Machine Learning for Software Engineering

AI in SE Team

24 November 2021

ИСП РАН

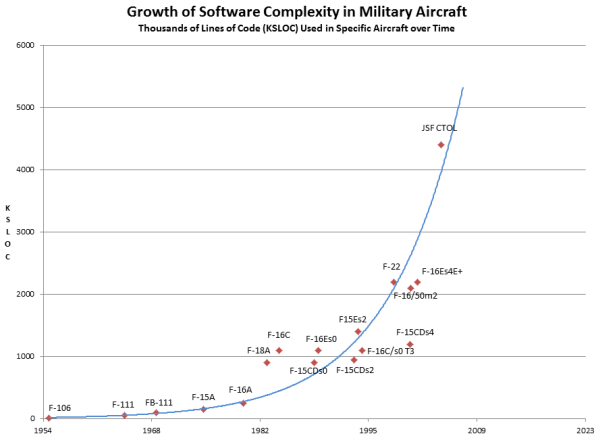
Agenda

1. Software engineering: context and challenges
2. Overview of problems
3. AI for source code analysis and comprehension
   - Source code representations
   - Notable approaches in details
4. Promising future work and useful materials

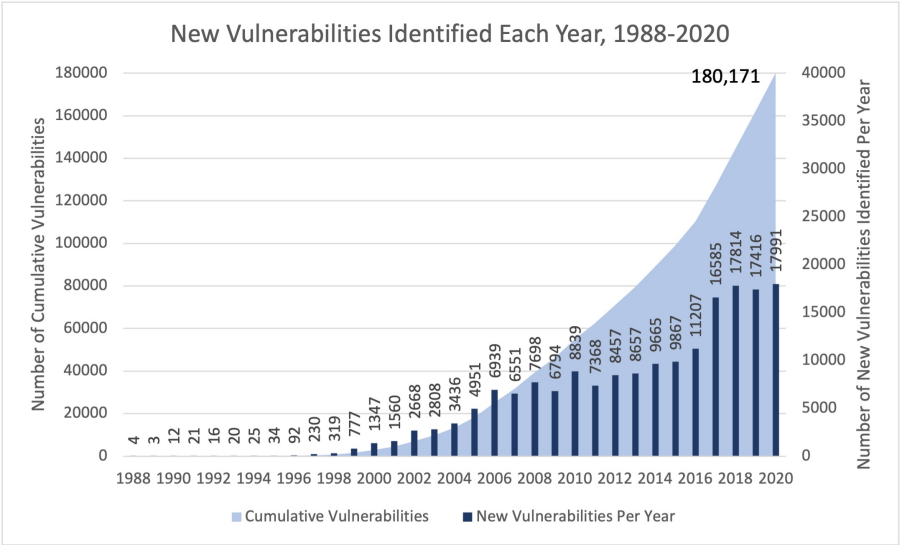
Software complexity growth

ИСП РАН

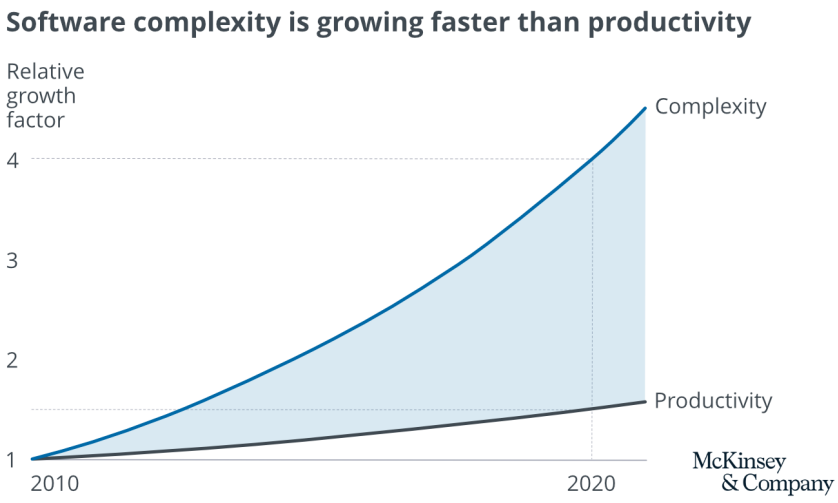
- Software "is eating"the world
- Increased size and complexity
- Increased number of vulnerabilities
- Shortage of talents
- Productivity gap



Software vulnerabilities growth



Productivity gap



Traditional methods and tools

Quality assurance:

- Manual testing and hand-written tests (e.g. unit, integration)
- Manual code review
- Static analysis for source code and binaries (e.g. memory leak checkers)
- Dynamic analysis of software (e.g. fuzzing)

Productivity tools:

- Version control systems
- IDEs and LSP servers
- Continuous integration (CI) systems
- Container and cloud systems

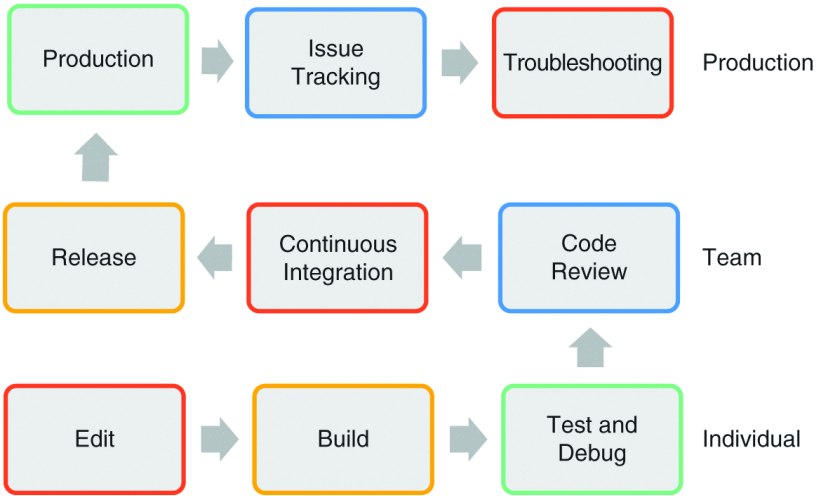
Problems with traditional tools

- Precise but expensive
- Certain tools are based on hand-crafted rules
- Inefficient usage of large amounts of available data
- Inability to detect certain issues and routine tasks that can't be automated
- Inefficient usage of system and human resources

Key takeaway

There is a need for complementary tools to reduce cognitive overload and to optimize software engeneering activities.

