Pre-training Code Representation with Semantic Flow Graph for Effective Bug localization

서울시립대학교
UNIVERSITY OF SEOUL

CIDA Lab.

전자전기컴퓨터공학부
2022440119 전형준

2024.11.19

Table of Contents

Problem Definition

BERT has been used for bug localization tasks and impressive results have been obtained. However, these BERT-based bug localization techniques suffer from two issues

1. **Inadequate capture of deep semantics in program code**
2. **Insufficient use of negative samples & Neglect of lexical similarity between bug reports and changesets.**

Problem Definition

1. Inadequate capture of deep semantics in program code

Unlike natural language, the programming language has a formal structure, which provides important code semantics that is unambiguous in general.

Pre-trained BERT model either totally ignores the code structure by treating code snippet as a sequence of tokens same as natural language or considers only the shallow structure of the code by using graph code representations such as data flow graph

Problem Definition

2. Insufficient use of negative samples & Neglect of lexical similarity between bug reports and changesets.

Some techniques select only one irrelevant changeset as a negative sample per bug report, leading to inefficient negative sample mining and poor model performance.

Existing BERT-based bug localization techniques only account for the semantic level similarity between bug reports and changesets, totally ignoring the lexical similarity

Problem Definition

6 files changed +306 -243 lines changed

Top

Search within code

dataset_curation.py +298 -179

```diff
 
+def remove_extra_newlines(text):
+    return re.sub(r'\n\s*\n+', '\n', text)
+
+def preprocess_cpp_code(cpp_code):
+    comment_removed_code = remove_cpp_comments(cpp_code).strip()
+    comment_removed_code = remove_extra_newlines(comment_removed_code)
+    formatted_code = format_cpp_code(comment_removed_code)
+    return formatted_code if formatted_code == None else formatted_code.strip()
 
-def calculate_levenshtein_distance(s1, s2):
-    return Levenshtein.distance(s1, s2)
+def preprocess_python_code(python_code):
+    comment_removed_code = remove_python_comments(python_code).strip()
+    comment_removed_code = remove_extra_newlines(comment_removed_code)
+    formatted_code = format_python_code(comment_removed_code)
+    return formatted_code if formatted_code == None else formatted_code.strip()
 
```

Semantic Flow Graph

a = m (b, c)

Data flow: edges b → a and c → a
The values of two variables have flown into another variable

a = (b &&m (c))

Semantic Flow Graph

a = m (b, c)

Data flow: edges b → a and c → a

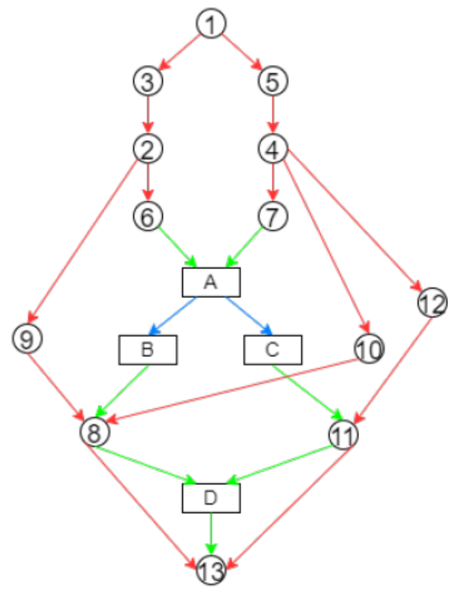
+) what kinds of program elements
+) which operations are taken into account

the value of an Integer type variable and
the value of a User-defined type variable
have flown into another Boolean type
variable through a function call

Semantic Flow Graph

double func (int a_1) {
  double x_2 = sqrt(a_3);
  double y_4 = log(a_5);
  if ($x_6 > y_7$)
    $x_8 = x_9 * y_{10}$;
  else
    $x_{11} = y_{12}$;
  return x_{13};
}

- Data Flow Edges
- Control Flow Edges
- Sequential Computation Flow Edges
- Related with Variables
- Related with Control Instructions

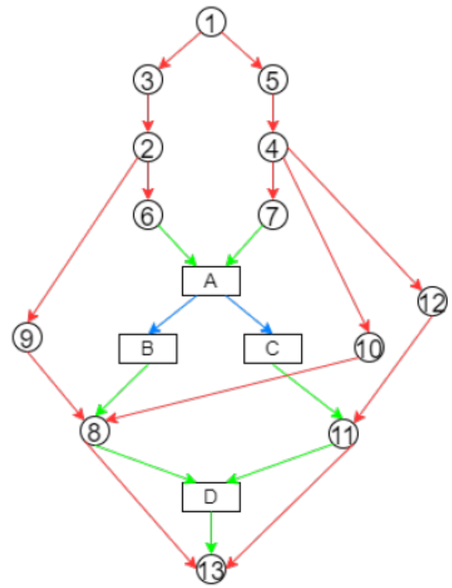


Nodes	Type	Role
1	int	ParameterIntr
2	double	Assigned
3	int	InvocationArgument
4	double	Assigned
5	int	InvocationArgument
6	double	CompareOperatorLeft
7	double	CompareOperatorRight
8	double	Assigned
9	double	MathOperatorLeft
10	double	MathOperatorRight
11	double	Assigned
12	double	Assignment
13	double	ReturnExper
A	IfCondition	---
B	IfThen	---
C	IfElse	---
D	IfConvergence	---

The Semantic Flow Graph (SFG)
for a code snippet is a tuple $< N, E, T, R >$

Semantic Flow Graph

```
double func (int a1) {
    double x2 = sqrt(a3);
    double y4 = log(a5);
    if ( x6 > y7 )
        x8 = x9 * y10;
    else
        x11 = y12;
    return x13;
}
```



Data Flow Edges
Control Flow Edges
Sequential Computation Flow Edges
Related with Variables
Related with Control Instructions

