Program Synthesis

Machine Learning for Code

Giovanni De Toni
giovanni.detoni@unitn.it
@giovanni_detoni

Department of Information Engineering and Computer Science (DISI)
University of Trento, Italy
15th December 2021

The initial Idea
Summer Institute of Symbolic Logic, Cornell University (1957)

Church, Alonzo. "**Application of recursive arithmetic to the problem of circuit synthesis.**" *Journal of Symbolic Logic* 28.4 (1963).

-3-

APPLICATION OF RECURSIVE ARITHMETIC TO THE PROBLEM OF CIRCUIT SYNTHESIS

Alonzo Church

RESTRICTED RECURSIVE ARITHMETIC

Primitive symbols are individual (i.e., numerical) variables x, y, z, t,···, singulary functional constants i_1, i_2,···,i_μ, the individual constant 0, the accent ' as a notation for <u>successor</u> (of a number), the notation ( ) for application of a singulary function to its argument, connectives of the propositional calculus, and brackets [ ].

Axioms are all tautologous wffs. Rules are modus ponens; substitution for individual variables; mathematical induction,

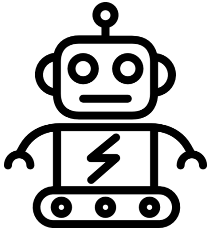
from $P \supset S^a_{a'}P|$ and $S^a_0 P|$ to infer P;

and any one of several alternative recursion schemata or sets of

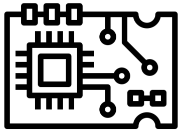
The initial Idea
Summer Institute of Symbolic Logic, Cornell University (1957)

Formal Specifications

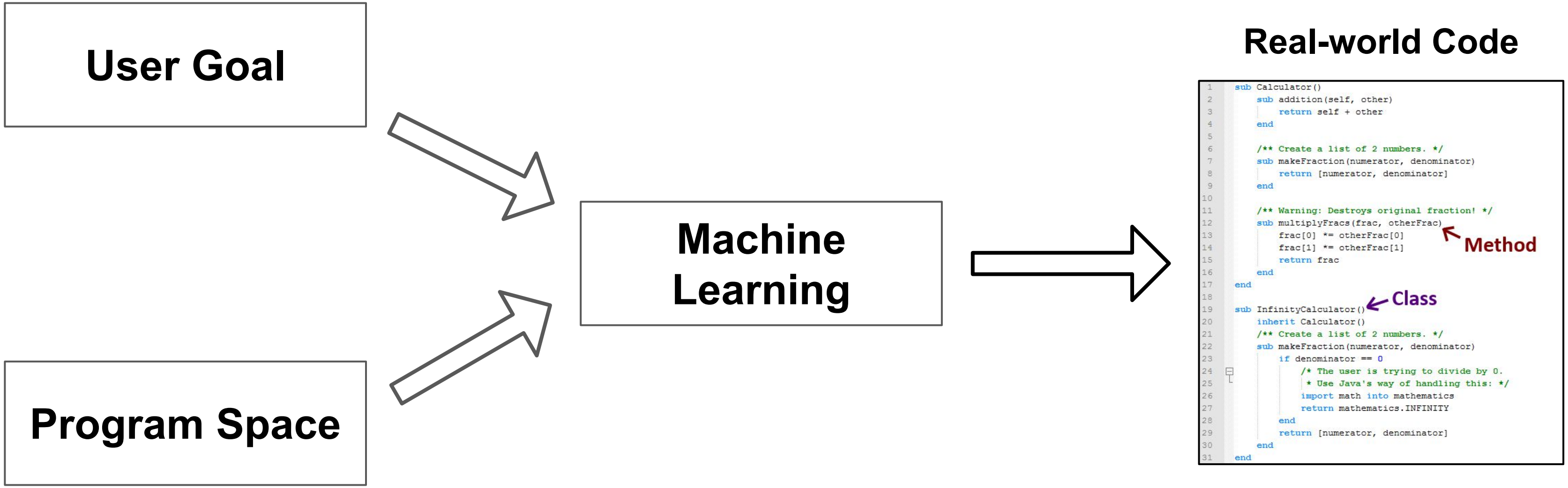
Given:	$A[0...n]$	such that	$\forall\, i \in \{0, n\} \;\; A[i] \in \mathbb{R}$
Generate:	$B[0...n]$	such that	$\forall\, i, j \in \{0, n\} : \; i < j \quad B[i] < B[j]$



Circuit



Modern Program Synthesis



Modern Program Synthesis

Input/Output Examples
Natural Language Specifications

User Goal

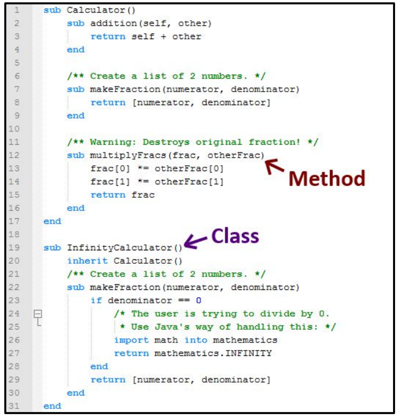
Domain Specific Language (DSL)
C++, Python, Java

Program Space

Search-based Methods
Neural Program Synthesis
Inductive programming
...

Machine Learning

Real-world Code



Applications: Code Generation

Natural Language Specifications

User Goal

C++, Python, etc.

Program Space

GitHub Copilot

Real-world Code

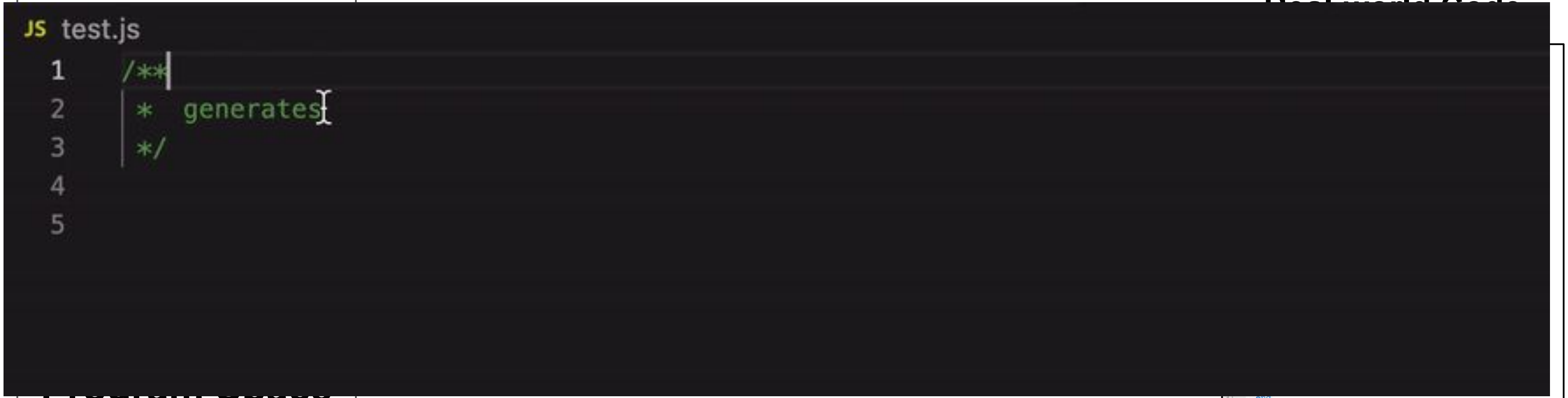
```
sub Calculator()
    sub addition(self, other)
        return self + other
    end

    /** Create a list of 2 numbers. */
    sub makeFraction(numerator, denominator)
        return [numerator, denominator]
    end

    /** Warning: Destroys original fraction! */
    sub multiplyFracs(frac, otherFrac)
        frac[0] *= otherFrac[0]
        frac[1] *= otherFrac[1]
        return frac
    end
end

sub InfinityCalculator()
    inherit Calculator()
    /** Create a list of 2 numbers. */
    sub makeFraction(numerator, denominator)
        if denominator == 0
            /* The user is trying to divide by 0.
             * Use Java's way of handling this: */
            import math into mathematics
            return mathematics.INFINITY
        end
        return [numerator, denominator]
    end
end
```

Method

Class

Applications: Code Generation

Natural Language Specifications



Applications: Data Management

Input/Output Examples

User Goal

Custom DSL

Program Space

Flash Fill

Excel Table

	A	B	C
1	Name and ID	First name and last name	ID #
2	Thomas, Rhonda 82132	Rhonda Thomas	
3	Emmett, Keara 34231	Keara Emmett	
4	Vogel, James 32493	James Vogel	
5	Jelen, Bill 23911	Bill Jelen	
6	Miller, Sylvia 78356	Sylvia Miller	
7	Lambert, Bobby 25900	Bobby Lambert	
8	Sweet, Julie 65477	Julie Sweet	
9	Williams, Don 43920	Don Williams	
10	Spake, Deborah 33488	Deborah Spake	

Applications: many more...

- **Smart auto-complete for IDEs** (Hindle et al., 2012, Bhoopchand et al., 2016)
- **Deobfuscating Android code** (Bichsel et al., 2016)
- **Automatic Bug identification** (Goues et al., 2019)
- **Code summarization** (Zügner et. al, 2021)
- **...**

Overview

- **Program Induction (Program by Example)**
- **Neural-Guided Program Synthesis**
- **Learning Program Representations**
- **Future Challenges**

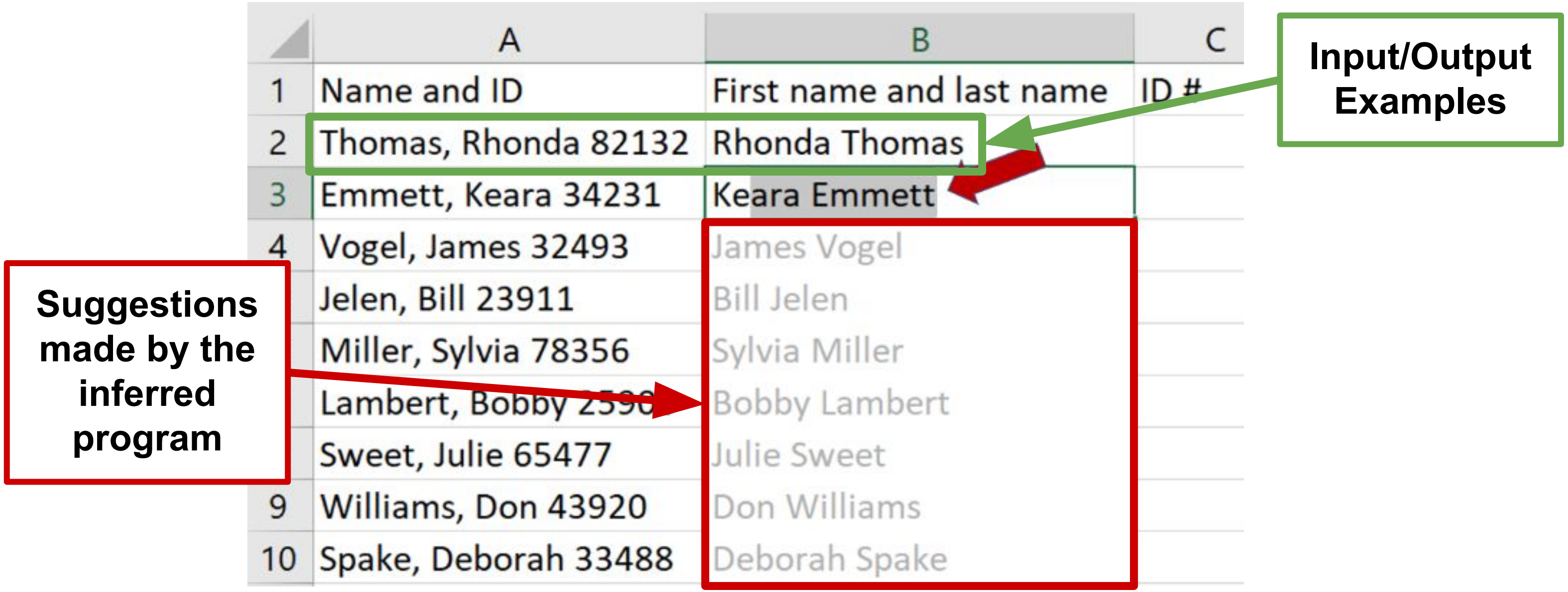
Overview

- **Program Induction (Program by Example)**
- **Neural-Guided Program Synthesis**
- **Learning Program Representations**
- **Future Challenges**

FlashFill (Gulwani, 2011)

	A	B	C
1	Name and ID	First name and last name	ID #
2	Thomas, Rhonda 82132	Rhonda Thomas	
3	Emmett, Keara 34231	Keara Emmett	
4	Vogel, James 32493	James Vogel	
5	Jelen, Bill 23911	Bill Jelen	
6	Miller, Sylvia 78356	Sylvia Miller	
7	Lambert, Bobby 25900	Bobby Lambert	
8	Sweet, Julie 65477	Julie Sweet	
9	Williams, Don 43920	Don Williams	
10	Spake, Deborah 33488	Deborah Spake	

FlashFill (Gulwani, 2011)



	A	B	C
1	Name and ID	First name and last name	ID #
2	Thomas, Rhonda 82132	Rhonda Thomas	
3	Emmett, Keara 34231	Keara Emmett	
4	Vogel, James 32493	James Vogel	
	Jelen, Bill 23911	Bill Jelen	
	Miller, Sylvia 78356	Sylvia Miller	
	Lambert, Bobby 2590	Bobby Lambert	
	Sweet, Julie 65477	Julie Sweet	
9	Williams, Don 43920	Don Williams	
10	Spake, Deborah 33488	Deborah Spake	

FlashFill (Gulwani, 2011)

$$
\begin{array}{rcl}
\text{String expr } P & := & \texttt{Switch}((\mathsf{b}_1, \mathsf{e}_1), \cdot\cdot, (\mathsf{b}_n, \mathsf{e}_n)) \\
\text{Bool b} & := & \mathsf{d}_1 \vee \cdot\cdot \vee \mathsf{d}_n \\
\text{Conjunct d} & := & \pi_1 \wedge \cdot\cdot \wedge \pi_n \\
\text{Predicate } \pi & := & \texttt{Match}(v_i, \mathsf{r}, k) \mid \neg\, \texttt{Match}(v_i, \mathsf{r}, k) \\
\text{Trace expr e} & := & \texttt{Concatenate}(\mathsf{f}_1, \cdot\cdot, \mathsf{f}_n) \\
\text{Atomic expr f} & := & \texttt{SubStr}(v_i, \mathsf{p}_1, \mathsf{p}_2) \\
 & \mid & \texttt{ConstStr}(s) \\
 & \mid & \texttt{Loop}(\lambda w : \mathsf{e}) \\
\text{Position p} & := & \texttt{CPos}(k) \mid \texttt{Pos}(\mathsf{r}_1, \mathsf{r}_2, \mathsf{c}) \\
\text{Integer expr c} & := & k \mid k_1 w + k_2 \\
\text{Regular Expression r} & := & \texttt{TokenSeq}(\mathsf{T}_1, \cdot\cdot, \mathsf{T}_m) \\
\text{Token T} & := & C+ \mid [\neg C]+ \\
 & \mid & \text{SpecialToken}
\end{array}
$$

