Learning to Infer Graphics Programs from Hand-Drawn Images

Kevin Ellis\textsuperscript{1}, Daniel Ritchie\textsuperscript{2}, Armando Solar-Lezama\textsuperscript{1}, Joshua B. Tenenbaum\textsuperscript{1}

\textsuperscript{1}: MIT. \textsuperscript{2}: Brown University.
Learning to Infer Graphics Programs from Hand-Drawn Images
Kevin Ellis¹, Daniel Ritchie², Armando Solar-Lezama¹, Joshua B. Tenenbaum¹

¹: MIT. ²: Brown University.
Learning to Infer Graphics Programs from Hand-Drawn Images

Kevin Ellis\textsuperscript{1}, Daniel Ritchie\textsuperscript{2}, Armando Solar-Lezama\textsuperscript{1}, Joshua B. Tenenbaum\textsuperscript{1}

\textsuperscript{1}: MIT. \textsuperscript{2}: Brown University.
Learning to Infer Graphics Programs from Hand-Drawn Images

Kevin Ellis\textsuperscript{1}, Daniel Ritchie\textsuperscript{2}, Armando Solar-Lezama\textsuperscript{1}, Joshua B. Tenenbaum\textsuperscript{1}

\textsuperscript{1}: MIT. \textsuperscript{2}: Brown University.
Learning to Infer Graphics Programs from Hand-Drawn Images

Kevin Ellis\textsuperscript{1}, Daniel Ritchie\textsuperscript{2}, Armando Solar-Lezama\textsuperscript{1}, Joshua B. Tenenbaum\textsuperscript{1}

\textsuperscript{1}: MIT. \textsuperscript{2}: Brown University.
Learning to Infer Graphics Programs from Hand-Drawn Images
Kevin Ellis¹, Daniel Ritchie², Armando Solar-Lezama¹, Joshua B. Tenenbaum¹

¹: MIT. ²: Brown University.

```latex
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)
```

Hand Drawing  Program

```
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)
```
The image depicts the pipeline for transforming an image (observed) into a program (latent). This pipeline consists of several stages:

1. **Image → Spec**
   - **Rendering**: The observed image is rendered into a set of spec/drawing commands.
   - **Learning + Stochastic search**: These commands are then transformed into a program using learning and stochastic search.

2. **Spec → Program**
   - **Execution**: The program is executed to generate the final output.
   - **Learning + Program synthesis**: The process is further refined through learning and program synthesis.

3. **Program**
   - **Extrapolation**
   - **Error correction**

### Drawing
- **Line**: (3,10,3,14,arrow)
- **Rectangle**: (11,8,15,10), (11,14,15,15), (13,10,13,14,arrow)

### Spec
- **Line**: (7,1,5*i+2,3,arrow)
- **Rectangle**: (5*i,3,5*i+4,5), (5*i,9,5*i+4,10)

### Program
- **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)

---

... etc. ...; 16 lines
Parsing images into \LaTeX\ TikZ Commands


Target image: $I$
Canvas: $\text{render}(S)$

$\text{CNN}$
$\otimes$
Image features

MLP
$\text{circle(}$
STN (attention)
MLP
$X=7,$
STN (attention)
MLP
$Y=12$)

Next drawing command
Renderer

$256 \times 256 \times 2$
$16 \times 16 \times 10$

(a): hand drawing
(b): noisy render of (a)'s spec
Parsing images into \LaTeX\ TikZ Commands


Target image: $I$

Canvas: $\text{render}(S)$

$\text{CNN}$

$\text{MLP}$

$\text{circle(}$

$\text{STN}$ (attention)

$\text{MLP}$

$X=7,$

$\text{STN}$ (attention)

$\text{MLP}$

$Y=12)$

Next drawing command

Renderer

256 × 256 × 2

16 × 16 × 10

(a): hand drawing

(b): noisy render of (a)’s spec

Average number of errors

in sample out of sample

# objects
Parsing images into $\LaTeX$ TikZ Commands


![Diagram showing the process of parsing images into LaTeX TikZ commands](image)

