This document is a supplementary document with the full results from the main study "Metamorphic Testing of Deep Code Models: A Systematic Literature Review".

1 TRANSFORMATIONS

This section shows the full list of transformations along with examples and usages in the primary studies. Several things should be noted about the transformations. First of all, all transformations are semantic-preserving, except some transformations applied by Bielik and Vechev [1] for type inference. However, these transformations are still label-preserving. Second, we chose to group renaming of parameters, local variables, global variables, and function names into one kind of transformation: identifier renaming. This is because it was in many cases not clear exactly what kind of identifiers were renamed, and the process of renaming an identifier is practically identical for each of them. Third, most transformations are applicable to a lot of languages while some are only applicable to specific languages. This is due to the diversity of programming languages used in each of the papers. Fourth, the examples, names, and descriptions are taken from the source papers if they are sufficiently clear/distinct. We renamed some transformations and modified some examples or descriptions for clarity. Finally, we prefer displaying the transformations with "description/examples" and "used by". Our way makes it much more convenient to find the most common transformations and investigate their workings, whereas the way that other reviews displays them may lead to ambiguity, as different papers may use the same names for different transformations.

Author 1 and Author 2: Each record is carefully double-checked. Accepted records are marked accordingly, while records that required discussion due to doubt or disagreement are indicated with X.

Author 3 and Author 4: The verification process for *Tasks, Metrics, Models, and Datasets* follows a different approach. Any doubtful records have been thoroughly reviewed and discussed collaboratively with Authors 3 and 4 to ensure consistency and accuracy.

Notes from Author 3 and Author 4:

- First, we check that the target exists in the real world.
- Then, we (Authors 3 and 4) randomly check records to understand the mindset of the first author.
- The mentioned dimensions should not be discussed in *Background*, *Related Works*, or *Future Works*; instead, they should be **used**.

Warning: This file contains the initial dataset, encompassing all records, including some that were marked as doubtful. Some records have undergone modifications following discussions and double-checking by the authors for the final paper. Therefore, minor differences may exist between the records in this file and the final tables presented in the published paper.

Transformation	Description/Example	Used by
Unfold shorthand expression	Replace instruction with equivalent instruction. E.g. `a += b` → `a = a + b`	[2, 3, 4, 5]
Swap operands (asymmetric)	Swap the operands of relational operators. E.g. `a > b` → `b < a`	[6, 7, 5]
Remove unused code	Remove variables, functions, and included libraries that are never used or called.	[6, 8, 9]
Split/aggregate declarations	Split a one-line declaration into multiple lines or aggregate multiple declarations into one line. E.g. `int a, b` → `int a; int b;`	[6, 3]
Split/aggregate declaration and initialization	Split a one-line initialization into multiple lines or aggregate multiple lines into one line. E.g. `int a = 1` → `int a; a = 1`	[3, 5]
Swap operands (symmetric)	Swap the operands of symmetric operators. E.g. `a + b` → `b + a`	[2, 10]
Exchange suffix/prefix	Change suffix/prefix operator. E.g. `i++` → `++i`	[7, 3]
Add neutral element	E.g. `true` → `0^1 == 1`	[11, 12]
Separate/attach elaborated type declaration	Split a composite type declaration into an elaborated type declaration or attach an elaborated type declaration to its definition. E.g. `struct mynode {int x,y;} a;` → `struct mynode {int x,y;}; struct mynode a;`	[6]
Undo type alias	Replace the declared type alias with the original type name. E.g. `typedef long long LL; LL a;` → `long long a;`	[6]
Use alternative tokens	Replace operators or symbols with their alternative tokens. E.g. (`&&` → `and`), or (`!=` → `not_eq`)	[6]
Use converse-negative expressions	Rewrite condition statements into converse-negative expressions using inverse operators. E.g. `a < b` → `!(a >= b)`	[6]
Use equivalent computations	Use equivalent mathematical expressions. E.g. `a = b + c` → `a = b - (-c)`	[6]
Move expression into variable (common name)	Move argument into variable declaration. Give the variable a common variable name. E.g. `max(a.length,b)` → `c = a.length; max(c,b);`	[2]

