Lecture 15
Neural and Neuro-Symbolic Synthesis

(with material from Alex Polozov)

Plan for this week

Tuesday: pre-LLM era

- statistical language models for code
- neural architectures
- better search with neural guidance

Thursday: LLM era

- synthesis from natural language
- how can we make LLMs generate better code?

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

Statistical Models in Synthesis

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:

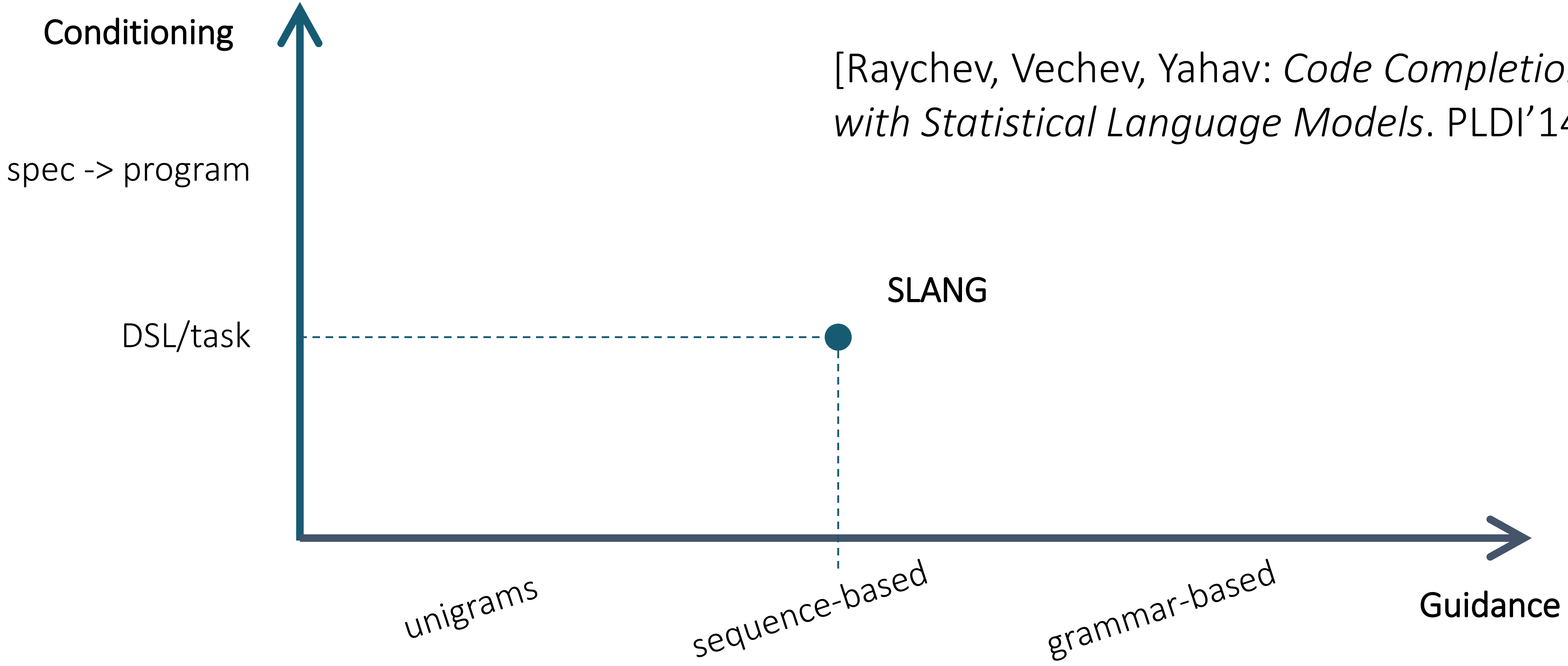
- Likely components (unigrams)
- Sequence-based: probability of next token (given previous tokens)
- Grammar-based: probability of grammar rule

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: inference phase

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

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

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

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

learned generative model:

- bigrams suggest candidates
- n-grams / RNNs rank them

Partial History	Id	Candidate Completions	Pr
$\langle \texttt{getDefault}, \texttt{ret}\rangle \cdot \langle \textbf{H2}, \textbf{smsMgr}\rangle$	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
$\langle \texttt{getDefault}, \texttt{ret}\rangle \cdot \langle \texttt{divideMsg}, 0\rangle \cdot \langle \textbf{H1}, \textbf{smsMgr}\rangle$	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
$\langle \texttt{length}, 0\rangle \cdot \langle \textbf{H2}, \textbf{message}\rangle$	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
$\langle \texttt{divideMsg}, \texttt{ret}\rangle \cdot \langle \textbf{H1}, \textbf{msgList}\rangle$	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

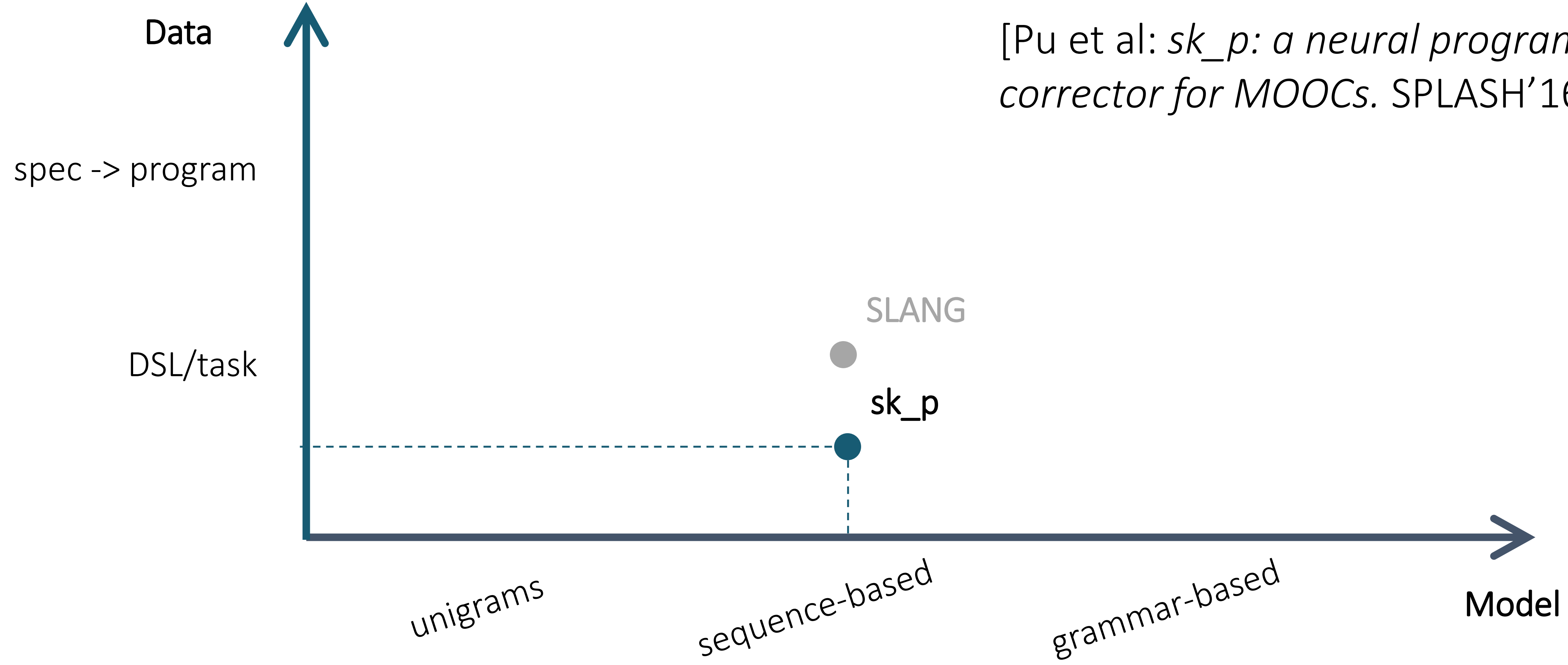
- Uses bigrams to get possible completions
- Uses n-grams / RNN to rank them
- Combines history completions into a coherent program

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

Input: incorrect program
+ test suite

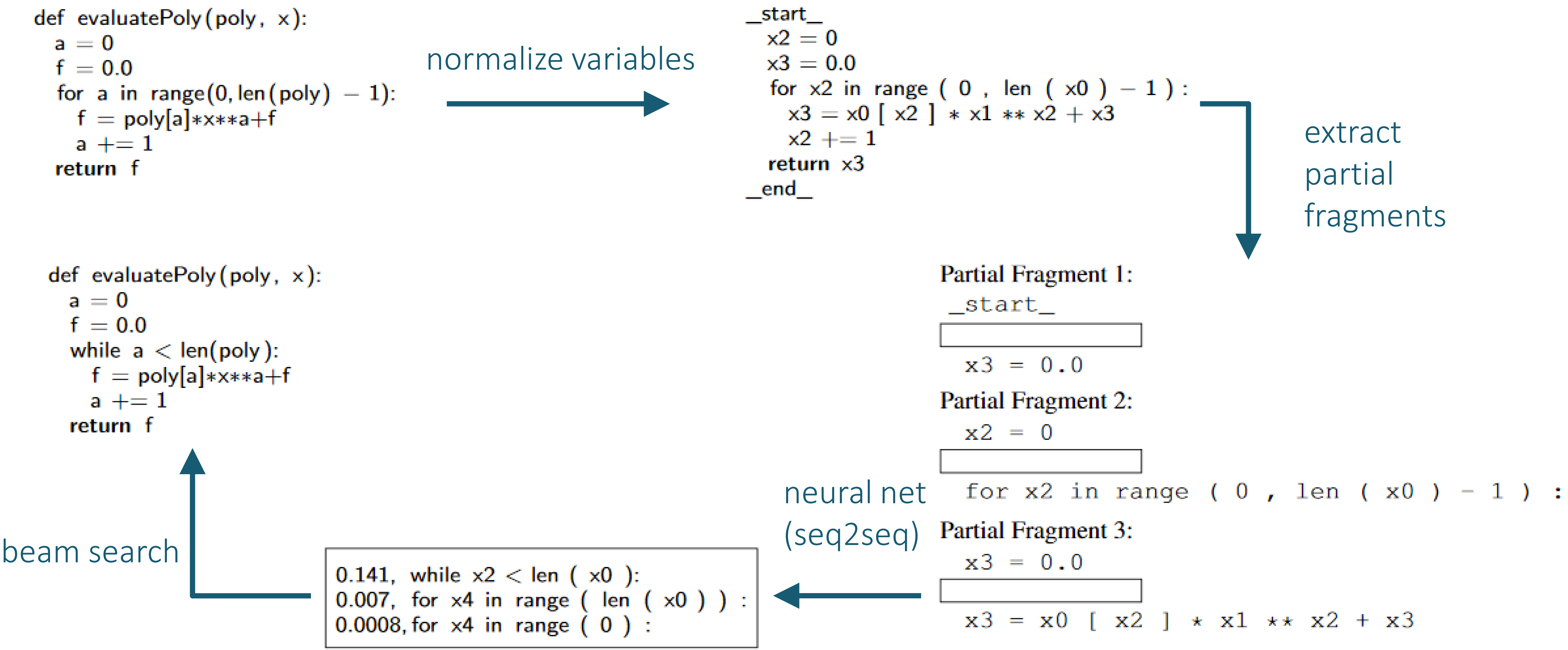
```
def evaluatePoly(poly, x):
  a = 0
  f = 0.0
  for a in range(0, len(poly) - 1):
    f = poly[a]*x**a+f
    a += 1
  return f
```

