Методы предобучения для работы с кодом

ВШЭ ФКН, Методы предобучения без учителя

Шабалин Александр

Code specifics

- Docstrings
- Code
- Comments
- Abstract Syntax Tree (AST)
- Limited keywords
- No ambiguity in meaning

```
def _parse_memory(s):
    """
    Parse a memory string in the format supported by Java (e.g. 1g, 200m) and
    return the value in MiB

    >>> _parse_memory("256m")
    256
    >>> _parse_memory("2g")
    2048
    """
    units = {'g': 1024, 'm': 1, 't': 1 << 20, 'k': 1.0 / 1024}
    if s[-1].lower() not in units:
        raise ValueError("invalid format: " + s)
    return int(float(s[:-1]) * units[s[-1].lower()])
```

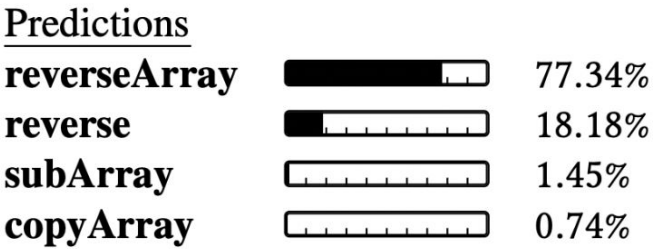
Downstream tasks

- **Code clone detection:** compare functionality of two code snippets
- **Code completion:** generate code continuation
- **Code summarization:** summarize code into NL description
- **Code generation:** generate code by docstring
- **Code refinement:** fix bugs in code
- **Code translation:** translate code from one language to another
- **Defect code detection:** predict whether a code is vulnerable to software systems or not

Code2vec, 2018 (Facebook)

Pre-training task: semantic labeling of code snippets

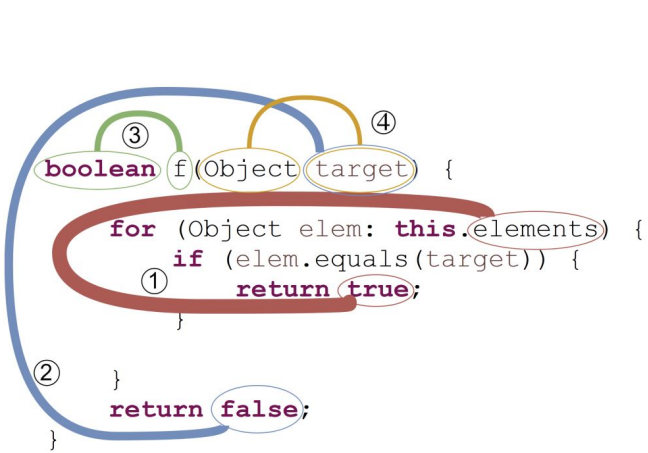
```
String[] f(final String[] array) {
    final String[] newArray = new String[array.length];
    for (int index = 0; index < array.length; index++) {
        newArray[array.length - index - 1] = array[index];
    }
    return newArray;
}
```

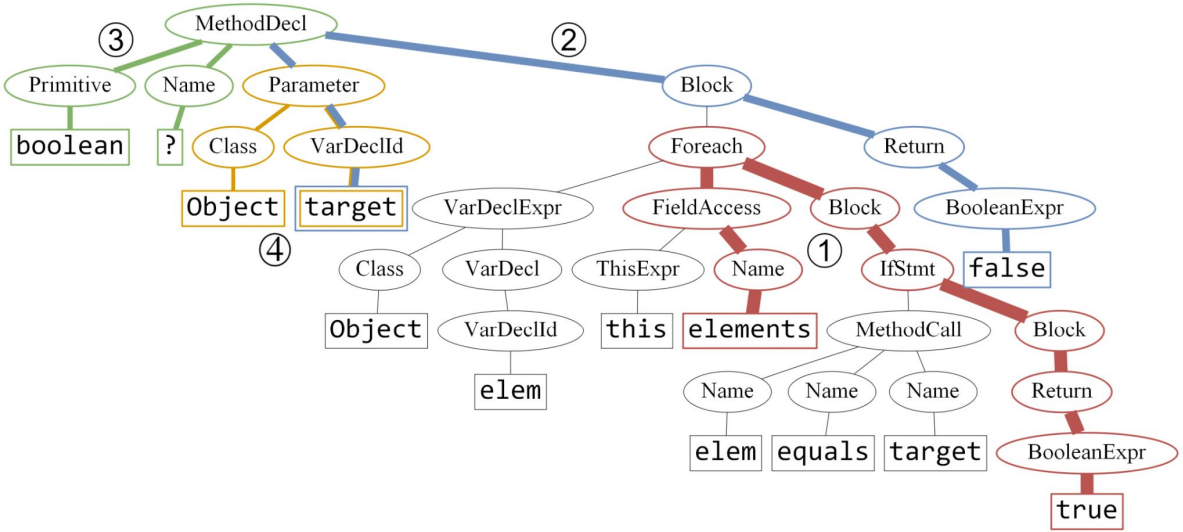


- Code snippet is represented as a bag of path-contexts
- Paths are aggregated using attention weights
- Prediction is computed with the resulting representation

https://arxiv.org/pdf/1803.09473.pdf
https://code2vec.org/

Code2vec, 2018 (Facebook)





(a)

Predictions:

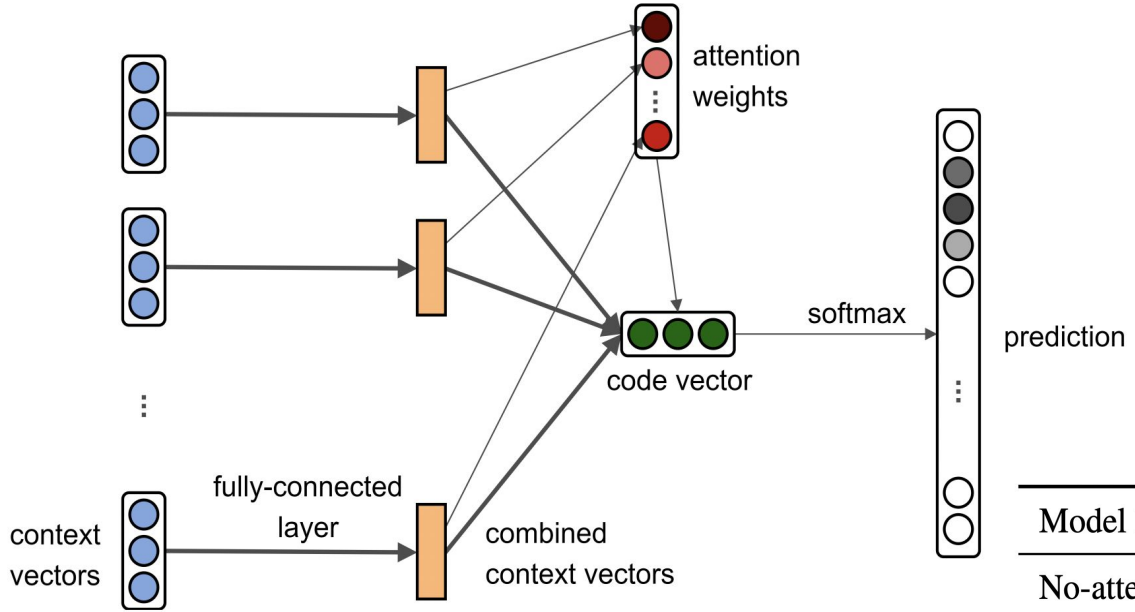
contains	90.93%
matches	3.54%
canHandle	1.15%
equals	0.87%
containsExact	0.77%