Move constant into variable (name not specified) X Expression Unfolding	Move constant into variable. The name of the new variable is not specified. E.g. `max(0,b)` → `c = 0 max(c,b);`	[2]
Toggle operators	Rearrange/change operators. E.g. `a - b + c` → `a - (b - c)`	[2]
Replace constant with arithmetic expression	E.g. 8 → `a - b` where a equals 16 and a equals 8. Variable names not specified.	[5]
Add new variable	Take an existing variable and declare a new variable of the same type. Assign the existing variable to the new variable and update the rest of the program. The new variable has a name similar to the type of the variable. E.g. `Calendar c = Calendar(...)` → `Calendar c = Calendar(...); Calendar calendar = c;`	[13]
Duplicate assignment	E.g. `a = 6` → `a = 6; a = 6;`	[14]

Table 1. Trivial Transformations

Transformation	Description/Examples	Used by
Rename identifier based on strategy	Use special strategies to choose identifier replacement.	[7, 15, 16, 17, 18, 4, 19, 20, 21, 22, 23]
Rename Identifier similarly	Rename identifier with semantically close word.	[15, 24, 25]
Rename identifier	Not specified what the identifier is replaced with.	[1, 5]
Rename identifier to template value	E.g. `int a` → `int var1`	[26, 8, 27]
Rename identifier random other identifier	Replace identifier with random other identifier from dataset.	[28, 29, 8]
Obfuscate identifier	Replace identifier with random sequence of characters. E.g. `int test_counter;` → `int Bq41wD7l;`	[6, 14]
Rename identifier to first letter of the first word	-	[13]
Abbreviate identifier	Replace identifier by each letter of each word in identifier. Maybe not each letter, first letter	[13]
Change first letter to lowercase Change first letter of class name to lowercase	-	[13]

Rename Single-letter Identifier	Rename single-letter identifier to random other single letter.	[25]
Rename identifier with short token	-	[9]
Rename identifier to to context value	E.g. `int count` → `int count_var_1`	[26]
Rename identifier with common name	Replace identifier with common identifier names from other programs.	[2]
Rename identifier with word in vocabulary	-	[30, 11, 12],
Rename identifier subtoken similarly	Rename identifier subtoken with similar subtoken.	[10]

Table 2. Renaming Transformations

Transformation	Description/Examples	Used by
Delete comment X8	-	[11, 12, 5]
Add comment X8	-	[11, 12]
Move comment X8	-	[11, 12]
Move word in comment	-	[29]
Delete word in comment	-	[29]
Copy word in comment	Copy word and insert after first word.	[29]

Table 3. Comment Transformations

Transformation	Description/Examples	Used by
Convert between bool literals and int literals	Use `1` for `true` and `0` for `false`.	[6, 9, 8]
Replace true/false	E.g. `true` → `a == a` (what `a` is is not specified)	[16, 26, 17, 18]
Use cast expressions	Although cast expressions are normally used to explicitly convert a data value to a new type, we can utilize the property to transform the same types like in `b = (int) a`, where `a` is an `int` variable.	[6, 13]
Convert string literals to char arrays	Use a char array to initialize a string variable.	[6, 8]
Convert char literals into ASCII values	E.g. `char c='A'` → `char c=65`	[6]