- **Target image:** $I$
- **Canvas:** $\text{render}(S)$
- **CNN**
- **Image features**
- **MLP**
- **circle**
- **STN (attention)**
- **MLP**
- **Renderer**
- **Next drawing command**

(a): hand drawing
(b): noisy render of (a)'s spec

![Graph showing average number of errors](image)
Synthesizing high-level programs from specs (spec=drawing commands)

Constraint-based program synthesis; SAT solver (Solar-Lezama 2008)

\[
\text{program}(S) = \arg \min_{p \in \text{DSL}} \text{cost}(p) \\
\text{cost}(p) \text{ consistent w/ } S
\]

\text{min cost} \approx \text{simple+short}

DSL: Domain Specific Language: variables, arithmetic, loops, conditionals

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program \rightarrow Statement; \cdots; Statement</td>
<td></td>
</tr>
<tr>
<td>Statement \rightarrow circle(Expression,Expression)</td>
<td></td>
</tr>
<tr>
<td>Statement \rightarrow rectangle(Expression,Expression,Expression,Expression,Expression)</td>
<td></td>
</tr>
<tr>
<td>Statement \rightarrow line(Expression,Expression,Expression,Expression,Expression,Boolean,Boolean)</td>
<td></td>
</tr>
<tr>
<td>Statement \rightarrow for(0 \leq \text{Var} &lt; Expression) { if (\text{Var} &gt; 0) { Program }; Program }</td>
<td></td>
</tr>
<tr>
<td>Statement \rightarrow reflect(Axis) { Program }</td>
<td></td>
</tr>
<tr>
<td>Expression \rightarrow \mathbb{Z} \times \text{Var} + \mathbb{Z}</td>
<td></td>
</tr>
<tr>
<td>Axis \rightarrow \mathbb{X} = \mathbb{Z}</td>
<td>\mathbb{Y} = \mathbb{Z}</td>
</tr>
<tr>
<td>\mathbb{Z} \rightarrow \text{an integer}</td>
<td></td>
</tr>
</tbody>
</table>
Learning to quickly synthesize programs

Learn search policy $\pi(\text{program subspace}|\text{spec})$

Think of the subspace as an “ansatz”

OBJECTIVE (cf Bias-Optimal Search, Schmidhuber 2004):

$$\pi^* = \arg\min_{\pi} \sum_{\text{spec}} \min_{\text{all subspaces}} \mathbb{E}[\text{time to exhaustively search the subspace}] \pi(\text{subspace}|\text{spec})$$

Entire program search space

- $\pi(\text{short, no loop/reflect}|S) = \square$
- $\pi(\text{long, loops}|S) = \square$
- $\pi(\text{long, no loop/reflect}|S) = \square$
- $\pi(\text{long, reflects}|S) = \square$
- etc.
Learning to quickly synthesize programs

Learn search policy $\pi(\text{program subspace}|\text{spec})$

Think of the subspace as an "ansatz"

OBJECTIVE (cf Bias-Optimal Search, Schmidhuber 2004):

$$\pi^* = \arg \min_\pi \sum_{\text{spec}} \min_{\text{all subspaces}} \text{E}\left[\text{time to exhaustively search the subspace}\right] \pi(\text{subspace}|\text{spec})$$

Entire program search space

- $\pi(\text{short, no loop/reflect}|S) = \square$
- $\pi(\text{long, loops}|S) = \square$
- $\pi(\text{long, no loop/reflect}|S) = \square$
- $\pi(\text{long, reflects}|S) = \square$
- etc.

Median time: 274s

Median time: 29s
Application: Error correction

learn prior over programs (simple $\approx$ better), jointly infer likely parse + program

Top-down influence upon perception
Application: Extrapolating drawings
Visual input → Program: Poster AB #25

Hand Drawing

LaTeX

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)