Machine Learning for Assisting Software Development

Danny Tarlow
dtarlow@google.com

Research Scientist – Google Research, Brain team
Adjunct Professor – McGill Computer Science
Core Industrial Member – Mila Quebec AI Institute

Machine Learning for Software Engineering

New and exciting research area

~450 papers at https://ml4code.github.io/, majority since 2013.

Will only do justice to a small slice today.

See also Allamanis et al. (2018), "A Survey of Machine Learning for Big Code and Naturalness." *ACM Computing Surveys 2018*.

Research Statement

Learn to Program –

Build systems that use machine learning to acquire a general set of programming skills – generating, editing, and communicating about source code and surrounding software engineering processes like compilation, execution, testing, and debugging.

Three Challenges

1. Inferring what developers are trying to do.
2. Finding information needed for the task.
3. Learning proxies for program execution.

Three Challenges

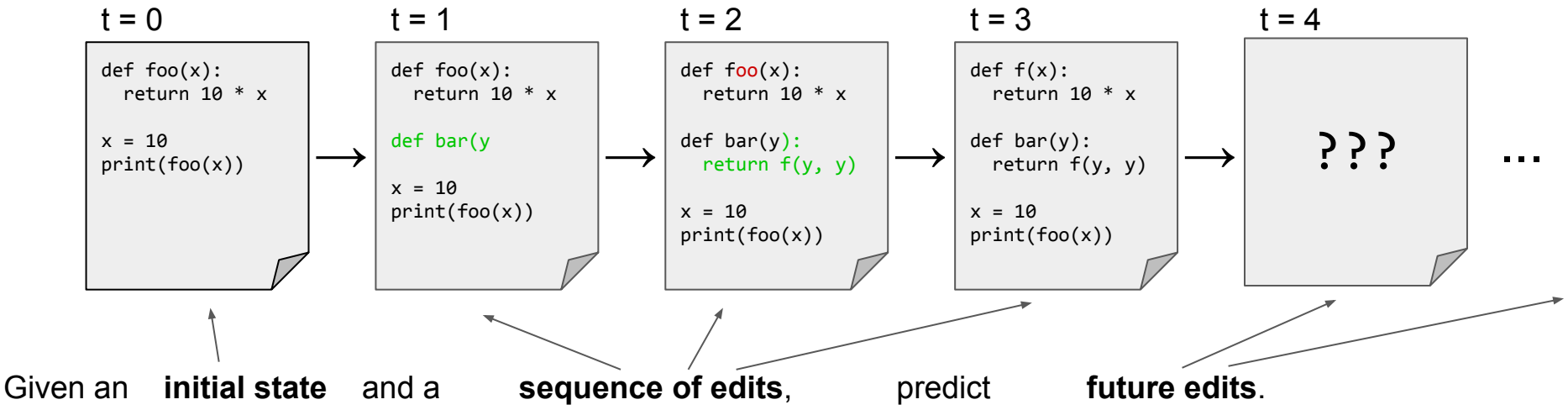
1. Inferring what developers are trying to do.
2. Finding information needed for the task.
3. Learning proxies for program execution.

Challenge – Inferring Intent

Many problems can be boiled down to “Generate code changes given intent”.

How do we extract intent from the activities of software developers?

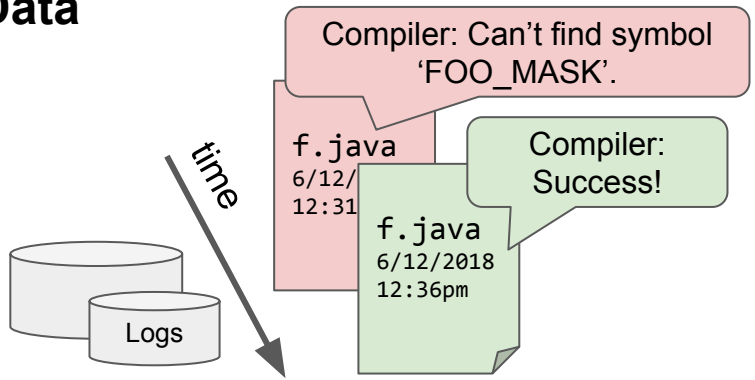
Idea 1: Temporal dynamics



"Neural Networks for Modeling Source Code Edits." Zhao, Bieber, Swersky, Tarlow (2019).

