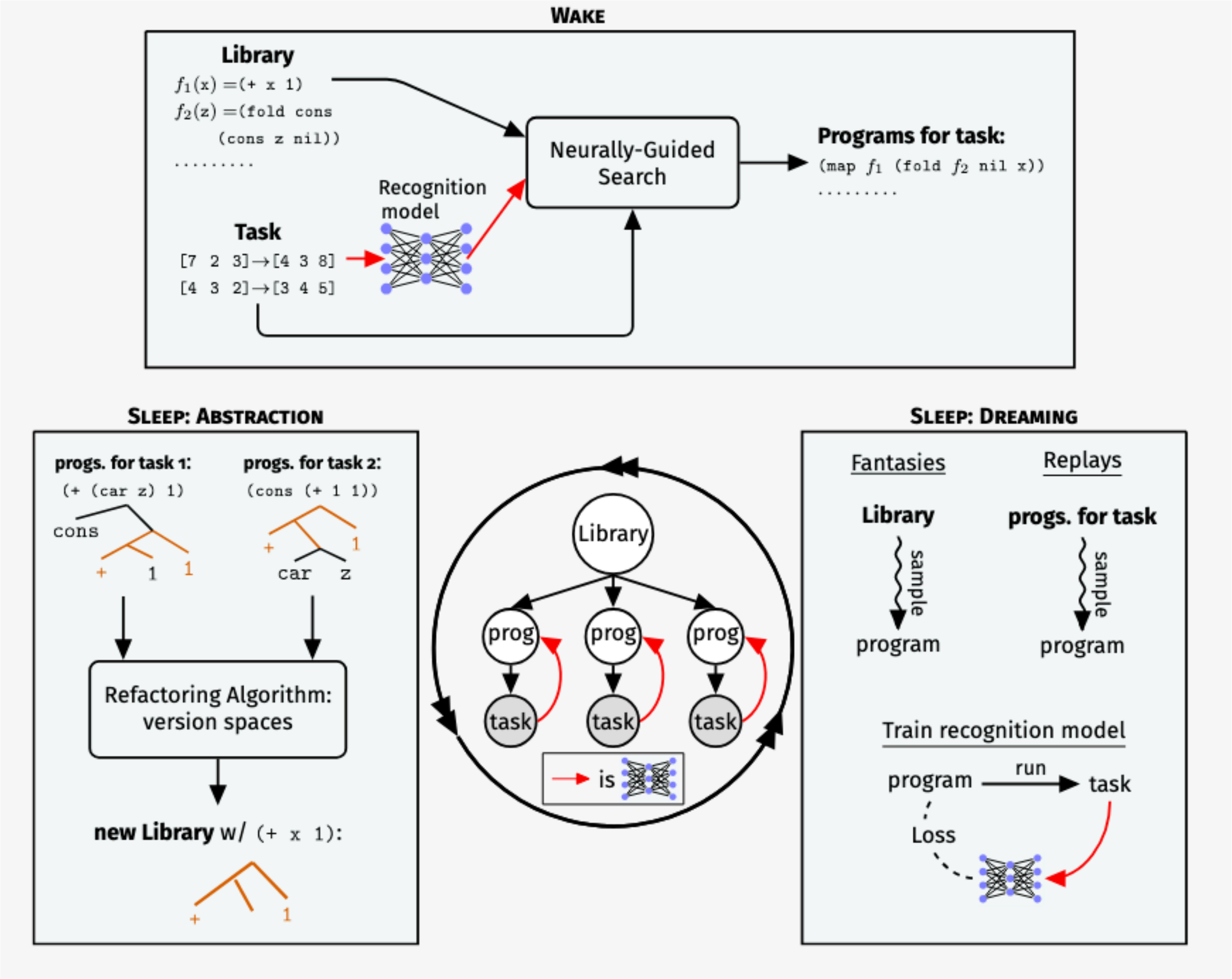
CS5733 Program Synthesis

#22. Neural and NS Synthesis : Take 2

Ashish Mishra, November 4, 2024

With material from Nadia Polikarpova and Armando
-Solar

DreamCoder.



WAKE
Library
f1(x) =(+ x 1)
f2(z) =(fold cons (cons z nil))
....
Task
[7 2 3]→[4 3 8]
[4 3 2]→[3 4 5]
Recognition model
Neurally-Guided Search
Programs for task:
(map f1 (fold f2 nil x))
....
SLEEP: ABSTRACTION
progs. for task 1:
(+ (car z) 1)
progs. for task 2:
(cons (+ 1 1))
cons
+
1
1
car
z
Refactoring Algorithm: version spaces
new Library w/ (+ x 1):
+
1
Library
prog
prog
prog
task
task
task
is
SLEEP: DREAMING
Fantasies
Replays
Library
sample
program
progs. for task
sample
program
Train recognition model
program
run
task
Loss

Plan for the week

- Today : Pre-LLM Era
  - statistical language models for code
  - neural architectures
  - better search with neural guidance
- Tomorrow : LLM Era
  - synthesis from natural language
  - how can we make LLMs generate better code?

Lessons from NLP

- Learning Complex distributions
- Many techniques from NLP can be brought into to learn Distributions over programs.
  - N-gram Models
  - Recurrent Models
- Sequence of tokens vs. Program Structures
- Searching with a learned distribution.

Statistical Language Models

Originated in Natural Language Processing

In general: a probability distribution over sentences in a language

- $P(s)$ for $s \in L$

In practice:

- must be in a form that can be used to guide generation / search
- and also that can be learned from the data we have

n-gram models

The big brown bear scares the children with its roar

$P(scares \mid bear,\ brown)$

Probability of a word depends on the previous n words

Represented with a table: $P\left(w_i \,\middle|\, w_{i-1},\ w_{i-2},\ \ldots,\ w_{i-n}\right)$

Bigger n makes more accurate, but also more difficult to learn, requires much bigger table

Downsides

- some words require more context than others
- some words carry very little information . E.g roar vs. bear

Statistical Models in Synthesis : Multiple axes

What are we modeling (conditioning)?

- A corpus of programs: what are likely programs in this language / DSL / for this specific task?
- Spec-program pairs: what are likely programs for this spec?

Kinds of guidance:

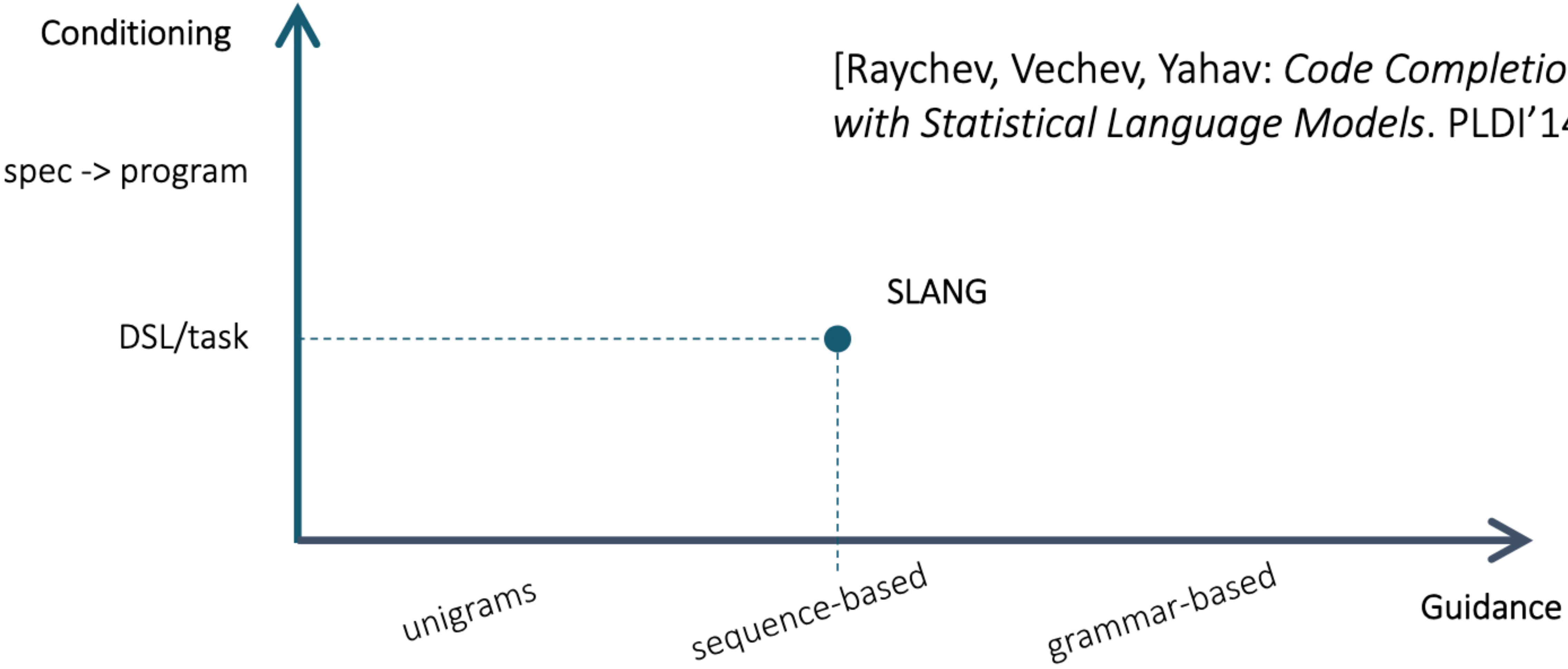
- Likely components (unigrams)
- Sequence-based: probability of next token (given previous tokens)
- Grammar-based: probability of grammar rule