Figure 1. Syntax of String Expressions P. v_i refers to a free string variable, while w refers to a bound integer variable. k denotes an integer constant and s denotes a string constant.

$$
\begin{array}{rcl}
[\![\texttt{Switch}((\mathsf{b}_1, \mathsf{e}_1), \cdot\cdot, (\mathsf{b}_n, \mathsf{e}_n))]\!]\,\sigma & = & \text{if } ([\![\mathsf{b}_1]\!]\sigma) \text{ then } [\![\mathsf{e}_1]\!]\sigma \\
 & & \vdots \\
 & & \text{else if } ([\![\mathsf{b}_n]\!]\sigma) \text{ then } [\![\mathsf{e}_n]\!]\sigma \\
 & & \text{else } \bot \\
[\![\mathsf{d}_1 \vee \ldots \vee \mathsf{d}_n]\!]\,\sigma & = & [\![\mathsf{d}_1]\!]\,\sigma \vee \ldots \vee [\![\mathsf{d}_n]\!]\,\sigma \\
[\![\pi_1 \wedge \ldots \wedge \pi_n]\!]\,\sigma & = & [\![\pi_1]\!]\,\sigma \wedge \ldots \wedge [\![\pi_n]\!]\,\sigma \\
[\![\texttt{Match}(v_i, \mathsf{r}, k)]\!]\,\sigma & = & \texttt{Match}(\sigma(v_i), \mathsf{r}, k) \\
[\![\texttt{Concatenate}(\mathsf{f}_1, \cdot\cdot, \mathsf{f}_n)]\!]\,\sigma & = & \texttt{Concatenate}([\![\mathsf{f}_1]\!]\,\sigma, \cdot\cdot, [\![\mathsf{f}_n]\!]\,\sigma) \\
[\![\texttt{Loop}(\lambda w : \mathsf{e})]\!]\,\sigma & = & \texttt{LoopR}(\lambda w : \mathsf{e}, 1, \sigma) \\
\texttt{LoopR}(\lambda w : \mathsf{e}, k, \sigma) & = & \text{let } t := [\![\mathsf{e}[k/w]]\!]\,\sigma \text{ in} \\
 & & \text{if } (t = \bot) \text{ then } \epsilon \text{ else} \\
 & & \texttt{Concatenate}(t, \texttt{LoopR}(\lambda w : \mathsf{e}, k{+}1, \sigma)) \\
[\![\texttt{SubStr}(v_i, \mathsf{p}_1, \mathsf{p}_2)]\!]\,\sigma & = & s[[\![\mathsf{p}_1]\!]\,s : [\![\mathsf{p}_2]\!]\,s], \text{ where } s = \sigma(v_i). \\
[\![\texttt{ConstStr}(s)]\!]\,\sigma & = & s \\
[\![\texttt{CPos}(k)]\!]\,s & = & \begin{cases} k & \text{if } k \geq 0 \\ \texttt{Length}(s) + k & \text{otherwise} \end{cases} \\
[\![\texttt{Pos}(\mathsf{r}_1, \mathsf{r}_2, \mathsf{c})]\!]\,s & = & t \text{ such that } \exists\, t_1, t_2 \text{ s.t. } 0 \leq t_1 < t \leq t_2, \\
 & & s[t_1 : t\text{-}1] \text{ matches } \mathsf{r}_1,\ s[t : t_2] \text{ matches } \mathsf{r}_2, \\
 & & \text{and } t \text{ is the } \mathsf{c}^{th} \text{ such position (in increasing/} \\
 & & \text{decreasing order if c is positive/negative.}
\end{array}
$$

Figure 2. Semantics of String Expressions P.

Excerpt of the Domain Specific Language (DSL) for FlashFill

FlashFill (Gulwani, 2011)

$$
\begin{array}{rcl}
\text{String expr } P & := & \texttt{Switch}((b_1, e_1), \cdot\cdot, (b_n, e_n)) \\
\text{Bool b} & := & d_1 \vee \cdot\cdot \vee d_n \\
\text{Conjunct d} & := & \pi_1 \wedge \cdot\cdot \wedge \pi_n \\
\text{Predicate } \pi & := & \texttt{Match}(v_i, r, k) \mid \neg\, \texttt{Match}(v_i, r, k) \\
\text{Trace expr e} & := & \texttt{Concatenate}(f_1, \cdot\cdot, f_n) \\
\text{Atomic expr f} & := & \texttt{SubStr}(v_i, p_1, p_2) \\
 & \mid & \texttt{ConstStr}(s) \\
 & \mid & \texttt{Loop}(\lambda w : e) \\
\text{Position p} & := & \texttt{CPos}(k) \mid \texttt{Pos}(r_1, r_2, c) \\
\text{Integer expr c} & := & k \mid k_1 w + k_2 \\
\text{Regular Expression r} & := & \texttt{TokenSeq}(T_1, \cdot\cdot, T_m) \\
\text{Token T} & := & C+ \mid [\neg C]+ \\
 & \mid & \text{SpecialToken}
\end{array}
$$

Figure 1. Syntax of String Expressions P. v_i refers to a free string variable, while w refers to a bound integer variable. k denotes an integer constant and s denotes a string constant.

Excerpt of the Domain Specific Language (DSL) for FlashFill

$$
\begin{array}{rcl}
[\![\texttt{Switch}((b_1, e_1), \cdot\cdot, (b_n, e_n))]\!]\, \sigma & = & \text{if } ([\![b_1]\!]\sigma) \text{ then } [\![e_1]\!]\sigma \\
 & & \vdots \\
 & & \text{else if } ([\![b_n]\!]\sigma) \text{ then } [\![e_n]\!]\sigma \\
 & & \text{else } \bot \\
[\![d_1 \vee \ldots \vee d_n]\!]\, \sigma & = & [\![d_1]\!]\, \sigma \vee \ldots \vee [\![d_n]\!]\, \sigma \\
[\![\pi_1 \wedge \ldots \wedge \pi_n]\!]\, \sigma & = & [\![\pi_1]\!]\, \sigma \wedge \ldots \wedge [\![\pi_n]\!]\, \sigma \\
[\![\texttt{Match}(v_i, r, k)]\!]\, \sigma & = & \texttt{Match}(\sigma(v_i), r, k)
\end{array}
$$

$$
[\![\texttt{Concatenate}(f_1, \cdot\cdot, f_n)]\!]\, \sigma = \texttt{Concatenate}([\![f_1]\!]\, \sigma, \cdot\cdot, [\![f_n]\!]\, \sigma)
$$

$$
\begin{array}{rcl}
[\![\texttt{Loop}(\lambda w : e)]\!]\, \sigma & = & \texttt{LoopR}(\lambda w : e, 1, \sigma) \\
\texttt{LoopR}(\lambda w : e, k, \sigma) & = & \text{let } t := [\![e[k/w]]\!]\, \sigma \text{ in} \\
 & & \text{if } (t = \bot) \text{ then } \epsilon \text{ else} \\
 & & \texttt{Concatenate}(t, \texttt{LoopR}(\lambda w : e, k{+}1, \sigma))
\end{array}
$$

$$
\begin{array}{rcl}
[\![\texttt{SubStr}(v_i, p_1, p_2)]\!]\, \sigma & = & s[[\![p_1]\!]\, s : [\![p_2]\!]\, s], \text{ where } s = \sigma(v_i). \\
[\![\texttt{ConstStr}(s)]\!]\, \sigma & = & s \\
[\![\texttt{CPos}(k)]\!]\, s & = & \begin{cases} k & \text{if } k \geq 0 \\ \texttt{Length}(s) + k & \text{otherwise} \end{cases}
\end{array}
$$

$$
\begin{array}{rcl}
[\![\texttt{Pos}(r_1, r_2, c)]\!]\, s & = & t \text{ such that } \exists\, t_1, t_2 \text{ s.t. } 0 \leq t_1 < t \leq t_2, \\
 & & s[t_1 : t\text{-}1] \text{ matches } r_1, \;\; s[t : t_2] \text{ matches } r_2, \\
 & & \text{and } t \text{ is the } c^{th} \text{ such position (in increasing/} \\
 & & \text{decreasing order if c is positive/negative.}
\end{array}
$$

Figure 2. Semantics of String Expressions P.

FlashFill (Gulwani, 2011)

```
GenerateStringProgram(S: Set of (σ, s) pairs)
T := ∅;
foreach (σ, s) ∈ S
    T := T ∪ ({σ}, GenerateStr(σ, s));
T := GeneratePartition(T);
σ̃′ := {σ | (σ, s) ∈ S};
foreach (σ̃, ẽ) ∈ T:
  let B[σ̃] := GenerateBoolClassifier(σ̃, σ̃′-σ̃)
Let (σ̃1, ẽ1), ..., (σ̃k, ẽk) be the k elements in
            T in increasing order of Size(ẽ).
return Switch((B[σ̃1], ẽ1), ..., (B[σ̃k], ẽk));
```

FlashFill (Gulwani, 2011)

```
GenerateStringProgram(S: Set of (σ, s) pairs)
T := ∅;
foreach (σ, s) ∈ S
    T := T ∪ ({σ}, GenerateStr(σ, s));
T := GeneratePartition(T);
σ̃' := {σ | (σ, s) ∈ S};
foreach (σ̃, ẽ) ∈ T:
    let B[σ̃] := GenerateBoolClassifier(σ̃, σ̃'-σ̃)
Let (σ̃1, ẽ1), ..., (σ̃k, ẽk) be the k elements in
        T in increasing order of Size(ẽ).
return Switch((B[σ̃1], ẽ1), ..., (B[σ̃k], ẽk));
```

Given each input/output pair (***σ*, *s***), generate all the possible **program expressions** that matches the input ***σ*** to the output **s.**

FlashFill (Gulwani, 2011)

```
GenerateStringProgram(S: Set of (σ, s) pairs)
T := ∅;
foreach (σ, s) ∈ S
    T := T ∪ ({σ}, GenerateStr(σ, s));
T := GeneratePartition(T);
σ̃' := {σ | (σ, s) ∈ S};
foreach (σ̃, ẽ) ∈ T:
   let B[σ̃] := GenerateBoolClassifier(σ̃, σ̃'-σ̃)
Let (σ̃1, ẽ1), ..., (σ̃k, ẽk) be the k elements in
            T in increasing order of Size(ẽ).
return Switch((B[σ̃1], ẽ1), ..., (B[σ̃k], ẽk));
```

Partition the examples such that inputs in the **same partition** are handled by the **same program** in the `Switch` construct.

FlashFill (Gulwani, 2011)

```
GenerateStringProgram(S: Set of (σ, s) pairs)
T := ∅;
foreach (σ, s) ∈ S
    T := T ∪ ({σ}, GenerateStr(σ, s));
T := GeneratePartition(T);
σ̃′ := {σ | (σ, s) ∈ S};
foreach (σ̃, ẽ) ∈ T:
  let B[σ̃] := GenerateBoolClassifier(σ̃, σ̃′-σ̃)
Let (σ̃1, ẽ1), . . . , (σ̃k, ẽk) be the k elements in
          T in increasing order of Size(ẽ).
return Switch((B[σ̃1], ẽ1), . . . , (B[σ̃k], ẽk));
```

We construct a **boolean classification scheme** to **match each input to a partition**, hence, to a specific trace (program).

FlashFill (Gulwani, 2011)

GenerateStringProgram(S: Set of (σ, s) pairs)

$T := \emptyset$;
foreach $(\sigma, s) \in S$
    $T := T \cup (\{\sigma\}, \texttt{GenerateStr}(\sigma, s))$;
$T :=$ GeneratePartition(T);
$\tilde{\sigma}' := \{\sigma \mid (\sigma, s) \in S\}$;
foreach $(\tilde{\sigma}, \tilde{e}) \in T$:
    let $B[\tilde{\sigma}] :=$ GenerateBoolClassifier($\tilde{\sigma}, \tilde{\sigma}' - \tilde{\sigma}$)
Let $(\tilde{\sigma}_1, \tilde{e}_1), \ldots, (\tilde{\sigma}_k, \tilde{e}_k)$ be the k elements in T in increasing order of Size($\tilde{e}$).
return Switch($(B[\tilde{\sigma}_1], \tilde{e}_1), \ldots, (B[\tilde{\sigma}_k], \tilde{e}_k)$);

We return the complete expression, that match each new input to its correspondent program

FlashFill (Gulwani, 2011)

EXAMPLE 10 (Phone Numbers). *The goal here is to parse phone numbers that occur in multiple formats and transform them into a uniform format, adding a default area code of "425" if the area code is missing. This example was provided by the product team.*

Input v_1	*Output*
323-708-7700	*323-708-7700*
(425)-706-7709	*425-706-7709*
510.220.5586	*510-220-5586*
235 7654	*425-235-7654*
745-8139	*425-745-8139*

String Program:
$\mathtt{Switch}((b_1, e_1), (b_2, e_2))$, *where*

$$
\begin{aligned}
b_1 &\equiv \mathtt{Match}(v_1, NumTok, 3),\ b_2 \equiv \neg\mathtt{Match}(v_1, NumTok, 3),\\
e_1 &\equiv \mathtt{Concatenate}(\mathtt{SubStr2}(v_1, NumTok, 1), \mathtt{ConstStr}(\text{"-"}),\\
&\qquad \mathtt{SubStr2}(v_1, NumTok, 2), \mathtt{ConstStr}(\text{"-"}),\\
&\qquad \mathtt{SubStr2}(v_1, NumTok, 3))\\
e_2 &\equiv \mathtt{Concatenate}(\mathtt{ConstStr}(\text{"425-"}), \mathtt{SubStr2}(v_1, NumTok, 1),\\
&\qquad \mathtt{ConstStr}(\text{"-"}), \mathtt{SubStr2}(v_1, NumTok, 2))
\end{aligned}
$$

FlashFill (Gulwani, 2011)

EXAMPLE 10 (Phone Numbers). *The goal here is to parse phone numbers that occur in multiple formats and transform them into a uniform format, adding a default area code of "425" if the area code is missing. This example was provided by the product team.*

Input v_1	*Output*
323-708-7700	323-708-7700
(425)-706-7709	425-706-7709
510.220.5586	510-220-5586
235 7654	425-235-7654
745-8139	425-745-8139

String Program:

$\texttt{Switch}((b_1, e_1), (b_2, e_2))$, *where*

$$b_1 \equiv \texttt{Match}(v_1, NumTok, 3),\ b_2 \equiv \neg\texttt{Match}(v_1, NumTok, 3),$$

$$e_1 \equiv \texttt{Concatenate}(\texttt{SubStr2}(v_1, NumTok, 1), \texttt{ConstStr}(\text{"-"}), \texttt{SubStr2}(v_1, NumTok, 2), \texttt{ConstStr}(\text{"-"}), \texttt{SubStr2}(v_1, NumTok, 3))$$

$$e_2 \equiv \texttt{Concatenate}(\texttt{ConstStr}(\text{"425-"}), \texttt{SubStr2}(v_1, NumTok, 1), \texttt{ConstStr}(\text{"-"}), \texttt{SubStr2}(v_1, NumTok, 2))$$