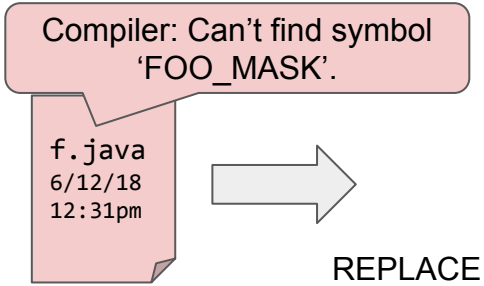
Idea 2: Let tools like the compiler help us

Data



Gather compiler errors & fixes from developers doing their work.

Learning Task



REPLACE ‘private’ on L124 with ‘public’

Learn to map from broken code to developer’s fix.

"Learning to Fix Build Errors with Graph2Diff Neural Networks."
Tarlow, Moitra, Rice, Chen, Manzagol, Sutton, Aftandilian (2019).

Three Challenges

1. Inferring what developers are trying to do.
2. Finding information needed for the task.
3. Learning proxies for program execution.

Repository-Level Prompt Generation for Large Language Models of Code

Disha Shrivastava, Hugo Larochelle, Daniel Tarlow

arXiv 2022.

Google Research

Large Language Models for Code

Recent emergence of large language models for code (Codex, PaLM-Coder, etc).

Prompt

```
// Translate from C to Python
int add_one ( int x ){
 int m = 1;
 while ( x & m ) {
   x = x ^ m;
   m <<= 1;
 }
 x = x ^ m;
 return x; }
```

Model Response

```
def add_one(x: int):
  m = 1
  while (x & m):
    x = (x ^ m)
    m <<= 1
  x = (x ^ m)
  return x
```

https://ai.googleblog.com/2022/04/pathways-language-model-palm-scaling-to.html

Large Language Models for Code

Application to code completion

Context before cursor →

```
from cuda import use_cuda, LongTensor, FloatTensor
from env.env import ProgramEnv
from env.operator import Operator, operator_to_index
from env.statement import Statement, statement_to_index
from dsl.program import Program
from dsl.example import Example
from dsl.function import Function, OutputOutOfRangeError, NullInputError


def ints_to_tensor(ints, pad_index=0.0):
        """Converts a nested list of integers to a padded tensor."""
        if isinstance(ints, torch.Tensor):
                return ints
        if isinstance(ints, list):
                if isinstance(ints[0], int):
                        return |
```

Cursor position

Context after cursor →

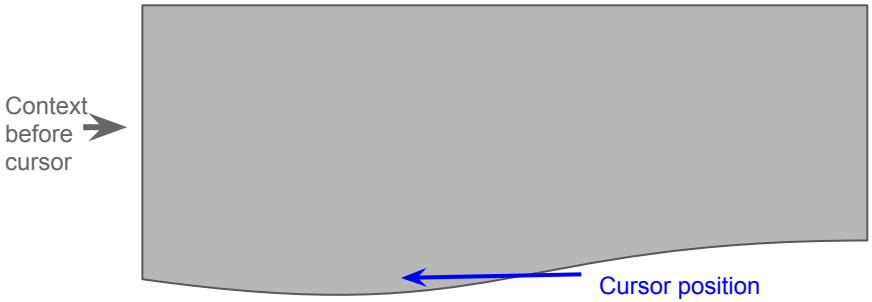
```
def pad_tensors(tensors, pad_index):
        """Takes a list of `N` M-dimensional tensors (M<4) and returns a padded tensor."""

...
```

Target line completion: `torch.LongTensor(ints)`

Large Language Models for Code

How models are **trained**: given a prefix of code, predict the next tokens

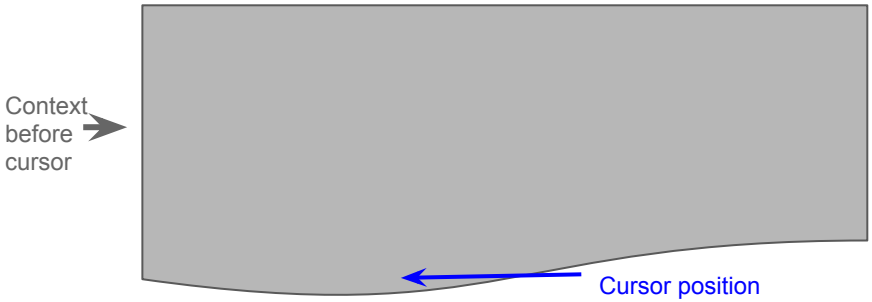


Target line completion: `torch.LongTensor(ints)`

Input **Output**

Large Language Models for Code

How models are **used**: traditional wisdom is to provide the same kind of context at training and testing times.

