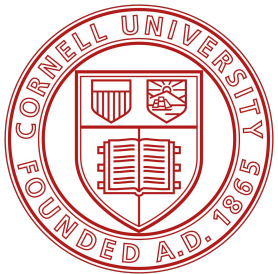
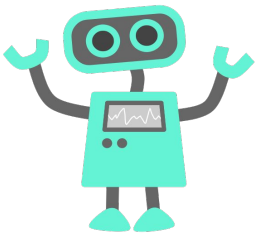
Non-Canonical Tasks in Synthesis and Learning

Kevin Ellis
Joint with Darren Key, Wen-Ding Li, Hao Tang
Cornell University
MAPS Symposium 2023, FSE



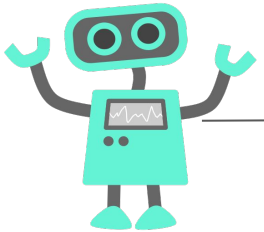
Program synthesizer

specification →  → `program`

Specification = Dependent Types
(types + logical predicates)

Program synthesizer

$$n: \text{Nat} \to x: \alpha \to \{\text{List } \alpha \mid \text{len } \nu = n\}$$



```
replicate = λn . λx . if n ≤ 0
  then Nil
  else Cons x (replicate (dec n) x)
```

Synquid. Polikarpova 2016

Specification = Input-Outputs

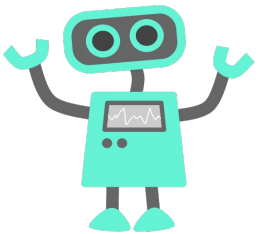
“Cornell University” ->

“C.U.”

“Eidgenössische Technische Hochschule” ->

“E.T.H.”

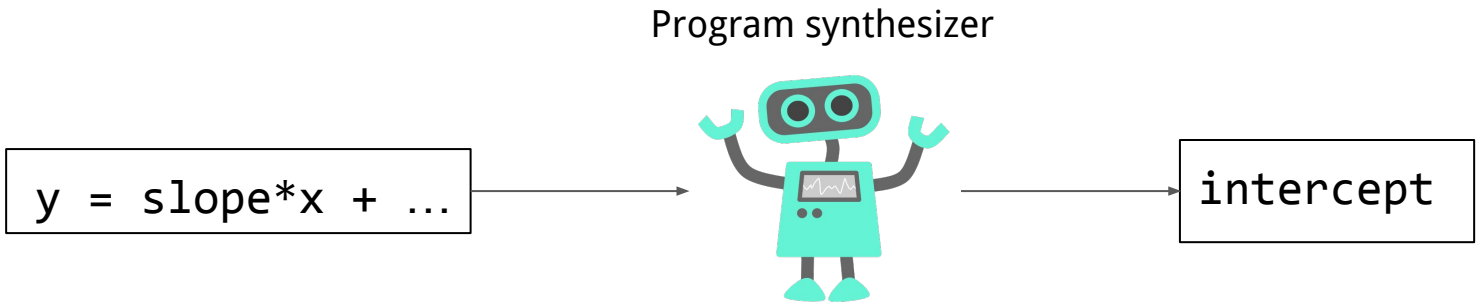
Program synthesizer



```
“.”.join(x.split())
```

Eg, FlashFill. Gulwani 2012

Specification = Partially completed program

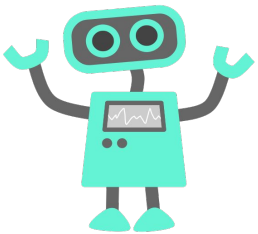


eg, Copilot, Codex

Specification = Partially completed program

```
# open file and loop over lines
with open(file, “r”) as handle:
```

Program synthesizer



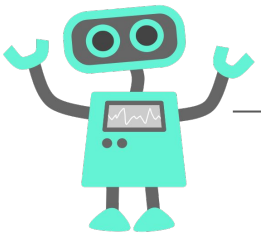
```
for ln in file:
```

eg, Copilot, Codex

Specification = Partially completed program

```
# flatten a list called `xss`
```

Program synthesizer



```
[x
 for xs in xss
 for x in xs ]
```

eg, Copilot, Codex

How we evaluate program synthesizers

Synthesizing Data Structure Transformations from Input-Output Examples *

	Name	Runtime	Runtime (no deduction)	Runtime (no types)	Expert examples	Random examples	Runtime (random)	Extra primitives	Description
Lists	add	**0.04**	0.05	3.87	5	4	**0.04**		Add a number to each element of a list.
	append	**0.23**	0.49	⊥	3	16	0.93		Append an element to a list.
	concat	**0.13**	0.22	68.95	5	23	0.20		Concatenate two lists together.
	dedup	**231.05**	⊥	⊥	7	-	-	member	Remove duplicate elements from a list.
	droplast	**316.39**	⊥	⊥	6	-	-		Drop the last element in a list.
	dropmax	**0.12**	0.19	77.05	3	7	0.16	max	Drop the largest number(s) in a list.
	dupli	**0.11**	0.86	378.35	3	5	0.20		Duplicate each element of a list.
	evens	**7.39**	45.52	⊥	5	8	30.08		Remove the odd numbers from a list.
	last	**0.02**	0.06	1.80	4	4	0.03		Return the last element in a list.
	length	**0.01**	0.14	41.36	4	5	0.04		Return the length of a list.
	max	**0.46**	9.53	⊥	7	8	8.19		Return the largest number in a list.

How we evaluate program synthesizers

DeepMind

2022-3-16

Approach	Validation Set				Test Set		
	10@1k	10@10k	10@100k	10@1M	10@1k	10@10k	10@100k
9B	16.9%	22.6%	27.1%	30.1%	14.3%	21.5%	25.8%
41B	16.9%	23.9%	28.2%	31.8%	15.6%	23.2%	27.7%
41B + clustering	21.0%	26.2%	31.8%	34.2%	16.4%	25.4%	29.6%

Table 5 | **Solve rates of our best systems on the validation set and test set .**

How we evaluate program synthesizers:

Success Rate, controlling for compute time

How we use program synthesizers

```
def sub_list(nums1 : list, nums2 : list) -> list:
    """
    Write a function to subtract two lists element-
   wise.
    """
```

How we use program synthesizers

```
def sub_list(nums1 : list, nums2 : list) -> list:
    """
    Write a function to subtract two lists element-
   wise.
    """
    return list(map(lambda x, y:  x-y, nums1, nums2))
```

Evan Pu added a new photo.

November 16, 2021 ·

copilot's buggy code suggestion (in gray) against the correct code (below)

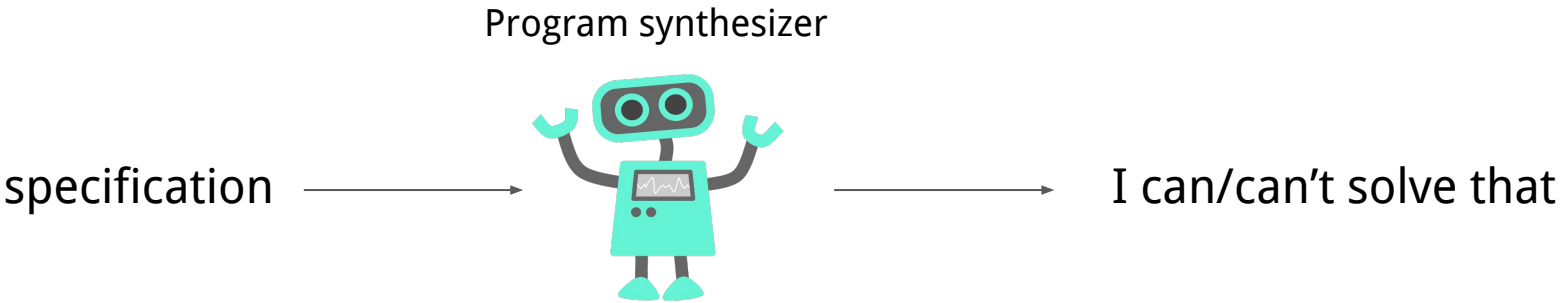
it is very subtle, but caused my search algorithm to bug out and invalidated 2 days worth of works

please use responsibly I guess is my take-away.

```
rect_params = t_rank[queue_ids[0]], b_rank[queue_ids[1]], l_rank[queue_ids[2]], r_rank[queue_ids[3]]
log_rect_prob = np.log(t[queue_ids[0]]) + np.log(b[queue_ids[1]]) + np.log(l[queue_ids[2]]) + np.log(r[queue_ids
return rect_params, log_rect_prob
log_rect_prob = np.log(t[rect_params[0]]) + np.log(b[rect_params[1]]) + np.log(l[rect_params[2]]) + np.log(r[rec
return rect_params, log_rect_prob
```

Task:

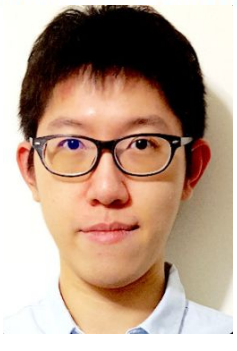
Predict whether the synthesizer should be trusted to solve a particular problem



Toward Trustworthy Neural Program Synthesis

Wen-Ding Li[*1], **Darren Key**[*1], **Kevin Ellis**[1]

[1]Department of Computer Science, Cornell University, USA
wl678@cornell.edu, dyk34@cornell.edu, kellis@cornell.edu



Trust in Traditional Program Synthesis

From Program Verification to Program Synthesis

Saurabh Srivastava
University of Maryland, College Park
saurabhs@cs.umd.edu

Sumit Gulwani
Microsoft Research, Redmond
sumitg@microsoft.com

Jeffrey S. Foster
University of Maryland, College Park
jfoster@cs.umd.edu

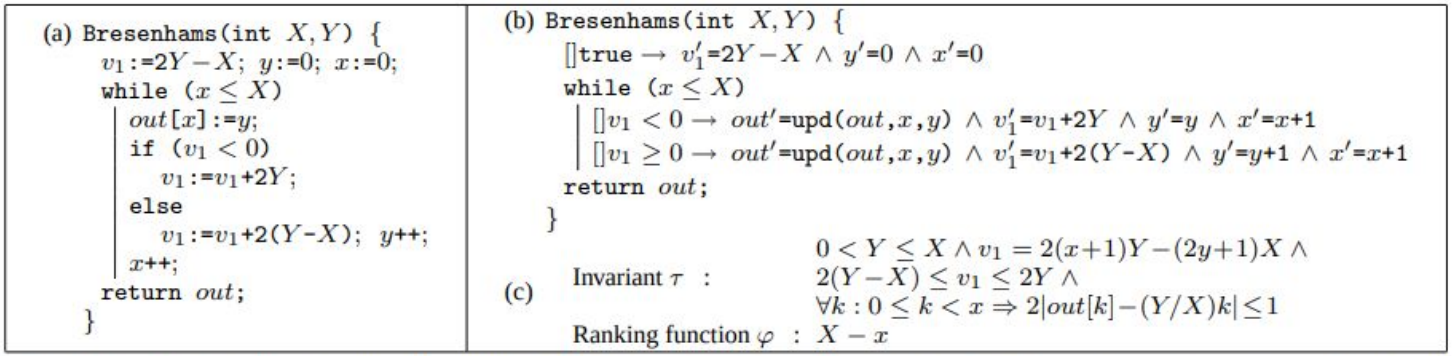


Figure 1. (a) Bresenham's line drawing algorithm (b) The invariant and ranking function that prove partial correctness and termination, respectively. (c) The algorithm written in transition system form, with statements as equality predicates, guarded appropriately.

Trust in Traditional Program Synthesis

$$\text{program} \vdash \text{specification}$$

Trust in ***Neural*** Program Synthesis

program $\vdash$ specification (?)

Trust in Neural Program Synthesis

$$\text{program} \vdash \text{natural language} \qquad (?)$$

Trust in Neural Program Synthesis

program $\vdash$ natural language (X)

Trust in Neural Program Synthesis

Neural network defines:

Pr[program | natural language]

The Trust Conundrum

Trust ~ Verification

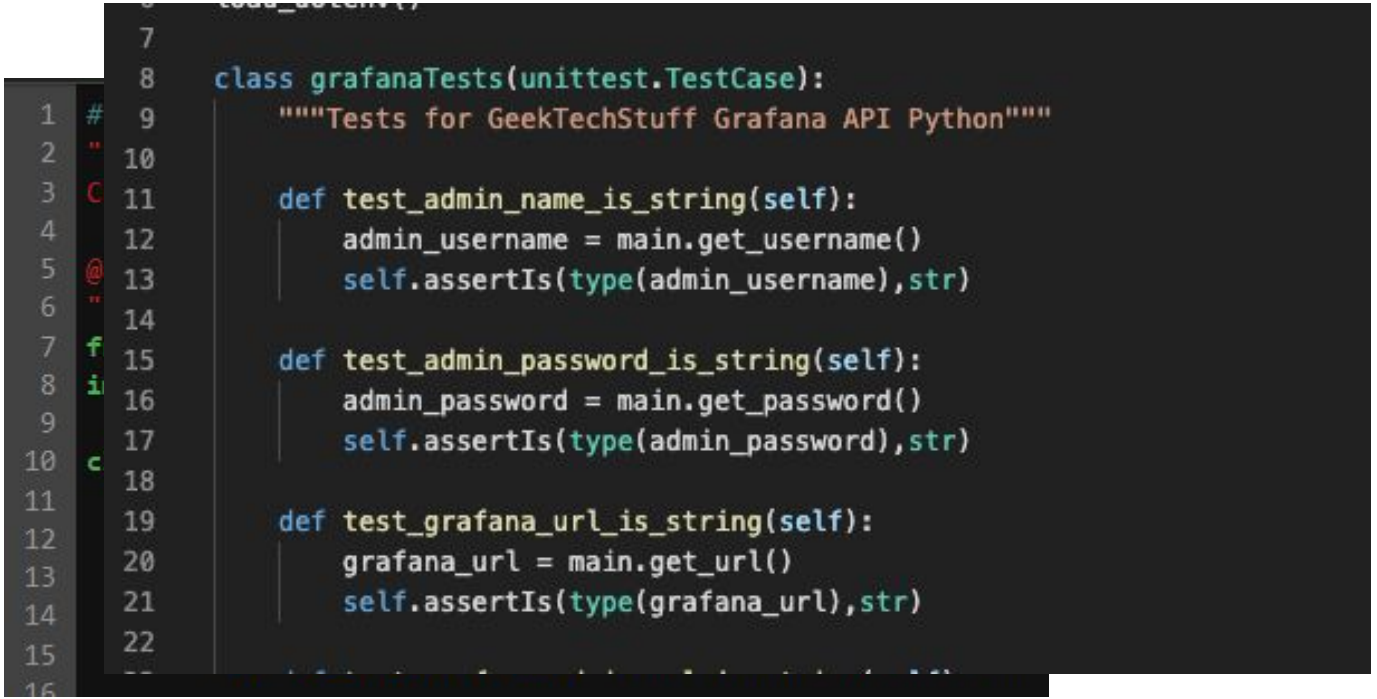
program $\vdash$ specification

My specification is informal…

…because train data is messy natural code

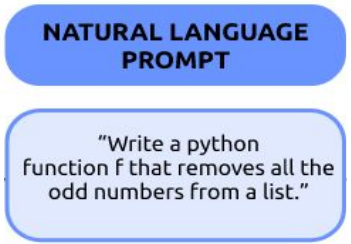
And I can’t verify against an informal specification

How do people build trust?