Code is splitted in the Bag of path-contexts in the form of

$$Rep(C) = \left\{ (x_s, p, x_t) \;\middle|\; \begin{array}{l} \exists (term_s, term_t) \in TPairs(C): \\ x_s = \phi(term_s) \wedge x_t = \phi(term_t) \\ \wedge\, start(p) = term_s \wedge end(p) = term_t \end{array} \right\}$$

https://arxiv.org/pdf/1803.09473.pdf
https://code2vec.org/

Code2vec, 2018 (Facebook)



Model Design	Precision	Recall	F1
No-attention	54.4	45.3	49.4
Hard-attention	42.1	35.4	38.5
Train-soft, predict-hard	52.7	45.9	49.1
Soft-attention	**63.1**	**54.4**	**58.4**
Element-wise soft-attention	**63.7**	**55.4**	**59.3**

Table 4. Comparison of model designs.

https://arxiv.org/pdf/1803.09473.pdf
https://code2vec.org/

Code2vec, 2018 (Facebook)

Model	Sampled Test Set (7454 methods) Precision	Recall	F1	Full Test Set (413915 methods) Precision	Recall	F1	prediction rate (examples / sec)
CNN+Attention [Allamanis et al. 2016]	47.3	29.4	33.9	-	-	-	0.1
LSTM+Attention [Iyer et al. 2016]	27.5	21.5	24.1	33.7	22.0	26.6	5
Paths+CRFs [Alon et al. 2018]	-	-	-	53.6	46.6	49.9	10
PathAttention (this work)	**63.3**	**56.2**	**59.5**	**63.1**	**54.4**	**58.4**	**1000**

Table 3. Evaluation comparison between our model and previous works.

Path-context input		Precision	Recall	F1
Full:	$\langle x_s, p, x_t \rangle$	63.1	54.4	58.4
Only-values:	$\langle x_s, \_\_, x_t \rangle$	44.9	37.1	40.6
No-values:	$\langle \_\_, p, \_\_ \rangle$	12.0	12.6	12.3
Value-path:	$\langle x_s, p, \_\_ \rangle$	31.5	30.1	30.7
One-value:	$\langle x_s, \_\_, \_\_ \rangle$	10.6	10.4	10.7

Table 5. Our model while hiding input components.

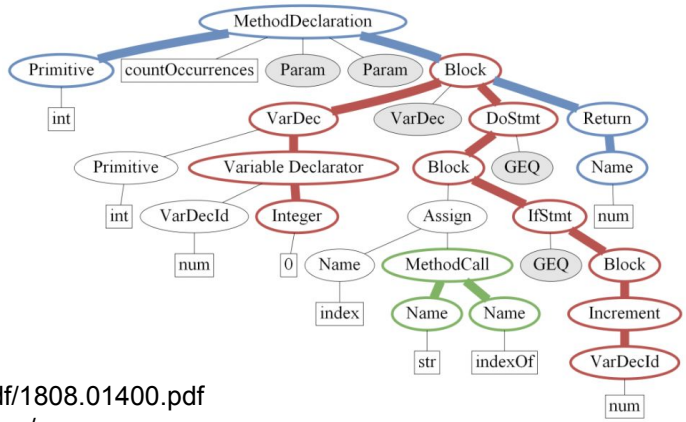
https://arxiv.org/pdf/1803.09473.pdf
https://code2vec.org/

Code2seq, 2019 (Facebook)

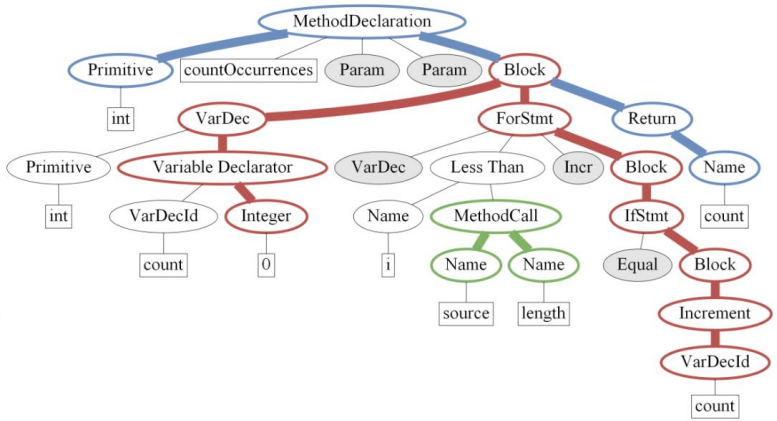
```java
int countOccurrences(String str, char ch) {
    int num = 0;
    int index = -1;
    do {
        index = str.indexOf(ch, index + 1);
        if (index >= 0) {
            num++;
        }
    } while (index >= 0);
    return num;
}
```

```java
int countOccurrences(String source, char value) {
    int count = 0;
    for (int i = 0; i < source.length(); i++) {
        if (source.charAt(i) == value) {
            count++;
        }
    }
    return count;
}
```

(a)



(b)



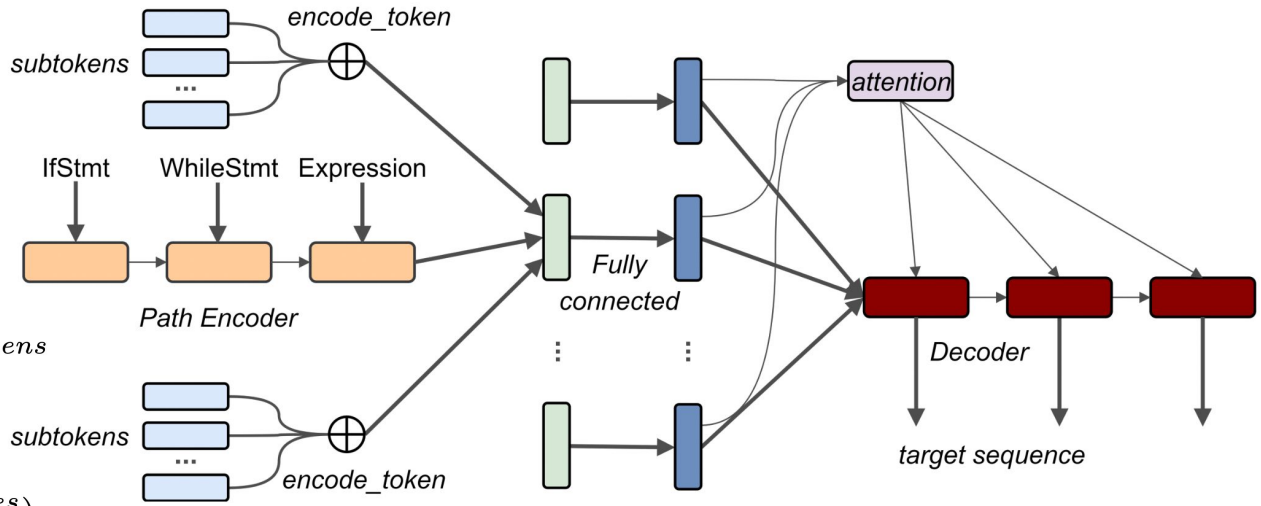
https://arxiv.org/pdf/1808.01400.pdf
https://code2seq.org/