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


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

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

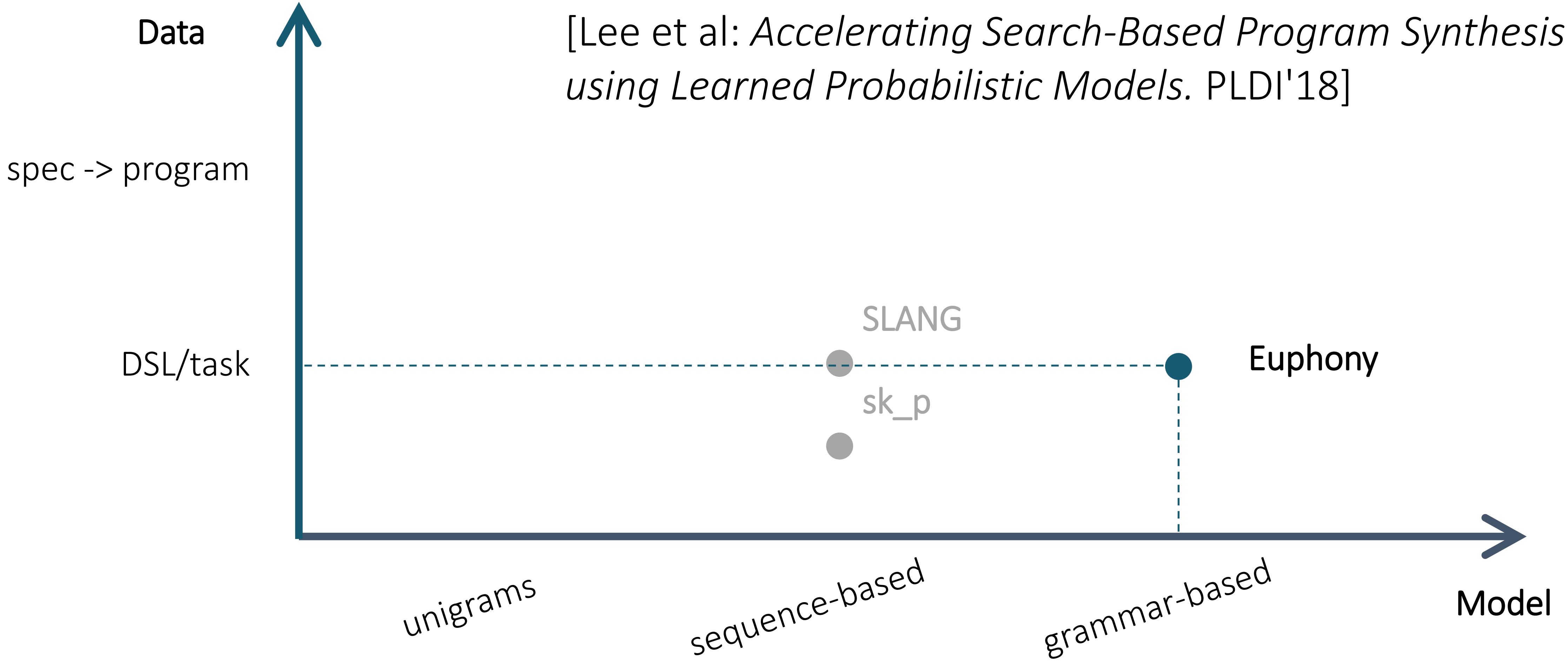
- Can repair syntax errors

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

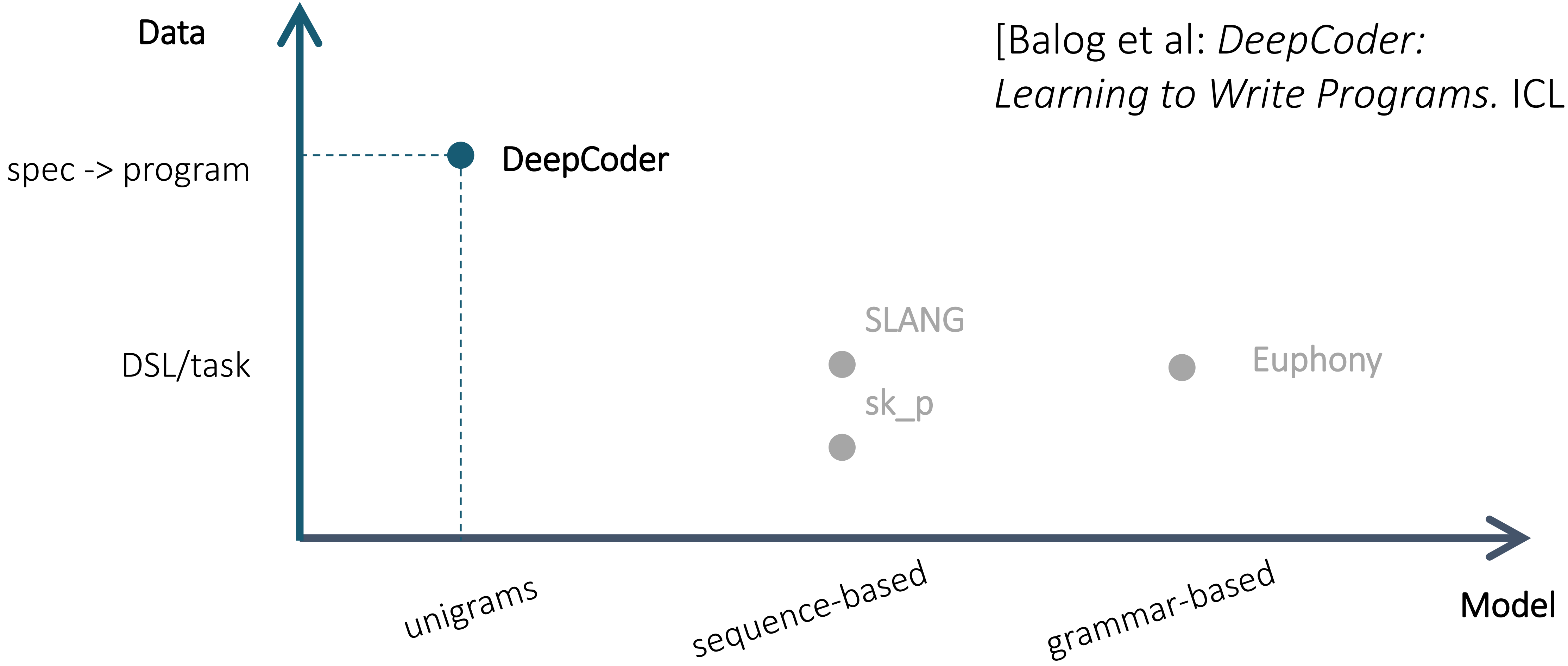
Trains a PHOG on a corpus of solutions to simple problems

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]



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

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

	FILTER	SORT	REVERSE	TAKE	DROP	ACCESS	ZIPWITH	SCANL1	+	-	*	MIN	MAX	COUNT	MINIMUM	MAXIMUM	SUM
component likelihoods	1.0	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

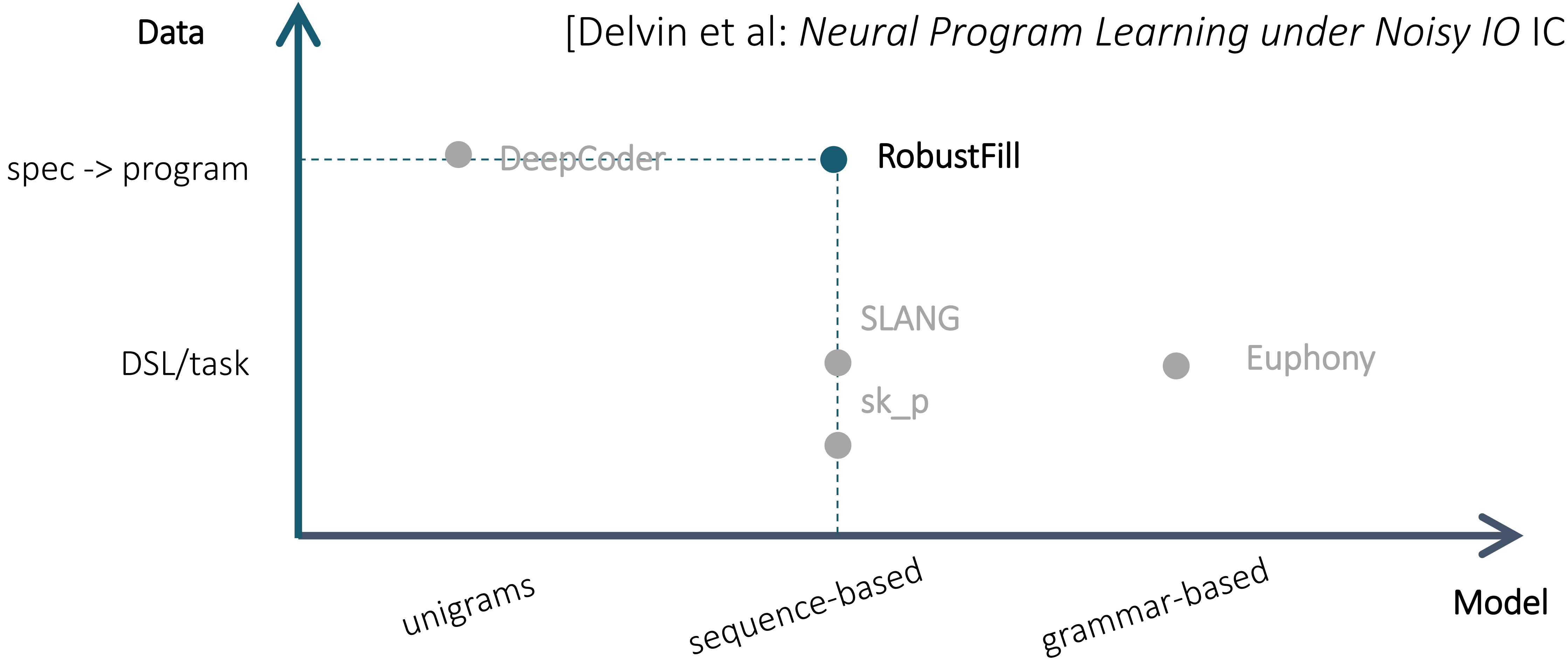
- Trained on synthetic data
- Can be easily combined with any enumerative search
- Significant speedups for a small list DSL

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]