Model architecture:

- n-grams, PHOG, neural, …

Statistical Models in Synthesis

[Raychev, Vechev, Yahav: *Code Completion with Statistical Language Models*. PLDI'14]



SLANG

Input: code snippet with holes

```
SmsManager smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
  ArrayList<String> msgList =
      smsMgr.divideMsg(message);
  ? {smsMgr, msgList}  // (H1)
} else {
  ? {smsMgr, message}  // (H2)
}
```

SLANG

Output: holes completed with (sequences) of method calls

```
SmsManager smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
  ArrayList<String> msgList =
      smsMgr.divideMsg(message);
  smsMgr.sendMultipartTextMessage(...msgList...);
} else {
  smsMgr.sendTextMessage(...message...);
}
```

SLANG

Main Idea:

- Reduce the problem of code completion to a natural-language processing problem of predicting probabilities of sentences.
- A scalable static analysis that extracts sequences of method calls from large codebases, and indexes them into statistical language models such as N-gram and Recurrent Neural Networks.
- A synthesis procedure that takes as input a partial program with holes and leverages probabilities learned in the language model to discover code completions for the holes. Our

SLANG: inference phase

- Sequence of events, generated by tracking for each object o
- Generate Abstract Histories

code snippet with holes

```
SmsManager smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
  ArrayList<String> msgList =
      smsMgr.divideMsg(message);
  ? {smsMgr, msgList}  // (H1)
} else {
  ? {smsMgr, message}  // (H2)
}
```

static analysis

abstract histories of objects

$\mathbf{smsMgr} \mapsto \{\langle \texttt{getDefault}, \texttt{ret}\rangle \cdot \langle \mathbf{H2}\rangle,\ \langle \texttt{getDefault}, \texttt{ret}\rangle \cdot \langle \texttt{divideMsg}, 0\rangle \cdot \langle \mathbf{H1}\rangle\}$

$\mathbf{message} \mapsto \{\langle \texttt{length}, 0\rangle,\ \langle \texttt{length}, 0\rangle \cdot \langle \mathbf{H2}\rangle\}$

$\mathbf{msgList} \mapsto \{\langle \texttt{divideMsg}, \texttt{ret}\rangle \cdot \langle \mathbf{H1}\rangle\}$

learned generative model:
- bigrams suggest candidates
- n-grams / RNNs rank them

Partial History	Id	Candidate Completions	Pr
$\langle \texttt{getDefault}, \texttt{ret}\rangle \cdot \langle \mathbf{H2}, \mathbf{smsMgr}\rangle$	11	$\langle \texttt{getDefault}, \texttt{ret}\rangle \cdot \langle \texttt{sendTextMessage}, 0\rangle$	0.0073
	12	$\langle \texttt{getDefault}, \texttt{ret}\rangle \cdot \langle \texttt{sendMultipartTextMessage}, 0\rangle$	0.0010
$\langle \texttt{getDefault}, \texttt{ret}\rangle \cdot \langle \texttt{divideMsg}, 0\rangle \cdot \langle \mathbf{H1}, \mathbf{smsMgr}\rangle$	21	$\langle \texttt{getDefault}, \texttt{ret}\rangle \cdot \langle \texttt{divideMsg}, 0\rangle \cdot \langle \texttt{sendMultipartTextMessage}, 0\rangle$	0.0033
	22	$\langle \texttt{getDefault}, \texttt{ret}\rangle \cdot \langle \texttt{divideMsg}, 0\rangle \cdot \langle \texttt{sendTextMessage}, 0\rangle$	0.0016
$\langle \texttt{length}, 0\rangle \cdot \langle \mathbf{H2}, \mathbf{message}\rangle$	31	$\langle \texttt{length}, 0\rangle \cdot \langle \texttt{length}, 0\rangle$	0.0132
	32	$\langle \texttt{length}, 0\rangle \cdot \langle \texttt{split}, 0\rangle$	0.0080
	33	$\langle \texttt{length}, 0\rangle \cdot \langle \texttt{sendTextMessage}, 3\rangle$	0.0017
	34	$\langle \texttt{length}, 0\rangle \cdot \langle \texttt{sendMultipartTextMessage}, 1\rangle$	0.0001
$\langle \texttt{divideMsg}, \texttt{ret}\rangle \cdot \langle \mathbf{H1}, \mathbf{msgList}\rangle$	41	$\langle \texttt{divideMsg}, \texttt{ret}\rangle \cdot \langle \texttt{sendMultipartTextMessage}, 3\rangle$	0.0821

SLANG

Predicts completions for sequences of API calls

Treats programs as (sets of) abstract histories

- Performs static analysis to abstract programs into finite histories

Training: learns bigrams, n-grams, RNNs on histories

Inference: given a history with holes

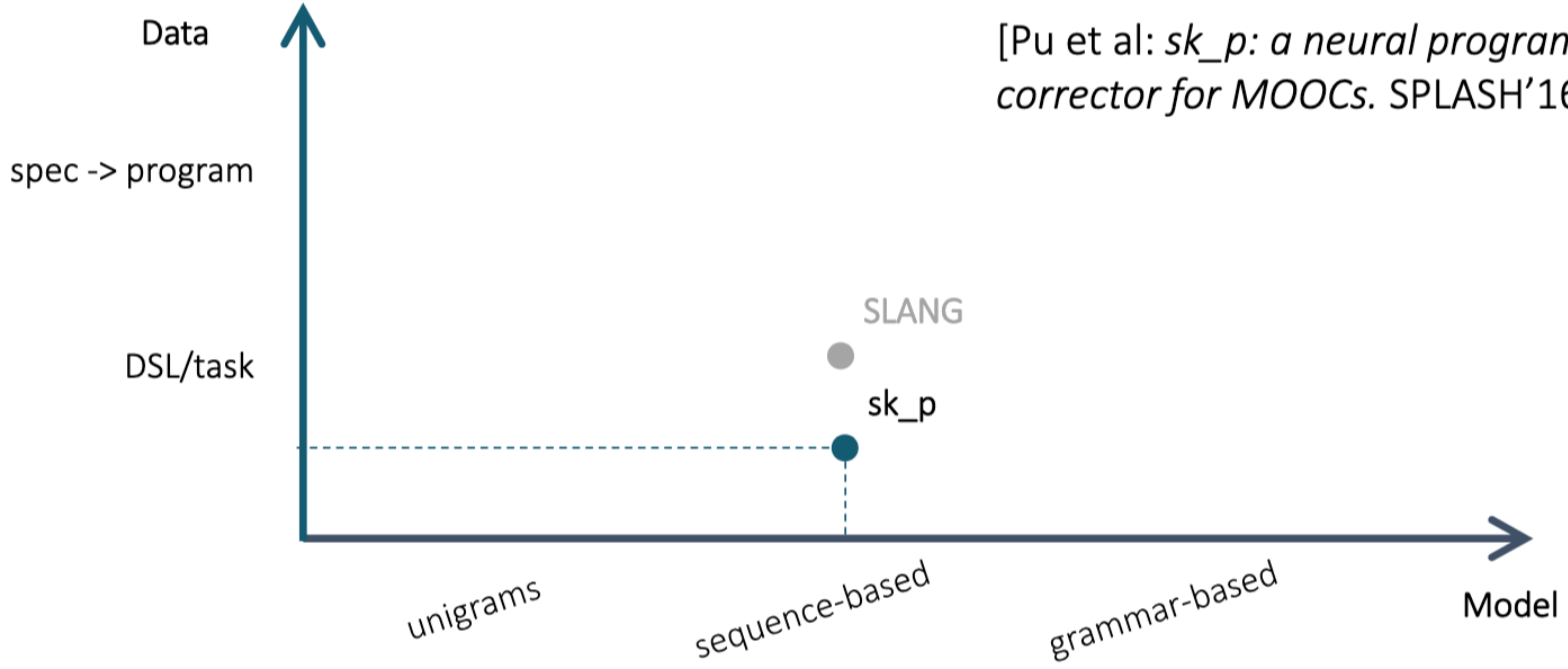
- Uses bigrams to get possible completions
- Uses n-grams / RNN to rank them
- Combines history completions into a coherent program

Features: fast (very little search)