Code2seq, 2019 (Facebook)

Algorithm
For a code snippet sample **k=200** random paths from AST

For each path:

$$encode\_token(w) = \sum_{s \in split(w)} E_s^{subtokens}$$



$$h_1, ..., h_l = LSTM(E_{v_1}^{nodes}, ..., E_{v_l}^{nodes})$$

$$encode\_path(v_1 ... v_l) = [h_l^{\rightarrow}; h_1^{\leftarrow}]$$

$$z = tanh\left(W_{in}\left[encode\_path(v_1 ... v_l); encode\_token(value(v_1)); encode\_token(value(v_l))\right]\right)$$

$$\boldsymbol{\alpha}^t = softmax\left(h_t W_a \boldsymbol{z}\right) \qquad c_t = \sum_i^n \alpha_i^t z_i$$

https://arxiv.org/pdf/1808.01400.pdf
https://code2seq.org/

Code2seq, 2019 (Facebook)

Code summarization: predict Java method's name by its body as a sequence of tokens.

Model	Java-small $700K$			Java-med $4M$			Java-large $16M$		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
ConvAttention (Allamanis et al., 2016)	50.25	24.62	33.05	60.82	26.75	37.16	60.71	27.60	37.95
Paths+CRFs (Alon et al., 2018)	8.39	5.63	6.74	32.56	20.37	25.06	32.56	20.37	25.06
code2vec (Alon et al., 2019)	18.51	18.74	18.62	38.12	28.31	32.49	48.15	38.40	42.73
2-layer BiLSTM (no token splitting)	32.40	20.40	25.03	48.37	30.29	37.25	58.02	37.73	45.73
2-layer BiLSTM	42.63	29.97	35.20	55.15	41.75	47.52	63.53	48.77	55.18
TreeLSTM (Tai et al., 2015)	40.02	31.84	35.46	53.07	41.69	46.69	60.34	48.27	53.63
Transformer (Vaswani et al., 2017)	38.13	26.70	31.41	50.11	35.01	41.22	59.13	40.58	48.13
code2seq	**50.64**	**37.40**	**43.02**	**61.24**	**47.07**	**53.23**	**64.03**	**55.02**	**59.19**
Absolute gain over BiLSTM	+8.01	+7.43	+7.82	+6.09	+5.32	+5.71	+0.50	+6.25	+4.01

https://arxiv.org/pdf/1808.01400.pdf
https://code2seq.org/

Code2seq, 2019 (Facebook)

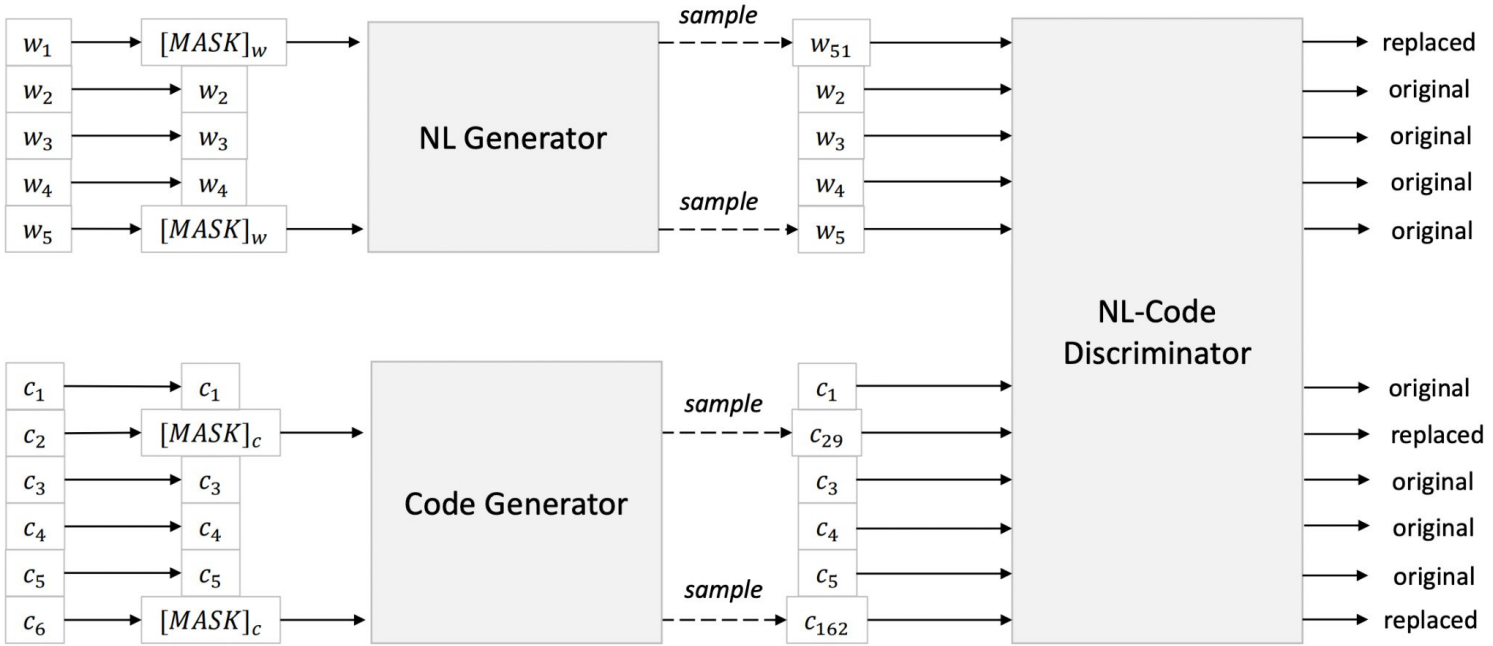
Code captioning (docstring generation): StackOverflow dataset generate a short description for a code snippet (C#).

Model	BLEU
MOSES† (Koehn et al., 2007)	11.57
IR†	13.66
SUM-NN† (Rush et al., 2015)	19.31
2-layer BiLSTM	19.78
Transformer (Vaswani et al., 2017)	19.68
TreeLSTM (Tai et al., 2015)	20.11
CodeNN† (Iyer et al., 2016)	20.53
code2seq	**23.04**

https://arxiv.org/pdf/1808.01400.pdf
https://code2seq.org/

CodeBERT, 2020 (Microsoft)

The typical input on which CodeBERT is trained on is combination of code and well-defined text comments.
MLM + RTD objectives



https://arxiv.org/pdf/2002.08155.pdf

CodeBERT, 2020 (Microsoft)

CodeSearchNet evaluation (find the most related code from a collection of candidate codes)

MODEL	RUBY	JAVASCRIPT	GO	PYTHON	JAVA	PHP	MA-AVG
NBOW	0.4285	0.4607	0.6409	0.5809	0.5140	0.4835	0.5181
CNN	0.2450	0.3523	0.6274	0.5708	0.5270	0.5294	0.4753
BIRNN	0.0835	0.1530	0.4524	0.3213	0.2865	0.2512	0.2580
SELFATT	0.3651	0.4506	0.6809	0.6922	0.5866	0.6011	0.5628
ROBERTA	0.6245	0.6060	0.8204	0.8087	0.6659	0.6576	0.6972
PT W/ CODE ONLY (INIT=S)	0.5712	0.5557	0.7929	0.7855	0.6567	0.6172	0.6632
PT W/ CODE ONLY (INIT=R)	0.6612	0.6402	0.8191	0.8438	0.7213	0.6706	0.7260
CODEBERT (MLM, INIT=S)	0.5695	0.6029	0.8304	0.8261	0.7142	0.6556	0.6998
CODEBERT (MLM, INIT=R)	0.6898	0.6997	0.8383	0.8647	0.7476	0.6893	0.7549
CODEBERT (RTD, INIT=R)	0.6414	0.6512	0.8285	0.8263	0.7150	0.6774	0.7233
CODEBERT (MLM+RTD, INIT=R)	**0.6926**	**0.7059**	**0.8400**	**0.8685**	**0.7484**	**0.7062**	**0.7603**

https://arxiv.org/pdf/2002.08155.pdf

CodeBERT, 2020 (Microsoft)

masked NL token

"Transforms a vector np.arange(-N, M, dx) to np.arange(min (|vec|), max(N,M),dx)]"

```
def vec_to_halfvec(vec):

    d = vec[1:] - vec[:-1]
    if ((d/d.mean()).std() > 1e-14) or (d.mean() < 0):
        raise ValueError('vec must be np.arange() in increasing order')
    dx = d.mean()
    lowest = np.abs(vec).min()
    highest = np.abs(vec).max()
    return np.arange(lowest, highest + 0.1*dx, dx).astype(vec.dtype)
```

masked PL token

		max	*min*	*less*	*greater*
NL	Roberta	96.24%	3.73%	0.02%	0.01%
	CodeBERT (MLM)	39.38%	60.60%	0.02%	0.0003%
PL	Roberta	95.85%	4.15%	-	-
	CodeBERT (MLM)	0.001%	99.999%	-	-

https://arxiv.org/pdf/2002.08155.pdf

CodeBERT, 2020 (Microsoft)

	RUBY	JAVASCRIPT	GO	PYTHON	JAVA	PHP	ALL
NUMBER OF DATAPOINTS FOR PROBING							
PL (2 CHOICES)	38	272	152	1,264	482	407	2,615
NL (4 CHOICES)	20	65	159	216	323	73	856
PL PROBING							
ROBERTA	73.68	63.97	72.37	59.18	59.96	69.78	62.45
PRE-TRAIN W/ CODE ONLY	71.05	77.94	89.47	70.41	70.12	82.31	74.11
CODEBERT (MLM)	**86.84**	**86.40**	**90.79**	**82.20**	**90.46**	**88.21**	**85.66**
PL PROBING WITH PRECEDING CONTEXT ONLY							
ROBERTA	**73.68**	**53.31**	51.32	55.14	42.32	52.58	52.24
PRE-TRAIN W/ CODE ONLY	63.16	48.53	**61.84**	56.25	**58.51**	58.97	56.71
CODEBERT (MLM)	65.79	50.74	59.21	**62.03**	54.98	**59.95**	**59.12**
NL PROBING							
ROBERTA	50.00	72.31	54.72	61.57	61.61	65.75	61.21
PRE-TRAIN W/ CODE ONLY	55.00	67.69	60.38	68.06	65.02	68.49	65.19
CODEBERT (MLM)	**65.00**	**89.23**	**66.67**	**76.85**	**73.37**	**79.45**	**74.53**

Table 3: Statistics of the data for NL-PL probing and the performance of different pre-trained models. Accuracies (%) are reported. Best results in each group are in bold.

https://arxiv.org/pdf/2002.08155.pdf

CodeBERT, 2020 (Microsoft)

Documentation generation: freeze pretrained encoder and train decoder upon it.

MODEL	RUBY	JAVASCRIPT	GO	PYTHON	JAVA	PHP	OVERALL
SEQ2SEQ	9.64	10.21	13.98	15.93	15.09	21.08	14.32
TRANSFORMER	11.18	11.59	16.38	15.81	16.26	22.12	15.56
ROBERTA	11.17	11.90	17.72	18.14	16.47	24.02	16.57
PRE-TRAIN W/ CODE ONLY	11.91	13.99	17.78	18.58	17.50	24.34	17.35
CODEBERT (RTD)	11.42	13.27	17.53	18.29	17.35	24.10	17.00
CODEBERT (MLM)	11.57	14.41	17.78	18.77	17.38	24.85	17.46
CODEBERT (RTD+MLM)	**12.16**	**14.90**	**18.07**	**19.06**	**17.65**	**25.16**	**17.83**

Table 4: Results on Code-to-Documentation generation, evaluated with smoothed BLEU-4 score.

https://arxiv.org/pdf/2002.08155.pdf

CodeBERT, 2020 (Microsoft)

Zero-shot documentation generation (unseen PL)

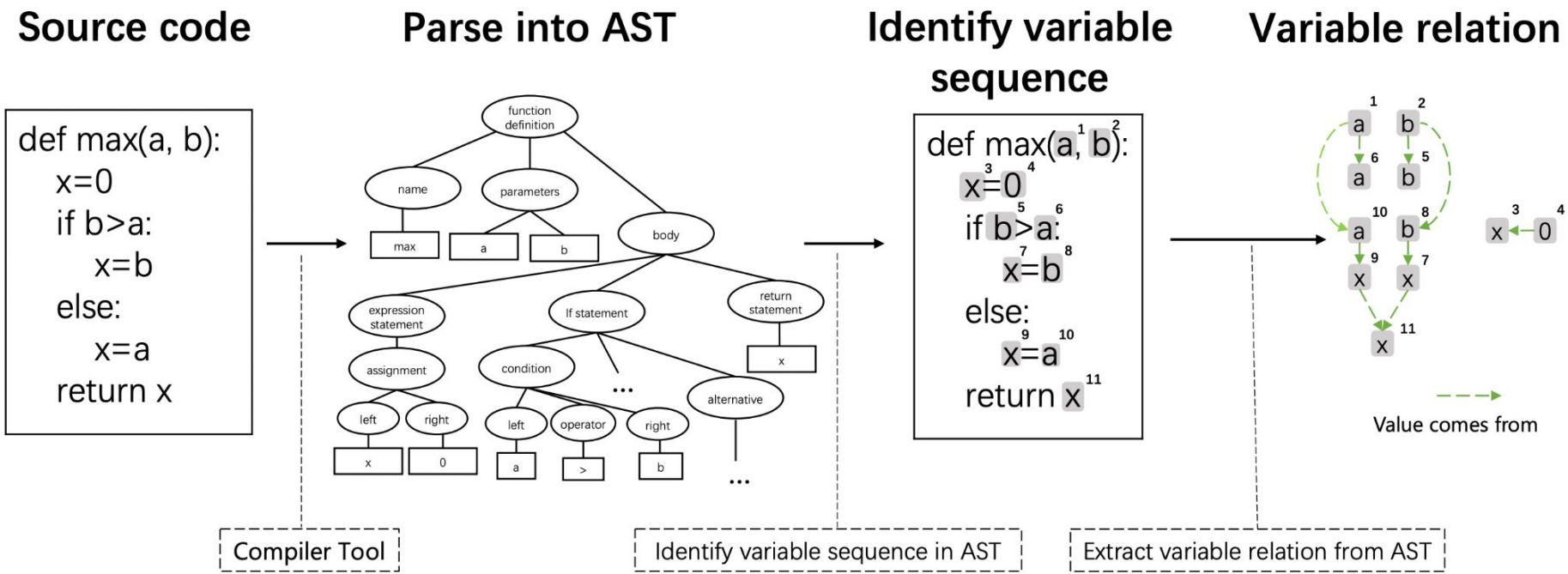
MODEL	BLEU
MOSES (KOEHN ET AL., 2007)	11.57
IR	13.66
SUM-NN (RUSH ET AL., 2015)	19.31
2-LAYER BILSTM	19.78
TRANSFORMER (VASWANI ET AL., 2017)	19.68
TREELSTM (TAI ET AL., 2015)	20.11
CODENN (IYER ET AL., 2016)	20.53
CODE2SEQ (ALON ET AL., 2019)	**23.04**
ROBERTA	19.81
PRE-TRAIN W/ CODE ONLY	20.65
CODEBERT (RTD)	22.14
CODEBERT (MLM)	22.32
CODEBERT (MLM+RTD)	**22.36**

