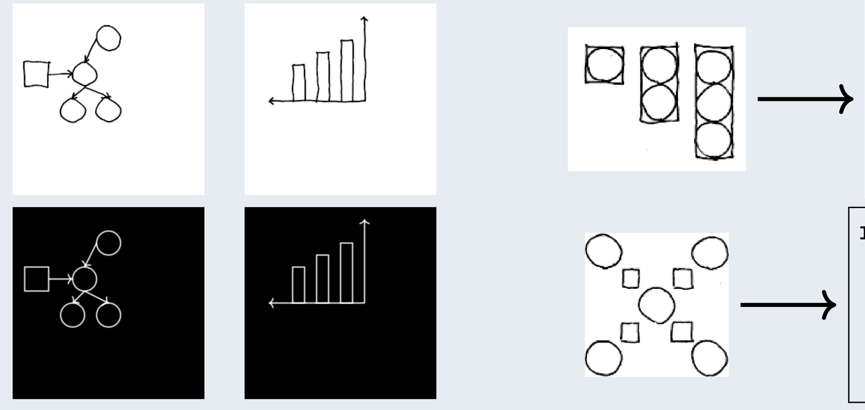
# Learning to Infer Graphics Programs from Hand-Drawn Images

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, Joshua B. Tenenbaum
Massachusetts Institute of Technology & Brown University

## Hand drawings to high-level graphics procedures

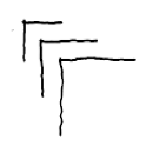
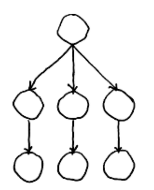
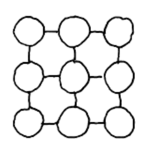
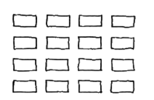
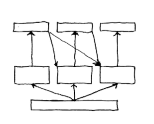
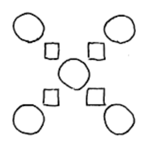
Convert image of hand drawing into LaTeX (left pair: top white is drawing, bottom black is LaTeX render). Infer high-level graphics program from drawing: captures higher-order structure like repetition, motifs, symmetries.



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

```
reflect(y=8)
 for(i<3)
  if(i>0)
   rectangle(3*i-1,2,3*i,3)
  circle(3*i+1,3*i+1)
```