Use typeid expression	Use the keyword "typeid" to dynamically decide data types. E.g. `int b = 0` ⟶ `typeid (a) b = 0` where a is an already defined int variable.	[6]
Convert int literals into expression	Get an int literal by using mathematical expression such as addition, subtraction, multiplication and division. E.g. `int b = 8` ⟶ `int b = 2 * 4`	[6]
Convert integers into hexadecimal numbers	E.g. `int b = 48` ⟶ `int b = 0x30`.	[6]
Convert typedef	Convert type to new type via typedef or remove typedef.	[8]
Promote integral type	Promote integral type to next higher type. E.g. `int` ⟶ `long`	[8]
Convert float to double	-	[8]
Convert array to vector	Convert array to C++ vector.	[8]
Substitute number, string, or boolean	Only applicable for the task of type inference. E.g. `2` ⟶ `7` or `"get"` ⟶ `"load"`	[1]
Add and subtract value	E.g. `a = 6` ⟶ `a = 6 + 8 - 8`	[14]
Add zero	E.g. `a = 6` ⟶ `a = 6 + 0`	[14]
Constant folding	Fold arithmetic expression into constant. E.g., `int b = 2 * 4` ⟶ `int b = 8`	[9]

Table 4. Data Transformations

Transformation	Description/Examples	Used by
Convert for-statement to while-statement	`for(init;condition;update;){body;}` ⟶ `init; while(condition){body;update;}` and vice versa.	[6, 7, 2, 10, 8, 27, 3, 4, 5]
Convert if-else to switch-case Reverse also?	`if(condition) {body;}` ⟶ `switch(condition){case true: body;}` and vice versa.	[6, 2, 27, 3, 5]
Swap independent statements	Change the order of two statements with no dependencies in the control flow graph.	[2, 29, 27, 5]
Split conditions of if-statements	Split the conditions of an if-statement into multiple statements when the logical operator is one of `||`, &&, <=, >=.	[6, 8, 3]
Convert if-else to conditional expression	`if(condition) {a;} else {b;}` ⟶ `condition ? a : b` and vice versa.	[6, 3]

Wrap expression in if statement	Wrap in `if(true)` or `if(false) else`.	[11, 12]
Wrap expression in lambda	Wrap expression into identity-lambda function (including function call).	[11, 12]
Swap if-else bodies	`if(condition){a;} else {b;}` → `if(!condition){b;} else {a}`.	[6]
Split if-else and vice versa	`if(condition){a;} else {b;}` → `if(condition){a;} if(!condition) {b;}`.	[4]
Transform if-else to if-elseif and vice versa	`if(condition){a;} else {b;}` → `if(condition){a;} elseif(!condition) {b;}`.	[5]
Transform do-while to while	-	[5]
Wrap in conditional	`expr` → `condition ? expr : expr;`.	[1]
Wrap in array	`expr` → `[expr, expr][const]`.	[1]
Substitute boolean	E.g. `true` → `!false`.	[7]
Replace boolean	E.g. `true` → `false` and propagate this change. Negate all uses of this variable.	[27]
Unroll while loop	Unroll while loop exactly one step.	[16]
Wrap in try-catch	Wrap expression in try-catch. The base class `Exception` is caught.	[16]
Wrap in do-while(false)	Wrap expression in `do-while(false)`.	[10]
Move declaration into or out of control statement X50 to 63	`for(int i = 0, i < n; i++)` → `int i; for(i = 0, i < n; i++)`.	[8]

Table 5. Control Flow Transformations

Transformation	Description/Examples	Used by
Add Function arguments (name from dictionary)	Add an argument to the function. The argument name is taken from a dictionary.	[11, 12]
Add Function arguments (name from program under test)	Add an argument to the function. The argument name is taken from the program under test.	[11, 12]
Delegate to function (name from dictionary) Delegate to function X8	Extract an expression to a function, invoke the function instead of the expression. The function name is taken from a dictionary.	[11, 12]
Delegate to function (name from program under test)X8	extract an expression to a function, invoke the function instead of the expression. The function name is taken from the program under test.	[11, 12]