Table 5: Code-to-NL generation on C# language.

https://arxiv.org/pdf/2002.08155.pdf

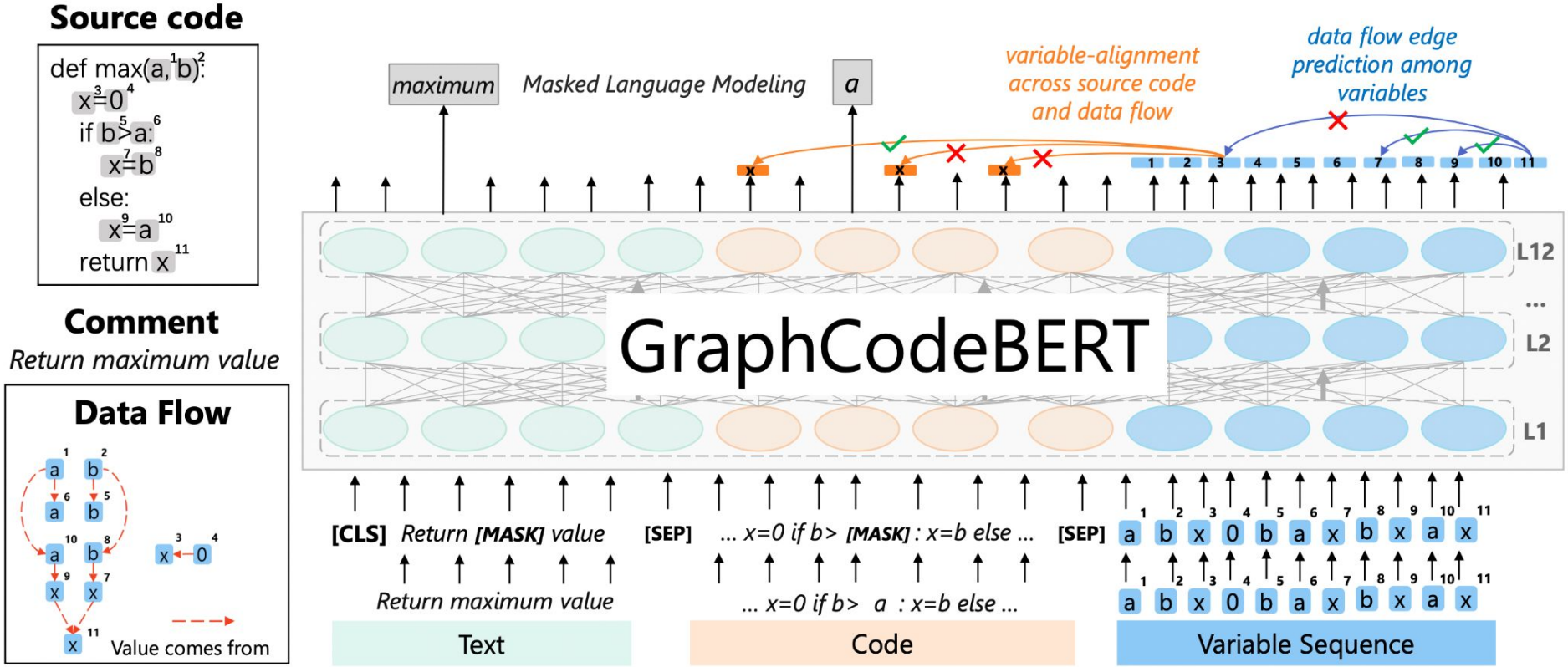
GraphCodeBERT, 2021 (Microsoft)

The model is based on the **Data Flow** instead of AST.
Data Flow is a graph where nodes represent *variables* and edges represent *relation* between them.



https://arxiv.org/pdf/2009.08366.pdf

GraphCodeBERT, 2021 (Microsoft)



https://arxiv.org/pdf/2009.08366.pdf

GraphCodeBERT, 2021 (Microsoft)

Graph-guided masked attention

$$Q_i = H^{n-1}W_i^Q,\ K_i = H^{n-1}W_i^K,\ V_i = H^{n-1}W_i^V$$

$$head_i = \mathrm{softmax}(\frac{\mathrm{Q_i K_i^T}}{\sqrt{\mathrm{d_k}}} + \mathrm{M})\mathrm{V_i} \qquad M \in \mathbb{R}^{|X|\times|X|}$$

$$\hat{G}^n = [head_1; ...; head_u]W_n^O$$

$$M_{ij} = \begin{cases} 0 & \text{if } q_i \in \{[CLS], [SEP]\} \text{ or } q_i, k_j \in W \cup C \text{ or } \langle q_i, k_j \rangle \in E \cup E' \\ -\infty & \text{otherwise} \end{cases}$$

text tokens

code tokens

data flow edges

mapping from code to data flow

https://arxiv.org/pdf/2009.08366.pdf

GraphCodeBERT, 2021 (Microsoft)

Pre-training tasks

- Default **MLM** task for *code* and *comment* parts.
- **Edge Prediction:** predict whether there is an edge between two variables in the Data Flow.