Program Induction (Program by Example)

- The generated program must satisfy **all** the examples provided
- **Conflicting** or **ambiguous** examples
  - How do we cope with that? Are there any strategies we can use to disambiguate?
- **Heuristics** needed to improve the results
- DSL language must be **expressive** enough

Overview

- **Program Induction (Program by Example)**
- Neural-Guided Program Synthesis
- Learning Program Representations
- Future Challenges

Overview

- Program Induction (Program by Example)
- **Neural-Guided Program Synthesis**
- Learning Program Representations
- Future Challenges

Neural-guided Program Synthesis

- **Exponential program space**
- Use a deep learning model to **guide** the program space search
- Deep learning deals with **ambiguous examples**
- Learn programs that can **better generalize**

Neural-guided Program Synthesis

- **Exponential program space**
- Use a deep learning model to **guide** the program space search
- Deep learning deals with **ambiguous examples**
- Learn programs that can **better generalize**

$$\pi : \mathbb{E} \to \mathbb{A} \qquad \pi(e) = \text{softmax}(f(e))$$

$$e_t \in \mathbb{E},\ a_{t+1} = \text{argmax}(\pi(e_t))$$

Neural-guided Program Synthesis

- **Exponential program space**
- Use a deep learning model to **guide** the program space search
- Deep learning deals with **ambiguous examples**
- Learn programs that can **better generalize**

Search Policy

$$\pi : \mathbb{E} \rightarrow \mathbb{A} \qquad \pi(e) = \mathrm{softmax}(f(e))$$

$$e_t \in \mathbb{E}, \; a_{t+1} = \mathrm{argmax}(\pi(e_t))$$

Neural-guided Program Synthesis

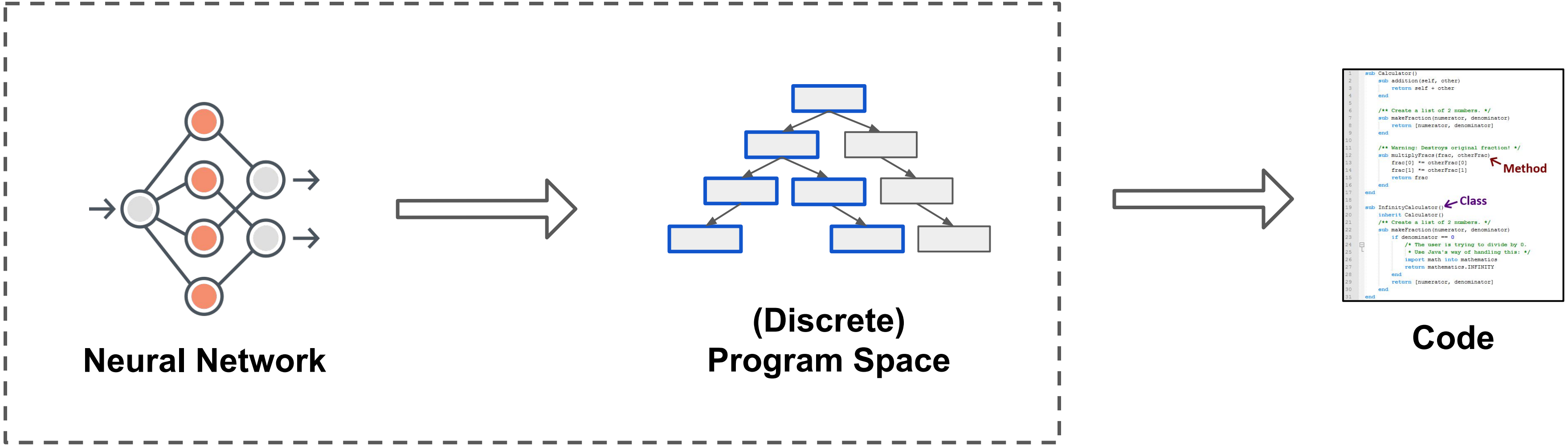
- **Exponential program space**
- Use a deep learning model to **guide** the program space search
- Deep learning deals with **ambiguous examples**
- Learn programs that can **better generalize**

$$\pi : \mathbb{E} \rightarrow \mathbb{A} \qquad \pi(e) = \text{softmax}(f(e))$$

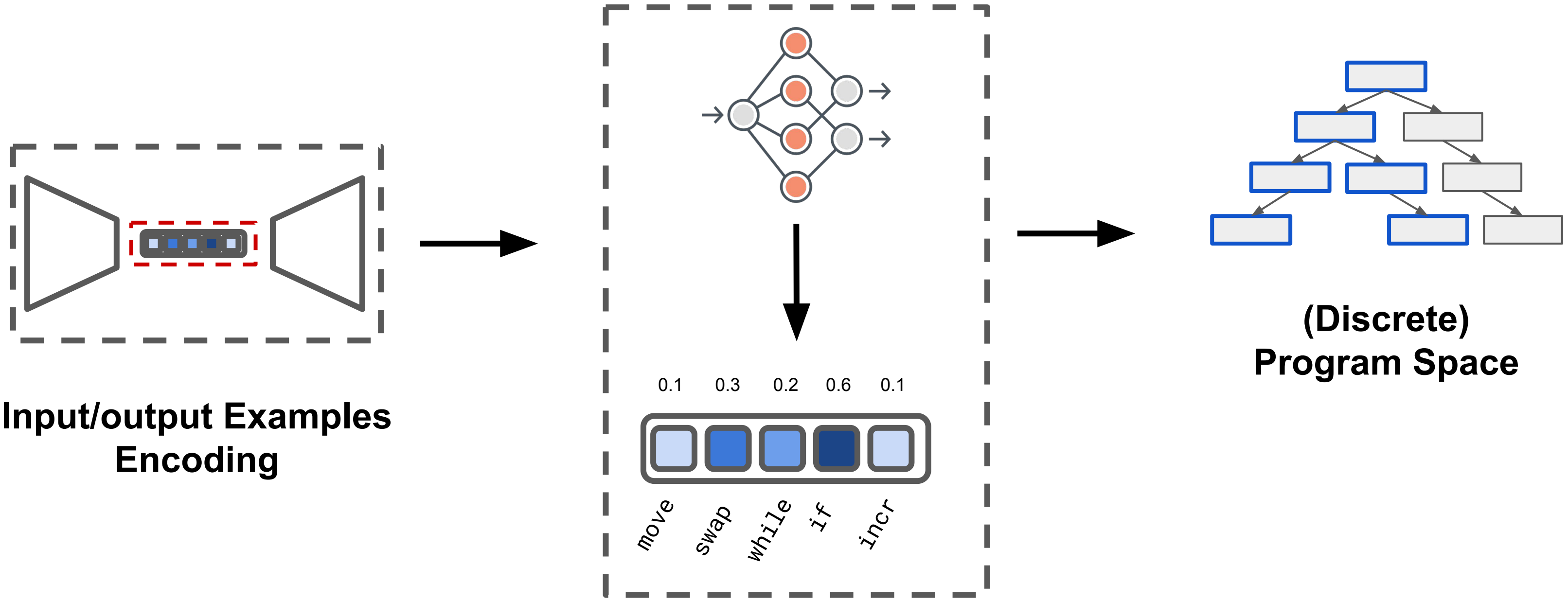
Choose the best next instruction

$$e_t \in \mathbb{E}, \ a_{t+1} = \text{argmax}(\pi(e_t))$$

Neural-guided Program Synthesis



DeepCoder (Balog et al., 2017)



Input/output Examples Encoding

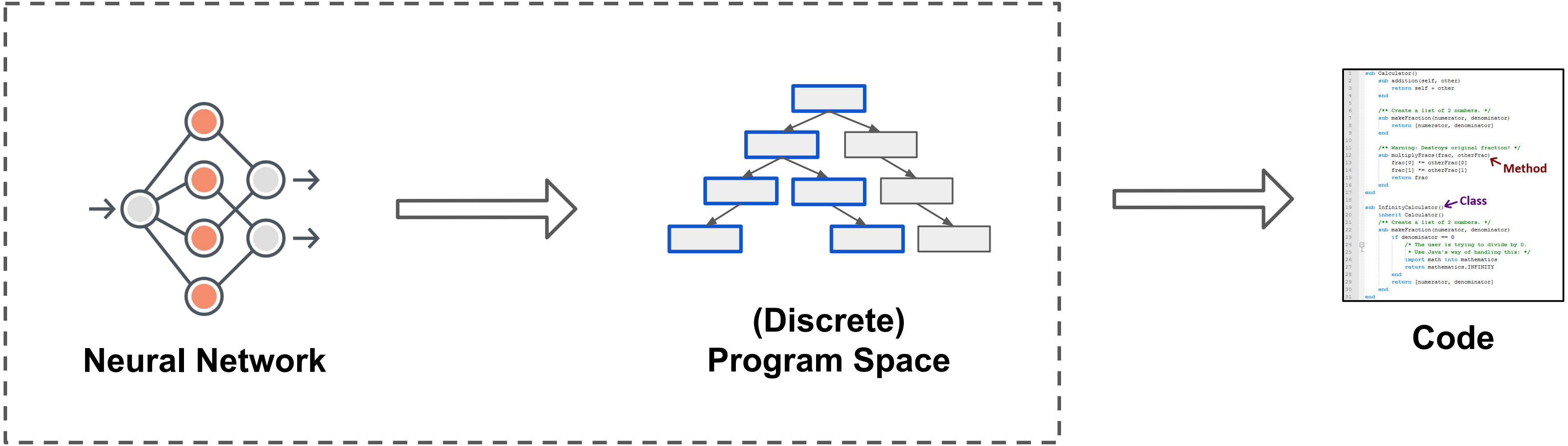
(Discrete) Program Space

DeepCoder (Balog et al., 2017)

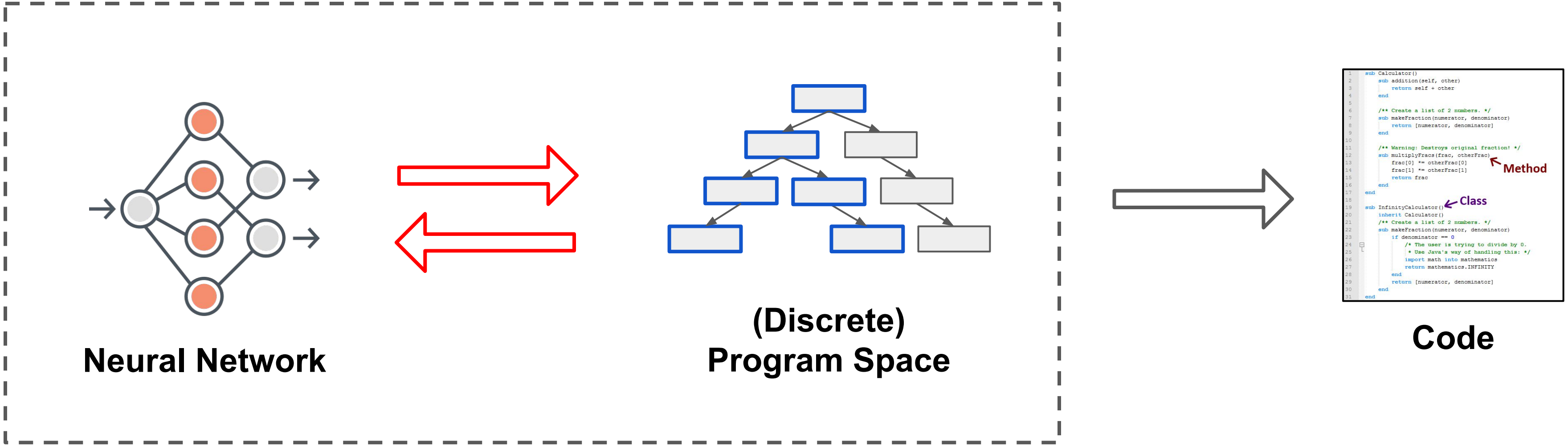
(+1)	(-1)	(*2)	(/2)	(*-1)	(**2)	(*3)	(/3)	(*4)	(/4)	(>0)	(>0)	(%2==1)	(%2==0)	HEAD	LAST	MAP	FILTER	SORT	REVERSE	TAKE	DROP	ACCESS	ZIPWITH	SCANL1	+	-	*	MIN	MAX	COUNT	MINIMUM	MAXIMUM	SUM
.0	.0	.1	.0	.0	.0	.0	.0	1.0	.0	.0	1.0	.0	.2	.0	.0	1.0	1.0	1.0	.7	.0	.1	.0	.4	.0	.0	.1	.0	.2	.1	.0	.0	.0	.0

Figure 2: Neural network predicts the probability of each function appearing in the source code.