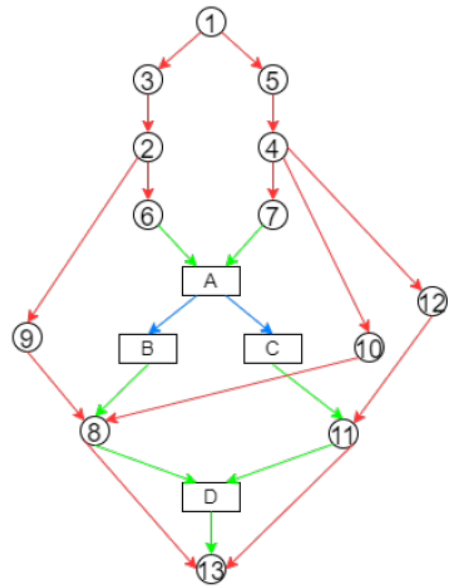
Nodes	Type	Role
1	int	ParameterIntr
2	double	Assigned
3	int	InvocationArgument
4	double	Assigned
5	int	InvocationArgument
6	double	CompareOperatorLeft
7	double	CompareOperatorRight
8	double	Assigned
9	double	MathOperatorLeft
10	double	MathOperatorRight
11	double	Assigned
12	double	Assignment
13	double	ReturnExper
A	IfCondition	---
B	IfThen	---
C	IfElse	---
D	IfConvergence	---

N is consisted of N_v and N_c

N_v = one to one mapping to variables
N_c = multiple node in N_c for a certain control instruction.

Semantic Flow Graph

double func (int a_1) {
  double x_2 = sqrt(a_3);
  double y_4 = log(a_5);
  if ($x_6 > y_7$)
    $x_8 = x_9 * y_{10}$;
  else
    $x_{11} = y_{12}$;
  return x_{13};
}



Data Flow Edges
Control Flow Edges
Sequential Computation Flow Edges
Related with Variables
Related with Control Instructions

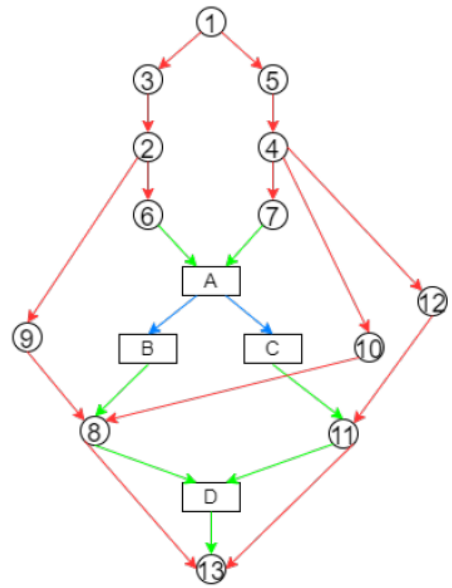
Nodes	Type	Role
1	int	ParameterIntr
2	double	Assigned
3	int	InvocationArgument
4	double	Assigned
5	int	InvocationArgument
6	double	CompareOperatorLeft
7	double	CompareOperatorRight
8	double	Assigned
9	double	MathOperatorLeft
10	double	MathOperatorRight
11	double	Assigned
12	double	Assignment
13	double	ReturnExper
A	IfCondition	---
B	IfThen	---
C	IfElse	---
D	IfConvergence	---

E is consisted of E_D, E_c and E_S

E_D = Intra-block and Inter-block data dependencies between variables.
E_c = specific control flow of the control instruction.
E_S = edge between N_v and N_c

Semantic Flow Graph

```
double func (int a1) {
    double x2 = sqrt(a3);
    double y4 = log(a5);
    if ( x6 > y7 )
        x8 = x9 * y10;
    else
        x11 = y12;
    return x13;
}
```



Data Flow Edges
Control Flow Edges
Sequential Computation Flow Edges
Related with Variables
Related with Control Instructions

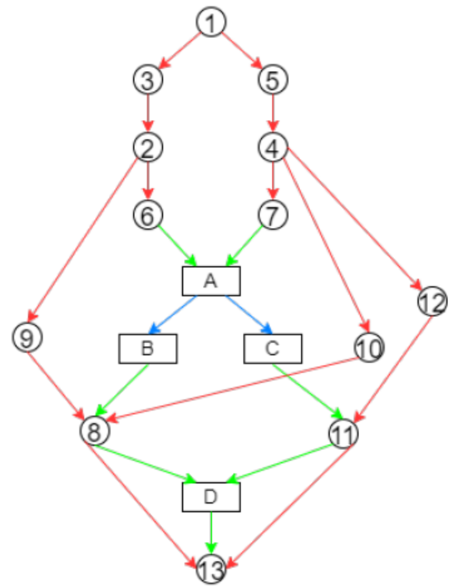
Nodes	Type	Role
1	int	ParameterIntr
2	double	Assigned
3	int	InvocationArgument
4	double	Assigned
5	int	InvocationArgument
6	double	CompareOperatorLeft
7	double	CompareOperatorRight
8	double	Assigned
9	double	MathOperatorLeft
10	double	MathOperatorRight
11	double	Assigned
12	double	Assignment
13	double	ReturnExper
A	IfCondition	---
B	IfThen	---
C	IfElse	---
D	IfConvergence	---

T maps each node in N to its type
“what kinds of program elements are related”

For N_v, T maps it to the corresponding type of the variable
For N_c, T maps the node to the specific part
of the control instruction it refers to.

Semantic Flow Graph

double func (int a_1) {
    double x_2 = sqrt(a_3);
    double y_4 = log(a_5);
    if ($x_6 > y_7$)
        $x_8 = x_9 * y_{10}$;
    else
        $x_{11} = y_{12}$;
    return x_{13};
}



