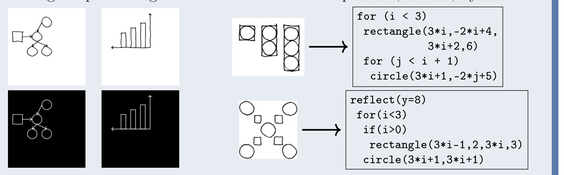
Learning to Infer Graphics Programs from Hand-Drawn Images

Kevin Ellis, Daniel Ritchie, Lucas Morales, Mathias Sablé Meyer, Joshua B. Tenenbaum, Armando Solar-Lezama

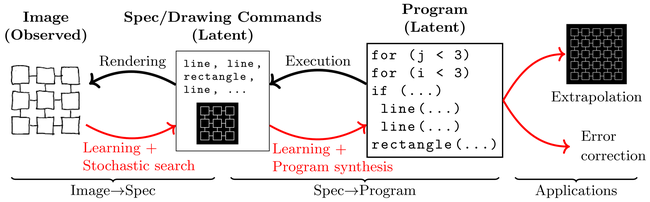
Massachusetts Institute of Technology & Brown University

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.

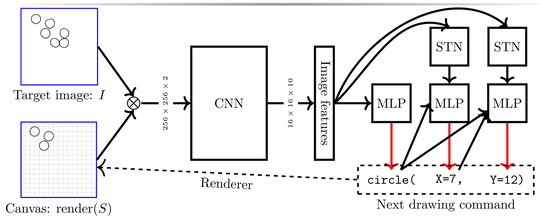


Two-Stage Pipeline

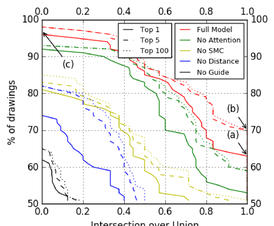


Black arrows: Top–down generative model; Program→Spec→Image. Red arrows: Bottom–up inference procedure. **Bold:** Random variables (image/spec/program)

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 × 16 grid, shown in gray. STN: differentiable attention mechanism. **Combined with stochastic search (Sequential Monte Carlo)**



(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 (ℤ×Var+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 ≤ Var < Expression) `{ if` (Var > 0) `{` Program `}; `Program `}`
Statement→ `reflect`(Axis) `{` Program `}`
Expression→ ℤ×Var+ℤ
Axis→ `X = ` ℤ | `Y = ` ℤ
ℤ → 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 π_θ 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

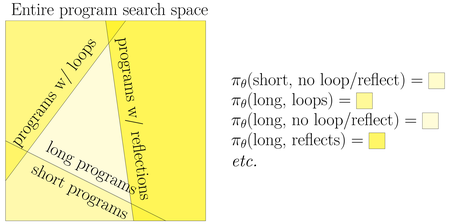
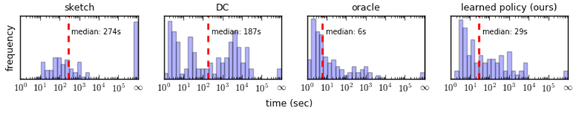


Figure 1: The bias-optimal search algorithm divides the entire (intractable) program search space in to (tractable) program subspaces (written σ), each of which contains a restricted set of programs. For example, one subspace might be short programs which don't loop. The policy π predicts a distribution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

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

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

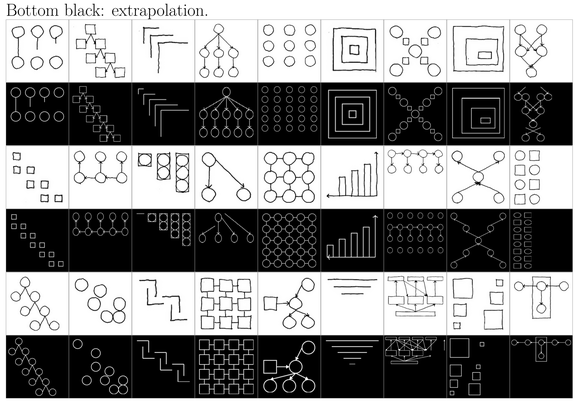
where $\sigma \in \text{BEST}(S)$ if a minimum cost program for S is in σ.



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. ∞ = timeout. Red dashed line is median time

Extrapolating Drawings

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

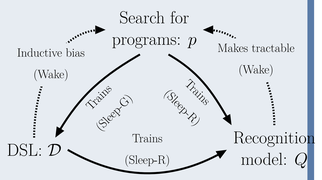


Example System Outputs

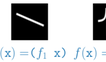
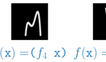
Drawing	Spec	Program	Compression factor
	`Line(2,15, 4,15)` `Line(4,9, 4,13)` `Line(3,11, 3,14)` `Line(2,13, 2,15)` `Line(3,14, 6,14)` `Line(4,13, 8,13)`	`for(i<3)` ` line(i,-1*i+6,` `  2*i+2,-1*i+6)` ` line(i,-2*i+4,i,-1*i+6)`	$\frac{6}{3} = 2\text{x}$
	`Line(5,13,2,10,arrow)` `Circle(5,9)` `Circle(8,5)`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

NEW: Wake-Sleep DSL learning with DreamCoder

- **Wake:** Synthesize programs from DSL + neural net ('recognition model') to guide search
- **Sleep-G:** Grow DSL, using grammar induction algorithms
- **Sleep-R:** Train neural net on programs found during waking ('experience replay') and samples from DSL ('dreaming')

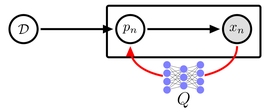


DreamCoder outputs for three different task domains

	List Functions	Text Editing	Symbolic Regression
Programs & Tasks	`[7 2 3]→[7 3]` `[1 2 3 4]→[3 4]` `[4 3 2 1]→[4 3]` $f(\ell) = (f_1\ \ell\ (\lambda\ (x)\ (> x\ 2)))$ `[7 3]→False` `[3]→False` `[9 0 0]→True` `[0]→True` `[0 7 3]→True` $f(\ell) = (f_3\ \ell\ 0)$ `[2 7 8 1]→8` `[3 19 14]→19` $f(\ell) = (f_2\ \ell)$	`+106 769-438→106.769.438` `+83 973-831→83.973.831` $f(s) = (f_0$ `"."` `"-"` $(f_0$ `"."` `" "` `(cdr s)))` Temple Anna H →TAH Lara Gregori→LG $f(s) = (f_2\ s)$	 $f(x) = (f_1\ x)$  $f(x) = (f_6\ x)$  $f(x) = (f_4\ x)$  $f(x) = (f_5\ x)$
DSL	$f_1(\ell,p) = (\text{foldr}\ \ell\ \text{nil}\ (\lambda\ (x\ a)\ (\text{if}\ (p\ x)\ (\text{cons}\ x\ a)\ a)))$ (f_1: *Higher-order filter function*) $f_2(\ell) = (\text{foldr}\ \ell\ 0\ (\lambda\ (x\ a)\ (\text{if}\ (> a\ x)\ x)))$ (f_2: *Maximum element in list ℓ*) $f_3(\ell,k) = (\text{foldr}\ \ell\ (\text{is-nil}\ \ell)\ (\lambda\ (x\ a)\ (\text{if}\ a\ a\ (= k\ x))))$ (f_3: *Whether ℓ contains* `k`)	$f_0(s,a,b) = (\text{map}\ (\lambda\ (x)\ (\text{if}\ (= x\ a)\ b\ x))\ s)$ (f_0: *substitutes characters*) $f_1(s,c) = (\text{foldr}\ s\ s\ (\lambda\ (x\ a)\ (\text{cdr}\ (\text{if}\ (= c\ x)\ s\ a))))$ (f_1: *Drop characters from* `s` *until* `c` *reached*) $f_2(s) = (\text{unfold}\ s\ \text{is-nil}\ \text{car}\ (\lambda\ (z)\ (f_1\ z\ \text{" "})))$ (f_2: *Abbreviates words*)	$f_0(x) = (+\ x\ \text{real})$ $f_1(x) = (f_0\ (*\ \text{real}\ x))$ $f_2(x) = (f_1\ (*\ x\ (f_0\ x)))$ $f_3(x) = (f_0\ (*\ x\ (f_2\ x)))$ $f_4(x) = (f_0\ (*\ x\ (f_3\ x)))$ (f_4: *4th order polynomial*) $f_5(x) = (/\ \text{real}\ x)$ $f_6(x) = (f_5\ (f_0\ x))$ (f_6: *rational function*)

Tasks from three domains we apply our algorithm to, each followed by the programs DreamCoder discovers for them. Bottom: Several examples from learned DSL. Notice that learned DSL primitives can call each other, and that DreamCoder rediscovers higher-order functions like filter (f_1 under List Functions)

Bayesian framing



$$\underbrace{p_n^* = \arg\max_{p_n} \mathbb{P}[x_n|p_n]\mathbb{P}[p_n|\mathcal{D}^*]}_{\text{Wake}}$$

$$\underbrace{\mathcal{D}^* = \arg\max_{\mathcal{D}} \mathbb{P}[\mathcal{D}] \prod_n \sum_{p_n} \mathbb{P}[x_n|p_n]\mathbb{P}[p_n|\mathcal{D}]}_{\text{Sleep-G}}$$

$$\underbrace{Q^* = \arg\min_{Q} \text{KL}\left(\mathbb{P}[p|x,\mathcal{D}]||Q(p|x)\right)}_{\text{Sleep-R}}$$

x_n: program synthesis problem. p_n: (latent) program solving x_n. $Q(p|x)$: (learned) neural recognition model. $\mathcal{D}$: (latent) DSL.