Neural-guided Program Synthesis

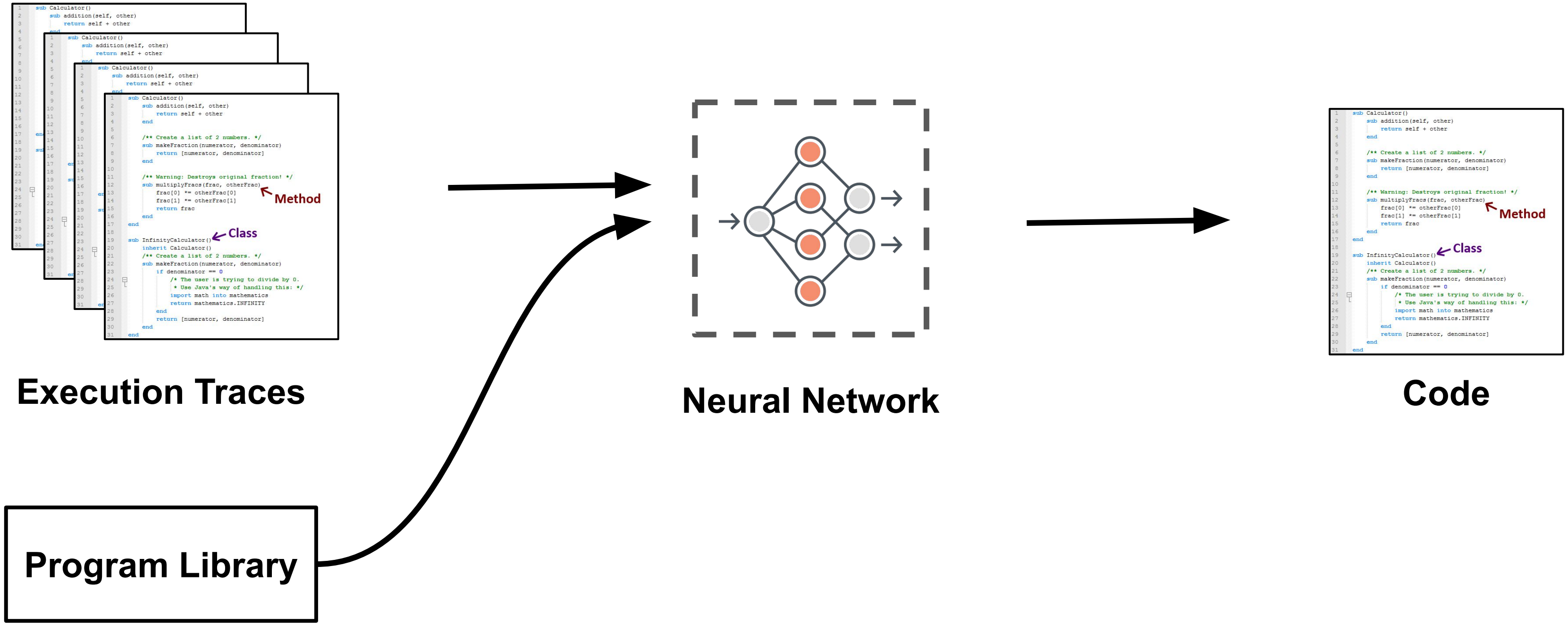


Neural-guided Program Synthesis

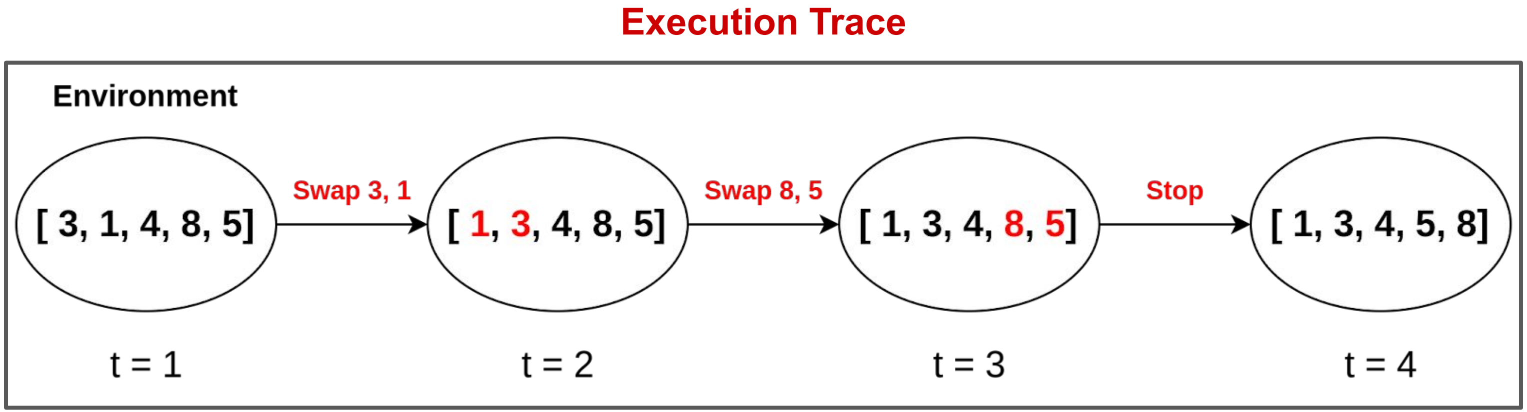


Neural Programmer-Interpreters

(Reed & de Freitas 2016)



Neural Programmer-Interpreters
(Reed & de Freitas 2016)



Neural Programmer-Interpreters
(Reed & de Freitas 2016)

Algorithm 1 Neural programming inference

Inputs: Environment observation e, program id i, arguments a, stop threshold α
function RUN(i, a)
    $h \leftarrow \mathbf{0}, r \leftarrow 0, p \leftarrow M^{\text{prog}}_{i,:}$ ▷ Init LSTM and return probability.
    while $r < \alpha$ **do**
        $s \leftarrow f_{enc}(e, a), h \leftarrow f_{lstm}(s, p, h)$ ▷ Feed-forward NPI one step.
        $r \leftarrow f_{end}(h), k \leftarrow f_{prog}(h), a_2 \leftarrow f_{arg}(h)$
        $i_2 \leftarrow \arg\max_{j=1..N} (M^{\text{key}}_{j,:})^T k$ ▷ Decide the next program to run.
        if $i ==$ ACT **then** $e \leftarrow f_{env}(e, p, a)$ ▷ Update the environment based on ACT.
        else RUN(i_2, a_2) ▷ Run subprogram i_2 with arguments a_2

Neural Programmer-Interpreters
(Reed & de Freitas 2016)

Algorithm 1 Neural programming inference

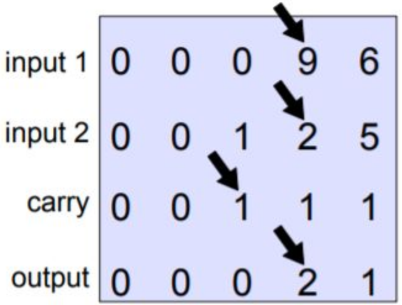
Inputs: Environment observation e, program id i, arguments a, stop threshold α
function RUN(i, a)
  $h \leftarrow \mathbf{0}, r \leftarrow 0, p \leftarrow M^{\mathrm{prog}}_{i,:}$ ▷ Init LSTM and return probability.
  while $r < \alpha$ **do**
    $s \leftarrow f_{enc}(e, a), h \leftarrow f_{lstm}(s, p, h)$ ▷ Feed-forward NPI one step.
    $r \leftarrow f_{end}(h), k \leftarrow f_{prog}(h), a_2 \leftarrow f_{arg}(h)$
    $i_2 \leftarrow \arg\max_{j=1..N} (M^{\mathrm{key}}_{j,:})^T k$ ▷ Decide the next program to run.
    if $i == \mathrm{ACT}$ **then** $e \leftarrow f_{env}(e, p, a)$ ▷ Update the environment based on ACT.
    else RUN(i_2, a_2) ▷ Run subprogram i_2 with arguments a_2

Neural Programmer-Interpreters
(Reed & de Freitas 2016)

Algorithm 1 Neural programming inference

Inputs: Environment observation e, program id i, arguments a, stop threshold α
function RUN(i, a)
    $h \leftarrow \mathbf{0}, r \leftarrow 0, p \leftarrow M^{\text{prog}}_{i,:}$ ▷ Init LSTM and return probability.
    while $r < \alpha$ **do**
        $s \leftarrow f_{enc}(e, a), h \leftarrow f_{lstm}(s, p, h)$ ▷ Feed-forward NPI one step.
        $r \leftarrow f_{end}(h), k \leftarrow f_{prog}(h), a_2 \leftarrow f_{arg}(h)$
        $i_2 \leftarrow \arg\max_{j=1..N} (M^{\text{key}}_{j,:})^T k$ ▷ Decide the next program to run.
        if $i == $ ACT **then** $e \leftarrow f_{env}(e, p, a)$ ▷ Update the environment based on ACT.
        else RUN(i_2, a_2) ▷ Run subprogram i_2 with arguments a_2

Neural Programmer-Interpreters
(Reed & de Freitas 2016)

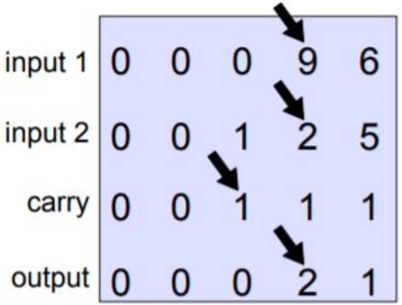


(a) Example scratch pad and pointers used for computing "96 + 125 = 221". Carry step is being implemented.

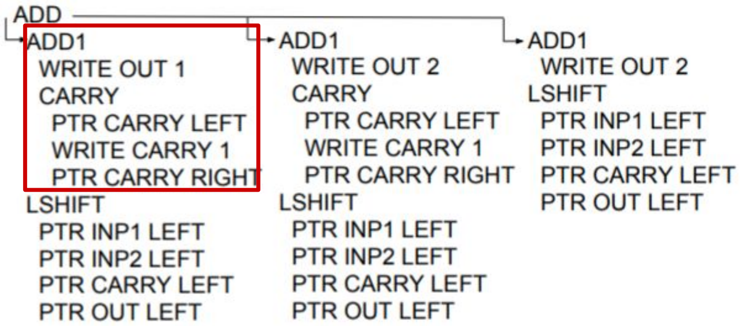
```
ADD
 ADD1                     ADD1                     ADD1
   WRITE OUT 1              WRITE OUT 2              WRITE OUT 2
   CARRY                    CARRY                  LSHIFT
     PTR CARRY LEFT           PTR CARRY LEFT         PTR INP1 LEFT
     WRITE CARRY 1            WRITE CARRY 1          PTR INP2 LEFT
     PTR CARRY RIGHT          PTR CARRY RIGHT        PTR CARRY LEFT
 LSHIFT                   LSHIFT                     PTR OUT LEFT
   PTR INP1 LEFT            PTR INP1 LEFT
   PTR INP2 LEFT            PTR INP2 LEFT
   PTR CARRY LEFT           PTR CARRY LEFT
   PTR OUT LEFT             PTR OUT LEFT
```

(b) Actual trace of addition program generated by our model on the problem shown to the left. Note that we substituted the ACT calls in the trace with more human-readable steps.

Neural Programmer-Interpreters
(Reed & de Freitas 2016)

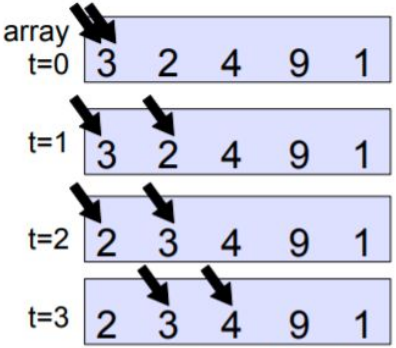


(a) Example scratch pad and pointers used for computing “96 + 125 = 221”. Carry step is being implemented.

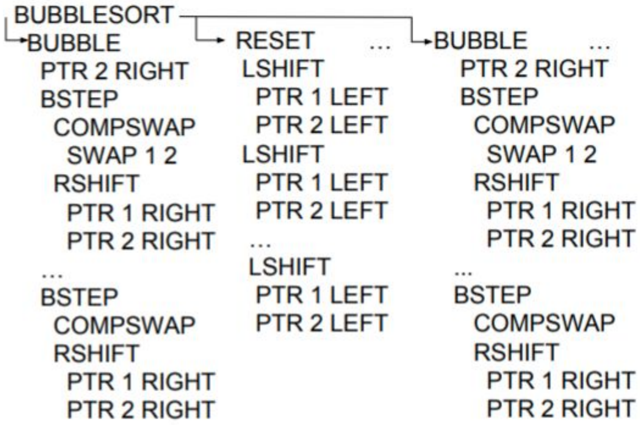


(b) Actual trace of addition program generated by our model on the problem shown to the left. Note that we substituted the ACT calls in the trace with more human-readable steps.

Neural Programmer-Interpreters
(Reed & de Freitas 2016)

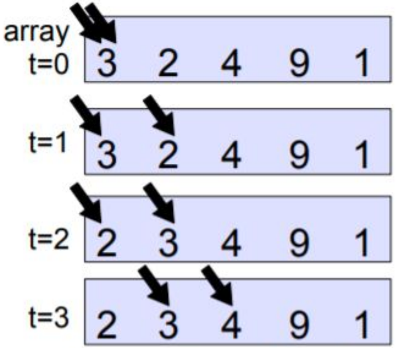


(a) Example scratch pad and pointers used for sorting. Several steps of the BUBBLE subprogram are shown.

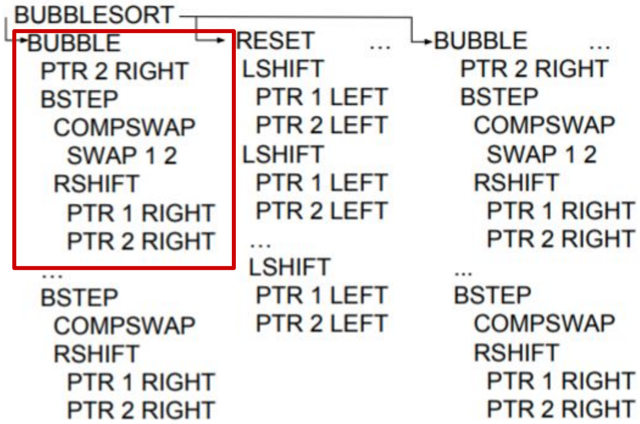


(b) Excerpt from the trace of the learned bubblesort program.

Neural Programmer-Interpreters
(Reed & de Freitas 2016)

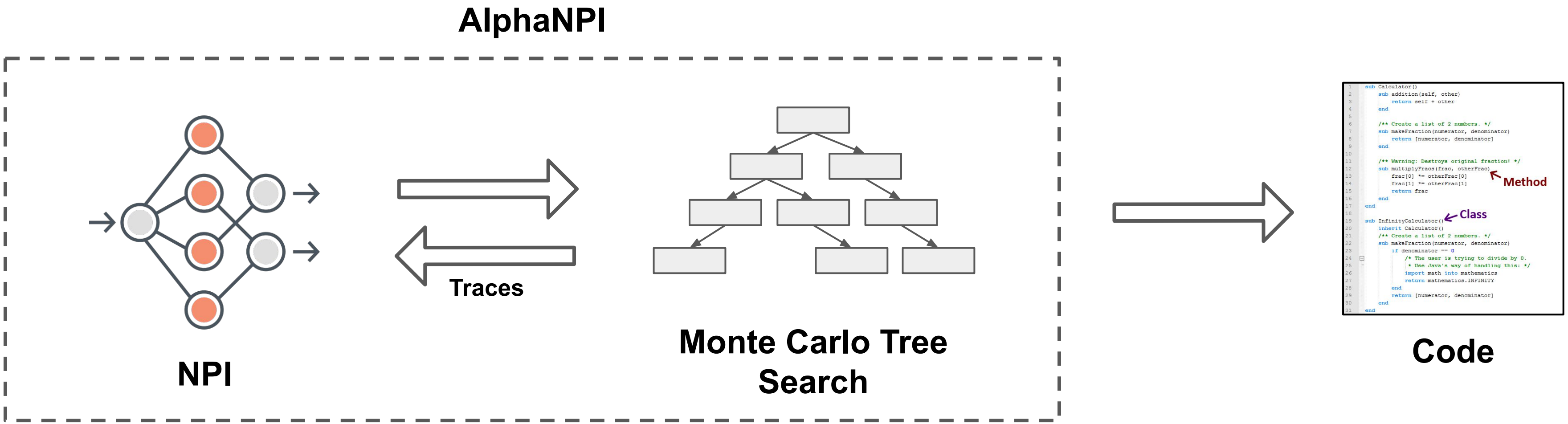


(a) Example scratch pad and pointers used for sorting. Several steps of the BUBBLE subprogram are shown.