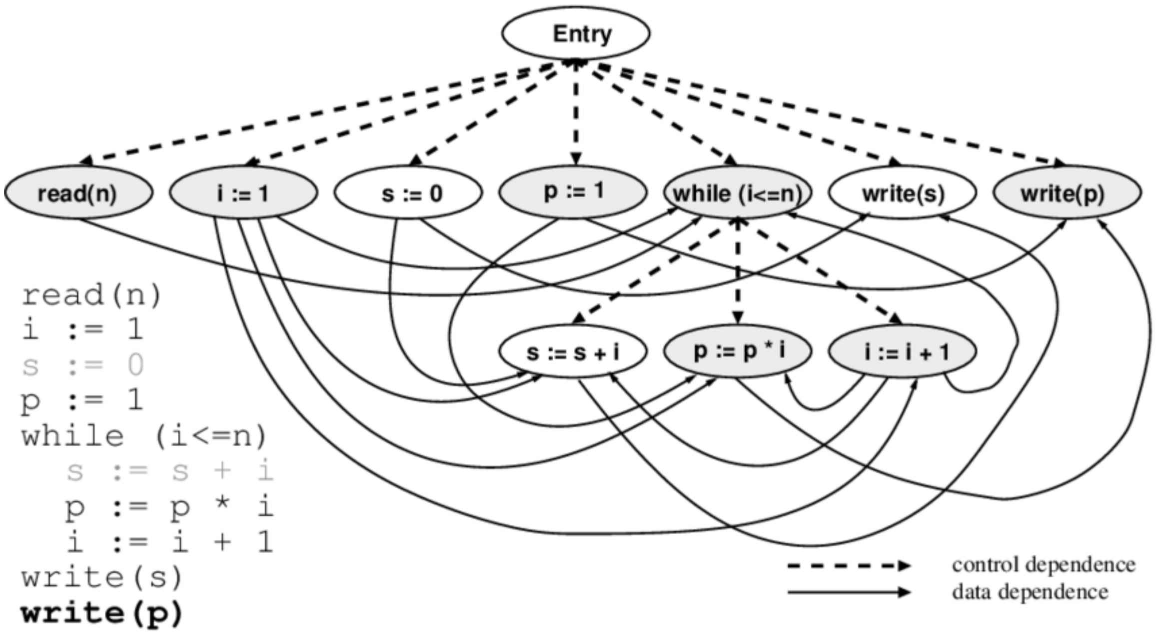
Data Flow Edges
Control Flow Edges
Sequential Computation Flow Edges
Related with Variables
Related with Control Instructions

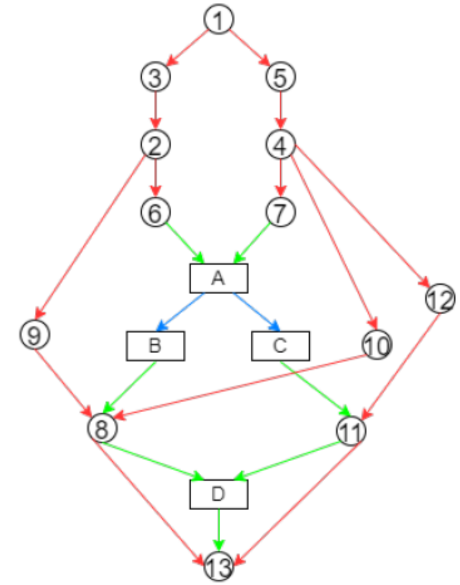
Nodes	Type	Role
1	int	ParameterIntr
2	double	Assigned
3	int	InvocationArgument
4	double	Assigned
5	int	InvocationArgument
6	double	CompareOperatorLeft
7	double	CompareOperatorRight
8	double	Assigned
9	double	MathOperatorLeft
10	double	MathOperatorRight
11	double	Assigned
12	double	Assignment
13	double	ReturnExper
A	IfCondition	---
B	IfThen	---
C	IfElse	---
D	IfConvergence	---

R maps each node in NV to its role in the computation "through which operations program elements are related"

For nodes in NC, we do not consider their roles as they are implicit in their types

Semantic Flow Graph





Existing representations like program dependence graph
Typically work at the statement granularity

The proposed SFG works at a finer granularity
with two types of nodes (N_c, N_v).

Data flow and control flow can be encoded through
the edges between nodes, and the type and computation role
Information can be encoded through node labels.

Semantic Flow Graph

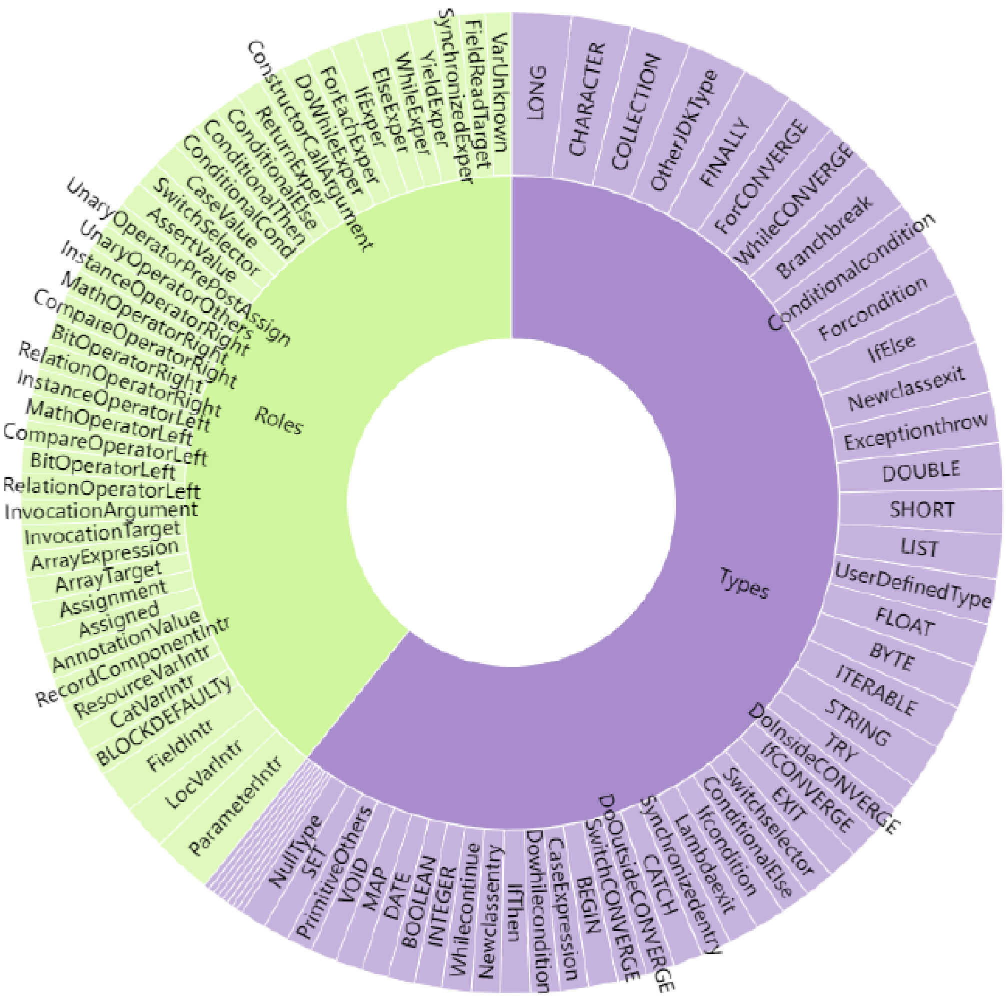
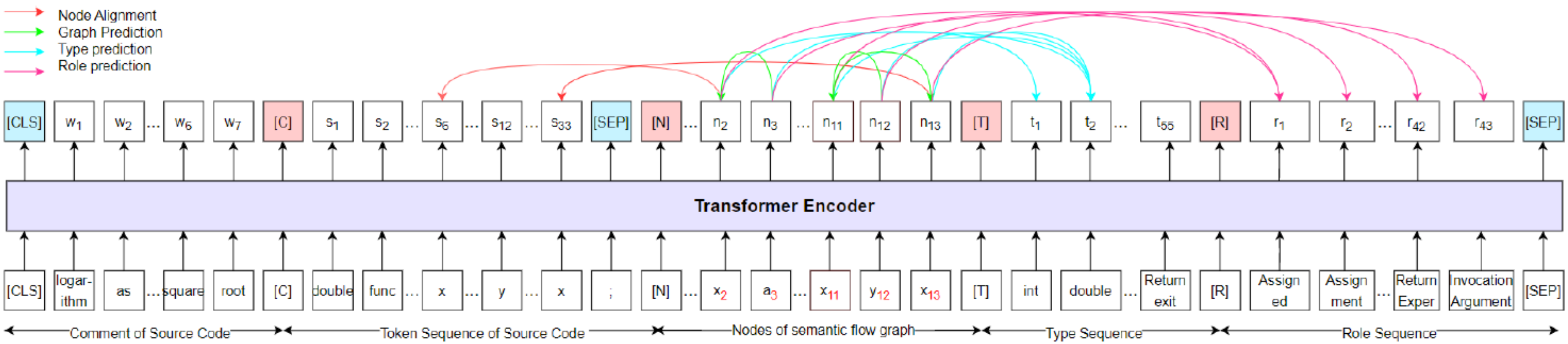