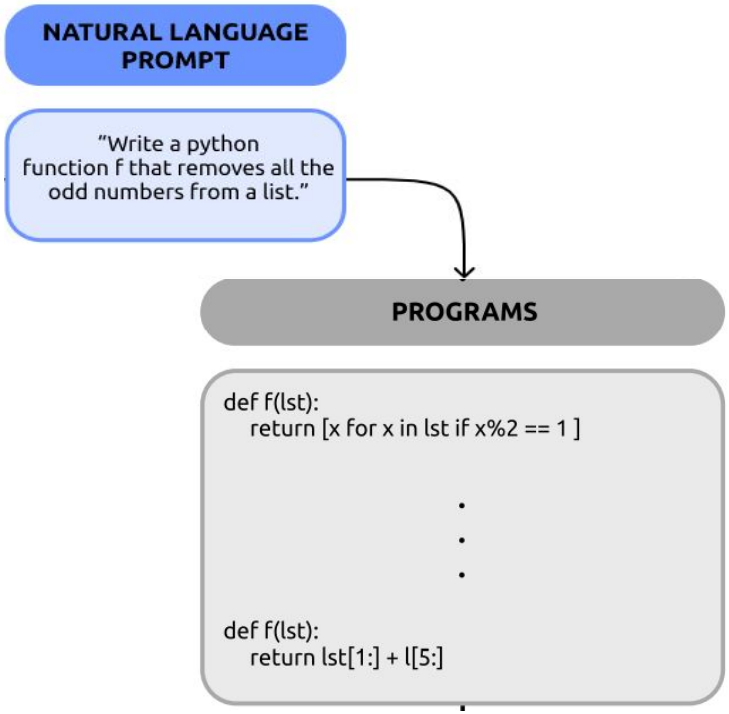


```
class grafanaTests(unittest.TestCase):
    """Tests for GeekTechStuff Grafana API Python"""

    def test_admin_name_is_string(self):
        admin_username = main.get_username()
        self.assertIs(type(admin_username),str)

    def test_admin_password_is_string(self):
        admin_password = main.get_password()
        self.assertIs(type(admin_password),str)

    def test_grafana_url_is_string(self):
        grafana_url = main.get_url()
        self.assertIs(type(grafana_url),str)
```

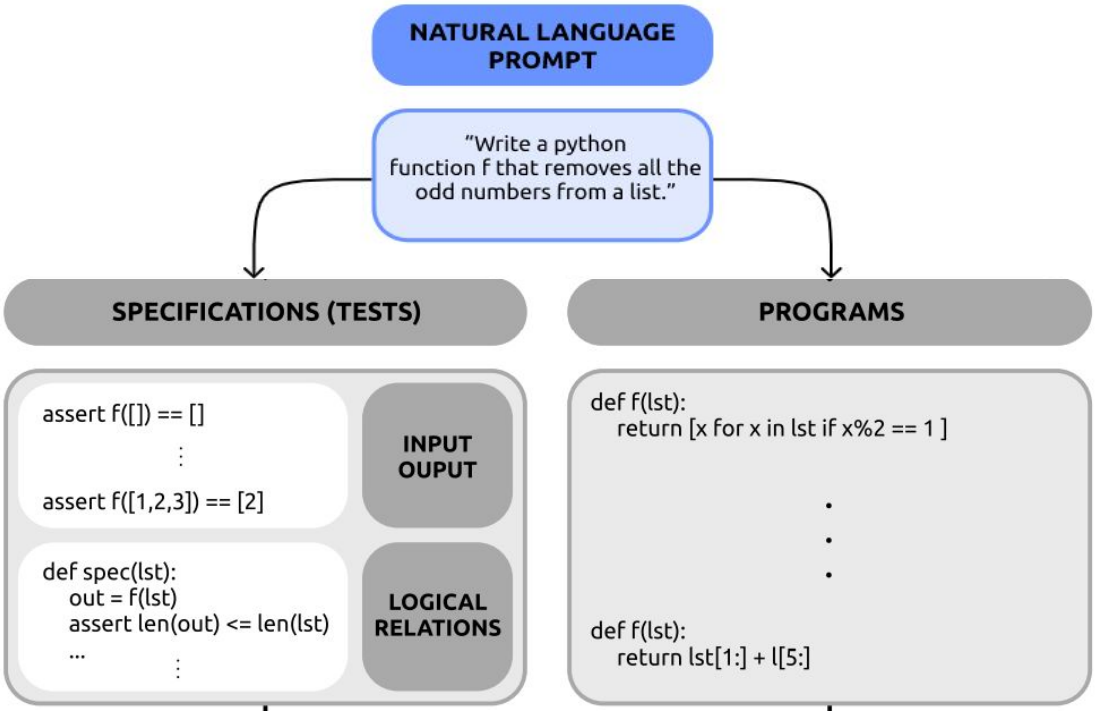
Speculyzer


NATURAL LANGUAGE PROMPT
"Write a python function f that removes all the odd numbers from a list."

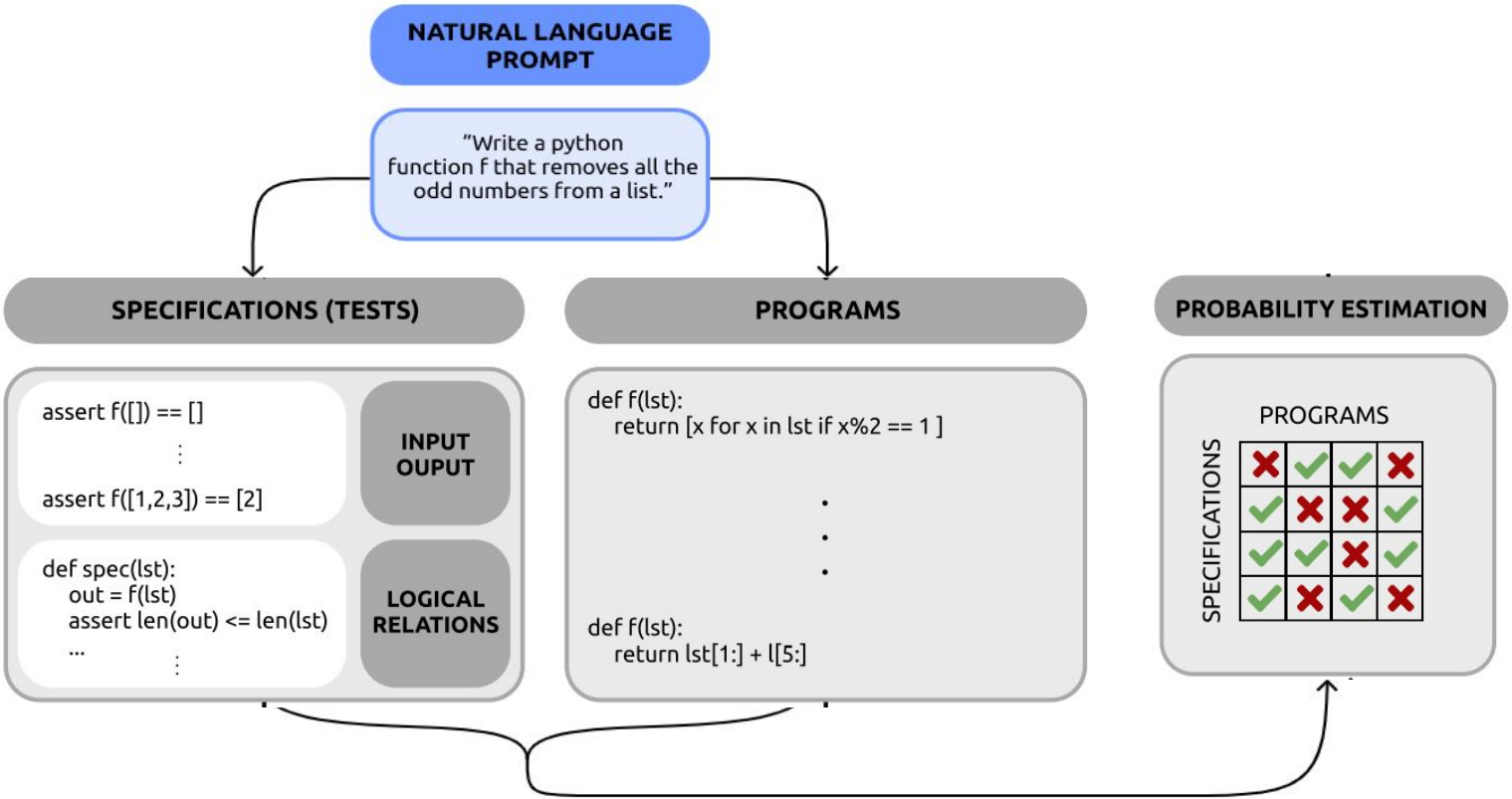
Speculyzer


NATURAL LANGUAGE PROMPT
"Write a python function f that removes all the odd numbers from a list."
PROGRAMS
def f(lst):
return [x for x in lst if x%2 == 1]
def f(lst):
return lst[1:] + l[5:]

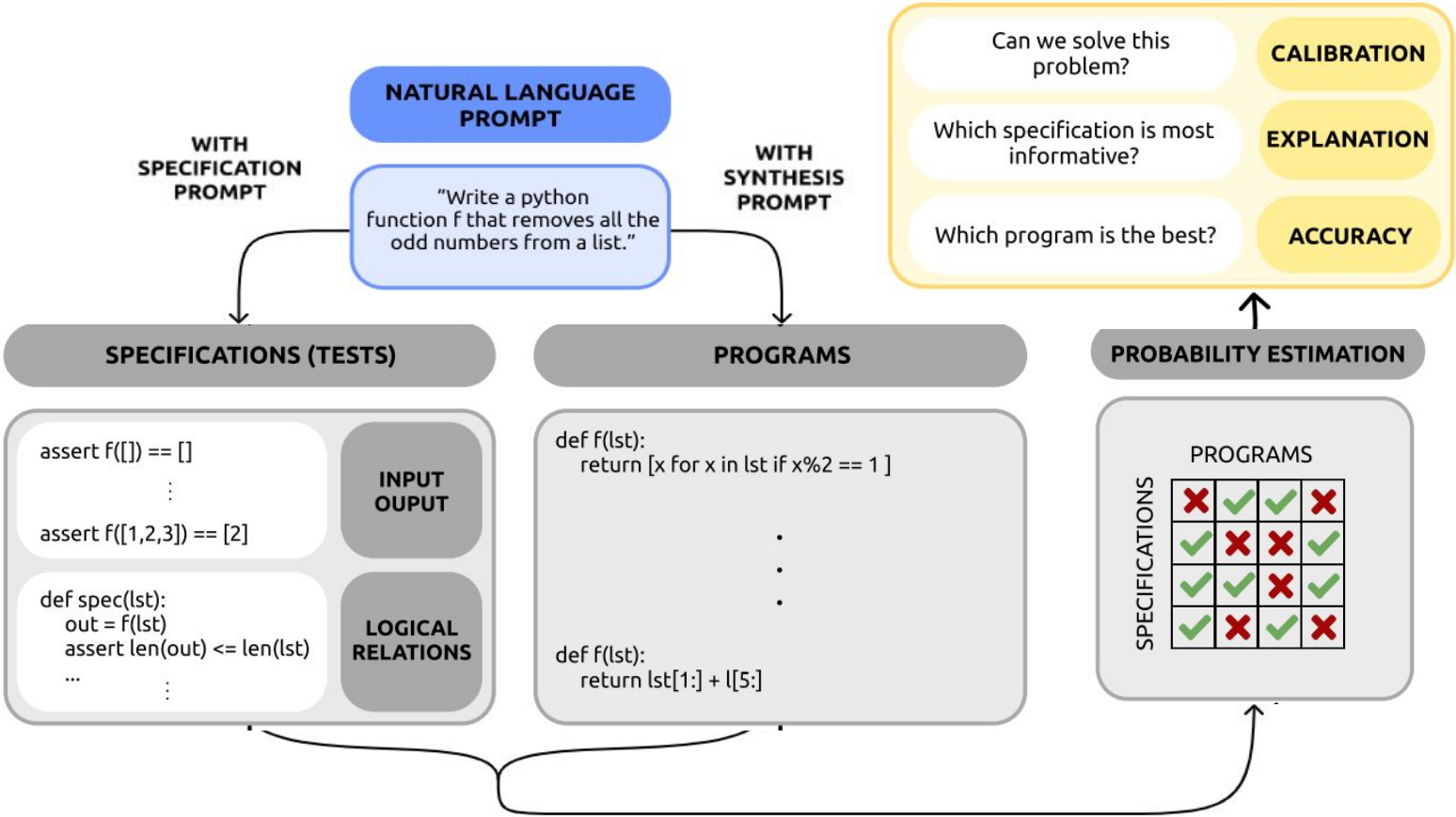
Speculyzer


NATURAL LANGUAGE PROMPT
"Write a python function f that removes all the odd numbers from a list."
SPECIFICATIONS (TESTS)
PROGRAMS
assert f([]) == []
assert f([1,2,3]) == [2]
INPUT OUPUT
def spec(lst):
out = f(lst)
assert len(out) <= len(lst)
...
LOGICAL RELATIONS
def f(lst):
return [x for x in lst if x%2 == 1]
def f(lst):
return lst[1:] + l[5:]