Add Function arguments (name not specified) Function Parameters	Add extra arguments which are simply initialized without changing the function output. The names of these arguments are not specified.	[6, 1]
Reorder function arguments	`saveText(string s, ofstream outfile)` → `saveText(ofstream outfile, string s)`	[6]
Merge function arguments	Merge function arguments into a struct type and then use them as struct members. E.g. `int add (int a, int b) {return a + b}` → `int add (args ab) {return ab.a + ab.b}`	[6]
Convert statements into functions	Remove initialization and assignment statements into a function, whose arguments are pointers to the influenced variables. The return type is `void`. The new function name is not specified.	[6]
Convert binary expressions into functions	Extract variables in the binary expressions as function arguments and returns the computing result. E.g. `c = a + b` → `c = add(a, b)`	[6]
Merge functions	Merge two functions that have the same return type into a combined function and add a switching argument to decide which sub-function is called.	[6]
Hide API calls	Wrap API calls into a user-defined function. E.g. call `freopen` in the body of `my_open` function.	[6]
Delegate to function (name not specified)	Extract a block to a function, invoke the function instead of the block. The function name is not specified.	[8]

Table 6. Function Transformations

s

Transformation	Description/Examples	Used by
Swap C/C++ libraries for reading and writing	E.g. use `cin`/`cout` instead of `printf`. <iostream> to <stdio.h>	[8, 3]
Use `stdin`/`stdout` instead of files	Use console input/output instead of files.	[8]
Enable/disable sync between C/C++ streams	-	[8]

Table 7. API Transformations

Transformation	Description/Examples	Used by

Add dead code	Add `if(false) {...}` where ... is filled in based on strategy.	[16, 10, 17, 18, 19]
Add print statement	Add `print(...)` where ... is filled in based on strategy.	[16, 26, 17, 18, 19]
Add unused variable (strategy)	Add unused variable based on strategy.	[10, 19, 20]
Add unused variable (name from dictionary)	Add unused variable with random name from dictionary.	[11, 12]
Add unused variable (name from program under test)	Add unused variable with random name from program under test.	[11, 12]
Add libraries	Add extra libraries used in other code snippets.	[6, 8]
Transfer code	Insert code from another program in a "dead" place.	[31, 2]
Insert unreachable loop or branch	Insert `if(false)` or `while(false)` block.	[32, 19]
Insert or delete empty statement/branch/loop	Insert or delete statement from insertable statement list (this list is not specified).	[32, 30]
Add temp variables	Introduce a temp variable for expressions in array indexes, return statements, if statements, loop statements, and function arguments. E.g, `return 0;` → `int t = 0; return t;`	[6]
Add global declarations (name not specified)	Add global variables and initialize them in the main function. The name of these global declarations is not specified.	[6]
Add redundant operands	Use redundant operands for addition, subtraction, multiplication, and division expressions. E.g. `a * b` → `a * b * c/2` where `c` is initialized to 2. The name of a new variable is not specified.	[6]
Add type alias	Add extra type alias with the corresponding variable declaration.	[6]
Add function declaration in classes	Declare a function with a class without defining it.	[6]
Add unused assignment Dead Code Insertion with Unused Variable Assignment	Add assignment to new variable with name taken from dataset.	[29]
Add unused variable with type information	Add unused variable with random type, call it `unused*TYPE*`.	[13]
Add unused variable (random string)	Add unused variable with random string as name.	[14]

Add unused variable (name not specified)	The name of the new variable is not specified.	[27]
Add global declarations (from dataset)	Add global declarations from dataset of global declarations.	[8]
Add typedef	Add typedef from dataset.	[8]
Add method arguments	Add unused random binary expressions to a method call.	[1]
Add side-effect free expression	Add random binary expression without side effects.	[1]
Add object expression	$\emptyset \longrightarrow \{x : y, z : expression\}$	[1]
Add template dead code	Add dead code from template. E.g. `if(false): TempVar = a` where only `a` is a local variable from outside the scope of the `if` statement.	[26]
Add context dead code	Add dead code from context. E.g. `if( b != b): b = a` where `a` and `b` are local variables	[26]
Add unused assignment (template+strategy)	Add assignment of template variable with value based on strategy.	[32]
Add junk code	Not specified.	[5]

Table 8. Dead code insertion Transformations

Transformation	Description/Examples	Used by
Add or remove whitespace X8	-	[11, 12]
Add or delete compound statement Also Move ?	For single line control flow statements, remove or insert `{}`.	[8]
Add explicit return to main method	-	[8]
Replace literal return with the variable return	Variable name not specified.	[8]
Auto-format code	-	[9]
Delete print statement	-	[5]

Table 9. Misc Transformations

2 TASKS AND METRICS

This section shows the metric used for each task and what primary studies use them.

