NEURAL-GUIDED PROGRAM SYNTHESIS

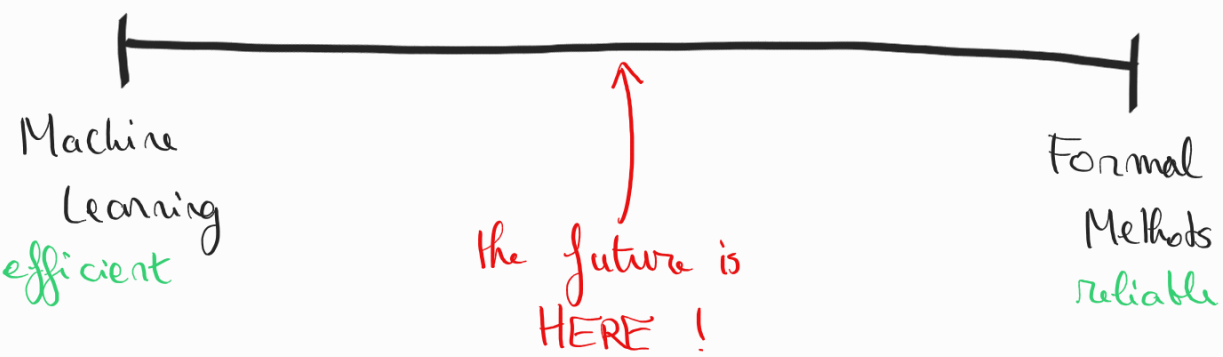
Nathanaël Fijalkow

Based on AAAI'22 paper with

G. Lagarde
T. Matricon
K. Ellis
P. Ohlmann
A. Potta

See DeepSynth on Github

All in one slide:



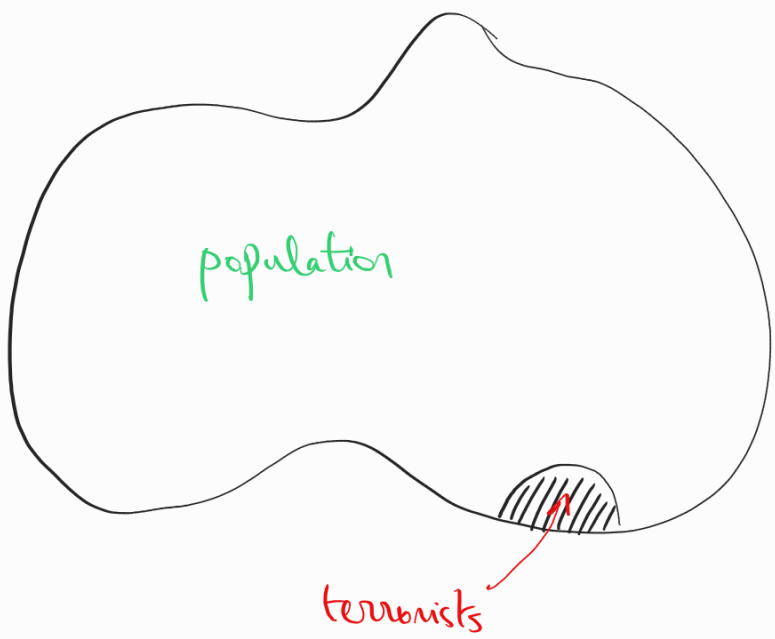
(1) FM → ML : improving ML's guarantees using FM