Software engineering activities



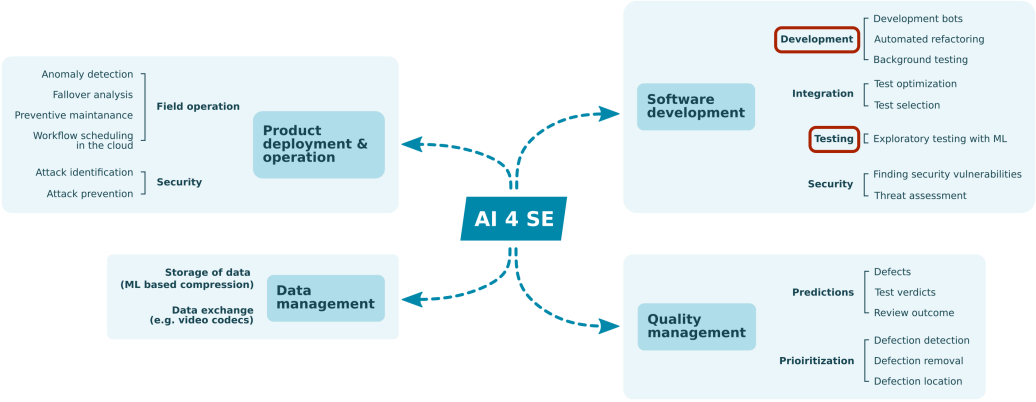
AI and data-driven approaches for the rescue

Large amounts of valuable data from different sources:

- Source code is the data itself (e.g. 200 million repositories on Github)
- Development history (version control, code review)
- User interaction data, failures and monitoring data from production
- Issues from bug trackers and question-answering platforms (e.g. SO)
- Data from programming contests

We can apply ML and data driven approaches to optimize SE activities and solve downstream tasks

Scope of AI4SE



Source code analysis and comprehension

Actively researched areas and tasks

Widespread:

- Source code summarization (commit text generation, name suggestion)
- Defect, code smell and tecnhical debt detection
- Semantic code search (similar code; by natural language query)
- Edit/fix pattern discovery and suggestion
- Neural program synthesis, transformation and repair

More rare:

- Refactoring detection and suggestion
- Intelligent source code autocompletion
- Bug, developer and reiviewer triaging

Companies, research groups and tools

ИСП РАН

Research and development:

- Microsoft (Deep Program Understanding Research Group, CoPilot)
- Facebook AI Research (Aroma, GetaFix, CompilerGym)
- Google Research (CuBERT, program synthesis)
- JetBrains (Machine Learning Methods in Software Engineering)

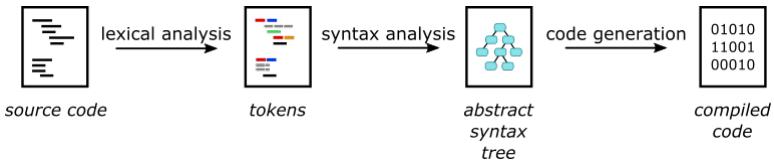
Production tools:

- **Kite** and **TabNine** for intelligent autocompletion
- **DeepCode** and **Embold** to find and fix vulnerabilities in code
- **Sourcery** and IBM's **Mono2Micro** for AI-assisted refactoring

Source code representations

ИСП РАН

- Source code and change metrics
- Source code token sequences
- Abstract Syntax Tree (AST)
- Edit sequences
- Lower-level intermediate representation (e.g. Control and Data Flow graphs, LLVM IR)



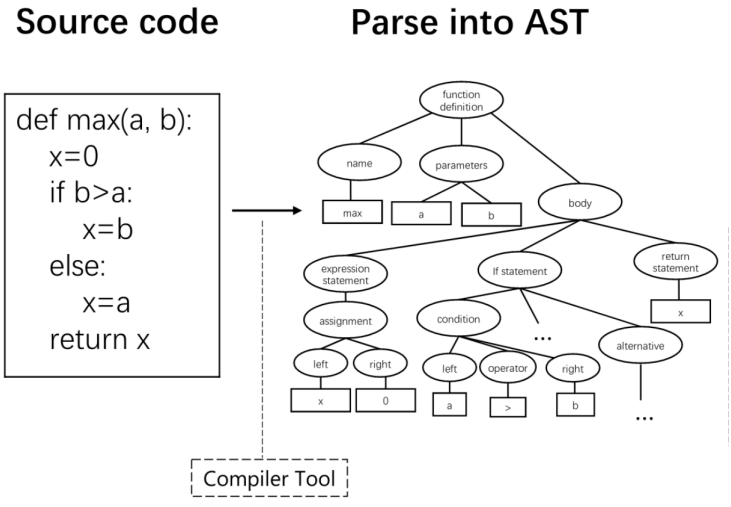
Source code and change metrics

ИСП РАН

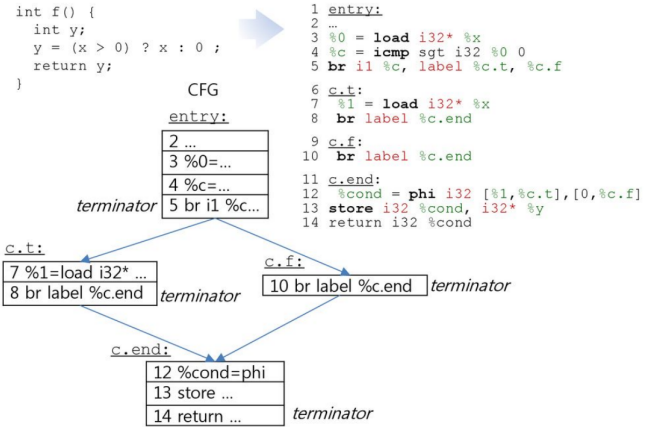
- Derived after the code or edit is processed
- Can be computed on different abstraction levels (e.g. method)
- Sufficient for certain tasks (e.g. commit filtering or code smell detection)

Metric Name	Description (applies to method level)
fanIN	Number of methods that reference a given method
fanOUT	Number of methods referenced by a given method
localVar	Number of local variables in the body of a method
parameters	Number of parameters in the declaration
commentTo CodeRatio	Ratio of comments to source code (line based)
countPath	Number of possible paths in the body of a method
complexity	McCabe Cyclomatic complexity of a method
execStmt	Number of executable source code statements
maxNesting	Maximum nested depth of all control structures

Abstract Syntax Tree (AST)



Low-level intermediate represenations



Obtaining raw representations

ИСП РАН

Language specific parsers

Good parsing quality, but often require building source code

- `Clang/LLVM` or `GCC` for C/C++
- Roslyn for C#
- `ast` module for Python
- `javaparser` for Java

Language agnostic parsers

Lightweight in exchange for parsing quality and loss of semantics

- srcML – lightweight multi-language parsing tool
- tree-sitter and ANTL parser generator tools

A Convolutional Attention Network for Extreme Summarization of Code (2016)

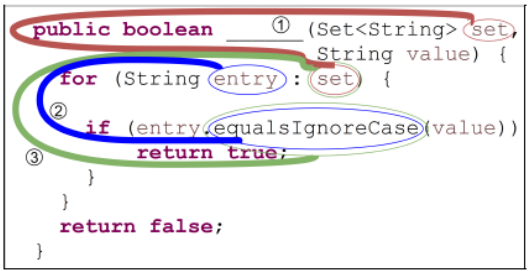
Problem

Extreme summarization of source code snippets into short, descriptive function name-like summaries.

