Scaling Neural Program Synthesis with Distribution-based Search

Nathanaël Fijalkow[1,2] Guillaume Lagarde[1] **Théo Matricon**[1] Kevin Ellis[3] Pierre Ohlmann[4] Akarsh Potta[5]

[1] CNRS, LaBRI and Université de Bordeaux, France

[2] The Alan Turing Institute of data science, United Kingdom

[3] Cornell University, United States

[4] University of Paris, France

[5] Indian Institute of Technology Bombay, India

February, 2022

Our contributions:

- A theoretical framework called distribution-based search for evaluating and comparing search algorithms in the context of machine-learned predictions.

Our contributions:

- A theoretical framework called distribution-based search for evaluating and comparing search algorithms in the context of machine-learned predictions.
- Two new search algorithms: Heap Search, an enumerative method, and SQRT Sampling, a probabilistic method. We prove a number of theoretical results about them, in particular that they are both loss optimal.

Our contributions:

- A theoretical framework called distribution-based search for evaluating and comparing search algorithms in the context of machine-learned predictions.
- Two new search algorithms: Heap Search, an enumerative method, and SQRT Sampling, a probabilistic method. We prove a number of theoretical results about them, in particular that they are both loss optimal.
- A method for running any search algorithm across parallel computing environments.

Our contributions:

- **A theoretical framework called distribution-based search for evaluating and comparing search algorithms in the context of machine-learned predictions.**
- Two new search algorithms: Heap Search, an enumerative method, and SQRT Sampling, a probabilistic method. We prove a number of theoretical results about them, in particular that they are both loss optimal.
- A method for running any search algorithm across parallel computing environments.

Examples:

Examples:

```
f([1, 2, 3]) = [4, 6, 1]
f([]) = []
f([10, 3, 4]) = [6, 1, 0]
f([47, 0, 9, 34]) = [1, 3, 2, 6]
```

Examples:

```
f([1, 2, 3]) = [4, 6, 1]
f([]) = []
f([10, 3, 4]) = [6, 1, 0]
f([47, 0, 9, 34]) = [1, 3, 2, 6]
```

What is f?

Examples:

```
f([1, 2, 3]) = [4, 6, 1]
f([]) = []
f([10, 3, 4]) = [6, 1, 0]
f([47, 0, 9, 34]) = [1, 3, 2, 6]
```

What is f?

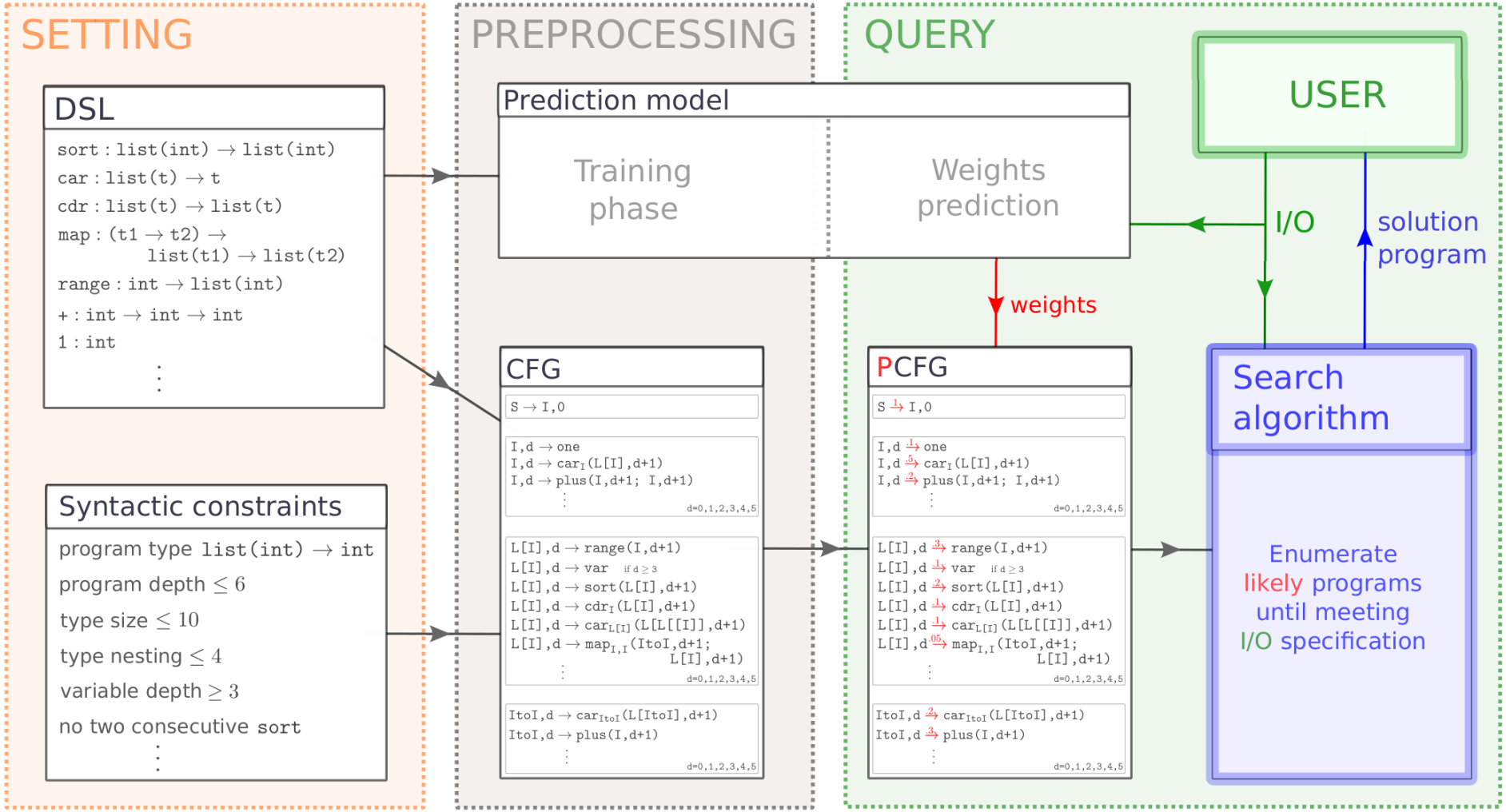
$$\texttt{f} : \texttt{list(int)} \rightarrow \texttt{list(int)}$$