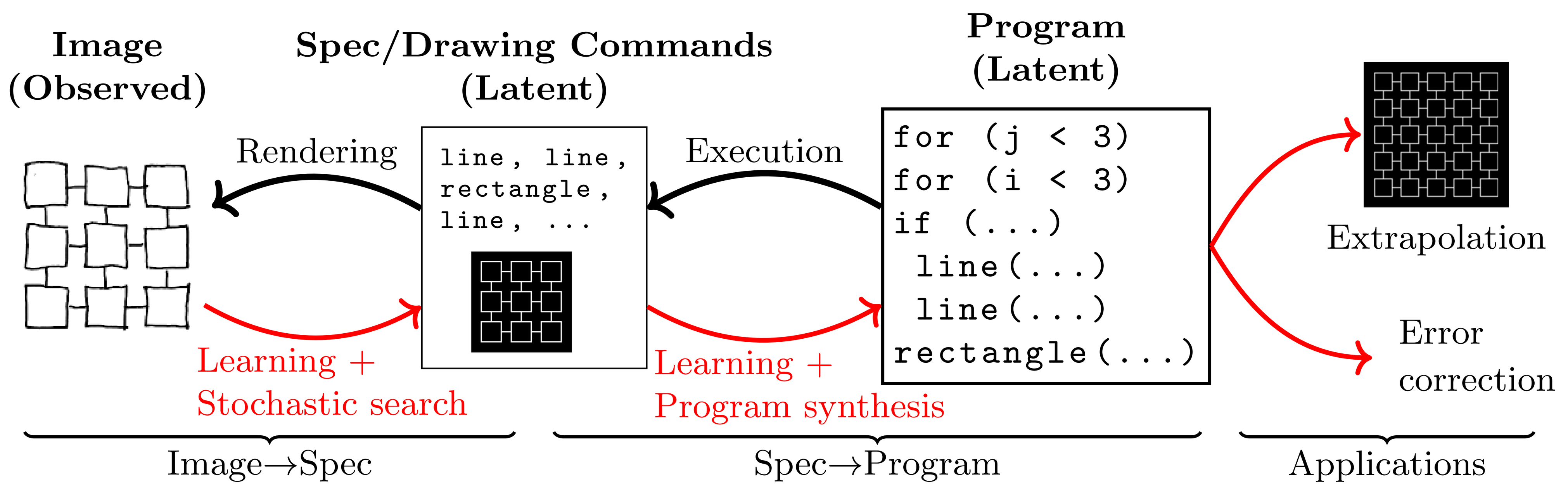
## Example System Outputs

| Drawing | Spec | Program | Compression factor |
|---|---|---|---|
|  | `Line(2,15, 4,15)`<br>`Line(4,9, 4,13)`<br>`Line(3,11, 3,14)`<br>`Line(2,13, 2,15)`<br>`Line(3,14, 6,14)`<br>`Line(4,13, 8,13)` | `for(i<3)`<br>` line(i,-1*i+6,`<br>`      2*i+2,-1*i+6)`<br>` line(i,-2*i+4,i,-1*i+6)` | $\frac{6}{3} = 2$x |
|  | `Line(5,13,2,10,arrow)`<br>`Circle(5,9)`<br>`Circle(8,5)`<br>`Line(2,8, 2,6,arrow)`<br>`Circle(2,5)`<br>*... etc. ...; 13 lines* | `circle(4,10)`<br>`for(i<3)`<br>` circle(-3*i+7,5)`<br>` circle(-3*i+7,1)`<br>` line(-3*i+7,4,-3*i+7,2,arrow)`<br>` line(4,9,-3*i+7,6,arrow)` | $\frac{13}{6} = 2.2$x |
|  | `Circle(5,8)`<br>`Circle(2,8)`<br>`Circle(8,11)`<br>`Line(2,9, 2,10)`<br>`Circle(8,8)`<br>`Line(3,8, 4,8)`<br>`Line(3,11, 4,11)`<br>*... etc. ...; 21 lines* | `for(i<3)`<br>` for(j<3)`<br>`  if(j>0)`<br>`   line(-3*j+8,-3*i+7,`<br>`        -3*j+9,-3*i+7)`<br>`   line(-3*i+7,-3*j+8,`<br>`        -3*i+7,-3*j+9)`<br>`  circle(-3*j+7,-3*i+7)` | $\frac{21}{6} = 3.5$x |
|  | `Rectangle(1,10,3,11)`<br>`Rectangle(1,12,3,13)`<br>`Rectangle(4,8,6,9)`<br>`Rectangle(4,10,6,11)`<br>*... etc. ...; 16 lines* | `for(i<4)`<br>` for(j<4)`<br>`  rectangle(-3*i+9,-2*j+6,`<br>`            -3*i+11,-2*j+7)` | $\frac{16}{3} = 5.3$x |
|  | `Line(3,10,3,14,arrow)`<br>`Rectangle(11,8,15,10)`<br>`Rectangle(11,14,15,15)`<br>`Line(13,10,13,14,arrow)`<br>*... etc. ...; 16 lines* | `for(i<3)`<br>` line(7,1,5*i+2,3,arrow)`<br>` for(j<i+1)`<br>`  if(j>0)`<br>`   line(5*j-1,9,5*i,5,arrow)`<br>`  line(5*j+2,5,5*j+2,9,arrow)`<br>` rectangle(5*i,3,5*i+4,5)`<br>` rectangle(5*i,9,5*i+4,10)`<br>`rectangle(2,0,12,1)` | $\frac{16}{9} = 1.8$x |
|  | `Circle(2,8)`<br>`Rectangle(6,9, 7,10)`<br>`Circle(8,8)`<br>`Rectangle(6,12, 7,13)`<br>`Rectangle(3,9, 4,10)`<br>*... etc. ...; 9 lines* | `reflect(y=8)`<br>` for(i<3)`<br>`  if(i>0)`<br>`   rectangle(3*i-1,2,3*i,3)`<br>`  circle(3*i+1,3*i+1)` | $\frac{9}{5} = 1.8$x |