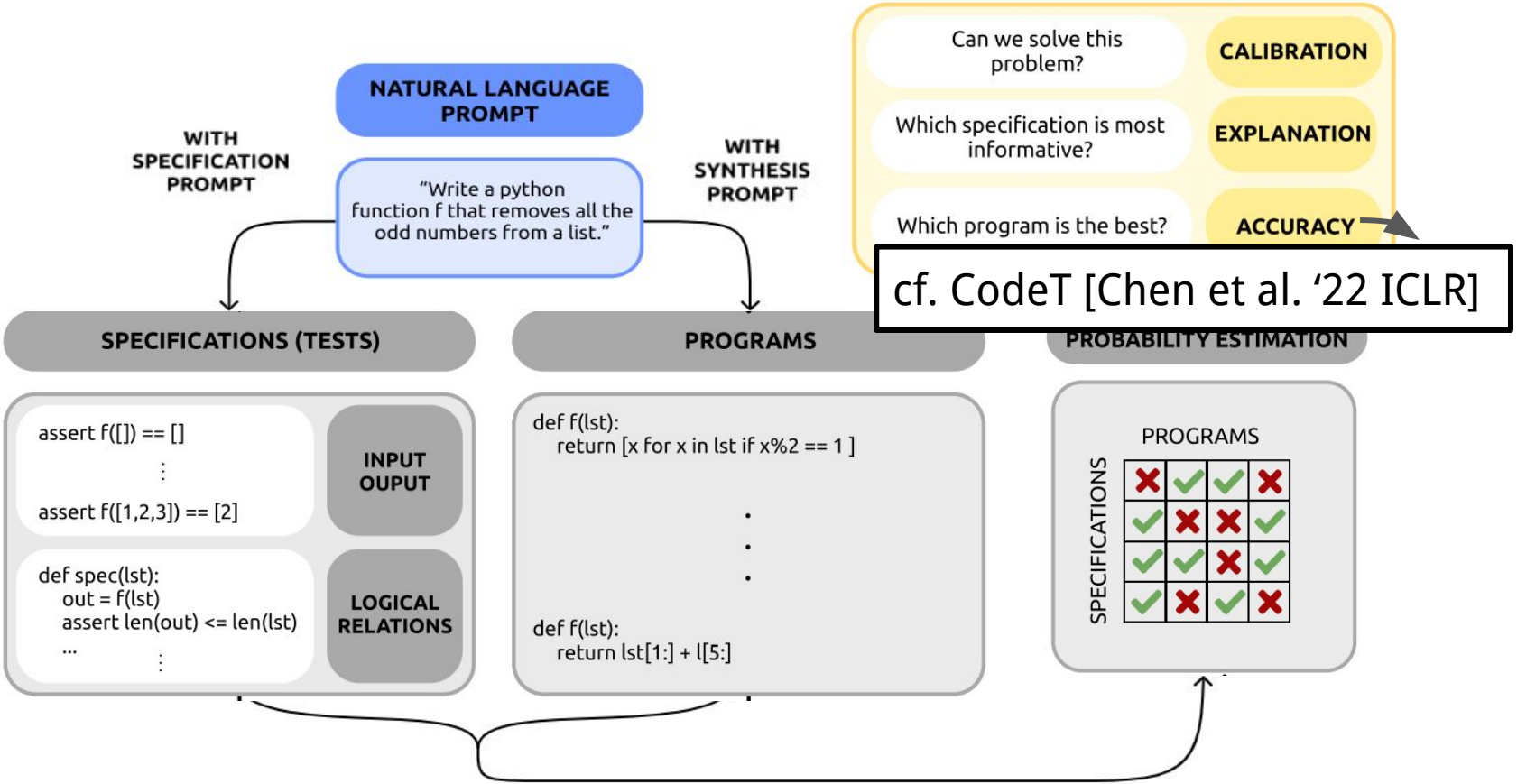
Speculyzer

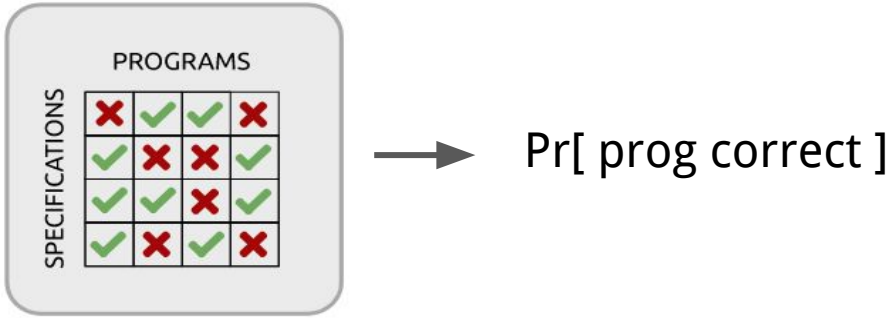

NATURAL LANGUAGE PROMPT
"Write a python function f that removes all the odd numbers from a list."
SPECIFICATIONS (TESTS)
PROGRAMS
PROBABILITY ESTIMATION
assert f([]) == []
assert f([1,2,3]) == [2]
INPUT OUPUT
def spec(lst):
out = f(lst)
assert len(out) <= len(lst)
LOGICAL RELATIONS
def f(lst):
return [x for x in lst if x%2 == 1]
def f(lst):
return lst[1:] + l[5:]
PROGRAMS
SPECIFICATIONS

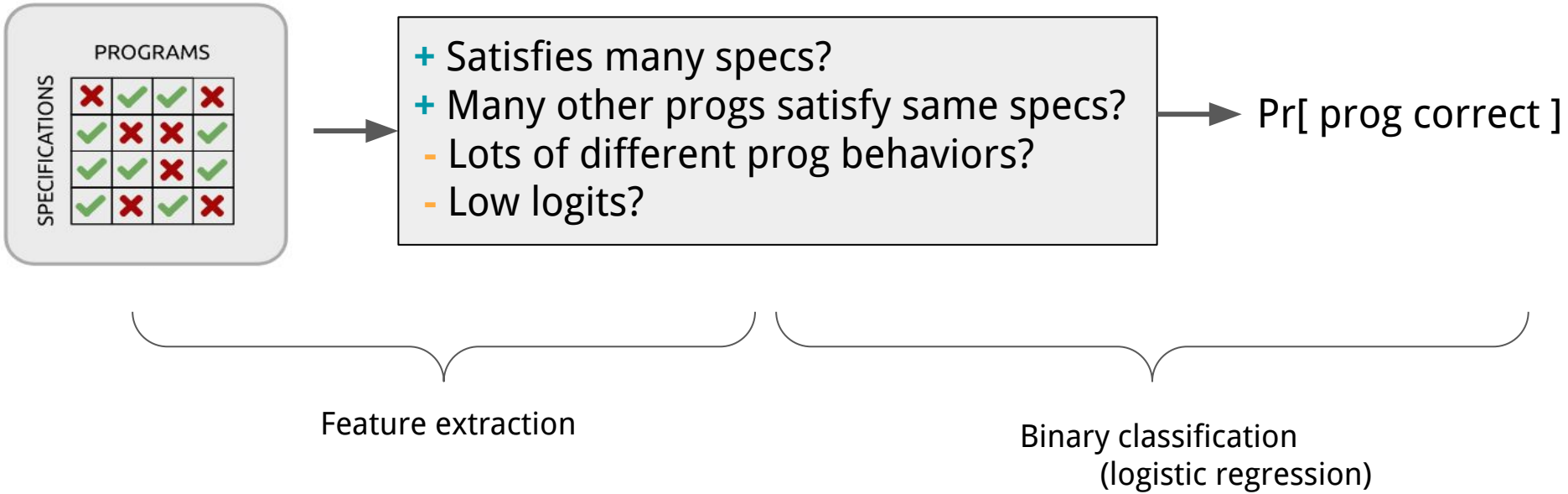
Speculyzer


NATURAL LANGUAGE PROMPT
"Write a python function f that removes all the odd numbers from a list."
WITH SPECIFICATION PROMPT
WITH SYNTHESIS PROMPT
Can we solve this problem?
CALIBRATION
Which specification is most informative?
EXPLANATION
Which program is the best?
ACCURACY
SPECIFICATIONS (TESTS)
PROGRAMS
PROBABILITY ESTIMATION
assert f([]) == []
assert f([1,2,3]) == [2]
INPUT OUPUT
def spec(lst):
out = f(lst)
assert len(out) <= len(lst)
LOGICAL RELATIONS
def f(lst):
return [x for x in lst if x%2 == 1]
def f(lst):
return lst[1:] + l[5:]
PROGRAMS
SPECIFICATIONS

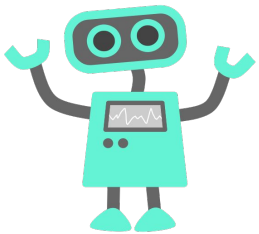
Speculyzer


NATURAL LANGUAGE PROMPT
"Write a python function f that removes all the odd numbers from a list."
WITH SPECIFICATION PROMPT
WITH SYNTHESIS PROMPT
Can we solve this problem?
CALIBRATION
Which specification is most informative?
EXPLANATION
Which program is the best?
ACCURACY
cf. CodeT [Chen et al. '22 ICLR]
SPECIFICATIONS (TESTS)
PROGRAMS
PROBABILITY ESTIMATION
assert f([]) == []
assert f([1,2,3]) == [2]
INPUT OUPUT
def spec(lst):
out = f(lst)
assert len(out) <= len(lst)
LOGICAL RELATIONS
def f(lst):
return [x for x in lst if x%2 == 1]
def f(lst):
return lst[1:] + l[5:]
PROGRAMS
SPECIFICATIONS

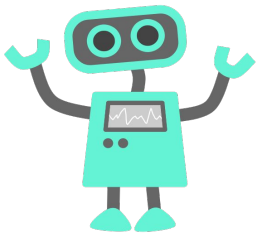

PROGRAMS
SPECIFICATIONS
Pr[prog correct]


PROGRAMS
SPECIFICATIONS
+ Satisfies many specs?
+ Many other progs satisfy same specs?
- Lots of different prog behaviors?
- Low logits?
Pr[prog correct]
Feature extraction
Binary classification
(logistic regression)

Speculyzer can decline to solve problems when uncertain

specification →  → I can/can’t solve that

Speculyzer can decline to solve problems when uncertain

specification →  →

I can/can’t solve that

$\exists$ prog :

Pr[prog correct] > 1-ε

Declining to solve problems when uncertain

Steps toward formalization

ϵ-correctness $\epsilon \in [0, 1]$

$$\mathbb{P}\big(p\,(I) \neq \bot \wedge p(I) \not\models \phi\big) \leq \epsilon$$

δ-coverage $\delta \in [0, 1]$

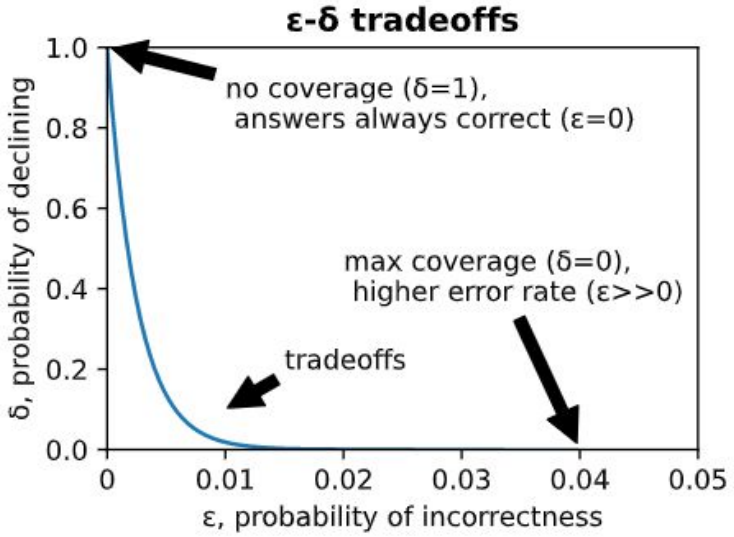
$$\mathbb{P}\big(p\,(I) = \bot\big) \leq \delta$$

p : synthesized program

I : user's input intent

ϕ : formal specification

$\bot$: decline to solve problem



Reality Check:

No neural net english->code system will achieve 0% errors

Task:

Generate tests that explain the generated code, so if it is incorrect, the user can discard it

PROGRAM

```
def is_bored(S):
    """You'll be given a string of words, and your task is to count the number
    of boredoms. A boredom is a sentence that starts with the word "I".
    Sentences are delimited by '.', '?' or '!'.
    For example:
    >>> is_bored("Hello world")
    0
    >>> is_bored("The sky is blue. The sun is shining. I love this weather")
    1"""
```

How to communicate what `program` does so that the user can accept/reject it?

$$\texttt{program} \vdash \text{specification}$$

Out-of-the-box LLM

PROGRAM

```
def is_bored(S):
    """You'll be given a string of words, and your task is to count the number
    of boredoms. A boredom is a sentence that starts with the word "I".
    Sentences are delimited by '.', '?' or '!'.
    For example:
    >>> is_bored("Hello world")
    0
    >>> is_bored("The sky is blue. The sun is shining. I love this weather")
    1"""
    boredoms = 0
    # replace . or ! or ? with . to simplify this problem
    S = S.replace('.','. ')
    S = S.replace('!','! ')
    S = S.replace('?','? ')
    sentences = S.split(' ')
    for sentence in sentences:
        if sentence.startswith('I'): boredoms = boredoms + 1
    return boredoms
```

Out-of-the-box LLM

PROGRAM

```
def is_bored(S):
    """You'll be given a string of words, and your task is to count the number
    of boredoms. A boredom is a sentence that starts with the word "I".
    Sentences are delimited by '.', '?' or '!'.
    For example:
    >>> is_bored("Hello world")
    0
    >>> is_bored("The sky is blue. The sun is shining. I love this weather")
    1"""
    boredoms = 0
    # replace . or ! or ? with . to simplify this problem
    S = S.replace('.','. ')
    S = S.replace('!','! ')
    S = S.replace('?','? ')
    sentences = S.split(' ')
    for sentence in sentences:
        if sentence.startswith('I'): boredoms = boredoms + 1
    return boredoms
```

RANDOM INPUT/ OUTPUT

```
assert is_bored("I love this weather.") == 1
```

How to communicate what `program` does so that the user can accepted/reject it?

$$\operatorname{argmax}\ \text{objective\_function}(prog, spec)$$
$$prog \vdash spec$$

Pick the thing which is true about the program
But which is not true about most other programs
“distinguishing”, “selective”

PROGRAM

```
def is_bored(S):
    """You'll be given a string of words, and your task is to count the number
    of boredoms. A boredom is a sentence that starts with the word "I".
    Sentences are delimited by '.', '?' or '!'.
    For example:
    >>> is_bored("Hello world")
    0
    >>> is_bored("The sky is blue. The sun is shining. I love this weather")
    1"""
    boredoms = 0
    # replace . or ! or ? with . to simplify this problem
    S = S.replace('.','. ')
    S = S.replace('!','! ')
    S = S.replace('?','? ')
    sentences = S.split(' ')
    for sentence in sentences:
        if sentence.startswith('I'): boredoms = boredoms + 1
    return boredoms
```

RANDOM INPUT/ OUTPUT

```
assert is_bored("I love this weather.") == 1
```

PROGRAM

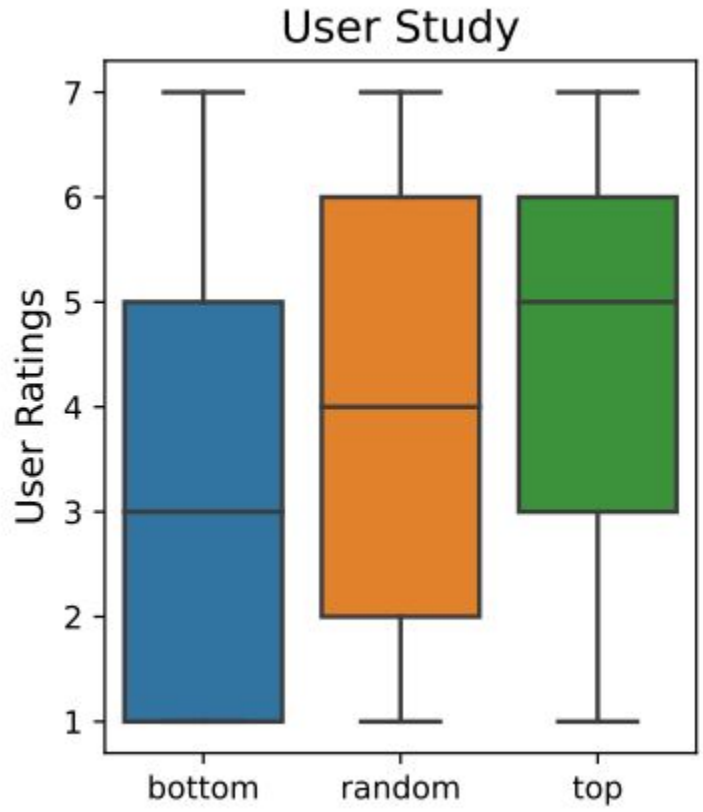
```
def is_bored(S):
    """You'll be given a string of words, and your task is to count the number
    of boredoms. A boredom is a sentence that starts with the word "I".
    Sentences are delimited by '.', '?' or '!'.
    For example:
    >>> is_bored("Hello world")
    0
    >>> is_bored("The sky is blue. The sun is shining. I love this weather")
    1"""
    boredoms = 0
    # replace . or ! or ? with . to simplify this problem
    S = S.replace('.','. ')
    S = S.replace('!','! ')
    S = S.replace('?','? ')
    sentences = S.split(' ')
    for sentence in sentences:
        if sentence.startswith('I'): boredoms = boredoms + 1
    return boredoms
```

DISTINGUISHING INPUT/ OUTPUT

```
assert is_bored("I have no idea what I'm doing") == 2
```

RANDOM INPUT/ OUTPUT

```
assert is_bored("I love this weather.") == 1
```



Worst of 100 LLM samples

Raw LLM

Top of 100 LLM samples

Speculyzer Recap

Task: Predict if the synthesizer should be trusted

Task: Generate informative test(s) that explain the generated code

What could trust unlock?