We show two kinds of metrics:

- Metrics to compare model **output**. Some metrics are more suitable to compare the outputs of some tasks than others. For example, one could argue that code summarization output for the transformed does not have to be exactly equal to the output for the original label, but that a semantically similar summary would be sufficient. However, for clone detection, a binary yes/no could be more appropriate.
- Metrics to compare model **input**. To judge the significance of a sequence of transformations, several metrics were used by some papers to make some judgment about the difference between the transformed snippet and the original snippet.

For each metric, a source defining the metric is added, except for Equality and Equality to the target label. Many metrics like adversarial success rate and robustness exist, but these are further interpretations based on output equality. If papers use this kind of metric, we classify the metric under equality. Only metrics for the robustness evaluations are shown.

Task	Metric	Used by
Method name prediction	F1 [7]	[12, 7, 28, 16, 9, 10, 27, 17]
Method name prediction	Precision [7]	[27]
Method name prediction	Recall [7]	[27]
Method name prediction	MRR [33]	[12]
Method name prediction	Equality	[7, 10, 27, 17]
Method name prediction	Equality to target label	[20]
Code summarization	F1 [7]	[18, 14]
Code summarization	BLEU [34]	[11, 15, 10, 14]
Code summarization	BLEU [34] decrease over 50%	[22]
Code summarization	ROUGE [35]	[15]
Code summarization	Equality	[10]
Code summarization	METEOR [36]	[15]
Code summarization	Prediction confidence decrement [22]	[22]
Code completion	F1 [7]	[18]
Code completion	BLEU [34]	[26]
Code completion	Levenshtein edit similarity [37]	[26]
Clone detection	F1 [7]	[18, 10, 30, 24, 5]
Clone detection	Precision [7]	[5]
Clone detection	Recall [7]	[5]
Clone detection	CodeBLEU [38]	[32]
Clone detection	Prediction confidence decrement [4]	[4, 22]

Clone detection	Equality	[10, 32, 4, 19, 23, 22]
Clone detection	Area under ROC curve [39]	[9]
Clone detection	Average precision [7]	[9]
Clone detection	Mean average precision [40]	[29]
Vulnerability detection	F1 [7]	[31, 30]
Vulnerability detection	Equality	[4, 23]
Vulnerability detection	Prediction confidence decrement [4]	[4]
Vulnerability detection (not explained how they apply this)	P/R-AUC [41]	[31]
Code translation	BLEU [34]	[29]
Code translation	BLEU [34] decrease over 50%	[22]
Code translation	Prediction confidence decrement [22]	[22]
Comment generation	BLEU [34]	[25]
Comment generation	ROUGE [35]	[25]
Comment generation	METEOR [36]	[25]
Defect detection	CodeBLEU [38]	[32]
Defect detection	Equality	[32, 4, 19, 22]
Defect detection	Prediction confidence decrement [4]	[4, 22]
Authorship Attribution	CodeBLEU [38]	[32]
Authorship attribution	Equality	[32, 8, 4, 19, 23, 22]
Authorship Attribution	Prediction confidence decrement [4]	[4, 22]
Authorship attribution	Equality	[6]
Type inference	Equality	[1]
Functionality classification	Equality	[10, 3, 4, 21]
Functionality classification	Equivalence with target label	[20]
Functionality classification	Prediction confidence decrement [4]	[4]
Code repair	Different semantics after repair[13]	[13]
Code repair	Successful/failed fix[13]	[13]
Misused variable detection X	Equality to target label	[7]
Code search	MRR [33]	[29]

Table 10. Robustness metrics for model output

Metric	Used by
Jaccard distance [42]	[11]
Amount of transformations applied	[6]
Percentage of changed lines of code	[6]
Time used per adversarial instance	[6]
Amount of model queries	[32, 23, 22]
Ratio of perturbation [32]	[32]
Validity [3]	[3]
Variable change rate [23]	[23]
Average semantic similarity	[22]
Average modification rate	[22]

Table 11. Metrics for model input

3 MODELS

This section shows the full list of all models under test and the primary studies that conducted experiments with these models.

