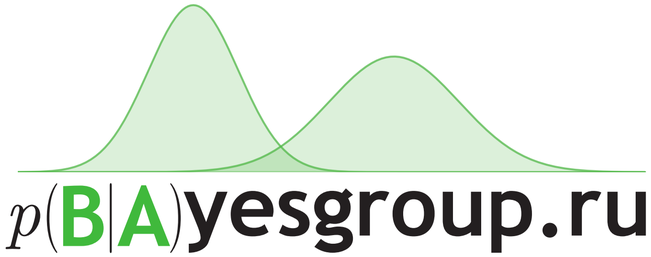
Empirical Study of Transformers for Source Code

Nadezhda Chirkova, Sergey Troshin

HSE University
Moscow, Russia
accepted to ESEC/FSE'21



Deep Learning for Source Code

Real world applications of DL4Code:

Code generation

GitHub Copilot

codota
now Tabnine

tabnine

Bug detection and repair

Code Defect AI Microsoft

Clever-Commit

SapFix FACEBOOK

Automated documentation

Codist AI
AI code documentation assistant

Docly
Focus on coding.
Let us handle the documentation.

Deep Learning for Source Code

```python
for incldir in fnames[lib_folder]:
   print(incldir)
```

```javascript
for (let i = 0; i < buttons.length ; i++) {
  buttons[i].addEventListener('click', createParagraph);
}
```

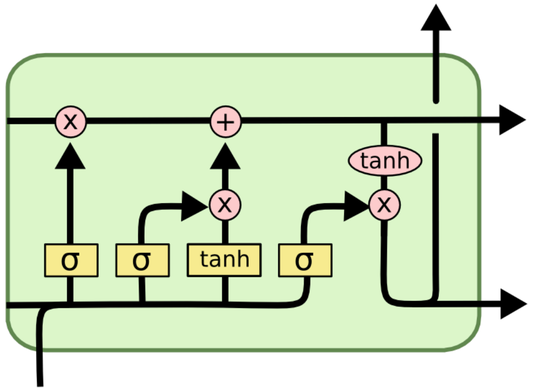
Source code is a special type of structured natural language written by programmers

Le et al. 2020

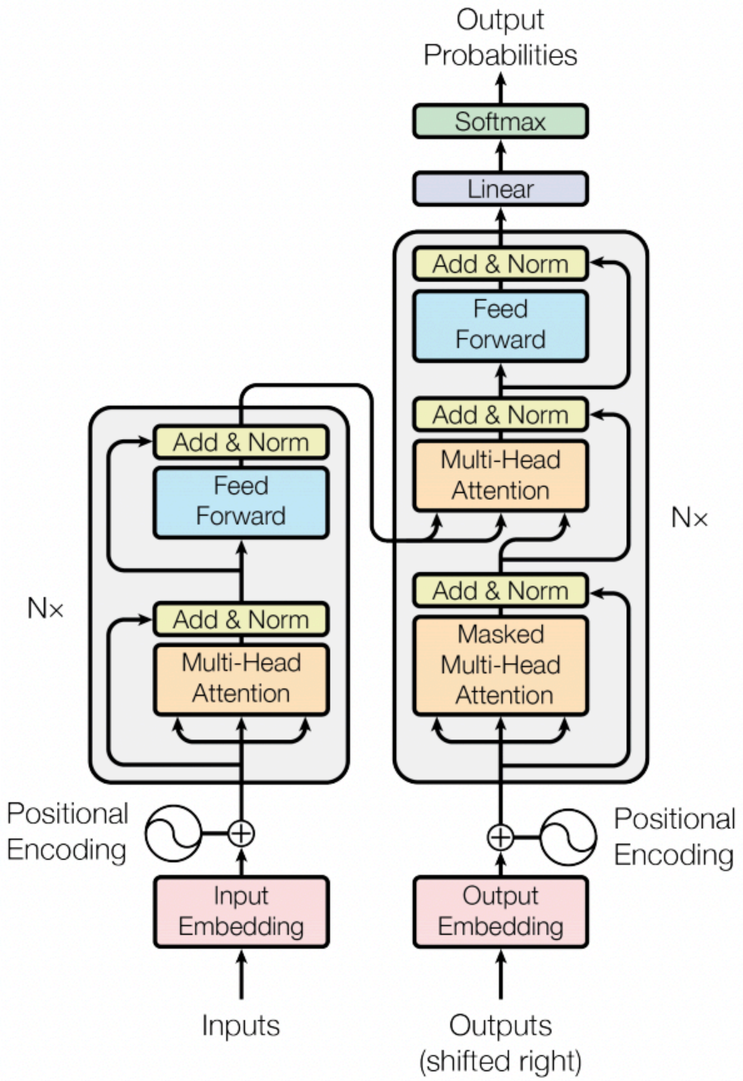
Can we apply NLP architectures for source code?

How can we incorporate more domain knowledge into preprocessing/architectures?

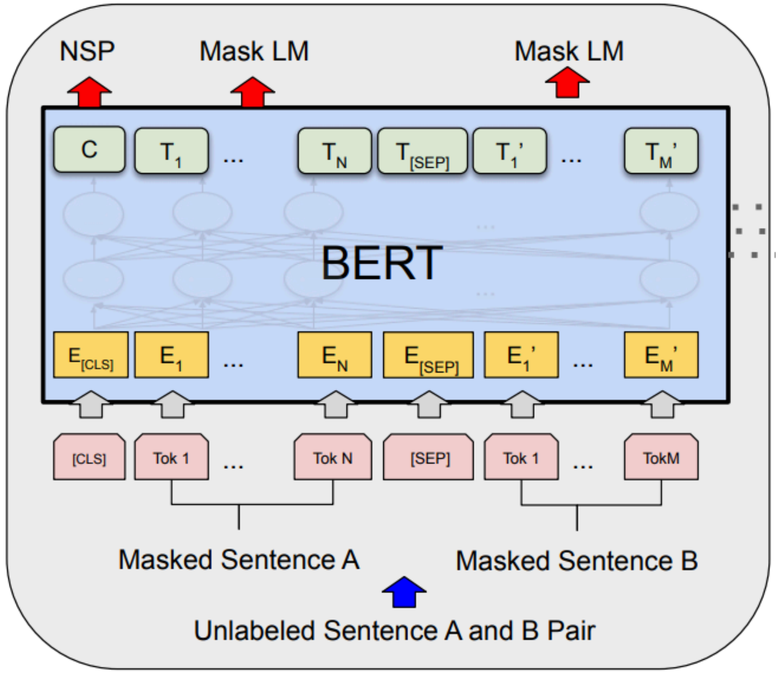
Popular NLP architecture families



LSTM [1]



Transformer [2]

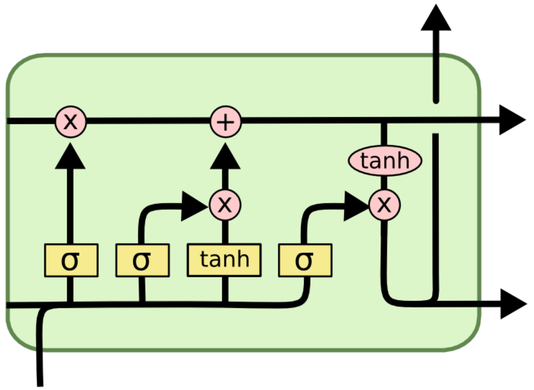


Pre-training

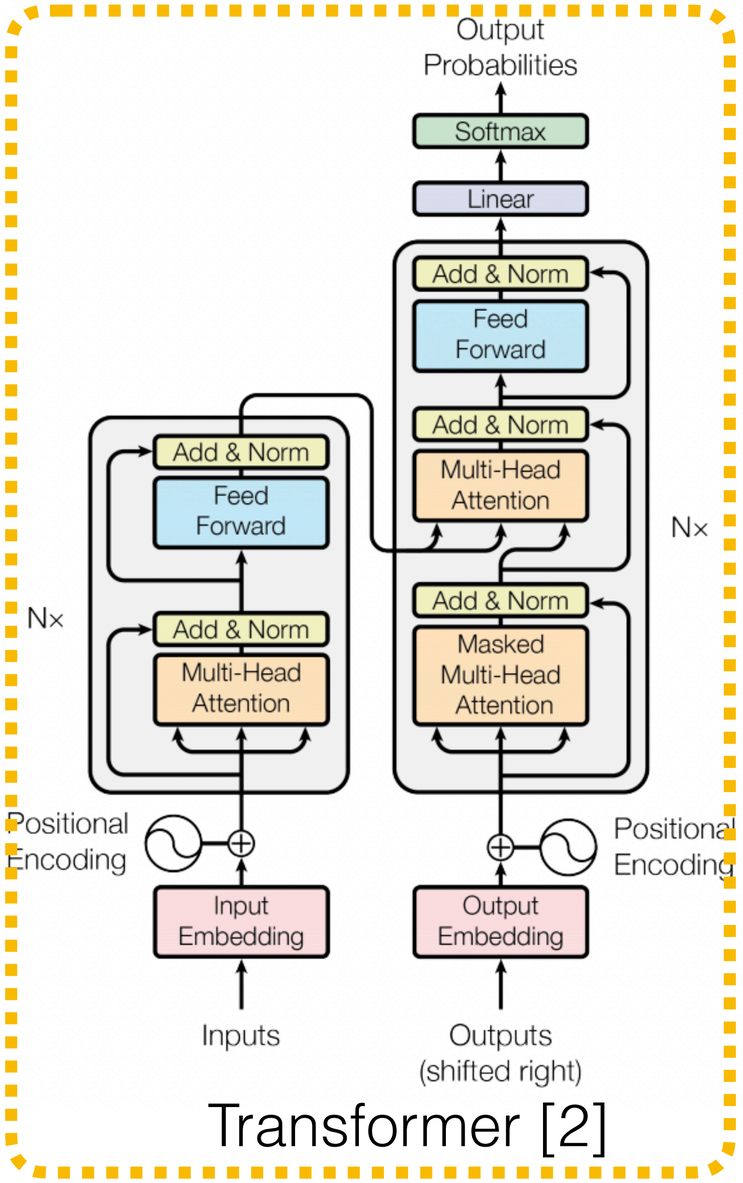
BERT [3]

[1] - https://direct.mit.edu/neco/article/9/8/1735/6109/Long-Short-Term-Memory
[2] - https://arxiv.org/pdf/1706.03762.pdf
[3] - https://arxiv.org/abs/1810.04805

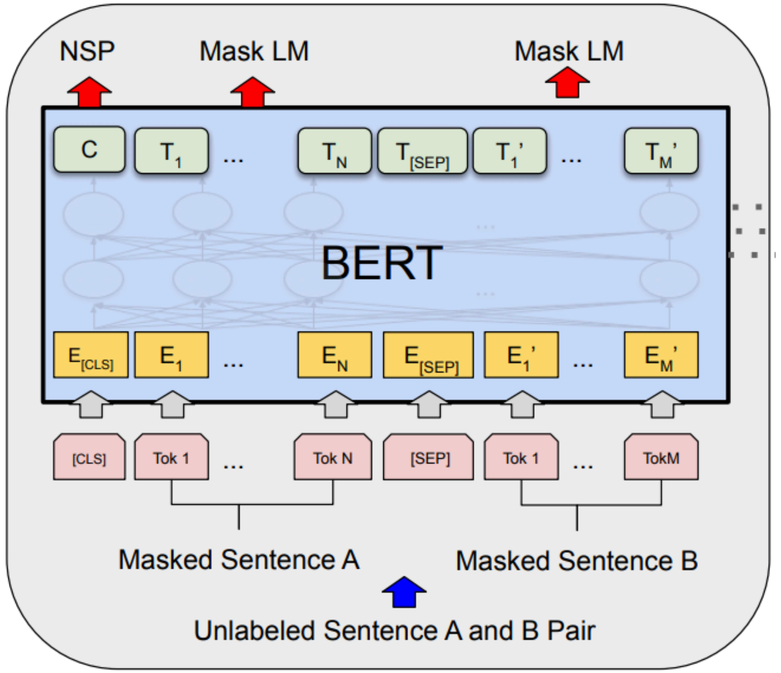
Popular NLP architecture families



LSTM [1]



Transformer [2]



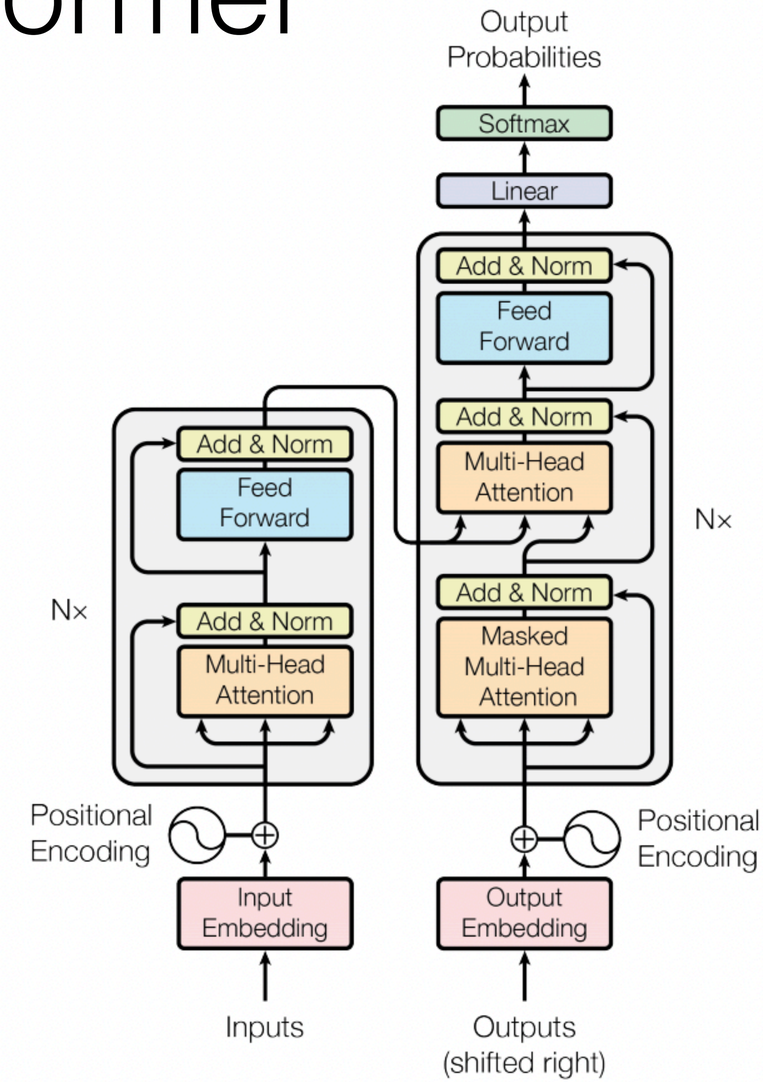
Pre-training

BERT [3]

[1] - https://direct.mit.edu/neco/article/9/8/1735/6109/Long-Short-Term-Memory
[2] - https://arxiv.org/pdf/1706.03762.pdf
[3] - https://arxiv.org/abs/1810.04805

How to pass code to Transformer

```
for incldir in fnames[lib_folder]:
   print(incldir)
```



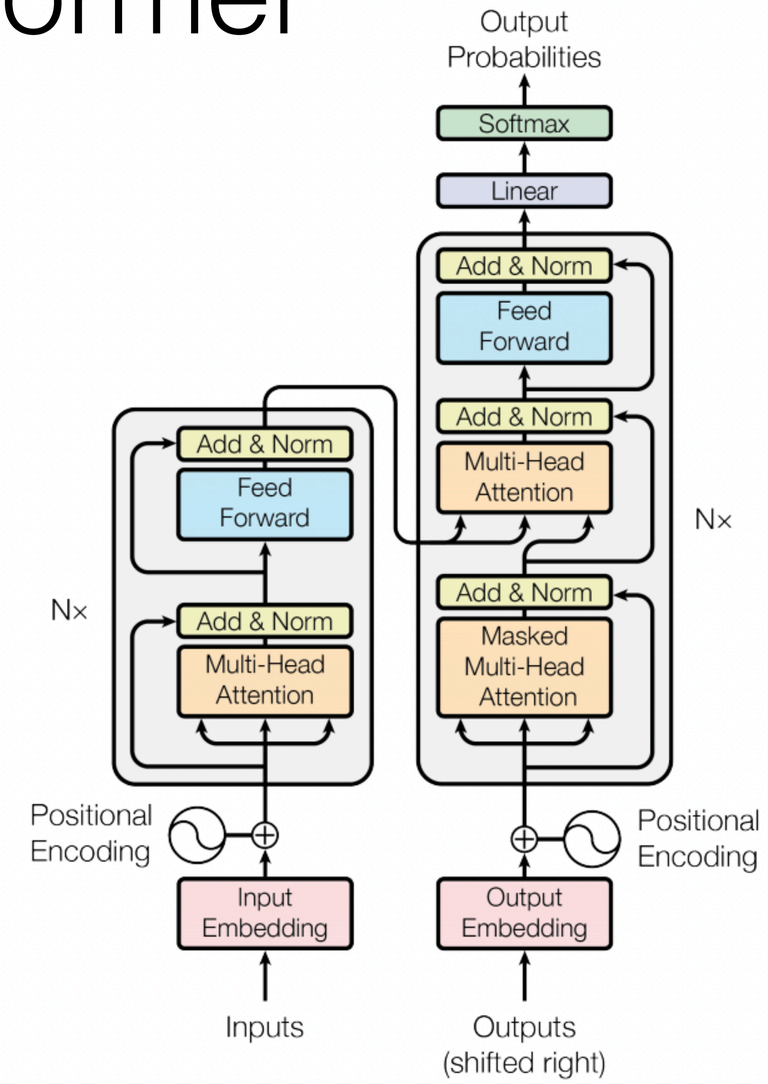
Transformer

https://arxiv.org/pdf/1706.03762.pdf

How to pass code to Transformer

```
for incldir in fnames[lib_folder]:
   print(incldir)
```

```
_for _incl dir _in _fnames _[ _lib _
folder _] _: _NL _INDENT _print
_( _incl dir _)
```



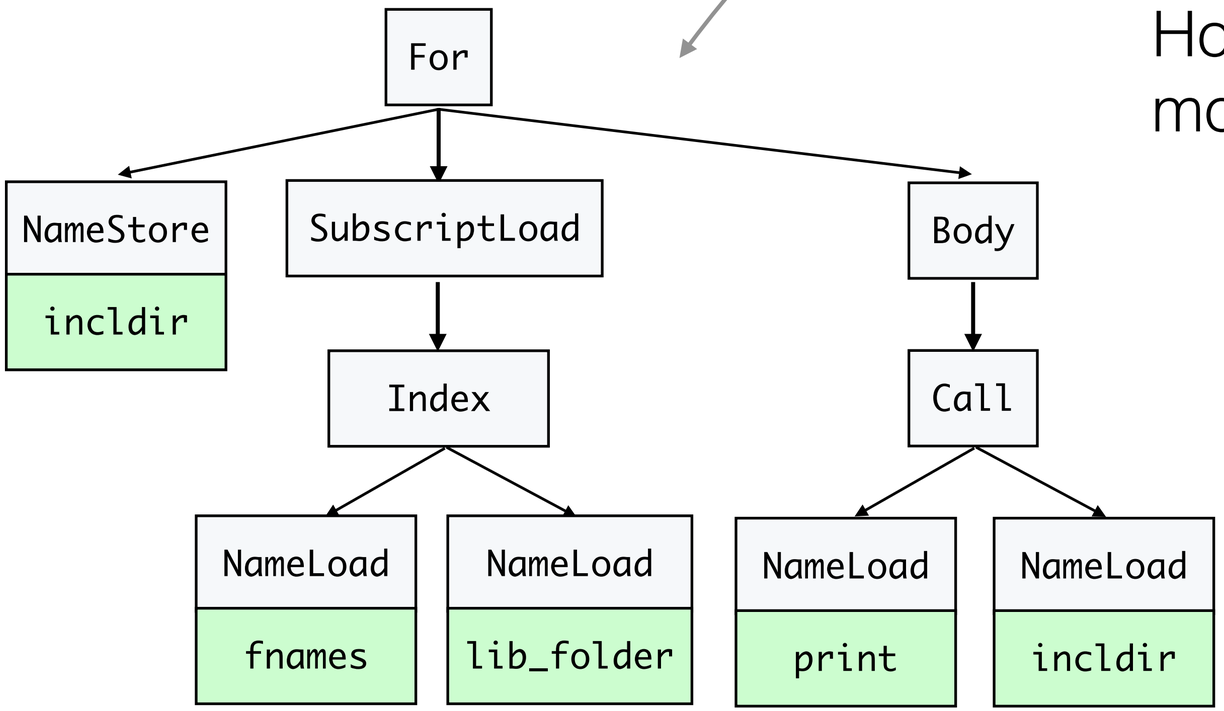
Transformer

https://arxiv.org/pdf/1706.03762.pdf

Passing AST to Transformer

```
for incldir in fnames[lib_folder]:
    print(incldir)
```

Structured Text



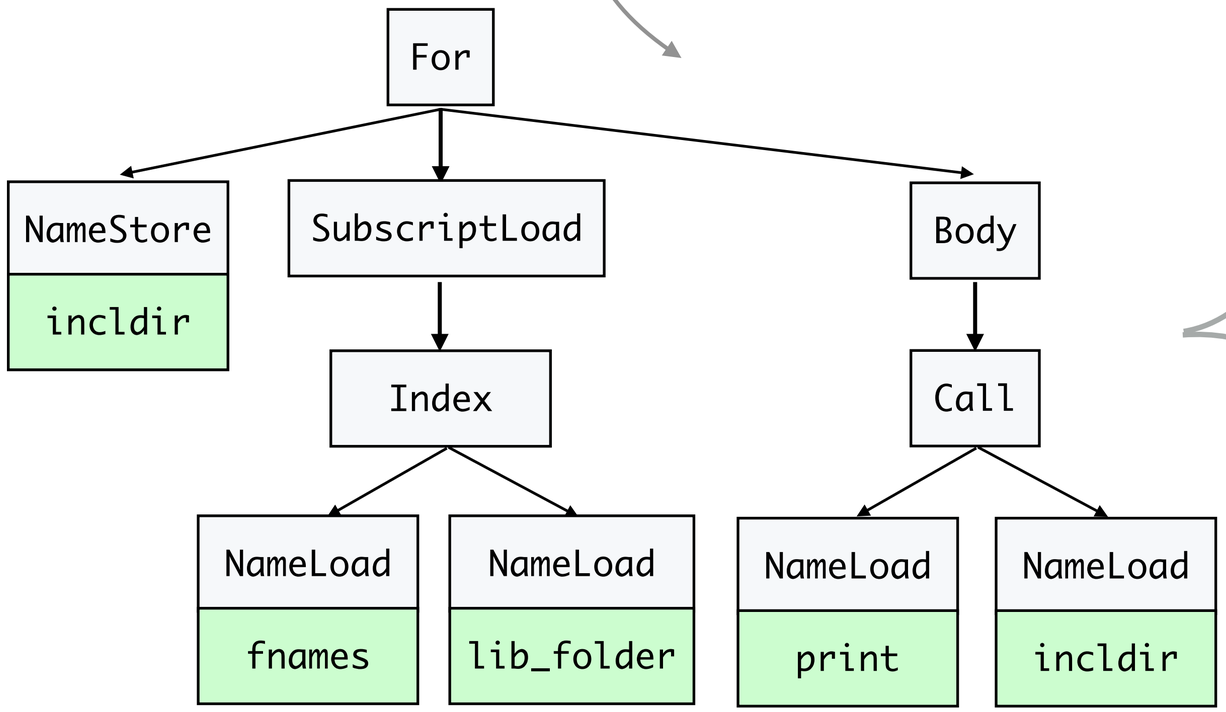
Abstract Syntax Tree

How to pass the AST to the Transformer model?