Motivation:

- Learning to summarize source code has important applications in software engineering, such as in code understanding and in code search
- Lack of comprehensive evaluation of algorithms on real-world data
- Previous architectures used either hard-coded features or do not extracttranslation-invariant features specifically
- Solution: introduce a special attentional neural network that employs convolution on the input tokens in acontext-dependent way

Code summarization in Java:



PL summarization specifics

Differences from NL summarization

- source code is mostly unambiguous and highly structured
- need to learn how the code instructions compose into a higher-level meaning
- necessity to learn patterns in code that use both structure and identifiers

Differences from translation

- input source code sequence large and the output summary small
- need to extract both temporally invariant and sentence-wide features
- source code presents the challenge of out-of-vocabulary words

Convolutional Attention Model: Inputs

ИСП РАН

- Input sequence of subtokens $\mathbf{c} = [c_{<s>}, c_1, \ldots, c_N, c_{</s>}]$
- Output sequence of subtokens $\mathbf{m} = [m_{<s>}, m_1, \ldots, m_M, m_{</s>}]$
- $< s >$ and $< /s >$ – special sub-tokens for start and end

Example:

```
boolean shouldRender()
```

```
try {
  return renderRequested||isContinuous;
} finally {
  renderRequested = false;
}
```

$[< s >, try, \{, return, render, requested, \ldots], [< s >, should, render, < /s >]$

Convolutional Attention Model

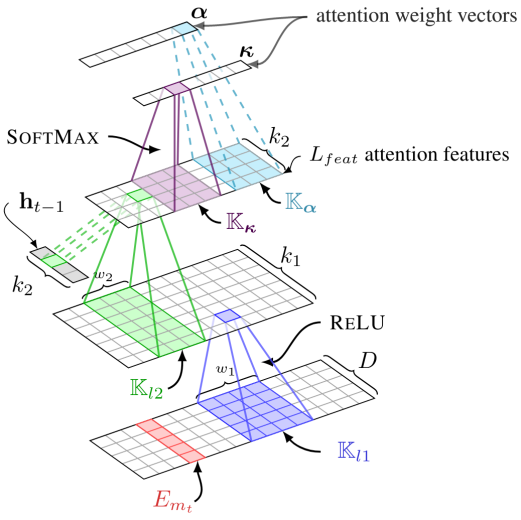
- Model predicts each subtoken sequentially $P(m_t | m_{<s>}, \ldots, m_{t-1}, c)$
- Subtokens $m_{<s>}, \ldots, m_{t-1}$ are passed into RNN that represents the input state with a vector h_{t-1}
- h_{t-1} and subtoken embeddings are used to compute matrix of attention features $\mathcal{L}_{feat}$
- $\mathcal{L}_{feat}$ is used to compute one or more normalized attention vectors
- vectors are used to predict probability distribution over targets m_i

attention_features (code tokens $\mathbf{c}$, context $\mathbf{h}_{t-1}$)
$C \leftarrow \textsc{LookupAndPad}(\mathbf{c}, E)$
$L_1 \leftarrow \textsc{ReLU}(\textsc{Conv1D}(C, \mathbb{K}_{l1}))$
$L_2 \leftarrow \textsc{Conv1D}(L_1, \mathbb{K}_{l2}) \odot \mathbf{h}_{t-1}$
$L_{feat} \leftarrow L_2 / \|L_2\|_2$
return L_{feat}

attention_weights (attention features L_{feat}, kernel $\mathbb{K}$)
return $\textsc{SoftMax}(\textsc{Conv1D}(L_{feat}, \mathbb{K}))$

- $E \in \mathbf{R}^{|V| \times D}$ – D-dimensional subtoken embeddings
- $\mathbf{K_{l_1}} \in \mathbf{R}^{D \times w_1 \times k_1}$ and $\mathbf{K_{l_2}} \in \mathbf{R}^{k_1 \times w_2 \times k_2}$ – convolution kernels
- $h_{t-1} \in \mathbf{R}^{\mathbf{k_2}}$ – information from previous subtokens
- $\mathcal{L}_{feat} \in \mathbf{R}^{(len(c)+const) \times k_2}$, k_2 features for each location
- h_t is computed using GRU

Convolutional Attention Model



Convolutional Attention Model

ИСП РАН

conv_attention (code $\mathbf{c}$, previous state $\mathbf{h}_{t-1}$)
$L_{feat} \leftarrow$ **attention_features**$(\mathbf{c}, \mathbf{h}_{t-1})$
$\boldsymbol{\alpha} \leftarrow$**attention_weights** $(L_{feat}, \mathbb{K}_{att})$
$\hat{\mathbf{n}} \leftarrow \sum_i \alpha_i E_{c_i}$
$\mathbf{n} \leftarrow \textsc{SoftMax}(E\,\hat{\mathbf{n}}^\top + \mathbf{b})$
return $\textsc{ToMap}(\mathbf{n}, V)$

- $\overline{n}$ $in\mathbf{R^D}$ – target embedding
- $b \in \mathbf{R}^{|V|}$ – bias vector
- $n \in \mathbf{R}^{|V|}$ – probability ditribution
- *ToMap* – returns a map of each subtoken $v_i \in V$
- model is trained using max. likelihood

Training setup

- 11 most popular Java open source projects
- extraction of methods with preprocessing
  1. filtering out overrides, constructors and abstract methods using static analysis
  2. substituion of tokens in recursive methods with `self`
  3. split intop subtokens
- SGD with RMSProp, Nesterov momentum, dropout and gradient clipping
- optimized hyperparameters are $k_1 = k_2 = 8, w_1 = 24, w_2 = 29, w_3 = 10$ dropout rate 50% and D = 128
- Measure `Exact match` and per-subtoken `F1`

Evaluation: Projects

Project Name	Git SHA	Description	F1							
			tf-idf		Standard Attention		**conv_attention**		**copy_attention**	
			Rank 1	Rank 5	Rank 1	Rank 5	Rank 1	Rank 5	Rank 1	Rank 5
cassandra	`53e370f`	Distributed Database	40.9	52.0	35.1	45.0	46.5	60.0	**48.1**	**63.1**
elasticsearch	`485915b`	REST Search Engine	27.8	39.5	20.3	29.0	30.8	45.0	**31.7**	**47.2**
gradle	`8263603`	Build System	30.7	45.4	23.1	37.0	35.3	52.5	**36.3**	**54.0**
hadoop-common	`42a61a4`	Map-Reduce Framework	34.7	48.4	27.0	45.7	38.0	54.0	**38.4**	**55.8**
hibernate-orm	`e65a883`	Object/Relational Mapping	53.9	63.6	49.3	55.8	57.5	67.3	**58.7**	**69.3**
intellij-community	`d36c0c1`	IDE	28.5	42.1	23.8	41.1	33.1	49.6	**33.8**	**51.5**
liferay-portal	`39037ca`	Portal Framework	59.6	70.8	55.4	70.6	63.4	75.5	**65.9**	**78.0**
presto	`4311896`	Distributed SQL query engine	41.8	53.2	33.4	41.4	46.3	59.0	**46.7**	**60.2**
spring-framework	`826a00a`	Application Framework	35.7	47.6	29.7	41.3	35.9	49.7	**36.8**	**51.9**
wildfly	`c324eaa`	Application Server	45.2	57.7	32.6	44.4	**45.5**	61.0	44.7	**61.7**