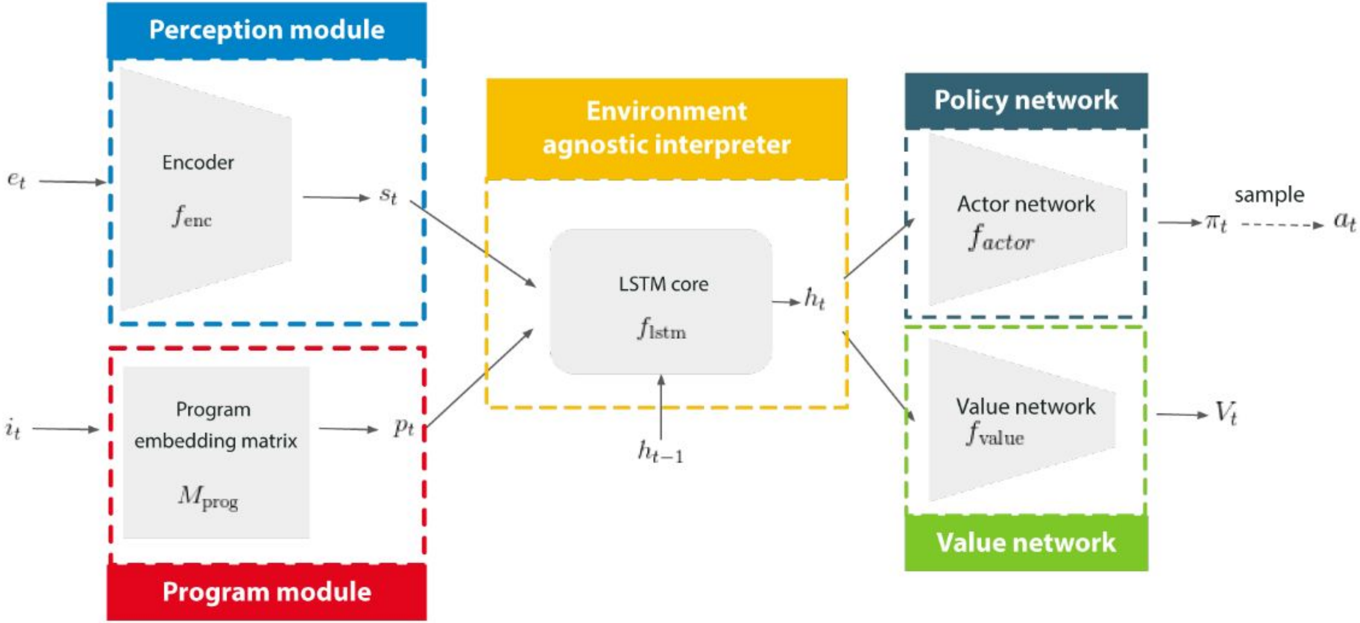


(b) Excerpt from the trace of the learned bubblesort program.

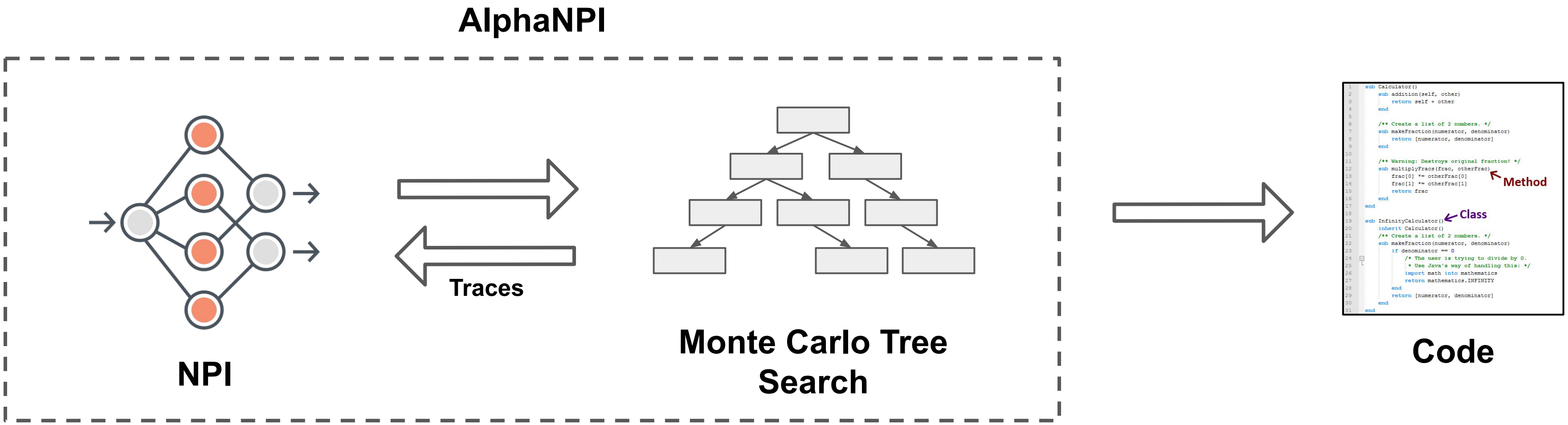
AlphaNPI (Pierrot et al., 2019)



AlphaNPI (Pierrot et al., 2019)

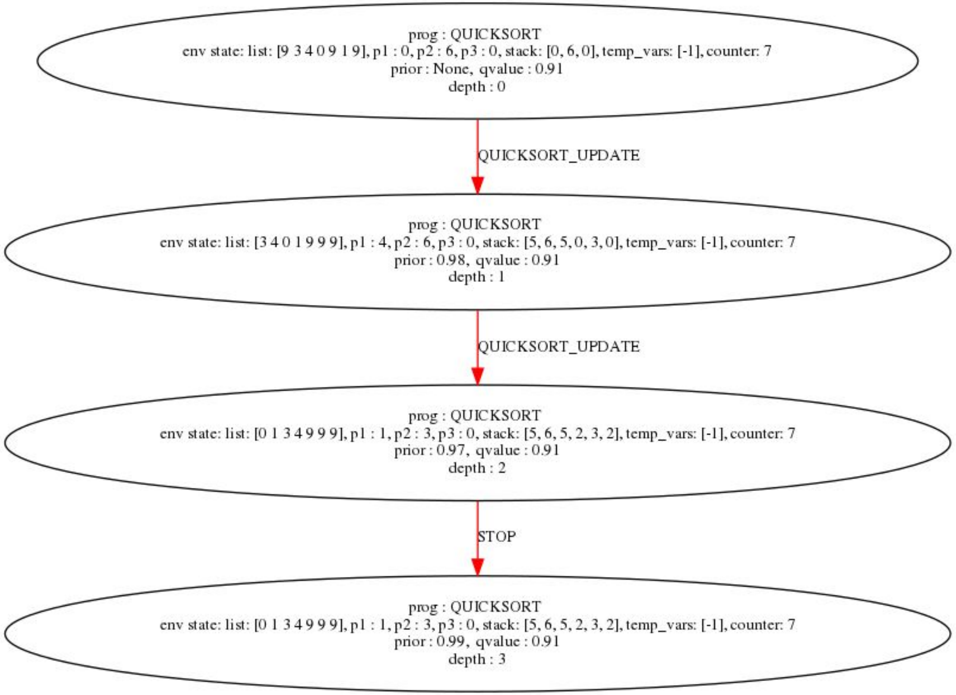


AlphaNPI (Pierrot et al., 2019)



$$l = \sum_{b \in \text{batches}} \underbrace{-(\pi^{mcts})^T \log(\pi)}_{\text{policy loss}} + \underbrace{(V - r)^2}_{\text{state loss}}$$

AlphaNPI (Pierrot et al., 2019)



The `quicksort` program (5)

```
PUSH
for 0 to n do
    QUICKSORT_UPDATE
end for
STOP
```

Overview

- Program Induction (Program by Example)
- **Neural-Guided Program Synthesis**
- Learning Program Representations
- Future Challenges

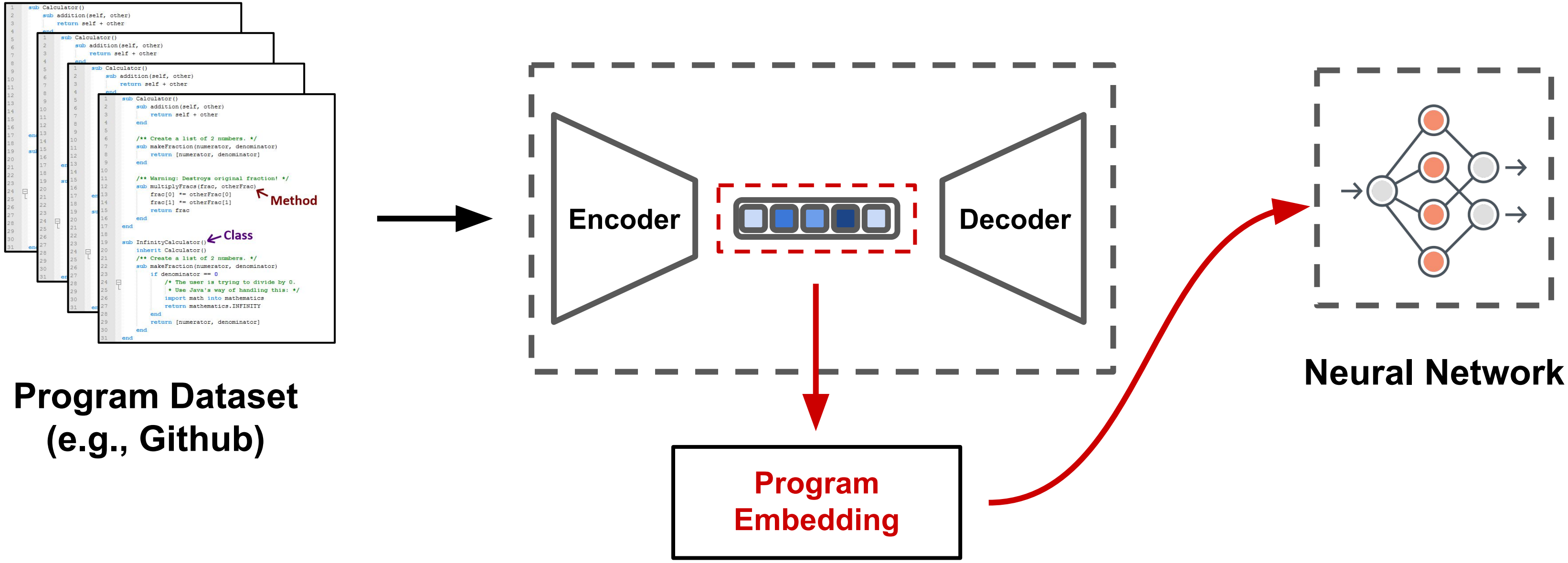
Overview

- Program Induction (Program by Example)
- Neural-Guided Program Synthesis
- **Learning Program Representations**
- Future Challenges

Learning Program Representations

1. **Discrete search space** vs **Continuous search space**
2. Code = **semantic** + **structural** components
3. **Large source code datasets** (e.g., Github, Bitbucket)
4. **Program embeddings** can be used for many downstream tasks

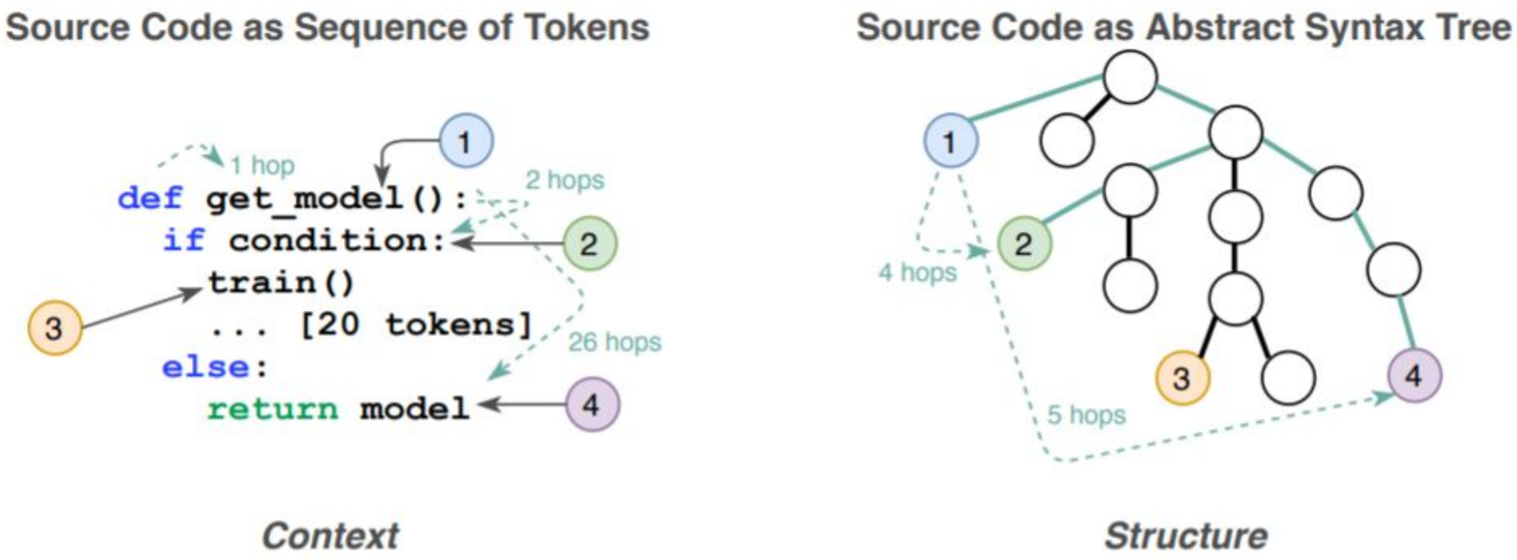
Learning Program Representations



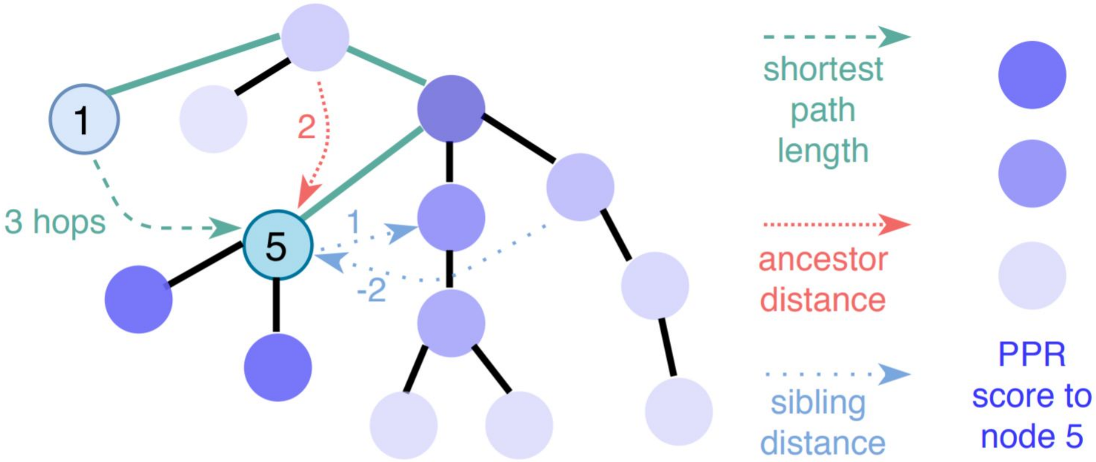
Code Transformer (Zügner et. al, 2021)