Figure 2: All defined types and roles.

Considered 20 types for nodes in N_v, 35 types for nodes in N_c.

With regard to role, the analyzer considers 43 different roles in total for nodes in N_v.

SemanticCodeBERT



The SemanticCodeBERT follows BERT (Bidirectional Encoder Representation from Transformers) as the backbone

Input X = Concat[[CLS], W, [C], S, [SEP], [N], N, [T], T, [R], R, [SEP]]

SemanticCodeBERT

1. Comment Input Sequence
Comments as a supplement to understand semantic information

2. Source Code Input Sequence
Sequence of Source Code tokens

3. Node Input Sequence
Nodes from Semantic flow graph

4. Type Input Sequence
$T = \{t1, \ldots, t55\}$ represents the set of all 55 possible types

5. Role Input Sequence
$R = \{r1, \ldots, r43\}$ is the set of all 43 possible roles

Input X = Concat[[CLS], W, [C], S, [SEP], [N], N, [T], T, [R], R, [SEP]]

Masked Attention

To filter irrelevant signals in Transformer, we use masked attention.

In SemanticCodeBERT, unmasked attention value indicates
Direct edge relation between two tokens.

The masked attention matrix is formulated as M:

$$M_{ij} = \begin{cases} 0 & x_i \in [CLS], [SEP]; \\ & w_i, s_j \in W \cup S; \\ & (s_i, n_j)/(n_j, s_i) \in E^1; \\ & (n_i, n_j) \in E^2; \\ & (n_i, t_j) \in E^3; \\ & (n_i, r_j) \in E^4; \\ -\infty & otherwise. \end{cases}$$

Pre-training Tasks

Node Alignment – train model to predict where the nodes are identified from.
Loss function L_{NA} can be defined using $p_{e_{ij}}$, which is the probability of edge from i-th code token and j-th node. (sigmoid function after dot product)

$\delta(e_{ij})$ = 1 if (si, nj) is in E1, else 0.

$$\mathcal{L}_{NA} = -\sum_{e_{ij} \in E^1_{mask}} [\delta(e_{ij}) log p_{e_{ij}} + (1 - \delta(e_{ij})) log(1 - p_{e_{ij}})].$$

Pre-training Tasks

Edge Prediction– train model to learn structural relationship from SFG. Loss function L_{GP} can be defined using $p_{e_{ij}}$, which is the probability of edge from i-th node and j-th node. (sigmoid function after dot product)

$\delta(e_{ij})$ = 1 if (ni, nj) is in E2, else 0.

$$\mathcal{L}_{GP} = -\sum_{e_{ij} \in E^2_{mask}} [\delta(e_{ij}) log p_{e_{ij}} + (1 - \delta(e_{ij})) log(1 - p_{e_{ij}})].$$

Pre-training Tasks

Type Prediction – train model to comprehend the types of nodes.
Loss function L_{TP} can be defined using $p_{e_{ij}}$, which is the probability of edge from i-th node and j-th type. (sigmoid function after dot product)

$\delta(e_{ij})$ = 1 if (ni, tj) is in E3, else 0.

$$\mathcal{L}_{TP} = -\sum_{e_{ij} \in E^3_{mask}} [\delta(e_{ij}) log p_{e_{ij}} + (1 - \delta(e_{ij})) log(1 - p_{e_{ij}})].$$

Pre-training Tasks

Rold Prediction– train model to predict the role of each nodes.
Loss function L_{RP} can be defined using $p_{e_{ij}}$, which is the probability of edge from i-th node and j-th role. (sigmoid function after dot product)

$\delta(e_{ij})$ = 1 if (ni, rj) is in E4, else 0.

$$\mathcal{L}_{RP} = -\sum_{e_{ij} \in E^4_{mask}} [\delta(e_{ij}) log p_{e_{ij}} + (1 - \delta(e_{ij})) log(1 - p_{e_{ij}})].$$