Evaluation: Averaged

ИСП РАН

	F1 (%)		Exact Match (%)		Precision (%)		Recall (%)	
	Rank 1	Rank 5	Rank 1	Rank 5	Rank 1	Rank 5	Rank 1	Rank 5
tf-idf	40.0	52.1	**24.3**	29.3	41.6	55.2	**41.8**	51.9
Standard Attention	33.6	45.2	17.4	24.9	35.2	47.1	35.1	42.1
conv_attention	43.6	57.7	20.6	29.8	57.4	73.7	39.4	51.9
copy_attention	**44.7**	**59.6**	23.5	**33.7**	**58.9**	**74.9**	40.1	**54.2**

Examples

ИСП РАН

boolean shouldRender()

```
try {
  return renderRequested||isContinuous;
} finally {
  renderRequested = false;
}
```

Suggestions: ▶is,render (27.3%) ▶is,continuous (10.6%) ▶is,requested (8.2%) ▶render,continuous (6.9%) ▶get,render (5.7%)

void reverseRange(Object[] a, int lo, int hi)

```
hi--;
while (lo < hi) {
  Object t = a[lo];
  a[lo++] = a[hi];
  a[hi--] = t;
}
```

Suggestions: ▶reverse,range (22.2%) ▶reverse (13.0%) ▶reverse,lo (4.1%) ▶reverse,hi (3.2%) ▶merge,range (2.0%)

int createProgram()

```
GL20 gl = Gdx.gl20;
int program = gl.glCreateProgram();
return program != 0 ? program : -1;
```

Suggestions: ▶create (18.36%) ▶init (7.9%) ▶render (5.0%) ▶initiate (5.0%) ▶load (3.4%)

VerticalGroup right()

```
align |= Align.right;
align &= ~Align.left;
return this;
```

Suggestions: ▶left (21.8%) ▶top (21.1%) ▶right (19.5%) ▶bottom (18.5%) ▶align (3.7%)

boolean isBullet()

```
return (m_flags & e_bulletFlag)
       == e_bulletFlag;
```

Suggestions: ▶is (13.5%) ▶is,bullet (5.5%) ▶is,enable (5.1%) ▶enable (2.8%) ▶mouse (2.7%)

float getAspectRatio()

```
return (height == 0) ?
    Float.NaN : width / height;
```

Suggestions: ▶get,UNK (9.0%) ▶get,height (8.7%) ▶get,width (6.5%) ▶get (5.7%) ▶get,size (4.2%)

code2seq: Generating Sequences from Structured Representations of Code (2019)

Problem

Generate source code description in natural language

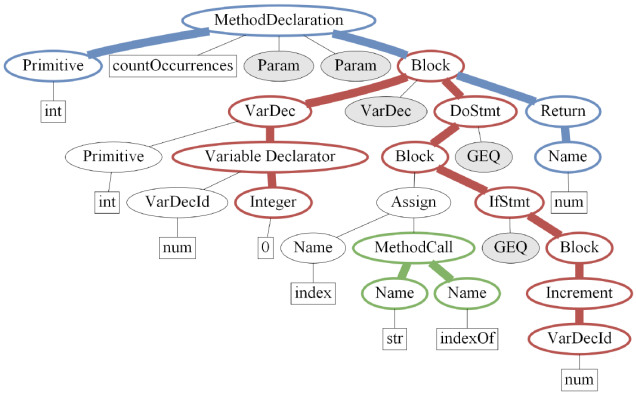
Motivation:

- Existing solutions use seq2seq — source code is represented as token sequence
- Problem: loss of structural information about program (syntax and semantics)
- Suggested solution: use `abstract syntax tree (AST)` representation for source code

Useful properties of AST

ИСП РАН

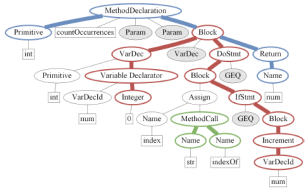
- Finite number of nodes
- Unified source code representation
- Allows to work with any programming language



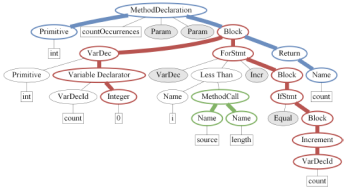
Vector representation of source code

ИСП РАН

```
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;
}
```




```
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;
}
```



- Input – a set of paths between leaf nodes in AST $x = (x_1, ..., x_k)$, where $x_i = (v_{i_1}, ..., v_{i_{l_i}})$ is a node sequence
- Standard NMT architecture
- Encoder transforms a sequence of AST paths (x) into a sequence of vectors of fixed size $z = (\mathbf{z}_1, ..., \mathbf{z}_k)$
- Decoder generates a sequence of output tokens one-by-one $y = (y_1, ..., y_m)$
- During decoding the probability of next token depends on the previously decoded ones:

$$p(y_1, ..., y_m | x_1, ..., x_n) = \prod_{j=1}^{m} p(y_j | y_{<j}, \mathbf{z}_1, ..., \mathbf{z}_n)$$

Encoded AST path x is represented as follows:

$$\mathbf{h}_1, ..., \mathbf{h}_l = \text{BiLSTM}(\mathbf{E}^{\text{nodes}}_{v_1}, ..., \mathbf{E}^{\text{nodes}}_{v_l})$$

$$\text{encode\_path}(v_1, ..., v_l) = [\mathbf{h}^{\rightarrow}_l; \mathbf{h}^{\leftarrow}_1],$$

where $\mathbf{E}^{\text{nodes}}$ — embedding matrix (a set of AST nodes is finite)

Tokens of different case types (e.g. camelCase, snake_case, kebab-case) are tokenized:

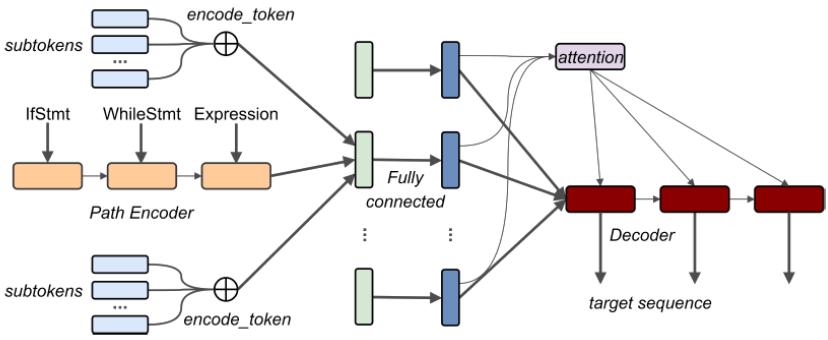
$$\text{encode\_token}(t) = \sum_{s \in \text{split}(t)} \mathbf{E}^{\text{subtokens}}_s$$

$$\mathbf{z} = \tanh(\mathbf{W}_{\text{in}}[\text{encode\_path}(v_1, ..., v_l); \text{encode\_token}(\text{value}(v_1)); \text{encode\_token}(\text{value}(v_l))]),$$