[ellisk42/ec] Bump protobuf from 3.8.0 to 3.15.0 (PR #90) Inbox ×

dependabot[bot] <notifications@github.com> Unsubscribe
to ellisk42/ec, Subscribed

This automated pull request fixes a security vulnerability (high severity).

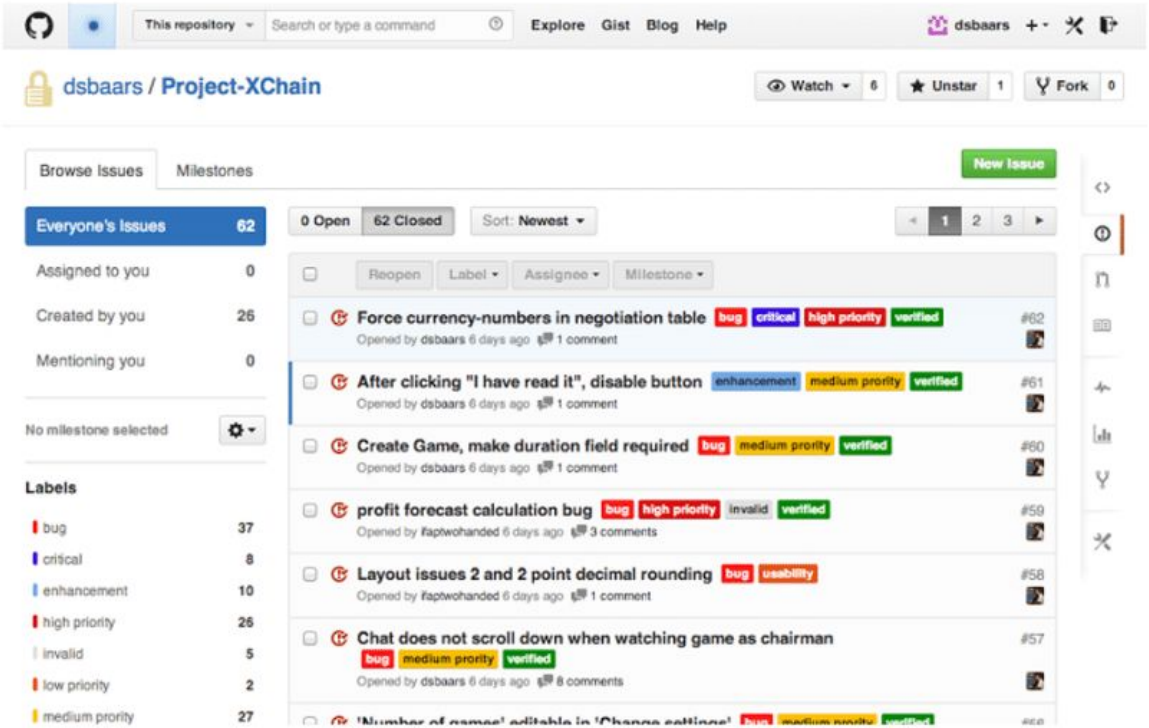
Learn more about Dependabot security updates.

Bumps protobuf from 3.8.0 to 3.15.0.

Release notes

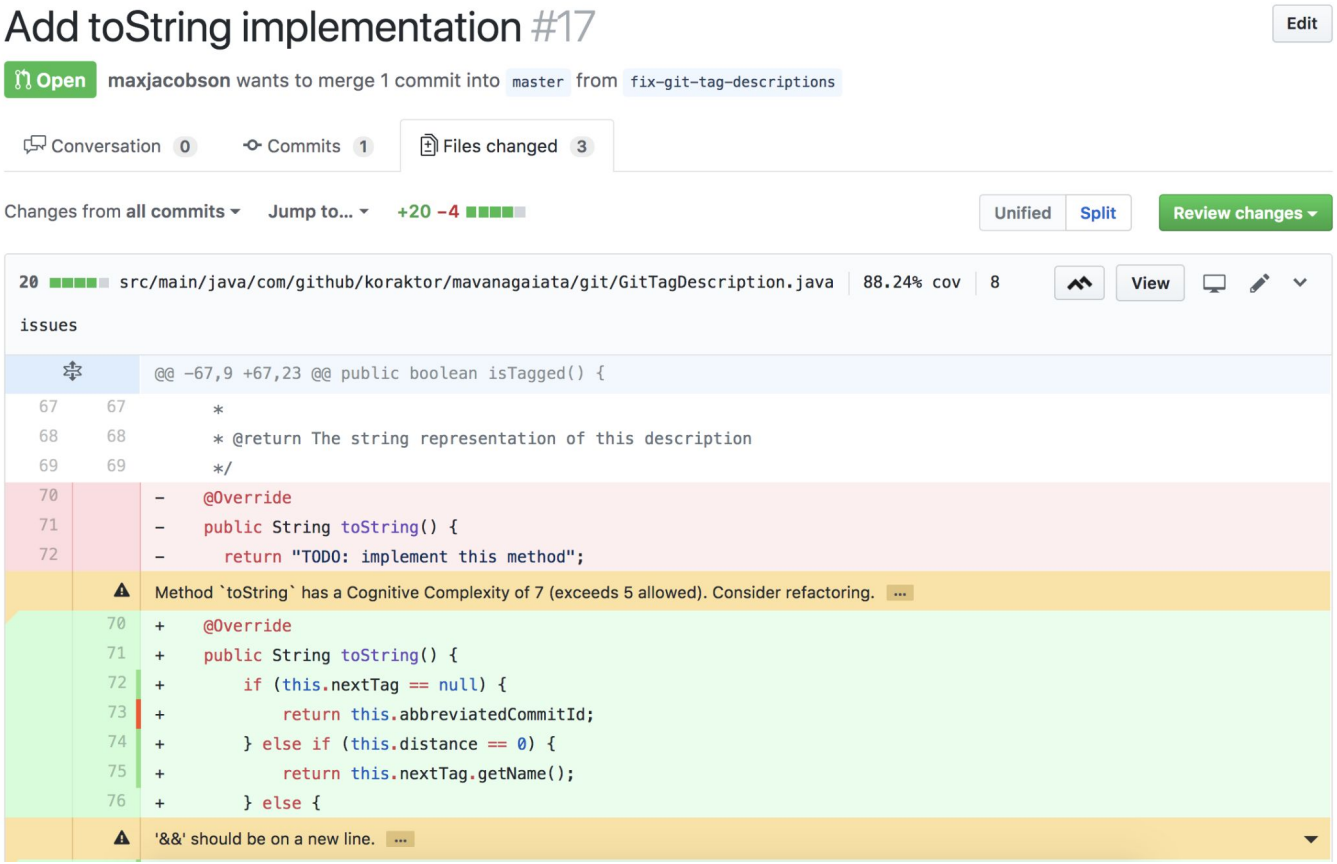
Sourced from protobuf's releases.

<science_fiction>



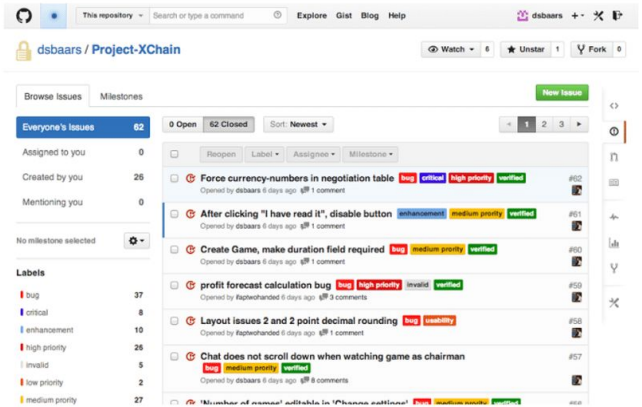
</science_fiction>

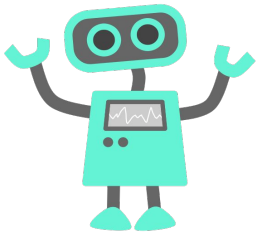
<science_fiction>

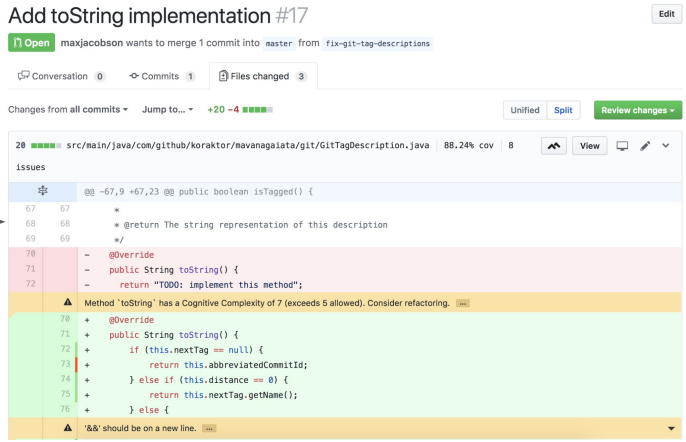


</science_fiction>

<science_fiction>







</science_fiction>

Task:

Neurosymbolic programming for perception and graphics

Joint work with Hao Tang!
ICML ‘23

From Perception to Programs: Regularize, Overparameterize, and Amortize

Hao Tang [1] Kevin Ellis [1]

Domain Specific Languages, Pre-provided Symbols

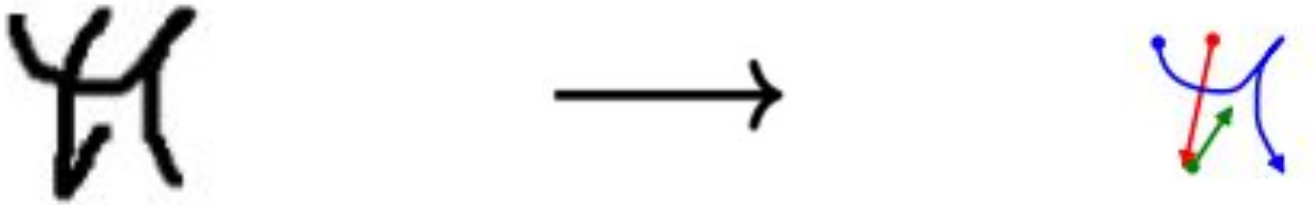
Neural Program Generator

Neural Program Executor

```
Draw("Top", "Circle", position, geometry)

for(i < 2, "translation", a)
  for(j < 2, "translation", b)
    Draw("Leg", "Cub", position + i*a + j*b, geometry)

for(i < 2, "translation", c)
  Draw("Layer", "Rec", position + i*c, geometry)
```

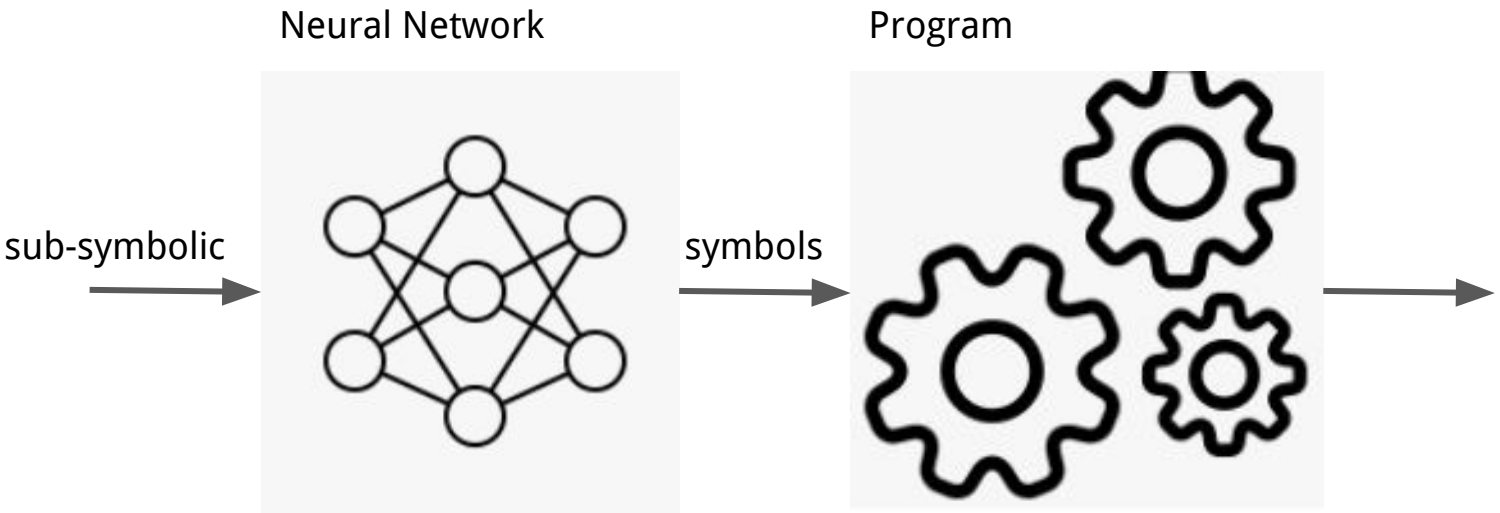
Tian et al. 2019, ICLR

Flexible Symbols

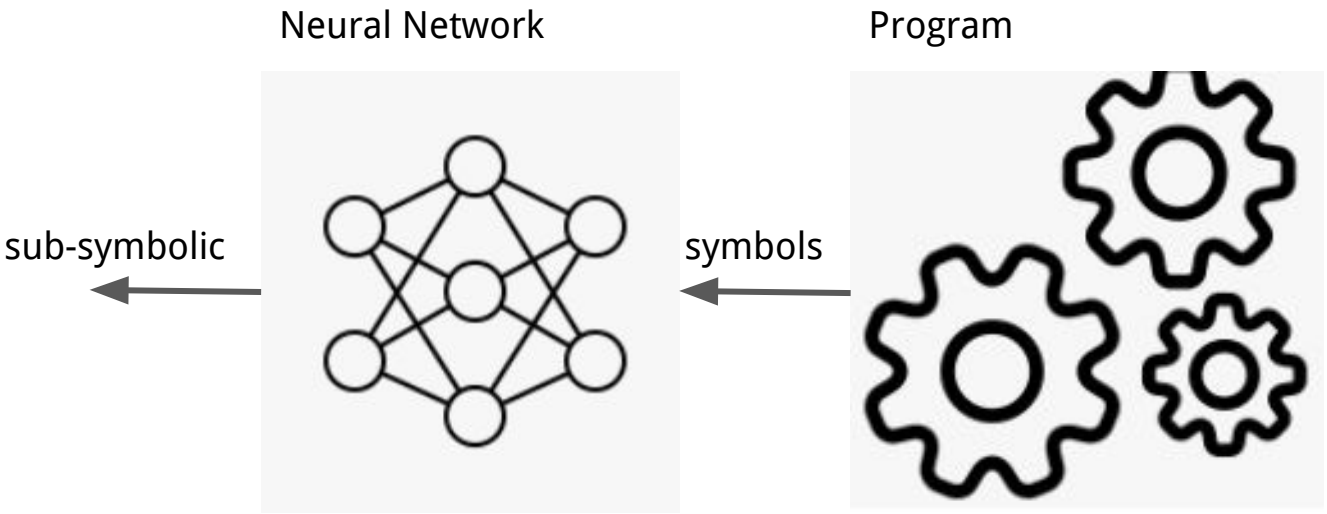


Feinman & Lake 2021

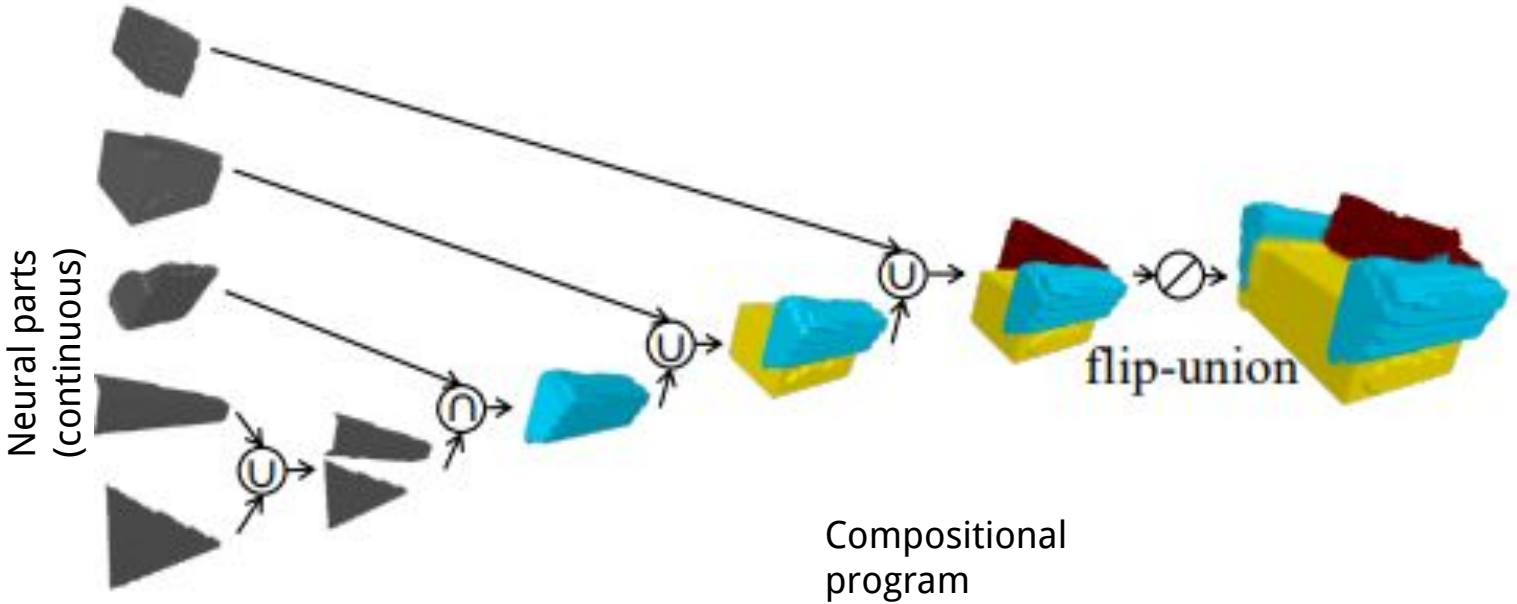
Flexible, Learnable Symbols, AND
Powerful, Program-Like Compositional Processing?



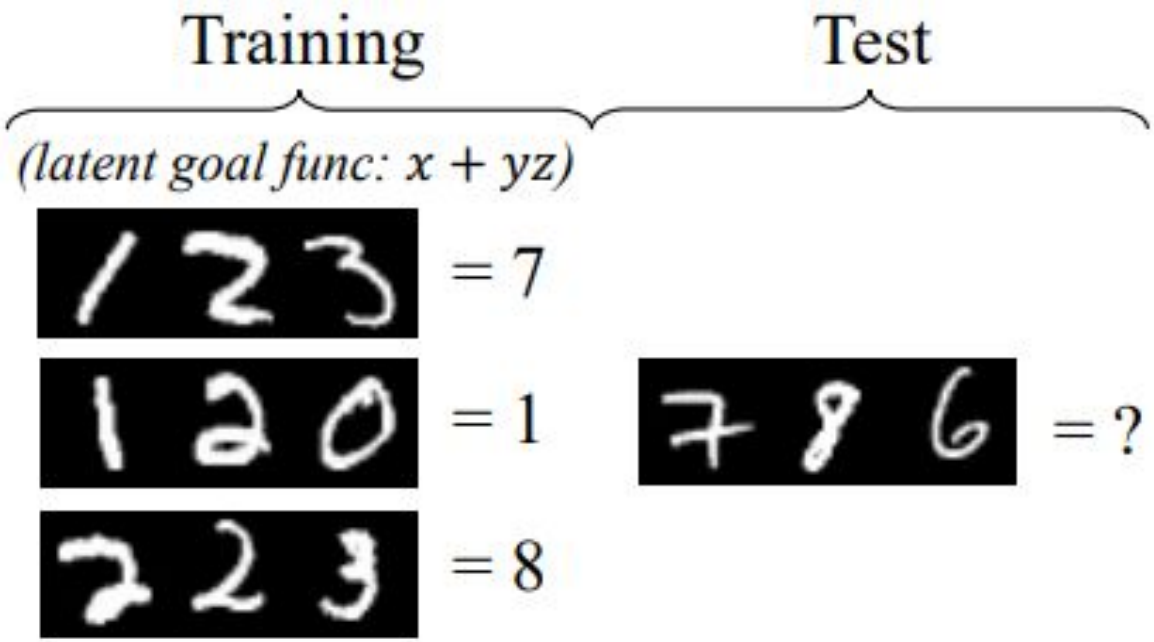
Flexible, Learnable Symbols, AND Powerful, Program-Like Compositional Processing? (generative)



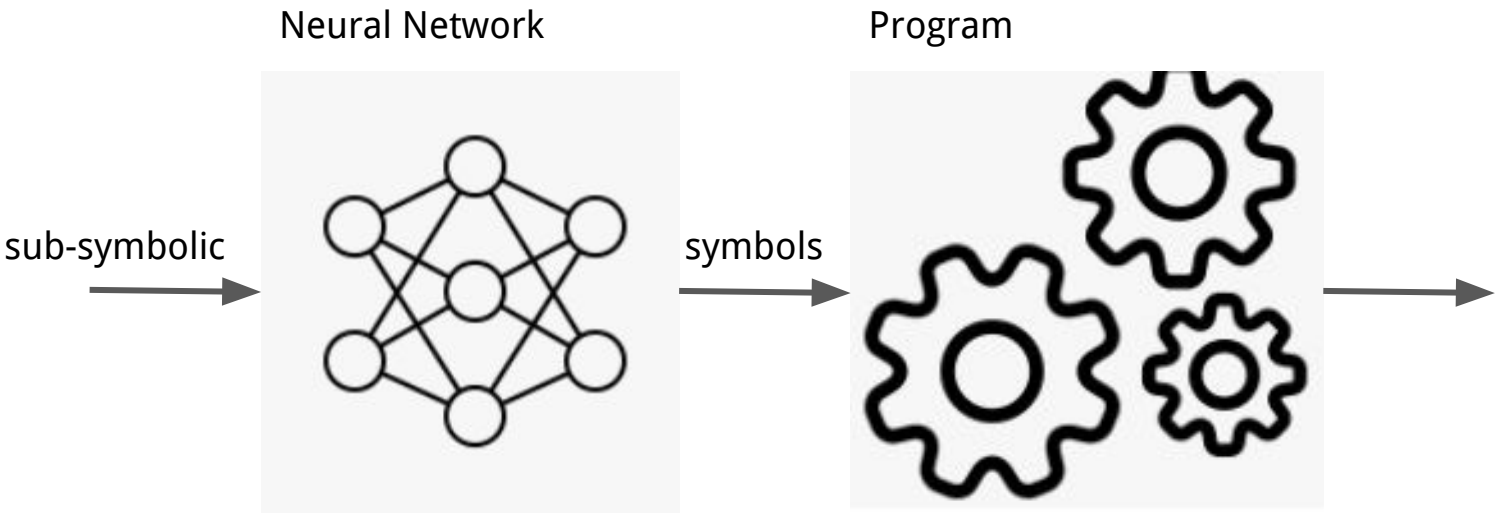
Generative Example



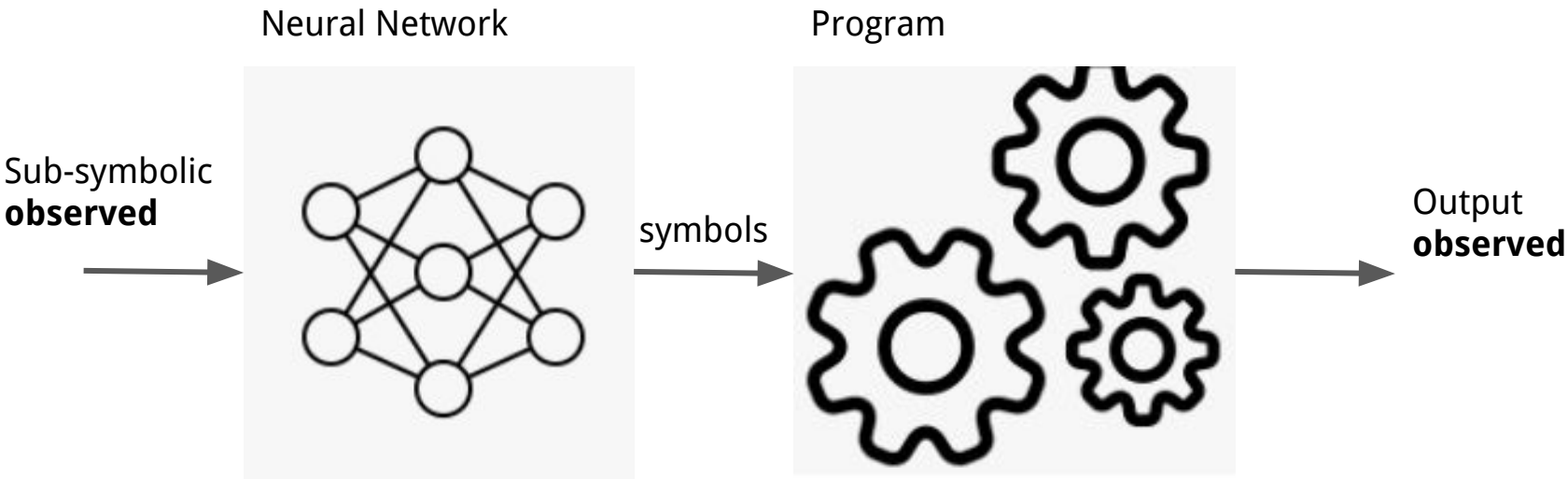
Discriminative Example



Learning Problem



Learning Problem

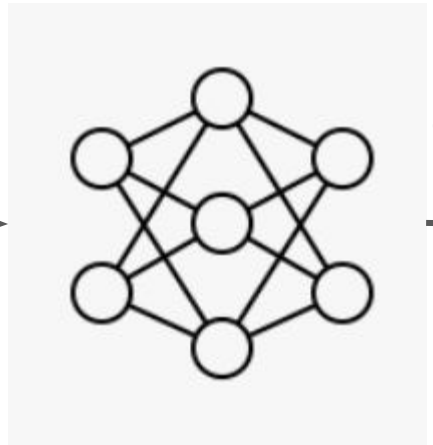


Learning Problem

Neural Network
latent weights

Program
Latent structure

Sub-symbolic
observed



symbols
latent

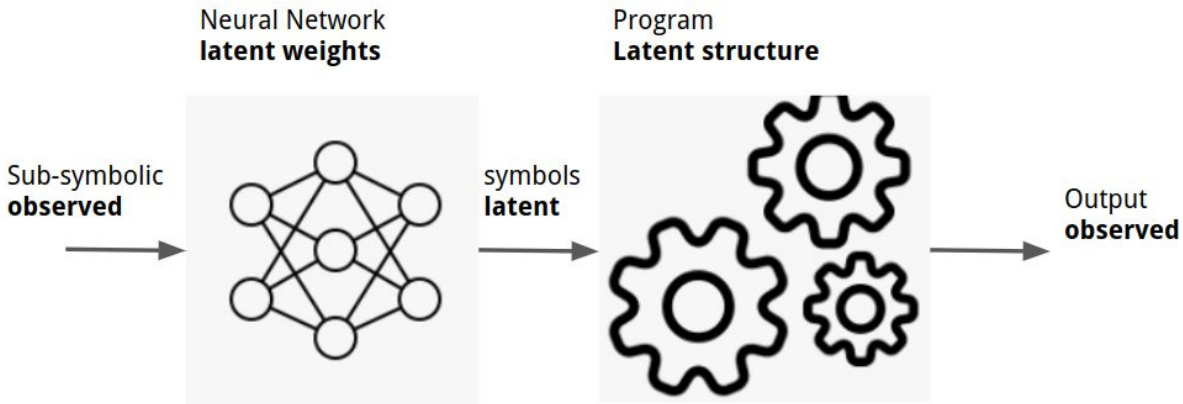
Output
observed

Challenges

Mixed Discrete-Continuous

Underconstrained

Symbol Grounding Problem

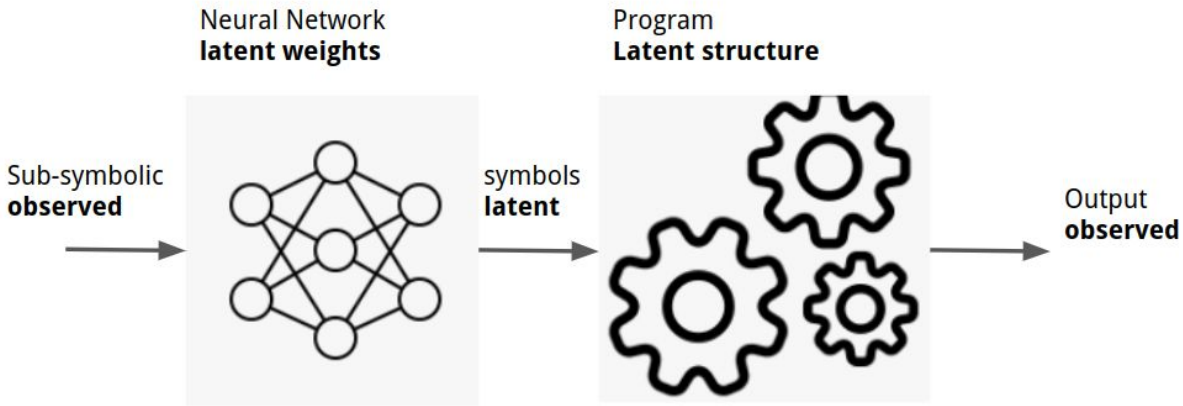


Opportunities

Compositional generalization from raw perceptual input

Systems that learn their own symbols!