Examples:

```
f([1, 2, 3]) = [4, 6, 1]
f([]) = []
f([10, 3, 4]) = [6, 1, 0]
f([47, 0, 9, 34]) = [1, 3, 2, 6]
```

What is f?

```
f : list(int) → list(int)
f var0 = map (λx. mod x 7) (map (+ 3) var0)
```



Pipeline for neural predictions for syntax guided program synthesis.

NN predicts a PCFG $\rightarrow$ induces a distribution $\mathcal{D}$ over programs

NN predicts a PCFG $\rightarrow$ induces a distribution $\mathcal{D}$ over programs

We look for a program P that meets IO specification
The predictions are given by the prior distribution $\mathcal{D}$

NN predicts a PCFG $\rightarrow$ induces a distribution $\mathcal{D}$ over programs

We look for a program P that meets IO specification
The predictions are given by the prior distribution $\mathcal{D}$

Goal: find P as quickly as possible

Evaluation criterion for an algorithm A given $\mathcal{D}$?

Evaluation criterion for an algorithm A given $\mathcal{D}$?

loss of $(A, \mathcal{D}) =$ the expectation of the number of tries to find the program sampled from the prior distribution

Evaluation criterion for an algorithm A given $\mathcal{D}$?

loss of $(A, \mathcal{D})$ = the expectation of the number of tries to find the program sampled from the prior distribution

A^* is 'loss optimal' if it generates each program **exactly once** and in **non increasing order** of probability.

Evaluation criterion for an algorithm A given $\mathcal{D}$?

loss of $(A, \mathcal{D}) =$ the expectation of the number of tries to find the program sampled from the prior distribution

A^* is 'loss optimal' if it generates each program **exactly once** and in **non increasing order** of probability.

Trade-off: Quality vs Quantity

Our contributions:

- A theoretical framework called distribution-based search for evaluating and comparing search algorithms in the context of machine-learned predictions.
- **Two new search algorithms: Heap Search, an enumerative method, and SQRT Sampling, a probabilistic method. We prove a number of theoretical results about them, in particular that they are both loss optimal.**
- A method for running any search algorithm across parallel computing environments.

Theorem

The Heap Search *algorithm is loss optimal: it enumerates every program exactly once and in non-increasing order of probabilities.*

It uses a data structure made of heaps and hashing tables to efficiently enumerate programs.

Sampling Algorithms: may generate a program mutiple times but do not require memory.

Sampling Algorithms: may generate a program mutiple times but do not require memory.

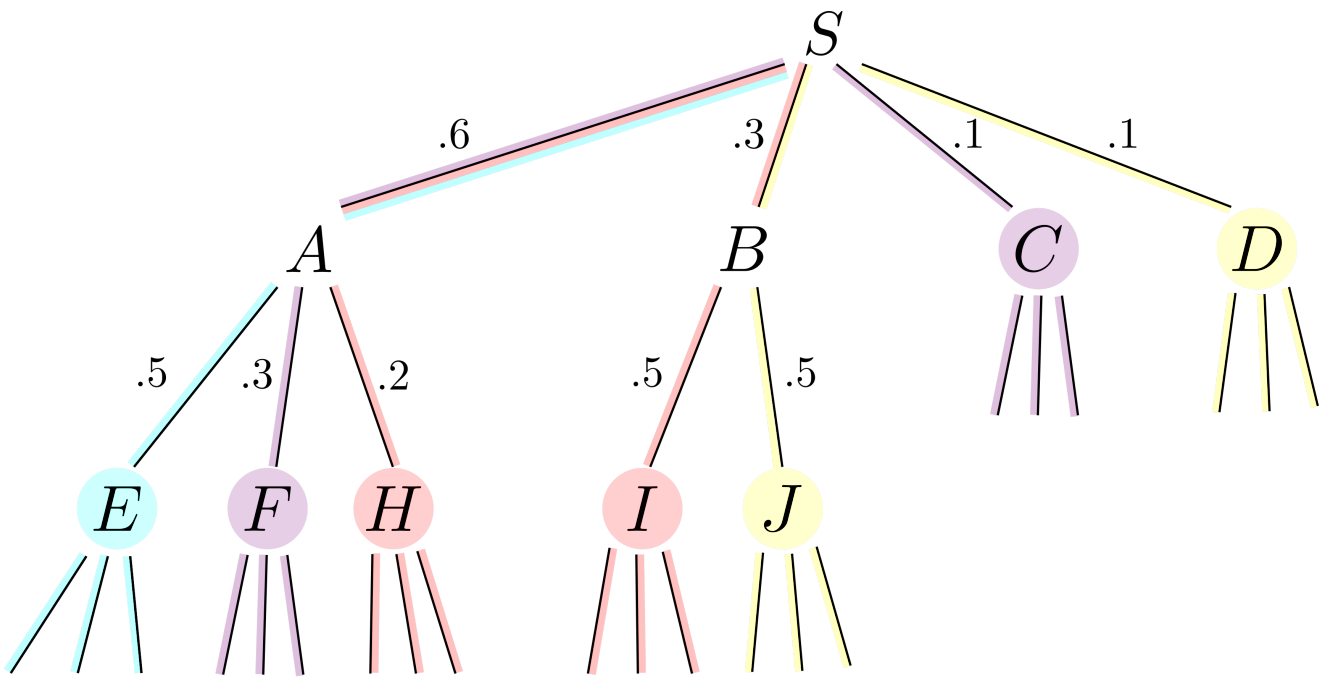
Theorem

The SQRT SAMPLING *algorithm is loss optimal among sampling algorithms.*

SQRT SAMPLING samples program from the square root distribution of the prior distribution $\mathcal{D}$.

Our contributions:

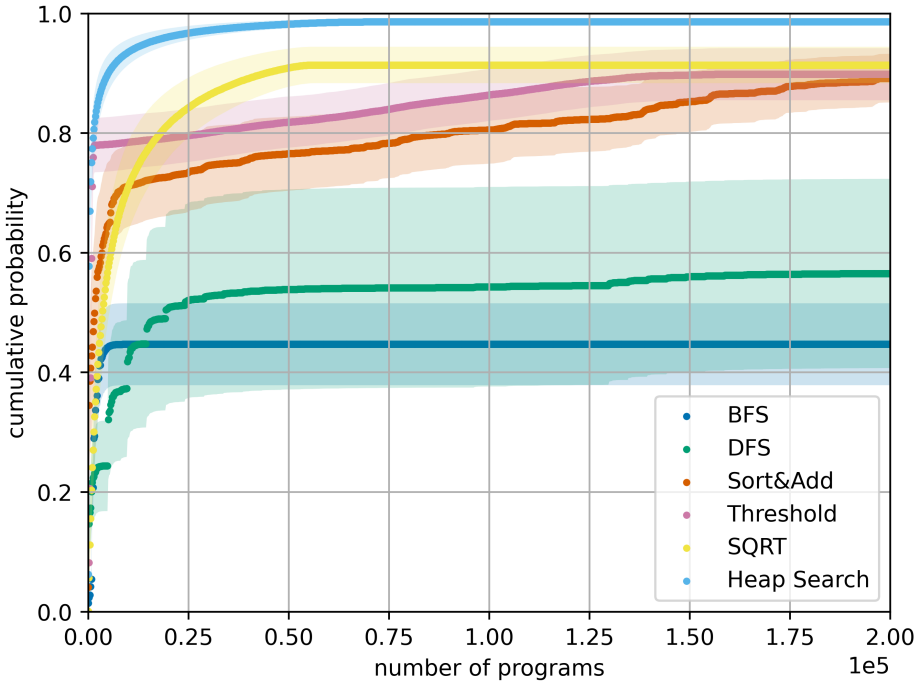
- A theoretical framework called distribution-based search for evaluating and comparing search algorithms in the context of machine-learned predictions.
- Two new search algorithms: Heap Search, an enumerative method, and SQRT Sampling, a probabilistic method. We prove a number of theoretical results about them, in particular that they are both loss optimal.
- **A method for running any search algorithm across parallel computing environments.**