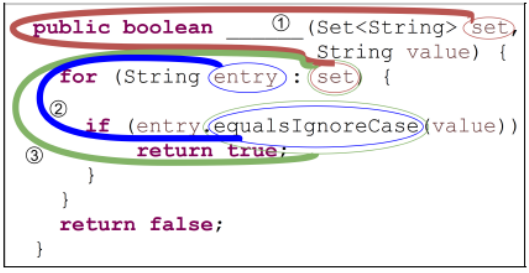
where $\mathbf{W}_{\text{in}}$ — matrix of size $(2d_{\text{path}} + 2d_{\text{token}}) \times d_{\text{hidden}}$

Initial decoder state: $\mathbf{h}_0 = \frac{1}{k}\sum_{i=1}^{k} \mathbf{z}_k$

Attention layer is responsible for selecting only relevant paths



Code summarization in Java:

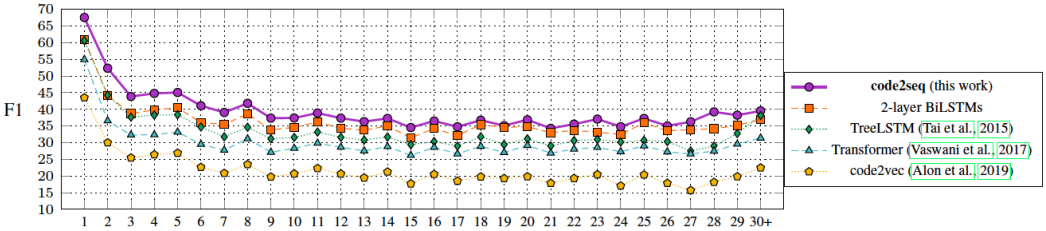


Experiment *A*: results

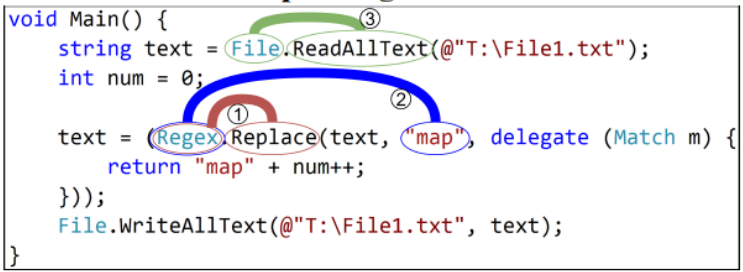
ИСП РАН

Model	Java-small			Java-med			Java-large		
	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

Dependency on code size



Code captioning in C#:



replace① a string② in a text③ file③

Experiment *B*: results

ИСП РАН

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**

Dataset is a set of pairs (question, answer) from StackOverflow for C# language

Selecting number of paths k

Results were collected with $k = 200$ (the number of paths for one AST). This value was picked empirically. Average number of paths in one AST is 220.

Values of $k > 300$ do not provide any improvements, except for rare large examples. Values of $k < 100$ significantly reduce the model performance. Moreover, it was shown that $k = 200$ is optimal for an average GPU.

Model	Precision	Recall	F1	ΔF1
code2seq (original model)	**60.67**	**47.41**	**53.23**	
No AST nodes (only tokens)	55.51	43.11	48.53	-4.70
No decoder	47.99	28.96	36.12	-17.11
No token splitting	48.53	34.80	40.53	-12.70
No tokens (only AST nodes)	33.78	21.23	26.07	-27.16
No attention	57.00	41.89	48.29	-4.94
No random (sample k paths in advance)	59.08	44.07	50.49	-2.74

CodeBERT: A Pre-Trained Model for Programming and Natural Languages (2020)

CodeBERT: Idea

Problem

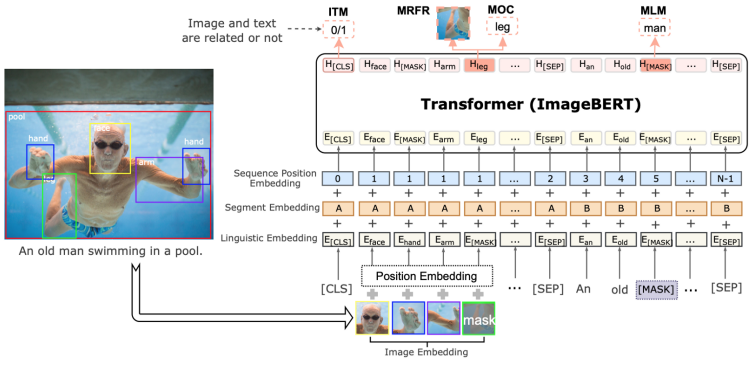
Learn general-purpose code representations that support downstream NL-PL applications.

Motivation:

- Existing solutions for NLP tasks use only natural language training data.
- Problem: there is no semantic connection between programming code and its natural description for such problems as natural language code search, code documentation generation
- Suggested solution: use multi-modal approach to connect natural language and source code

CodeBERT: Bimodal data

- Bimodal approach is using two different types of data to detect semantic connection
- Multi-modal pre-trained model, like VideoBERT, ImageBERT



CodeBERT: Pre-traininig data

- Dataset used is CodesearchNet challenge
- Train on bimodal data (natural language - programming language)
- Model was trained both on bimodal and unimodal (PL without documents)

TRAINING DATA	*bimodal* DATA	*unimodal* CODES
GO	319,256	726,768
JAVA	500,754	1,569,889
JAVASCRIPT	143,252	1,857,835
PHP	662,907	977,821
PYTHON	458,219	1,156,085
RUBY	52,905	164,048
ALL	2,137,293	6,452,446

Table 1: Statistics of the dataset used for training CodeBERT.

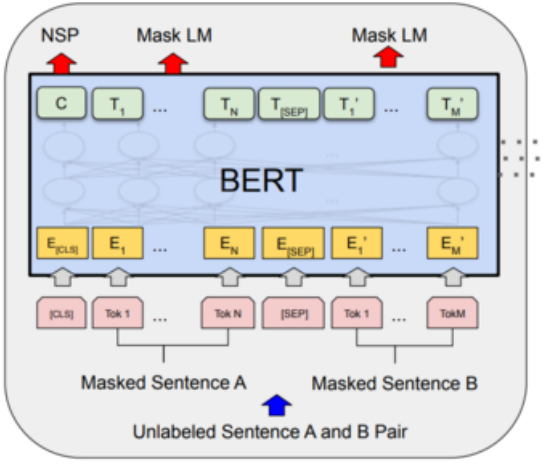
CodeBERT: Data example

ИСП РАН

```
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()])
```

CodeBERT: Masked Language Modeling



$$
\begin{aligned}
m_i^w &\sim unif\{1, |\mathbf{w}|\} \ for \ i = 1 \ to \ |\mathbf{w}| \\
m_i^c &\sim unif\{1, |\mathbf{c}|\} \ for \ i = 1 \ to \ |\mathbf{c}| \\
\mathbf{w}^{masked} &= REPLACE(\mathbf{w}, m^{\mathbf{w}}, [MASK]) \\
\mathbf{c}^{masked} &= REPLACE(\mathbf{c}, m^{\mathbf{c}}, [MASK]) \\
\mathbf{x} &= \mathbf{w} + \mathbf{c}
\end{aligned}
$$

$$
\mathcal{L}_{MLM}(\theta) = \sum_{i \in m^{\mathbf{w}} \cup m^{\mathbf{c}}} - log \ p^{D_1}(x_i | w^{masked}, c^{masked})
$$