Model	Used by
CodeBERT [43]	[44, 45, 46, 11, 2, 28, 10, 29, 32, 4, 19, 23, 30, 24, 22]
LSTM [47]	[48, 1, 9, 8, 4, 30, 21, 24, 5, 28]
GraphCodeBERT [49]	[44, 45, 46, 50, 10, 29, 32, 4, 19, 23, 24, 22]
Seq2seq [51]	[52, 53, 48, 7, 15, 16, 18, 27, 17, 25, 28]
Code2vec [54]	[55, 56, 50, 12, 2, 9, 10, 27, 14, 20]
Code2seq [57]	[55, 58, 50, 16, 9, 10, 17, 14]
CodeT5 [59]	[46, 50, 10, 32, 19, 22]
GGNN [60]	[50, 1, 2, 27, 20]
ASTNN [61]	[28, 3, 30, 21, 5]
DeepTyper [62]	[1, 9]
GPT Based [63]	[52, 50, 46, 10, 26]
TBCNN [64]	[28, 30]
ContraBERT [29]	[29]
PLBART [65]	[46, 10]
GCN [66]	[67, 1]
GNT [68]	[1]

Their own model, based on [69]	[31]
Tufano [70]	[13]
CoCoNut [71]	[13]
SequenceR [72]	[73, 13]
Recoder [74]	[73, 13]
CodeNN [75]	[28]
on Devign [61]	[28]
Transformer [76]	[9]
RoBERTa [77]	[9]
GPT-3 Codex [78]	[9]
ChatGPT 3.5[79]	[10]
ChatGLM [80]	[10]
GitHub Co-pilot [81]	[26]
CodeParrot [82]	[26]
GPT-Neo [83]	[26]
GPT-J [84]	[26]
CodeGen [85]	[26]
RFC [86]	[6]
RNN-RFC [86]	[6]
CFCS [87]	[6]
NeuralCodeSum [51]	[14]
GNN-FILM [88]	[20]
GRU [89]	[30]
LSCNN [90]	[30]
CLDH [91]	[30]
TBCCD [92]	[5]
CSCG [93]	[25]
Rencos [94]	[25]

Table 12. Models under test

4 DATASETS

This section shows the benchmarks used for each primary study.

Dataset	Used by

CodeSearchNet [95]	[53, 44, 11, 2, 28, 16, 9, 26, 10, 29, 18, 7]
Online Judge [64]	[28, 3, 4, 30, 21, 24]
BigCloneBench [96]	[10, 32, 4, 19, 23]
java-small [54]	[55, 58, 12, 7, 16, 27, 14]
Devign dataset [61]	[28, 32, 4, 23, 19]
java-large [54]	[55, 58, 2, 27, 17, 20]
Py150k [97]	[52, 58, 16, 18, 17]
Google code Jam dataset [98]	[32, 4, 23]
Online Judge Clone [91]	[30, 24, 5]
java-med [54]	[55, 58, 10, 27]
CodeChef [99]	[4, 30]
Java dataset by [100]	[15, 25]
Java250 [101]	[10]
CodeXGlue [102]	[29]
Compiled their own from Google Code Jams [103] 2012-2017	[6]
Compiled their own from Google Code Jam [103] 2017	[8]
Compiled their own from Google Code Jam [103] (Not specified which one)	[19]
NeuralCodeSum [51]	[14]
Compiled their own dataset [104]	[1]
VDISC [105]	[31]
VarMisuse [106]	[2]
Defects4J [107]	[44, 73, 13]
BFP [70]	[13]
LeetCode solutions [108]	[26]
Python dataset by [109]	[25]
Unavailable	[22]

Table 13. Datasets

REFERENCES

[1] Pavol Bielik and Martin Vechev. “Adversarial Robustness for Code”. en. In: *Proceedings of the 37th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, Nov. 2020, pp. 896–907. URL: https://proceedings.mlr.press/v119/bielik20a.html (visited on 09/12/2024).