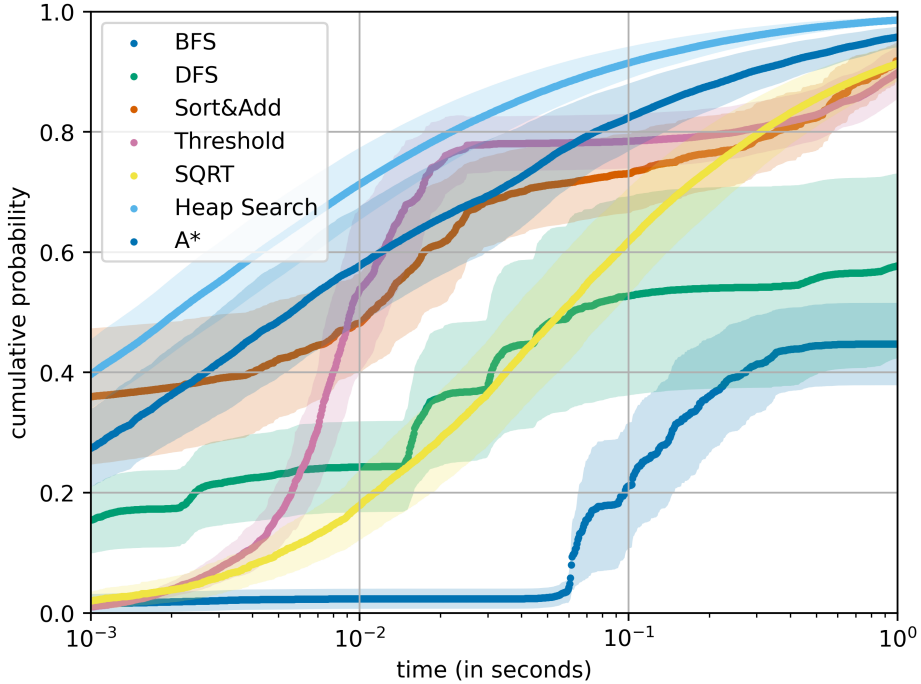
$$\begin{array}{l} S \xrightarrow{.6} A \\ S \xrightarrow{.3} B \\ S \xrightarrow{.1} C \\ S \xrightarrow{.1} D \\ A \xrightarrow{.1} E \\ A \xrightarrow{.3} F \\ A \xrightarrow{.2} H \\ B \xrightarrow{.5} I \\ B \xrightarrow{.5} J \\ E \longrightarrow \dots \\ \vdots \quad \vdots \quad \vdots \end{array} = .3 \times \begin{array}{l} S \xrightarrow{1} A \\ A \xrightarrow{1} E \\ E \to \cdots \\ \vdots \quad \vdots \quad \vdots \end{array} + .28 \times \begin{array}{l} S \xrightarrow{.857} A \\ S \xrightarrow{.143} C \\ A \xrightarrow{1} F \\ F \longrightarrow \dots \\ \vdots \quad \vdots \quad \vdots \end{array} + .27 \times \begin{array}{l} S \xrightarrow{.667} A \\ S \xrightarrow{.333} B \\ A \xrightarrow{1} H \\ B \xrightarrow{1} I \\ H \longrightarrow \cdots \\ \vdots \quad \vdots \quad \vdots \end{array} + .25 \times \begin{array}{l} S \xrightarrow{.75} B \\ S \xrightarrow{.25} D \\ B \xrightarrow{1} J \\ J \longrightarrow \dots \\ \vdots \quad \vdots \quad \vdots \end{array}$$

The grammar splitter: a balanced partition with imbalance $\alpha = \frac{.3}{.25} = 1.2$.