Synthesizing neurosymbolic programs

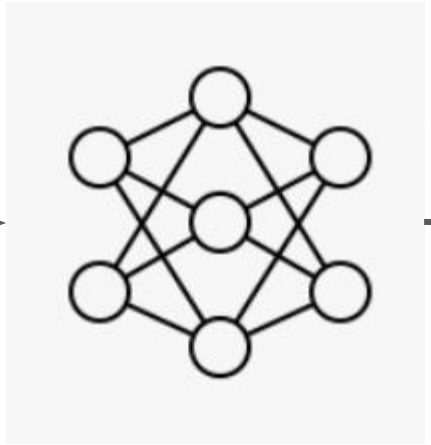


Guiding Principles:
Make Everything Continuous. Use Gradient Descent.

Neural Network
latent weights

Program
Latent ~~structure~~ continuous parameters

Sub-symbolic
observed



symbols
latent

Output
observed

Relaxing discrete program spaces

Many approaches:

Terpret [Gaunt et al 2016]

dILP [Evans et al 2017]

EQLNet [Sahoo et al 2018]

DiffLog [Si et al 2019]

…

ROAP, this work [Tang et al 2023]

Relaxing discrete program spaces

Many approaches:

Terpret [Gaunt et al 2016]

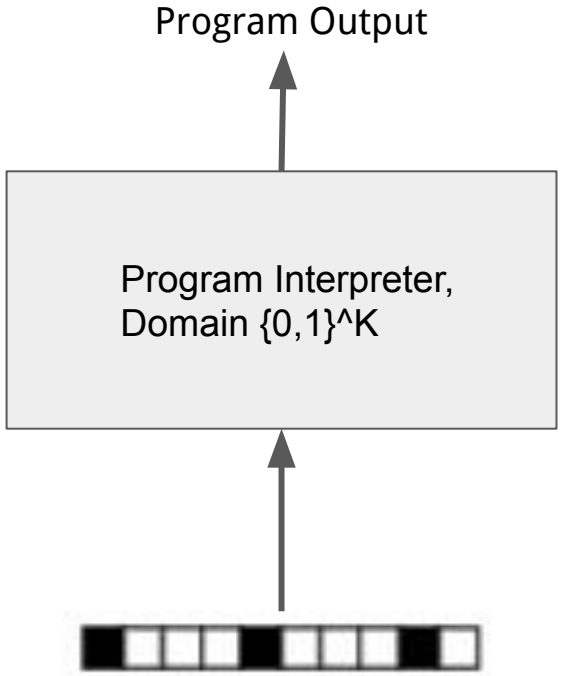
dILP [Evans et al 2017]

EQLNet [Sahoo et al 2018]

DiffLog [Si et al 2019]

…

ROAP, this work [Tang et al 2023]



Relaxing discrete program spaces

Many approaches:

Terpret [Gaunt et al 2016]

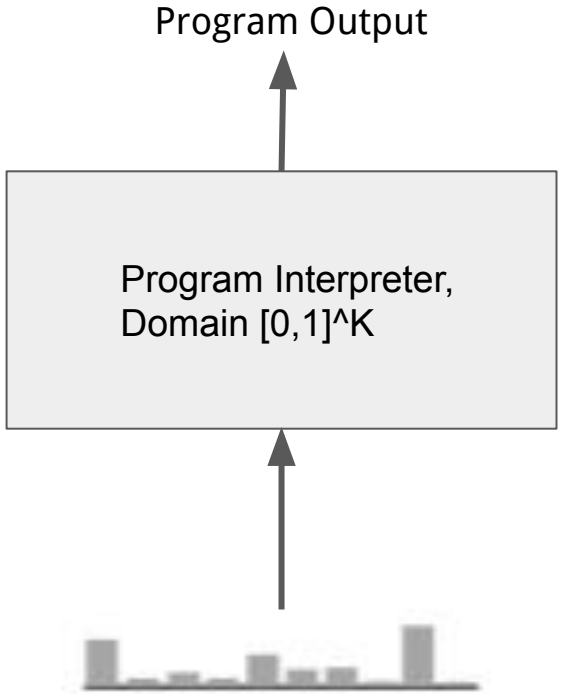
dILP [Evans et al 2017]

EQLNet [Sahoo et al 2018]

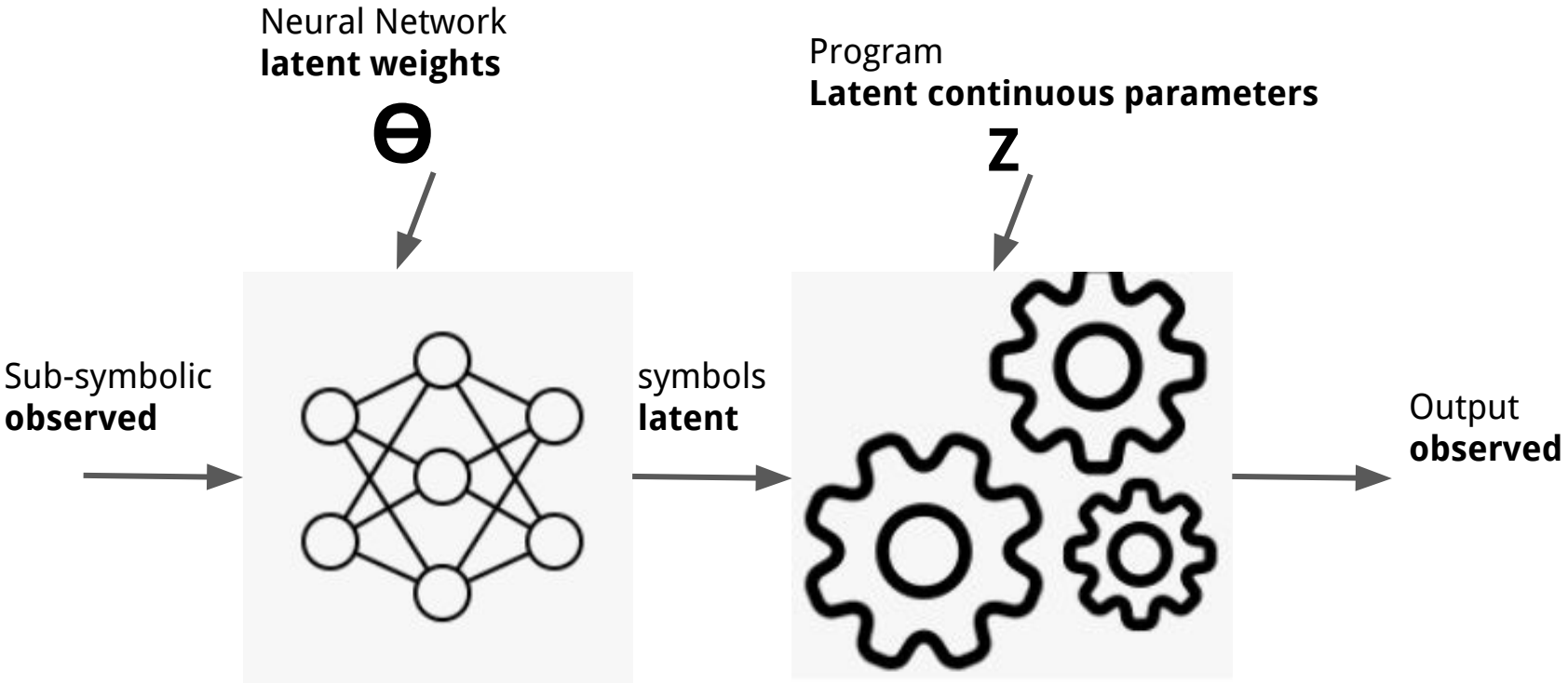
DiffLog [Si et al 2019]

…

ROAP, this work [Tang et al 2023]



A Standard Deep Learning Problem



Out-of-the-box gradient descent doesn’t work

Need extra tricks:

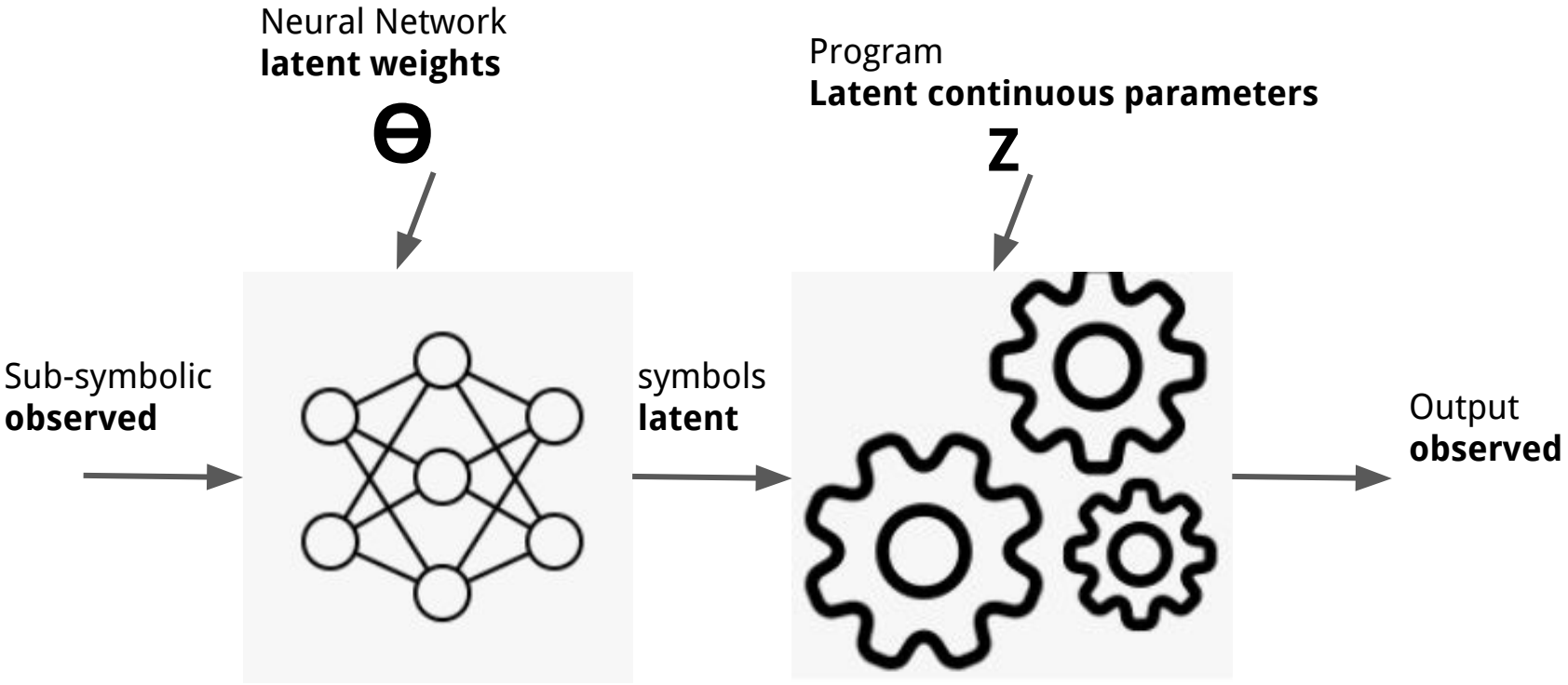
1. Multitasking
2. Overparameterization
3. Special regularizer

Out-of-the-box gradient descent doesn’t work

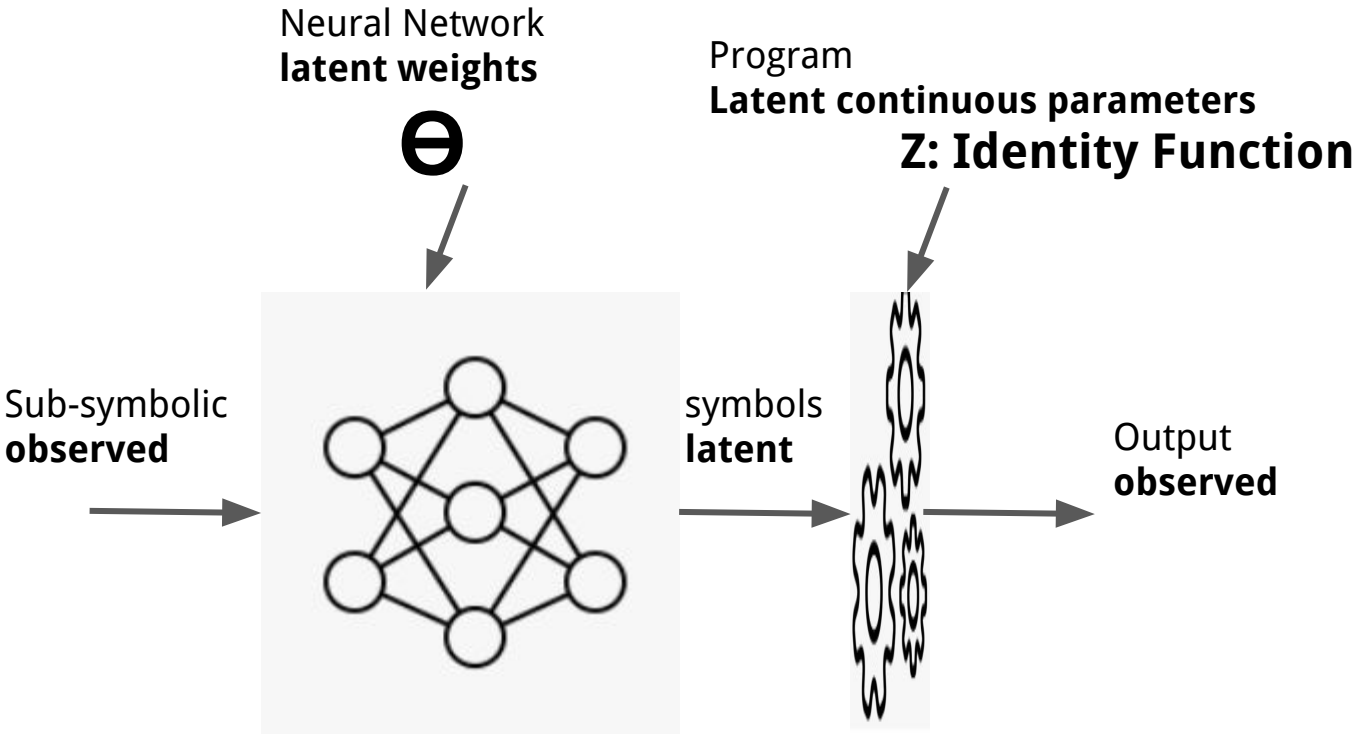
Need extra tricks:

1. **Multitasking**
2. Overparameterization
3. Special regularizer

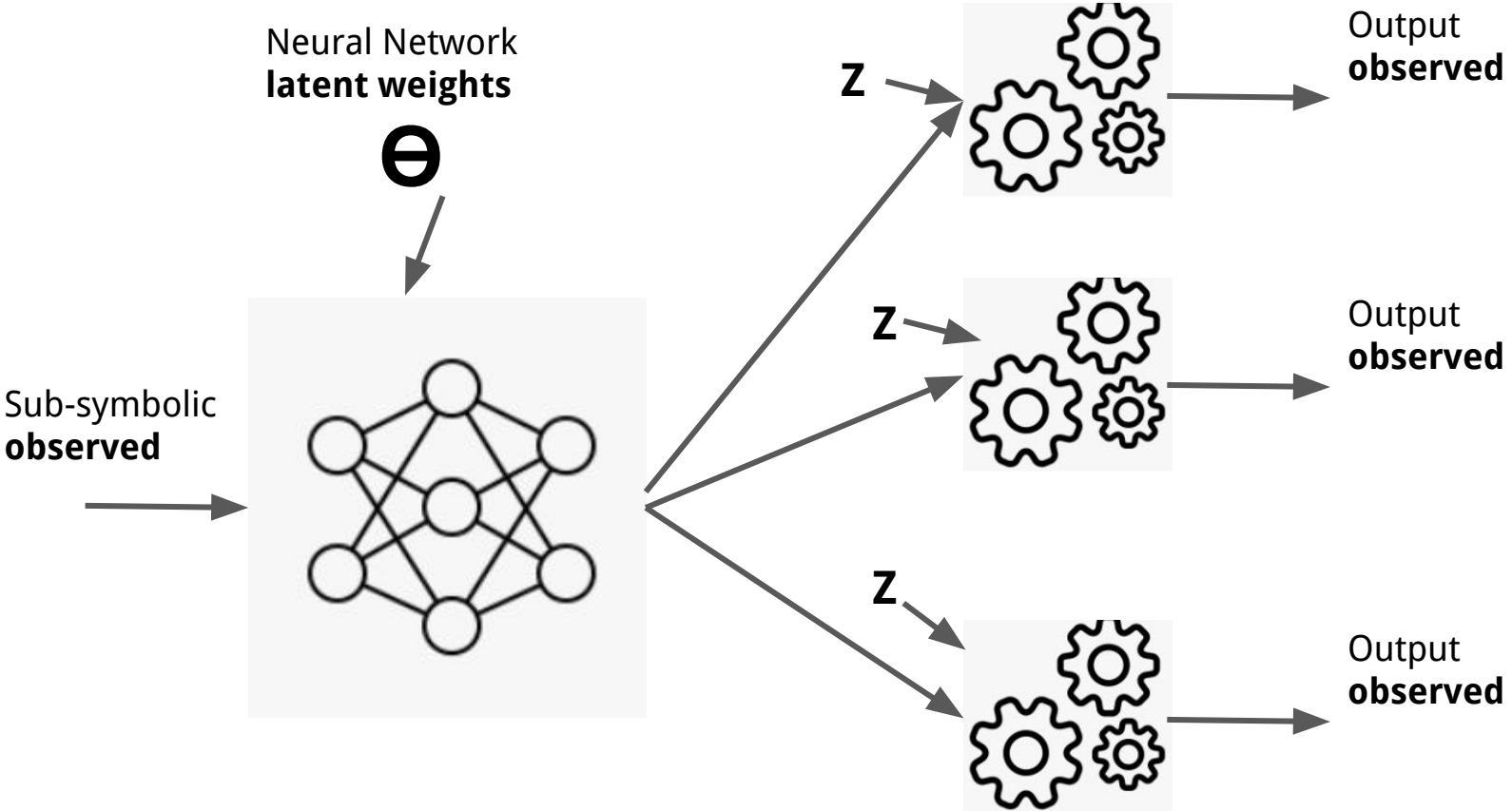
Under-constrained Optimization Problem



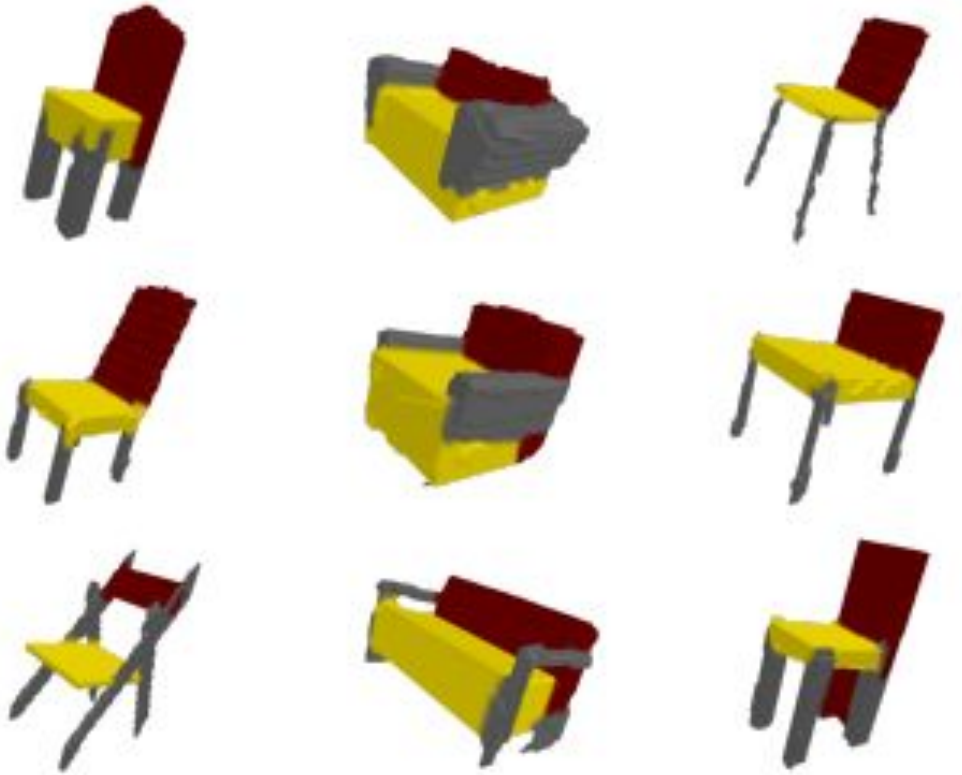
Under-constrained Optimization Problem



Under-constrained Optimization Problem: Therefore, Multitask



Under-constrained Optimization Problem: Therefore, Multitask



Out-of-the-box gradient descent doesn’t work

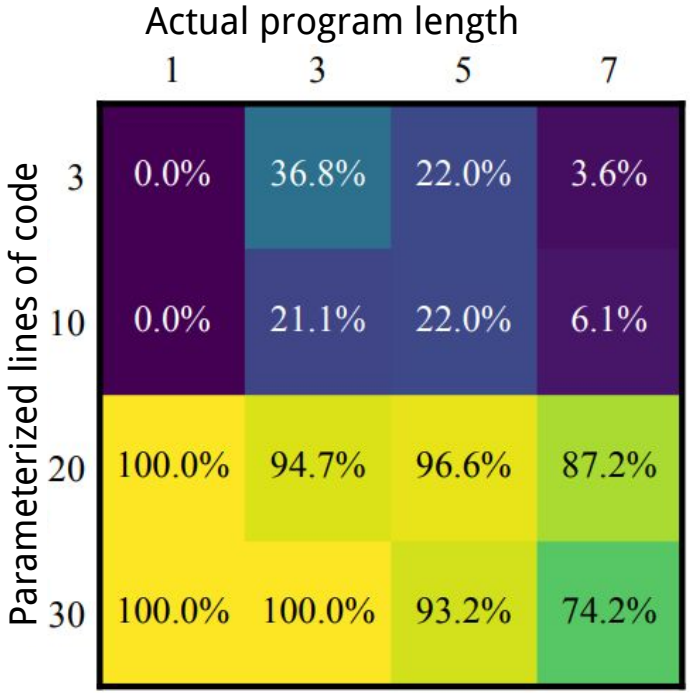
Need extra tricks:

1. Multitasking
2. **Overparameterization**
3. Special regularizer

Overparameterization

Neural nets converge with gradient descent due to massively overparametrization

Analog of overparameterization for programs: Give it more lines of code



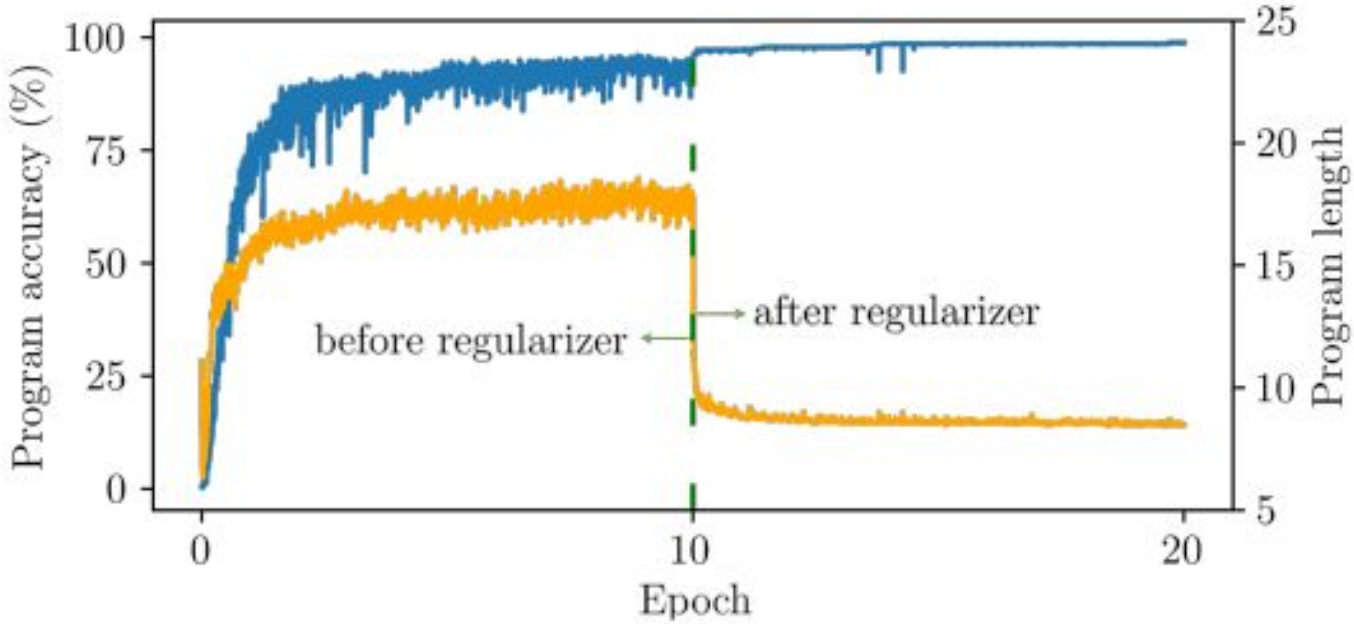
Out-of-the-box gradient descent doesn’t work

Need extra tricks:

1. Multitasking
2. Overparameterization
3. **Special regularizer**

New regularizer for relaxed programs: Softly penalized expected program length

Overparameterize: Converge… to long, bloated, overfit programs

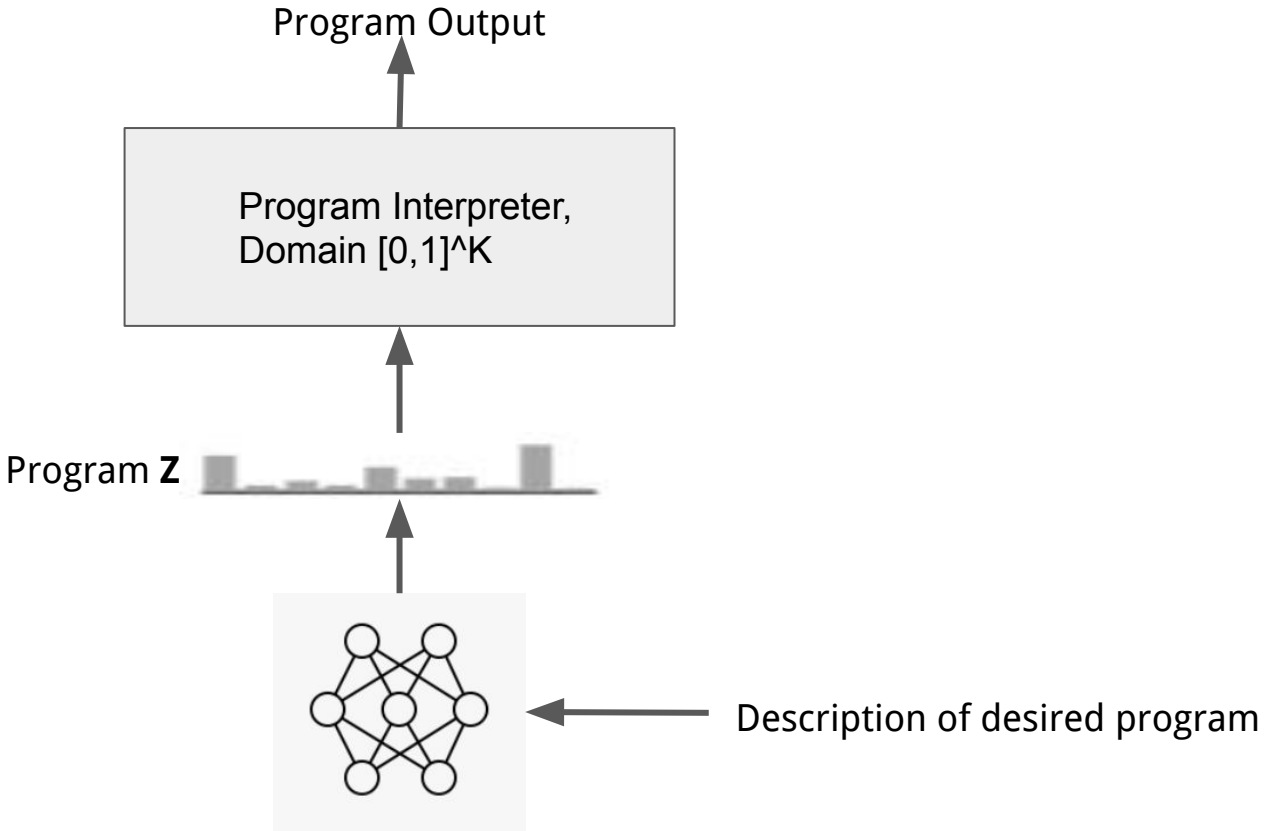


Out-of-the-box gradient descent doesn’t work

Need extra tricks:

1. Multitasking, **which allows amortized program synthesis**
2. Overparameterization
3. Special regularizer

Amortized Program Synthesis: Learning to Generate Programs



Justifying Each Trick

1. Need multitasking in order to force non-degenerate solutions, imposes extra constraints

2. Need overparameterization for gradient descent to find programs

3. Overparametrization causes bad programs => Need special regularizer

4. Multitasking allows learning to find programs

Justifying Each Trick

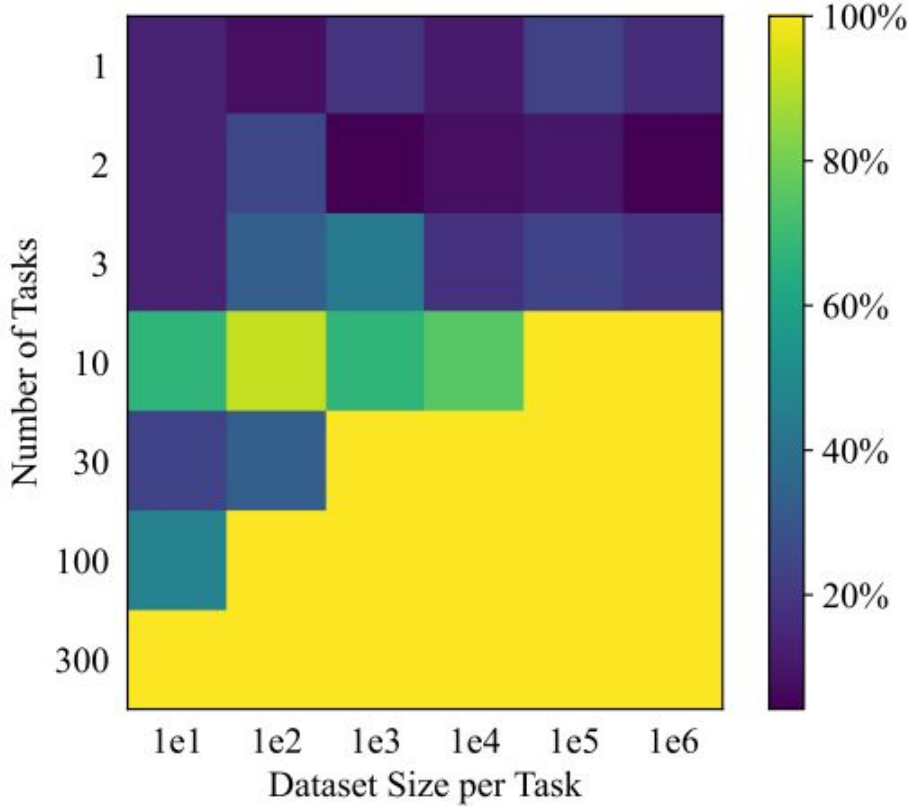
1. Need multitasking in order to force non-degenerate solutions,
   imposes extra constraints

2. Need overparameterization for gradient descent to find programs

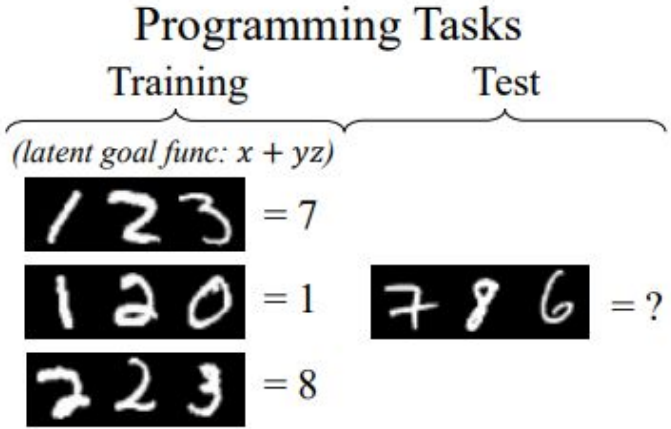
3. Overparametrization causes bad programs => Need special regularizer

4. Multitasking allows learning to find programs

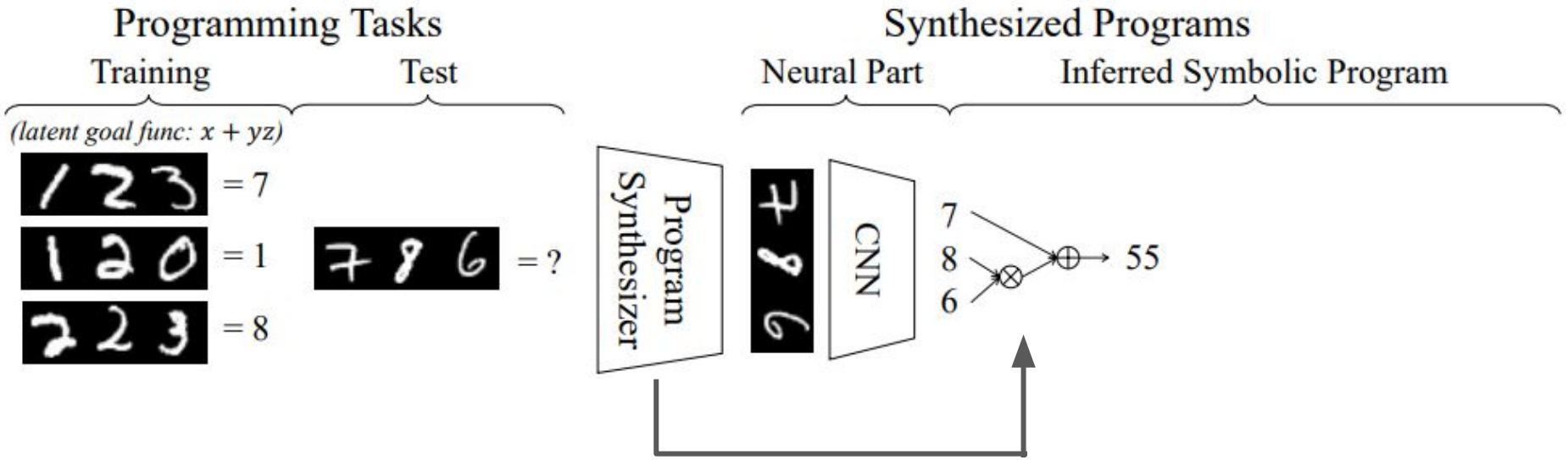
Constraints necessary,
But There Exists More Than One Kind of Constraint



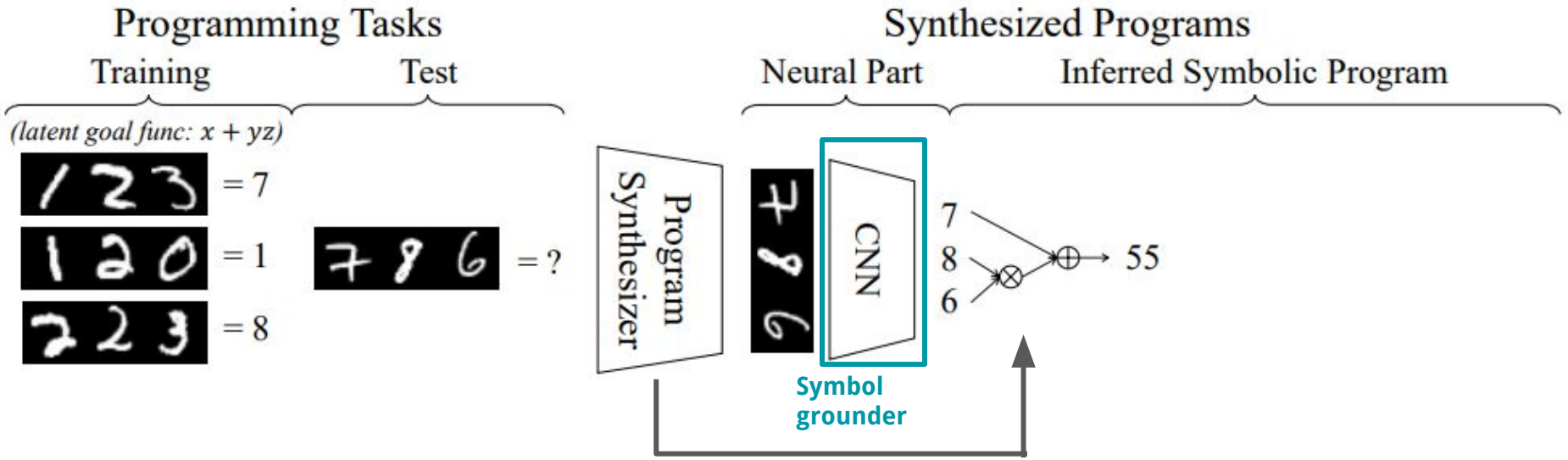
Results: Discovering Number Concepts



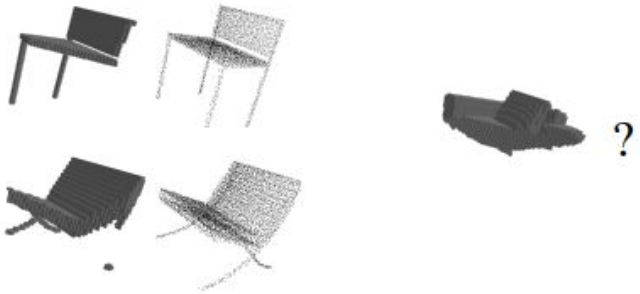
Results: Discovering Number Concepts



Results: Discovering Number Concepts

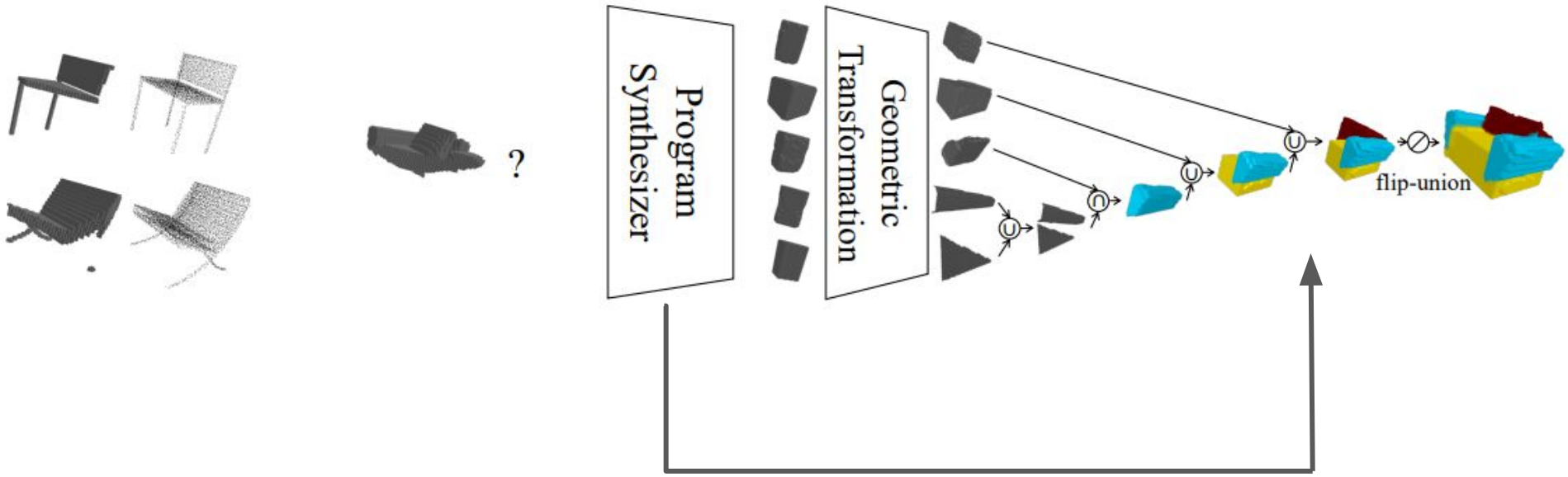


Results: 3D Reconstruction

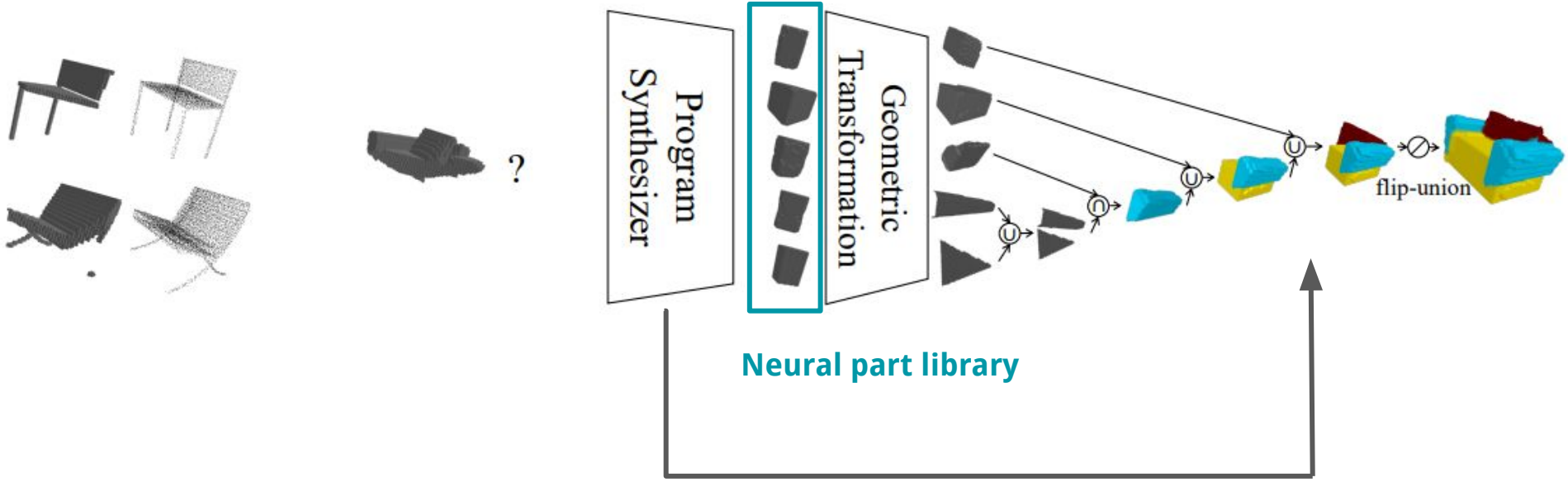


?

Results: 3D Reconstruction



Results: 3D Reconstruction



Lessons

Symbol Grounding can emerge from the interaction of different constraints

Doesn’t *need* natural language as scaffolding

Gradient descent works for neural nets because of “gradient gadgets”

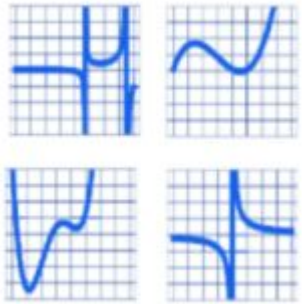
Need to *invent new gadgets* for neurosymbolic programs

the end!

Learning to make programming problems

Simplified setting: Programs take no arguments and their output is the spec

Symbolic Regression

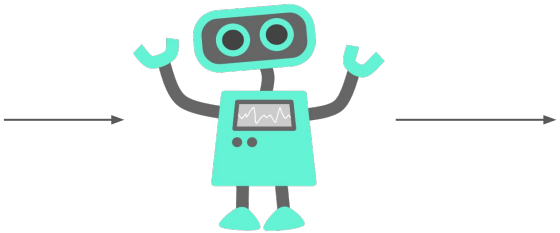


$y = f(x)$

LOGO/Turtle

```
for w in range(6):
  s0=get_state()
  draw_square(w)
  reset_state(s0)
```

Simplified Setting



```
for w in range(6):
  s0=get_state()
  draw_square(w)
  reset_state(s0)
```