[2] Fengjuan Gao, Yu Wang, and Ke Wang. "Discrete Adversarial Attack to Models of Code". In: *Proc. ACM Program. Lang.* 7.PLDI (June 2023), 113:172–113:195. DOI: 10.1145/3591227. URL: https://dl.acm.org/doi/10.1145/3591227 (visited on 09/10/2024).

[3] Junfeng Tian et al. "Generating Adversarial Examples of Source Code Classification Models via Q-Learning-Based Markov Decision Process". In: *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*. ISSN: 2693-9177. Dec. 2021, pp. 807–818. DOI: 10.1109/QRS54544.2021.00090. URL: https://ieeexplore.ieee.org/document/9724884 (visited on 09/18/2024).

[4] Zhao Tian, Junjie Chen, and Zhi Jin. "Code Difference Guided Adversarial Example Generation for Deep Code Models". In: *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ISSN: 2643-1572. Sept. 2023, pp. 850–862. DOI: 10.1109/ASE56229.2023.00149. URL: https://ieeexplore.ieee.org/document/10298520 (visited on 09/11/2024).

[5] Weiwei Zhang et al. "Challenging Machine Learning-Based Clone Detectors via Semantic-Preserving Code Transformations". In: *IEEE Transactions on Software Engineering* 49.5 (May 2023). Conference Name: IEEE Transactions on Software Engineering, pp. 3052–3070. ISSN: 1939-3520. DOI: 10.1109/TSE.2023.3240118. URL: https://ieeexplore.ieee.org/document/10028657 (visited on 09/12/2024).

[6] Qianjun Liu et al. "A Practical Black-Box Attack on Source Code Authorship Identification Classifiers". In: *IEEE Transactions on Information Forensics and Security* 16 (2021). Conference Name: IEEE Transactions on Information Forensics and Security, pp. 3620–3633. ISSN: 1556-6021. DOI: 10.1109/TIFS.2021.3080507. URL: https://ieeexplore.ieee.org/document/9454564 (visited on 10/03/2024).

[7] Penglong Chen et al. "Generating Adversarial Source Programs Using Important Tokens-based Structural Transformations". In: *2022 26th International Conference on Engineering of Complex Computer Systems (ICECCS)*. Mar. 2022, pp. 173–182. DOI: 10.1109/ICECCS54210.2022.00029. URL: https://ieeexplore.ieee.org/document/9763729 (visited on 09/18/2024).

[8] Erwin Quiring, Alwin Maier, and Konrad Rieck. *Misleading Authorship Attribution of Source Code using Adversarial Learning*. arXiv:1905.12386 [cs, stat]. May 2019. DOI: 10.48550/arXiv.1905.12386. URL: http://arxiv.org/abs/1905.12386 (visited on 09/13/2024).

[9] Paras Jain et al. "Contrastive Code Representation Learning". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. arXiv:2007.04973 [cs, stat]. 2021, pp. 5954–5971. DOI: 10.18653/v1/2021.emnlp-main.482. URL: http://arxiv.org/abs/2007.04973 (visited on 09/16/2024).

[10] Dexin Liu and Shikun Zhang. "ALANCA: Active Learning Guided Adversarial Attacks for Code Comprehension on Diverse Pre-trained and Large Language Models". In: *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. ISSN: 2640-7574. Mar. 2024, pp. 602–613. DOI: 10.1109/SANER60148.2024.00067. URL: https://ieeexplore.ieee.org/abstract/document/10589851 (visited on 09/18/2024).

[11] