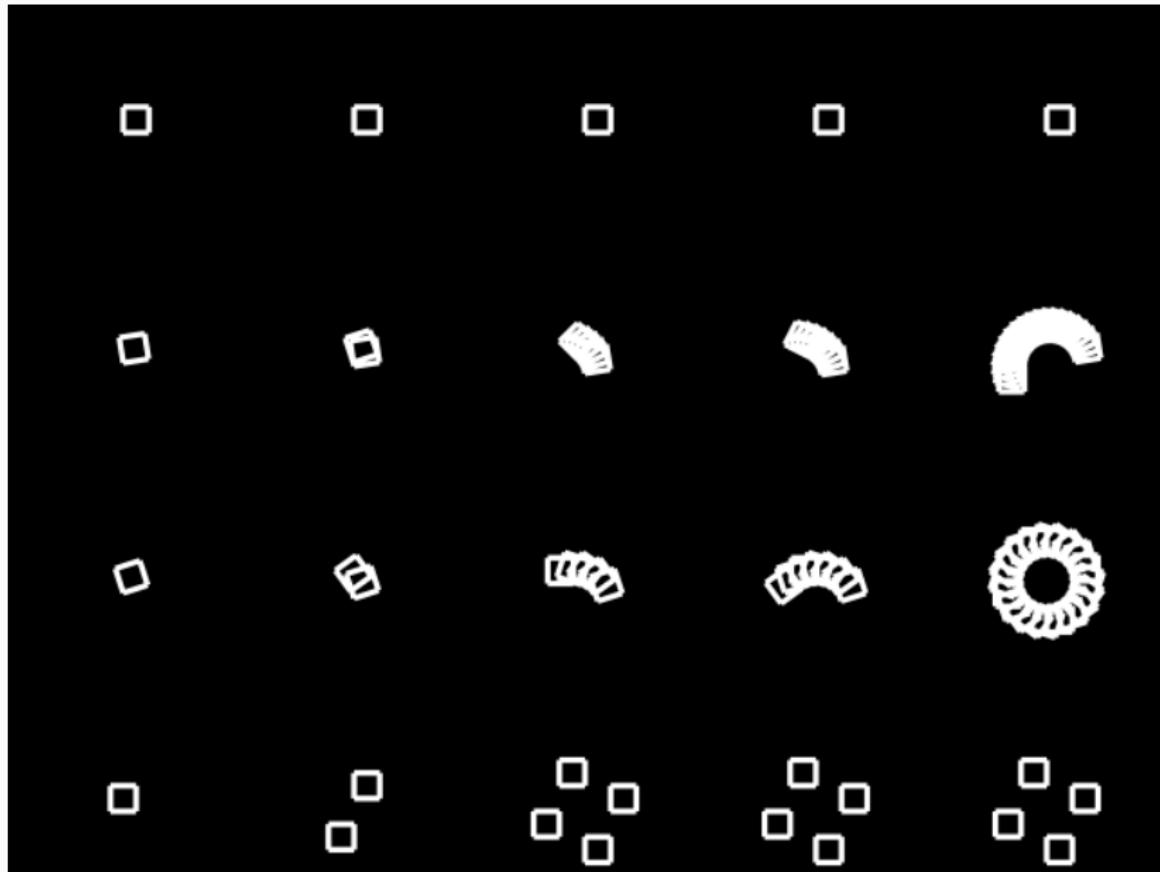
## Turtle graphics — Illustration of learned DSL



## Turtle graphics — Illustration of learned DSL



## Turtle graphics — Illustration of learned DSL



## Turtle graphics — Illustration of learned DSL



## Turtle graphics — Illustration of learned DSL

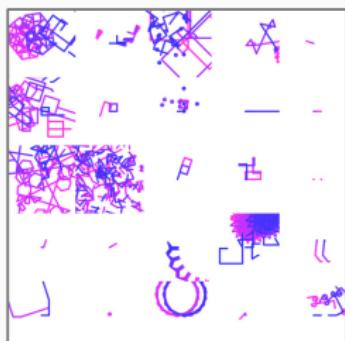
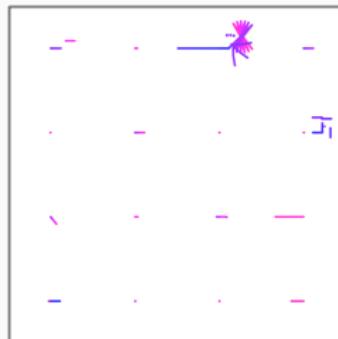