1. Learn a **language-agnostic** model for code over multiple languages
2. Exploit **both context** and **structure** of the source code
   a. They shows that context alone leads to lower performance
3. Extends Transformer to **encode possible structure** of the input domain
4. Provide good results on the **code summarization task**

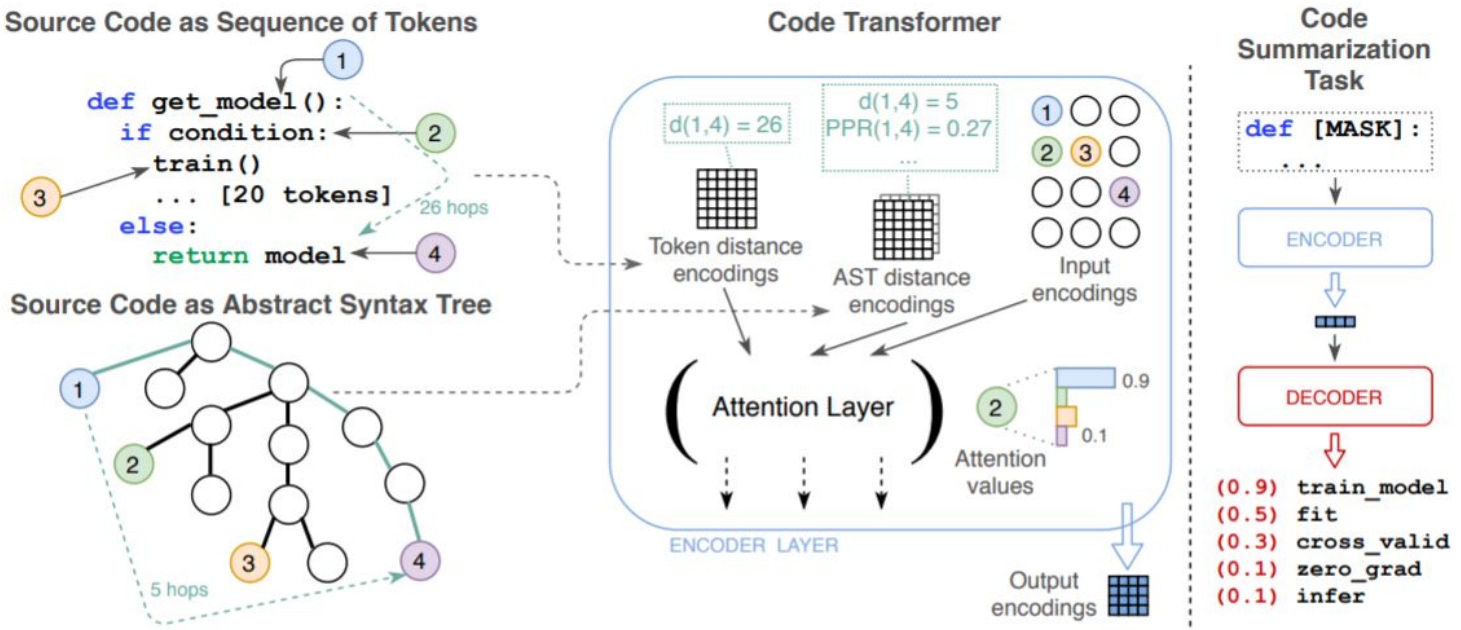
Code Transformer (Zügner et. al, 2021)



Code Transformer (Zügner et. al, 2021)



Code Transformer (Zügner et. al, 2021)



Code Transformer (Zügner et. al, 2021)

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$$A_{i,j} = Q_i^T K_j = E_i^T W_q^T W_k E_j + E_i^T W_q^T W_k \boxed{\phi(r_{i\rightarrow j})} + u^T W_k E_j + v^t W_r \boxed{\phi(r_{i\rightarrow j})}$$

The Attention formula is adapted from Dai et al. (2019) and Yang et al. (2019). They include the relative position encoding.

$r_{i\rightarrow j}$ indicates the **relative distance** between token **i** and token **j** in the sequence.

Code Transformer (Zügner et. al, 2021)

```java
static String execCommand(File f, String... cmd) throws IOException {
    String[] args = new String[cmd.length + 1];
    System.arraycopy(cmd, 0, args, 0, cmd.length);
    args[cmd.length] = f.getCanonicalPath();
    String output = Shell.execCommand(args);
    return output;
}
```

Code Transformer (Zügner et. al, 2021)

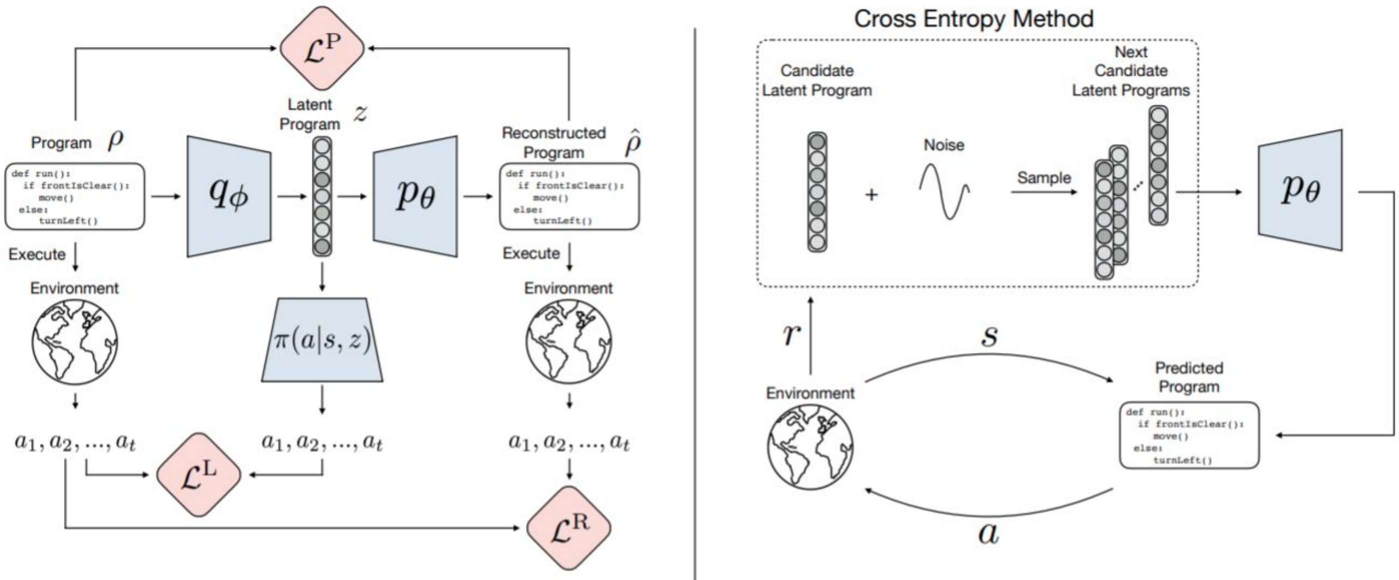
```
static String  MASKED  (File f, String... cmd) throws IOException {
    String[] args = new String[cmd.length + 1];
    System.arraycopy(cmd, 0, args, 0, cmd.length);
    args[cmd.length] = f.getCanonicalPath();
    String output = Shell.execCommand(args);
    return output;
}
```

Code Transformer (Zügner et. al, 2021)

```
static String  MASKED  (File f, String... cmd) throws IOException {
    String[] args = new String[cmd.length + 1];
    System.arraycopy(cmd, 0, args, 0, cmd.length);
    args[cmd.length] = f.getCanonicalPath();
    String output = Shell.execCommand(args);
    return output;
}
```

Model	Prediction
GREAT	get canonical path
code2seq	exec
Ours w/o structure	get output
CODE TRANSFORMER	exec command
Ground Truth	exec command

LEAPS (Trivedi et al., 2021)

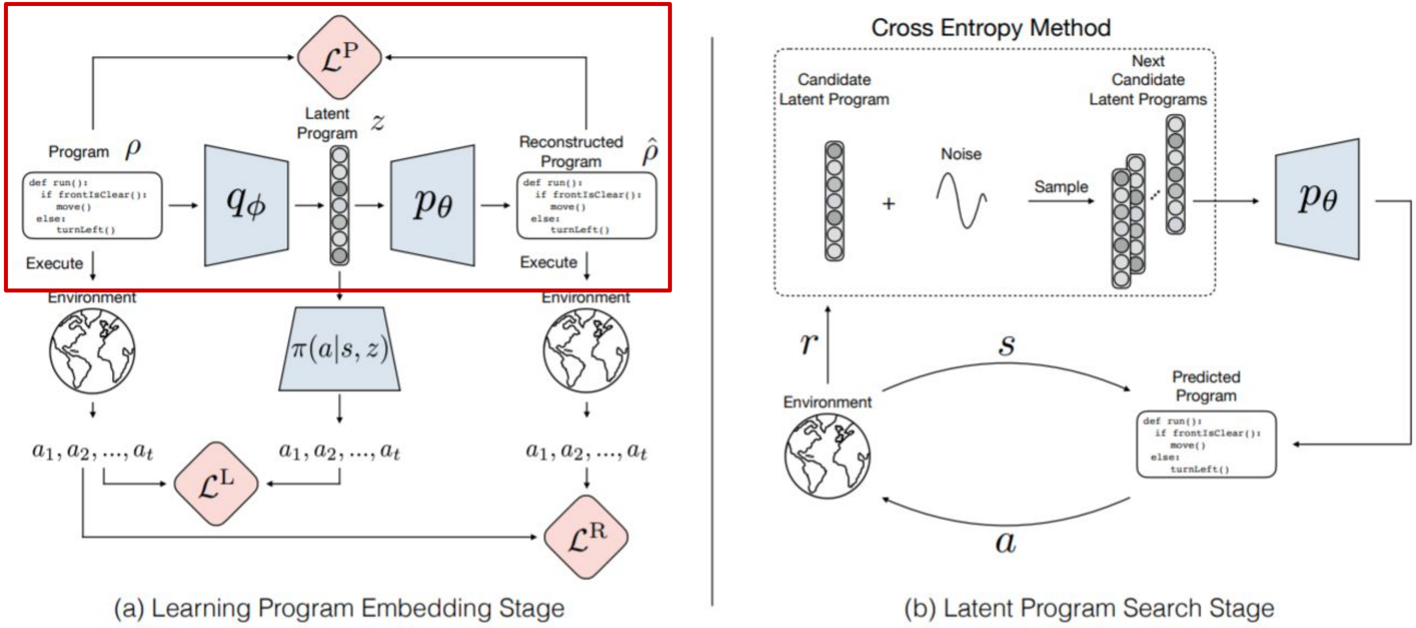


(a) Learning Program Embedding Stage

(b) Latent Program Search Stage

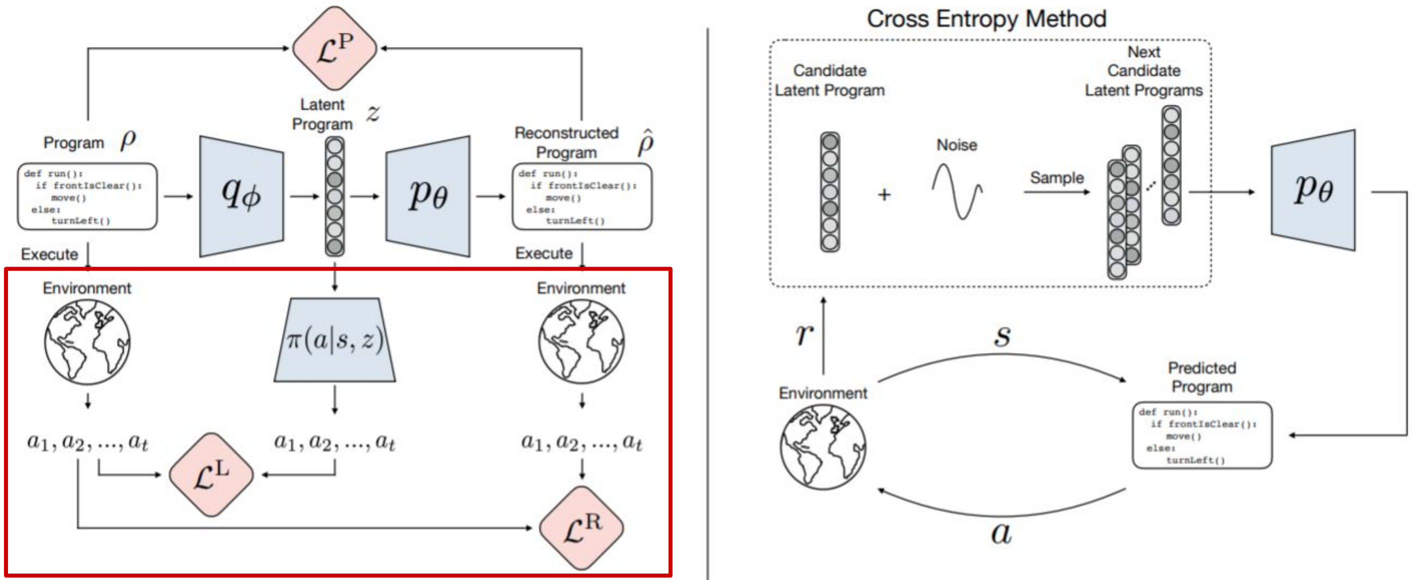
LEAPS (Trivedi et al., 2021)

Reconstruction Loss



LEAPS (Trivedi et al., 2021)