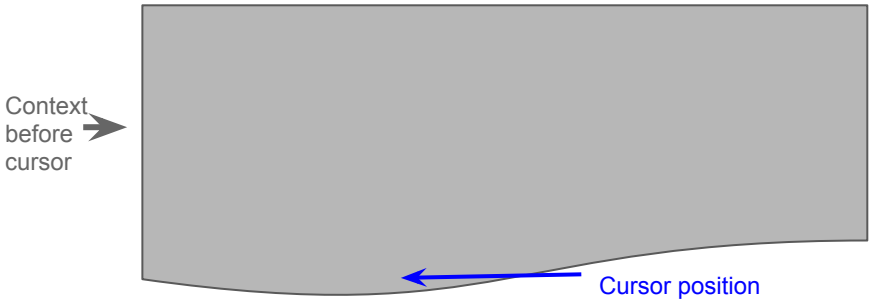


Target line completion: `torch.LongTensor(ints)`

Input **Output**

Large Language Models for Code

How models are **used**: traditional wisdom is to provide the same kind of context at training and testing times. Large language models don't seem to require this.

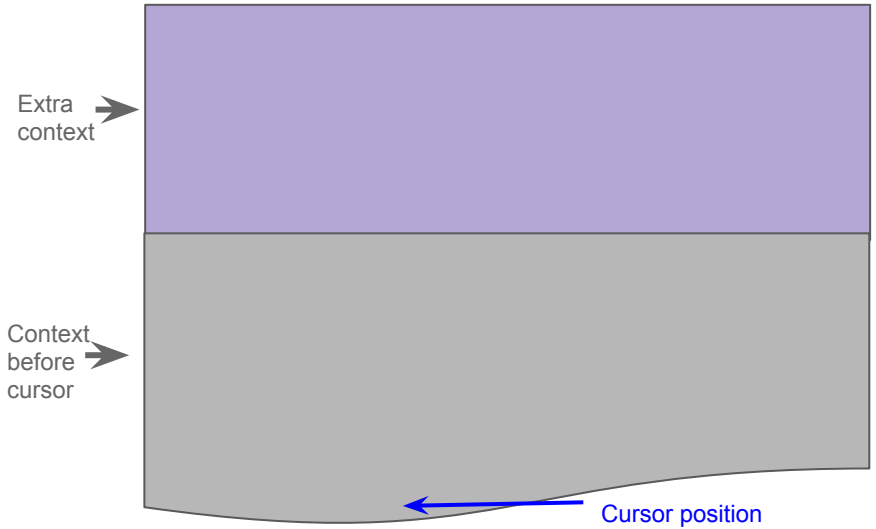


Target line completion: `torch.LongTensor(ints)`

Input **Output**

Large Language Models for Code

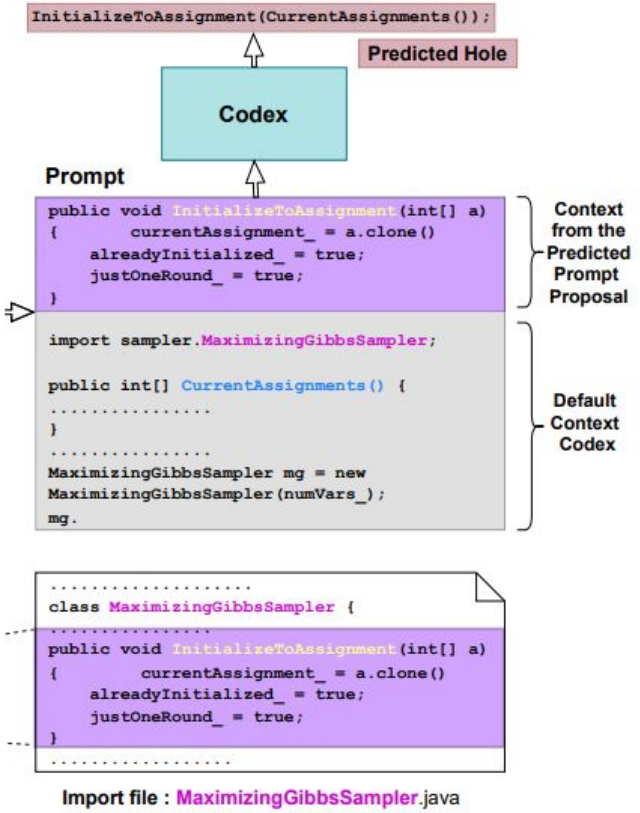
Idea: Add **extra context** to the input using domain knowledge, only at testing time!