$$loss_{EdgePred} = -\sum_{e_{ij} \in E_c} [\delta(e_{ij} \in E_{mask}) log p_{e_{ij}} + (1 - \delta(e_{ij} \in E_{mask})) log(1 - p_{e_{ij}})]$$

set of balanced positive and negative examples

set of masked edges

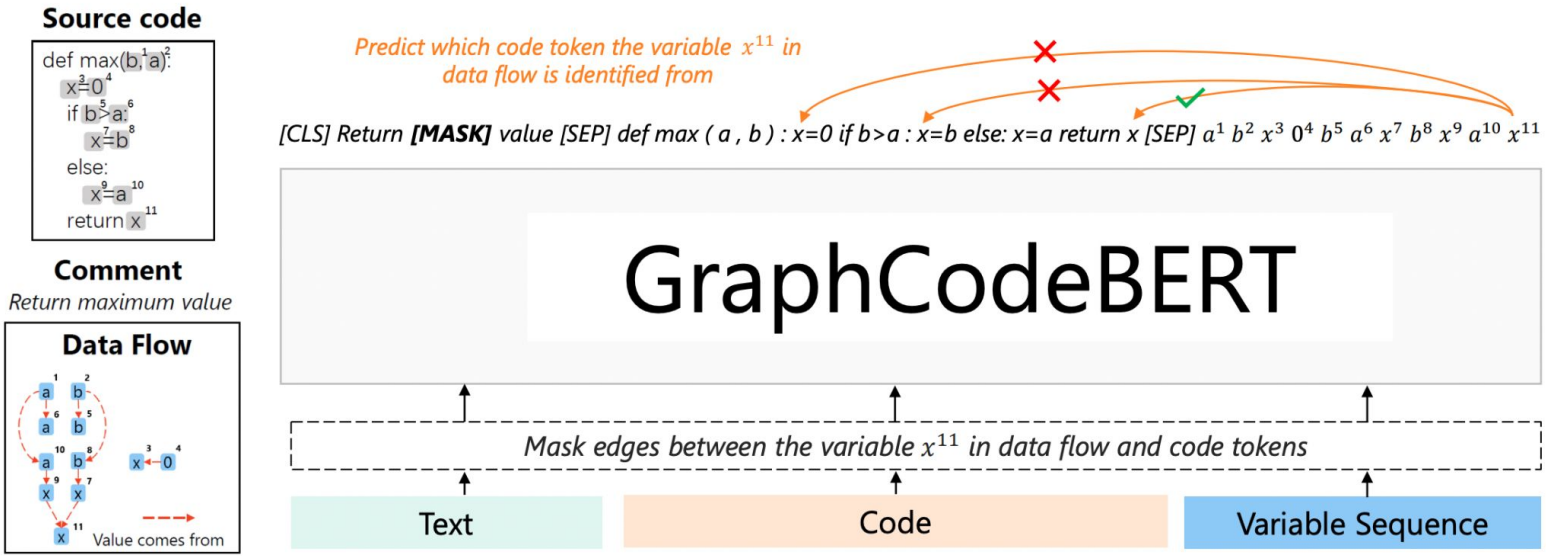
Probability is calculated by *dot product* using encodings of corresponding nodes.

https://arxiv.org/pdf/2009.08366.pdf

GraphCodeBERT, 2021 (Microsoft)

Pre-training tasks

- **Node Alignment:** predict whether a variable in code corresponds to a node in the Data Flow.



$$loss_{NodeAlign} = -\sum_{e_{ij} \in E'_c} [\delta(e_{ij} \in E'_{mask}) log p_{e_{ij}} + (1 - \delta(e_{ij} \in E'_{mask})) log(1 - p_{e_{ij}})]$$

https://arxiv.org/pdf/2009.08366.pdf

GraphCodeBERT, 2021 (Microsoft)

CodeSearchNet evaluation (find the most related code from a collection of candidate codes) (MRR)

model	Ruby	Javascript	Go	Python	Java	Php	Overall
NBow	0.162	0.157	0.330	0.161	0.171	0.152	0.189
CNN	0.276	0.224	0.680	0.242	0.263	0.260	0.324
BiRNN	0.213	0.193	0.688	0.290	0.304	0.338	0.338
selfAtt	0.275	0.287	0.723	0.398	0.404	0.426	0.419
RoBERTa	0.587	0.517	0.850	0.587	0.599	0.560	0.617
RoBERTa (code)	0.628	0.562	0.859	0.610	0.620	0.579	0.643
CodeBERT	0.679	0.620	0.882	0.672	0.676	0.628	0.693
GraphCodeBERT	**0.703**	**0.644**	**0.897**	**0.692**	**0.691**	**0.649**	**0.713**

https://arxiv.org/pdf/2009.08366.pdf

GraphCodeBERT, 2021 (Microsoft)

Code Clone Detection
(BigCloneBench)

Model	Precision	Recall	F1
Deckard	0.93	0.02	0.03
RtvNN	0.95	0.01	0.01
CDLH	0.92	0.74	0.82
ASTNN	0.92	0.94	0.93
FA-AST-GMN	**0.96**	0.94	0.95
RoBERTa (code)	0.949	0.922	0.935
CodeBERT	0.947	0.934	0.941
GraphCodeBERT	0.948	**0.952**	**0.950**

Code translation

Method	Java→C#		C#→Java	
	BLEU	Acc	BLEU	Acc
Naive	18.54	0.0	18.69	0.0
PBSMT	43.53	12.5	40.06	16.1
Transformer	55.84	33.0	50.47	37.9
RoBERTa (code)	77.46	56.1	71.99	57.9
CodeBERT	79.92	59.0	72.14	58.8
GraphCodeBERT	**80.58**	**59.4**	**72.64**	**58.8**

https://arxiv.org/pdf/2009.08366.pdf

GraphCodeBERT, 2021 (Microsoft)

Code refinement
(fix bugs in code)

Method	small		medium	
	BLEU	Acc	BLEU	Acc
Naive	78.06	0.0	90.91	0.0
LSTM	76.76	10.0	72.08	2.5
Transformer	77.21	14.7	89.25	3.7
RoBERTa (code)	77.30	15.9	90.07	4.1
CodeBERT	77.42	16.4	91.07	5.2
GraphCodeBERT	**80.02**	**17.3**	**91.31**	**9.1**

https://arxiv.org/pdf/2009.08366.pdf

GraphCodeBERT, 2021 (Microsoft)

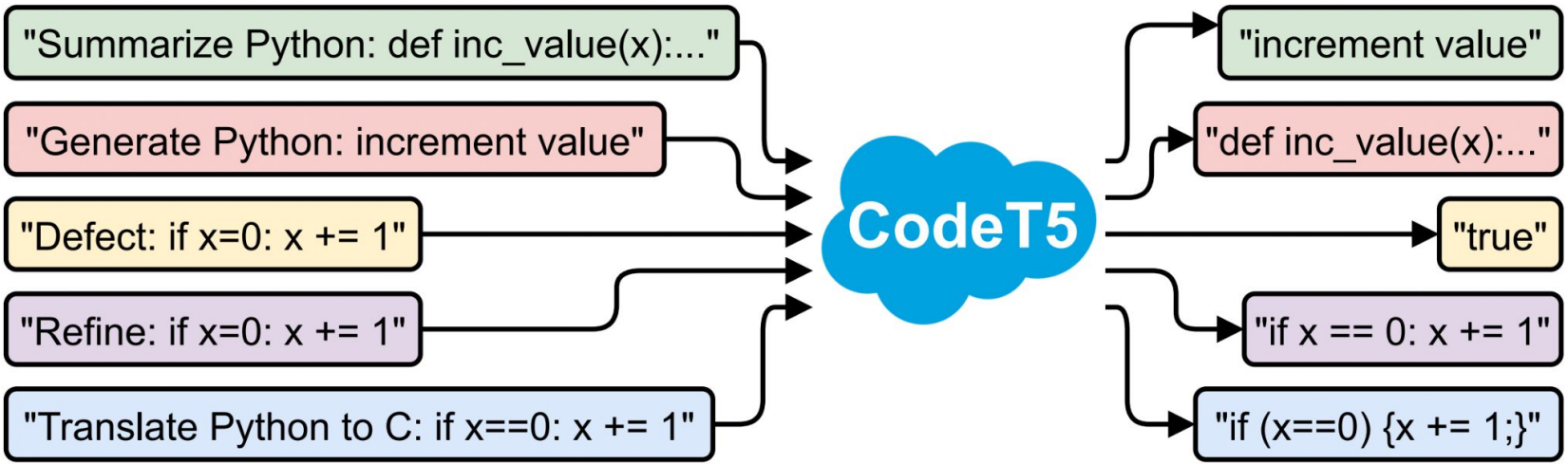
Code refinement
(fix bugs in code)