## Acknowledgments

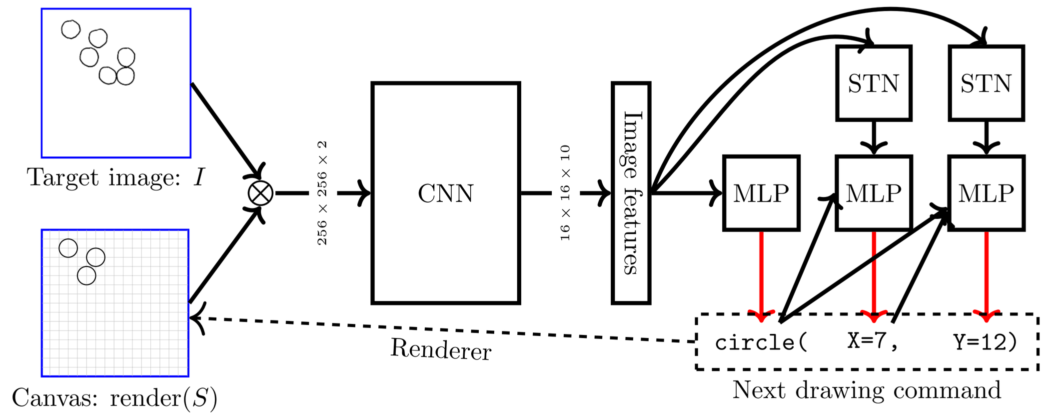
We are grateful for advice from Will Grathwohl and Jiajun Wu on the neural architecture. Funding from NSF GRFP, NSF Award #1753684, the MUSE program (DARPA grant FA8750-14-2-0242), and AFOSR award FA9550-16-1-0012.

## Two-Stage Pipeline




Black arrows: Top–down generative model; Program→Spec→Image. Red arrows: Bottom–up inference procedure. **Bold:** Random variables (image/spec/program)

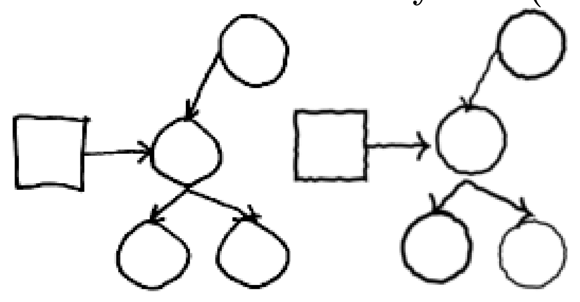
## Parsing Images into Drawing Commands (specs)




Blue: network inputs. Black: network operations. Red: draws from a multinomial. `Typewriter font`: network outputs. Renders on a $16 \times 16$ grid, shown in gray. STN: differentiable attention mechanism. **Combined with stochastic search (Sequential Monte Carlo)**

## Generalizing to real hand drawings

**Train on simulated noisy data (left) & learn distance metric between images (right)**



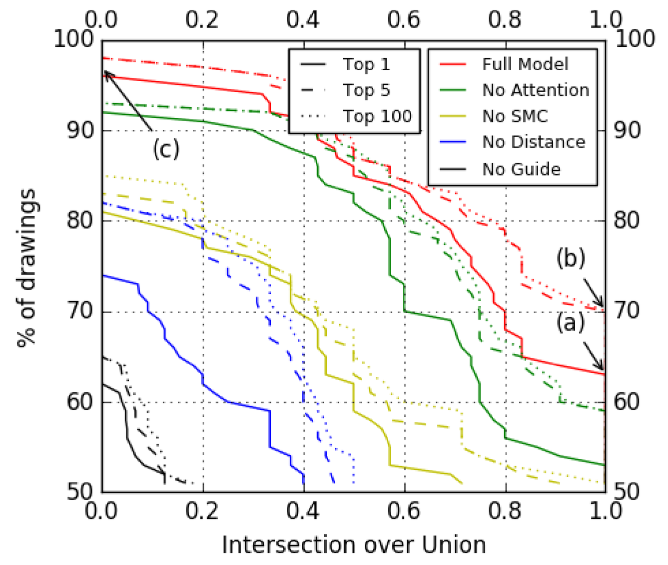
(a): hand drawing (b): noisy render of (a)

**Learned distance metric**

Serves as likelihood surrogate for SMC

$$-\log L_{\text{learned}}(\text{render}(S_1)|\text{render}(S_2)) \approx |S_1 - S_2| + |S_2 - S_1|$$

$S_1$: noisy render of random scene
$S_2$: nonnoisy render of random scene




(Left) NN outputs vs ground truth on hand drawings (measured by IoU), as we consider larger sets of samples (1, 5, 100). (a) for 63% of drawings the model's top prediction is exactly correct; (b) for 70% of drawings the ground truth is in the top 5 model predictions; (c) for 4% of drawings all of the model outputs have no overlap with the ground truth. Red: the full model. Other colors: ablations. Model is at ceiling for synthetic LaTeX output.

## Domain-Specific Language for Graphics Programs

We allow loops (`for`) with conditionals (`if`), vertical/horizontal reflections (`reflect`), variables (Var) and affine transformations ($\mathbb{Z}\times$Var+$\mathbb{Z}$).

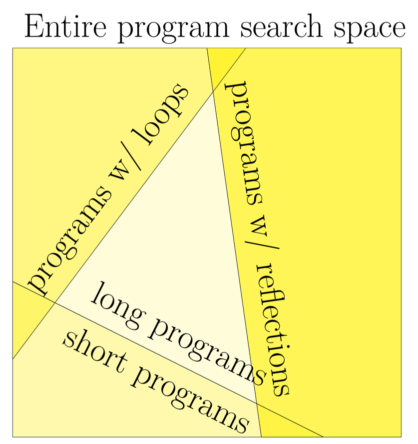
Program→ Statement; ⋯; Statement
Statement→ `circle`(Expression,Expression)
Statement→ `rectangle`(Expression,Expression,Expression,Expression)
Statement→ `line`(Expression,Expression,Expression,Expression,Boolean,Boolean)
Statement→ `for`($0 \leq$ Var $<$ Expression) `{ if` (Var $> 0$) `{` Program `};` Program `}`
Statement→ `reflect`(Axis) `{` Program `}`
Expression→ $\mathbb{Z}\times$Var+$\mathbb{Z}$
Axis→ `X =` $\mathbb{Z}$ | `Y =` $\mathbb{Z}$
$\mathbb{Z} \rightarrow$ an integer

## Learning & Constraint-Based Program Synthesis

**Sketch**: state-of-the-art program synthesizer. Solar-Lezama 2008.
Solves, for spec $S$ & program $p$:

$$\text{program}(S) = \underset{p\in\text{DSL, s.t. } p \text{ consistent w/ } S}{\arg\min} \text{cost}(p)$$

**Learn policy $\pi_\theta$ to accelerate program synthesizer's search.**
$\pi_\theta(\cdot|S) \in \Delta^\Sigma$, where $\Sigma \ni \sigma$ a set of synthesis problem (i.e., $\sigma \in \Sigma$ is a sketch)
**Inference strategy**: Timeshare according to $\pi_\theta(\cdot|S)$, like in Levin Search




$\pi_\theta$(short, no loop/reflect) =
$\pi_\theta$(long, loops) =
$\pi_\theta$(long, no loop/reflect) =
$\pi_\theta$(long, reflects) =
*etc.*

Figure 1: The bias-optimal search algorithm divides the entire (intractable) program search space in to (tractable) program subspaces (written $\sigma$), each of which contains a restricted set of programs. For example, one subspace might be short programs which don't loop. The policy $\pi$ predicts a distribution over program subspaces: weight assigned by $\pi$ indicated by shading

Differentiable loss ($\mathcal{D}$ a corpus of synthesis problems):

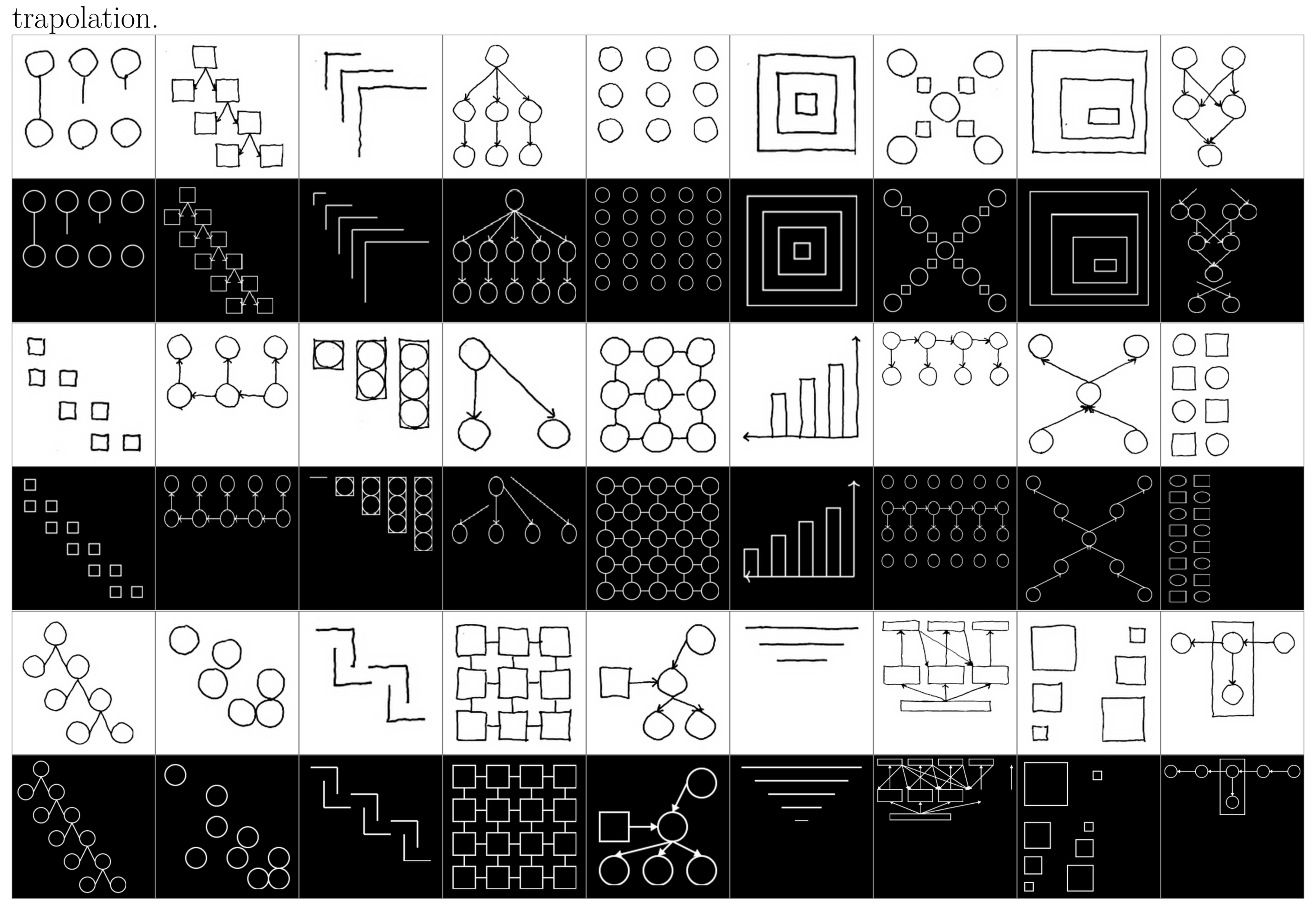
$$\textsc{Loss}(\theta;\mathcal{D}) = \mathbb{E}_{S\sim\mathcal{D}}\left[\min_{\sigma\in\textsc{Best}(S)} \frac{t(\sigma|S)}{\pi_\theta(\sigma|S)}\right] + \lambda\|\theta\|_2^2 \tag{1}$$

where $\sigma \in \textsc{Best}(S)$ if a minimum cost program for $S$ is in $\sigma$.