(2) ML → FM : improving FM's performance using ML

AIRPORT SCREENING

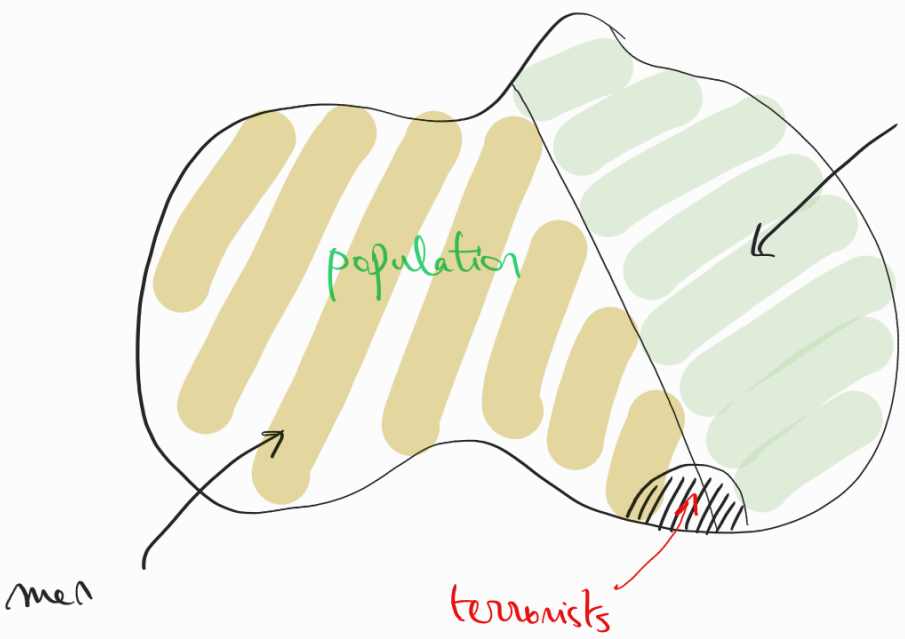
The following example is REAL
BUT does NOT reflect in any way
the speaker's political views

Problem :



Who to scan in order to catch terrorists ?

Assumption : prior (think Baysesian)



Say :
men are 9 times more likely to be terrorists

Mathematically :

$\mathcal{D}$ distribution over $\mathcal{X}$ population

We want to do sampling **with replacement**

↑ we'll come back to that

Objective: **minimise**

$\mathbb{E}$ [**# scans** before catching a terrorist]

↑ we call that the **loss**

Naive: $\mathcal{D}$

Optimal: $\sqrt{\mathcal{D}}$

defined by $\sqrt{\mathcal{D}}(x) \propto \sqrt{\mathcal{D}(x)}$:

$$\sqrt{\mathcal{D}}(x) = \frac{\sqrt{\mathcal{D}(x)}}{\sum_{x' \in X} \sqrt{\mathcal{D}(x')}}$$

Simplest instance of the SQRT sampling theorem:

$\mathcal{D}$: **p** HEAD + **(1-p)** TAIL $X = \{\text{HEAD}, \text{TAIL}\}$

$\mathcal{D}'$: p' HEAD + $(1-p')$ TAIL

$$\min_{p'} \alpha(\mathcal{D}', \mathcal{D}) = \min_{p'} \frac{p}{p'} + \frac{1-p}{1-p'}$$

$$p' = \frac{\sqrt{p}}{\sqrt{p} + \sqrt{1-p}}$$

If men are **9** times more likely to be terrorists
they should be scanned **3** times more often

INDUCTIVE PROGRAM SYNTHESIS

Fix a DSL : domain specific language

Input : a few examples

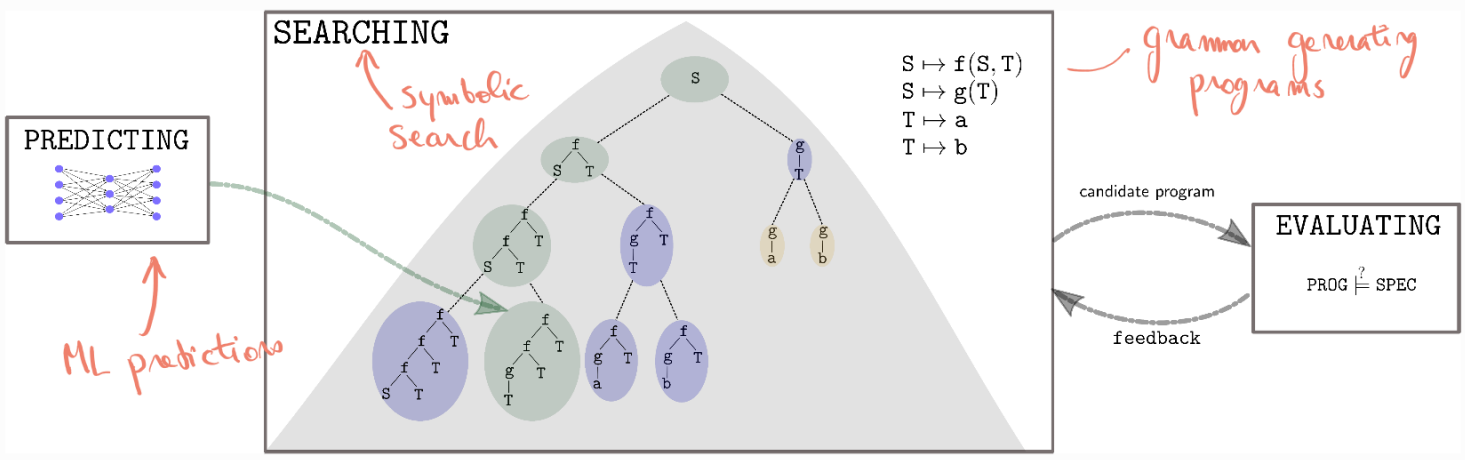
Output : a program satisfying the examples

Example :

Input	→	Output
[1,5,4,2]	→	[2,4]
[6,3,0,8]	→	[0,6,8]

→ SORT ; FILTER[EVEN]

NEURAL GUIDED INDUCTIVE SYNTHESIS



First practical instance : DeepCoder , 2017

Key idea : patterns found in examples reveal which primitives are used in a solution program

Example :

[1,5,4,2]	→	[2,4]
[6,3,0,8]	→	[0,6,8]

likely primitives : SORT , FILTER(EVEN)

PROGRAM REPRESENTATION

DSL

↑

domain specific language

list of primitives

```
sort : list(int) → list(int)
map  : (t0 → t1) → list(t0) → list(t1)
succ : int → int
  ⋮
```

COMPILATION

CFG

```
list(int) → Sort ( list(int))
list(int) → map ( int → int , list(int))
list(int) → input

int → int → succ ()
  ⋮
```

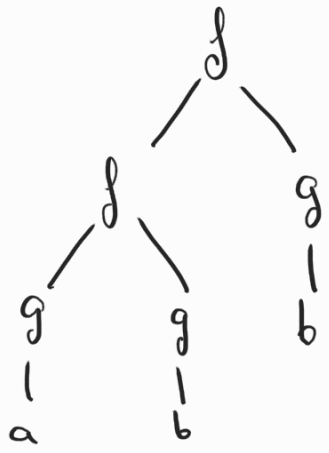
PROBABILISTIC CONTEXT FREE GRAMMARS

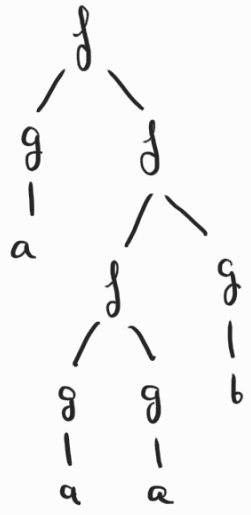
CFG $\begin{cases} S \longrightarrow f(S,S) + g(T) \\ T \longrightarrow a + b \end{cases}$

f, g : primitives

a, b : variables or constants

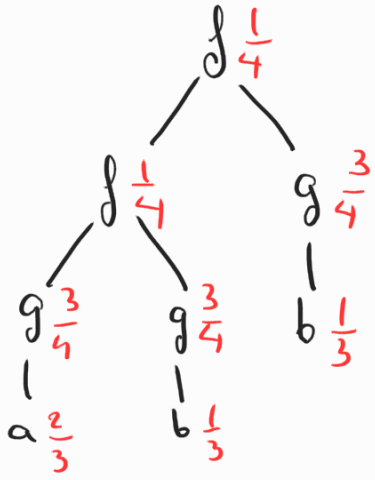
ex:





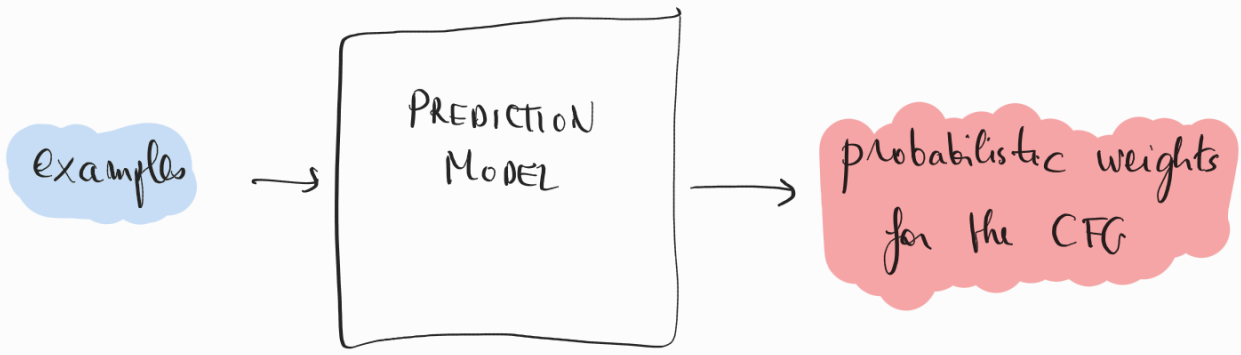
PCFG $\begin{cases} S \longrightarrow \frac{1}{4} f(S,S) + \frac{3}{4} g(T) \\ T \longrightarrow \frac{2}{3} a + \frac{1}{3} b \end{cases}$

ex:

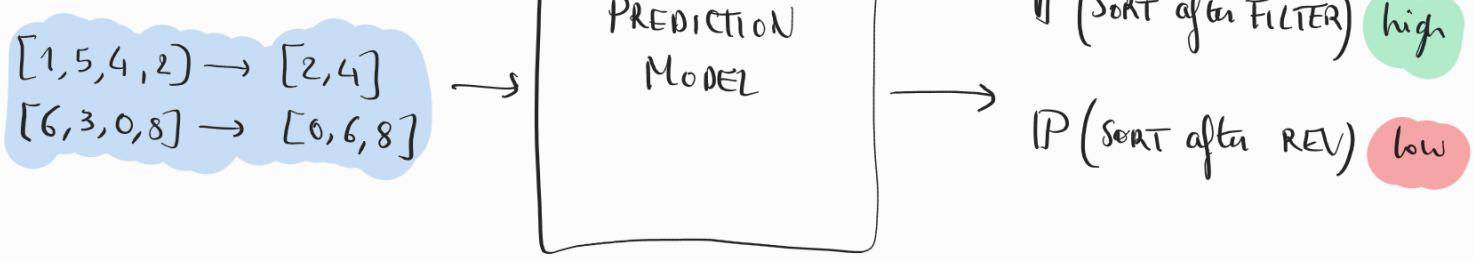


A PCFG induces a distribution over trees = programs

PREDICTIONS



Example:



The prediction model infers:

$$\mathbb{P}(\text{program} \mid \text{examples})$$

↳ So we have a prior distribution $\mathcal{D}$ over programs

QUESTION: HOW TO DEPLOY THE PREDICTION MODEL ?

$X = \mathbb{N}_{\geq 1}$ $\qquad \mathcal{D}(n) = \frac{1}{2^n}$ $\qquad \left(\sum_{n \geq 1} \frac{1}{2^n} = 1 \right)$

A_1 : enumerate $1, 2, 3, 4, \ldots$

$$\mathcal{L}(A_1) = \sum_{n \geq 1} \frac{n}{2^n} = 2$$

A_2 : sample with $\mathcal{D}$ with replacement

$$\mathcal{L}(A_2) = \sum_{n \geq 1} \frac{2^n}{2^n} = \infty$$

A_3 : sample with $\sqrt{\mathcal{D}}$: $\sqrt{\mathcal{D}}(n) = \frac{\sqrt{2}+1}{2^{n/2}}$

$$\mathcal{L}(A_3) = (\ldots) \simeq 5{,}83$$

Two approaches :