Problem Definition

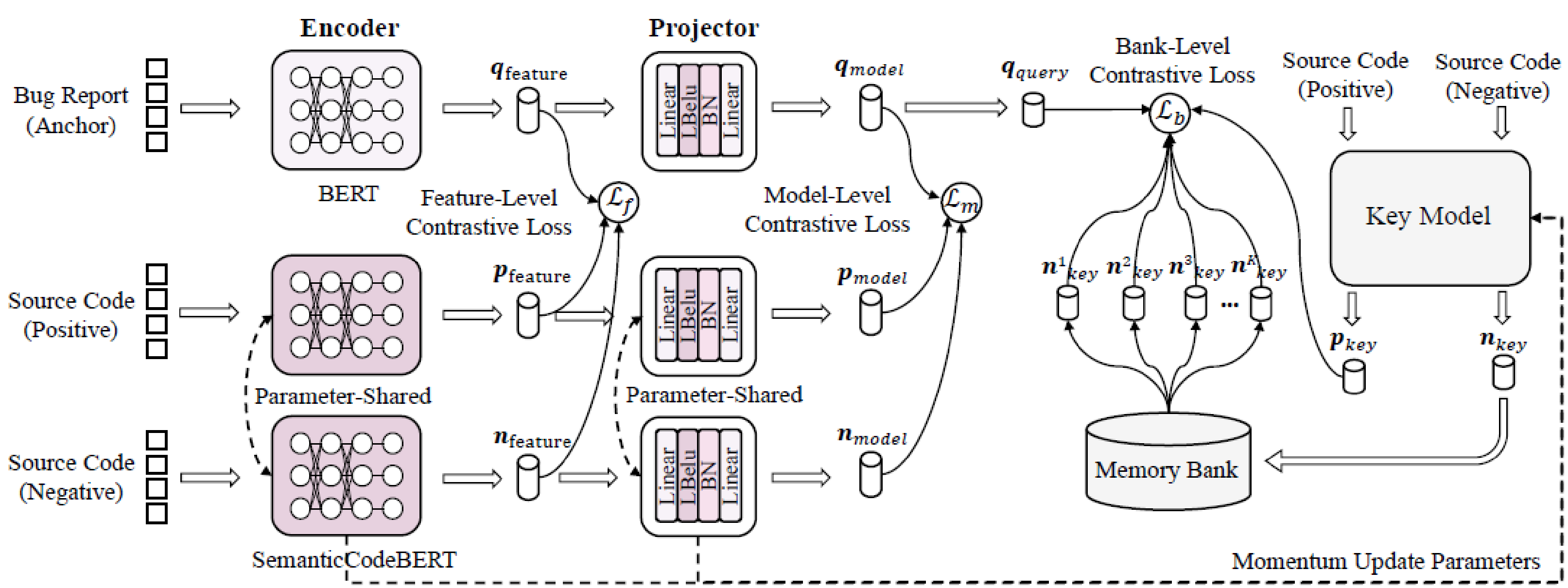
Given a set Q = $\{q1, q2, . . . , qM\}$ of M bug reports,
the bug localization task aims to discover more relevant changesets
From K = $\{k1, k2, . . . , kN\}$, a set including N changesets

For a bug report $q \in$ Q, a bug-inducing changeset $p \in$ K and
a not bug-inducing changeset $n \in$ K are selected to form a triplet (q, p, n).

The goal of learned similarity function s is to provide a high value
for $s\,(q, p)$ (between the anchor q and the positive sample p) and
a low value for $s\,(q, n)$ (between the anchor q and the negative sample n)

Representation Learning.

The proposed model consists of three parts, an encoder network, projector network, and momentum update mechanism with a memory bank.



Encoder Network

Bug reports consist of natural language descriptions and project changesets consist of programming language code.

This is a common and convenient behavior for text editors/IDEs which is missing. Would be nice to have it. Switching to keyboard and doing "shift+arrow down" seems like a lot more effort for achieving the same effect.

ImageData.getTransparencyMask - incorrect javadoc or implementation wrong

$$\begin{cases} \mathbf{q}_{feature} = \mathrm{BERT}(q_{tok}), \\ \mathbf{p}_{feature} = \mathrm{SemanticCodeBERT}(p_{tok}), \\ \mathbf{n}_{feature} = \mathrm{SemanticCodeBERT}(n_{tok}), \end{cases}$$

Input tokens obtained by tokenizers are refined into R^d dimension vector.

Projector Network

After the feature vectors are extracted, we use a multi-layer perception neural network as a projector to compress the vectors of bug reports and changesets into a compact shared embedding space

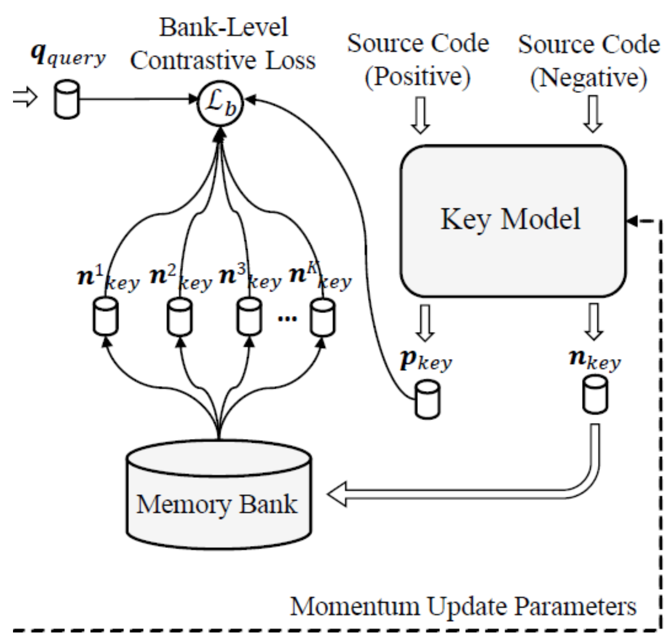
$$\begin{cases} \mathbf{q}_{model} = \mathbf{W}_b^2 norm(\phi(\mathbf{W}_b^1 \mathbf{q}_{feature})), \\ \mathbf{p}_{model} = \mathbf{W}_c^2 norm(\phi(\mathbf{W}_c^1 \mathbf{p}_{feature})), \\ \mathbf{n}_{model} = \mathbf{W}_c^2 norm(\phi(\mathbf{W}_c^1 \mathbf{n}_{feature})), \end{cases}$$

Feature Vectors are refined into $R^{d'}$ dimension vector.