CodeBERT: Replaced Token Detection

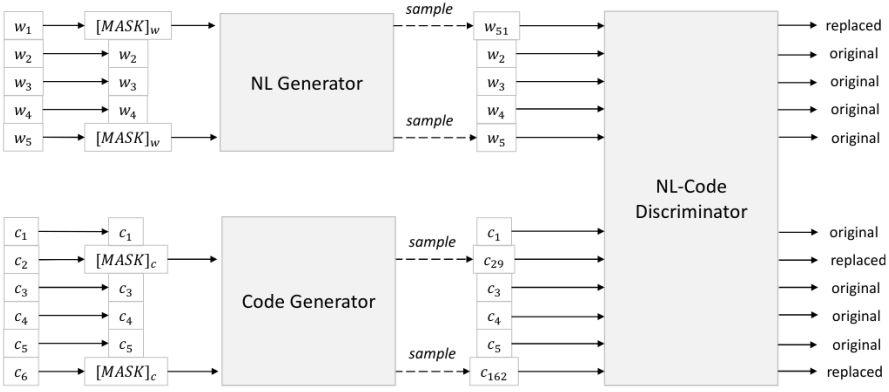


Figure 2: An illustration about the replaced token detection objective. Both NL and code generators are language models, which generate plausible tokens for masked positions based on surrounding contexts. NL-Code discriminator is the targeted pre-trained model, which is trained via detecting plausible alternatives tokens sampled from NL and PL generators. NL-Code discriminator is used for producing general-purpose representations in the fine-tuning step. Both NL and code generators are thrown out in the fine-tuning step.

CodeBERT: Replaced Token Detection

ИСП РАН

$$\hat{w}_i \sim p^{G_w}(w_i|\mathbf{w}^{masked}) for\ i = 1 \in m^{\mathbf{w}}$$
$$\hat{c}_i \sim p^{G_c}(c_i|\mathbf{c}^{masked}) for\ i = 1 \in m^{\mathbf{c}}$$
$$\mathbf{w}^{corrupt} = REPLACE(\mathbf{w}, m^{\mathbf{w}}, \hat{w})$$
$$\mathbf{c}^{corrupt} = REPLACE(\mathbf{c}, m^{\mathbf{c}}, \hat{c})$$
$$\mathbf{x}^{corrupt} = \mathbf{w}^{corrupt} + \mathbf{c}^{corrupt}$$

$$\mathcal{L}_{RTD}(\theta) = \sum_{i=1}^{|\mathbf{w}|+|\mathbf{c}|} \delta(i) log p^{D_2}(\mathbf{x}^{corrupt}, i) + (1 - \delta(i)) \left(1 - log p^{D_2}(\mathbf{x}^{corrupt}, i)\right)$$

$$\delta(i) = \begin{cases} 1, & if \mathbf{x}_i^{corrupt} = \mathbf{x}_i \\ 0, & otherwise \end{cases}$$

$$\mathcal{L}_{final} = \min_{\theta} \mathcal{L}_{MLM}(\theta) + \mathcal{L}_{RTD}(\theta)$$

CodeBERT: Results: Code retrieval

ИСП РАН

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**

CodeBERT: Results: Documantation generation

ИСП РАН

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.

CodeBERT: Results

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%	-	-

Hoppity: learning graph transformations to detect and fix bugs in programs (2020)

Problem

Detect and auto-fix defects in programs written in JavaScript.

Motivation:

- Size and complexity of source code $\Rightarrow$ lots of vulnerabilities
- Existing tools are focused on specific vulnerabilities or projects
- Lack of end-to-end tools for JavaScript programming language

- Dynamic typing
- Syntax and semantics, that lead to specific errors:
  - Access to undefined properties
  - Incorrect usage of comparison operators (`!=` vs `!==`)
  - Issues with variable scope (var)
- Lack of good tooling for error detection

JavaScript specifics

ИСП РАН

```
function clearEmployeeListOnLinkClick(){
  document.querySelector("a").addEventListener("click",
    function(event){
      document.querySelector("ul").InnerHTML = "";
    }
  );
}
```

(a) `InnerHTML` should have been `innerHTML`.

```
if (matches) {
  return {
    episode: Number(matches.groups.episode),
    hosts: matches.groups.hosts.split(/([,&]+|\sand\s)/).
              map(el => S(el).trim().s)
  };
}
```

(b) Highlighted parentheses should have been removed.

```
module.exports = function (grunt) {
  grunt.initConfig({
    execute: {...}, copy: {...}, checktextdomain: {...}
    wp_readme_to_markdown: {...}, makepot: {...}})
  ...
  grunt.registerTask('default',['wp_readme_to_markdown'
    ,'makepot','execute','checktextdomain'])
};
```

(c) `copy` function should have also been included in the highlighted list.

```
export default {
  computed: {
    level () {
      return dictMap.skillLevel[
        parseInt((this.value === 0 ? 1 : this.value)/20)];
    }
  },...
}
```

(d) `parseInt` should have been removed because `===` implies `this.value` is an integer.

Approach

Idea

Given an intenral graph representation of a program containing bug output a sequence of graph transformations (graph-to-graph)

Difference from existing solutions:

- Single model for detection and fixing of bugs
- Support for multiple types of bugs
- Support for more complex transformations (e.g. addition/removal of expressions)

Problem formulation

Task of structured prediction on a graph representation of a program.

Pairs $(g_{\text{bug}}, g_{\text{fix}})$, where g_{bug} — program graph with bug; g_{fix} — graph for fixed program.

Model for predicting up to T transformations

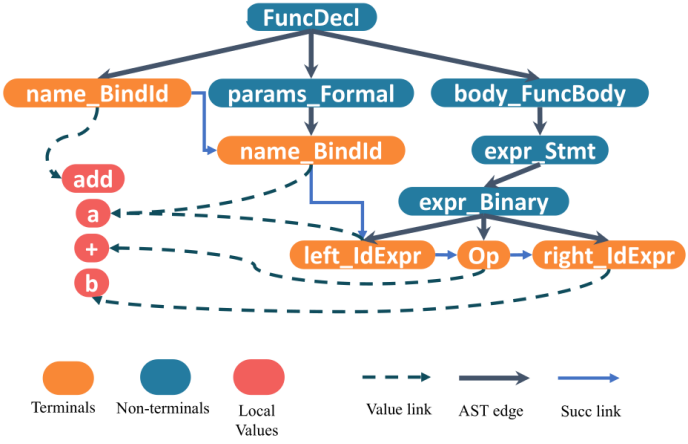
$$p(g_{\text{fix}}|g_{\text{bug}};\theta) = p(g_1|g_{\text{bug}};\theta)p(g_2|g_1;\theta)\dots p(g_{\text{fix}}|g_{T-1};\theta)$$

Program representation

- Program graph — enhanced AST
- `SuccToken` edges that connect leaf nodes
- Value-nodes with values for leaf nodesco значениями для листовых вершин
- `ValueLink` edges that connect leaf nodes with value nodes

Value-nodes and `ValueLink` are necessary for representing and performing transforming independent from specific names.