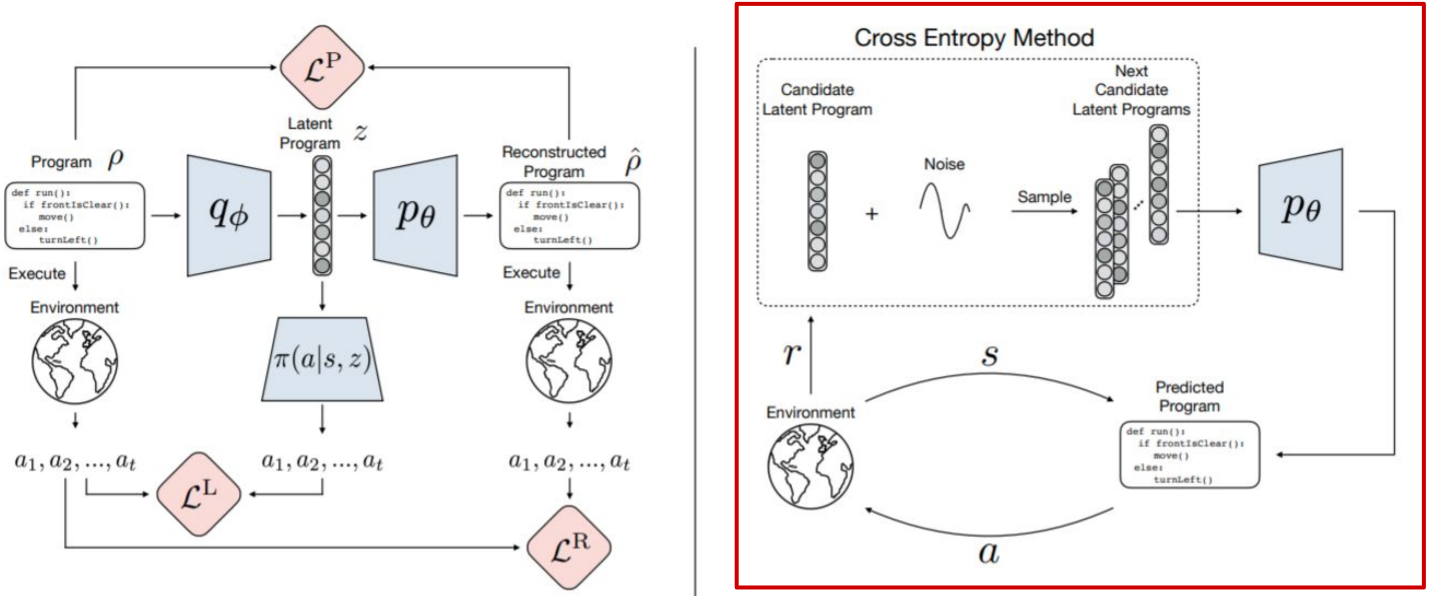
Behaviour Loss



(a) Learning Program Embedding Stage

(b) Latent Program Search Stage

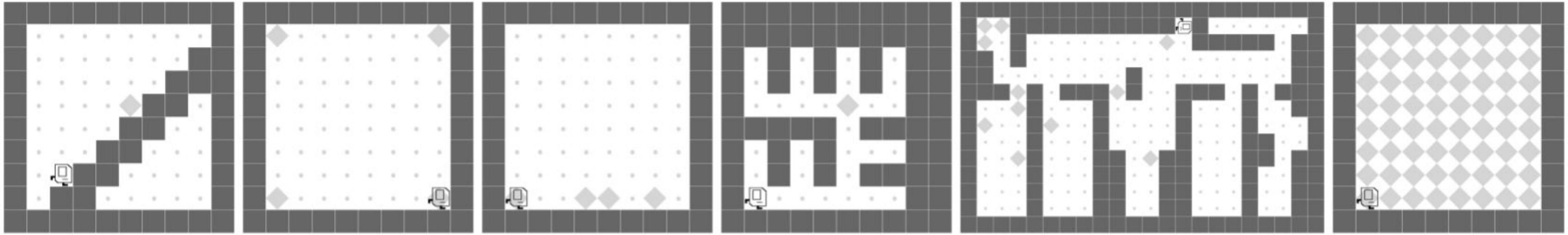
LEAPS (Trivedi et al., 2021)



(a) Learning Program Embedding Stage

(b) Latent Program Search Stage

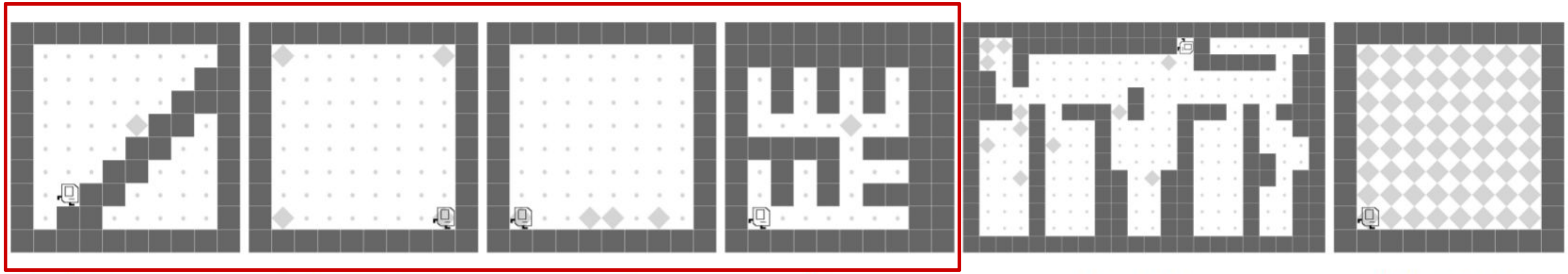
LEAPS (Trivedi et al., 2021)



(a) StairClimber (b) FourCorner (c) TopOff (d) Maze (e) CleanHouse (f) Harvester

Tasks from the Karel domain

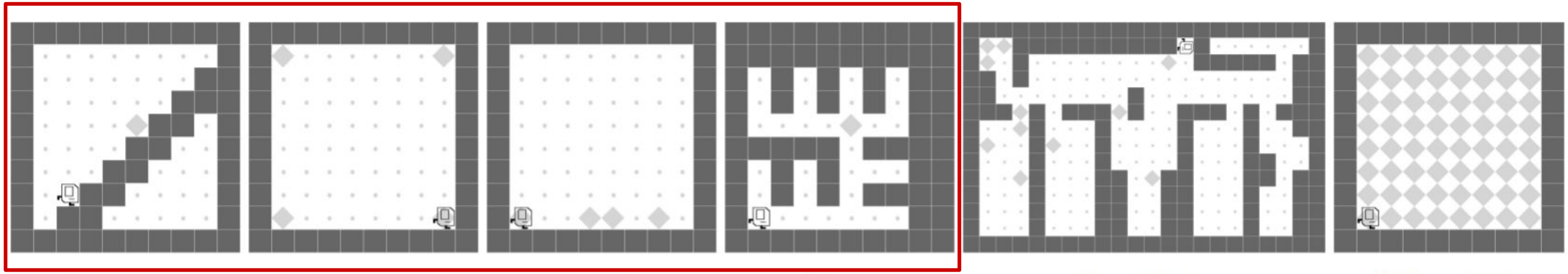
LEAPS (Trivedi et al., 2021)



(a) STAIRCLIMBER (b) FOURCORNER (c) TOPOFF (d) MAZE (e) CLEANHOUSE (f) HARVESTER

Tasks from the Karel domain

LEAPS (Trivedi et al., 2021)



Tasks from the Karel domain

https://clvrai.github.io/leaps/

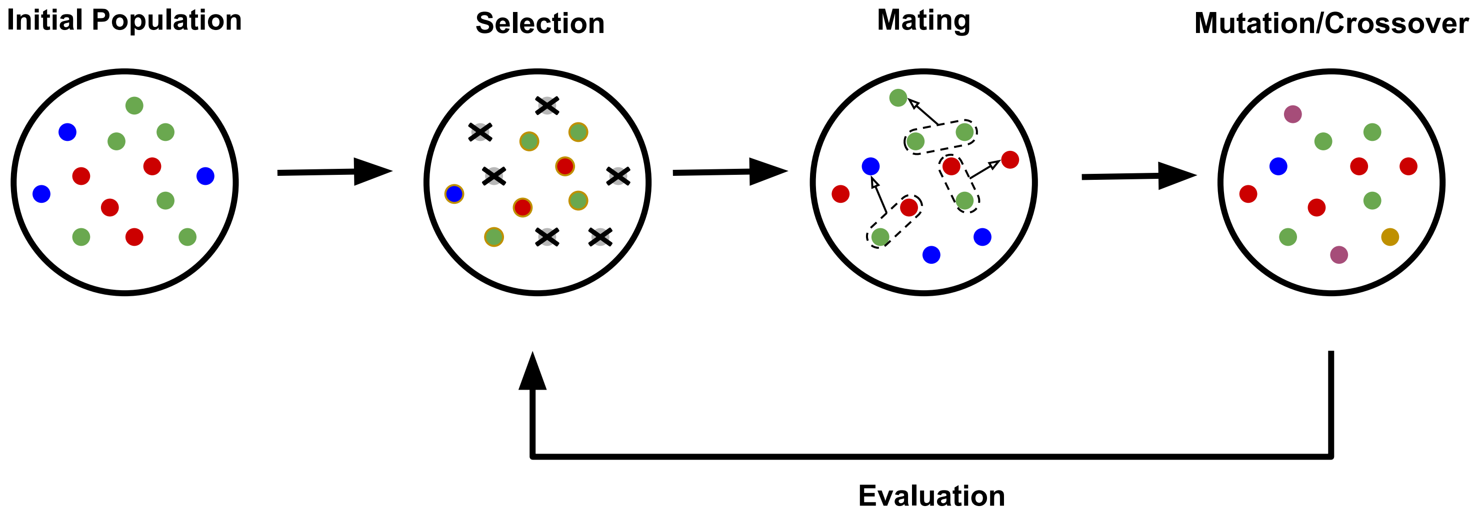
Program Synthesis as Latent Continuous Optimization
(Liskowski et al., 2020)

- Combine **Evolutionary Algorithm + Program Embeddings**
- Use **CMA-ES** as numerical optimization strategy
- Benchmark their method on a **set of 16 program synthesis tasks**
    a. The tasks are very simple programs, such as Mal'cev term or discriminators

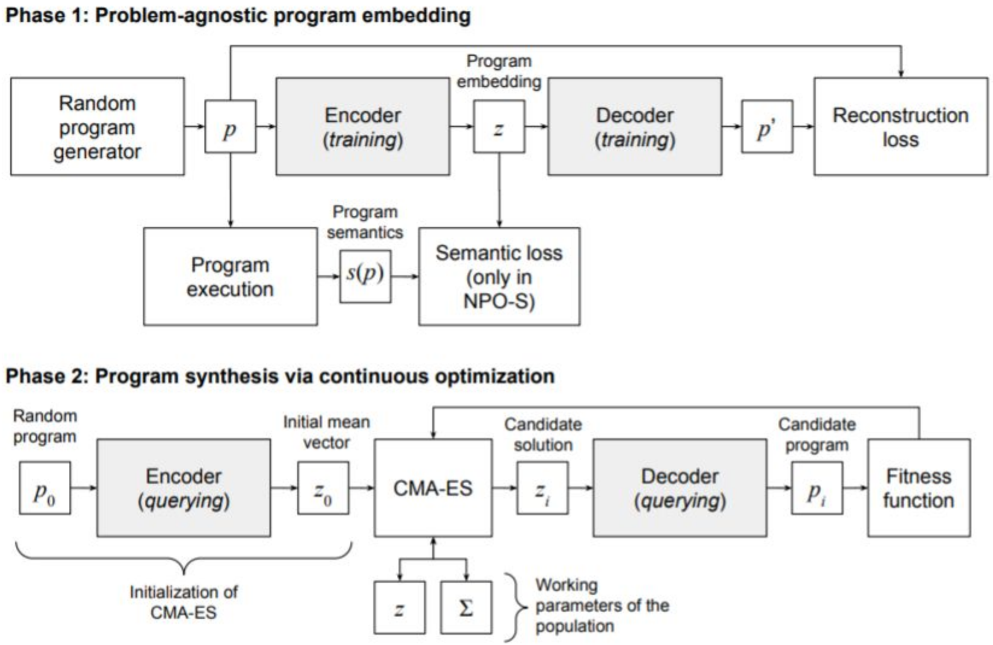
$$\text{Mal'cev term :} \quad m(x, x, y) = m(y, x, x) = y$$

$$\text{Discrim.:} \quad t^a(x, y, z) = \begin{cases} x & \text{if } x \neq y \\ z & \text{if } x = y \end{cases}$$

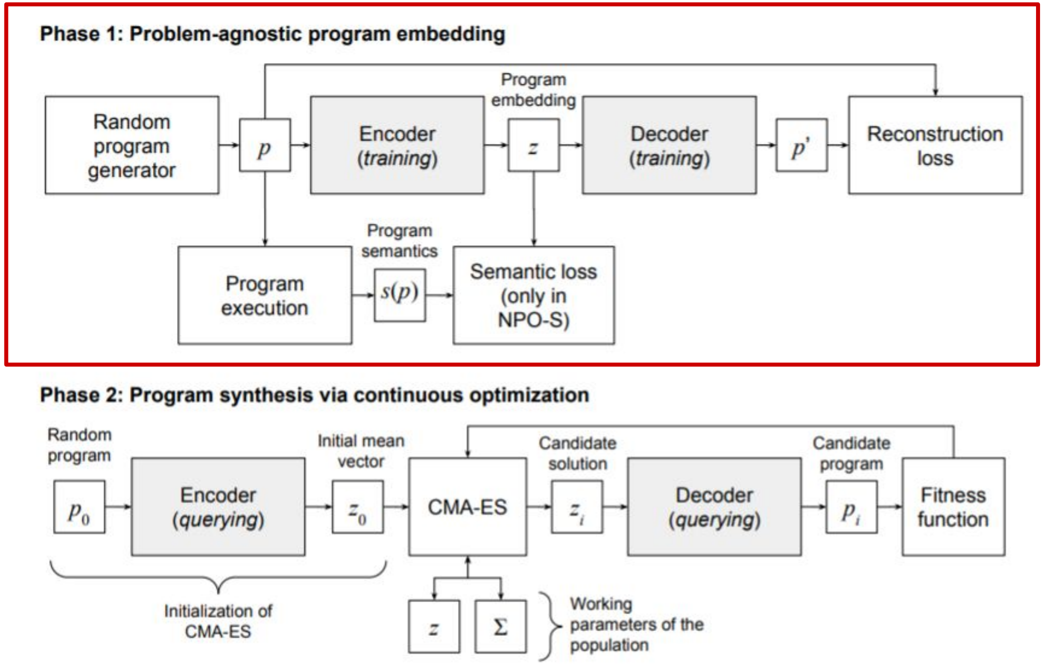
Program Synthesis as Latent Continuous Optimization
(Liskowski et al., 2020)



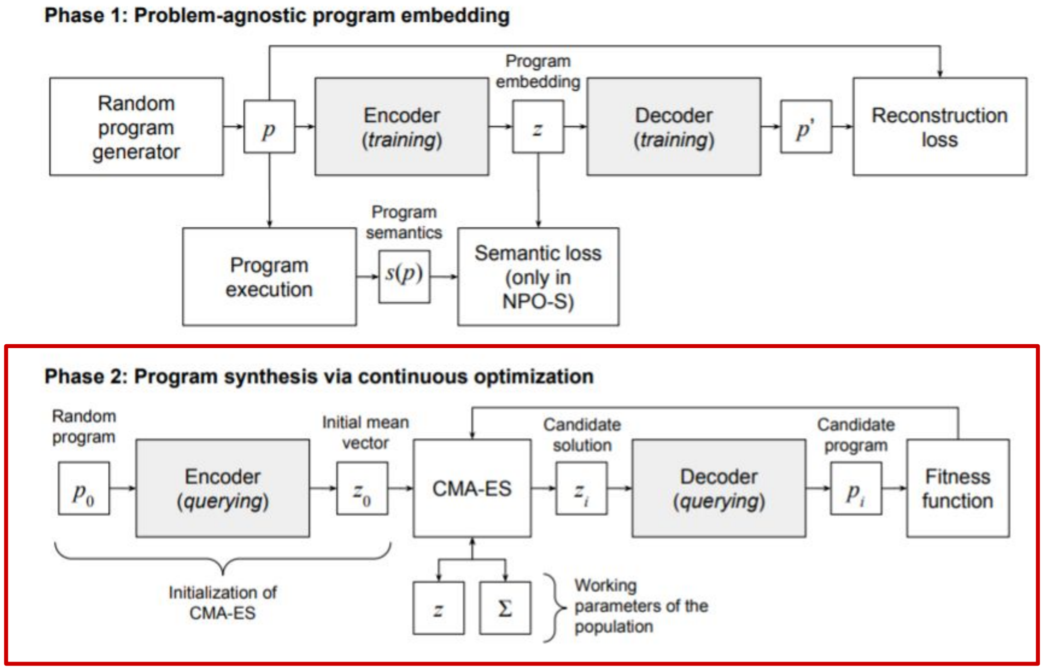
Program Synthesis as Latent Continuous Optimization
(Liskowski et al., 2020)



Program Synthesis as Latent Continuous Optimization
(Liskowski et al., 2020)



Program Synthesis as Latent Continuous Optimization
(Liskowski et al., 2020)



AutoML-Zero (Real et al., 2020)

Google AI Blog

The latest from Google Research

AutoML-Zero: Evolving Code that Learns

Thursday, July 9, 2020

Posted by Esteban Real, Staff Software Engineer and Chen Liang, Software Engineer, Google Research, Brain Team

Machine learning (ML) has seen tremendous successes recently, which were made possible by ML algorithms like deep neural networks that were discovered through years of expert research. The difficulty involved in this research fueled *AutoML*, a field that aims to automate the design of ML algorithms. So far, AutoML has focused on constructing solutions by combining sophisticated hand-designed components. A typical example is that of *neural architecture search*, a subfield in which one builds neural networks automatically out of complex layers (e.g., convolutions, batch-norm, and dropout), and the topic of much research.