Momentum Update Mechanism with Memory Bank

It is important to consider large-scale negative samples in contrastive learning for representations of changesets

We use memory bank to store rich changesets representation obtained from different batches for later contrast.



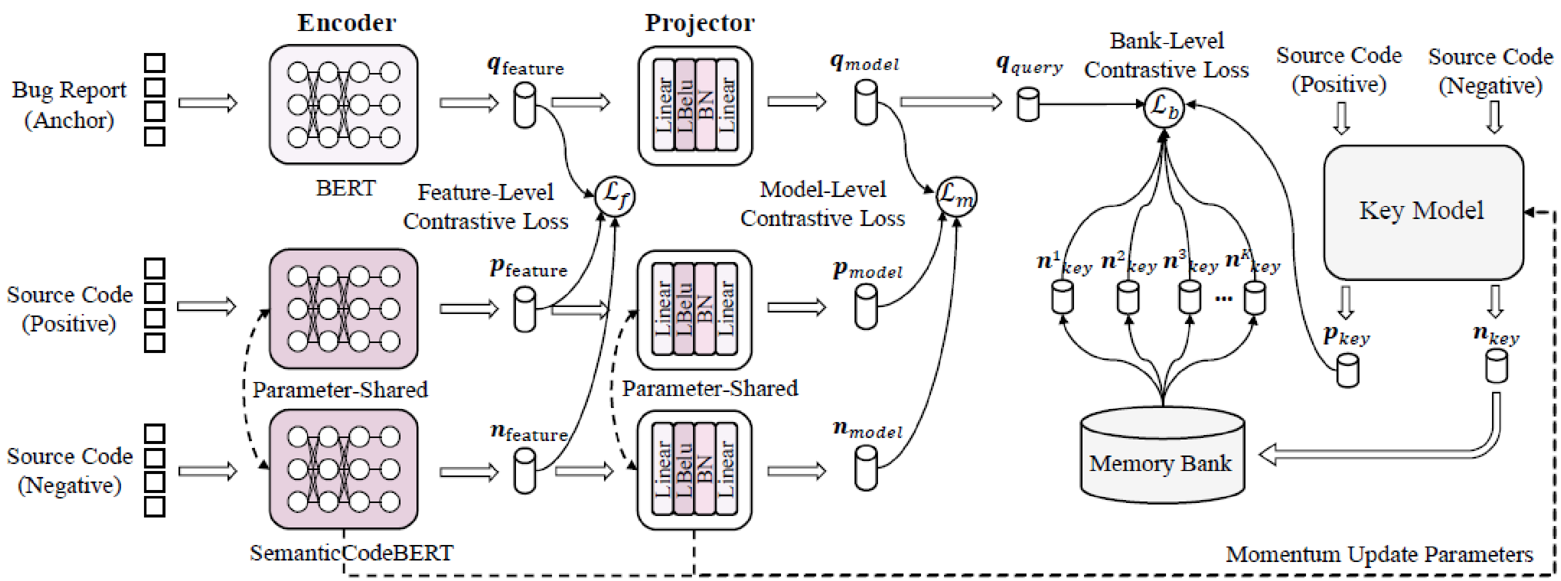
Similarity Estimation

We use the hierarchical contrastive loss to leverage the lower feature-level similarity, higher model-level similarity, and broader bank-level similarity for matching the bug report with relevant changesets.

1. **Feature-level Similarity (s_f^+, s_f^-):**
   - s_f^+: Cosine similarity between $q_{feature}$ (query feature) and $p_{feature}$ (positive feature).
   - s_f^-: Cosine similarity between $q_{feature}$ and $n_{feature}$ (negative feature).
2. **Model-level Similarity (s_m^+, s_m^-):**
   - s_m^+: Cosine similarity between q_{model} (query model) and p_{model} (positive model).
   - s_m^-: Cosine similarity between q_{model} and n_{model} (negative model).
3. **Bank-level Similarity (s_b^+, s_b^-):**
   - s_b^+: Cosine similarity between q_{query} (query) and p_{key} (positive key).
   - s_b^-: Cosine similarity between q_{query} and n_{key}^i (i-th negative sample in the Memory Bank).

HMCBL

Similarity Estimation



$$\mathcal{L}_f = -log \frac{exp(s^{f+}/\gamma)}{exp(s^{f+}/\gamma) + exp(s^{f-}/\gamma)}.$$

$$\mathcal{L}_m = -log \frac{exp(s^{m+}/\gamma)}{exp(s^{m+}/\gamma) + exp(s^{m-}/\gamma)}.$$

$$\mathcal{L}_b = -log \frac{exp(s^{b+}/\gamma)}{exp(s^{b+}/\gamma) + \sum_{k=1}^{K} exp(s_i^{b-}/\gamma)},$$

$$\mathcal{L} = \alpha_f \mathcal{L}_f + \alpha_m \mathcal{L}_m + \alpha_b \mathcal{L}_b,$$

Evaluation Metrics

Precision@K ($P@K$): $P@K$ evaluates how many of the top-K changesets in a ranking are relevant to the bug report

$$P@K = \frac{1}{|B|} \sum_{i=1}^{|B|} \frac{|Rel_{B_i}|}{K}.$$

Mean Average Precision (MAP): MAP quantifies the ability of a model to locate all changesets relevant to a bug report

$$AvgP = \sum_{j=1}^{M} \frac{P@j \times pos(j)}{N}.$$

$$MAP = \frac{1}{|B|} \sum_{i=1}^{|B|} \frac{1}{AvgP_{B_i}},$$

Evaluation Metrics

Mean Reciprocal Rank(MRR): MRR quantifies the ability of a model to locate the first relevant changeset to a bug report

$$MRR = \frac{1}{|B|}\sum_{i=1}^{|B|}\frac{1}{1stRank_{B_i}}.$$

Dataset

Table 1: Six projects used for evaluation.