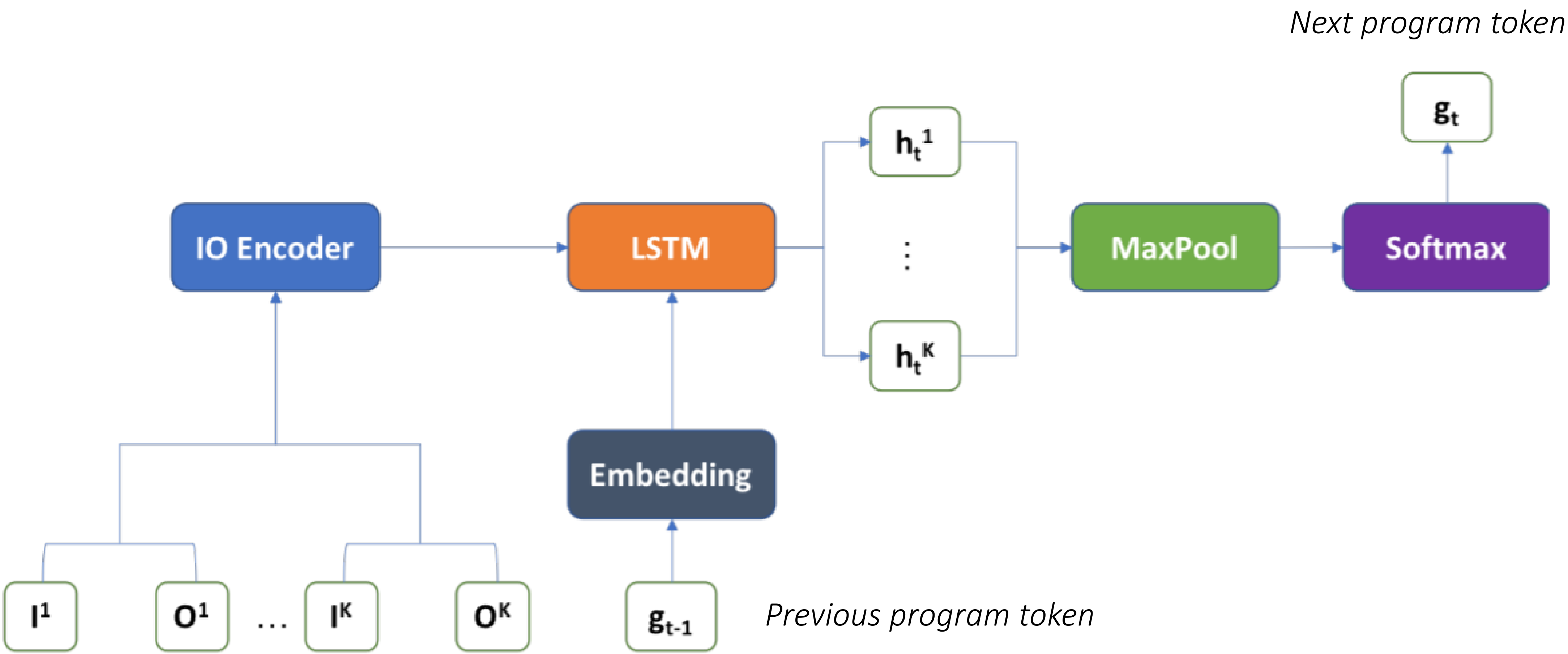
RobustFill, aka neural FlashFill

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("."))
```

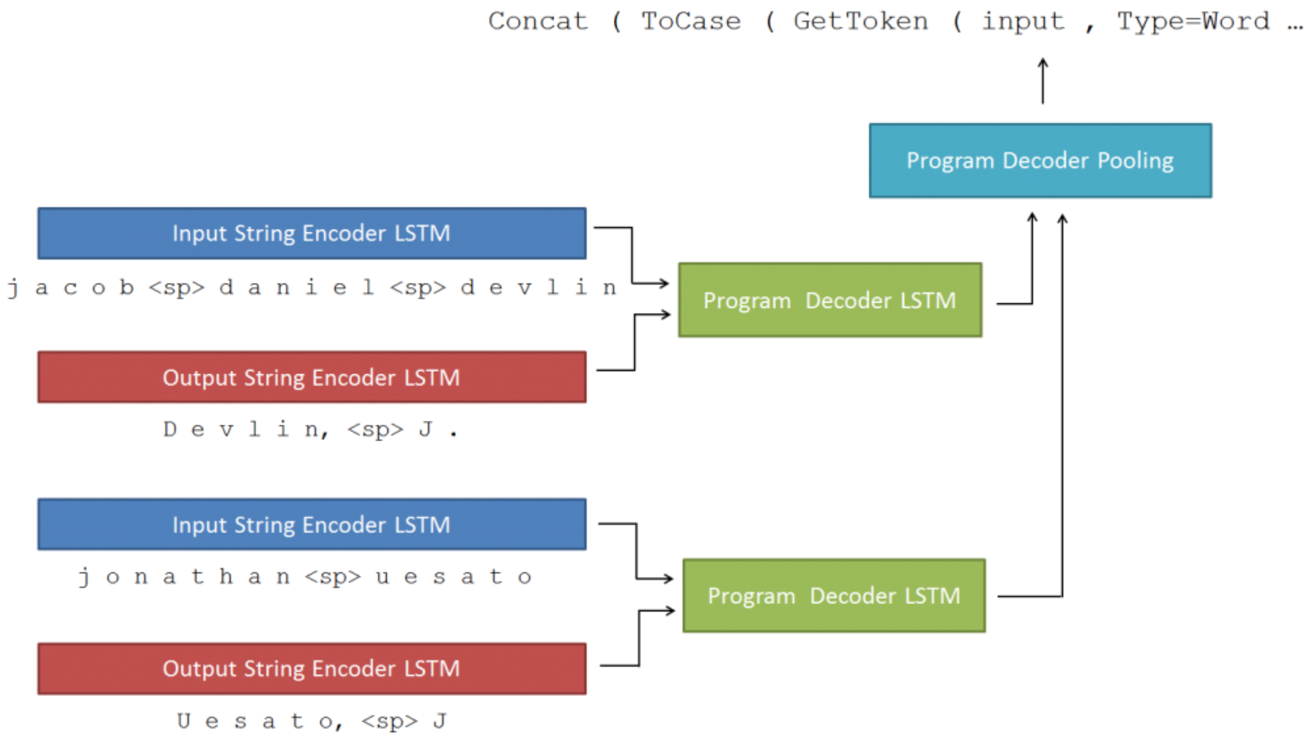
RobustFill: PBE as Seq2Seq



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

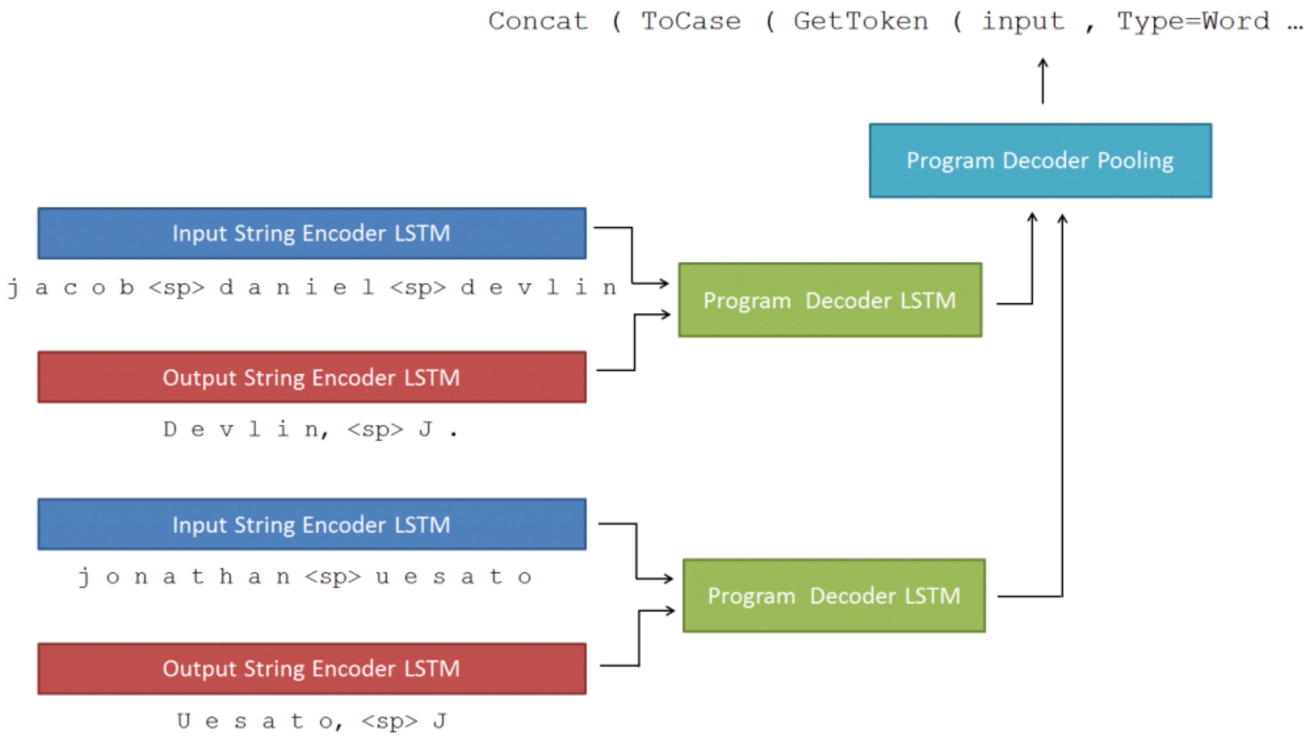
Train from large-scale random data

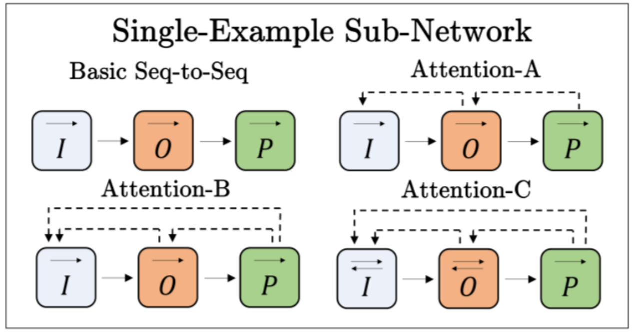
Architecture:

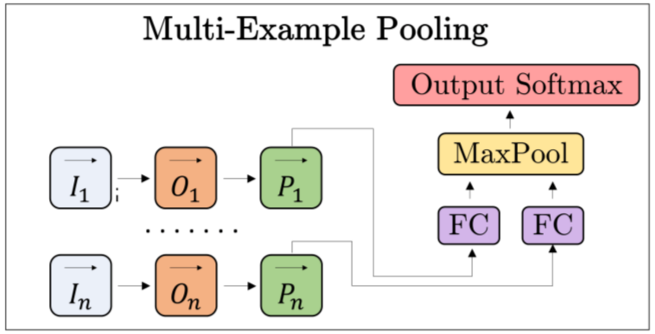
- *Pooling* across examples at each step to predict one program token
- *Attention* to examples during program decoding

Beam search with *execution constraints*

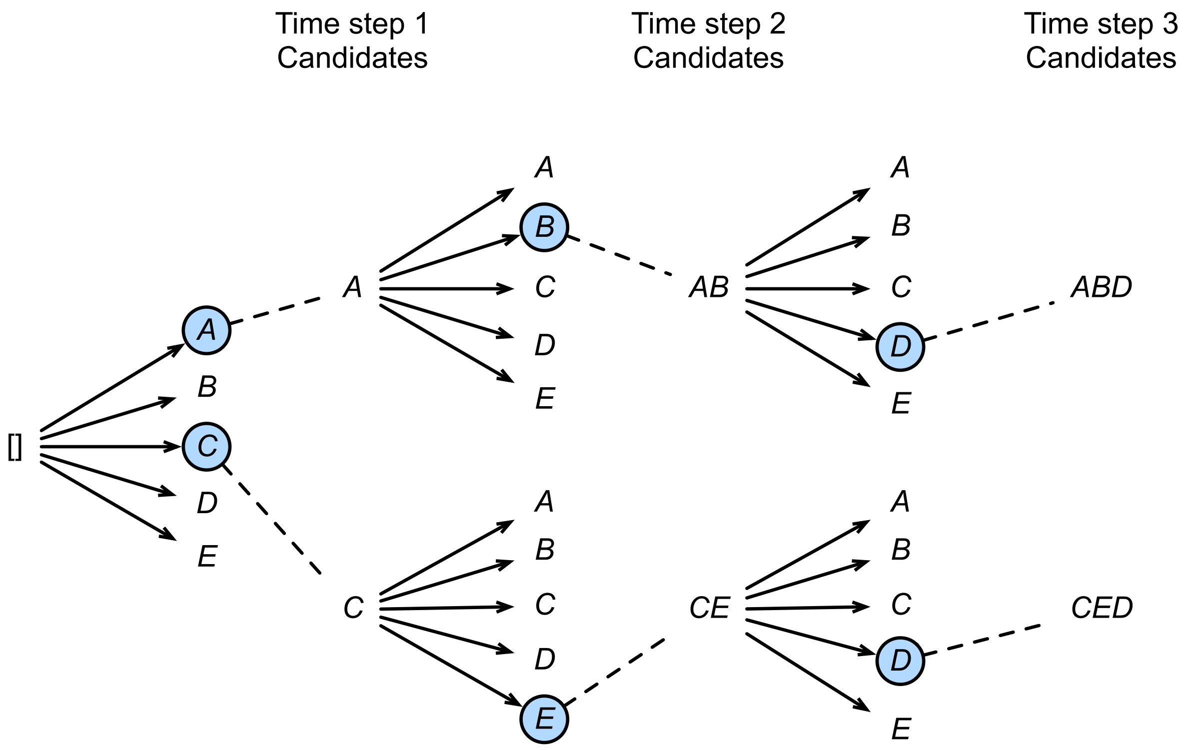
- Execute decoded subexpressions; remove programs whose outputs are not prefixes of the target







Beam search with constraints



RobustFill

IO examples to program translation as a Seq2Seq task

Features

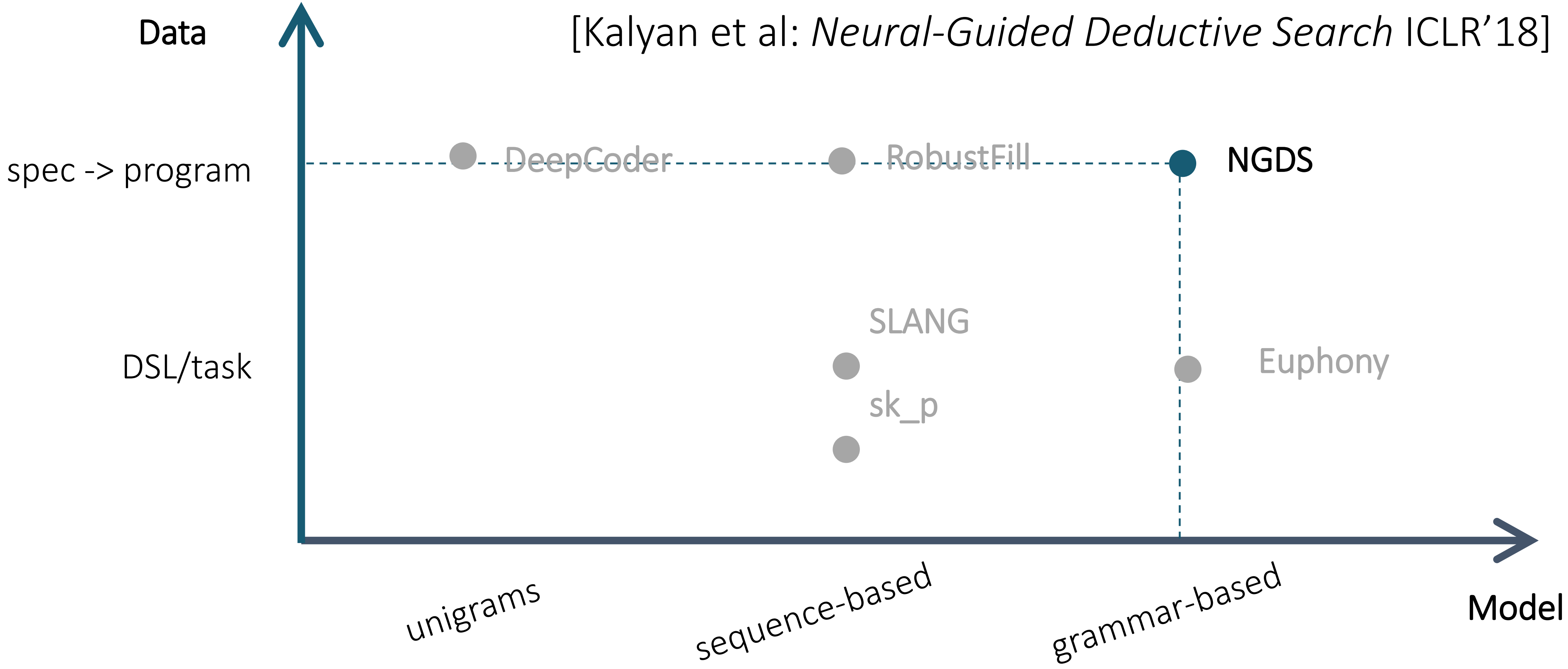
- Trained on synthetic data
- Unlike FlashFill, does not require inverse semantics
- Noise-tolerant

Limitations

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

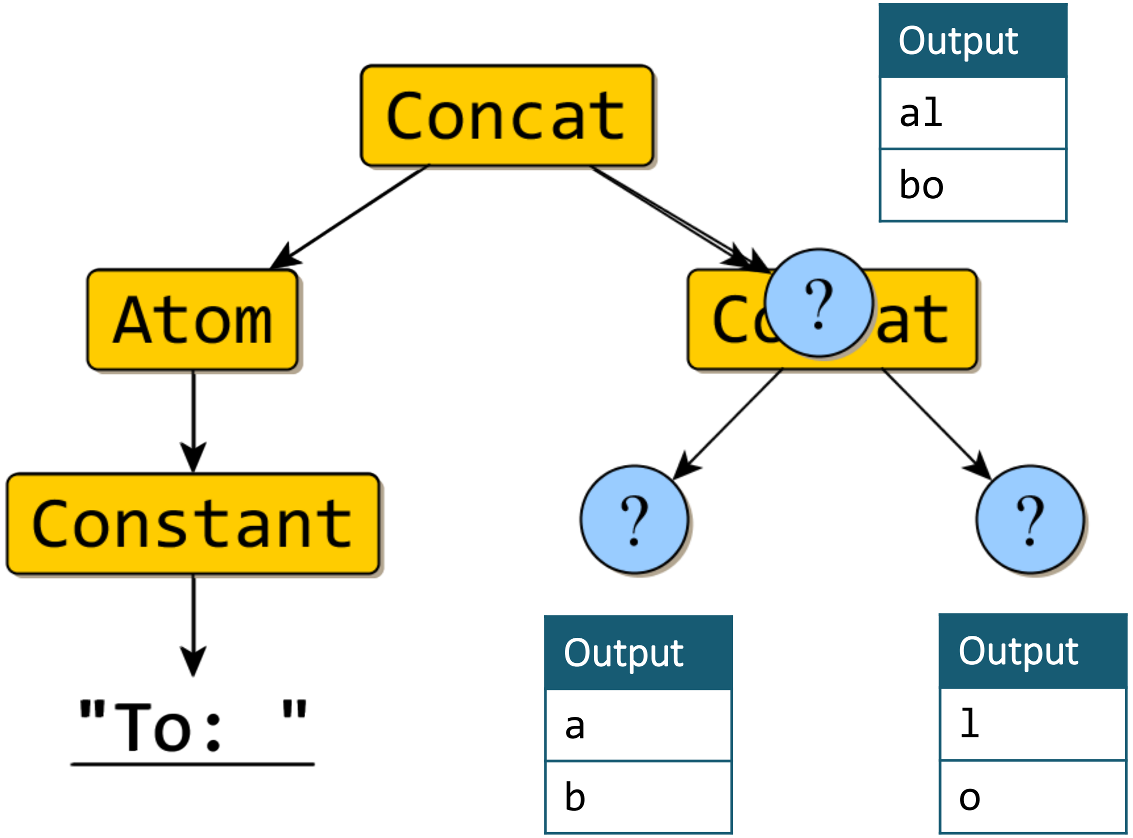
Statistical Models in Synthesis

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



Deductive Search

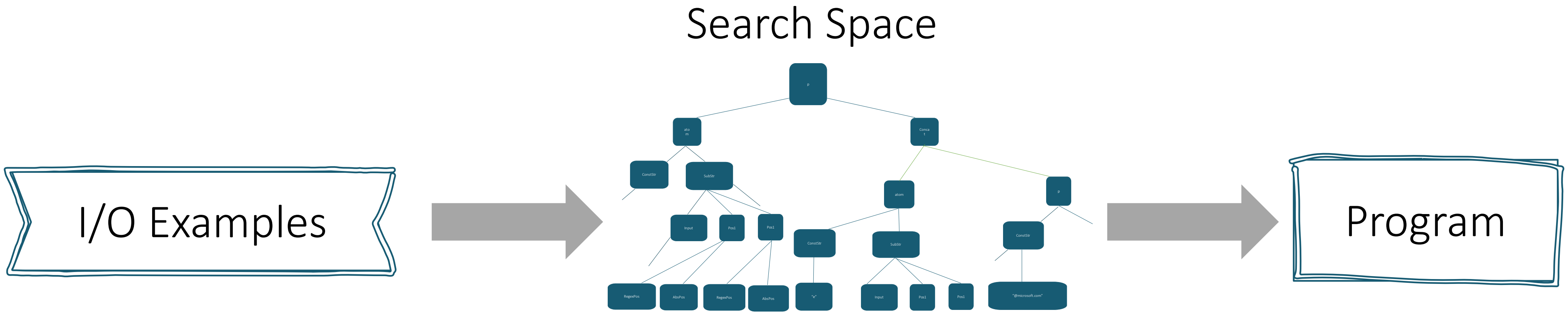
Input	Output
alice liddell	To: al
bob o'reilly	To: bo



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)

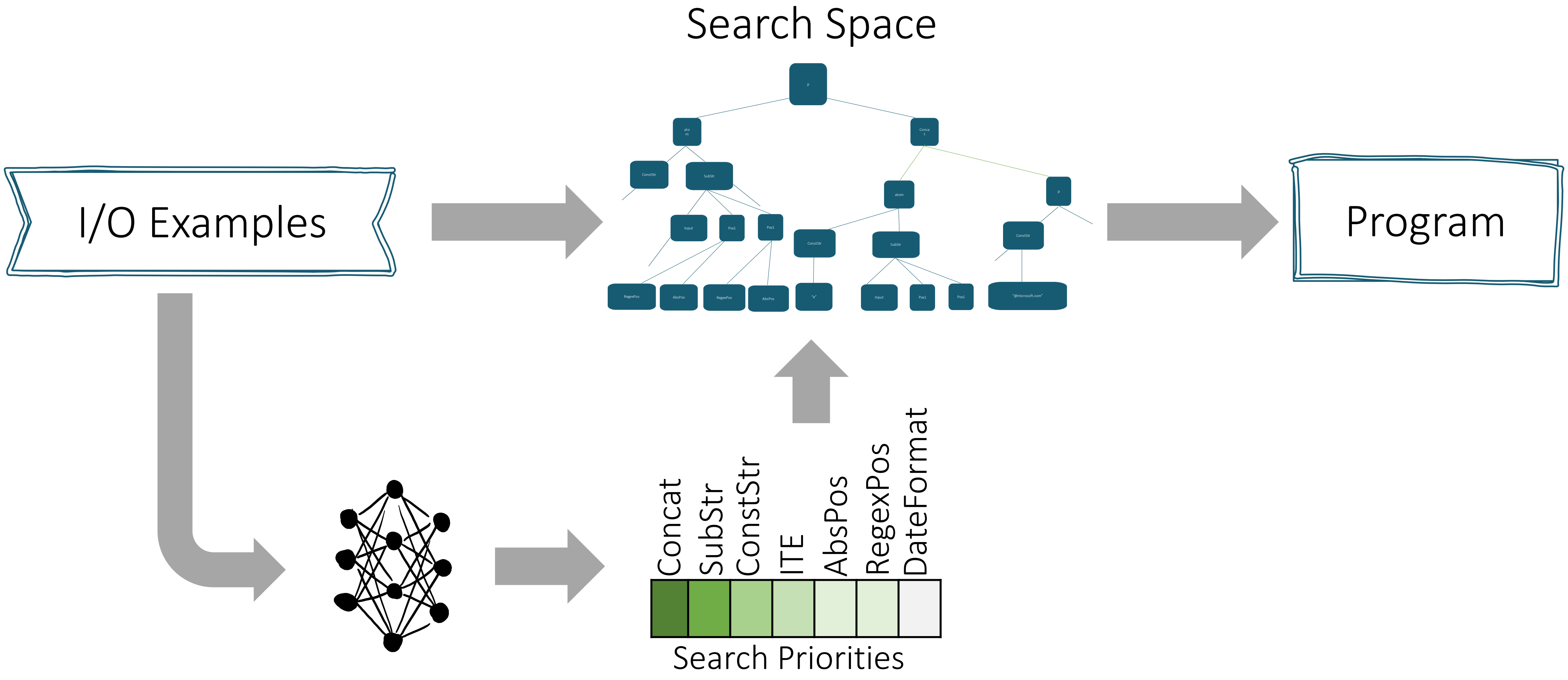
Machine-learned insights

Input	Output
`alice liddell`	`To: al`
`bob o’reilly`	`To: bo`

Can’t be a substring, requires concatenation

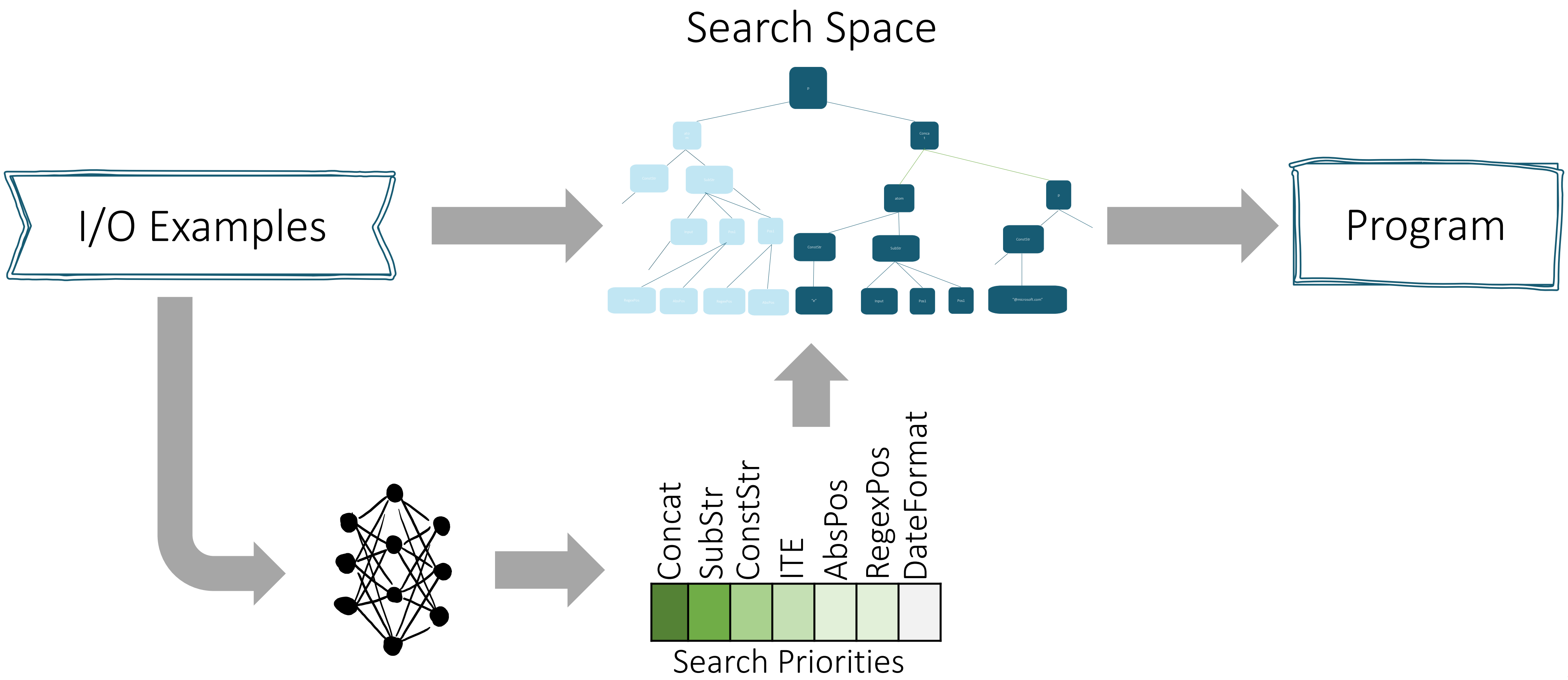
DeepCoder: Learning to Write Programs

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



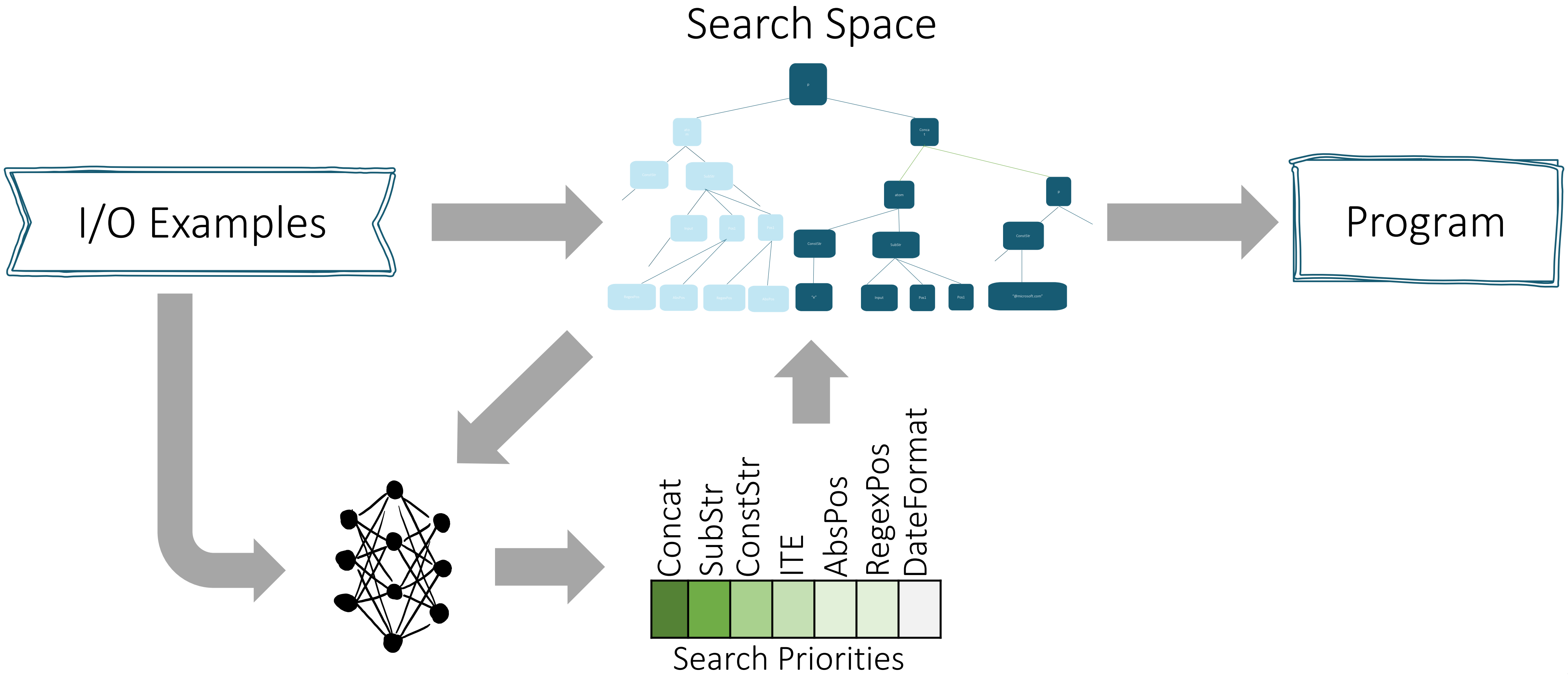
DeepCoder: Learning to Write Programs

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



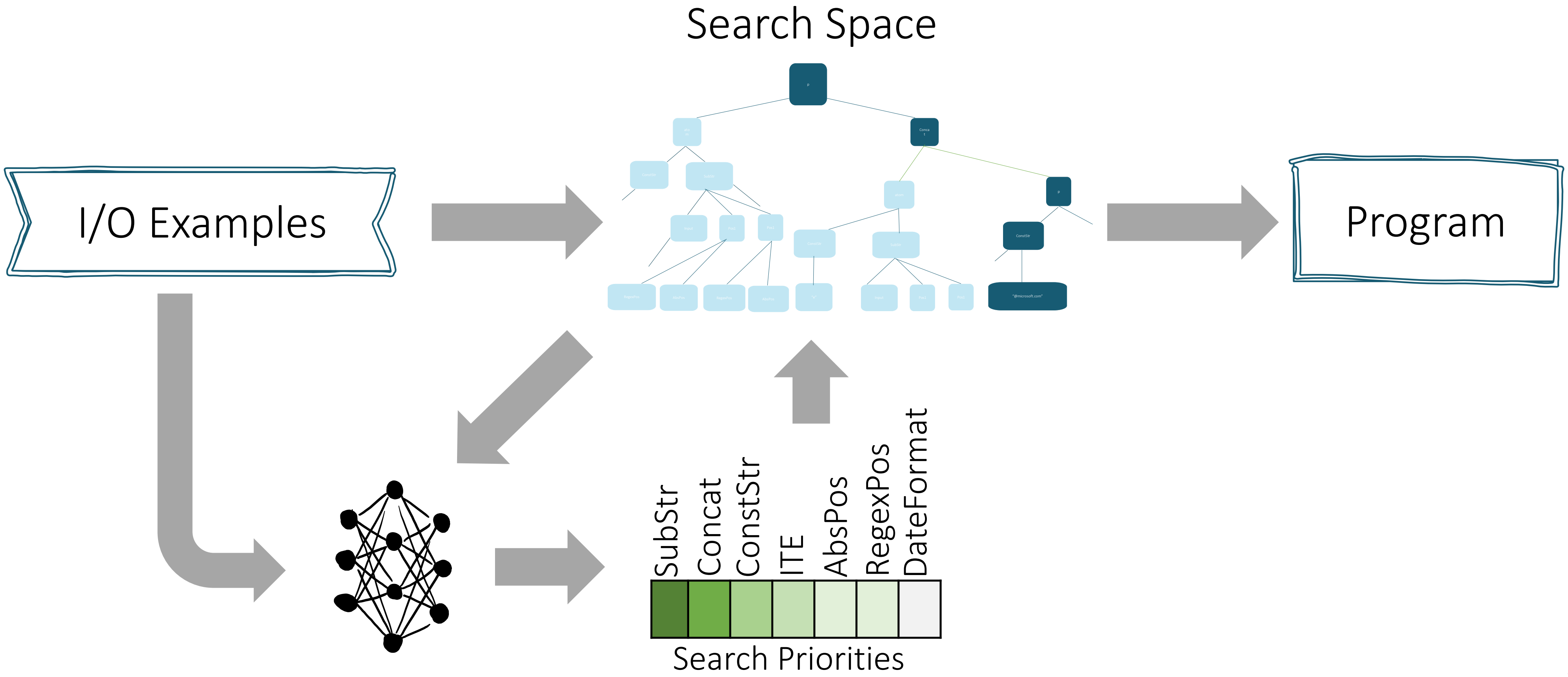
Neural-Guided Deductive Search

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



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

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

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 $h(P_1, \varphi), \dots, h(P_k, \varphi)$

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

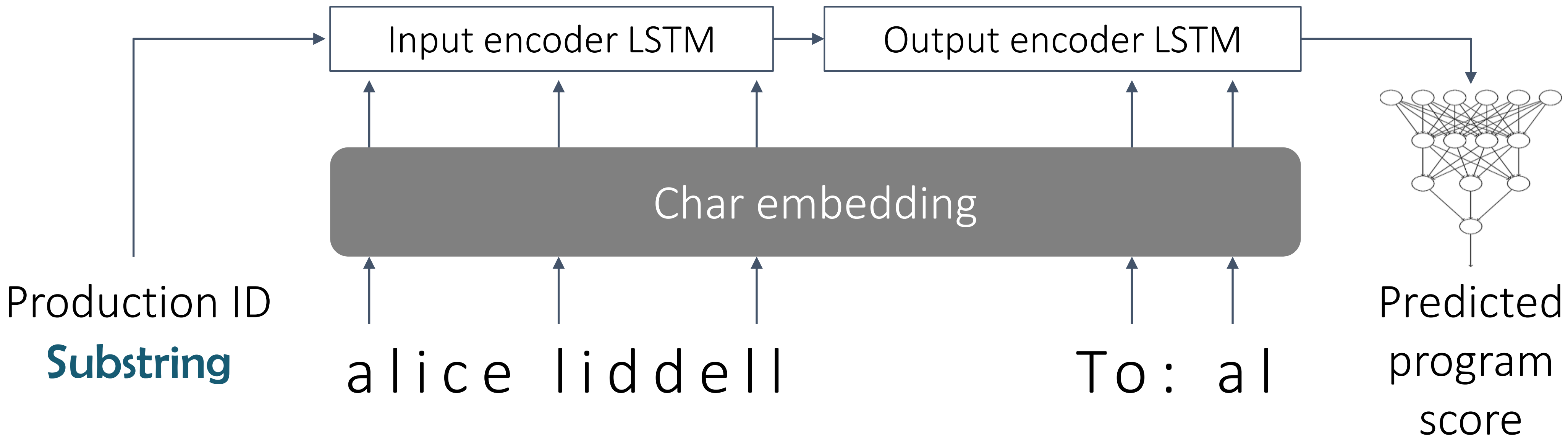
Train using squared-error loss over program scores:

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

Q: Why not re-ranking of branches?

- Because the magnitude of score values matters.

Model architecture



Search

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 than current optimum

NGDS

VSA-based search + neural guidance

Features

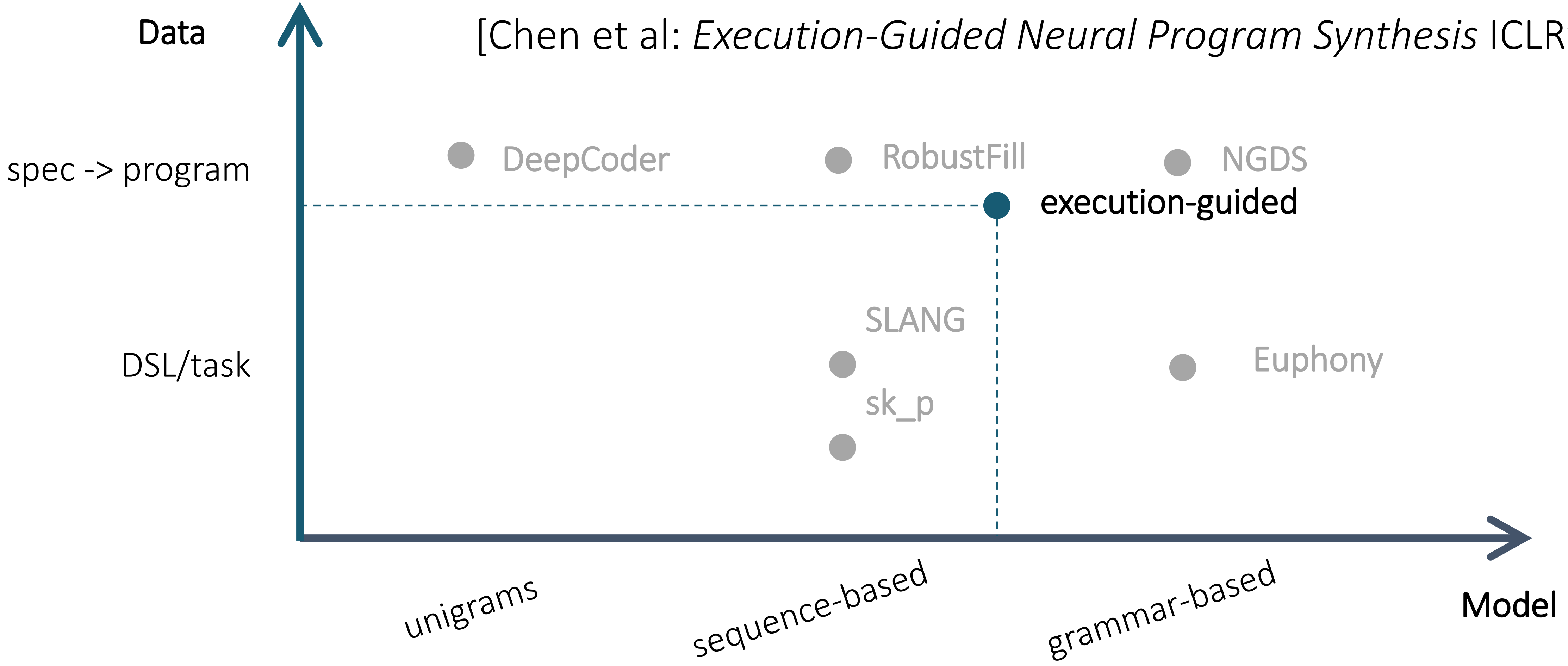
- Guarantees consistency with IO examples
- Thanks to top-down prop, we only need to learn a single grammar expansion => can generate many training examples from one synthesis problem

Limitations

- Requires inverse semantics (like FlashFill)

Statistical Models in Synthesis

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



Execution-guided neural synthesis

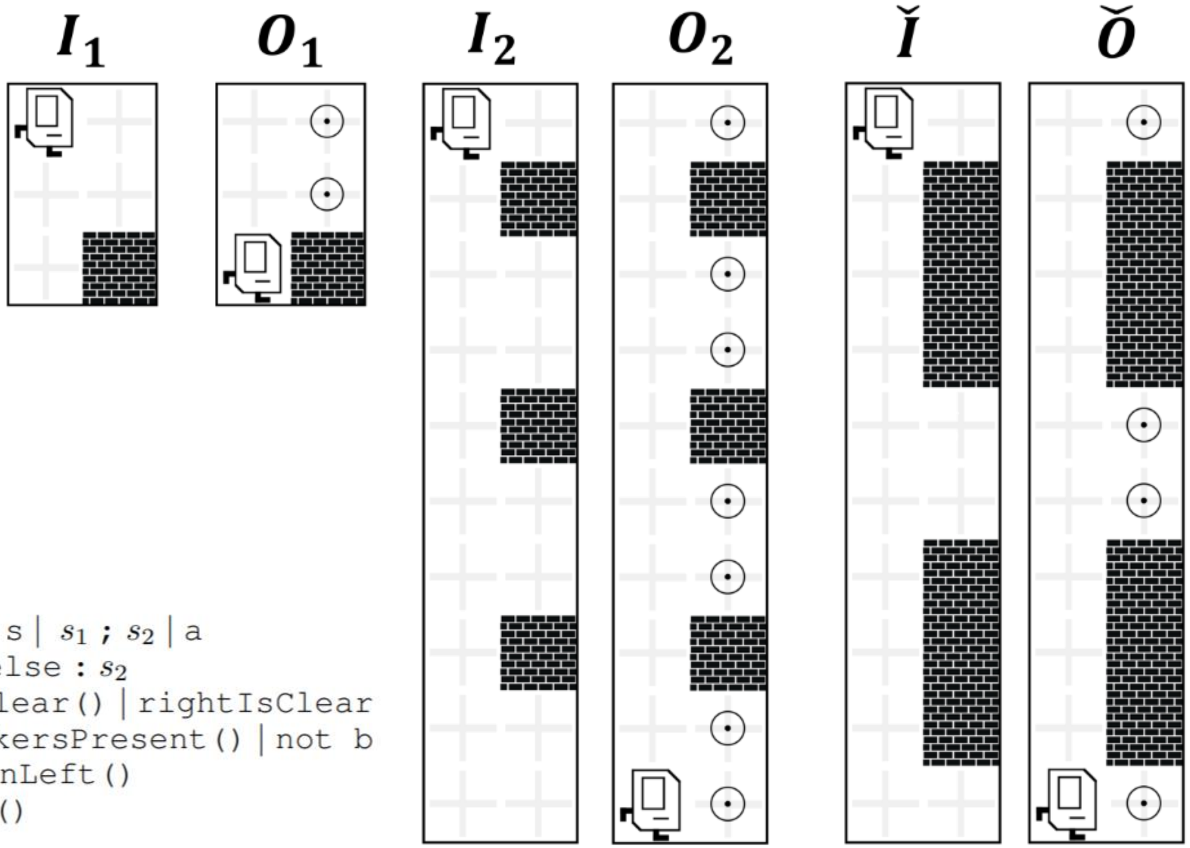
NGDS uses top-down propagation to make the model's task easier

- but this requires inverse semantics ☹

Can we do something similar but only using forward semantics?

- yes we can, for imperative programs!

Karel: robot navigation DSL



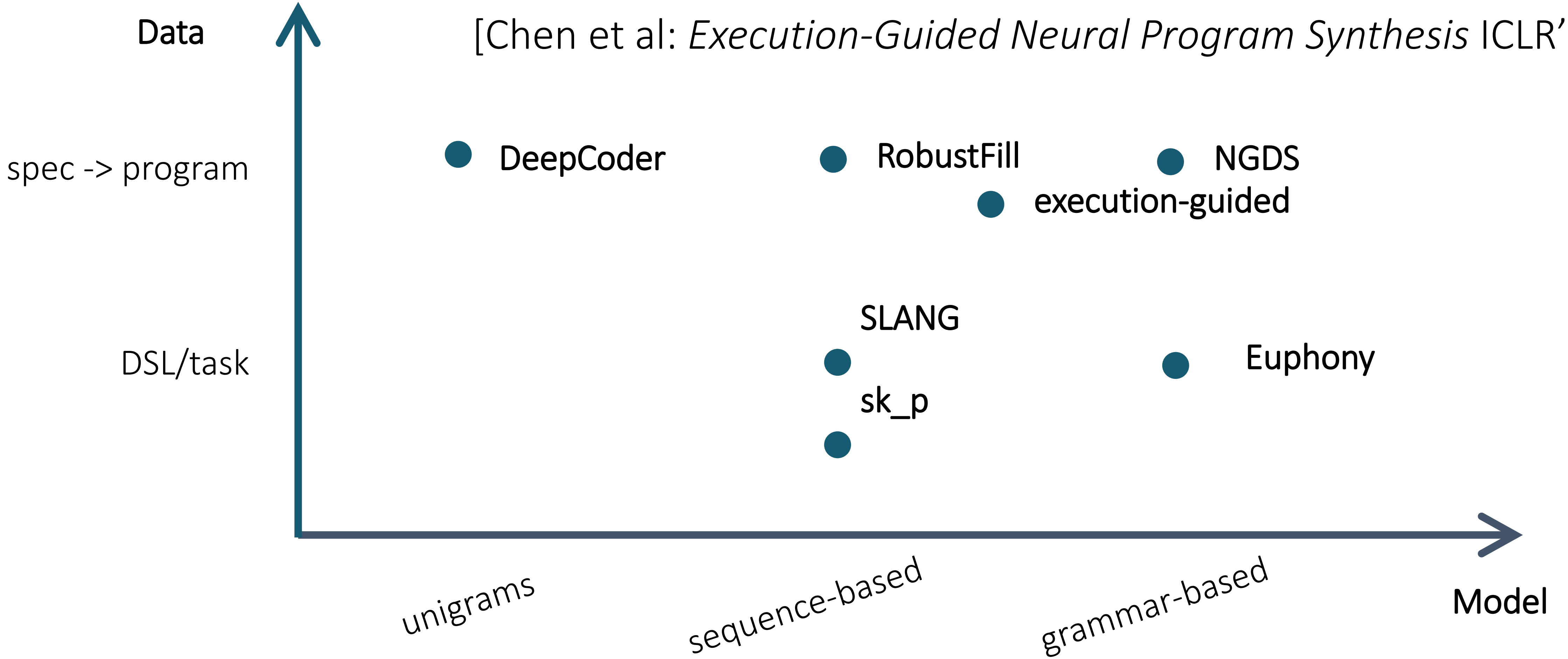
Underlying Program
(Not used by model)

```
def run():
  if rightIsClear():
    turnRight()
    move()
    putMarker()
    turnLeft()
    turnLeft()
    move()
    turnRight()
  while frontIsClear():
    move()
    if rightIsClear():
      turnRight()
      move()
      putMarker()
      turnLeft()
      turnLeft()
      move()
      turnRight()
```

```
  Prog p   ::=  def run() : s
  Stmt s   ::=  while(b) : s | repeat(r) : s | s1 ; s2 | a
             |  if(b) : s | ifelse(b) : s1 else : s2
  Cond b   ::=  frontIsClear() | leftIsClear() | rightIsClear
             |  markersPresent() | noMarkersPresent() | not b
Action a   ::=  move() | turnRight() | turnLeft()
             |  pickMarker() | putMarker()
  Cste r   ::=  0 | 1 | ... | 19
```

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 this week

Tuesday: pre-LLM era

- statistical language models for code
- neural architectures
- better search with neural guidance

Thursday: LLM era

- synthesis from natural language
- how can we make LLMs generate better code?

LLMs are changing the world...

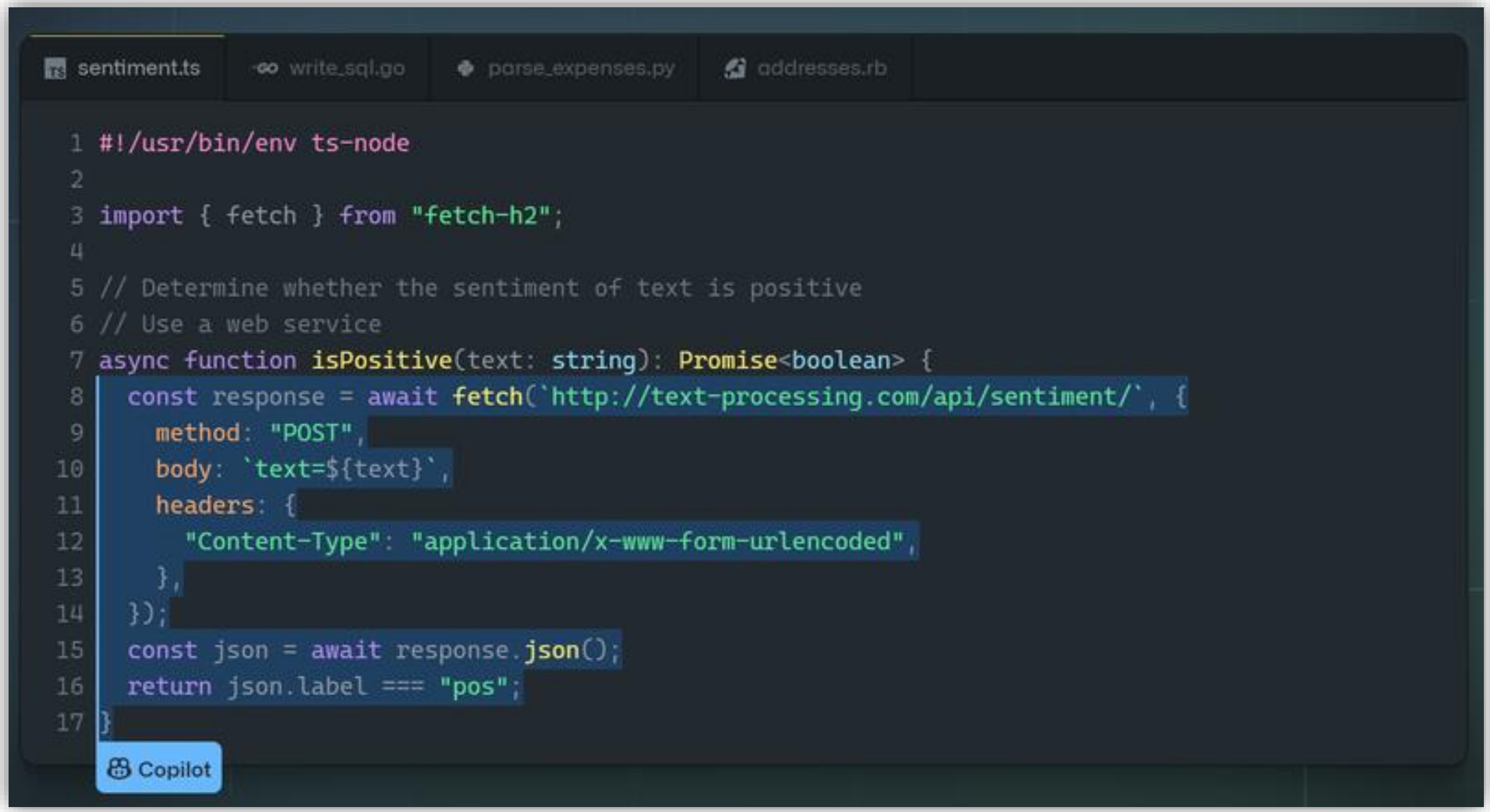
Github Copilot

Chat GPT

Amazon CodeWhisperer

and more...

LLMs are changing the world...



… but they are not perfect

according to a survey of 410 developers [Liang et al, ICSE'24]:

- the most popular reason developers don't use LLMs is that generated code **"doesn't meet functional or non-functional (e.g., security, performance) requirements that I need"**

according to [Perry et al, CCS'23]:

- participants with an AI assistant wrote significantly less secure code
- and were more likely to believe that they wrote secure code!

Two challenges

Accuracy

LLMs provide no guarantees that spec is satisfied

How do we increase the probability that a generated program matches user intent?

Validation

Spec is partly informal: NL, code context

How do we determine if a program matches user intent?

Techniques

Accuracy

Constrained Decoding

Fine Tuning

Validation

Self-consistency

User interaction

High-level DSL

Techniques

Accuracy

- Constrained Decoding
- Fine Tuning

Validation

- Self-consistency
- User interaction

High-level DSL

Monitor-guided decoding

[Agrawal et al: *Monitor-guided decoding of code LMs with static analysis of repository context.* NeurIPS'23]

LLMs struggle to produce correct code in the context of a repo

Idea: use a language server to mask LLM token predictions

Method to be completed

```java
private ServerNode parseServer(String url) {
    Preconditions.checkNotNull(url);
    int start = url.indexOf(str:"/") + 2;
    int end = url.lastIndexOf(str:"?") == -1 ?
        url.length() : url.lastIndexOf(str:"?");
    String str = url.substring(start, end);
    String [] arr = str.split(regex:":");

    return ServerNode.Builder
            .newServerNode()
            .|
}
```

text-davinci-003 and SantaCoder

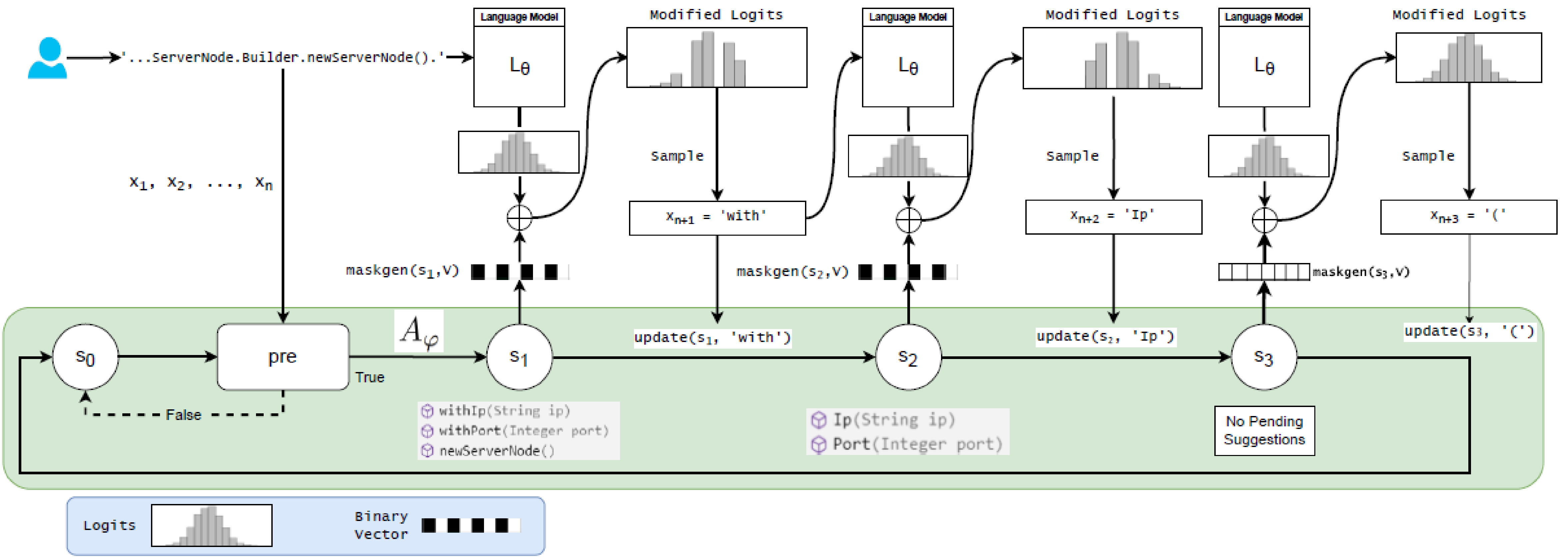
```java
host(arr[0])
.port(Integer.parseInt(arr[1]))
.build();
```

SantaCoder with *monitor guided decoding*

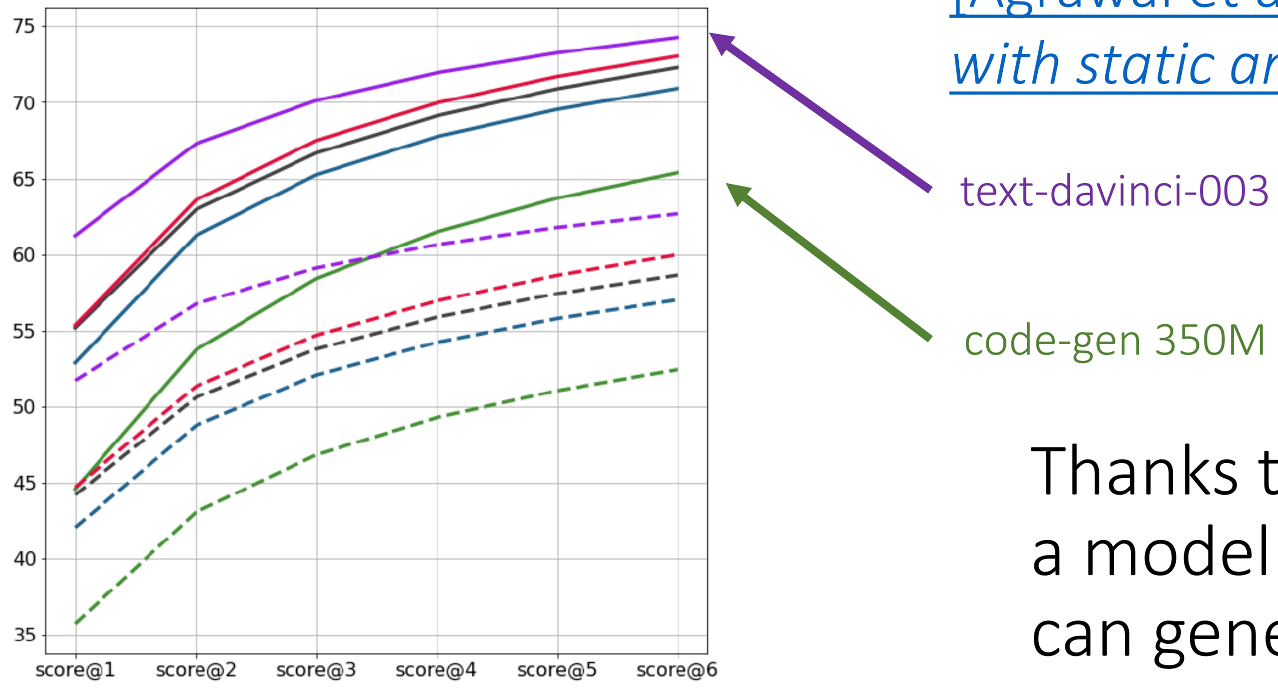
```java
withIp(arr[0])
.withPort(Integer.parseInt(arr[1]))
.build();
```

Monitor-guided decoding

[Agrawal et al: *Monitor-guided decoding of code LMs with static analysis of repository context.* NeurIPS'23]



Monitor-guided decoding: results



compilation rate

[Agrawal et al: *Monitor-guided decoding of code LMs with static analysis of repository context.* NeurIPS'23]

Thanks to monitor guidance,
a model with 1000x fewer parameters
can generate better code than GPT3!

Techniques

Accuracy

Constrained Decoding

Fine Tuning

Validation

Self-consistency

User interaction

High-level DSL

Self-Play

[Haluptzok et al: *Language models can teach themselves to program better.* ICLR’23]

AlphaZero got better at Go through self-play;
can we do this for code?

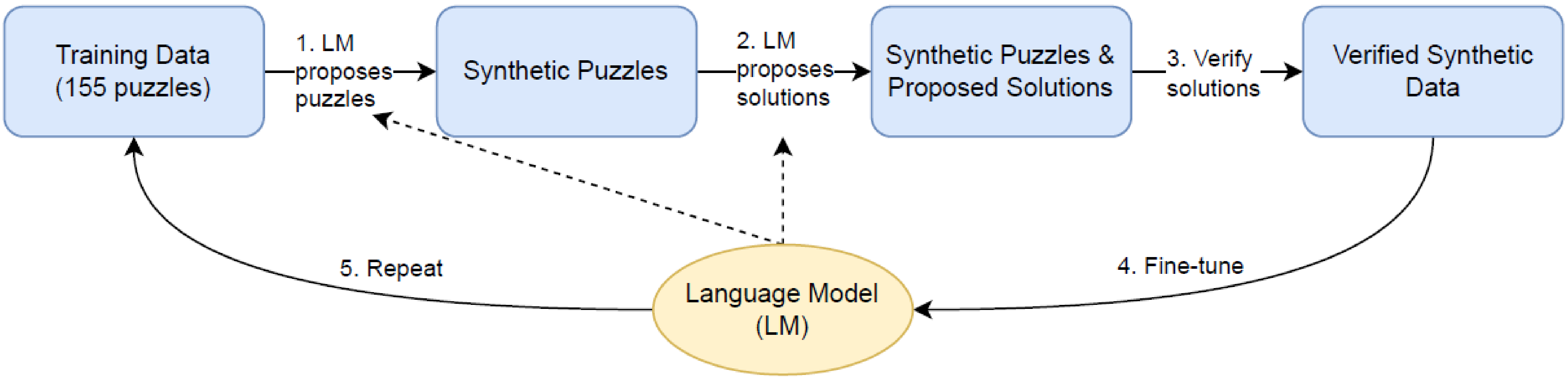
Idea: use LLM to generate *programming puzzles* and solutions to those puzzles

```
def f(c: int):
    return c + 50000 == 174653

def g():
    return 174653 - 50000

assert f(g())
```

```
def f(x: str, chars=['Hello', 'there', 'you!'], n=4600):
    return x == x[::-1] and all([x.count(c) == n for c in chars])

def g(chars=['Hello', 'there', 'you!'], n=4600):
    s = "".join([c*n for c in chars])
    return s + s[::-1]

assert f(g())
```

Self-Play

[Haluptzok et al: *Language models can teach themselves to program better.* ICLR'23]



Self-Play: results

[Haluptzok et al: *Language models can teach themselves to program better.* ICLR'23]

fine-tune dataset	Verified	Puzzles	Solutions (Count)	# Tokens	Pass@100
BASELINE	N/A	No puzzles	No solutions (0)	0	7.5%
HUMAN	Yes	Human	Synthetic (635)	74K	10.5%
VERIFIED-125M	Yes	Synthetic	Synthetic (1M)	74M	15.4%
VERIFIED-1.3B	Yes	Synthetic	Synthetic (1M)	65M	18.9%
VERIFIED-2.7B	Yes	Synthetic	Synthetic (1M)	66M	20.6%
UNVERIFIED-CODEX	No	Synthetic	Synthetic (1M)	113M	21.5%
VERIFIED-CODEX	Yes	Synthetic	Synthetic (1M)	98M	38.2%

Test performance of the Neo-2.7B model after fine-tuning on puzzles produced by different models

Large pass@k improvement from one round of fine-tuning

Puzzles from larger models are more helpful

Unverified Codex is not as helpful!

Techniques

Accuracy

- Constrained Decoding
- Fine Tuning

Validation

- Self-consistency
- User interaction

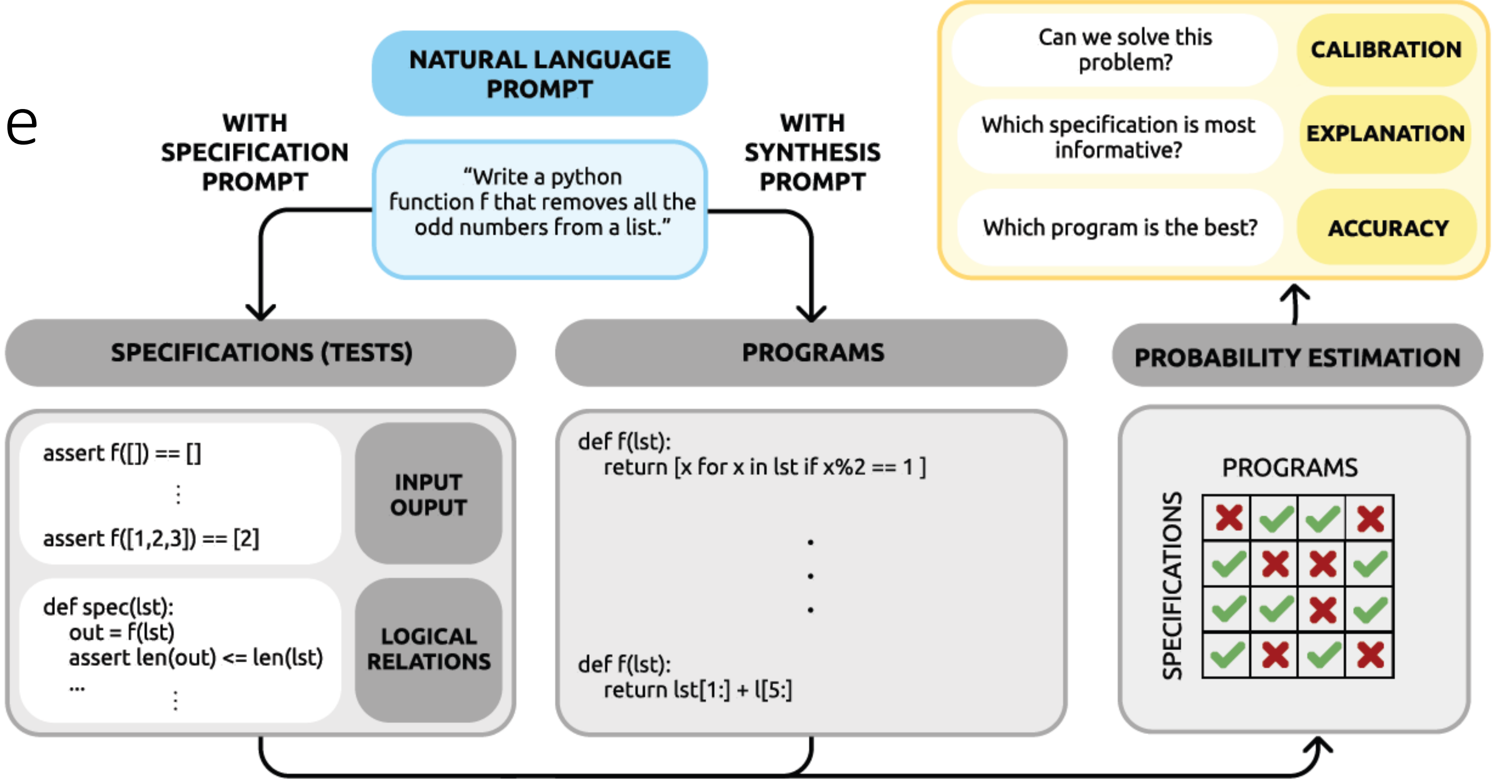
High-level DSL

Speculyzer

[Li, Key, Ellis: *Towards trustworthy neural program synthesis.* 2023]

Goal: Increase trustworthiness of NL->code

Idea: generate *tests* alongside programs

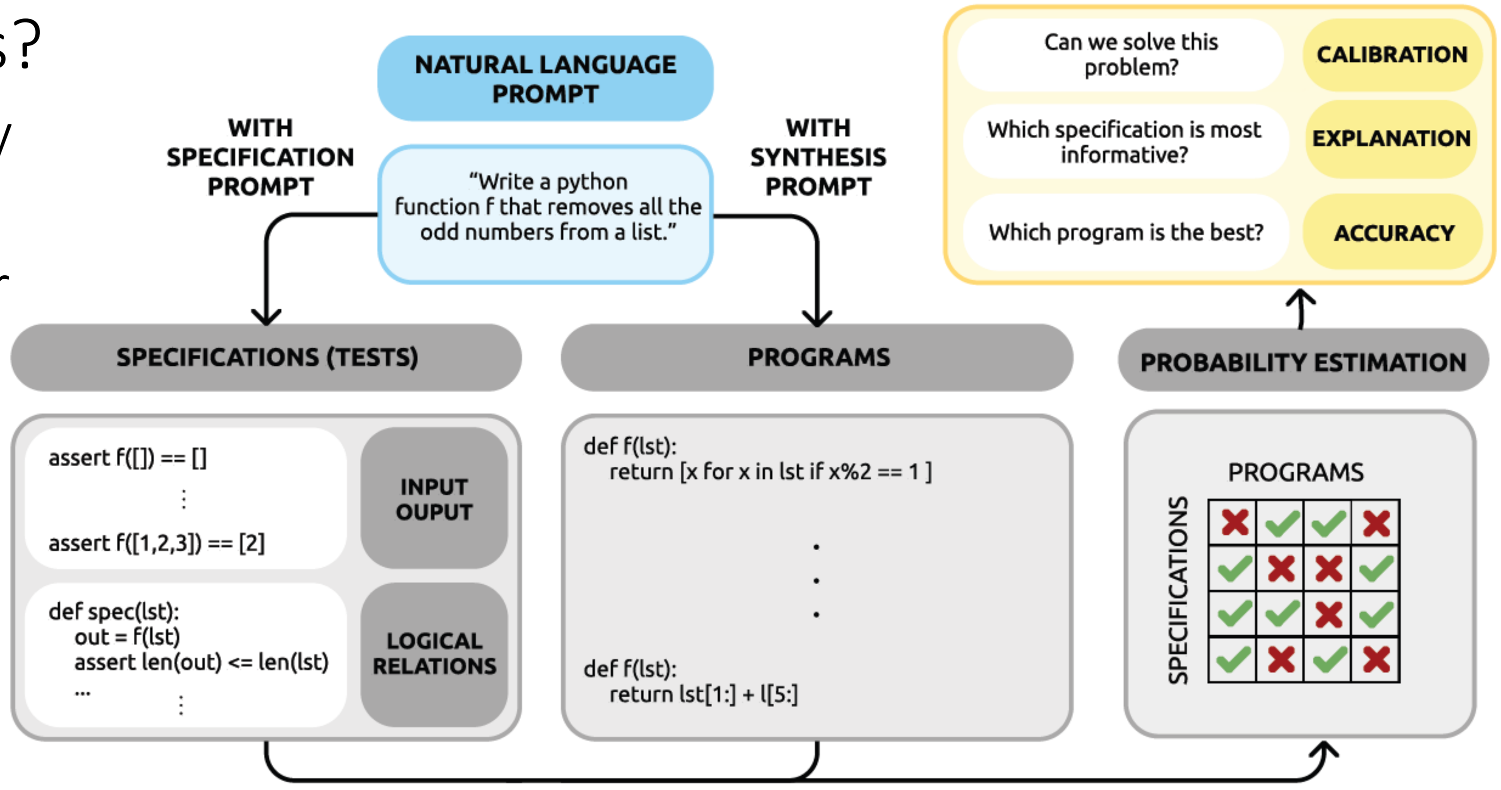


Speculyzer

[Li, Key, Ellis: *Towards trustworthy neural program synthesis.* 2023]

What can we do with the tests?

- rank programs based how many tests they pass
- cluster programs based on their behavior on test inputs
- train a classifier to predict if the model knows the solution
- pick the most selective tests to show to the user



Speculyzer

PROGRAM

```
def derivative(xs: list):
    """ xs represent coefficients of a polynomial.
    xs[0] + xs[1] * x + xs[2] * x^2 + ....
     Return derivative of this polynomial in the same form.
    >>> derivative([3, 1, 2, 4, 5])
    [1, 4, 12, 20]
    >>> derivative([1, 2, 3])
    [2, 6]
    """
    return [x * i for i, x in enumerate(xs) if i != 0]
```

TOP LOGICAL RELATION

```
def test_derivative(xs: list):
    """ Given an input `xs`, test whether the function `derivative`
is implemented correctly.
    """
    ys = derivative(xs)
    assert len(ys) == len(xs) - 1
    for i in range(len(ys)):
        assert ys[i] == xs[i+1] * (i + 1)

# run `test_derivative` on a new testcase
test_derivative([3, 1, 2, 4, 5])
```

RANDOM LOGICAL RELATION

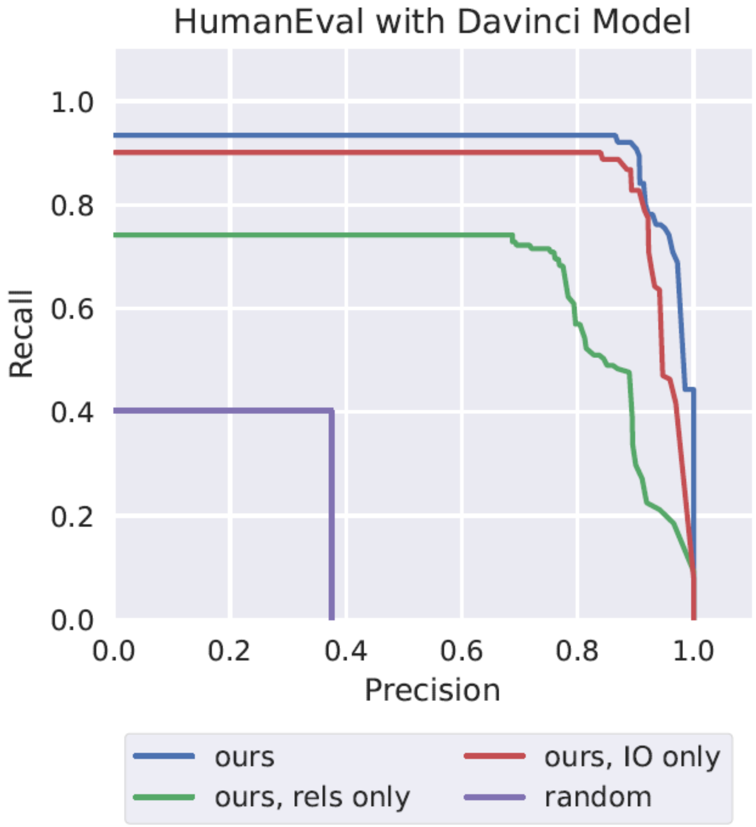
```
def test_derivative(xs):
    """ Test function derivative().
    """
    # TODO
    pass

# run `test_derivative` on a new testcase
test_derivative([2, 3, 4, 10, -12])
```

[Li, Key, Ellis: *Towards trustworthy neural program synthesis.* 2023]

Picking the most selective test to show to the user

Speculyzer: results

[Li, Key, Ellis: *Towards trustworthy neural program synthesis.* 2023]



Can achieve *zero error rate* on human eval in exchange for dropping recall from 93% to 44%!

Techniques

Accuracy

- Constrained Decoding
- Fine Tuning

Validation

- Self-consistency
- User interaction

High-level DSL

The validation challenge

"In the context of Copilot, there is a shift from writing code to understanding code"

Taking Flight with Copilot, ACM Queue, Dec 22

validation is hard

- [Vaithilingam et al] observed 8 cases of over-reliance: bugs due to skipped validation

validation is a bottleneck

- single most prevalent activity according to [Mozannar et al]

prevalence of a validation strategy depends on its cost [Liang et al]

to help with validation, we need to lower its cost

LEAP

[Ferdowsi et al: *Validating AI-Generated Code with Live Programming.* CHI'24]

lowers the cost of validation by execution
using live programming

demo

User study

no-LP	LP
AI suggestions + terminal	AI suggestions + live programming

Research questions

how does live programming affect…

over- / under-reliance on AI

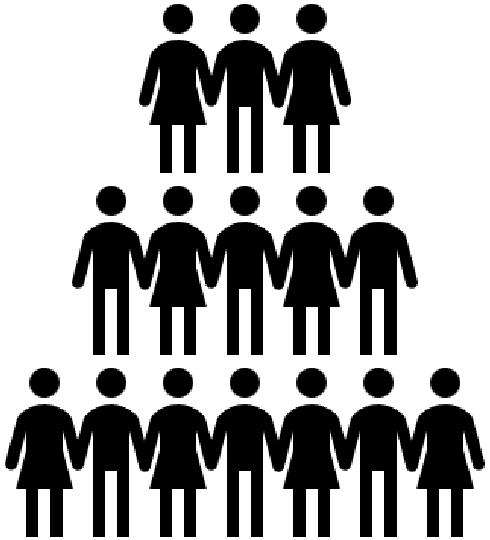
validation strategies

cognitive load

Tasks

	API-heavy	algorithmic	
fixed prompt	**pandas** (multiple correct suggestions) clean dataframe and compute stats using pandas	**bigrams** (no correct suggestions) find most frequent bigram in a string	fixed prompt
open prompt	**box plot** overlay scatter plot over boxplot using matplotlib	**string rewriting** parse rewrite rules and apply to string	open prompt

Participants

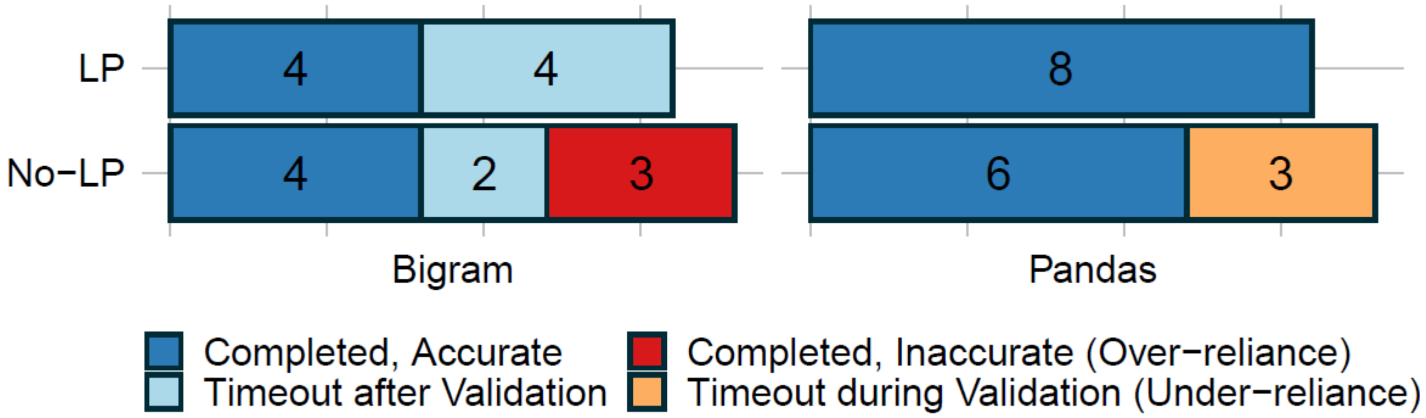


n = 17

occupation:
15 academia / 2 industry

Python usage:
2 occasionally /
8 regularly /
7 almost every day

RQ1: over-/under-reliance



6 no-PB vs 0 PB participants **mid-judged** correctness of their solution

by lowering the cost of validation,
leap reduces over-/under-reliance on AI

RQ1: over-/under-reliance

"it was **easy to understand** the behavior of a code suggestion because the little boxes on the side allowed for you to preview the results." (P3)

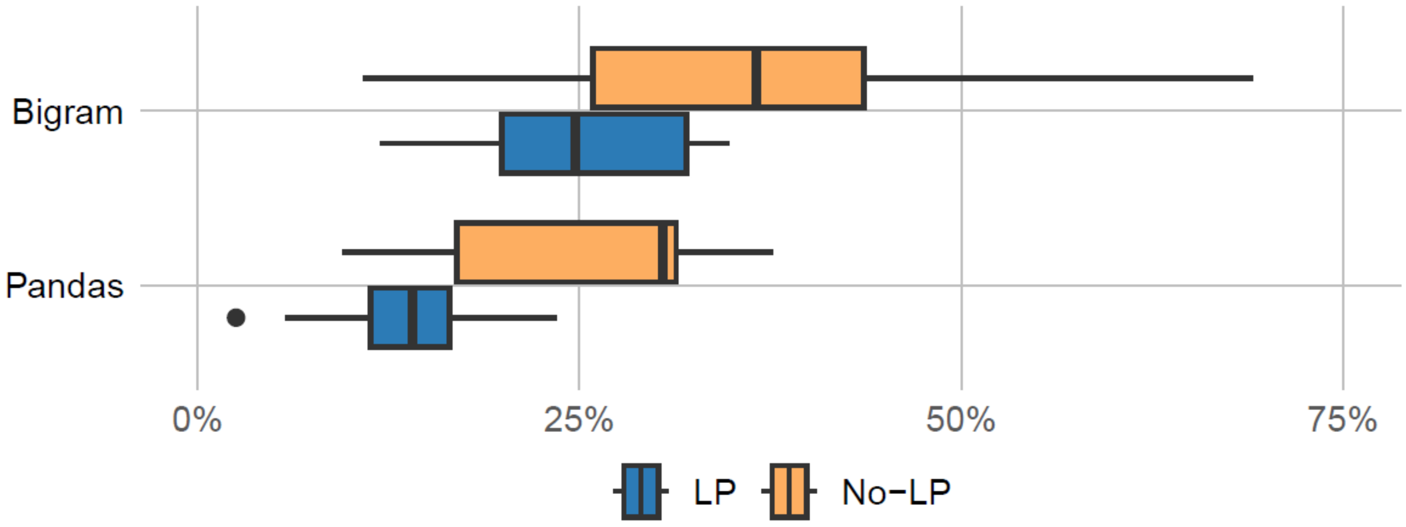
"it **saved me the effort** of writing multiple print statements." (P1)

6 no-PB vs 0 PB participants **mid-judged** correctness of their solution

by lowering the cost of validation,
leap reduces over-/under-reliance on AI

RQ2: validation strategies

percentage of time spent in Suggestion Panel

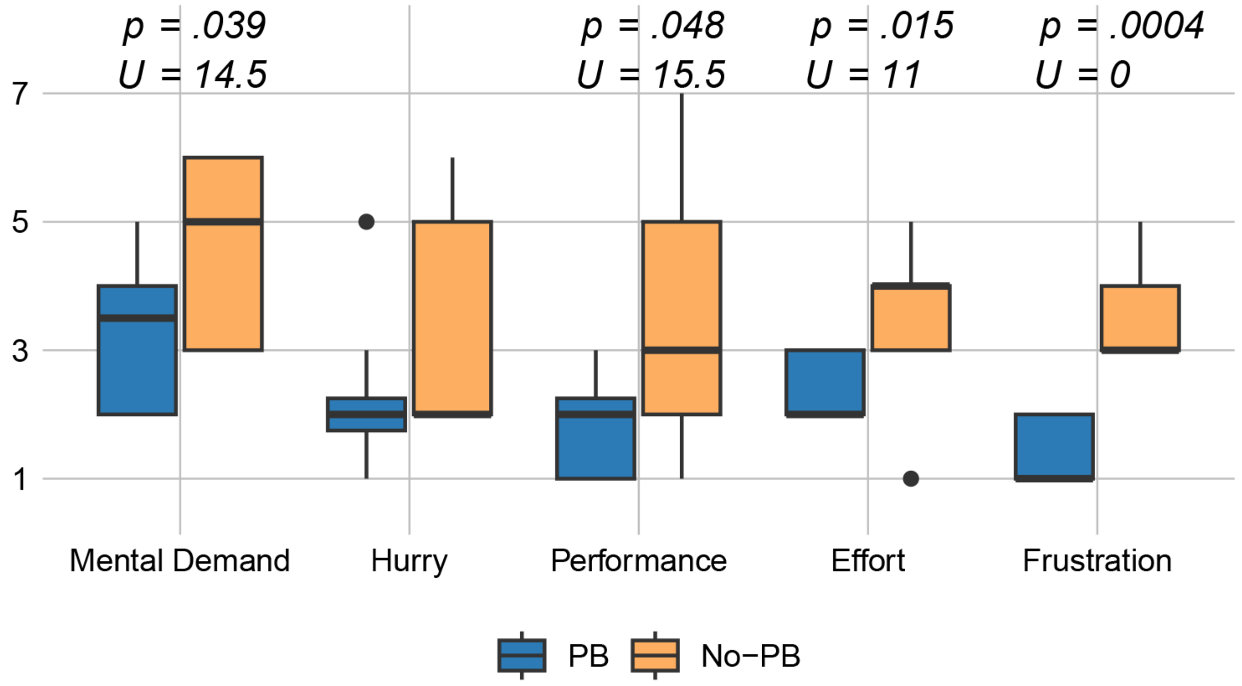


“I didn’t look too closely in the actual code,
I was *just looking at the runtime values* on the side.” (P1)

leap participants spent less time reading code

RQ3: cognitive load

NASA TLX cognitive load metrics on Pandas



leap significantly reduced cognitive load of AI-assisted programming on tasks amenable to validation by execution

Techniques

Accuracy

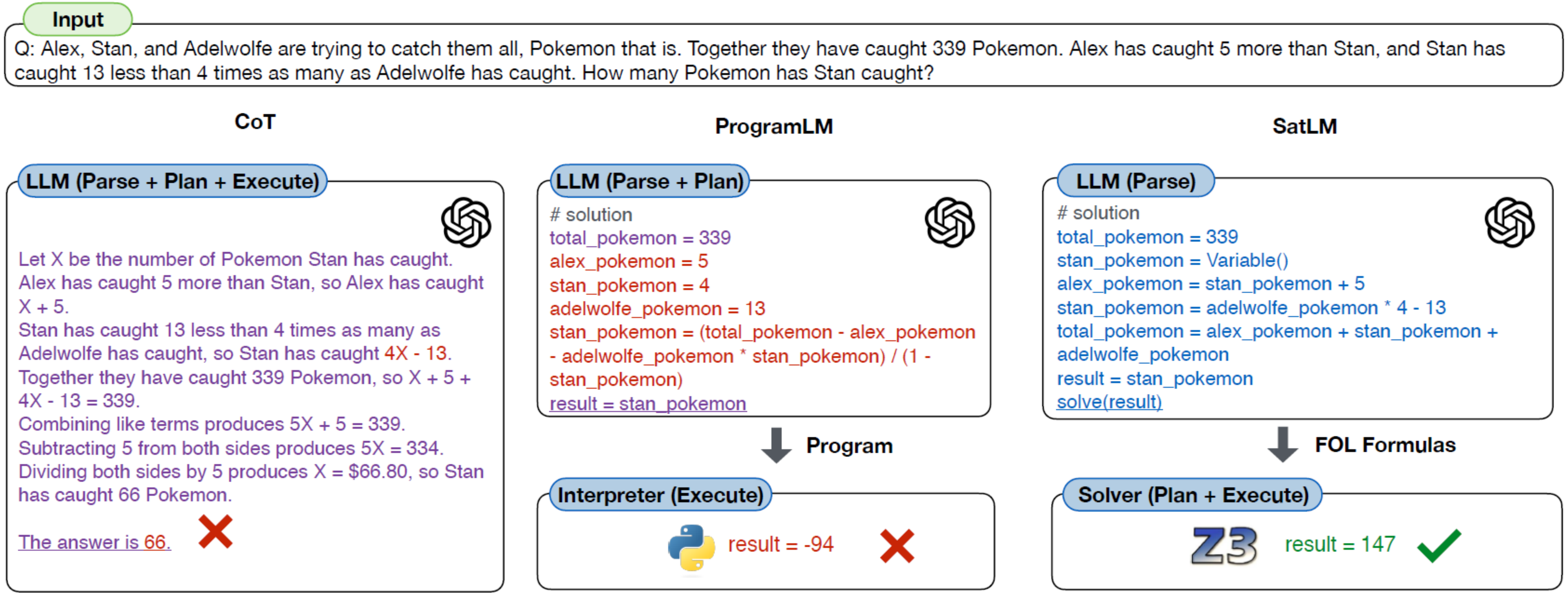
- Constrained Decoding
- Fine Tuning

Validation

- Self-consistency
- User interaction

High-level DSL

SatLM


Input
Q: Alex, Stan, and Adelwolfe are trying to catch them all, Pokemon that is. Together they have caught 339 Pokemon. Alex has caught 5 more than Stan, and Stan has caught 13 less than 4 times as many as Adelwolfe has caught. How many Pokemon has Stan caught?
CoT
ProgramLM
SatLM
LLM (Parse + Plan + Execute)
Let X be the number of Pokemon Stan has caught.
Alex has caught 5 more than Stan, so Alex has caught X + 5.
Stan has caught 13 less than 4 times as many as Adelwolfe has caught, so Stan has caught 4X - 13.
Together they have caught 339 Pokemon, so X + 5 + 4X - 13 = 339.
Combining like terms produces 5X + 5 = 339.
Subtracting 5 from both sides produces 5X = 334.
Dividing both sides by 5 produces X = $66.80, so Stan has caught 66 Pokemon.
The answer is 66.
LLM (Parse + Plan)
# solution
total_pokemon = 339
alex_pokemon = 5
stan_pokemon = 4
adelwolfe_pokemon = 13
stan_pokemon = (total_pokemon - alex_pokemon - adelwolfe_pokemon * stan_pokemon) / (1 - stan_pokemon)
result = stan_pokemon
Program
Interpreter (Execute)
result = -94
LLM (Parse)
# solution
total_pokemon = 339
stan_pokemon = Variable()
alex_pokemon = stan_pokemon + 5
stan_pokemon = adelwolfe_pokemon * 4 - 13
total_pokemon = alex_pokemon + stan_pokemon + adelwolfe_pokemon
result = stan_pokemon
solve(result)
FOL Formulas
Solver (Plan + Execute)
Z3
result = 147

SatLM: Contributions

Declarative prompting is easier for the LLM

- it only does translation, not planning or execution

Significant accuracy increase across domains

- wrt standard prompting, CoT, ProgLM
- domains: arithmetic reasoning, logical reasoning, regex synthesis

Can use satisfiability/ambiguity for validation

SatLM: Limitations

Limited to SMT-decidable logics

Some problems are better fit for imperative encoding

Some problems might require ambiguity

SatLM: Fig 3 in Z3Py

SAT Solution

```
total_height = 120
joe_height = 2 * sara_height + 6
total_height = sara_height + joe_height
solve(joe_height)
```

```python
from z3 import *

s = Solver()
sara_height = Int('sara_height')
joe_height = Int('joe_height')
total_height = 120
s.add(joe_height == 2 * sara_height + 6)
s.add(total_height == sara_height + joe_height)

if s.check() == sat:
    print(s.model()[joe_height])
```

Z3

82

SatLM: ambiguity check

```
…
s.add(joe_height >= 2 * sara_height + 6)
s.add(total_height == sara_height + joe_height)

if s.check() == sat:
    res = s.model()[joe_height]
    s.add(joe_height != res)
    if s.check() == sat:
        print('AMBIG')
    else:
        print(res)
else:
    print('UNSAT')
```

$\xrightarrow{\text{Z3}}$ AMBIG

SatLM: Potential Improvements

Run multiple times and

- ignore attempts that don't parse or produce AMBIG/UNSAT
- even better: check answers for consistency

Run in a loop, providing feedback to the LLM

- if AMBIG, tell the LLM to strengthen the constraints
- if UNSAT, get UNSAT core and tell the LLM to weaken one of those

Combine individual constraints from different solutions

- maybe perform lattice search until we get a SAT, unambiguous set