Program representation: example




```
function add(a) { a + b; }
```

Vector representation of graphs

- AST $g = (V, E)$, where $v \in V$ — a set of nodes, and E — a set of edges of different types
- K — number of unique edge types in graph
- $f(g) \to (\mathbb{R}^d, \mathbb{R}^{|V| \times d})$, for d-dimensional representation of graph $\vec{g}$ and nodes $\vec{v}$
- For parametrization $f(\cdot)$ — graph neural network, variation of Graph Isomorphism Network (GIN)

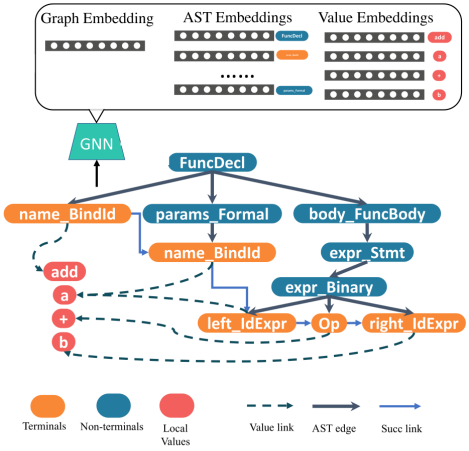
GIN variation

$$\mathbf{h}_v^{(l+1),k} = \tanh\Big(\sum_{u \in \mathcal{N}^k(v)} \mathbf{W}_1^{l,k} \mathbf{h}_u^{(l)}\Big), \quad \forall k \in 1, 2, \ldots, K$$

$$\mathbf{h}_v^{(l+1)} = \tanh\big(\mathbf{W}_2^{l}[\mathbf{h}_v^{(l+1),1}, \mathbf{h}_v^{(l+1),2}, \ldots, \mathbf{h}_v^{(l+1),K}] + \mathbf{h}_v^{(l)}\big)$$

- $\mathbf{W}_1^{l,k} \in \mathbb{R}^{d \times d}$, $\mathbf{W}_2^{l} \in \mathbb{R}^{dK \times d}$ — model parameters
- $\mathcal{N}^k(v)$ — a set of neighbours of node v with edge type k
- Embedding $\vec{v} = \mathbf{h}_v^{(L)}$, where L — number of propagation iterations
- Embedding $\vec{g}$ — average of max pooling aggregations $\mathbf{h}_v^{l}, \forall l \in 0, 1, \ldots, L$

Graph neural network




```
function add(a) { a + b; }
```

Graph transformations: operators

Each transformation step is one of 5 operators:

1. **ADD** — add AST node
2. **DEL** — delete AST node
3. **REP_VAL** — replace AST node value
4. **REP_TYPE** — replace AST node type
5. **NO_OP** — end of transformation sequence

Each operator is based on a common set of low-level *primitives*.

Three low-level primitives:

1. Location primitive
2. Value primitive
3. Type primitive

Controller

Vector $\vec{c} \in \mathbb{R}^d$ (d — number of embedding dimensions) encodes global state and transformation history.

Location primitive

- Responsible for selecting node (source code region) for transformation
- Different programs have different number of nodes
- Uses special neural network models — pointer networks (Vinyals et al. 2015)
- After obtaining node embedding $\{\vec{v}\}_{v \in V}$, select node using

$$\text{loc}(\vec{c}, g) = \arg \max_{(v \in V)} \vec{v}^T \vec{c}$$

Value primitive

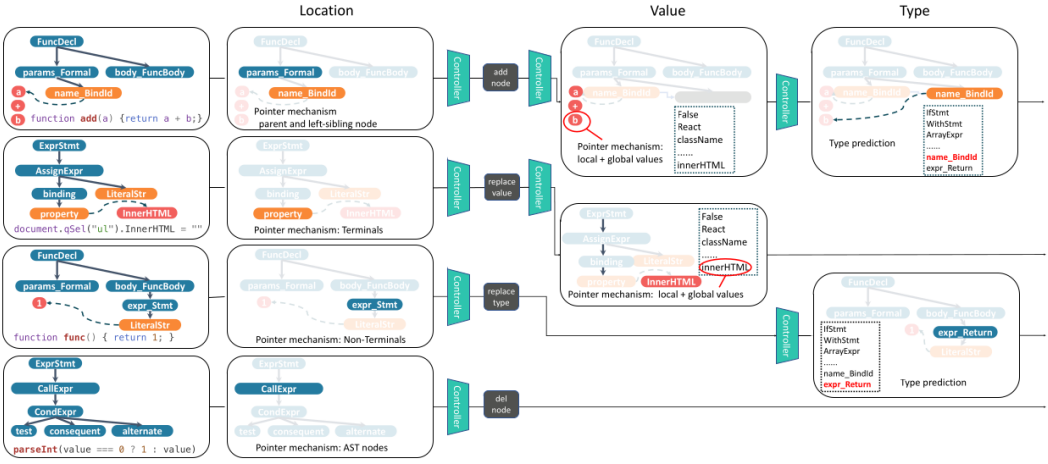
- Assigns concrete value to AST leaf node
- Uses attention mechanism, which allows to select:
  - local values from current module (Local Value Table V_{val})
  - global values, which are more frequent in leaf nodes for the given language (Global Value Dictionary D_{val})
- Values are picked using:

$$\text{val}(\vec{c}, g) = \arg \max_{(t \in D_{\text{val}} \cup V_{\text{val}})} \vec{t}^{T} \vec{c}$$

Type primitve

- Assigns type to non-terminal AST node
- A set of types for the given language is finite $\Rightarrow$ multi-class classification problem
- Set size is further reduced with the help of syntax rules for AST (grammar rules validity)

Graph transformations: operators and primitives



Graph transformations: step t

Graph g_{t-1}, embedding $\vec{g}_{t-1}$ and macro-context vector $\vec{C}_{M_{t-1}}$

1. Updaate macro-context: $\vec{C}_{M_t} = \text{LSTM}(\vec{g}_{t-1} | \vec{C}_{M_{t-1}})$
2. Select node v for transformation: $\text{loc}(\vec{C}_{M_t}, \vec{g}_{t-1})$
3. Select operator e_t for the given v_t and $\vec{C}_{M_t}$ (for NO_OP end of sequence)
   - paper does not provide information on how, seems like softmax layer :)
4. Compute micro-context for operator: $\vec{c}_{m_t} = \text{LSTM}(\vec{e}_t | \text{LSTM}(\vec{v}_t | \vec{C}_{M_t}))$
5. Apply operator e_t with micro-context $\vec{c}_{m_t}$ (used to obtain val), obtain graph g_t