Dataset	Bugs	Changesets		
		Commits	Files	Hunks
AspectJ	200	2,939	14,030	23,446
JDT	94	13,860	58,619	150,630
PDE	60	9,419	42,303	100,373
SWT	90	10,206	25,666	69,833
Tomcat	193	10,034	30,866	72,134
ZXing	20	843	2,846	6,165

Compared Models

BLUiR : A structured IR-based fault localization tool, which builds AST to extract the program constructs of each source code file

FBL-BERT: The state-of-the-art approach for automatically retrieving bug-inducing changesets given a bug report, which uses the popular BERT model to more accurately match the semantics in the bug report text with the bug-inducing changesets

GraphCodeBERT: A pre-trained model that considers data flow to better encode the relation between variables.

UniXcoder: An unified cross-modal pre-trained model, which leverages cross-modal information like Abstract Syntax Tree and comments to enhance code representation.

Experiment

Table 2: Retrieval performance of different models.

Projects	Technique	*Commits–*					*Files–*					*Hunks–*				
		MRR	MAP	P@1	P@3	P@5	MRR	MAP	P@1	P@3	P@5	MRR	MAP	P@1	P@3	P@5
ZXing	*BLUiR*	0.077	0.016	0.071	0.024	0.014	0.073	0.023	0.000	0.024	0.014	0.056	0.035	0.000	0.071	0.086
	FBL-BERT	0.155	0.061	0.100	0.133	0.120	0.212	0.163	0.100	0.133	0.220	0.328	0.210	0.200	0.233	0.240
	GraphCodeBERT	0.189	0.118	0.143	0.143	0.118	0.280	0.155	0.214	0.143	0.214	0.346	0.118	0.225	0.111	0.067
	UniXcoder	0.354	0.167	0.414	0.171	0.120	0.359	0.143	0.333	0.224	0.200	0.331	0.164	0.214	0.261	0.282
	Ours	**0.439**	**0.226**	**0.429**	**0.250**	**0.225**	**0.421**	**0.185**	**0.357**	**0.226**	**0.271**	**0.422**	**0.212**	**0.333**	**0.444**	**0.400**
PDE	*BLUiR*	0.009	0.001	0.000	0.000	0.000	0.018	0.003	0.000	0.008	0.005	0.024	0.005	0.000	0.008	0.010
	FBL-BERT	0.103	0.013	0.067	0.033	0.027	0.260	0.079	0.167	0.128	0.151	0.288	0.093	0.200	0.144	0.127
	GraphCodeBERT	0.180	0.042	0.142	0.087	0.058	0.264	0.094	0.167	0.129	0.148	0.284	0.074	0.206	0.124	0.129
	UniXcoder	0.178	0.029	0.095	0.063	0.072	0.267	0.090	0.167	0.135	0.129	0.289	0.102	0.212	0.144	0.129
	Ours	**0.248**	**0.045**	**0.190**	**0.103**	**0.076**	**0.274**	**0.095**	**0.214**	**0.137**	**0.160**	**0.294**	**0.134**	**0.286**	**0.182**	**0.160**
AspectJ	*BLUiR*	0.016	0.013	0.007	0.014	0.015	0.098	0.065	0.028	0.076	0.108	0.086	0.048	0.007	0.017	0.159
	FBL-BERT	0.107	0.061	0.058	0.080	0.083	0.176	0.085	0.154	0.095	0.097	0.183	0.093	0.173	0.111	0.099
	GraphCodeBERT	0.172	0.065	0.167	0.065	0.060	0.178	0.071	0.167	0.065	0.060	0.188	0.086	0.167	0.120	0.116
	UniXcoder	0.270	0.148	0.245	0.160	0.158	0.209	0.119	0.167	0.140	**0.152**	0.250	0.134	0.250	0.150	0.138
	Ours	**0.309**	**0.169**	**0.278**	**0.198**	**0.196**	**0.272**	**0.148**	**0.250**	**0.157**	0.146	**0.262**	**0.143**	**0.250**	**0.161**	**0.163**
JDT	*BLUiR*	0.019	0.001	0.015	0.005	0.003	0.027	0.003	0.000	0.010	0.012	0.033	0.005	0.000	0.005	0.009
	FBL-BERT	0.118	0.016	0.064	0.043	0.030	0.403	0.060	0.319	0.184	0.128	0.429	0.062	0.319	0.195	0.167
	GraphCodeBERT	0.125	0.022	0.061	0.035	0.030	0.423	0.058	0.308	0.179	0.118	0.385	0.041	0.231	0.179	0.118
	UniXcoder	0.182	0.018	0.182	0.061	0.038	0.434	0.062	0.379	0.166	0.131	0.364	0.045	0.288	0.182	0.123
	Ours	**0.306**	**0.026**	**0.288**	**0.096**	**0.064**	**0.489**	**0.080**	**0.462**	**0.195**	**0.167**	**0.443**	**0.088**	**0.322**	**0.206**	**0.167**
SWT	*BLUiR*	0.005	0.001	0.000	0.000	0.000	0.020	0.003	0.016	0.005	0.006	0.014	0.001	0.000	0.000	0.013
	FBL-BERT	0.067	0.015	0.023	0.027	0.026	0.555	0.131	0.535	0.233	0.173	0.526	0.131	0.488	0.217	0.164
	GraphCodeBERT	0.105	0.018	0.048	0.026	0.022	0.535	0.137	0.525	0.220	0.175	0.536	0.132	0.516	0.220	0.159
	UniXcoder	0.129	0.035	0.107	0.106	0.063	0.548	0.149	0.524	0.233	0.183	0.535	0.143	0.535	0.205	0.179
	Ours	**0.283**	**0.085**	**0.159**	**0.177**	**0.170**	**0.560**	**0.153**	**0.540**	**0.249**	**0.192**	**0.540**	**0.147**	**0.540**	**0.228**	**0.179**
Tomcat	*BLUiR*	0.007	0.002	0.000	0.002	0.002	0.014	0.003	0.000	0.010	0.007	0.014	0.005	0.000	0.012	0.013
	FBL-BERT	0.141	0.055	0.062	0.077	0.088	0.463	0.114	0.381	0.222	0.183	0.482	0.129	0.412	0.216	0.182
	GraphCodeBERT	0.253	0.062	0.188	0.104	0.084	0.287	0.067	0.271	0.104	0.080	0.395	0.118	0.363	0.216	0.211
	UniXcoder	0.328	0.057	0.338	0.120	0.084	0.364	0.065	0.353	0.125	0.085	0.396	0.097	0.378	0.139	0.118
	Ours	**0.386**	**0.073**	**0.360**	**0.135**	**0.107**	**0.487**	**0.122**	**0.406**	**0.247**	**0.232**	**0.484**	**0.132**	**0.423**	**0.225**	**0.211**

Observations

First, compared with the traditional bug localization method which relies on more direct term matching between a bug report and a changeset, the neural network methods perform better by obtaining semantic representations for the calculation of similarity

Second, our proposed method outperforms the state-of-the-art method (FBLBERT) by a clear margin.