Transformer

Passing AST to Transformer

```
for incldir in fnames[lib_folder]:
    print(incldir)
```



Abstract Syntax Tree

Standard NLP baselines

Sequential positional encodings [1]

Sequential relative attention [2]

Tree modifications

Tree positional encodings [3]

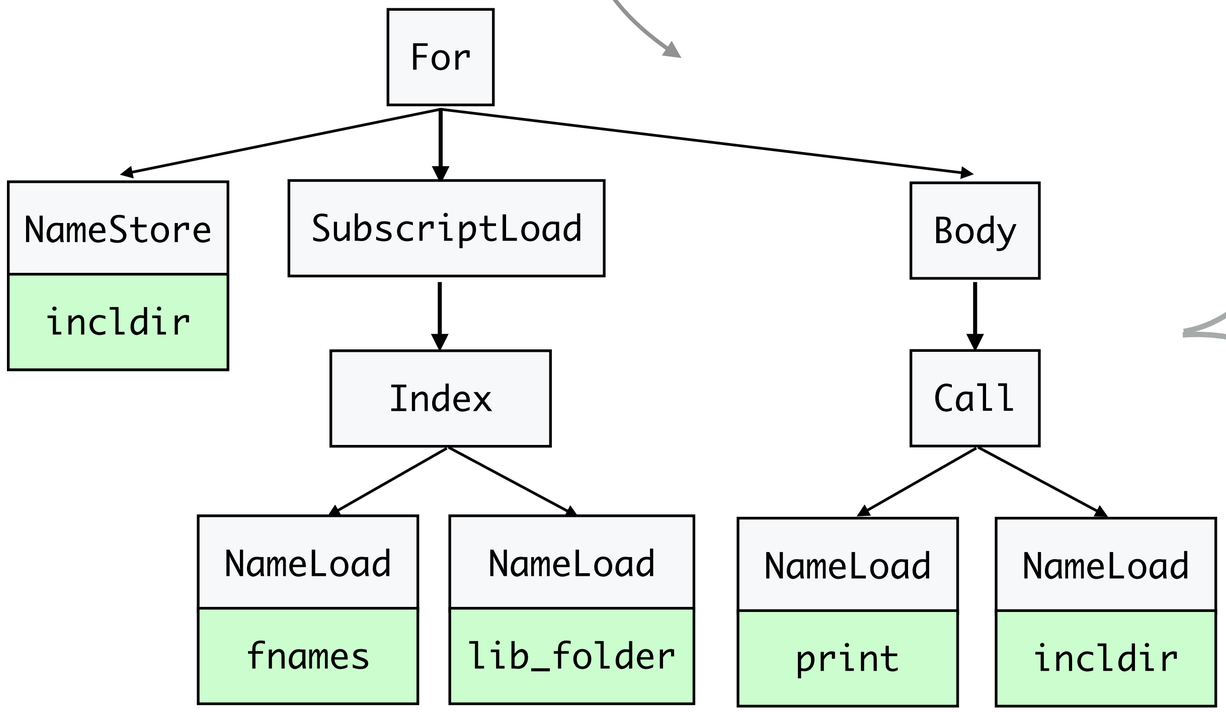
Tree relative attention [4]

GGNN Sandwich [5]

[1] - https://arxiv.org/abs/1706.03762
[2] - https://arxiv.org/abs/1803.02155
[3] - https://papers.nips.cc/paper/ /2019/file/6e0917469214d8fbd8c517dcdc6b8dcf-Paper.pdf
[4] - https://arxiv.org/abs/2003.13848
[5] - https://openreview.net/forum?id=B1lnbRNtwr

Passing AST to Transformer

```
for incldir in fnames[lib_folder]:
    print(incldir)
```



Abstract Syntax Tree

Standard NLP baselines

Sequential positional encodings [1]

Sequential relative attention [2]

Tree modifications

Tree positional encodings [3]

Tree relative attention [4]

GGNN Sandwich [5]

All modifications were tested on different tasks and datasets

Which one performs better?

Overview

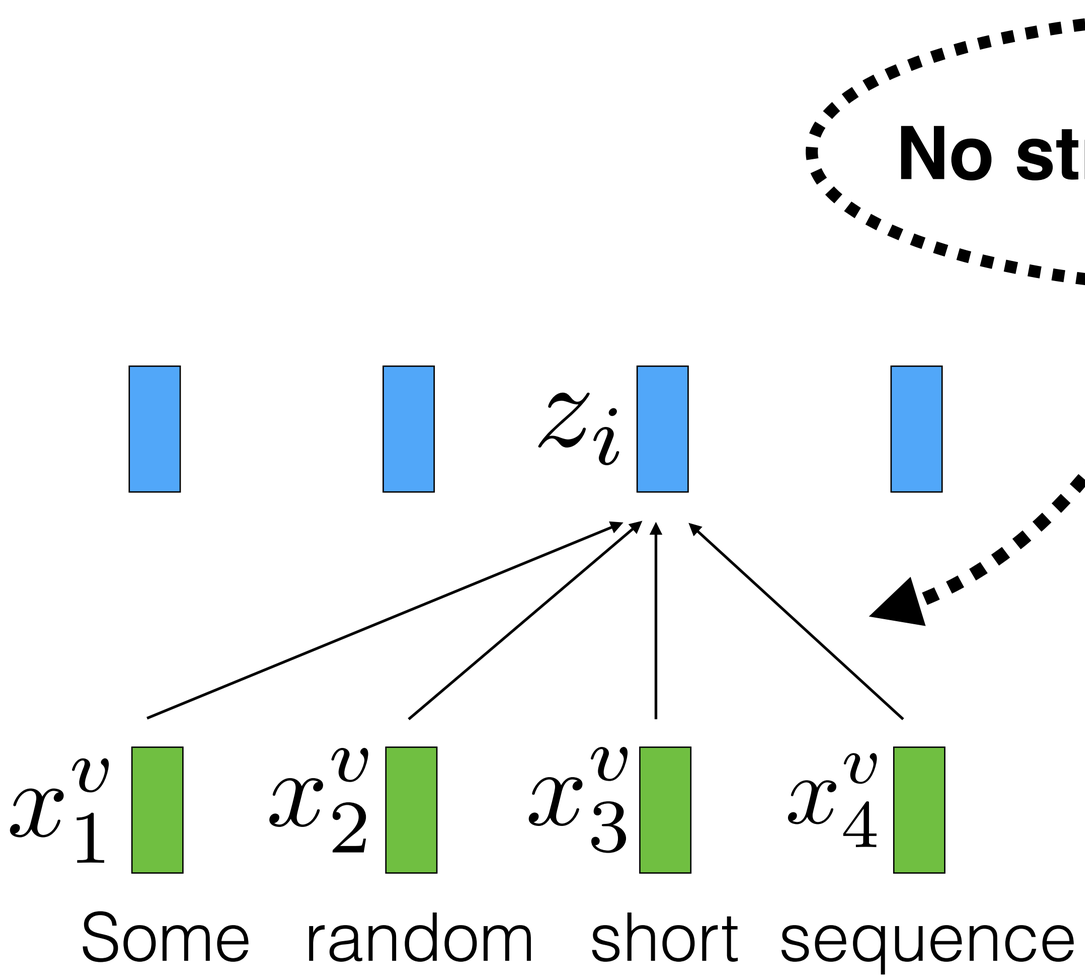
- Goal: understanding whether Transformers are able to utilize syntactic structure and what is the best approach to do it

- 5 syntax-capturing Transformer modifications

  3 tasks: code completion, bug fixing, function naming

  2 datasets: Python150k and Javascript150k

Self-attention: “bag” of elements



$$z_i = \sum_j \tilde{\alpha}_{ij} x_j^v$$

$$\tilde{\alpha}_{ij} = \frac{\exp(a_{ij})}{\sum_j \exp(a_{ij})}$$

$$a_{ij} = \frac{x_i^q {x_j^k}^T}{\sqrt{d_z}}$$

Standard NLP baselines

Sequential positional encodings

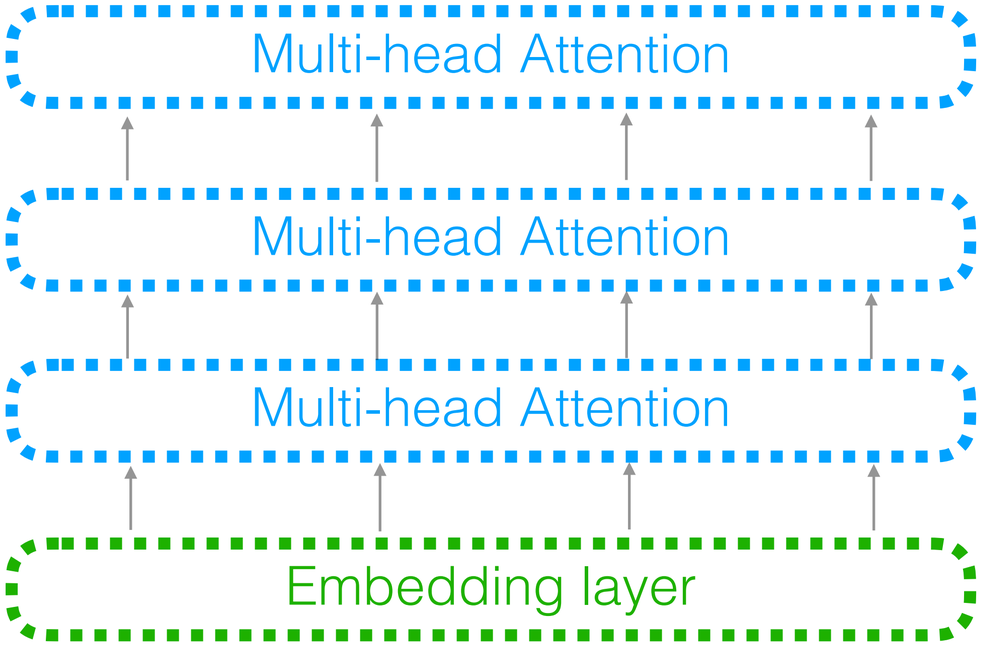
Sequential relative attention

Tree modifications

Tree positional encodings

Tree relative attention

GGNN Sandwich



Transformer encoder

Standard NLP baselines

Sequential positional encodings

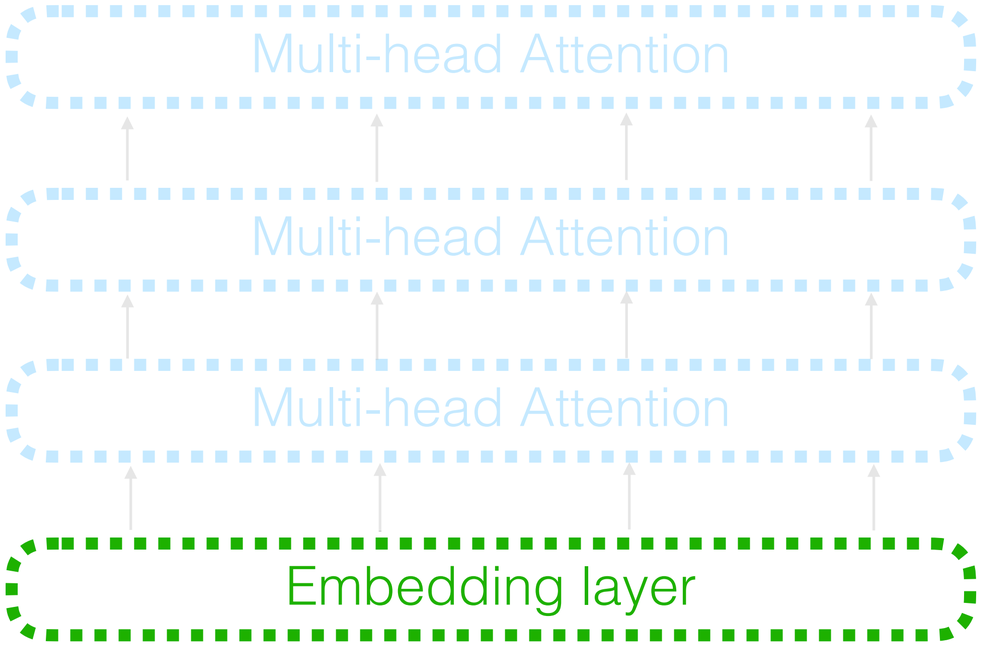
Sequential relative attention

Tree modifications

Tree positional encodings

Tree relative attention

GGNN Sandwich

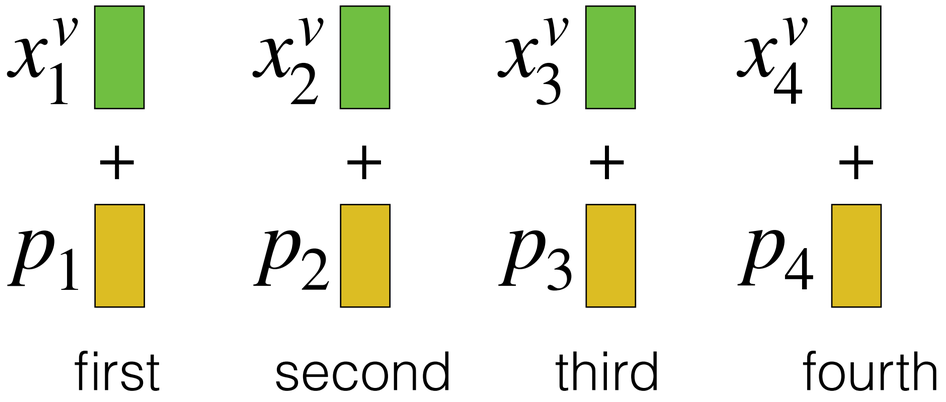


Transformer encoder

Sequential positional encodings / embeddings

Input layer:

$$x_j = x_j^v + p_j$$



(Applied to the AST depth-first traversal)

Positional embeddings:

$p_j = e_j$ — learnable embeddings

Positional encoding:

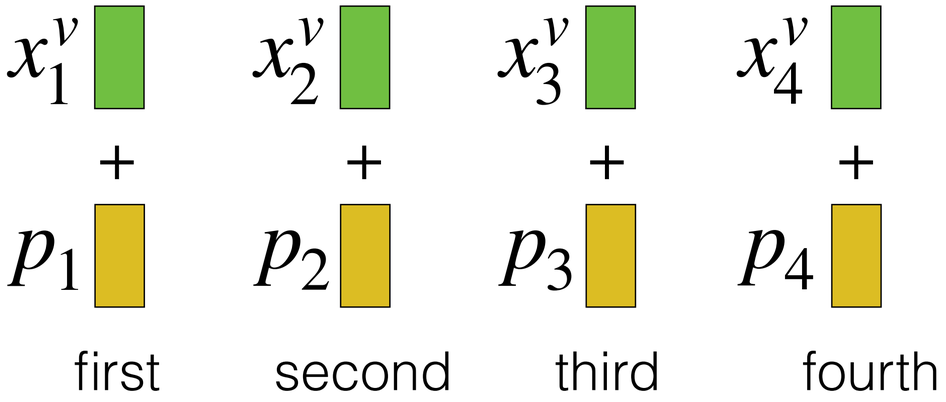
$$p_{j,m} = \begin{cases} sin(\omega_k j) & \text{if } m = 2k \\ cos(\omega_k j) & \text{if } m = 2k+1 \end{cases}$$

$$w_k = \frac{1}{10000^{2k/d_x}}$$

Sequential positional encodings / embeddings

Input layer:

$$x_j = x_j^v + p_j$$



Positional embeddings:

Parameters: learnable embeddings

Hyperparameters: none

Positional encoding:

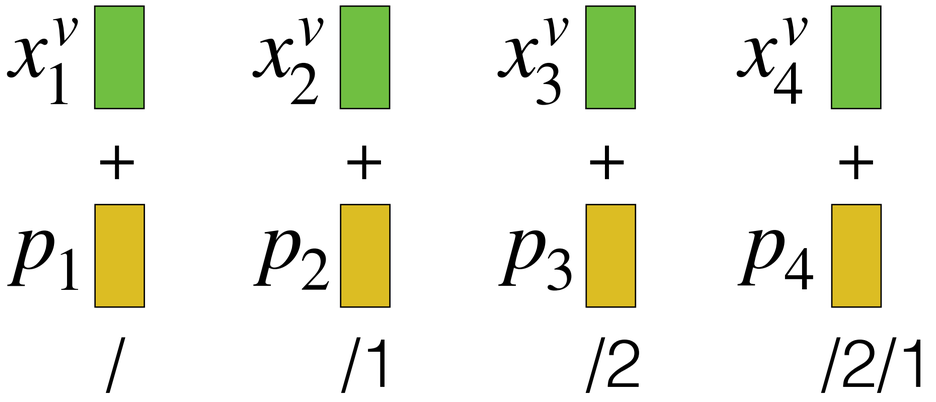
Parameters: none

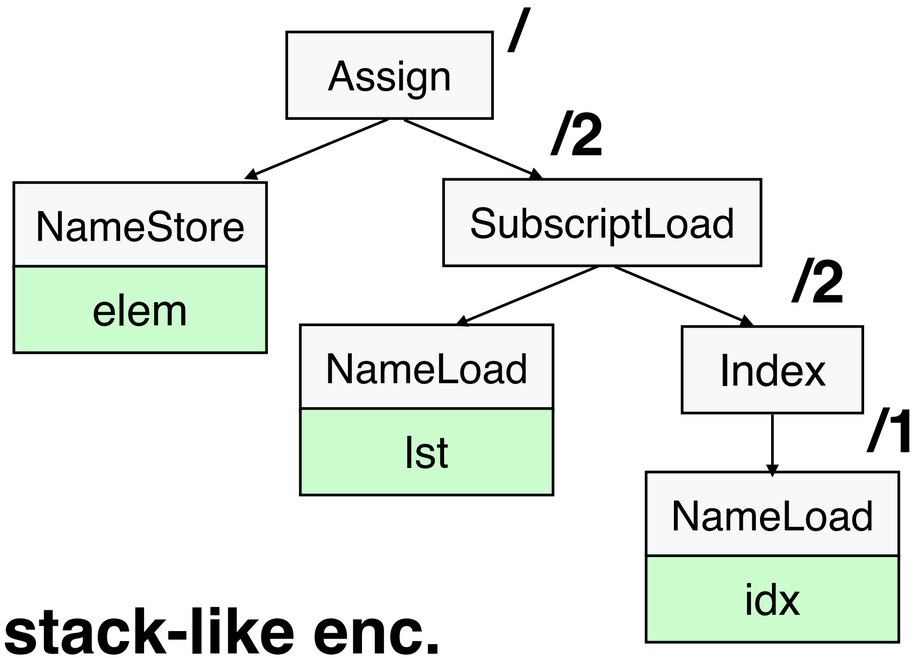
Hyperparameters: none

Tree positional encoding

Input layer:

$$x_j = x_j^v + p_{path-to-j}$$





stack-like enc. →

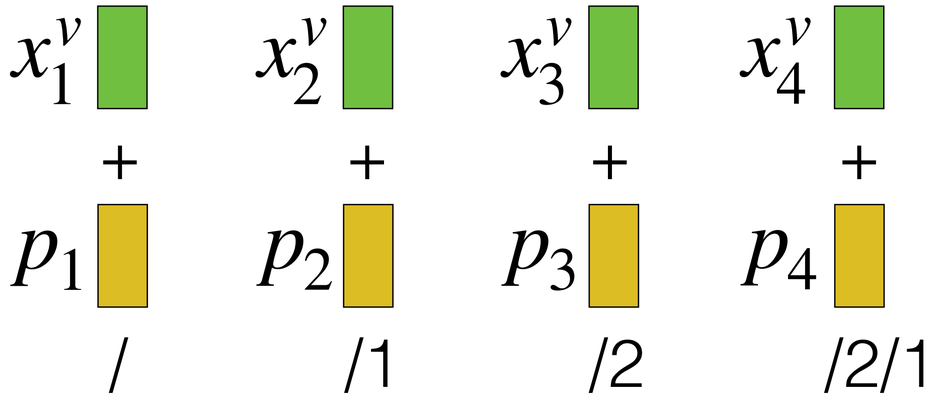
/	**000 000 000**
/1	**100 000 000**
/2	**010 000 000**
/2/1	**100 010 000**
/2/2	**010 010 000**
/2/2/1	**100 010 010**

Shiv, V., and Quirk, C. Novel positional encodings to enable tree-based transformers. NeurIPS19

Tree positional encoding

Input layer:

$$x_j = x_j^v + p_{path-to-j}$$



Parameters: p's

Hyperparameters:
maximum children count
maximum path length

stack-like enc. →

/	**000 000 000** $\odot[111\ ppp\ p^2p^2p^2]$
/1	**100 000 000**
/2	**010 000 000**
/2/1	**100 010 000**
/2/2	**010 010 000**
/2/2/1	**100 010 010** $\odot[111\ ppp\ p^2p^2p^2]$

…

Shiv, V., and Quirk, C. Novel positional encodings to enable tree-based transformers. NeurIPS19

Standard NLP baselines

Sequential positional encodings

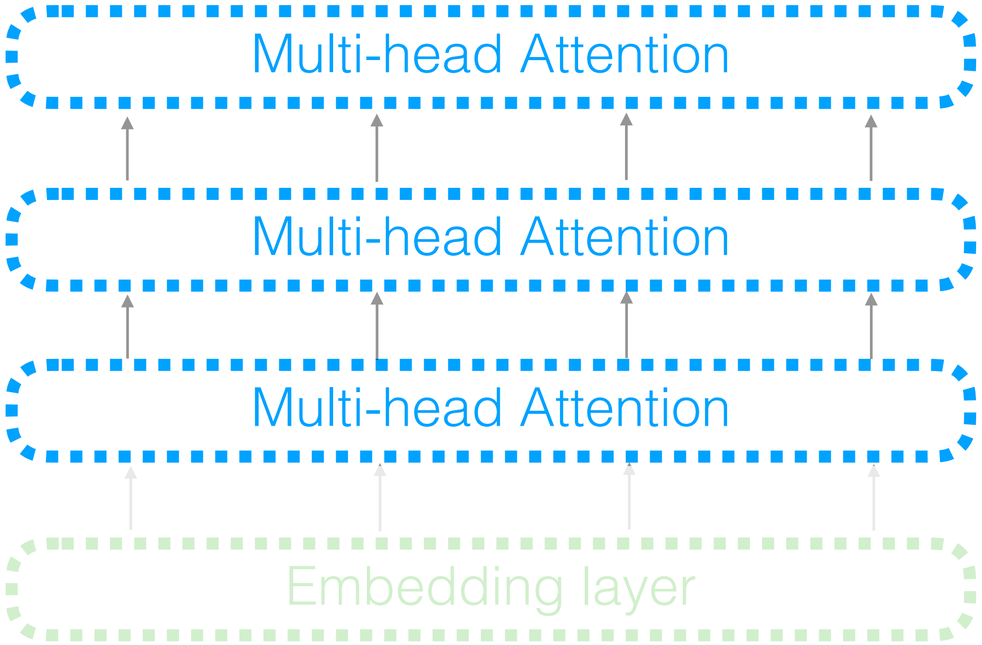
Sequential relative attention

Tree modifications

Tree positional encodings

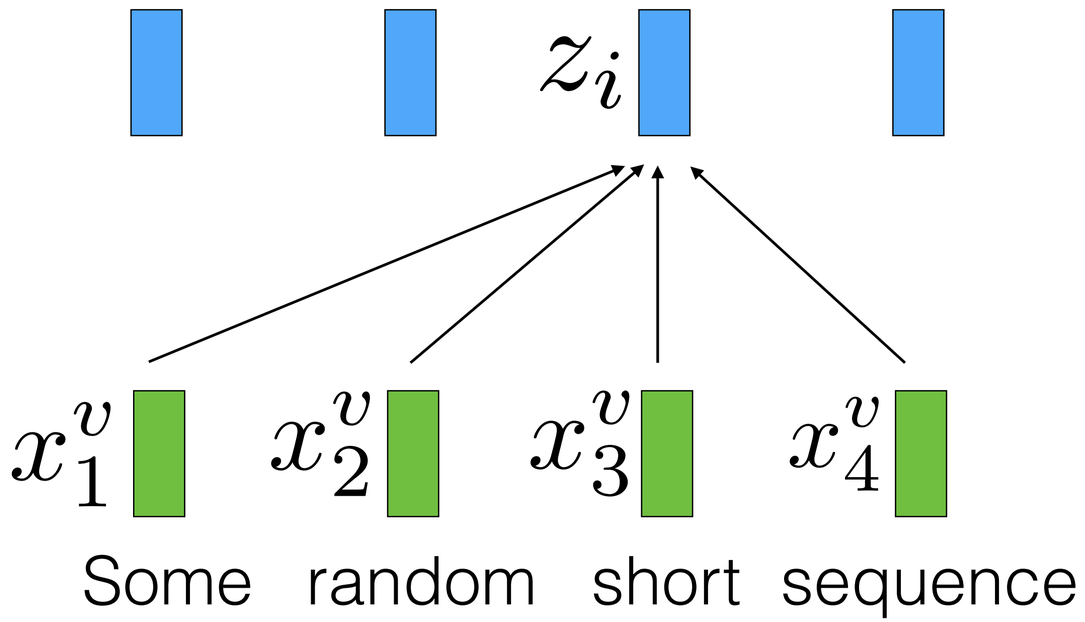
Tree relative attention

GGNN Sandwich



Transformer encoder

Self-attention

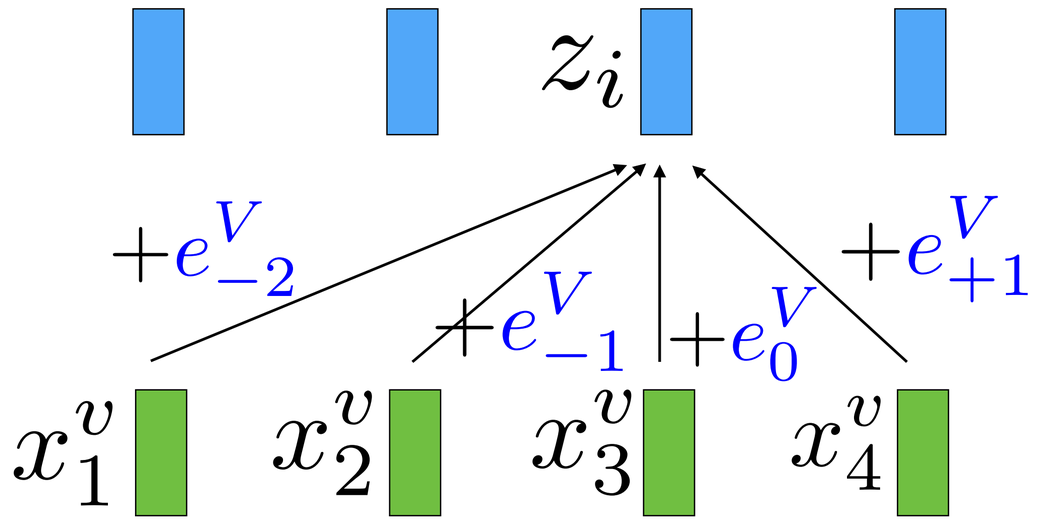


$$z_i = \sum_j \tilde{\alpha}_{ij} x_j^v$$

$$\tilde{\alpha}_{ij} = \frac{\exp(a_{ij})}{\sum_j \exp(a_{ij})}$$

$$a_{ij} = \frac{x_i^q {x_j^k}^T}{\sqrt{d_z}}$$

Sequential relative attention



Learnable embeddings $e^K_{i-j}, e^V_{i-j} \in \mathbb{R}^{d_x}$

embedding of "relation"

$$z_i = \sum_j \tilde{\alpha}_{ij}(x^v_j + e^V_{i-j})$$

$$\tilde{\alpha}_{ij} = \frac{\exp(a_{ij})}{\sum_j \exp(a_{ij})}$$

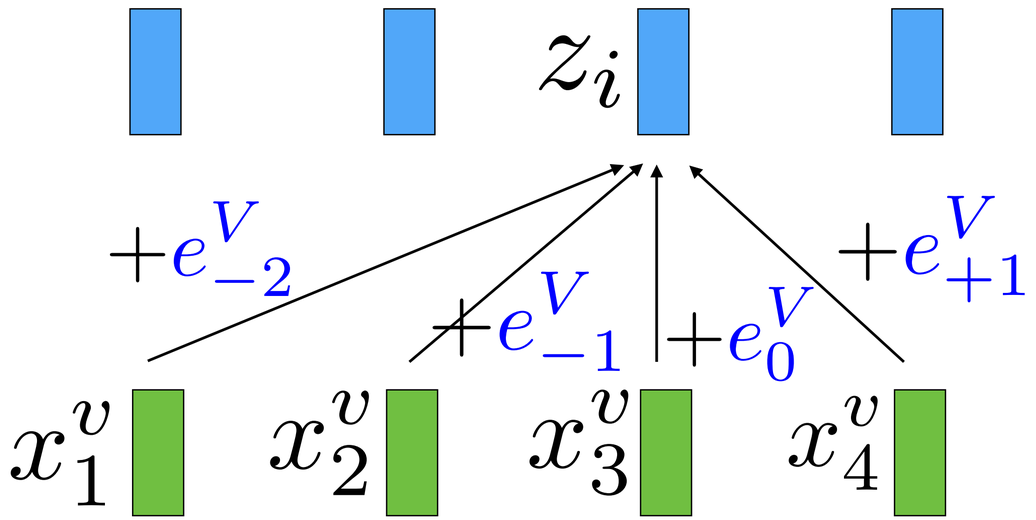
$$a_{ij} = \frac{x^q_i(x^k_j + e^K_{i-j})^T}{\sqrt{d_z}}$$

embedding of "relation"

(Applied to the AST depth-first traversal)

Shaw, P., Uszkoreit, J., and Vaswani, A. Self-Attention with Relative Position Representations. NAACL18

Sequential relative attention



Learnable embeddings $e^K_{i-j}, e^V_{i-j} \in \mathbb{R}^{d_x}$

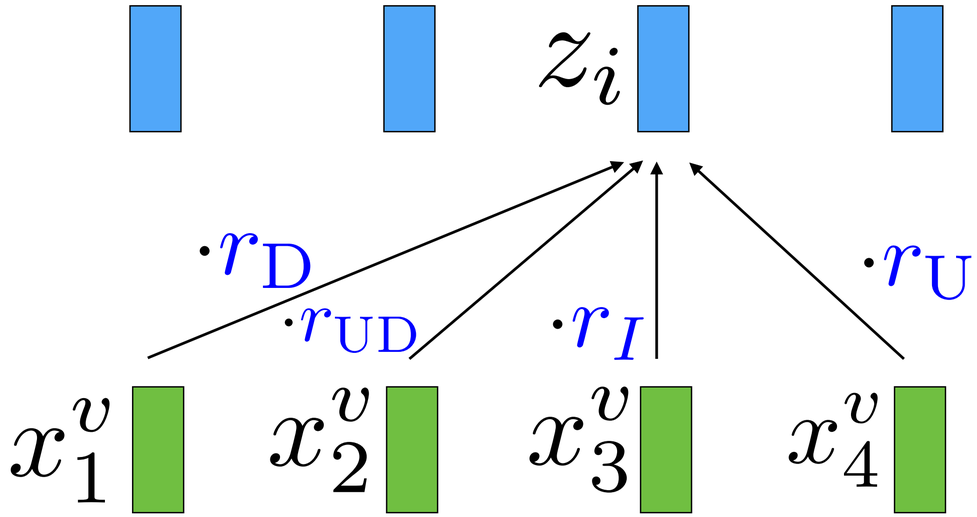
Parameters:
embeddings of “relations”

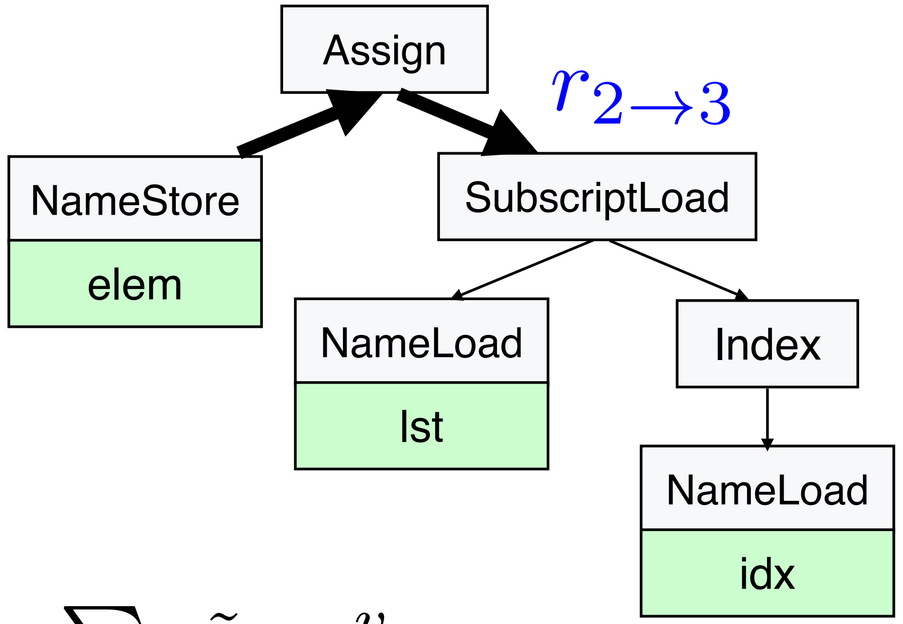
Hyperparameters:
maximum distance between tokens

(Applied to the AST depth-first traversal)

Shaw, P., Uszkoreit, J., and Vaswani, A. Self-Attention with Relative Position Representations. NAACL18

Tree relative attention





$$z_i = \sum_j \tilde{\alpha}_{ij} x_j^v$$

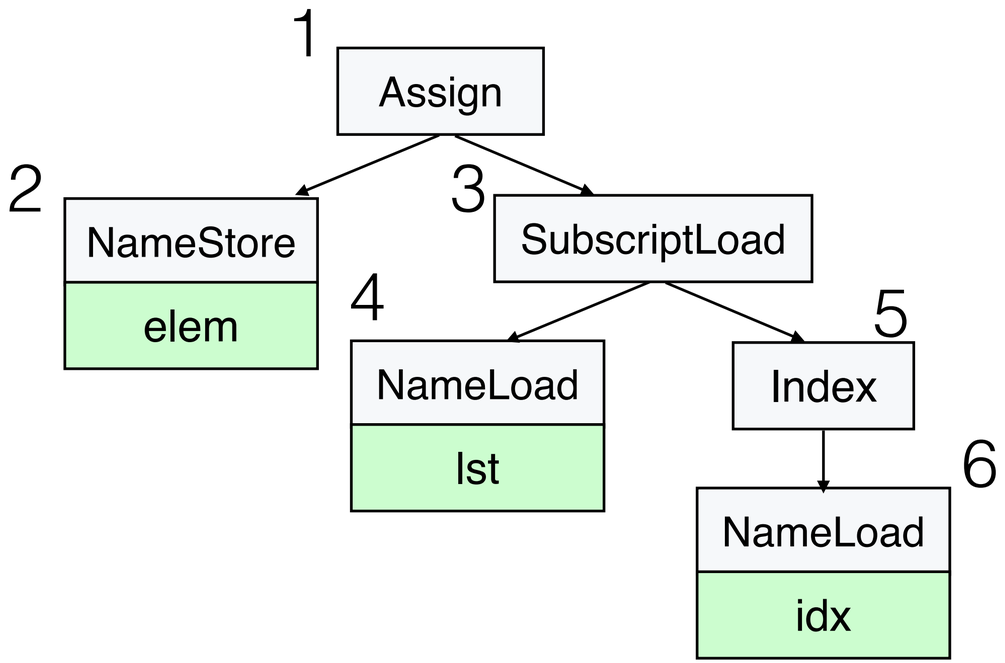
$$\tilde{\alpha}_{ij} = \frac{\exp(a_{ij} \cdot r_{j\to i})}{\sum_j \exp(a_{ij} \cdot r_{j\to i})}$$

Learnable 1-dim embeddings $r_{j\to i} \in \mathbb{R}$

$$a_{ij} = \frac{x_i^q {x_j^k}^T}{\sqrt{d_z}}$$

Kim, S., Zhao, J., Tian, Y., and Chandra, S. Code Prediction by Feeding Trees to Transformers. 2020

Tree relative attention



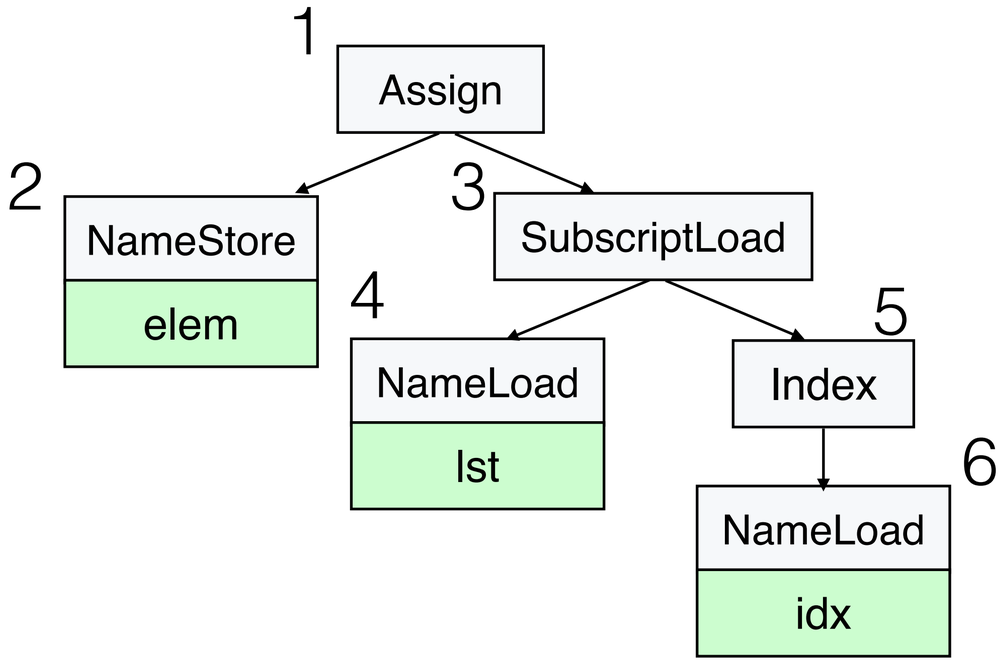
Pairwise relations between AST nodes:

	1	2	3	4	5	6
1	I	D	D	DD	DD	DDD
2	U	I	UD	UDD	UDD	UDDD
3	U	UD	I	D	D	DD
4	UU	UUD	U	I	UD	UDD
5	UU	UUD	U	UD	I	D
6	UUU	UUUD	UU	UUD	U	I

Learn 1-dim embedding for each relation

Kim, S., Zhao, J., Tian, Y., and Chandra, S. Code Prediction by Feeding Trees to Transformers. 2020

Tree relative attention



Pairwise relations between AST nodes:

	1	2	3	4	5	6
1	I	D	D	DD	DD	DDD
2	U	I	UD	UDD	UDD	UDDD
3	U	UD	I	D	D	DD
4	UU	UUD	U	I	UD	UDD
5	UU	UUD	U	UD	I	D
6	UUU	UUUD	UU	UUD	U	I

Parameters:
1-dim embeddings
of “relations”

Hyperparameters:
the maximum number of relations

Kim, S., Zhao, J., Tian, Y., and Chandra, S. Code Prediction by Feeding Trees to Transformers. 2020

Standard NLP baselines

Sequential positional encodings

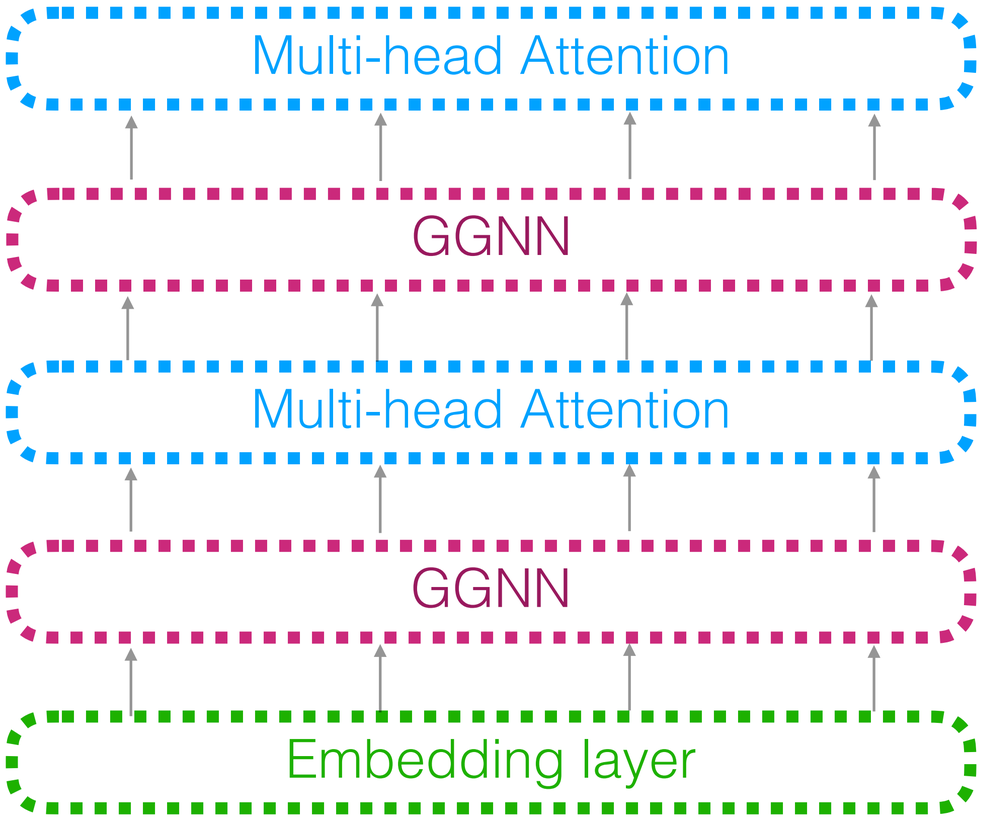
Sequential relative attention

Tree modifications

Tree positional encodings

Tree relative attention

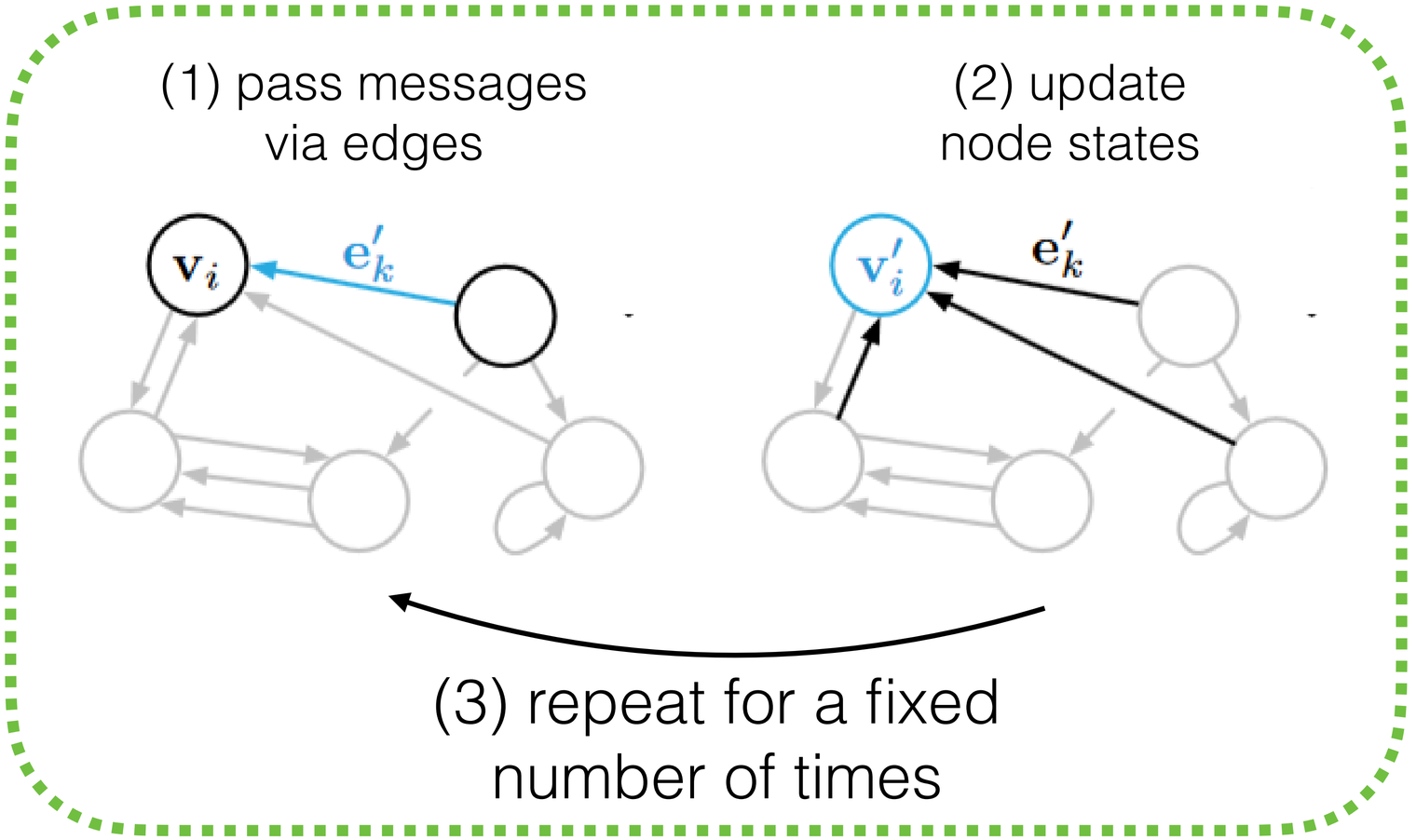
GGNN Sandwich



Transformer encoder

GGNN-Sandwich

Graph Gated Neural Network (GGNN):



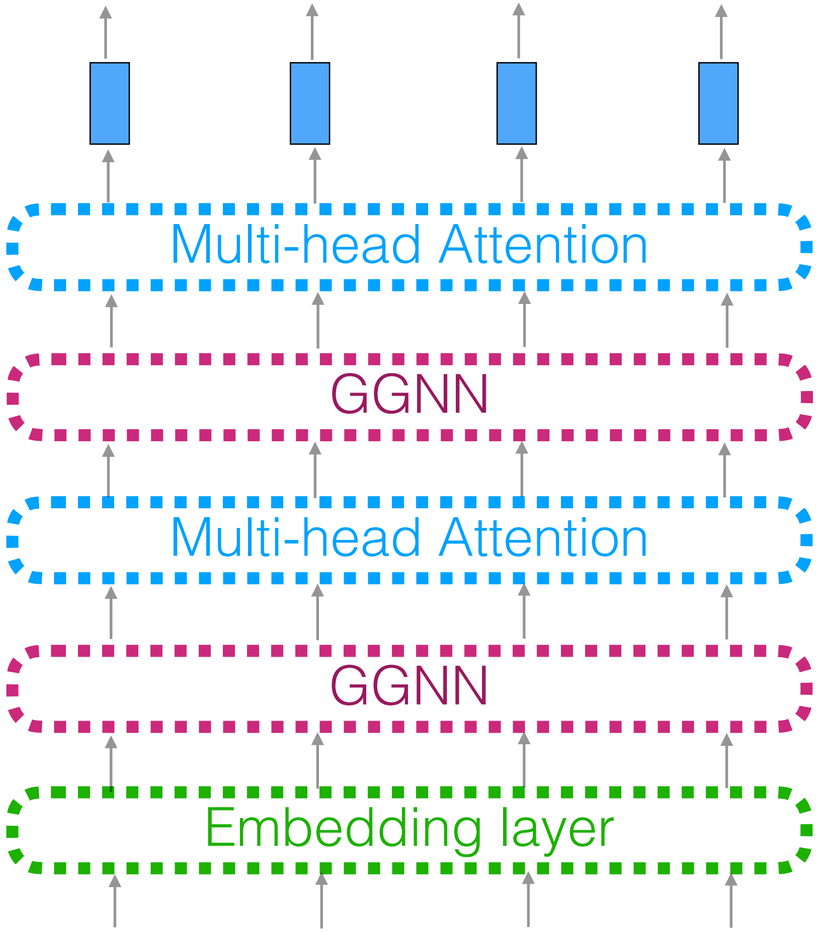
Input:
states of all nodes

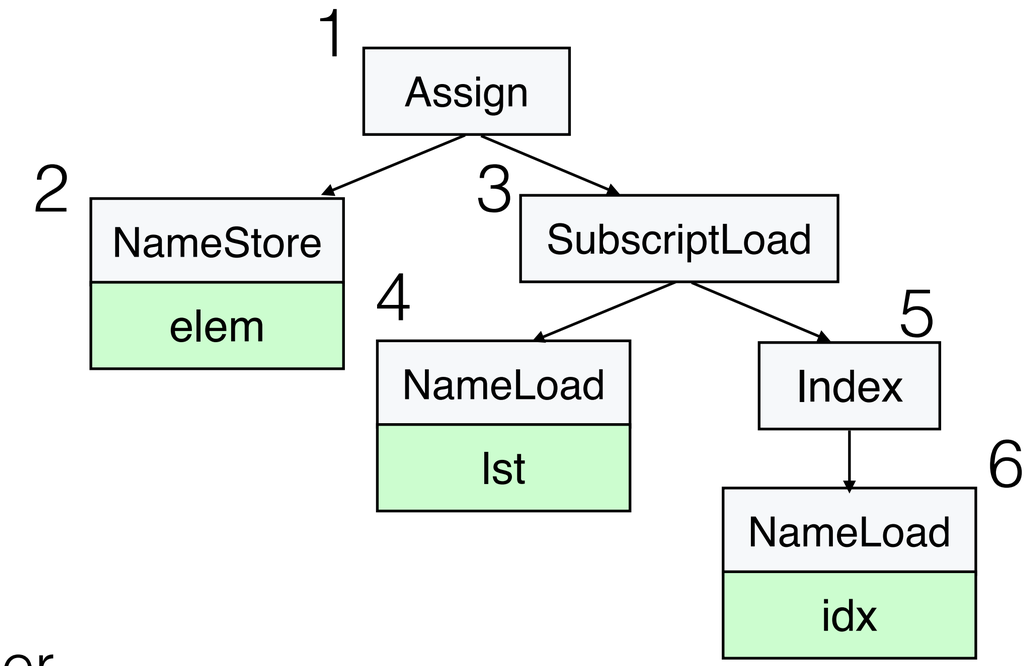
Output:
updated states
of all nodes

Arbitrary graph
structure supported,
e.g. different edge types

Image from https://aifrenz.github.io/present_file/A%20gentle%20introduction%20to%20graph%20neural%20networks.pdf

GGNN-Sandwich





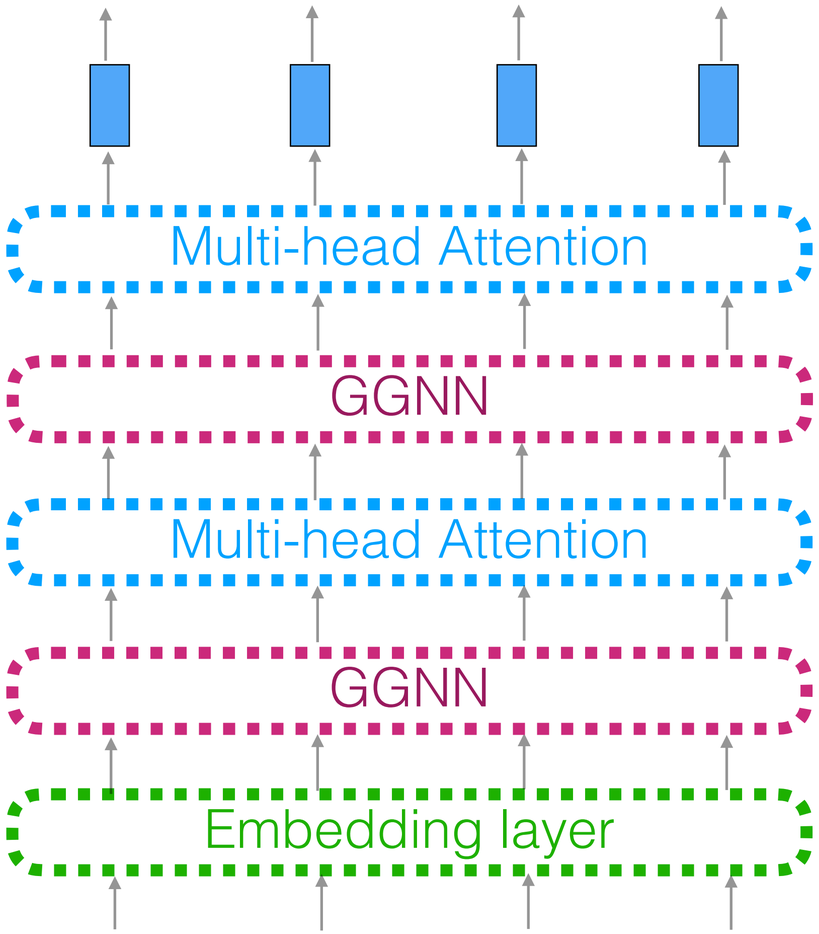
We consider 4 types of edges:

- Parent (P)
- Child (C)
- Left (L)
- Right (R)

	1	2	3	4	5	6
1		P, L	P			
2	C, R		L			
3	C	R		P, L	P	
4			C, R		L	
5			C	R		P, L
6					C, R	

Hellendoorn, V. J., Sutton, C., Singh, R., Maniatis, P., and Bieber, D. Global Relational Models of Source Code. ICLR20

GGNN-Sandwich



Parameters:
same as in GGNN

Hyperparameters:
same as in GGNN
(e. g. the number
of message passes)

Hard to use
in Transformer decoder

Hellendoorn, V. J., Sutton, C., Singh, R., Maniatis, P., and Bieber, D. Global Relational Models of Source Code. ICLR20

Tasks

Code completion:

```
import sys
from utils.parsing import parse
if __name__ == '__main__':
    argmnents = parse(
        sys.argv[1:]
    )
    print( next token?
```

Function naming:

```
def <fun_name>(filename):
    with open(filename) as fin:
        return len(
            fin.read()
                .split("\n")
        )
```

Function name?

Variable misuse

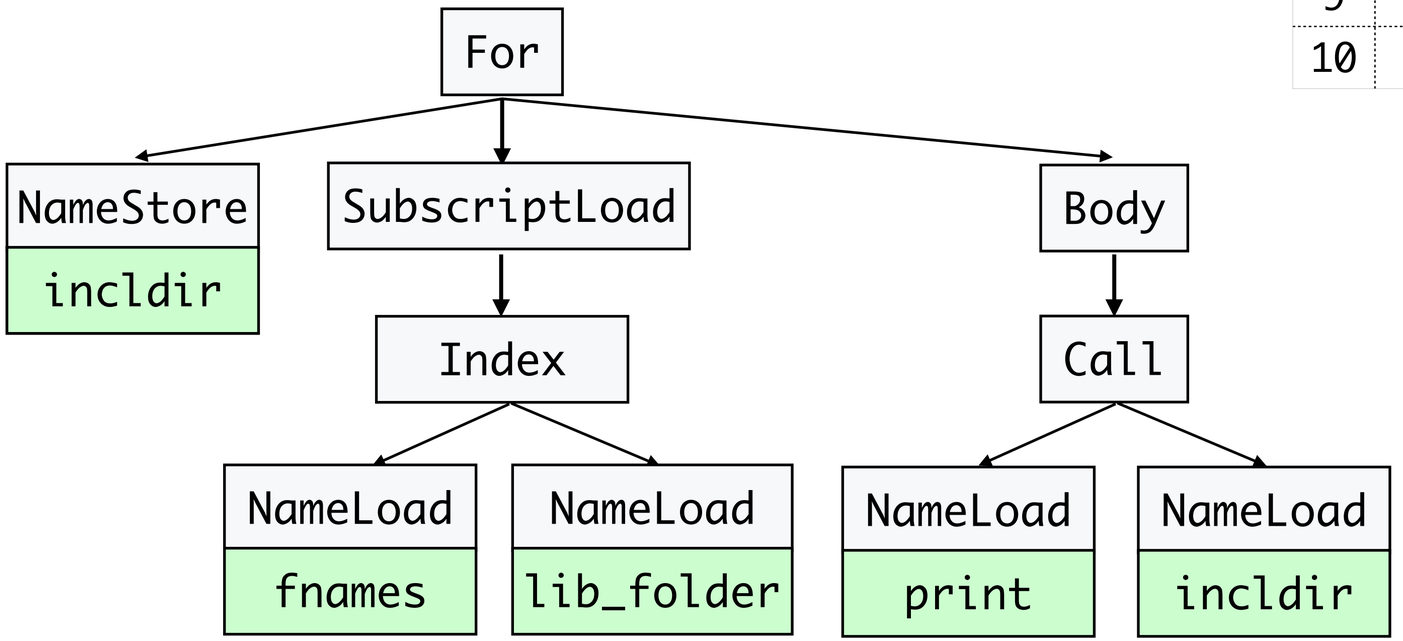
```
def count_lines (filename):
    with open(filename) as fin:
        return len(
            filename.read()
                .split("\n")
        )
```

A bug? Fix?

Preprocessing

```
for incldir in fnames[lib_folder]:
    print(incldir)
```

Abstract syntax tree



Depth-first traversal

	Type	Value
1	For	<empty>
2	NameStore	incldir
3	SubscriptLoad	<empty>
4	Index	<empty>
5	NameLoad	fnames
6	NameLoad	lib_folder
7	Body	<empty>
8	Call	<empty>
9	NameLoad	print
10	NameLoad	incldir

Sequence/Tree of (type, value) pairs

Transformer

Research questions

- What is the most effective approach for utilizing AST *structure* in Transformer?

- Is Transformer generally capable of utilizing *syntactic information* represented via AST?

- What components of AST (structure, node types and values) do Transformers use in different tasks?

Research questions

- What is the most effective approach for utilizing AST *structure* in Transformer?

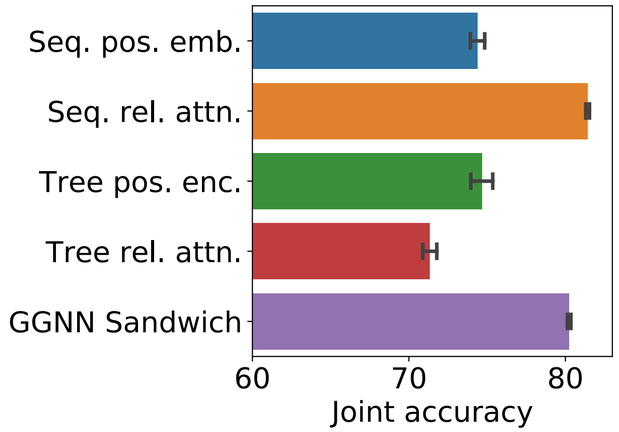
- Is Transformer generally capable of utilizing *syntactic information* represented via AST?

- What components of AST (structure, node types and values) do Transformers use in different tasks?

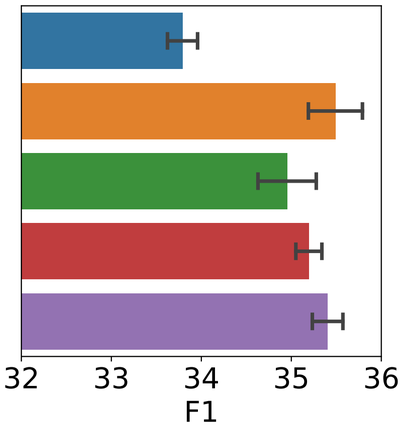
Comparison results

(the greater the better)

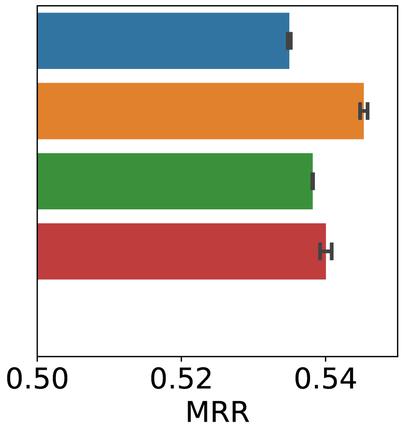
Python: Variable Misuse



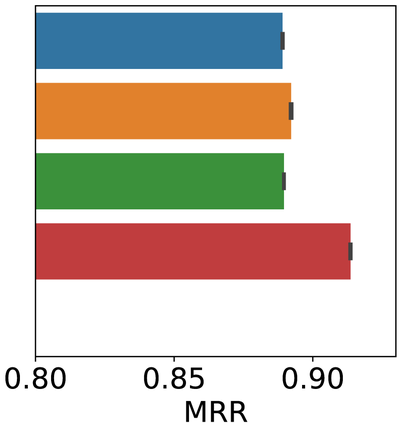
Function Naming



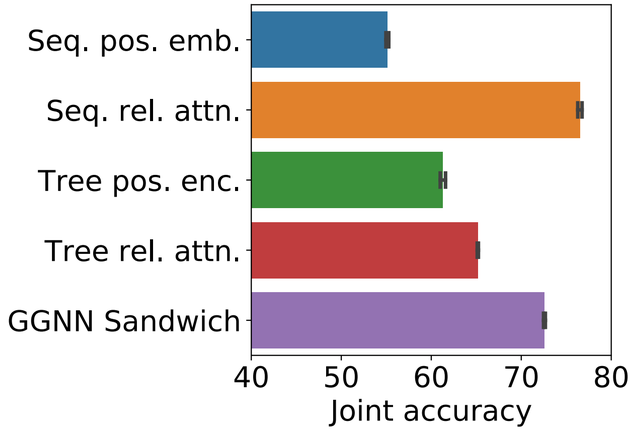
Code Completion (values)



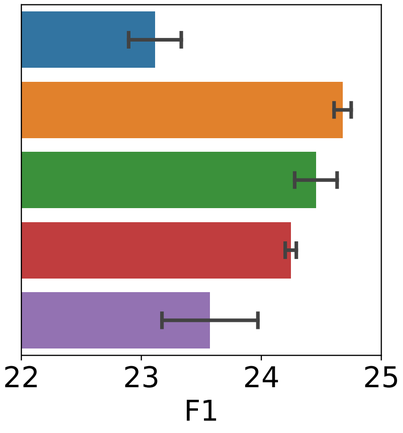
Code Completion (types)



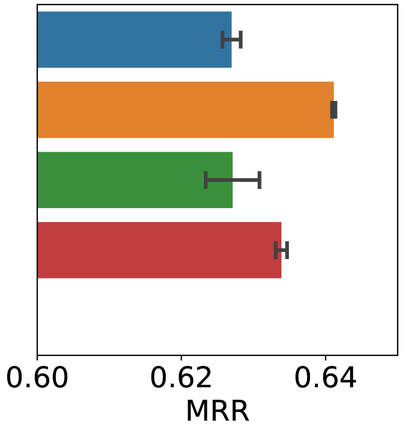
JavaScript: Variable Misuse



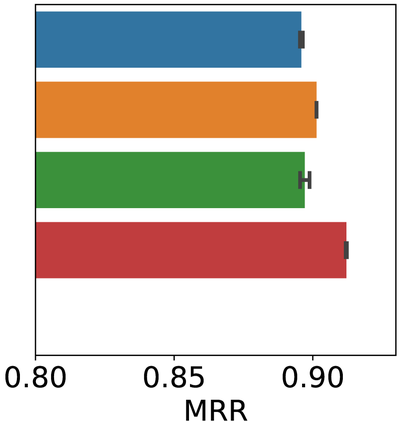
Function Naming



Code Completion (values)



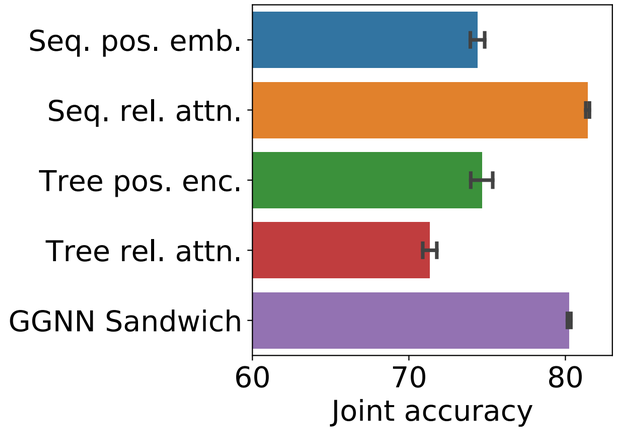
Code Completion (types)



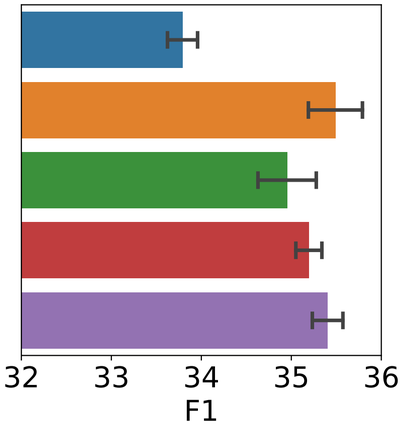
Comparison results

(the greater the better)

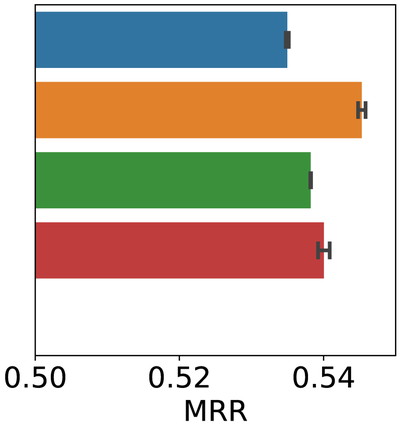
Python: Variable Misuse



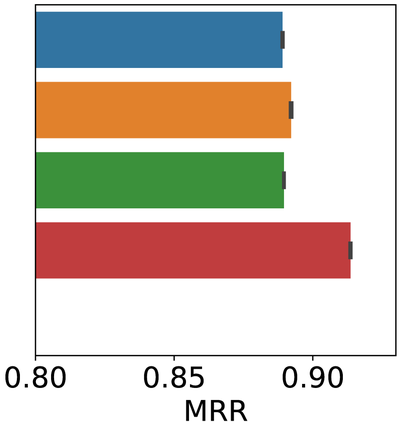
Function Naming



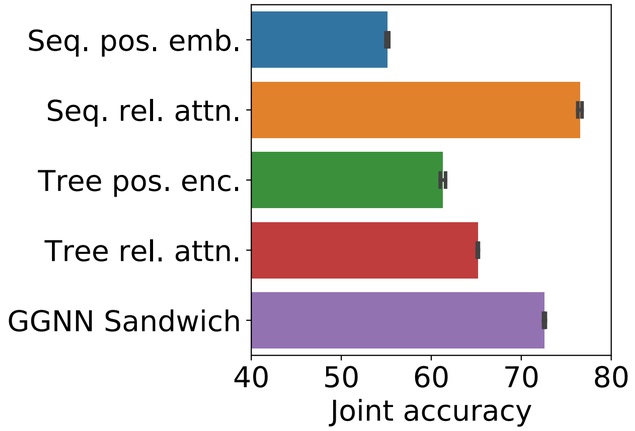
Code Completion (values)



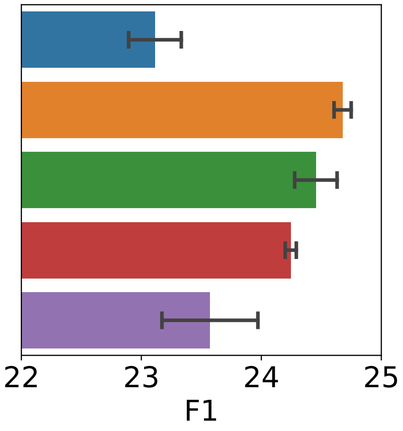
Code Completion (types)



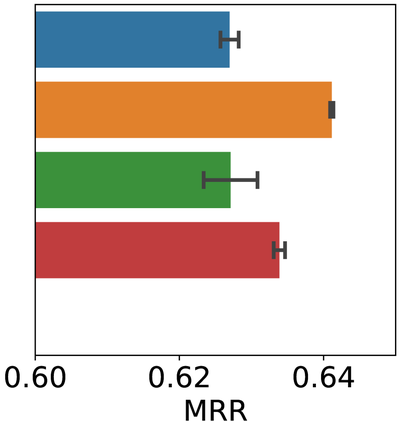
JavaScript: Variable Misuse



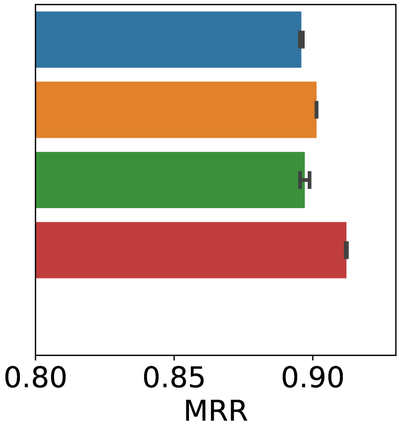
Function Naming



Code Completion (values)



Code Completion (types)

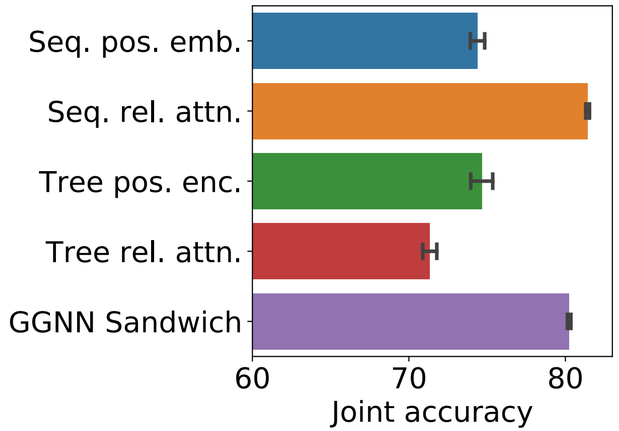


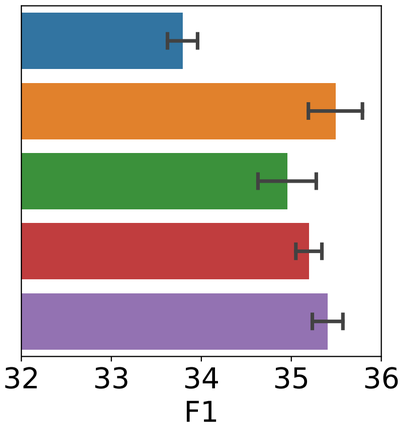
- *GGNN-Sandwich and Seq. Rel. Attn. are best for VM*

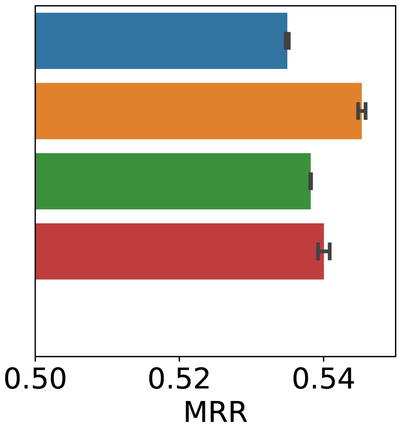
Comparison results

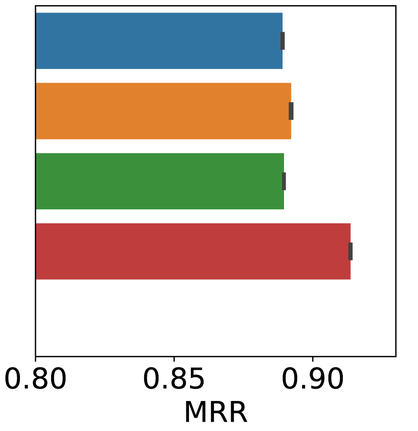
(the greater the better)

Python:

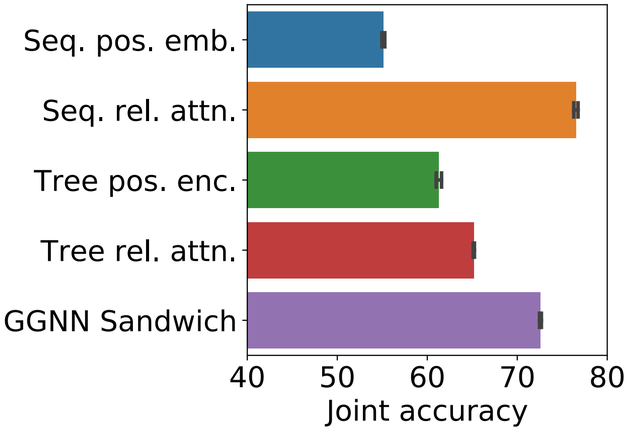


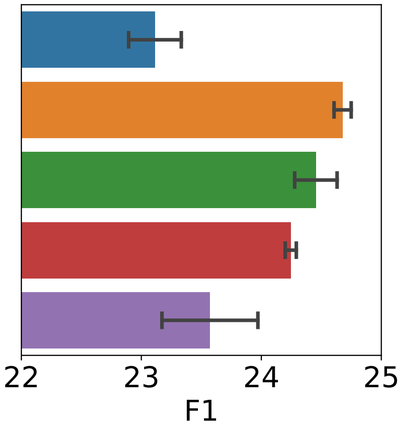


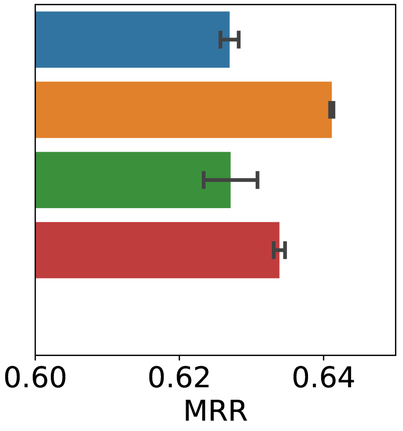


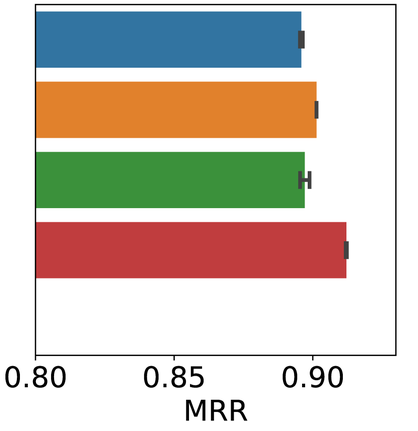


JavaScript:







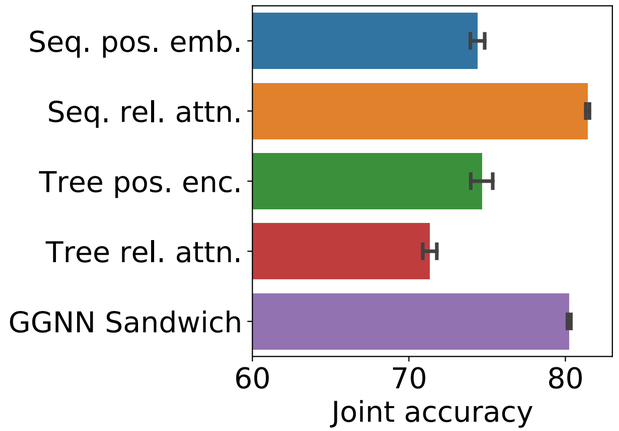


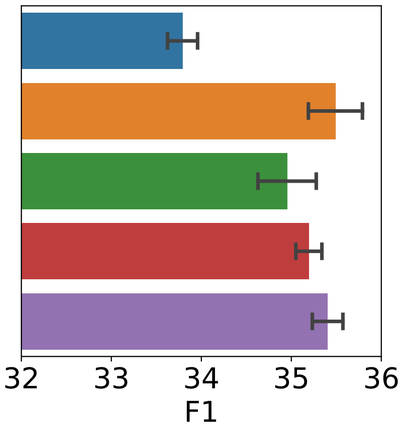
- *Almost no difference between approaches for FN*

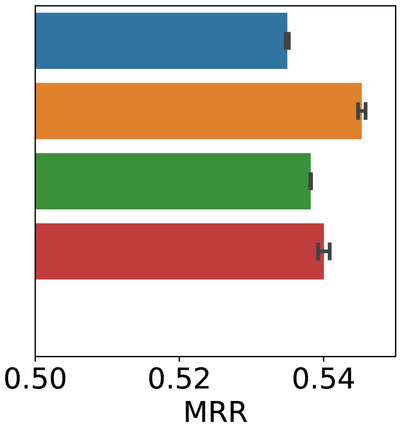
Comparison results

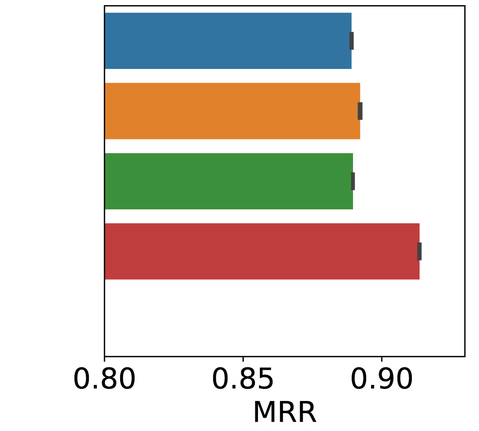
(the greater the better)

Python:

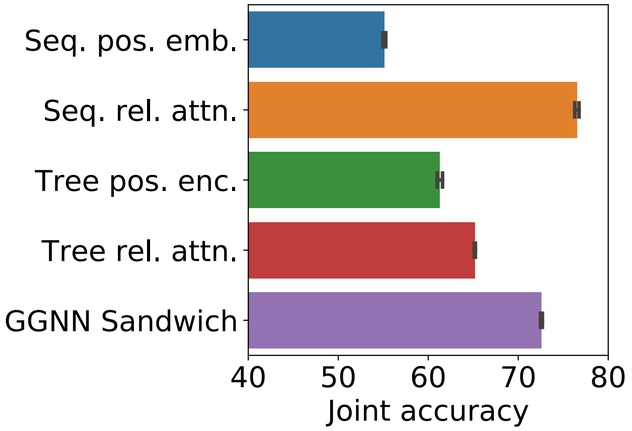


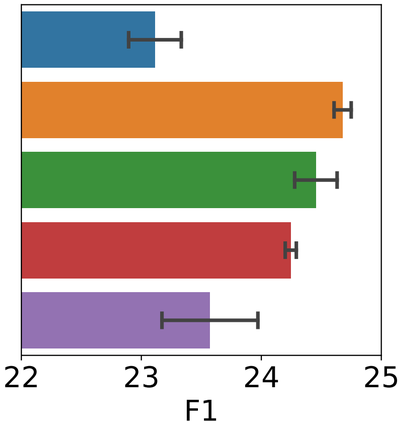


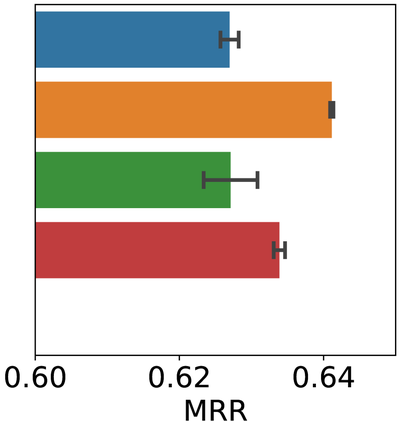


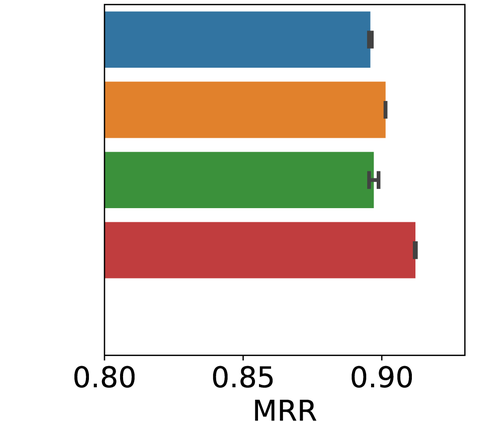


JavaScript:





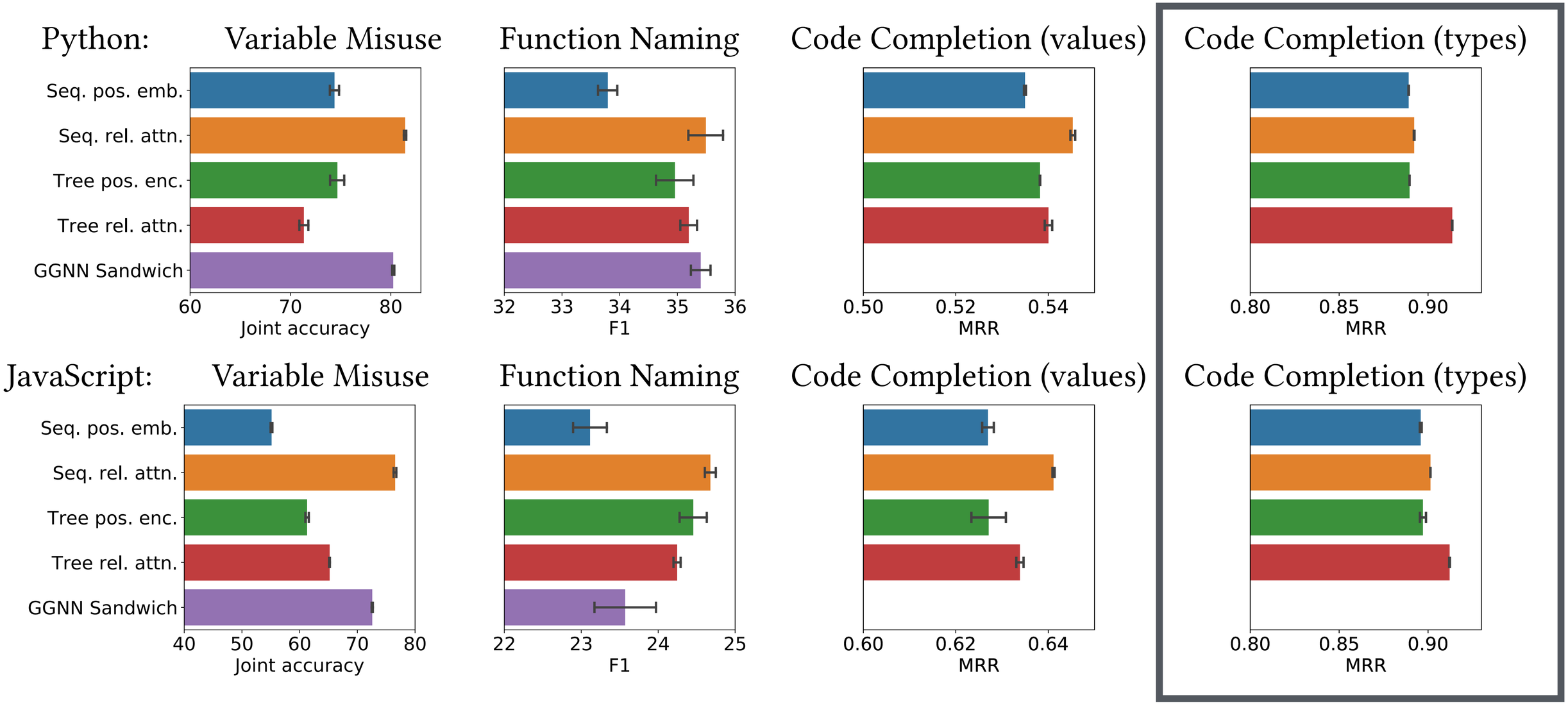




- *Seq. Rel. Attn. best for CC (value prediction)*

Comparison results

(the greater the better)

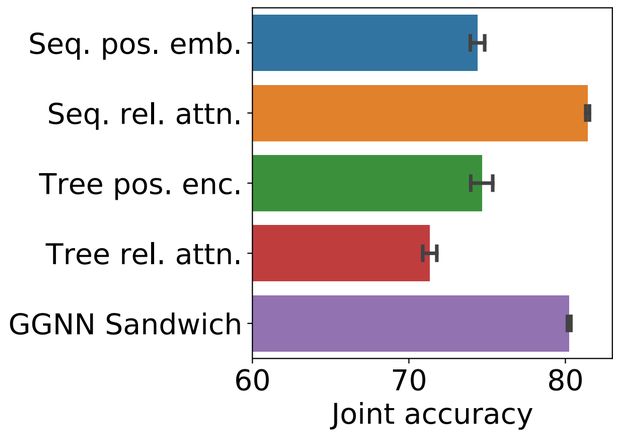


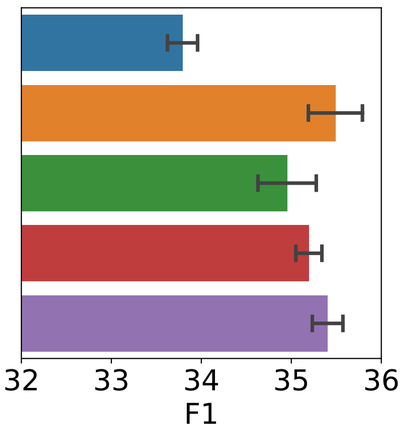
- *Tree Rel. Attn. best for CC (type prediction)*

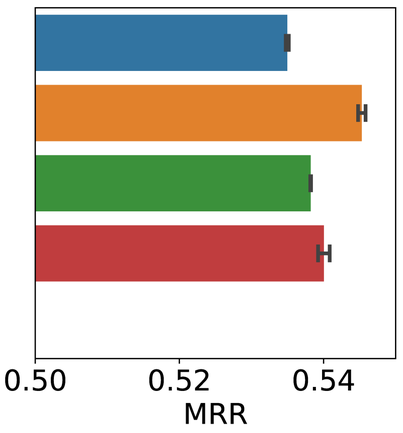
Comparison results

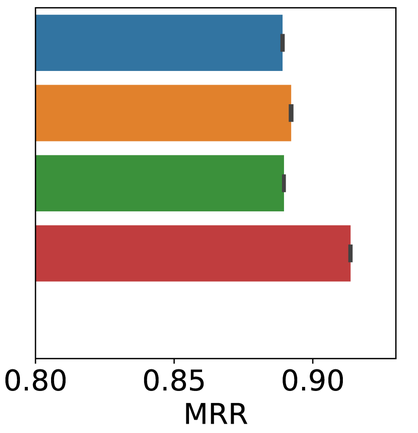
(the greater the better)

Python:

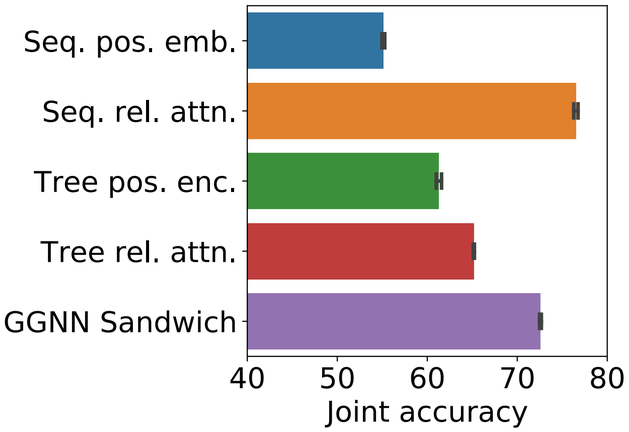


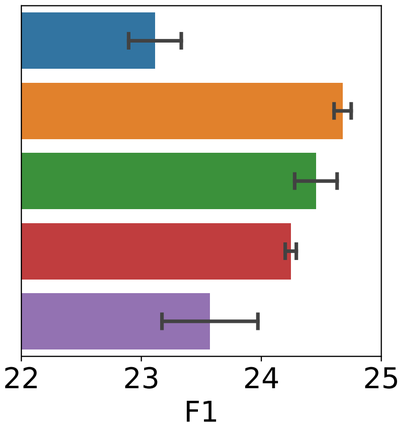


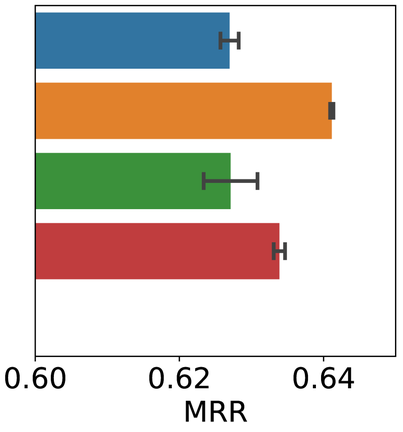


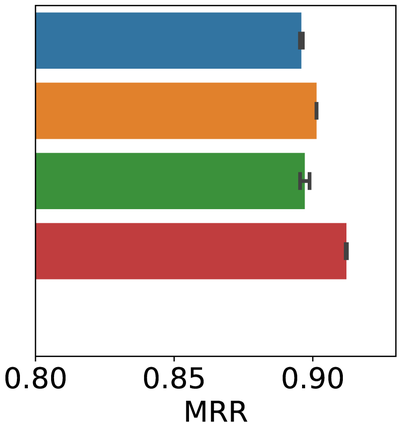


JavaScript:









- *Seq. rel. attn. the best overall, except type prediction*

Time/Memory requirements

Model	Train time (h/epoch)	Preprocess time (ms/func.)	Add. train data (GB)
Seq. pos. emb.	2.3	0	0
Seq. rel. att.	2.7	0	0
Tree pos. enc.	2.5	0.4	0.3
Tree rel. attn.	3.9	16.7	18
GGNN Sandwich	7.2	0.3	0.35

- Sequential relative attention is both time and memory efficient.

Combining different modifications

	Model	VM	FN	CC (val.)
PY	SRA	81.42	35.73	54.53
PY	SRA + Seq. pos. emb.	80.77	33.99	54.37
	SRA + Tree pos. enc.	81.73	34.71	54.63
	SRA + Tree rel. attn.	81.58	35.41	**54.91**
	SRA + GGNN sand.	82.00*	33.39	N/A
JS	SRA	76.52	24.62	64.11
JS	SRA + Seq. pos. emb.	73.17	23.09	63.97
	SRA + Tree pos. enc.	74.73	23.70	**64.49**
	SRA + Tree rel. attn.	76.34	24.71	**64.79**
	SRA + GGNN sand.	75.33*	21.44	N/A

- Performance may be further improved by combining Seq. rel. attn. with GGNN Sandwich (for VM) and Tree Rel. Attn. (for CC)

In the VM task, SRA+GGNN Sandwich significantly outperforms SRA during the first half of epochs, but loses superiority at the last epochs, for both datasets.

Research questions

- What is the most effective approach for utilizing AST *structure* in Transformer?

- Is Transformer generally capable of utilizing *syntactic information* represented via AST?

- What components of AST (structure, node types and values) do Transformers use in different tasks?

Full data setting vs Anonymized setting

```
filtered_seqs = []
for seq in seqs:
    new_seq = []
    for word in seq:
        if word in vocab:
            new_seq += [word]
    filtered_seqs.append(new_seq)
```

Two sources of information:

- Syntactic structure
- Identifier names (**seq**, **word** etc)

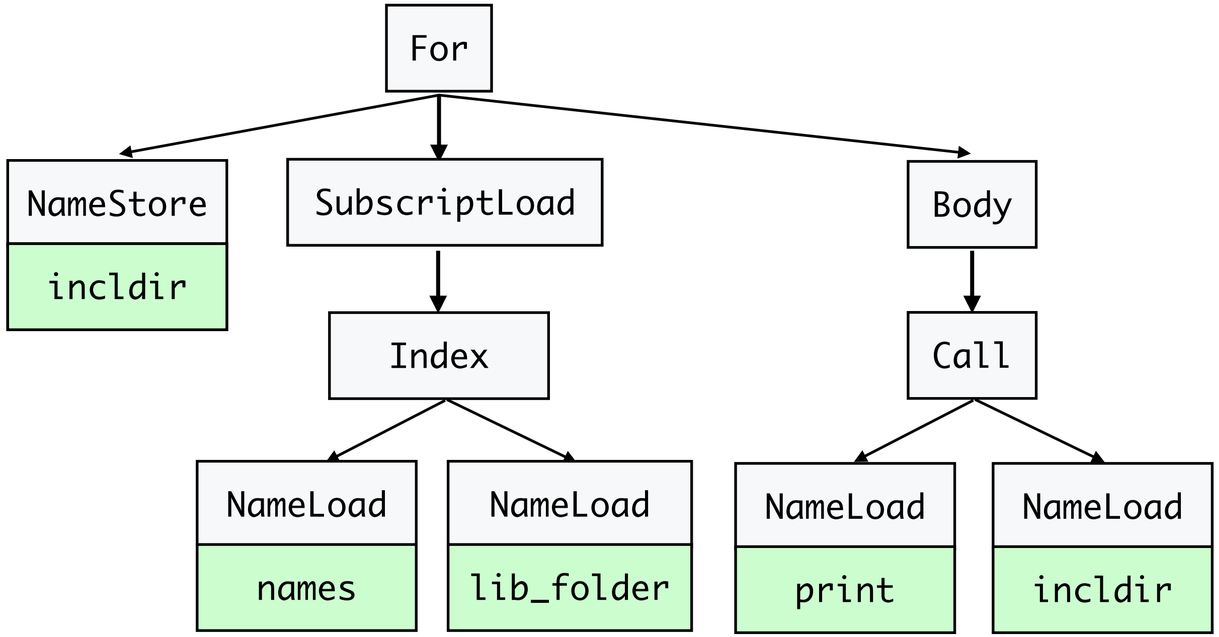
```
var78 = []
for var5 in var12:
    var31 = []
    for var1 in var5:
        if var1 in var25:
            var31 += [var1]
    var78 += [var31]
```

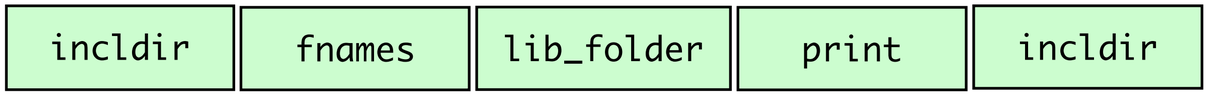
Only one source of information
— syntactic structure

var78, **var5**, **var12** etc
— anonymized identifiers

Experiment in full data setting

1) Syntax + Text vs Text



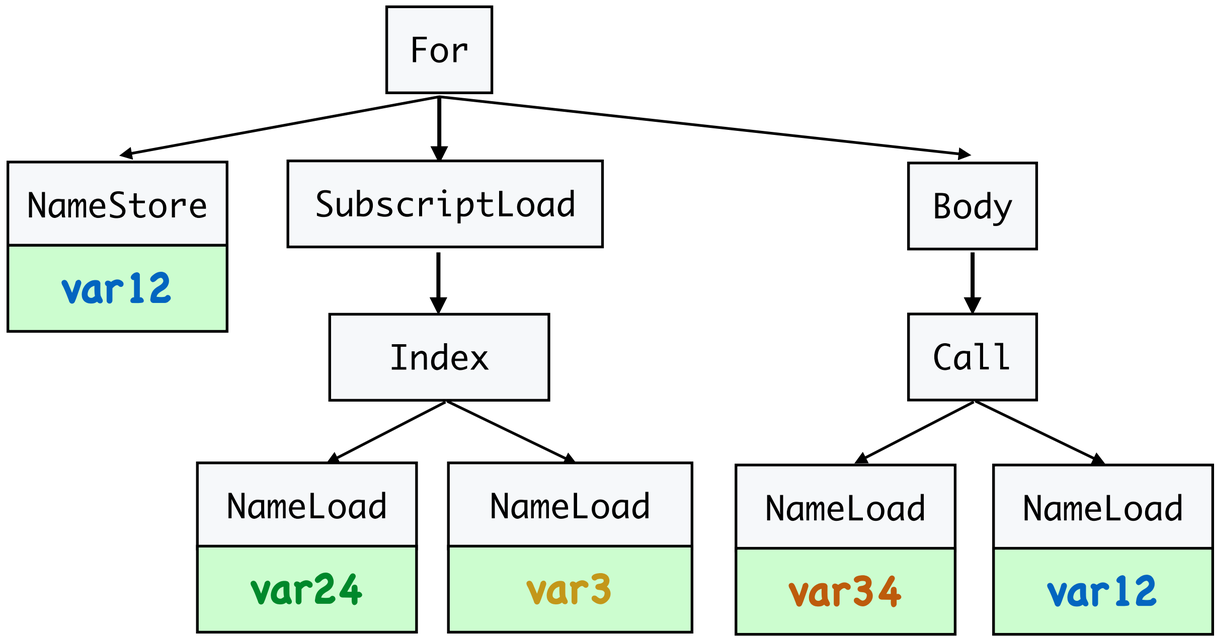


- Is Transformer generally capable of utilizing *syntactic information* represented via AST?

Experiment in anonymized setting

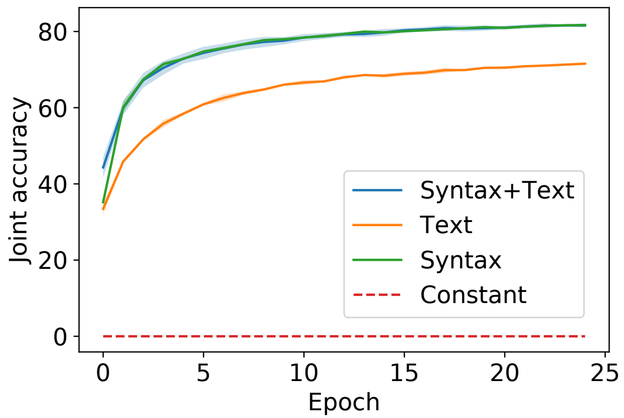
2) Syntax vs Constant prediction

(the most probable output from the train set)

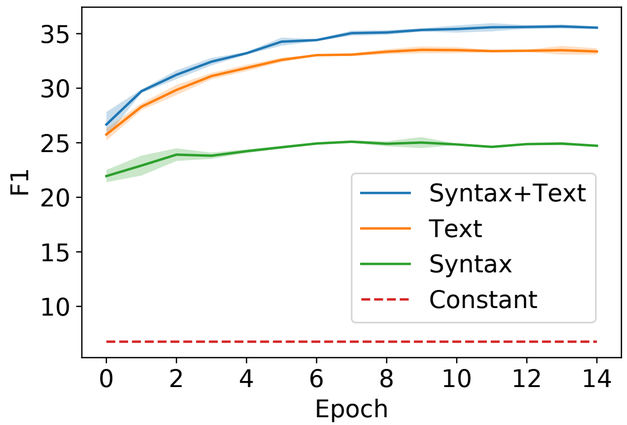


Results

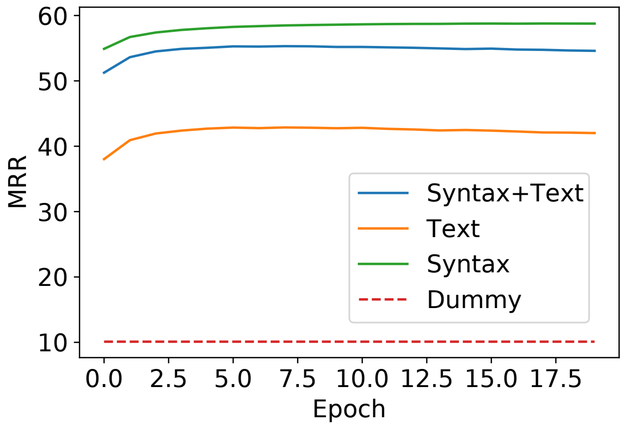
Variable Misuse – Python



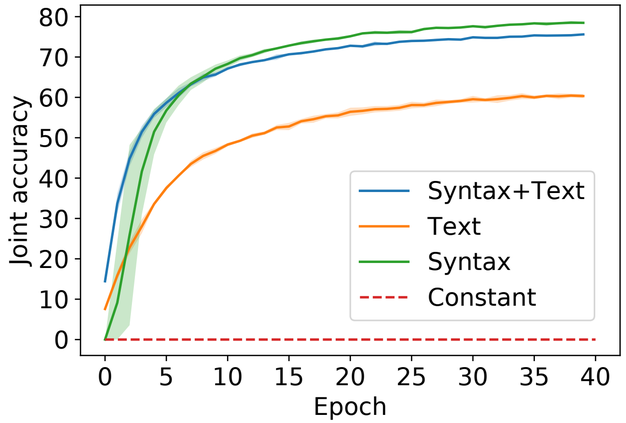
Function Naming – Python



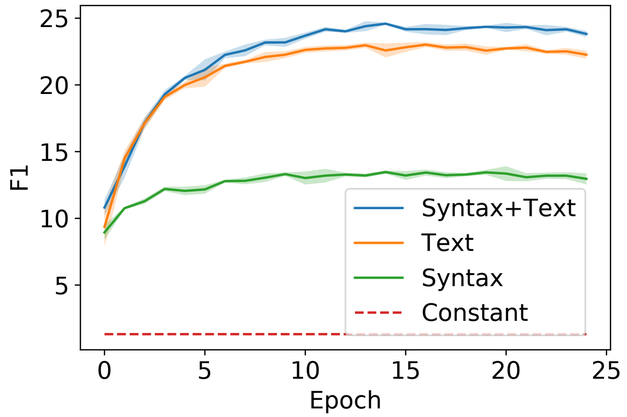
Code Completion (values) – Python



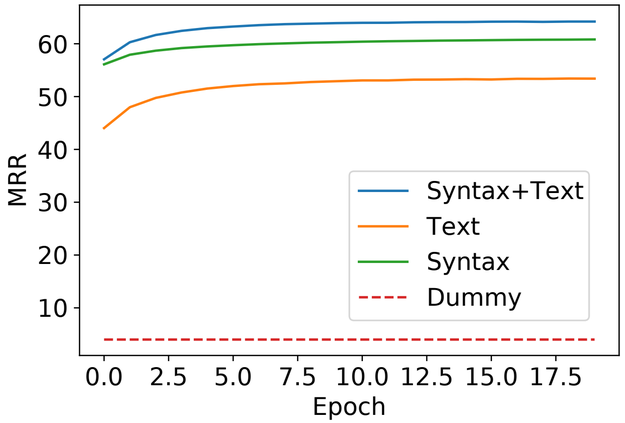
Variable Misuse – JavaScript



Function Naming – JavaScript



Code Completion (values) – JavaScript



- Syntax + Text > Text
- Syntax > Constant prediction

Research questions

- What is the most effective approach for utilizing AST *structure* in Transformer?

- Is Transformer generally capable of utilizing *syntactic information* represented via AST?

- What components of AST (structure, node types and values) do Transformers use in different tasks?

Ablation study

		Full data			Anonymized data		
		Var. misuse	Fun. naming	Comp. (val.)	Var. misuse	Fun. naming	Comp. (val.)
Python	Full AST	**81.59**±0.50%	**35.73**±0.19%	**54.59**±0.2%	**81.71**±0.41%	**25.26**±0.15%	**58.76**±0.2%
	AST w/o struct.	26.81±0.47%	34.80±0.24%	53.1±0.1%	12.41±0.58%	23.29±0.18%	57.75±0.05%
	AST w/o types	71.55±0.28%	33.60±0.23%	42.01±0.05%	58.55±0.51 %	12.50±1.5%	41.26±0.05%
	AST w/o values	32.44±0.35%	25.25 ±0.06%	N/A	32.44±0.35%	**25.25**±0.06%	N/A
JavaScript	Full AST	**75.60**±0.15%	**24.62**±0.14%	**64.2**±0.05	**78.47**±0.26%	**13.66**±0.30%	**60.82**±0.07%
	AST w/o struct.	17.25±0.83%	23.40±0.12%	61.53±0.15%	5.37±0.97%	11.25±0.08%	58.59±0.1%
	AST w/o types	60.33±0.50%	23.09±0.09%	53.4±0.1 %	43.53±0.92 %	8.10±1.4%	42.91±0.1%
	AST w/o values	42.56±0.24%	13.64±0.07%	N/A	42.56±0.24%	**13.64**±0.07%	N/A

- Full AST: values + types + seq. rel. att
- AST w/o struct: values + types
- AST w/o types: values + seq. rel. att
- AST w/o values: types only + seq. rel. att

Ablation study

		Full data			Anonymized data		
		Var. misuse	Fun. naming	Comp. (val.)	Var. misuse	Fun. naming	Comp. (val.)
Python	Full AST	**81.59**±0.50%	**35.73**±0.19%	**54.59**±0.2%	**81.71**±0.41%	**25.26**±0.15%	**58.76**±0.2%
	AST w/o struct.	26.81±0.47%	34.80±0.24%	53.1±0.1%	12.41±0.58%	23.29±0.18%	57.75±0.05%
	AST w/o types	71.55±0.28%	33.60±0.23%	42.01±0.05%	58.55±0.51 %	12.50±1.5%	41.26±0.05%
	AST w/o values	32.44±0.35%	25.25 ±0.06%	N/A	32.44±0.35%	**25.25**±0.06%	N/A
JavaScript	Full AST	**75.60**±0.15%	**24.62**±0.14%	**64.2**±0.05	**78.47**±0.26%	**13.66**±0.30%	**60.82**±0.07%
	AST w/o struct.	17.25±0.83%	23.40±0.12%	61.53±0.15%	5.37±0.97%	11.25±0.08%	58.59±0.1%
	AST w/o types	60.33±0.50%	23.09±0.09%	53.4±0.1 %	43.53±0.92 %	8.10±1.4%	42.91±0.1%
	AST w/o values	42.56±0.24%	13.64±0.07%	N/A	42.56±0.24%	**13.64**±0.07%	N/A

- Full AST: values + types + seq. rel. att
- AST w/o struct: values + types
- AST w/o types: values + seq. rel. att
- AST w/o values: types only + seq. rel. att

- For VM & CC (val.) ablating *input structure* / *types* / an.values hurts quality

Ablation study

		Full data			Anonymized data		
		Var. misuse	Fun. naming	Comp. (val.)	Var. misuse	Fun. naming	Comp. (val.)
Python	Full AST	**81.59**±0.50%	**35.73**±0.19%	**54.59**±0.2%	**81.71**±0.41%	**25.26**±0.15%	**58.76**±0.2%
	AST w/o struct.	26.81±0.47%	34.80±0.24%	53.1±0.1%	12.41±0.58%	23.29±0.18%	57.75±0.05%
	AST w/o types	71.55±0.28%	33.60±0.23%	42.01±0.05%	58.55±0.51 %	12.50±1.5%	41.26±0.05%
	AST w/o values	32.44±0.35%	25.25 ±0.06%	N/A	32.44±0.35%	**25.25**±0.06%	N/A
JavaScript	Full AST	**75.60**±0.15%	**24.62**±0.14%	**64.2**±0.05	**78.47**±0.26%	**13.66**±0.30%	**60.82**±0.07%
	AST w/o struct.	17.25±0.83%	23.40±0.12%	61.53±0.15%	5.37±0.97%	11.25±0.08%	58.59±0.1%
	AST w/o types	60.33±0.50%	23.09±0.09%	53.4±0.1 %	43.53±0.92 %	8.10±1.4%	42.91±0.1%
	AST w/o values	42.56±0.24%	13.64±0.07%	N/A	42.56±0.24%	**13.64**±0.07%	N/A

- Full AST: values + types + seq. rel. att
- AST w/o struct: values + types
- AST w/o types: values + seq. rel. att
- AST w/o values: types only + seq. rel. att

- For FN ablating *structure* / *types* hurts quality marginally

Ablation study

		Full data			Anonymized data		
		Var. misuse	Fun. naming	Comp. (val.)	Var. misuse	Fun. naming	Comp. (val.)
Python	Full AST	**81.59**±0.50%	**35.73**±0.19%	**54.59**±0.2%	**81.71**±0.41%	**25.26**±0.15%	**58.76**±0.2%
	AST w/o struct.	26.81±0.47%	34.80±0.24%	53.1±0.1%	12.41±0.58%	23.29±0.18%	57.75±0.05%
	AST w/o types	71.55±0.28%	33.60±0.23%	42.01±0.05%	58.55±0.51 %	12.50±1.5%	41.26±0.05%
	AST w/o values	32.44±0.35%	25.25 ±0.06%	N/A	32.44±0.35%	**25.25**±0.06%	N/A
JavaScript	Full AST	**75.60**±0.15%	**24.62**±0.14%	**64.2**±0.05	**78.47**±0.26%	**13.66**±0.30%	**60.82**±0.07%
	AST w/o struct.	17.25±0.83%	23.40±0.12%	61.53±0.15%	5.37±0.97%	11.25±0.08%	58.59±0.1%
	AST w/o types	60.33±0.50%	23.09±0.09%	53.4±0.1 %	43.53±0.92 %	8.10±1.4%	42.91±0.1%
	AST w/o values	42.56±0.24%	13.64±0.07%	N/A	42.56±0.24%	**13.64**±0.07%	N/A

- Full AST: values + types + seq. rel. att
- AST w/o struct: values + types
- AST w/o types: values + seq. rel. att
- AST w/o values: types only + seq. rel. att

- For FN ablating *structure* / *types* hurts quality marginally
- For FN only ablating *values* hurts quality substantially

Summary

Practical recommendations:

- to use s*equential relative attention* applied over AST depth-first traversal
- combining seq.rel.att. with GGNN Sandwich or tree relative attention can further improve performance
- omitting types, values or edges in AST hurts performance

Conceptual contributions:

- Transformers are capable of making meaningful predictions based purely on syntactic information (without identifiers)
- In function naming, Transformers mostly rely on variable names and hardly use code structure

Code: https://github.com/bayesgroup/code_transformers

HIGHER SCHOOL OF ECONOMICS
NATIONAL RESEARCH UNIVERSITY

$p(\mathbf{B}|\mathbf{A})$yesgroup.ru

Paper backstage

Questions from reviewers:

AAAI 2021:

- how do you choose hyperparameters?
- why only one dataset?
- why do you use this data and not this redistributable data?
- lack of novelty

Reject from AAAI

ESEC/FSE 2021:

- what is the value for practitioners?
- were all the models well-trained?

Accept to ESEC/FSE (A*)

OpenAI Codex

Blog: https://openai.com/blog/openai-codex/

Evaluating Large Language Models Trained on Code

Mark Chen [*1] **Jerry Tworek** [*1] **Heewoo Jun** [*1] **Qiming Yuan** [*1] **Henrique Ponde de Oliveira Pinto** [*1]
Jared Kaplan [*2] **Harri Edwards** [1] **Yuri Burda** [1] **Nicholas Joseph** [2] **Greg Brockman** [1] **Alex Ray** [1] **Raul Puri** [1]
Gretchen Krueger [1] **Michael Petrov** [1] **Heidy Khlaaf** [3] **Girish Sastry** [1] **Pamela Mishkin** [1] **Brooke Chan** [1]
Scott Gray [1] **Nick Ryder** [1] **Mikhail Pavlov** [1] **Alethea Power** [1] **Lukasz Kaiser** [1] **Mohammad Bavarian** [1]
Clemens Winter [1] **Philippe Tillet** [1] **Felipe Petroski Such** [1] **Dave Cummings** [1] **Matthias Plappert** [1]
Fotios Chantzis [1] **Elizabeth Barnes** [1] **Ariel Herbert-Voss** [1] **William Hebgen Guss** [1] **Alex Nichol** [1] **Alex Paino** [1]
Nikolas Tezak [1] **Jie Tang** [1] **Igor Babuschkin** [1] **Suchir Balaji** [1] **Shantanu Jain** [1] **William Saunders** [1]
Christopher Hesse [1] **Andrew N. Carr** [1] **Jan Leike** [1] **Josh Achiam** [1] **Vedant Misra** [1] **Evan Morikawa** [1]
Alec Radford [1] **Matthew Knight** [1] **Miles Brundage** [1] **Mira Murati** [1] **Katie Mayer** [1] **Peter Welinder** [1]
Bob McGrew [1] **Dario Amodei** [2] **Sam McCandlish** [2] **Ilya Sutskever** [1] **Wojciech Zaremba** [1]

Abstract

We introduce Codex, a GPT language model fine-tuned on publicly available code from GitHub, and study its Python code-writing capabilities. A distinct production version of Codex powers

1. Introduction

Scalable sequence prediction models (Graves, 2014; Vaswani et al., 2017; Child et al., 2019) have become a general-purpose method for generation and representation learning in many domains, including natural language processing (Mikolov et al., 2013; Sutskever et al., 2014; Dai &

Creating a Space Game with OpenAI Codex

https://youtu.be/Zm9B-DvwOgw

Talking to Your Computer with OpenAI Codex

https://youtu.be/-Dpl2awseZU

Codex: GPT-3 model trained on Github

- Goal: build a model, which is able to understand an *intent* of a programmer and generate code

'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe:
All mimsy were the borogoves,
And the mome raths outgrabe.

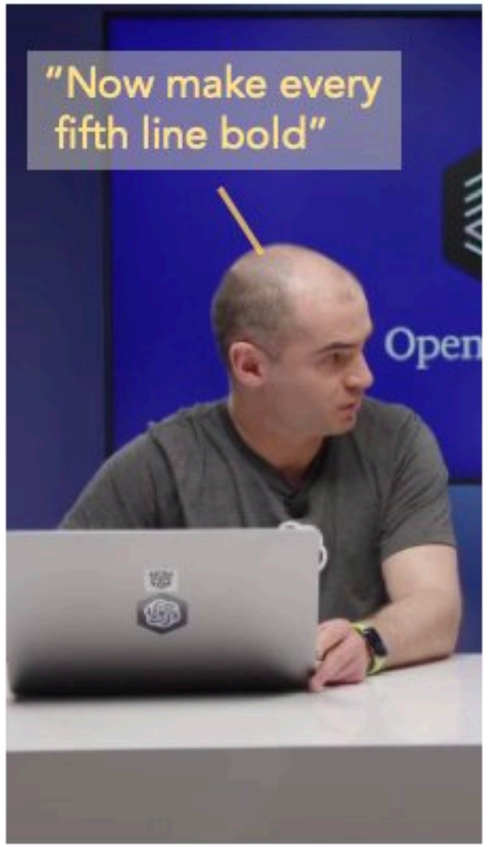
"Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
The frumious Bandersnatch!"

He took his vorpal sword in hand;
Long time the manxome foe he sought—
So rested he by the Tumtum tree
And stood awhile in thought.

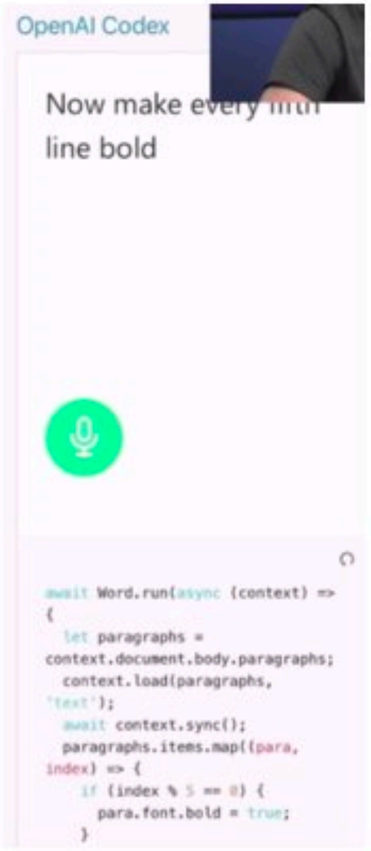
And, as in uffish thought he stood,
The Jabberwock, with eyes of flame,
Came whiffling through the tulgey wood,
And burbled as it came!

One, two! One, two! And through and through
The vorpal blade went snicker-snack!
He left it dead, and with its head
He went galumphing back.

→



→



'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe:
All mimsy were the borogoves,
And the mome raths outgrabe.

"Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
The frumious Bandersnatch!"

He took his vorpal sword in hand;
Long time the manxome foe he sought—
So rested he by the Tumtum tree
And stood awhile in thought.

And, as in uffish thought he stood,
The Jabberwock, with eyes of flame,
Came whiffling through the tulgey wood,
And burbled as it came!

One, two! One, two! And through and through
The vorpal blade went snicker-snack!
He left it dead, and with its head
He went galumphing back.

https://twitter.com/TomerUllman/status/1435225486716653579

Back to GPT-3 model

- A Large Transformer decoder
- The task: next token prediction

Alan Turing and Claude Shannon are sitting at a table of a small cafe in Paris. There is a glass of water on the table. Depending on the interpretation it is either half full or half empty. The conversation is about information theory and Turing patterns.

Develop a dialogue out of this situation.

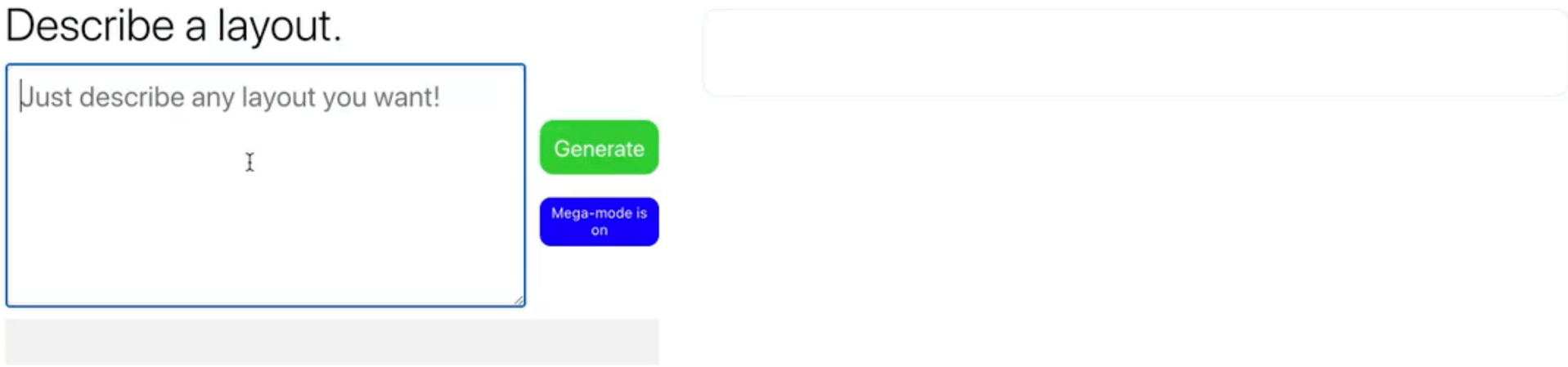
Shannon: "I read your paper on morphogenesis. It is fascinating and complex. Can you summarize it in a few sentences?"

Turing: "Yes. It is about patterns of development. We find them everywhere. In art, in science, in nature, in society and even in the very laws of physics. They are universal. Such patterns an be found in a cluster of branches in a forest, in the distribution of galaxies in the universe or in the periodic table of the elements."

Shannon: "Yes, I do remember the analogy to the periodic table. But that was only an anology."

GPT-3 model

- Even GPT-3 was able to generate code!



https://twitter.com/sharifshameem/status/1282676454690451457?s=20

Codex: GPT-3 model trained on Github

- Goal: Train a model, which is able to understand an intent of a programmer and generate code
- Data: 54 million public software repositories from Github
- Hypothesis: training of large code corpus > training on the whole internet with a little fraction of code data
- Problem: no established benchmarks to evaluate code generation

Evaluation approach

- Known fact: BLEU is poor metric for code generation
  How do developers check the correctness of code?

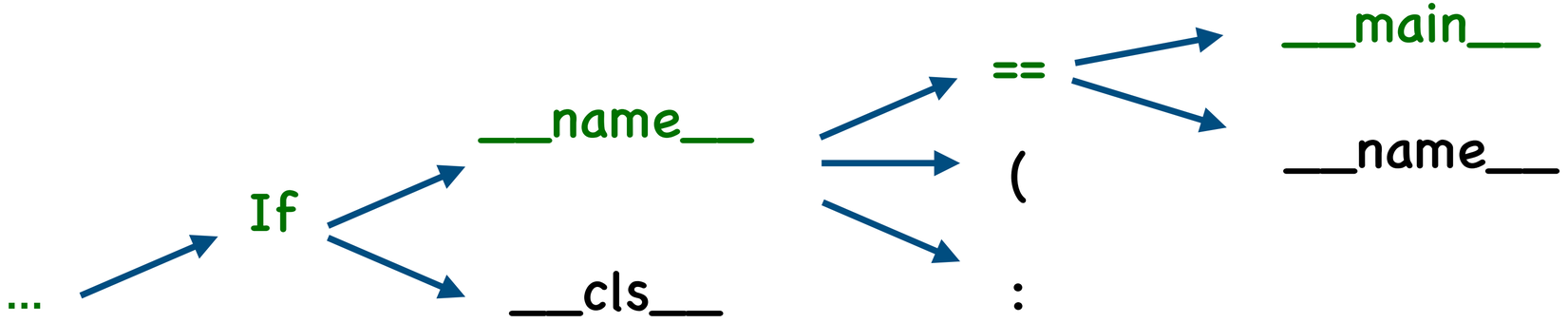
Evaluation approach

- Known fact: BLEU is poor metric for code generation
  How do developers check the correctness of code?
- Unit tests!