Experimental results on synthesis policy: See lower right of poster

## Extrapolating Drawings

Automatically extrapolate by increasing loop bounds. Top white: Hand drawing. Bottom black: extrapolation.



## Error correction

'Top down' influences upon perception: reasoning engine (program synthesizer) can influence agent's percept through higher-level considerations like symmetry and alignment.

$$\hat{S}(I) = \underset{S\in\mathcal{F}(I)}{\arg\max}\, L_{\text{learned}}(I|\text{render}(S)) \times \mathbb{P}_\theta[S|I] \times \mathbb{P}_\beta[\text{program}(S)]$$

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

where $\mathcal{F}(I)$ is set of parses output by neural net on image $I$; $\mathbb{P}_\beta[\cdot]$ is prior over programs parameterized by $\beta$; and expectation is taken over a corpus of program synthesis problems.

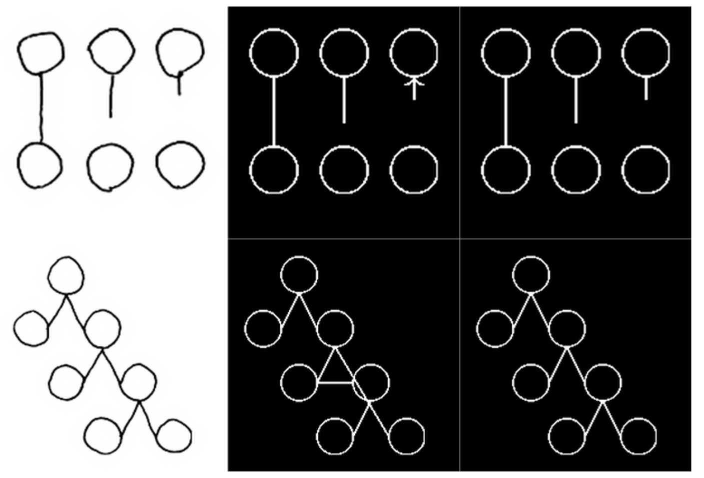
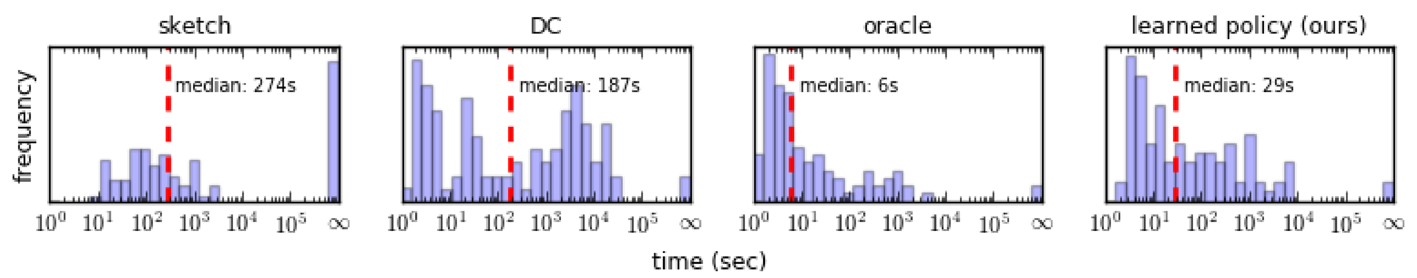


Figure 2: Left: hand drawings. Center: interpretations favored by the deep network. Right: interpretations favored after learning a prior over programs. The prior favors simpler programs, thus (top) continuing the pattern of not having an arrow is preferred, or (bottom) continuing the "binary search tree" is preferred.

## Synthesis times




Time to synthesize a minimum cost program. Sketch: out-of-the-box performance of Sketch. DC: Deep–Coder style baseline that predicts program components, trained like Balog 2016. Oracle: upper bounds the performance of any bias–optimal search policy. $\infty$ = timeout. Red dashed line is median time