## Turtle graphics — Illustration of learned DSL

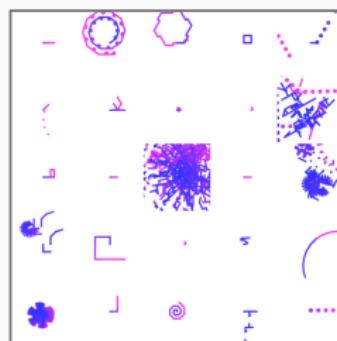


# Turtle graphics — Dreams

Before training

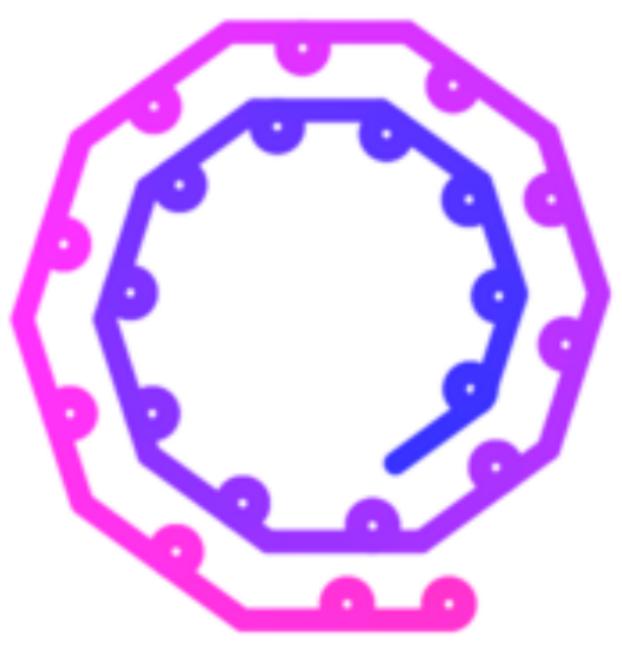


Plateau 5 minutes

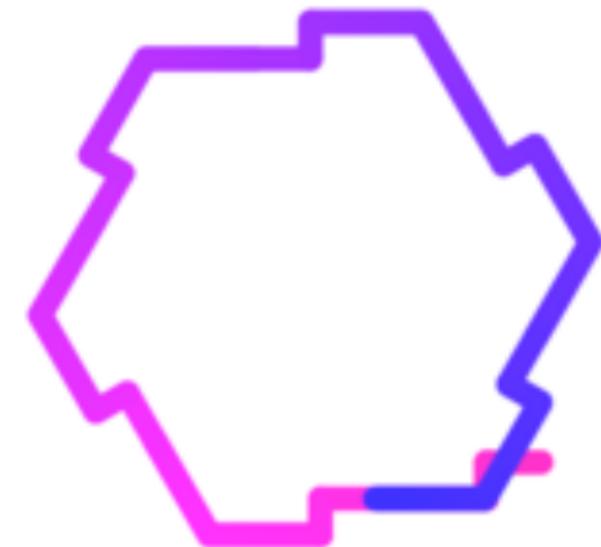


Plateau 2 hours

## Turtle graphics — Dreaming from learned generative model



## Turtle graphics — Dreaming from learned generative model



## Turtle graphics — Dreaming from learned generative model



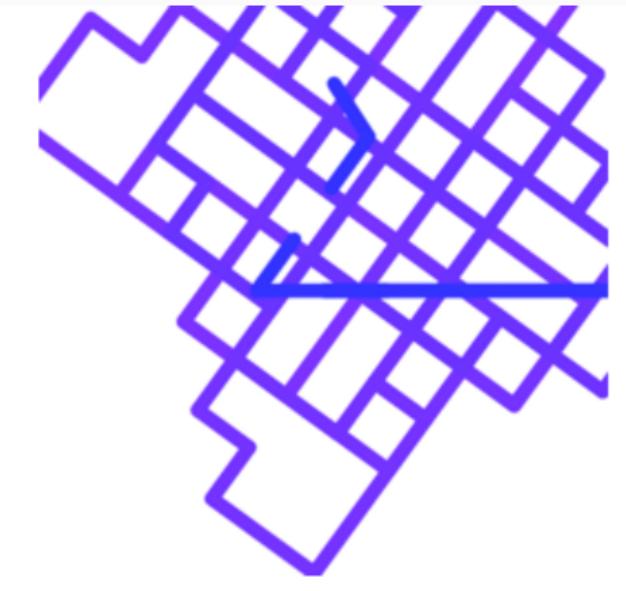
## Turtle graphics — Dreaming from learned generative model