SAMPLING vs **ENUMERATION**

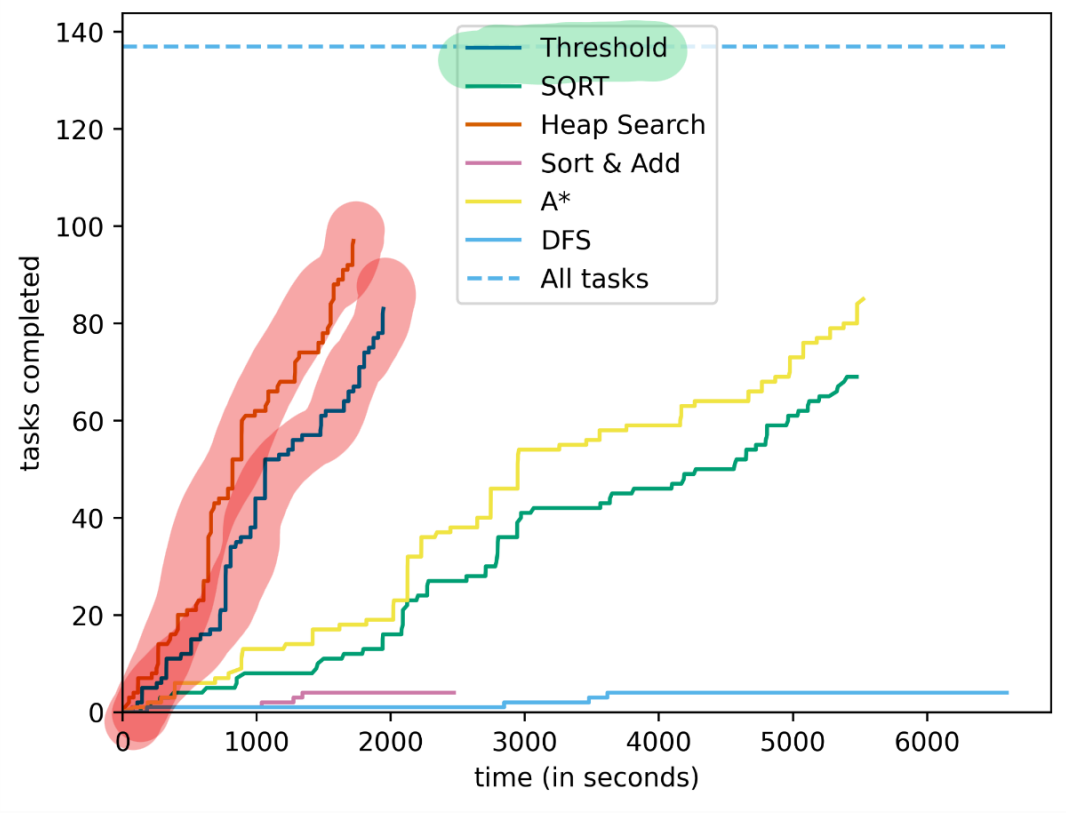
TECHNICAL CONTRIBUTIONS:

(1) HeapSearch: optimal (minimizing loss) bottom-up enumeration algorithm for PCFGs
Idea: collection of heap for storing partial programs

(2) Grammar Splitter: divide search across parallel architecture for PCFGs
Idea: partitioning derivations in a fair way

(3) SQRT sampling: optimal sampling for PCFGs
Idea: optimal for sampling with replacement.

I will not go into the algorithmic details ...



Interpretation:

- A* and HS enumerate exactly the same programs in the same order
  - ↳ HS is 6× faster: better data structures
- TS and HS do different orders
  - ↳ HS outputs more likely programs

SUMMARY

We have a set of GENERIC tools for performing SYMBOLIC SEARCH on grammars enhanced with MACHINE LEARNING PREDICTIONS

Check out our tool : DEEPSYNTH

WHAT'S NEXT ?

- Reinforcement learning
- Reactive synthesis
- Specification mining
- Transductions
- your favourite SYMBOLIC SEARCH question