An alternative approach to using these hand-designed components in AutoML is to search for entire algorithms from scratch. This is challenging because it requires the exploration of vast and sparse search spaces, yet it has great potential benefits — it is not biased toward what we already know and potentially allows for the discovery of new and better ML architectures. By analogy, if one were building a house from scratch, there is more potential for flexibility or improvement than if one was constructing a house using only prefabricated rooms. However, the discovery of such housing

Search blog ...

Labels

Archive

Feed

Follow @googleai

Give us feedback in our Product Forums.

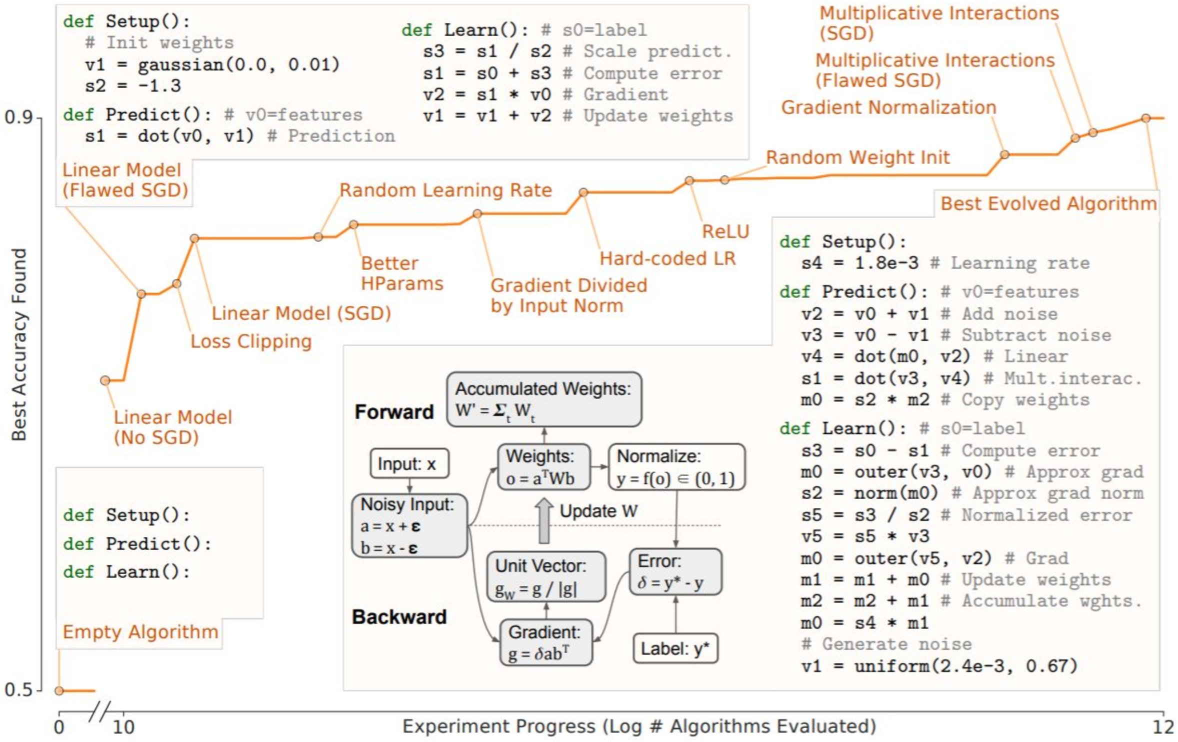
AutoML-Zero (Real et al., 2020)

```
# (Setup, Predict, Learn) = input ML algorithm.
# Dtrain / Dvalid = training / validation set.
# sX/vX/mX: scalar/vector/matrix var at address X.
def Evaluate(Setup, Predict, Learn, Dtrain,
Dvalid):
  # Zero-initialize all the variables (sX/vX/mX).
  initialize_memory()
  Setup() # Execute setup instructions.
  for (x, y) in Dtrain:
    v0 = x # x will now be accessible to Predict.
    Predict() # Execute prediction instructions.
    # s1 will now be used as the prediction.
    s1 = Normalize(s1) # Normalize the prediction.
    s0 = y # y will now be accessible to Learn.
    Learn() # Execute learning instructions.
  sum_loss = 0.0
  for (x, y) in Dvalid:
    v0 = x
    Predict() # Only Predict(), not Learn().
    s1 = Normalize(s1)
    sum_loss += Loss(y, s1)
  mean_loss = sum_loss / len(Dvalid)
  # Use validation loss to evaluate the algorithm.
  return mean_loss
```

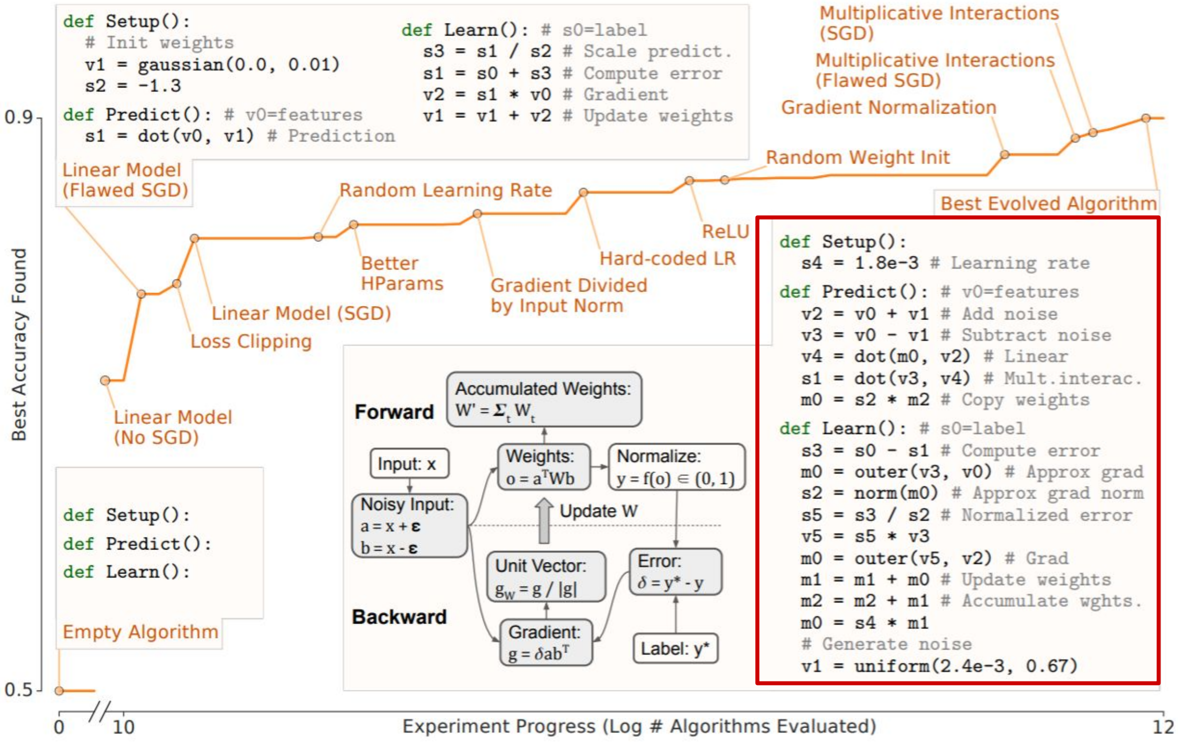
AutoML-Zero (Real et al., 2020)

```
# (Setup, Predict, Learn) = input ML algorithm.
# Dtrain / Dvalid = training / validation set.
# sX/vX/mX: scalar/vector/matrix var at address X.
def Evaluate(Setup, Predict, Learn, Dtrain,
Dvalid):
  # Zero-initialize all the variables (sX/vX/mX).
  initialize_memory()
  Setup() # Execute setup instructions.
  for (x, y) in Dtrain:
    v0 = x # x will now be accessible to Predict.
    Predict() # Execute prediction instructions.
    # s1 will now be used as the prediction.
    s1 = Normalize(s1) # Normalize the prediction.
    s0 = y # y will now be accessible to Learn.
    Learn() # Execute learning instructions.
  sum_loss = 0.0
  for (x, y) in Dvalid:
    v0 = x
    Predict() # Only Predict(), not Learn().
    s1 = Normalize(s1)
    sum_loss += Loss(y, s1)
  mean_loss = sum_loss / len(Dvalid)
  # Use validation loss to evaluate the algorithm.
  return mean_loss
```

AutoML-Zero (Real et al., 2020)



AutoML-Zero (Real et al., 2020)



Future Challenges

1. Deal with underspecification to understand **what the user really want**

2. Study **reasoning/planning over the latent space**

3. **Novel algorithm discovery with minimal supervision**

4. Apply program synthesis techniques to **everyday software programming**

Future Challenges

1. Deal with underspecification to understand **what the user really want**

2. Study **reasoning/planning over the latent space**

3. **Novel algorithm discovery with minimal supervision**

4. Apply program synthesis techniques to **everyday software programming**

For those interested in, some thesis are available on the broad topic of program synthesis, interactive program synthesis, etc.!

Thank you!

Questions?

Resources

Program Induction

1. Gulwani, Sumit. **"Automating string processing in spreadsheets using input-output examples."** *ACM Sigplan Notices* 46.1 (2011). (https://dl.acm.org/doi/pdf/10.1145/1925844.1926423)
2. Graves, Alex, et al. **Hybrid computing using a neural network with dynamic external memory**. Nature 538, 471–476 (2016). (https://doi.org/10.1038/nature20101)
3. Graves, Alex, Greg Wayne, and Ivo Danihelka. "**Neural turing machines.**" *arXiv preprint arXiv:1410.5401* (2014). (https://arxiv.org/abs/1410.5401)
4. Balog, M., et al. "**DeepCoder: Learning to write programs.**" *5th International Conference on Learning Representations, ICLR 2017-Conference Track Proceedings*. 2019. (https://arxiv.org/pdf/1611.01989)

Neural-Guided Program Synthesis

5. Reed, Scott, and Nando De Freitas. "**Neural programmer-interpreters.**" *arXiv preprint arXiv:1511.06279* (2015). (https://arxiv.org/abs/1511.06279)
6. Pierrot, Thomas, et al. "**Learning Compositional Neural Programs with Recursive Tree Search and Planning**." *Advances in Neural Information Processing Systems* 32 (2019): 14673-14683. (https://openreview.net/forum?id=rJg1aBHgUB)
7. Bunel, Rudy, et al. "**Leveraging Grammar and Reinforcement Learning for Neural Program Synthesis.**" *International Conference on Learning Representations*. 2018. (https://openreview.net/forum?id=H1Xw62kRZ)

Resources

Learning Program Representations

1. Chen, Mark, et al. "**Evaluating large language models trained on code**." *arXiv preprint arXiv:2107.03374* (2021). (https://arxiv.org/pdf/2107.03374)
2. Zügner, Daniel, et al. "**Language-Agnostic Representation Learning of Source Code from Structure and Context.**" *International Conference on Learning Representations*. 2021. (https://openreview.net/pdf?id=B1lnbRNtwr)
3. Hellendoorn, Vincent J., et al. "**Global relational models of source code.**" *International Conference on Learning Representations*. 2020. (https://openreview.net/forum?id=Xh5eMZVONGF)
4. Allamanis, Miltiadis, Marc Brockschmidt, and Mahmoud Khademi. "**Learning to Represent Programs with Graphs.**" *International Conference on Learning Representations*. 2018. (https://openreview.net/forum?id=BJOFETxR-)

1. Trivedi, Dweep, et al. "**Learning to Synthesize Programs as Interpretable and Generalizable Policies.**" *Thirty-Fifth Conference on Neural Information Processing Systems*. 2021. (https://papers.nips.cc/paper/2021/file/d37124c4c79f357cb02c655671a432fa-Paper.pdf)
2. Liskowski, Paweł, et al. "**Program synthesis as latent continuous optimization: evolutionary search in neural embeddings.**" *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 2020. (https://dl.acm.org/doi/pdf/10.1145/3377930.3390213)
3. Real, Esteban, et al. "**Automl-zero: Evolving machine learning algorithms from scratch.**" *International Conference on Machine Learning*. PMLR, 2020. (http://proceedings.mlr.press/v119/real20a/real20a.pdf)

Resources

General Other Resources

1. **Machine Learning for Big Code and Naturalness** - https://ml4code.github.io/
   General website which should serve as a gathering point of many research works related to program synthesis and machine learning for code. See also the associated paper https://arxiv.org/abs/1709.06182.

2. **MIT Course “Introduction to program synthesis”** - https://people.csail.mit.edu/asolar/SynthesisCourse/index.htm

Acknowledgements

Images:
https://www.flaticon.com/authors/good-ware
https://www.flaticon.com/authors/surang
https://www.veryicon.com/icons/miscellaneous/color-work-icon/artificial-neural-network.html

Some of the pictures were taken directly from the discussed papers. Otherwise noted, all the rights are reserved to the corresponding authors.

The initial part of this talk (introduction and flash fill analysis) was greatly inspired by the fantastic talk of Alex Polozov, that you can find at this link https://www.microsoft.com/en-us/research/video/advanced-machine-learning-day-3-neural-program-synthesis/