Limitations: all invocation pairs must appear in training set

Statistical Models in Synthesis

[Pu et al: *sk_p: a neural program corrector for MOOCs.* SPLASH'16]



sk_p

A Data-driven Synthesis approach

Input: incorrect program
+ test suite

```
def evaluatePoly(poly, x):
  a = 0
  f = 0.0
  for a in range(0, len(poly) - 1):
    f = poly[a]*x**a+f
    a += 1
  return f
```

sk_p

Output: corrected program

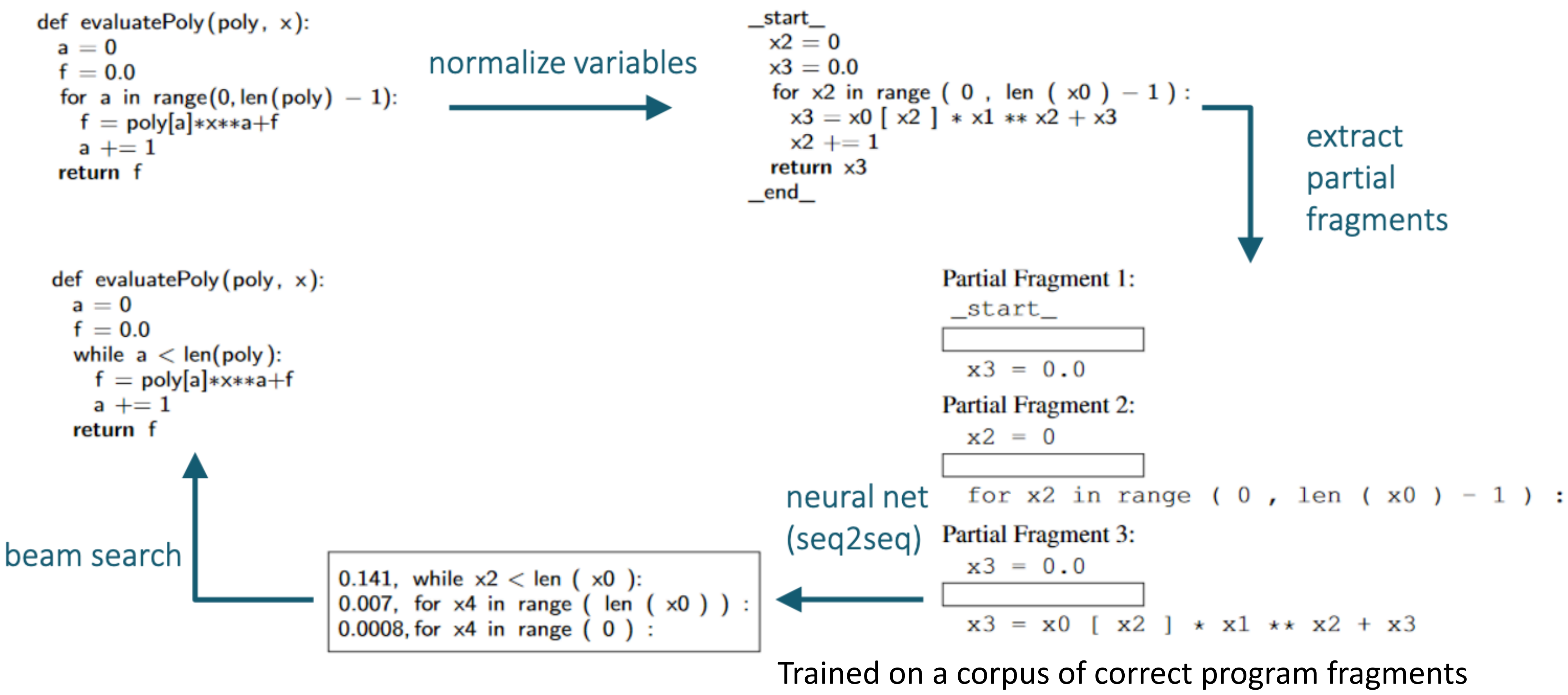
```
def evaluatePoly(poly, x):
  a = 0
  f = 0.0
  while a < len(poly):
    f = poly[a]*x**a+f
    a += 1
  return f
```

sk_p : A Data-driven Synthesis approach for MOOC

Main Idea:

- A learning algorithm is used during training time to produce a model of the problem at hand.
- Given an incomplete or erroneous program (the seed program), this model can produce a distribution of candidate completions or corrections.
- This distribution is used by a synthesis algorithm to find candidate solutions that have high probability according to the model and also are correct according to a potentially incomplete specification.

sk_p


def evaluatePoly(poly, x):
a = 0
f = 0.0
for a in range(0, len(poly) − 1):
f = poly[a]∗x∗∗a+f
a += 1
return f
normalize variables
start
x2 = 0
x3 = 0.0
for x2 in range (0 , len (x0) − 1) :
x3 = x0 [x2] ∗ x1 ∗∗ x2 + x3
x2 += 1
return x3
end
extract partial fragments
Partial Fragment 1:
start
x3 = 0.0
Partial Fragment 2:
x2 = 0
for x2 in range (0 , len (x0) − 1) :
Partial Fragment 3:
x3 = 0.0
x3 = x0 [x2] ∗ x1 ∗∗ x2 + x3
neural net (seq2seq)
0.141, while x2 < len (x0):
0.007, for x4 in range (len (x0)) :
0.0008, for x4 in range (0) :
beam search
def evaluatePoly(poly, x):
a = 0
f = 0.0
while a < len(poly):
f = poly[a]∗x∗∗a+f
a += 1
return f
Trained on a corpus of correct program fragments

Training

- Each correct fragment is converted to an input-output training pair:
    - The partial fragment (with a hole) is the input, and the missing statement is the output.

Example Training Input:

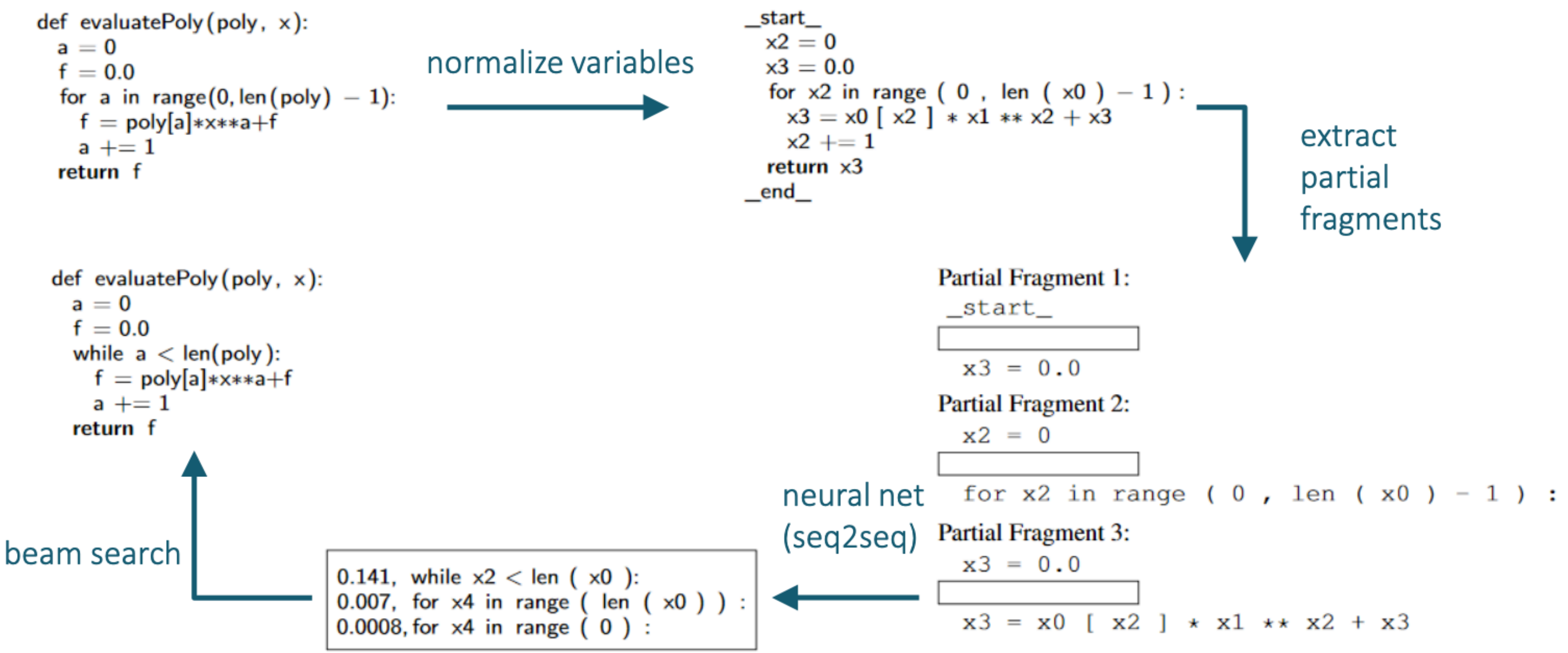
```
else:

  x2 += x0[x3] * (x1 ** x3)
```

Example Training Output:

```
while x3 < len ( x0 ) :
```

sk_p



Trained on a corpus of correct program fragments

sk_p

Program corrections for MOOCs

Treats programs as a sequence of tokens

- Abstracts away variables names

Uses the skipgram model to predict which statement is most likely to occur between the two