Method	small		medium	
	BLEU	Acc	BLEU	Acc
Naive	78.06	0.0	90.91	0.0
LSTM	76.76	10.0	72.08	2.5
Transformer	77.21	14.7	89.25	3.7
RoBERTa (code)	77.30	15.9	90.07	4.1
CodeBERT	77.42	16.4	91.07	5.2
GraphCodeBERT	**80.02**	**17.3**	**91.31**	**9.1**

Methods	Ruby	Javascript	Go	Python	Java	Php	Overall
GraphCodeBERT	**0.703**	**0.644**	**0.897**	**0.692**	**0.691**	**0.649**	**0.713**
-w/o EdgePred	0.701	0.632	0.894	0.687	0.688	0.640	0.707
-w/o NodeAlign	0.685	0.635	0.887	0.682	0.690	0.640	0.703
-w/o Data Flow	0.679	0.620	0.882	0.672	0.676	0.628	0.693

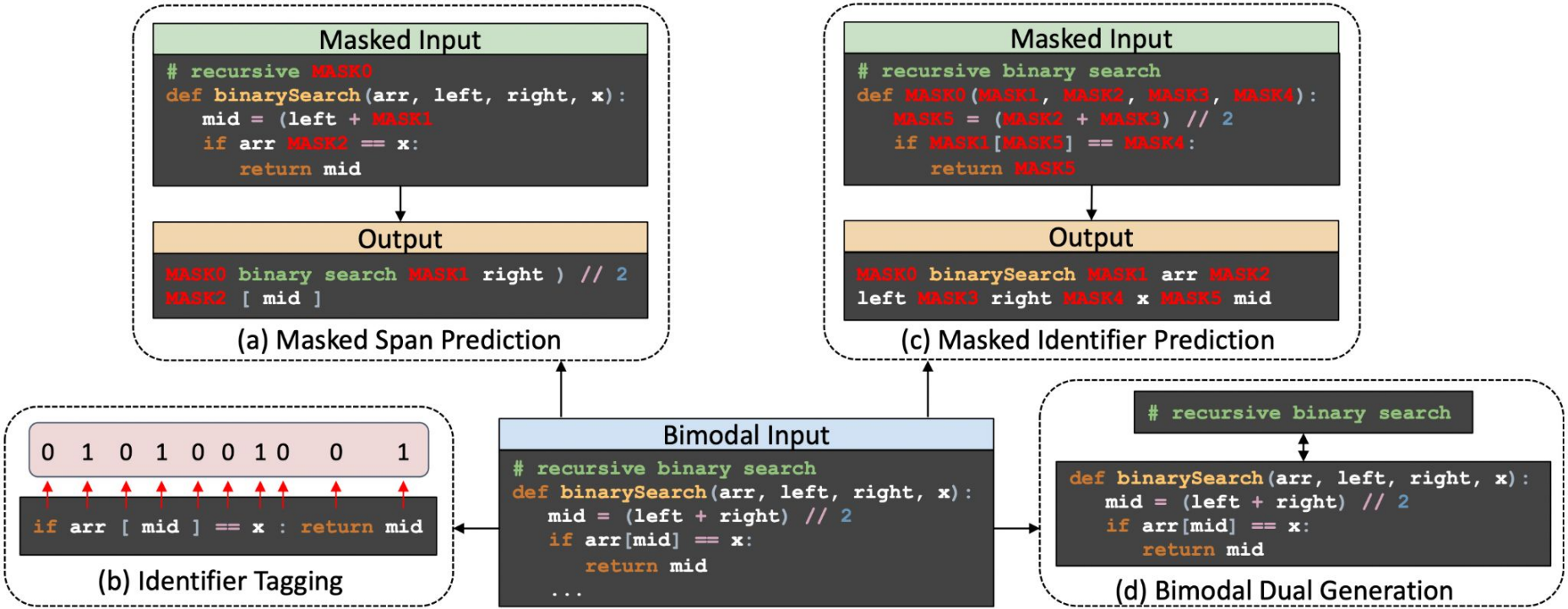
https://arxiv.org/pdf/2009.08366.pdf

CodeT5, 2021 (Salesforce)



https://arxiv.org/pdf/2109.00859.pdf

CodeT5, 2021 (Salesforce)



https://arxiv.org/pdf/2109.00859.pdf

CodeT5, 2021 (Salesforce)

Methods	Ruby	JavaScript	Go	Python	Java	PHP	Overall
RoBERTa	11.17	11.90	17.72	18.14	16.47	24.02	16.57
CodeBERT	12.16	14.90	18.07	19.06	17.65	25.16	17.83
DOBF	-	-	-	18.24	19.05	-	-
PLBART	14.11	15.56	18.91	19.30	18.45	23.58	18.32
CodeT5-small	14.87	15.32	19.25	20.04	19.92	25.46	19.14
+dual-gen	15.30	15.61	19.74	19.94	19.78	26.48	19.48
+multi-task	15.50	15.52	19.62	20.10	19.59	25.69	19.37
CodeT5-base	15.24	16.16	19.56	20.01	20.31	26.03	19.55
+dual-gen	**15.73**	16.00	19.71	20.11	20.41	**26.53**	19.75
+multi-task	15.69	**16.24**	**19.76**	**20.36**	**20.46**	26.09	**19.77**

Table 2: Smoothed BLEU-4 scores on the code summarization task. The "Overall" column shows the average scores over six PLs. Best results are in bold.

https://arxiv.org/pdf/2109.00859.pdf

CodeT5, 2021 (Salesforce)

Methods	EM	BLEU	CodeBLEU
GPT-2	17.35	25.37	29.69
CodeGPT-2	18.25	28.69	32.71
CodeGPT-adapted	20.10	32.79	35.98
PLBART	18.75	36.69	38.52
CodeT5-small	21.55	38.13	41.39
+dual-gen	19.95	39.02	42.21
+multi-task	20.15	35.89	38.83
CodeT5-base	22.30	40.73	43.20
+dual-gen	**22.70**	**41.48**	**44.10**
+multi-task	21.15	37.54	40.01

Table 3: Results on the code generation task. EM denotes the exact match.

https://arxiv.org/pdf/2109.00859.pdf

CodeT5, 2021 (Salesforce)

Methods	EM	BLEU	CodeBLEU
GPT-2	17.35	25.37	29.69
CodeGPT-2	18.25	28.69	32.71
CodeGPT-adapted	20.10	32.79	35.98
PLBART	18.75	36.69	38.52
CodeT5-small	21.55	38.13	41.39
+dual-gen	19.95	39.02	42.21
+multi-task	20.15	35.89	38.83
CodeT5-base	22.30	40.73	43.20
+dual-gen	**22.70**	**41.48**	**44.10**
+multi-task	21.15	37.54	40.01

Table 3: Results on the code generation task. EM denotes the exact match.