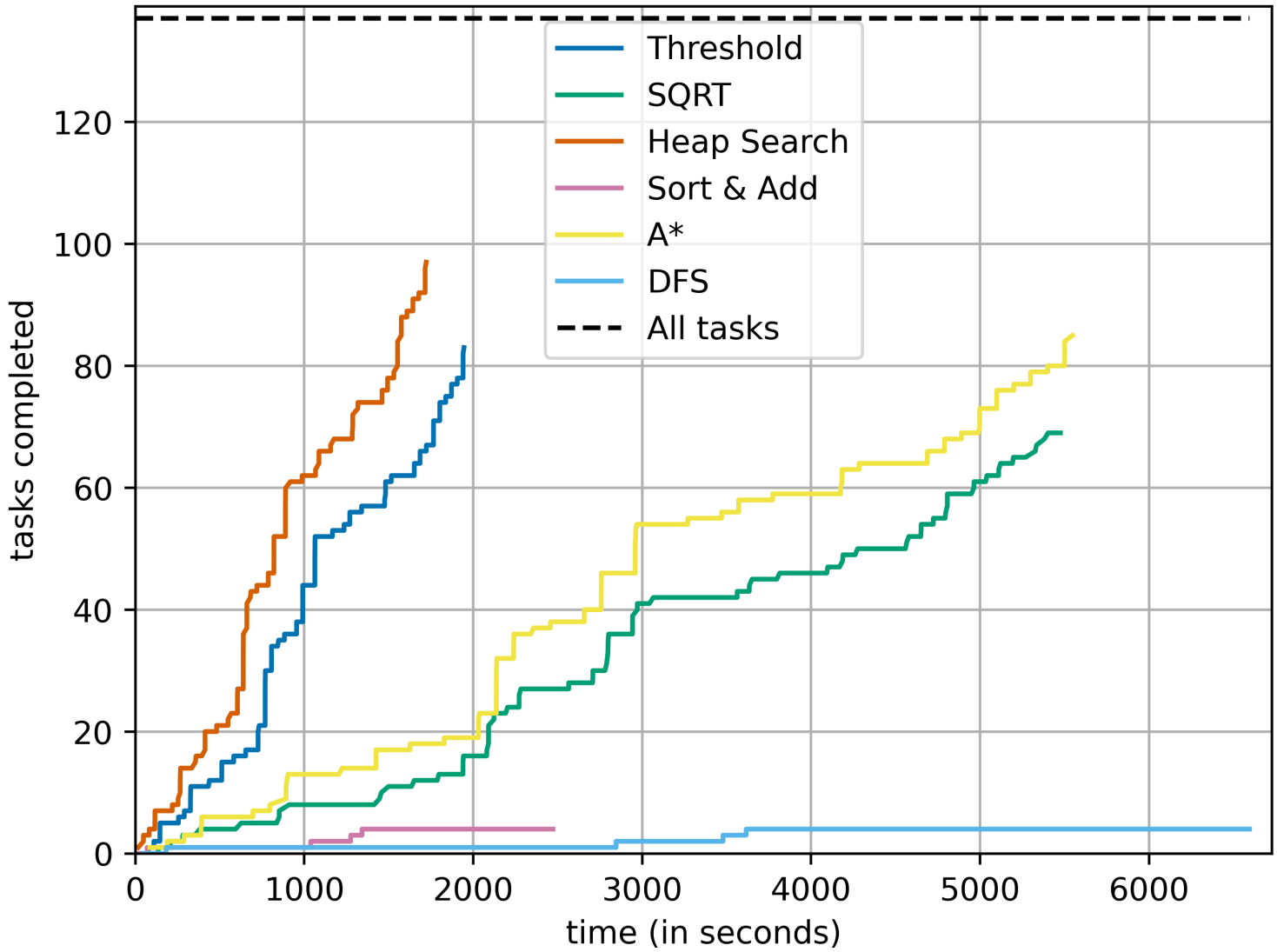
Experiments



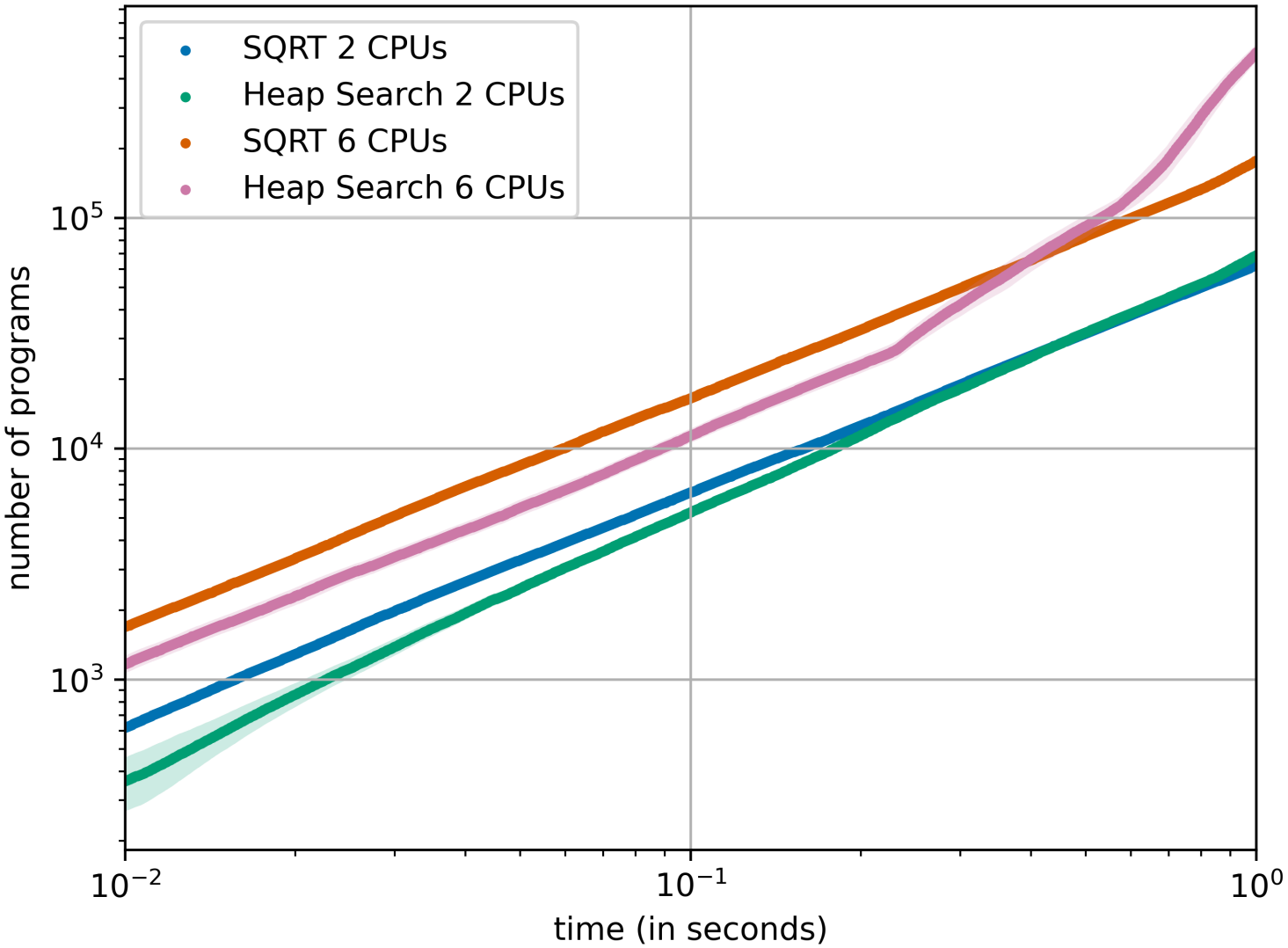
Cumulative probability against number of program output



Cumulative probability against time in log-scale



Comparing all search algorithms on the DreamCoder reduced dataset with machine-learned PCFGs



Parallel implementations of HEAP SEARCH and SQRT SAMPLING using the grammar splitter

Our contributions:

- A theoretical framework called distribution-based search for evaluating and comparing search algorithms in the context of machine-learned predictions.
- Two new search algorithms: Heap Search, an enumerative method, and SQRT Sampling, a probabilistic method. We prove a number of theoretical results about them, in particular that they are both loss optimal.
- A method for running any search algorithm across parallel computing environments.

Code: github.com/nathanael-fijalkow/DeepSynth