## Turtle graphics — Dreaming from learned generative model



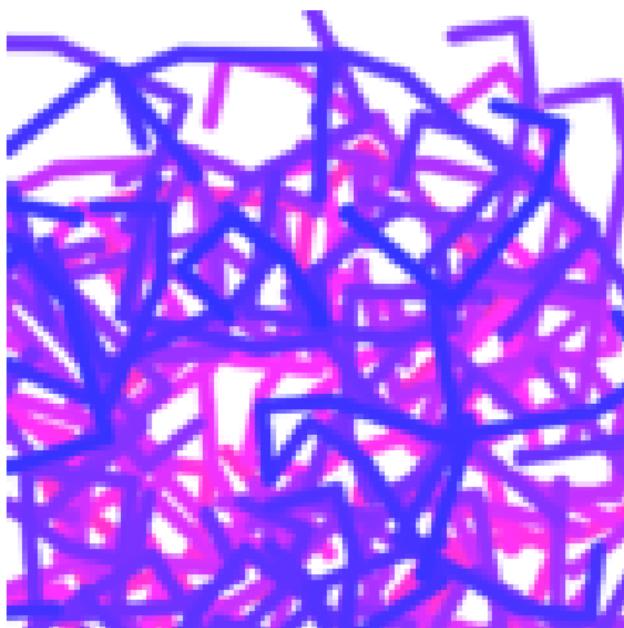
## Turtle graphics — Dreaming from learned generative model



## Turtle graphics — Dreaming from learned generative model



## Turtle graphics — Dreaming from learned generative model



# Vision

## More human-like machine intelligence

- Acquiring a domain-specific representation (DSL)
- Learning to use that representation (recognition model)

DreamCoder: an algorithm for jointly realizing these goals

```
f2(p,f,n,x) = (if (p x) nil  
                  (cons (f x) (f2 (n x))))  
(f2: unfold)  
f3(i,l) = (if (= i 0) (car l)  
              (f3 (f1 i) (cdr l)))  
(f3: index)  
f4(f,l,x) = (if (empty? l) x  
                  (f (car l) (f4 (cdr l))))  
(f4: fold)  
f5(f,l) = (if (empty? l) nil  
              (cons (f (car l)) (f5 (cdr l))))  
(f5: map)
```

Symbolic Regression	
$f(x) = (f_1 \times)$	$f(x) = (f_6 \times)$
$f(x) = (f_4 \times)$	$f(x) = (f_3 \times)$
$f_0(x) = (+ \times \text{real})$	
$f_1(x) = (f_0 (\star \text{real} \times))$	
$f_2(x) = (f_1 (\star \times (f_0 \times)))$	
$f_3(x) = (f_0 (\star \times (f_2 \times)))$	
$f_4(x) = (f_0 (\star \times (f_3 \times)))$	
<i>(f<sub>4</sub>: 4th order polynomial)</i>	
$f_5(x) = (/ \text{real} \times)$	
$f_6(x) = (f_5 (f_0 \times))$	
	<i>(f<sub>6</sub>: rational function)</i>



## More human-like machine intelligence

- Acquiring a domain-specific representation (DSL)
- Learning to use that representation (recognition model)

DreamCoder: an algorithm for jointly realizing these goals

```
f2(p,f,n,x) = (if (p x) nil  
                  (cons (f x) (f2 (n x))))  
(f2: unfold)  
f3(i,l) = (if (= i 0) (car l)  
            (f3 (f1 i) (cdr l)))  
(f3: index)  
f4(f,l,x) = (if (empty? l) x  
              (f (car l) (f4 (cdr l))))  
(f4: fold)  
f5(f,l) = (if (empty? l) nil  
              (cons (f (car l)) (f5 (cdr l))))  
(f5: map)
```

Symbolic Regression	
	$f(x) = (f_1 \times)$
	$f(x) = (f_6 \times)$
	$f(x) = (f_4 \times)$
	$f(x) = (f_3 \times)$
$f_0(x) = (+ \times \text{real})$	
$f_1(x) = (f_0 (\star \text{real} \times))$	
$f_2(x) = (f_1 (\star x (f_0 \times)))$	
$f_3(x) = (f_0 (\star x (f_2 \times)))$	
$f_4(x) = (f_0 (\star x (f_3 \times)))$	<i>(f4: 4th order polynomial)</i>
$f_5(x) = (/ \text{real} \times)$	
$f_6(x) = (f_5 (f_0 \times))$	
	<i>(f6: rational function)</i>