Methods	Defect Accuracy	Clone F1
RoBERTa	61.05	94.9
CodeBERT	62.08	96.5
DOBF	-	96.5
GraphCodeBERT	-	97.1
PLBART	63.18	**97.2**
CodeT5-small	63.40	97.1
+dual-gen	63.47	97.0
+multi-task	63.58	-
CodeT5-base	**65.78**	**97.2**
+dual-gen	62.88	97.0
+multi-task	65.02	-

Table 5: Results on the code defect detection and clone detection tasks.

https://arxiv.org/pdf/2109.00859.pdf

CodeT5, 2021 (Salesforce)

Methods	Java to C#		C# to Java		Refine Small		Refine Medium	
	BLEU	EM	BLEU	EM	BLEU	EM	BLEU	EM
Naive Copy	18.54	0	18.69	0	78.06	0	90.91	0
RoBERTa (code)	77.46	56.10	71.99	57.90	77.30	15.90	90.07	4.10
CodeBERT	79.92	59.00	72.14	58.80	77.42	16.40	91.07	5.20
GraphCodeBERT	80.58	59.40	72.64	58.80	**80.02**	17.30	**91.31**	9.10
PLBART	83.02	64.60	78.35	65.00	77.02	19.21	88.50	8.98
CodeT5-small	82.98	64.10	79.10	65.60	76.23	19.06	89.20	10.92
+dual-gen	82.24	63.20	78.10	63.40	77.03	17.50	88.99	10.28
+multi-task	83.49	64.30	78.56	65.40	77.03	20.94	87.51	11.11
CodeT5-base	**84.03**	**65.90**	**79.87**	**66.90**	77.43	21.61	87.64	13.96
+dual-gen	81.84	62.00	77.83	63.20	77.66	19.43	90.43	11.69
+multi-task	82.31	63.40	78.01	64.00	78.06	**22.59**	88.90	**14.18**

Table 4: BLEU-4 scores and exact match (EM) accuracies for code translation (Java to C# and C# to Java) and code refinement (small and medium) tasks.

https://arxiv.org/pdf/2109.00859.pdf

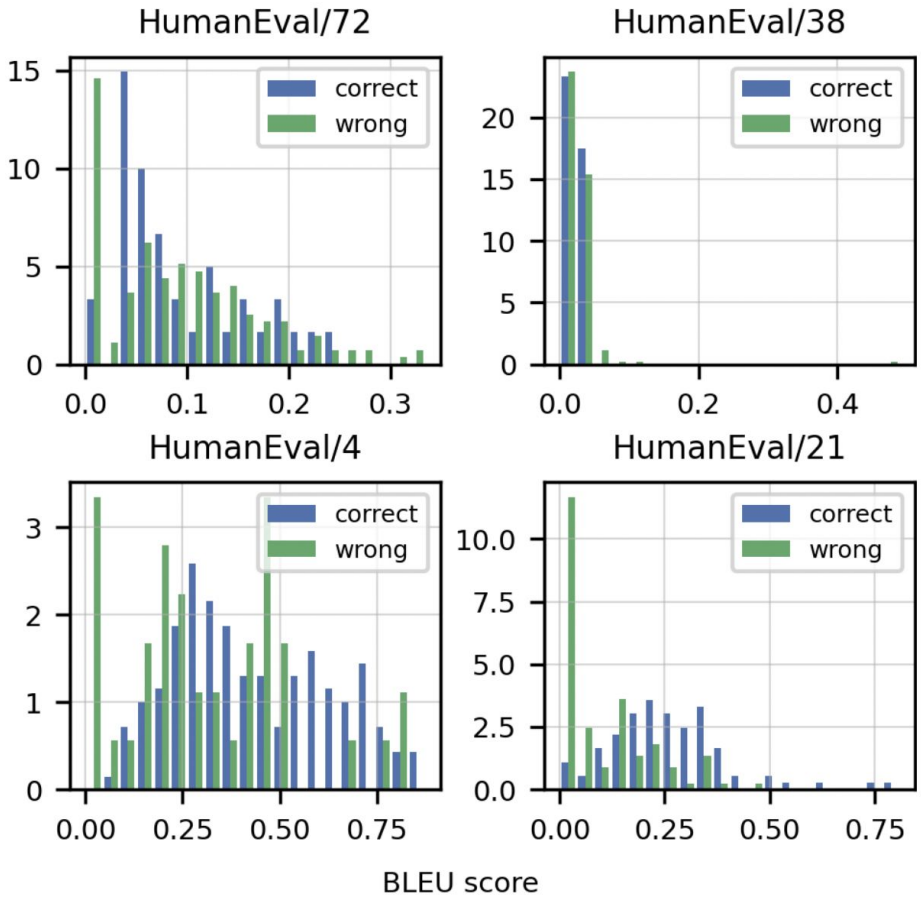
Codex, 2021 (OpenAI)

a.k.a. GitHub Copilot

- GPT-3 as a backbone model.
- Focus on code generation task.
- Use HumanEval dataset (write function by header and docstring).
  164 hand-written problems with **unit tests**.

- **Important!** pass@k metric for evaluation (the problem is solved if any of **k** generated solution passes all tests).

  Alternatives:
  - Exact match
  - BLEU



https://arxiv.org/pdf/2107.03374.pdf, https://openai.com/blog/openai-codex

Codex, 2021 (OpenAI)

a.k.a. GitHub Copilot

Training

Data: 179 GB of Python files (159 GB after filtering)

! Fine-tuning from GPT-3 gave **no improvement** vs training from scratch

https://arxiv.org/pdf/2107.03374.pdf, https://openai.com/blog/openai-codex

Codex, 2021 (OpenAI)

a.k.a. GitHub Copilot

Training

Data: 179 GB of Python files (159 GB after filtering)

! Fine-tuning from GPT-3 gave **no improvement** vs training from scratch

Tokenization: encoding whitespaces of different lengths with different tokens lowered the sequence length by 30%.

Most functions from GitHub are **not self-contained**, while samples from HumanEval are. Therefore, **need to fine-tune** model on this domain.

- Competitive programming websites
- Repositories with continuous integration

https://arxiv.org/pdf/2107.03374.pdf, https://openai.com/blog/openai-codex

Codex, 2021 (OpenAI)

a.k.a. GitHub Copilot

Training

Data: 179 GB of Python files (159 GB after filtering)

! Fine-tuning from GPT-3 gave **no improvement** vs training from scratch

Tokenization: encoding whitespaces of different lengths with different tokens lowered the sequence length by 30%.

Most functions from GitHub are **not self-contained**, while samples from HumanEval are. Therefore, **need to fine-tune** model on this domain.

- Competitive programming websites
- Repositories with continuous integration

	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%

https://arxiv.org/pdf/2107.03374.pdf, https://openai.com/blog/openai-codex