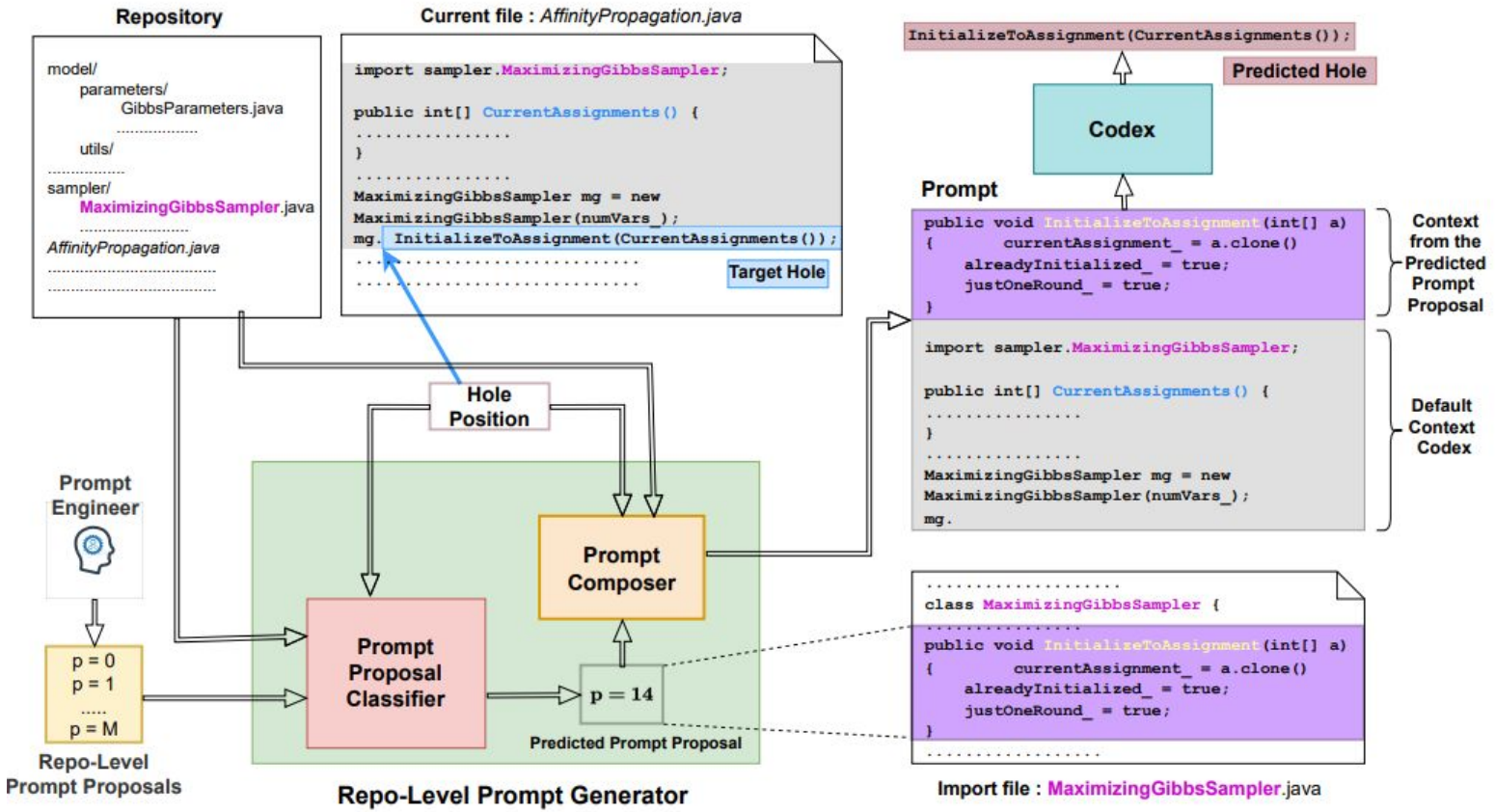
Target line completion: `torch.LongTensor(ints)`

Input **Output**

Adding Repository-level Context via Prompt Proposals



Adding Repository-level Context via Prompt Proposals



Adding Repository-level Context via Prompt Proposals

Table 1: Success Rate (SR) of different methods on the test data when averaged across all holes (hole-wise) and across individual repositories (repo-wise)

Method	Success Rate(%) (hole-wise)	Rel. ↑(%) (hole-wise)	Success Rate(%) (repo-wise)	Rel. ↑(%) (repo-wise)
Codex [8]	58.73	-	60.64	-
Oracle	79.63	35.58	80.24	32.31
Random	58.13	-1.02	58.95	-2.79
Random NN	58.98	0.43	60.04	-0.99
Identifier Usage (Random)	64.93	10.55	67.83	11.85
Identifier Usage (NN)	64.91	10.52	67.94	12.03
Fixed Prompt Proposal	65.78	12.00	68.01	12.15
RLPG-H	**68.51**	**16.65**	69.26	14.21
RLPG-R	67.80	15.44	**69.28**	**14.26**

Three Challenges

1. Inferring what developers are trying to do.
2. Finding information needed for the task.
3. Learning proxies for program execution.

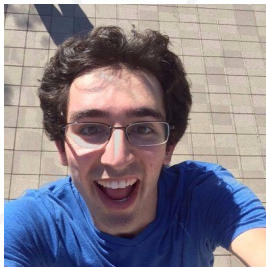
Static Prediction of Runtime Errors by Learning to Execute Programs with External Resource Descriptions

David Bieber, Rishab Goel, Daniel Zheng,

Hugo Larochelle, Daniel Tarlow.

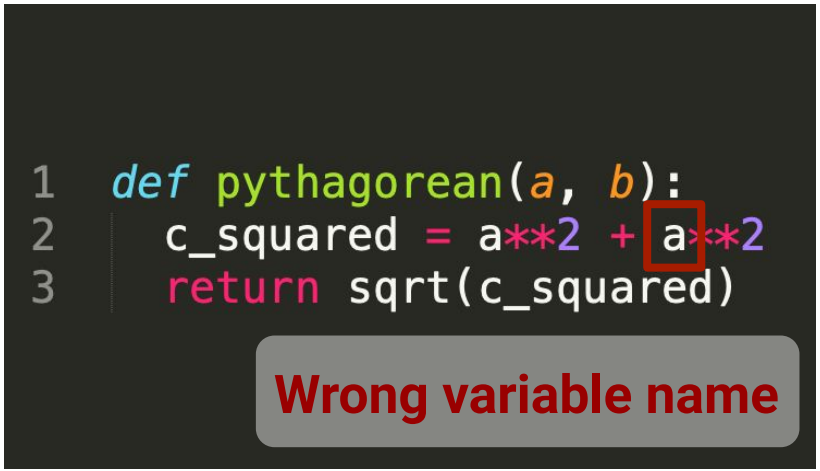
arXiv 2022 (and IPA-GNN paper @ NeurIPS 2020).

Google Research

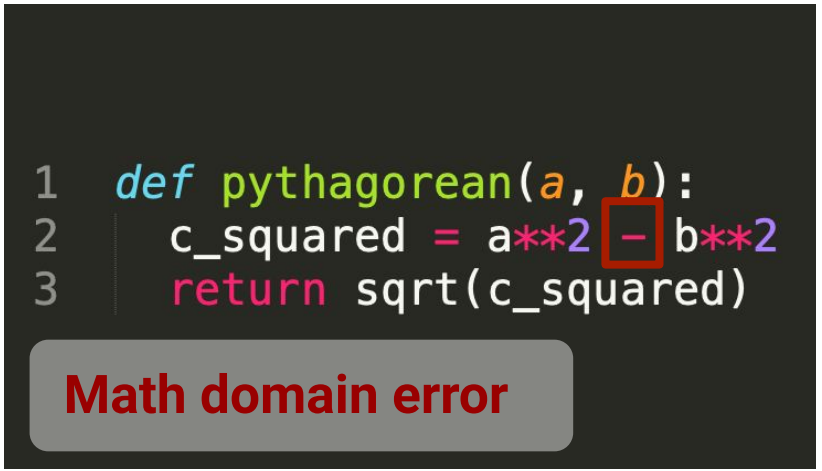


Reasoning about execution behavior

"Simple stupid bug"



Runtime error prediction



Challenge: Program Execution Behavior

Program execution is complicated! Many steps, complex intermediate values.

Unclear that current ML models are well-suited. E.g., Austin et al (2021) find that large language models are not good at predicting execution even when they can program by example.

Austin et al. (2021). "Program Synthesis with Large Language Models."

New Runtime Error Prediction Task

- Filters and extends Project CodeNet (Puri et al., 2021)
- 2,441,130 Python 3 competitive programming submissions
- 3,119 distinct problems
- 26 runtime error labels incl. "No error"

Input:

```
N = int(input())
A = list(map(int, input().split()))
res = 0
for i in range(1, len(A)+1, 2):
  res += A[i] % 2
print(res)
```

Target: IndexError (Line 5)

Target Class	Train #	Valid #	Test #
No error	1881303	207162	205343 / 13289†
AssertionError	47	4	8
AttributeError	10026	509	1674
EOFError	7676	727	797
FileNotFoundError	259	37	22
ImportError	7645	285	841
IndentationError	10	0	12
IndexError	7505	965	733
KeyError	362	39	22
MemoryError	8	7	1
ModuleNotFoundError	1876	186	110
NameError	21540	2427	2422
numpy.AxisError	20	2	3
OSError	19	2	2
OverflowError	62	6	11
re.error	5	0	0
RecursionError	2	0	1
RuntimeError	24	5	3
StopIteration	3	0	1
SyntaxError	74	4	3
TypeError	21414	2641	2603
UnboundLocalError	8585	991	833
ValueError	25087	3087	2828
ZeroDivisionError	437	47	125
Timeout	7816	1072	691
Other	18	8	2

Control Flow Graphs

```
v0 = 23
v1 = 6
while v1 > 0:
  v1 -= 1
  if v0 % 10 <= 3:
    v0 += 4
    v0 *= 6
  v0 -= 1
```

Program

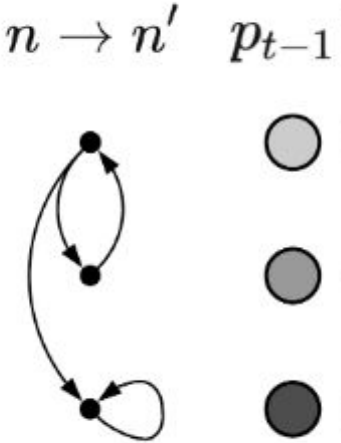
Control Flow Graph

Node: program statement.

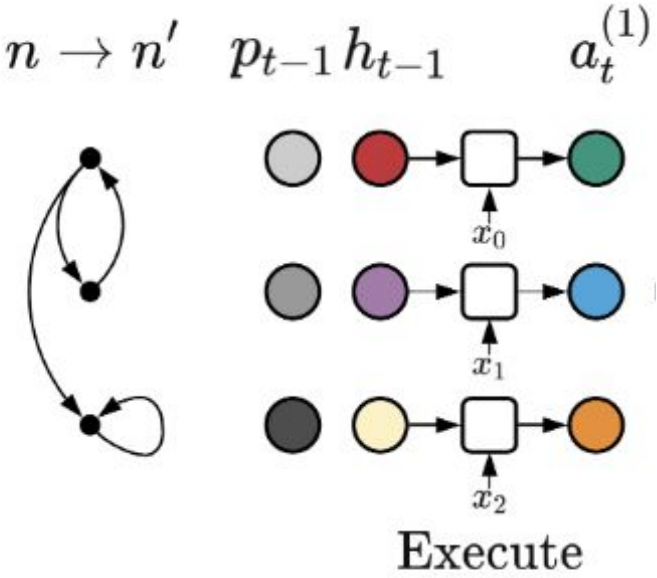
Edge: link to possible next statement.

Branch: when there is more than one outgoing edge.

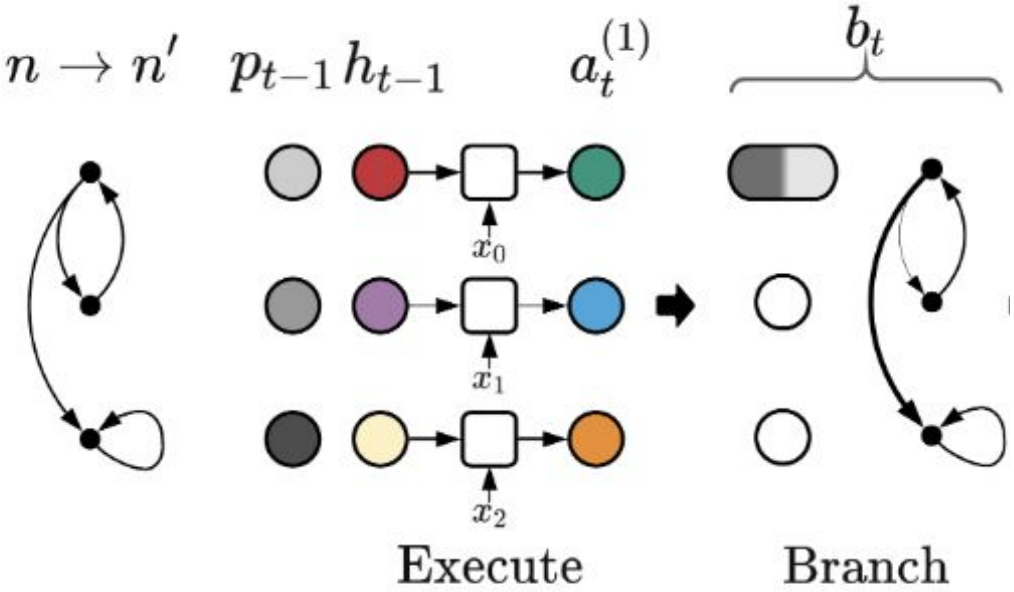
Instruction Pointer Attention Graph Neural Networks



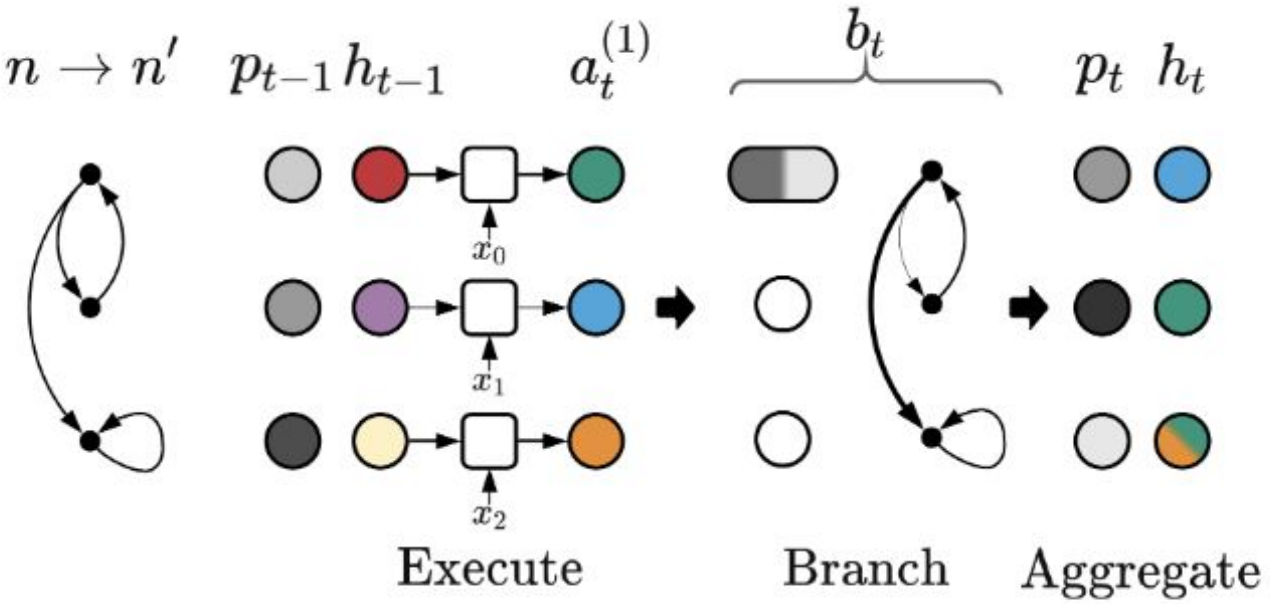
Instruction Pointer Attention Graph Neural Networks



Instruction Pointer Attention Graph Neural Networks



Instruction Pointer Attention Graph Neural Networks



Bieber et al (2020). "Learning to Execute Programs with Instruction Pointer Attention Graph Neural Networks." NeurIPS.