Training phase: dataset

- Dataset $\mathcal{D} = \{(g_{\text{bug}}^{(i)}, g_{\text{fix}}^{(i)})\}_{i=1}^{|\mathcal{D}|}$ с Github $\sim$ 500 thousands of pairs
- SHIFT AST and JSON diff for extracting AST transformation sequences
- Three separate setups for different number of transformations:
  1. `OneDiff` — fixes with exactly 1 transformation
  2. `ZeroOneDiff` — fixes with 0 or 1 transformations
  3. `ZeroOneTwoDiff` — fixes with 0, 1 or 2 transformations
- Additional filtration of ASTs with number of nodes >500

Training phase

- Goal — expectation maximization for transformation:

$$\max_{\theta} \mathbb{E}_{[(g_{\text{bug}}, g_{\text{fix}}) \sim \mathcal{D}]} p\left(g_{\text{fix}} | g_{\text{bug}}; \theta\right)$$

- Loss function — sum of cross entropy losses for each transformation
- Adam with $\beta_1 = 0.9, \beta_2 = 0.99$, initial learning rate 10^{-3}, batch size 10 and 3 epochs
- Embedding layers, operator layers and controller layer (LSTM) are jointly opimized

Inference phase

- Select max: $G = \arg\max_{g_{\text{fix}}} p(g_{\text{fix}}|g_{\text{bug}};\theta)$
- The search space is huge $\Rightarrow$ use beam search
- The search space is limited by parameter B: the breadth of beam search; used $B = 1$ and $B = 3$
- Output ranked list of Top-B probable transformations

	ADD	REP_TYPE	REP_VAL	DEL	total
train	6,473	1,864	251,097	31,281	290,715
validate	790	245	31,357	3,957	36,349
test	796	233	31,387	3,945	36,361

Table 1: Statistic of OneDiff dataset. See appendix for more information of other dataset.

	Total		*Location*		*Operator*	*Value*		*Type*	
	Top-3	Top-1	Top-3	Top-1	Top-1	Top-3	Top-1	Top-3	Top-1
TOTAL	**26.1**	14.2	35.5	20.4	34.4	52.3	29.1	76.1	66.7
ADD	52.9	39.2	69.6	51.4	70.6	65.7	55.1	76.8	68.5
REP_VAL	23.4	11.9	33.3	18.5	31.7	53.0	28.8	-	-
REP_TYPE	71.7	52.4	73.0	52.8	79.4	-	-	74.7	61.0
DEL	39.6	24.8	44.0	27.5	45.8	-	-	-	-
Random	.08	.07	2.28	1.4	27.7	.01	.01	.27	0

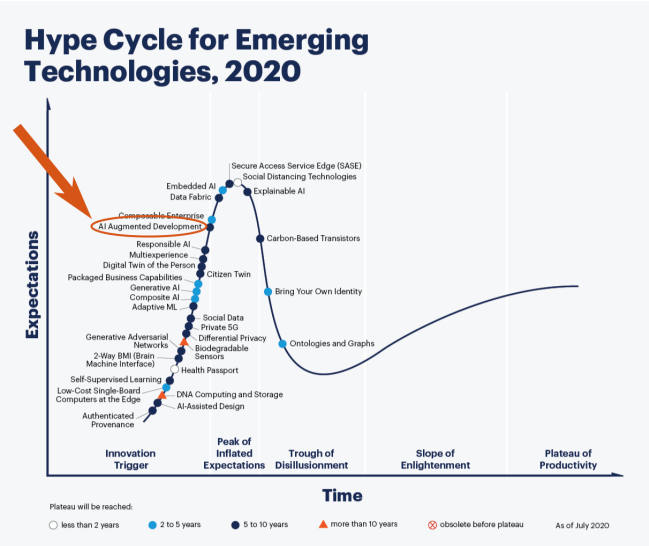
Table 2: Evaluation of model on the OneDiff dataset: accuracy (%).

	Total		*Location*		*Operator*	*Value*		*Type*	
	Top-3	Top-1	Top-3	Top-1	Top-1	Top-3	Top-1	Top-3	Top-1
`ZeroOneTwoDiff`	40.8	29.7	18.9	3.9	30.3	35.0	6.5	38.6	3.4
`ZeroOneDiff`	51.6	34.5	27.1	5.5	35.6	45.4	10.4	73.9	58.9
`OneDiff`	**26.1**	14.2	35.5	20.4	34.4	52.3	29.1	76.1	66.7
Random	.08	.07	2.28	1.4	27.7	.01	.01	.27	0

Table 7: Evaluation of models on each dataset. The Random model is evaluated on the `OneDiff` dataset and is shown for comparison.

Summary

We are not there yet



Actively researched areas and tasks

ИСП РАН

Widespread:

- Source code summarization (commit text generation, name suggestion)
- Defect, code smell and tecnhical debt detection
- Semantic code search (similar code; by natural language query)
- Edit/fix pattern discovery and suggestion
- Neural program synthesis, transformation and repair

More rare:

- Refactoring detection and suggestion
- Intelligent source code autocompletion
- Bug, developer and reiviewer triaging

Promising directions

1. More high-quality data
2. Better quality for downstream tasks
3. AI4SE infrastructure and integrations with common workflows
4. Explainable AI for Software Engineering
5. Trusted AI for Software Engineering
    - Licence violoations
    - Suggestions of buggy or exploitable code
    - Vulnerabilities in pre-trained models

Useful materials

- Machine Learning on Source Code (https://ml4code.github.io/)
- Miltos Allamanis https://miltos.allamanis.com/
- Awesome Machine Learning On Source Code (https://github.com/src-d/awesome-machine-learning-on-source-code)
- JetBrains Research Youtube AI4SE series (https://www.youtube.com/playlist?list=PLJyTG7NfyQ8maHkU8dTJJWm81giLab380)

Companies, research groups and tools

ИСП РАН

Research and development:

- Microsoft (Deep Program Understanding Research Group, CoPilot)
- Facebook AI Research (Aroma, GetaFix, CompilerGym)
- Google Research (CuBERT, program synthesis)
- JetBrains (Machine Learning Methods in Software Engineering)

Production tools:

- **Kite** and **TabNine** for intelligent autocompletion
- **DeepCode** and **Embold** to find and fix vulnerabilities in code
- **Sourcery** and IBM's **Mono2Micro** for AI-assisted refactoring

AI4SE team at ISP RAS

ИСП РАН

Source Code Analysis Assistant

- Mining fix-patterns from development history
- Learning source code embeddings for downstream tasks
- Detection of technical debt and refactoring suggestions

Mobile Application Testing Assistant

- Automated exploratory testing (Deep RL)
- UI element detection
- Visual anomaly detection (rendering issues, occlusions)
- Performance anomaly detection