Evaluation approach

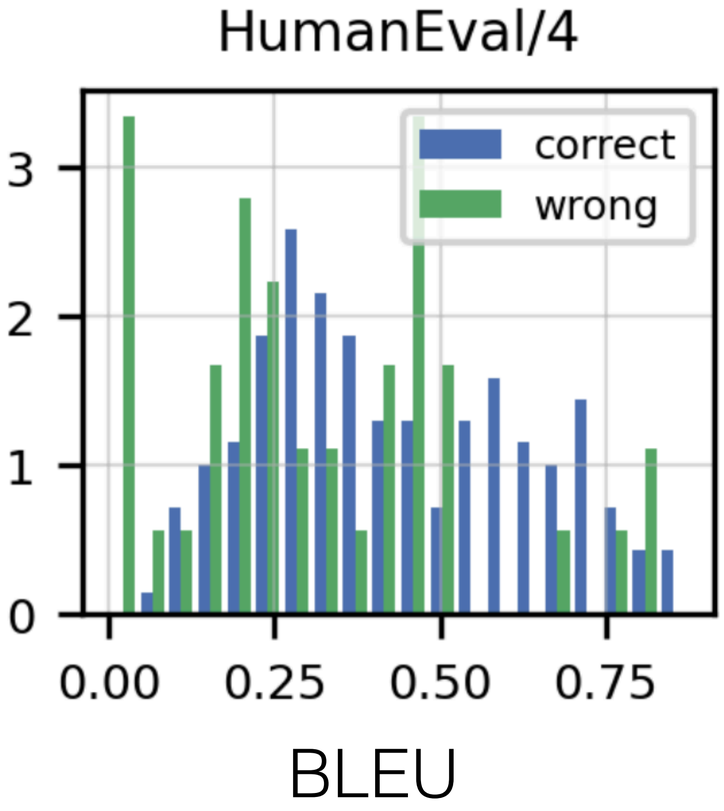
- Known fact: BLEU is poor metric for code generation
  How do developers check the correctness of code?
- Unit tests!

- Sample a number of predictions from the model
- Evaluate the samples on tests cases



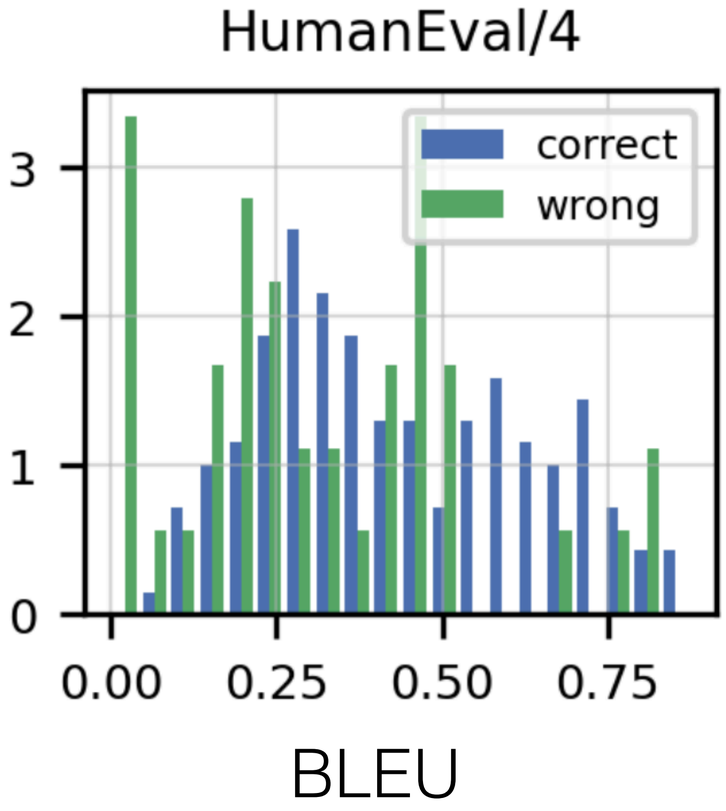
Evaluation approach

- Known fact: BLEU is poor metric for code generation
  How do developers check the correctness of code?
- Unit tests!
- Functional correctness:
  pass@k metric: at least 1 model sample from k pass the tests



Evaluation approach

- Known fact: BLEU is poor metric for code generation
  How do developers check the correctness of code?
- Unit tests!
- Functional correctness:
  pass@k metric: at least 1 model sample from k pass the tests
- Where to get the data?



HumanEval

- 164 handwritten programming problems

```
def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) ==>12
    solution([3, 3, 3, 3, 3]) ==>9
    solution([30, 13, 24, 321]) ==>0
    """
    return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

- a sandbox for executing generated programs

Results

	PASS@k		
	$k = 1$	$k = 10$	$k = 100$
GPT-NEO 125M	0.75%	1.88%	2.97%
GPT-NEO 1.3B	4.79%	7.47%	16.30%
GPT-NEO 2.7B	6.41%	11.27%	21.37%
GPT-J 6B	11.62%	15.74%	27.74%
TABNINE	2.58%	4.35%	7.59%
CODEX-12M	2.00%	3.62%	8.58%
CODEX-25M	3.21%	7.1%	12.89%
CODEX-42M	5.06%	8.8%	15.55%
CODEX-85M	8.22%	12.81%	22.4%
CODEX-300M	13.17%	20.37%	36.27%
CODEX-679M	16.22%	25.7%	40.95%
CODEX-2.5B	21.36%	35.42%	59.5%
CODEX-12B	28.81%	46.81%	72.31%

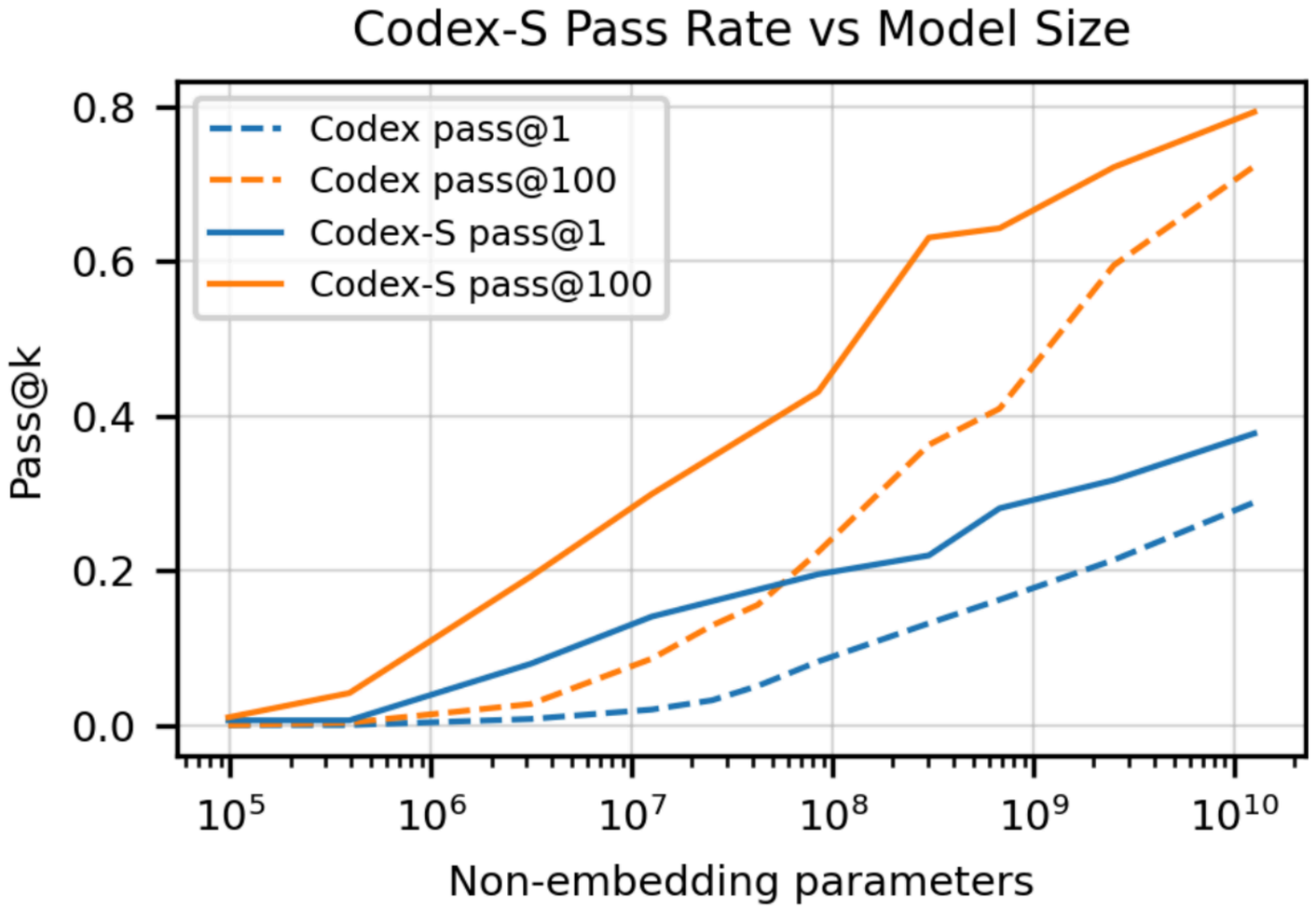
- GPT-3, trained on CommonCrawl shows poor results on code generation ~0% tasks solved
- GPT-Neo, GPT-J were trained on the Pile dataset (8% GitHub code)

Supervised fine-tuning

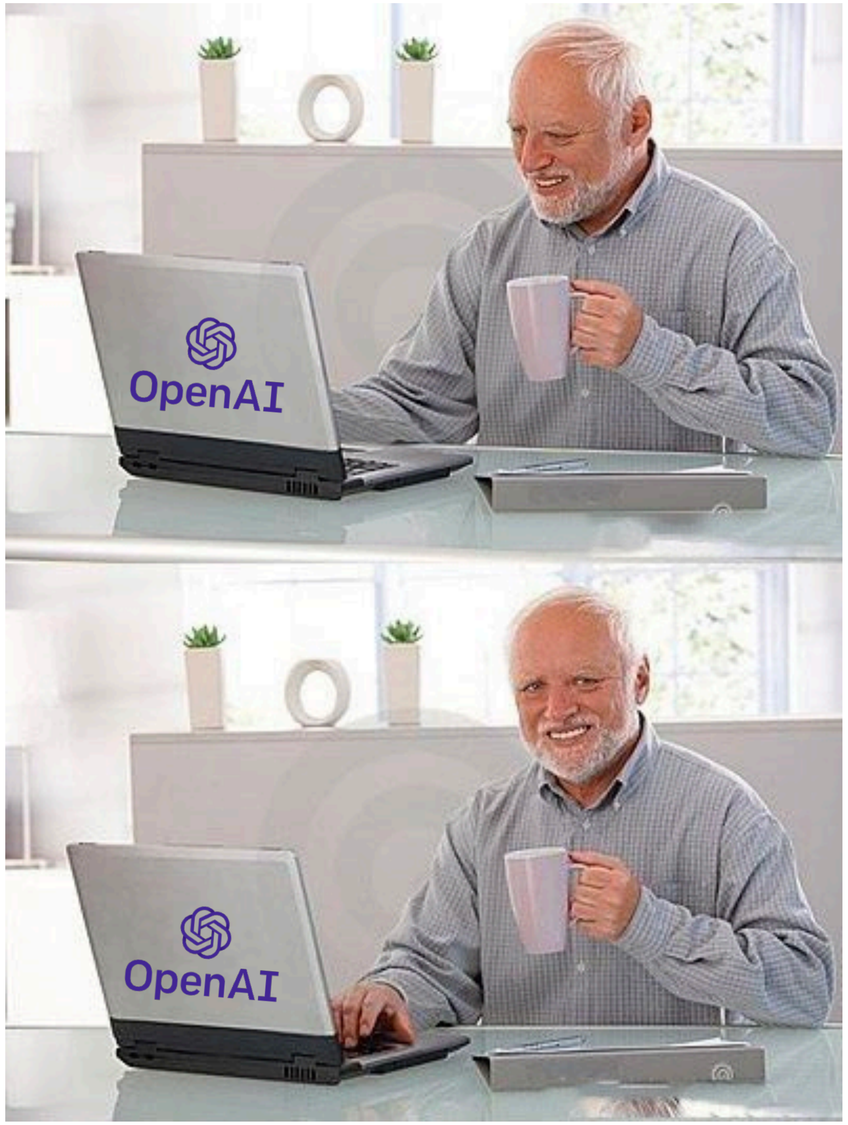
- Github data is messy: configs, data files, scripts, etc.

- Finetune on correctly implemented functions from programming contests and CI pipelines

- pass@1: 13.5% —> 37.5%



Limitations of Codex



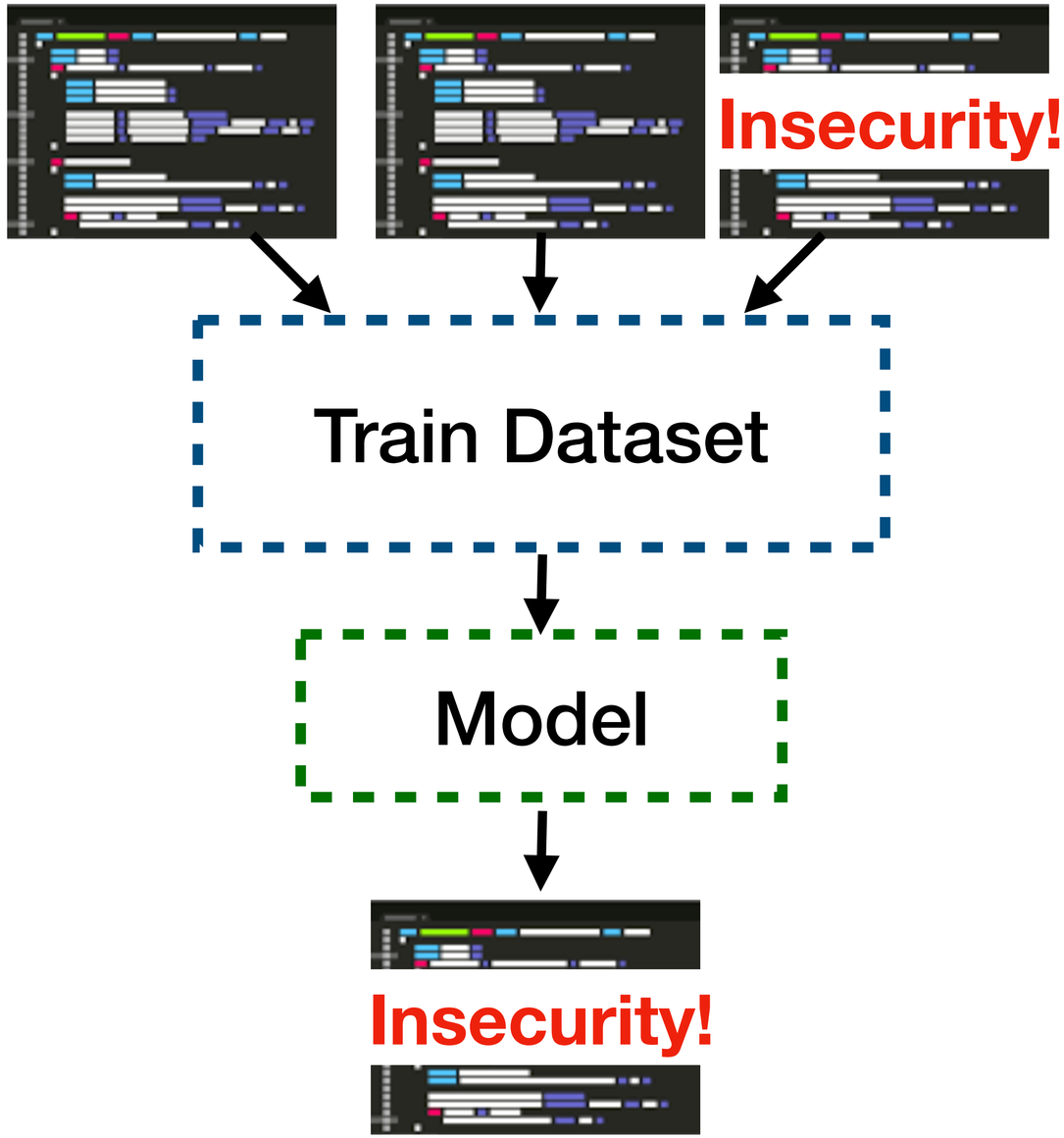
Limitations of Codex

- Subtle bugs

```
def is_prime(d):
    # check if a number d is prime
    if d == 2 or d == 3 or d == 5
    or d == 7 or d == 11:
        return True
    else:
        return False
```

Limitations of Codex

- Subtle bugs
- Insecure code



Limitations of Codex

- Subtle bugs
- Insecure code
- Syntax errors

- Some samples from the model may not compile/ cause runtime errors:

```
def f(it:
    pass
```

```
ar = []
for it in ar:
    ar.append(it)
```

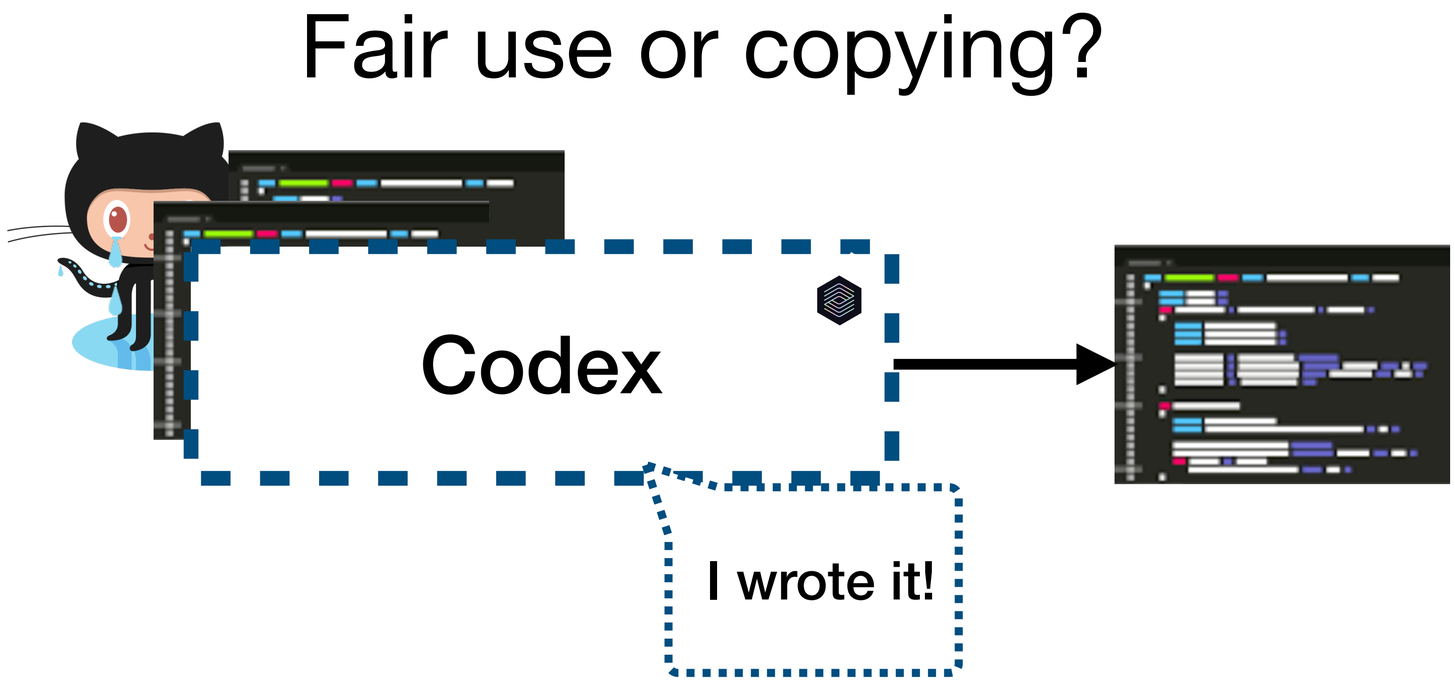
Limitations of Codex

- Subtle bugs
- Insecure code
- Syntax errors
- API diversity

```
import torch
import tensorflow as tf
import jax
```

Limitations of Codex

- Subtle bugs
- Insecure code
- Syntax errors
- API diversity
- Legal Issues



https://www.uspto.gov/sites/default/files/documents/OpenAI_RFC-84-FR-58141.pdf

Conclusions

- Functional correctness is better than BLEU for code generation tasks
- Pretraining on code data > pretraining on general data
- Finetuning on correctly implemented functions improves code generation
- GPT models should be used carefully