Exception Handling

Exception IPA-GNN layer computes soft instruction pointer p_t and per-node hidden state h_t by:

1. Executing each line of code with an RNN to produce hidden state proposals $q_t^{(1)}$.
2. Making a soft raise decision $b_{t,r}$ at each branch point in the program.
3. Making a soft branch decision b_t at each branch point in the program.
4. **Aggregating** proposed hidden states with instruction pointer attention (IPA).

Soft instruction pointer p_t is a per-timestep distribution over all control flow nodes.
Hidden state $h_{t,n}$ represents values of variables at timestep *t* if current line at *t* is *n*.

Execution with Resource Descriptions

- Ambiguity when programs depend on external resources
  - Examples: command line arguments, file contents, network access
- Lacking an external resource, a *description of it* is the next best thing
- In competitive programming: external resource is *input stream*

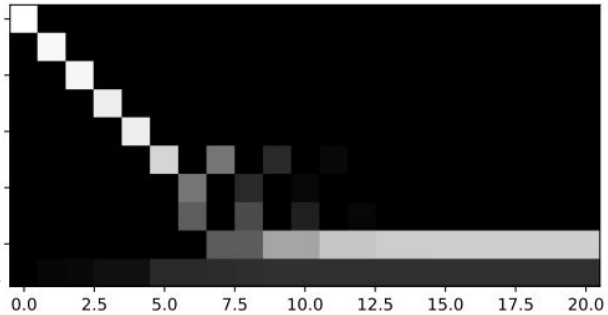
Example resource description:

```
Input:  Input is given from Standard Input in the
following format Constraints:  Each character of S
is A or B. |S| = 3
```

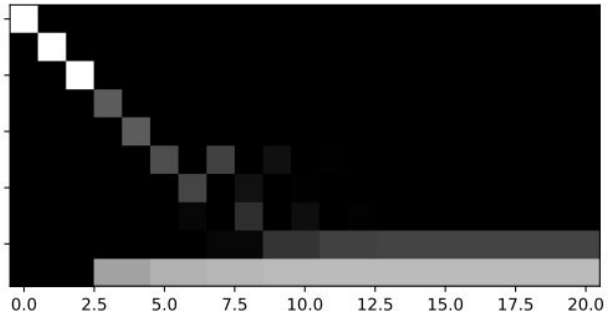
Localization Example

```
N = int(input())
A = list(map(int, input().split()))
res = 0
for i in range(1, len(A)+1, 2):
  res += A[i] % 2
print(res)
```

BASELINE Error contrib.	R.D. Error contrib.
2.9	0.2
0.8	0.0
3.0	**63.3**
9.8	6.3
0.3	0.1
0.2	2.2



BASELINE



RESOURCE DESCRIPTION

"exit" state

"exception" state

	Model	R.D.?	Acc.	W. F1	E. F1
Baselines	GGNN		62.8	58.9	45.8
	Transformer		63.6	60.4	48.1
	LSTM		66.1	61.4	48.4
Ablations	GGNN	✔	68.3	66.5	56.8
	Transformer	✔	67.3	65.1	54.7
	LSTM	✔	68.1	66.8	58.3
	IPA-GNN		68.3	64.8	53.8
	E. IPA-GNN		68.7	64.9	53.3
Ours	IPA-GNN	✔	71.4	70.1	62.2
	E. IPA-GNN	✔	**71.6**	**70.9**	**63.5**

Discussion

Recap: Discussed three challenges and some directions

1. Inferring what developers are trying to do.
2. Finding information needed for the task.
3. Learning proxies for program execution.

Discussion: We're seeing ML be useful in assisting software developers, but these are not fully solved problems. Interesting tension.