Features

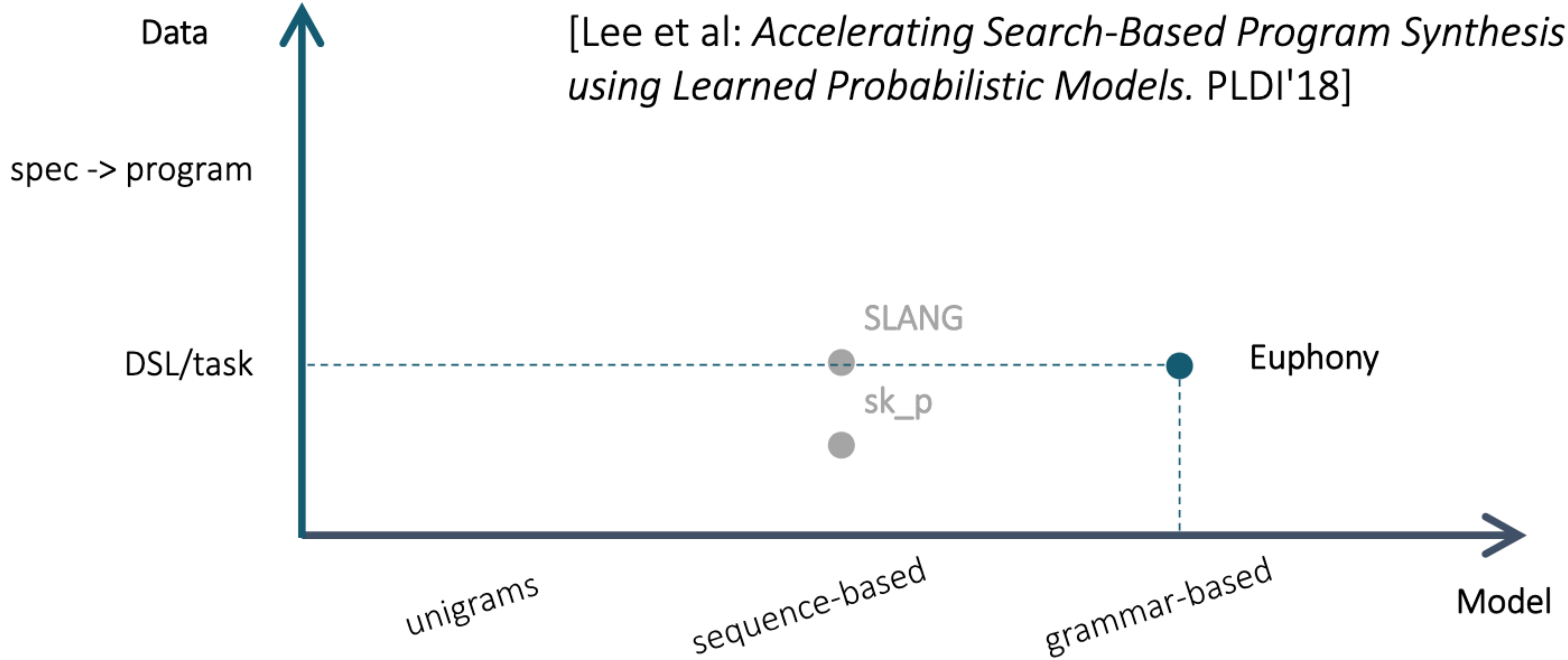
- Can repair syntax errors

Limitations

- Needs all algorithmically distinct solutions to appear in the training set

Statistical Models in Synthesis

[Lee et al: *Accelerating Search-Based Program Synthesis using Learned Probabilistic Models.* PLDI'18]



Euphony

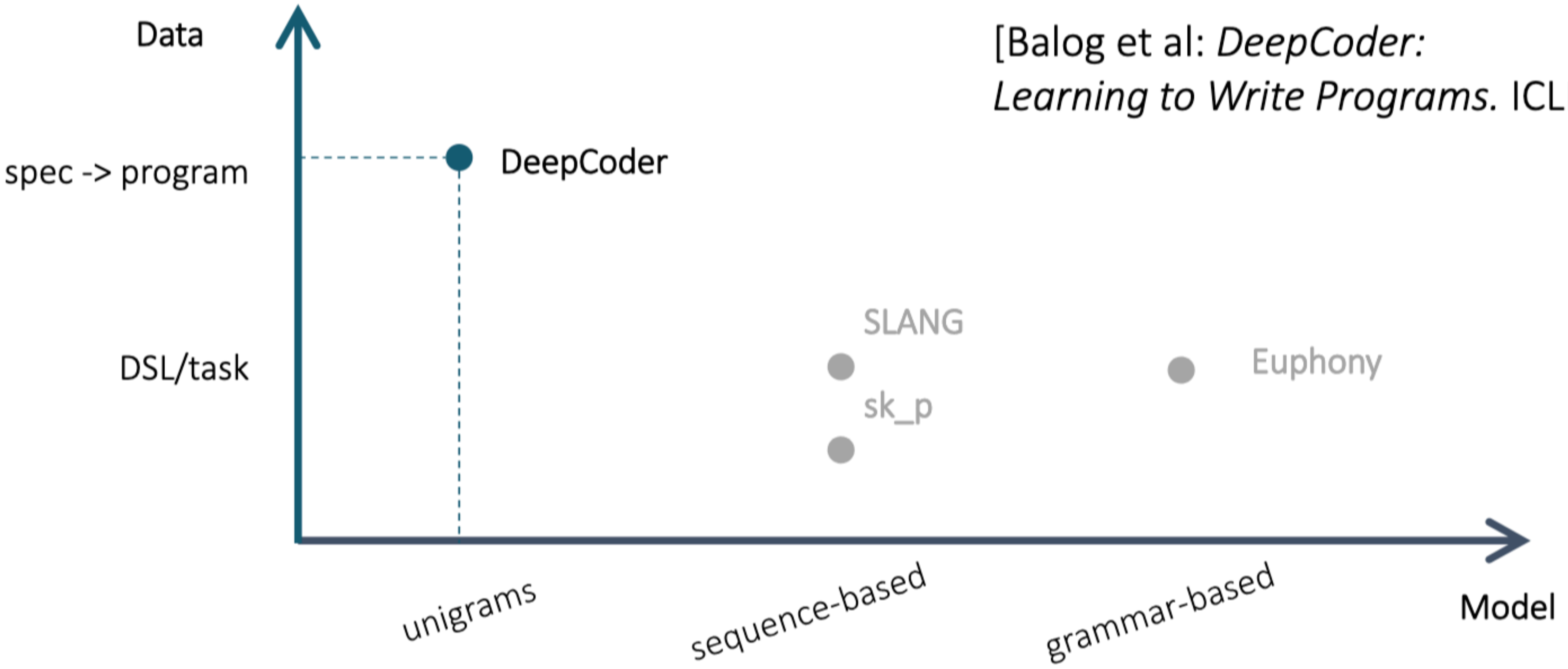
Trains a PHOG on a corpus of solutions to simple problems

Uses it to guide top-down search with A*

Normalizes constants (transfer learning)

Statistical Models in Synthesis

[Balog et al: *DeepCoder: Learning to Write Programs.* ICLR'17]



Learning Inductive Program Synthesis (LIPS)

- DSL and Attributes
    - An attribute function A: Program P in DSL -> Finite Attribute Vectors A (P).
        - E.g. Presence or absence of HOFs, like does the program contain sort
    - Attributes are a link between ML and Search.
        - ML predicts q (A(P) | Observations)
- Data Generation: Synthetic data generation in DSL
- ML Model
- Search

DeepCoder

- An Instance of LIPS
- DSL and Attributes:
  - Attributes: binary attributes indicating the presence or absence of high-level functions in the target program. To
  - DSL : A query language like SQL or LINQ using High-level functions over lists.

a ← [int]	**An input-output example:**
b ← FILTER (<0) a	*Input*:
c ← MAP (*4) b	[-17, -3, 4, 11, 0, -5, -9, 13, 6, 6, -8, 11]
d ← SORT c	*Output*:
e ← REVERSE d	[-12, -20, -32, -36, -68]

DeepCoder

- Data Generation
  - Enumerate Programs in DSL and Pruning.
  - To generate valid inputs for a program, they enforce a constraint on the output value bounding integers to some predetermined range.
- ML Model
  - Employs Encoder-Decoder NNs to model and learn the mapping from input-output examples to attributes.
  - learns to predict presence or absence of individual functions of the DSL.
- Search
  - DFS, Sketch and λ^2

DeepCoder

Input: IO-examples

```
[-17 -3 4 11 0 -5 -9 13 6 6 -8 11]
→ [-12 -20 -32 -36 -68]
```

DeepCoder

Output: Program in a list DSL

```
a <- [int]
b <- Filter (<0) a
c <- Map (*4) b
d <- Sort c
e <- Reverse d
```

DeepCoder

Input: IO-examples

```
[-17 -3 4 11 0 -5 -9 13 6 6 -8 11]
→ [-12 -20 -32 -36 -68]
```

neural network

component likelihoods

(+1)	(-1)	(*2)	(/2)	(*-1)	(**2)	(*3)	(/3)	(*4)	(/4)	(>0)	(>0)	(%2==1)	(%2==0)	HEAD	LAST	MAP	FILTER
.0	.0	.1	.0	.0	.0	.0	.0	1.0	.0	.0	1.0	.0	.2	.0	.0	1.0	1.0

SORT	REVERSE	TAKE	DROP	ACCESS	ZIPWITH	SCANL1	+	-	*	MIN	MAX	COUNT	MINIMUM	MAXIMUM	SUM
1.0	.7	.0	.1	.0	.4	.0	.0	.1	.0	.2	.1	.0	.0	.0	.0

existing search technique + sort-and-add

Output: Program in a list DSL

DeepCoder

Predicts likely components from IO examples

Features

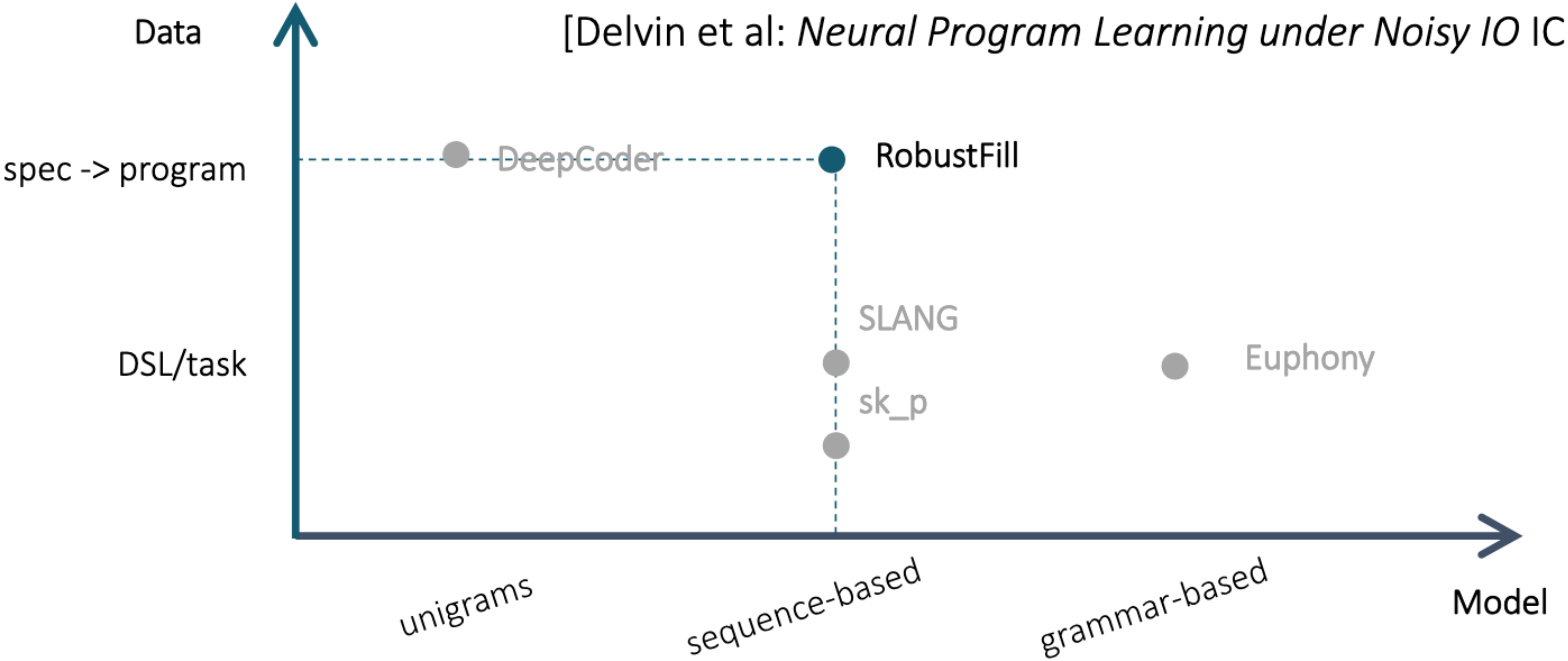
- Trained on synthetic data
- Can be easily combined with any enumerative search
- Significant speedups for a small list DSL

Limitations

- Unclear whether it scales to larger DSLs or more complex data structures
- e.g. uses a simple feed-forward neural net, cannot encode arbitrary-length examples

Statistical Models in Synthesis

[Delvin et al: *Neural Program Learning under Noisy IO* ICML'17]



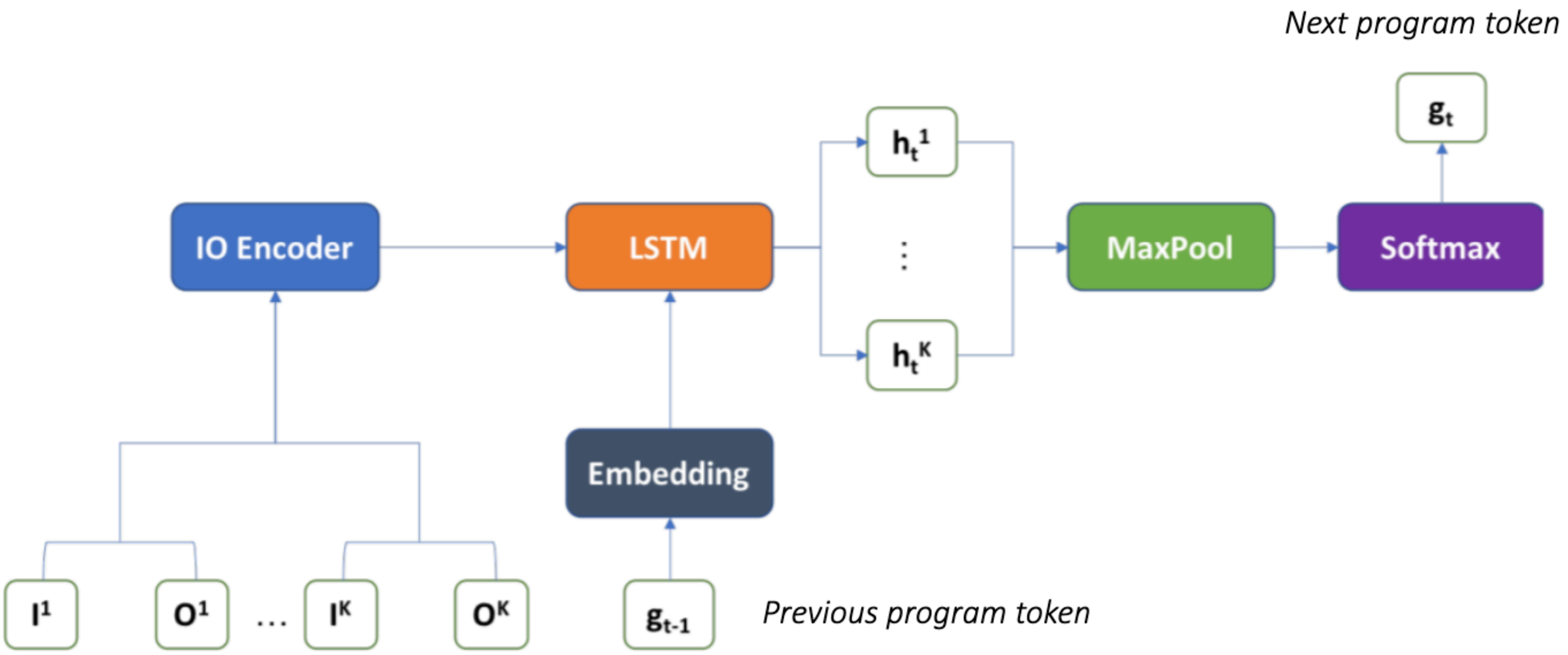
RobustFill, aka neural FlashFill

Input String	Output String
jacob daniel devlin	Devlin, J.
jonathan uesato	Useato, J
Surya Bhupatiraju	Bhupatiraju S.
Rishabh q. singh	Singh, R.
abdelrahman mohamed	Mohamed, A.
pushmeet kohli	Kohli, P.

RobustFill

```
Concat(
   ToCase(
       GetToken(
           input,
           Type=Word,
           Index=-1),
       Type=Proper),
   Const(", "),
   ToCase(
       SubString(
         GetToken(
             input,
             Type=Word,
             Index=1),
         Start=0,
         End=1),
       Type=Proper),
   Const("."))
```

RobustFill: PBE as Seq2Seq with Attention



Each sequence is encoded with a non-attentional LSTM

- final hidden state is used as the initial hidden state of the next LSTM.

Attention

Key idea: Summarizing into a single vector is a big bottleneck. Every output should have direct access to the whole input

Exploit some degree of locality:

- different tokens of the output depend primarily on small subsets of tokens from the input.
- attention mechanism allows each output token to pay attention to a different subset of input tokens.

RobustFill

Key Idea: use attention within an individual input/output pair, but then aggregate over the distributions proposed from each of the examples.

```
in: "Armando Solar-Lezama"
out: "A. Solar-Lezama"
Program: Concat(SubString(in, Pos("", Word), Pos(Char,"")),
". ", SubString(in, Pos(" ",Word), Pos("", End));
```

Three Parts: an expression that extracts the first initial,
concatenated with a constant,
an expression that extracts everything after the first space

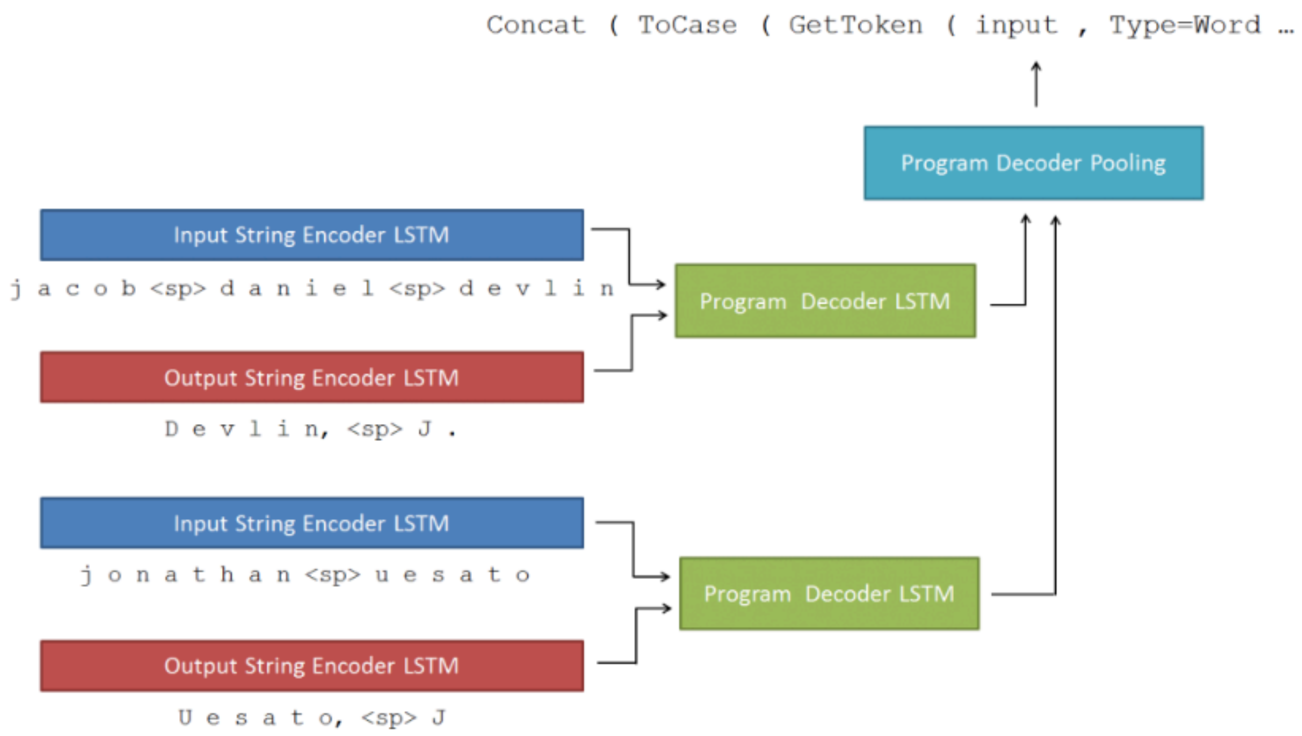
RobustFill

Key ideas:

Embed I/O examples with LSTM encoders

Emit program tokens with LSTM decoders

Train from large-scale random data



RobustFill

Key ideas:

Embed I/O examples with LSTM encoders

Emit program tokens with LSTM decoders

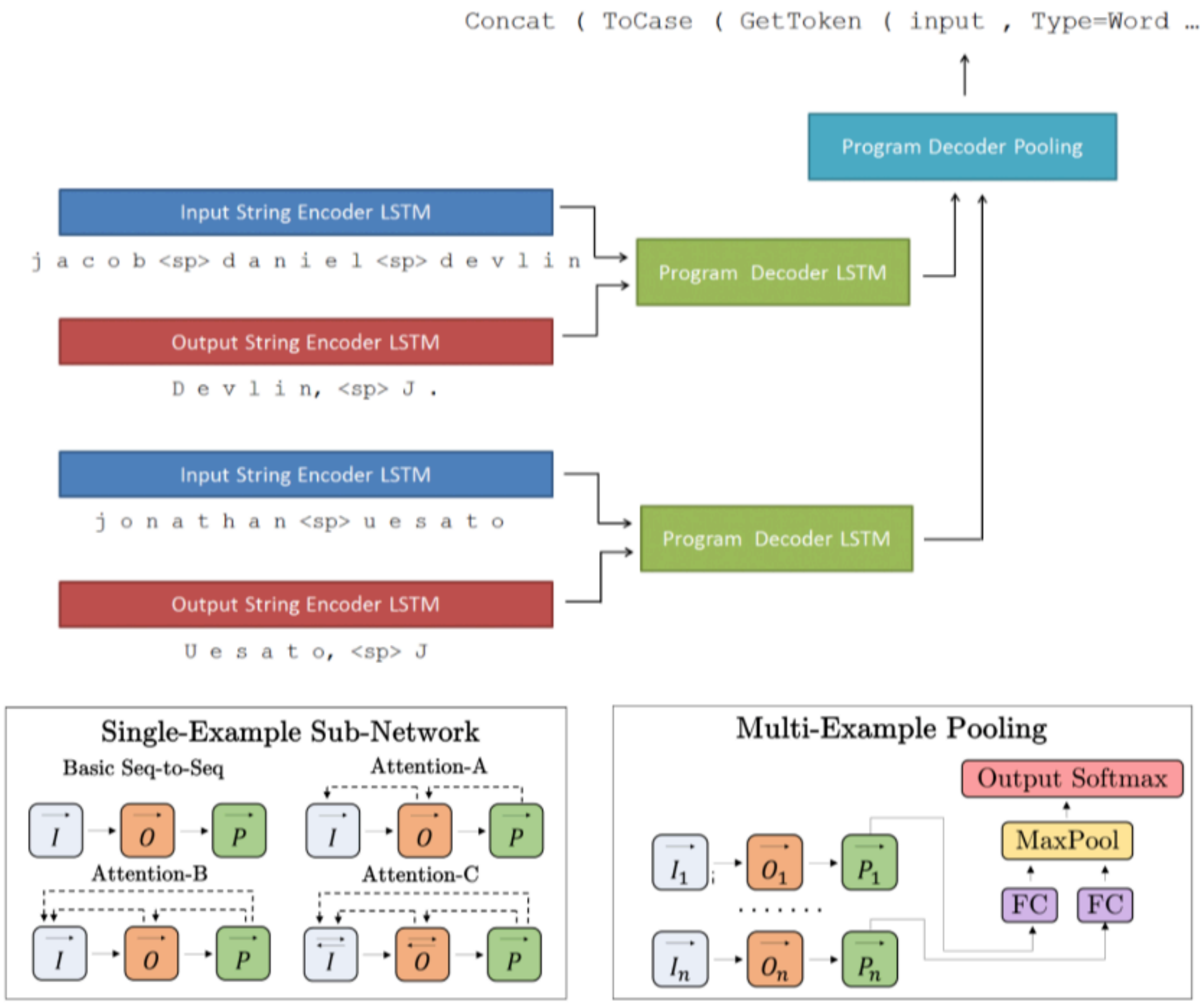
Train from large-scale random data

Architecture:

- *Pooling* across examples at each step to predict one program token
- *Attention* to examples during program decoding

Beam search with *execution constraints*

- Execute decoded subexpressions; remove programs whose outputs are not prefixes of the target



RobustFill

IO examples to program translation as a Seq2Seq task

Features

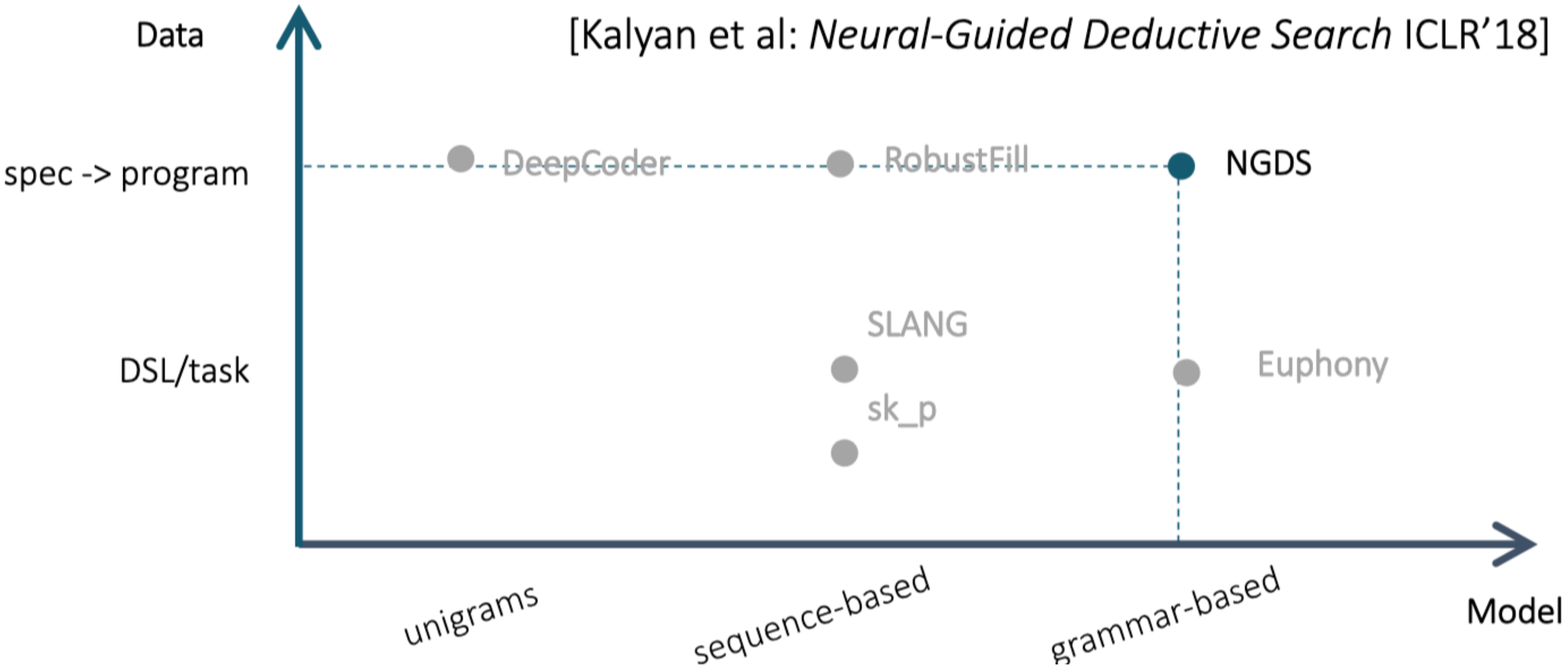
- Trained on synthetic data
- Unlike FlashFill, does not require inverse semantics

Limitations

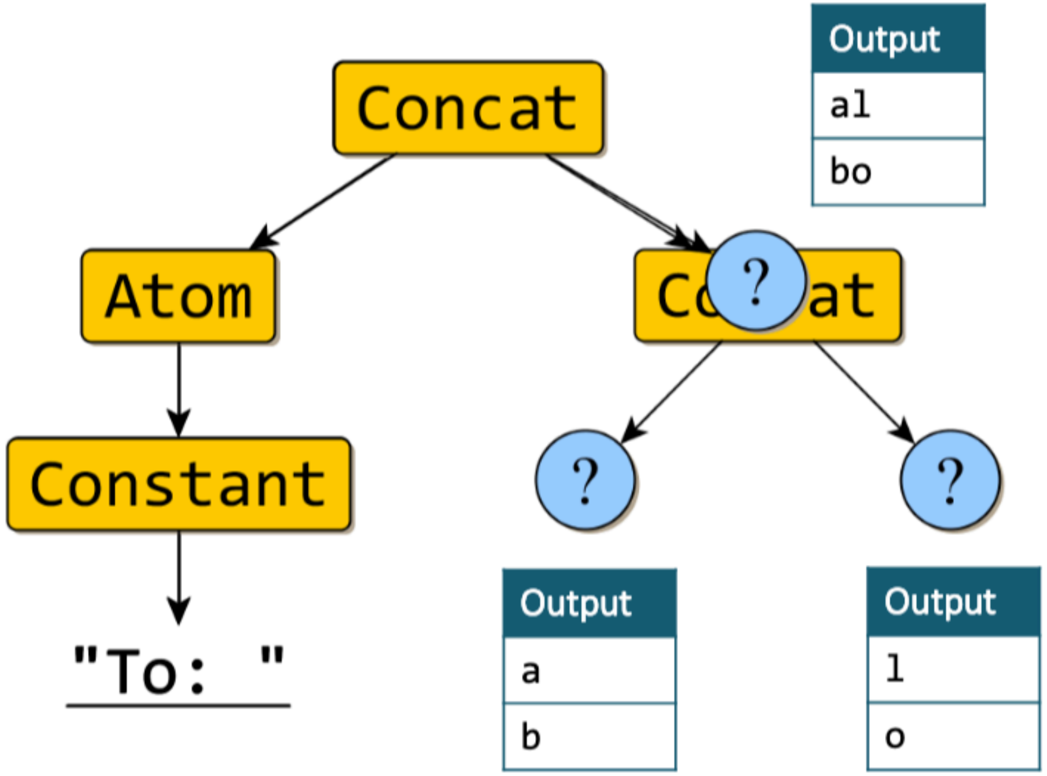
- Does not guarantee consistency with IO examples
- Requires constraints/postprocessing to ensure grammar syntax
- Hard to design synthetic data generation realistically

Statistical Models in Synthesis

[Kalyan et al: *Neural-Guided Deductive Search* ICLR'18]



Deductive Search



Input	Output
alice liddell	To: al
bob o'reilly	To: bo

1. Select a hole.
2. Select an operator to expand.
3. Propagate the examples.

✓ Correct by construction

✓ Constraint propagation exists for many operations & domains

✓ Easy to add a ranking function

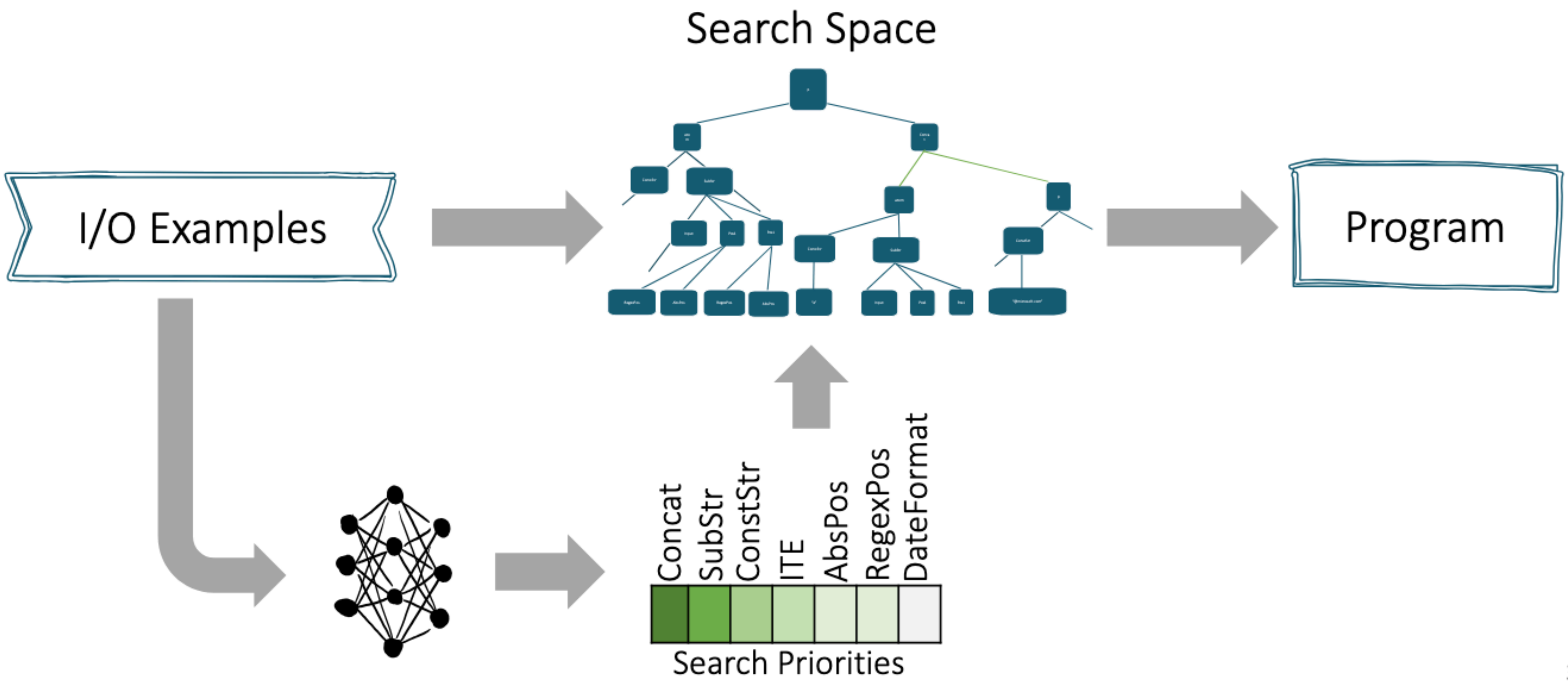
✗ Exponentially slow

Deductive Search

Why so slow? Explores the entire search space
(unless deduction prunes some of it)

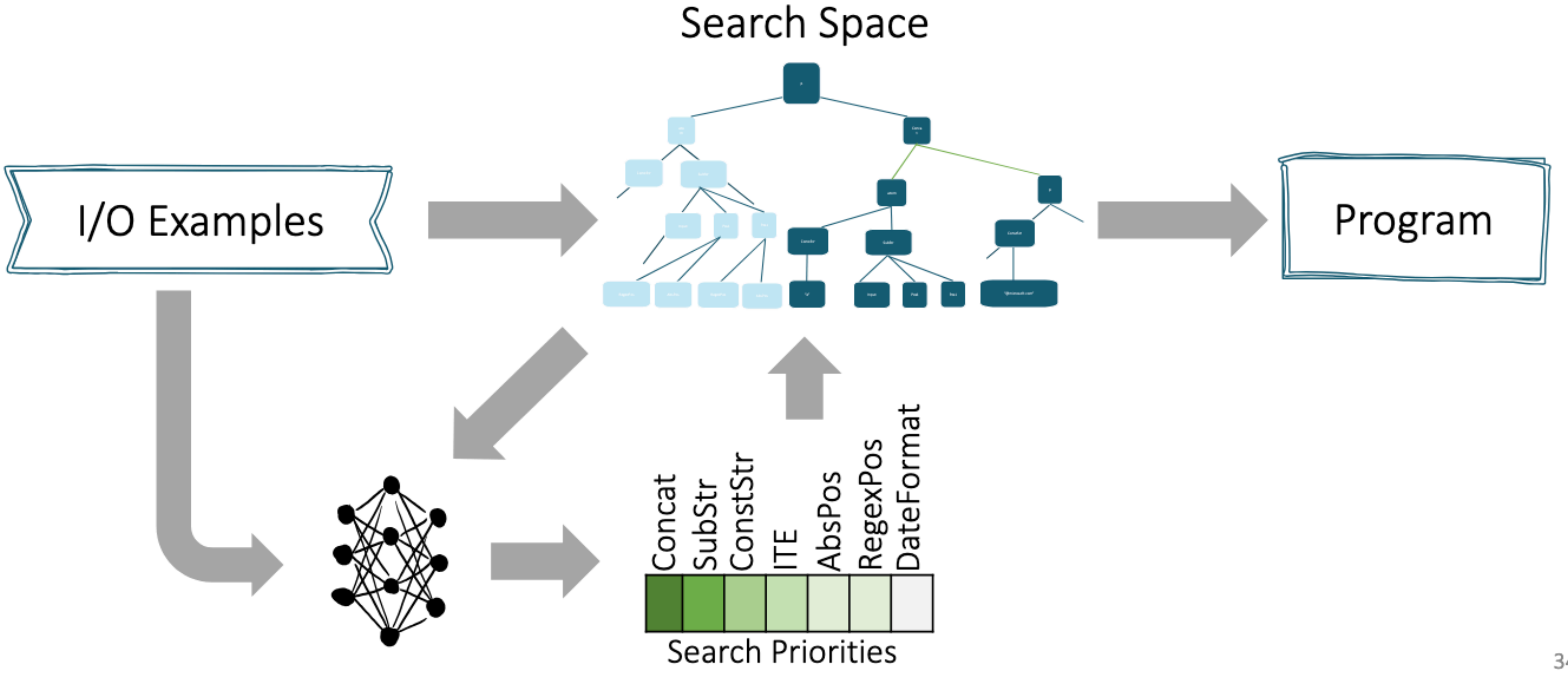
DeepCoder: Learning to Write Programs

Idea: Order the search space based on a priority list from DNN *before starting*



Neural-Guided Deductive Search

Idea: Order the search space based on a priority list from DNN *at each step*



Search branch prediction

Collect a complete dataset of intermediate search results:

at a search branch $N \coloneqq F_1(\dots) \mid F_2(\dots) \mid \cdots \mid F_k(\dots)$
given a spec $\varphi = \{x \rightsquigarrow y\}$
produced programs $P_1, \dots, P_k$ with scores $h(P_1, \varphi), \dots, h(P_k, \varphi)$

A ranking function **h**

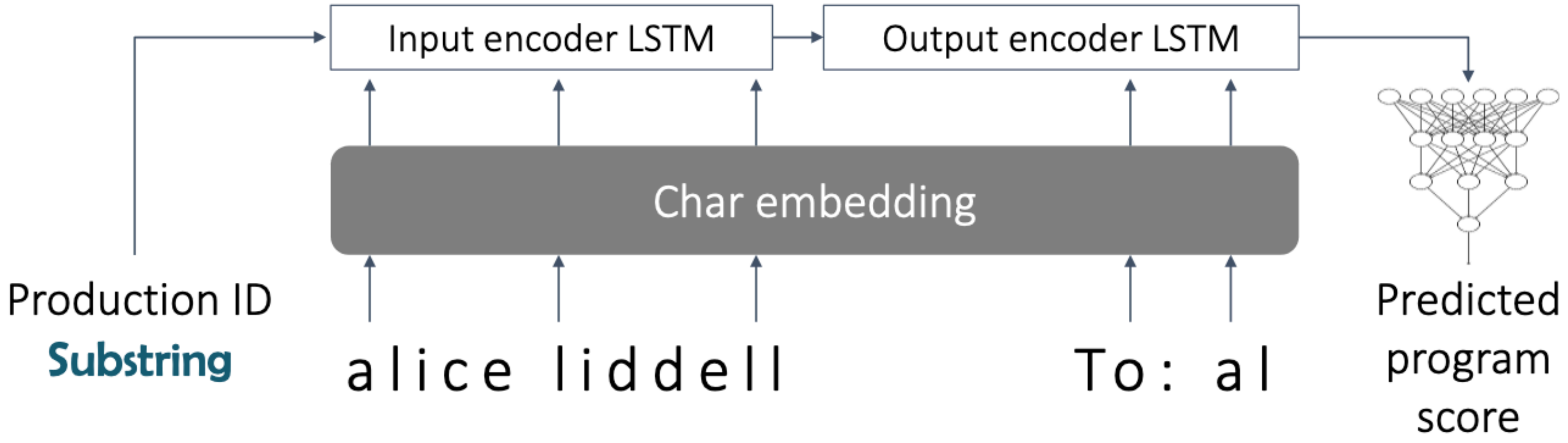
Learn a predictive model f s.t. $f(F_j, \varphi) \approx h(P_j, \varphi)$

- φ is an input-output example spec: $\varphi = \{x \mapsto y\}$
- f: (enum production_id, string x, string y) -> float

Train using squared-error loss over program scores:

Objective: $\mathcal{L}(f;\ F_j, \varphi) = \left[f(F_j, \varphi) - h(P_j, \varphi)\right]^2$

LSTM-based Model for predicting the score



Search

Picking just the topmost rule to expand may be incomplete

Threshold-based

- For a fixed threshold θ
- explore all branches within θ from the best

Branch-and-bound

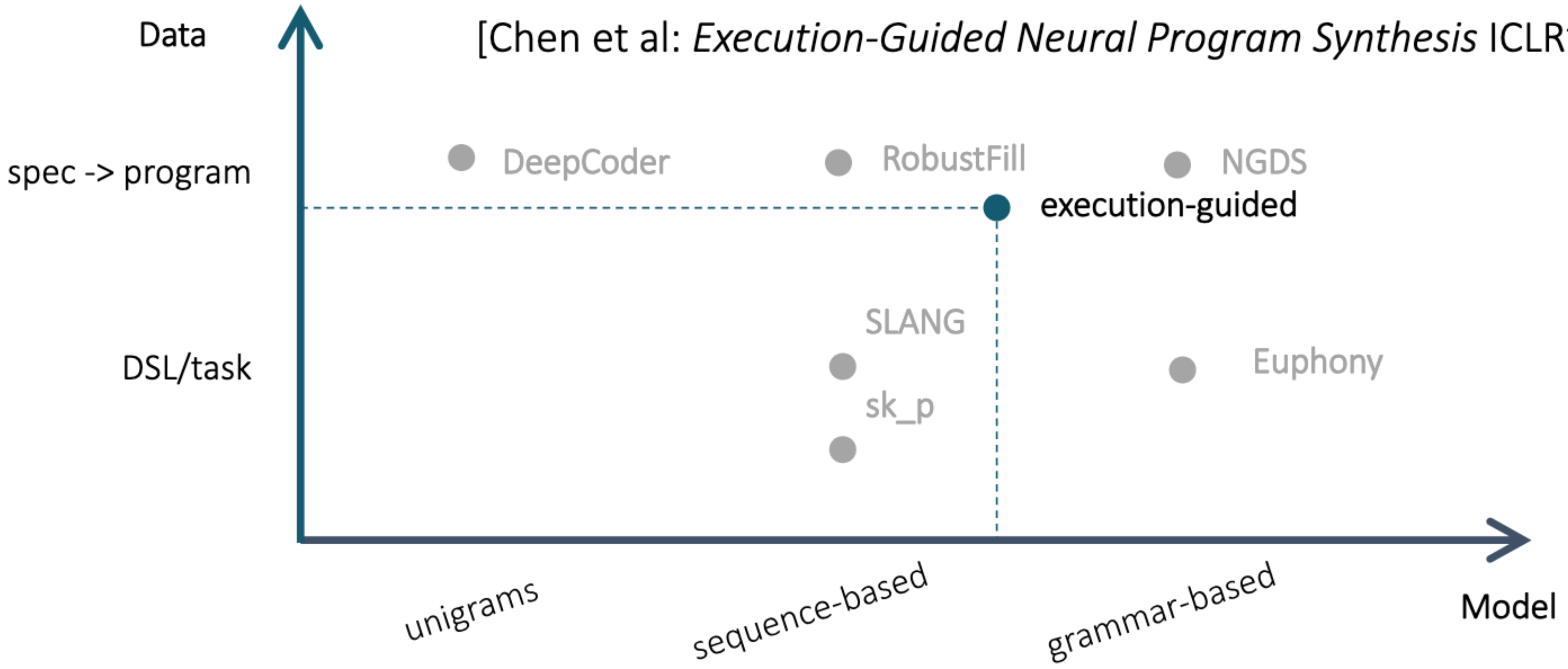
- Explore branches depth-first in the order of scores
- Discard unexplored branches if they are predicted to be worse tha
current optimum

Next Reading

- Kalyan et al: Neural-Guided Deductive Search ICLR'18

Statistical Models in Synthesis

[Chen et al: *Execution-Guided Neural Program Synthesis* ICLR'18]



Takeaways

Neural networks excel at noticing patterns in input data

- don’t expect magic, task must be solvable by a human

Needs appropriate network architecture

- e.g. LSTM for sequential examples, CNN for grids, …

Needs a search algorithm

- A*, branch-and-bound, beam, MCTS, sequential monte-carlo, …

Takeaways (training)

To train a model, you need enough data + appropriate loss

- For NNs: 10-100K diverse data points for an "average" task

How to increase data efficiency?

- abstract the programs (Slang, Skip, Euphony)
- for spec->program can use synthetic data because we are learning semantics, not properties of the corpus (DeepCoder, Robustfill)
- the less context the guidance needs, the more data points we can extract from a given set of programs (NGDS)

Plan for the week

- Today : Pre-LLM Era
  - statistical language models for code
  - neural architectures
  - better search with neural guidance
- Next/Last Class of the session : LLM Era
  - synthesis from natural language
  - how can we make LLMs generate better code?