Third, compared with GraphCodeBERT and UniXcoder, our model using SemanticCodeBERT as a changeset encoder consistently achieves better performance in almost all experimental configurations.

Ablation study

Table 3: Ablation study of pre-training tasks of Semantic-CodeBERT with Semantic Flow Graph (SFG).

Dataset	Pre-training Tasks	MRR	MAP	P@1	P@3	P@5
ZXing	-w/	0.189	0.118	0.143	0.143	0.118
	-w/ N.& E.	0.372	0.102	0.333	0.111	0.067
	-w/ N.& E.& T.& R.	**0.439**	**0.226**	**0.429**	**0.250**	**0.225**
PDE	-w/	0.180	0.042	0.142	0.087	0.058
	-w/ N.& E.	0.219	0.032	0.143	0.076	0.072
	-w/ N.& E.& T.& R.	**0.248**	**0.045**	**0.190**	**0.103**	**0.076**
AspectJ	-w/	0.172	0.065	0.167	0.065	0.060
	-w/ N.& E.	0.289	0.158	0.250	0.184	0.170
	-w/ N.& E.& T.& R.	**0.309**	**0.169**	**0.278**	**0.198**	**0.196**
JDT	-w/	0.125	0.022	0.061	0.035	0.030
	-w/ N.& E.	0.139	0.021	0.095	0.044	0.048
	-w/ N.& E.& T.& R.	**0.306**	**0.026**	**0.288**	**0.096**	**0.064**
SWT	-w/	0.105	0.018	0.048	0.026	0.022
	-w/ N.& E.	0.197	0.058	0.063	0.085	0.141
	-w/ N.& E.& T.& R.	**0.283**	**0.085**	**0.159**	**0.177**	**0.170**
Tomcat	-w/	0.253	0.062	0.188	0.104	0.084
	-w/ N.& E.	0.300	0.048	0.346	0.113	0.077
	-w/ N.& E.& T.& R.	**0.386**	**0.073**	**0.360**	**0.135**	**0.107**

Ablation study

Table 4: Ablation study of Hierarchical Momentum Contrastive Bug Localization (HMCBL) technique, where GCBERT and SCBERT are short of GraphCodeBERT and SemanticCodeBERT.

Technique	Dataset	MRR	MAP	P@1	P@3	P@5
BERT -w/o *HMCBL* (*FBL-BERT*)	ZXing	0.155	0.061	0.100	**0.133**	**0.120**
	PDE	0.103	0.013	0.067	0.033	0.027
	AspectJ	0.107	0.061	0.058	0.080	0.083
	JDT	0.118	0.016	**0.064**	0.043	0.030
	SWT	0.067	**0.015**	0.023	**0.027**	**0.026**
	Tomcat	0.141	0.055	0.062	0.077	0.088
GCBERT -w/o *HMCBL*	ZXing	0.162	0.106	0.143	0.095	0.086
	PDE	0.167	0.018	0.119	0.071	0.045
	AspectJ	0.123	0.067	0.076	**0.073**	**0.084**
	JDT	0.120	0.022	0.061	0.035	**0.036**
	SWT	0.090	**0.019**	0.048	0.021	0.022
	Tomcat	0.151	0.035	0.059	0.064	0.063
SCBERT -w/o *HMCBL*	ZXing	0.222	0.112	0.143	0.190	0.150
	PDE	0.230	**0.049**	0.142	0.095	0.069
	AspectJ	0.271	0.148	0.250	0.161	0.165
	JDT	0.217	**0.051**	0.136	**0.111**	**0.091**
	SWT	0.250	0.062	0.095	0.167	**0.185**
	Tomcat	0.285	0.053	0.265	0.092	0.069
BERT -w/ *HMCBL*	ZXing	**0.179**	**0.040**	**0.143**	0.095	0.061
	PDE	**0.156**	**0.032**	**0.119**	**0.063**	**0.051**
	AspectJ	**0.162**	**0.097**	**0.118**	**0.141**	**0.149**
	JDT	**0.128**	**0.017**	0.030	**0.070**	**0.100**
	SWT	**0.082**	0.013	**0.048**	0.024	0.021
	Tomcat	**0.235**	**0.055**	**0.169**	**0.098**	**0.096**
GCBERT -w/ *HMCBL*	ZXing	**0.189**	**0.118**	**0.143**	**0.143**	**0.118**
	PDE	**0.180**	**0.042**	**0.142**	**0.087**	**0.058**
	AspectJ	**0.172**	**0.065**	**0.167**	0.065	0.060
	JDT	**0.125**	**0.022**	**0.061**	**0.035**	0.030
	SWT	**0.105**	0.018	**0.048**	**0.026**	**0.022**
	Tomcat	**0.253**	**0.062**	**0.188**	**0.104**	**0.084**
SCBERT -w/ *HMCBL*	ZXing	**0.439**	**0.226**	**0.429**	**0.250**	**0.225**
	PDE	**0.248**	0.045	**0.190**	**0.103**	**0.076**
	AspectJ	**0.309**	**0.169**	**0.278**	**0.198**	**0.196**
	JDT	**0.306**	0.026	**0.288**	0.096	0.064
	SWT	**0.283**	**0.085**	**0.159**	**0.177**	0.170
	Tomcat	**0.386**	**0.073**	**0.360**	**0.135**	**0.107**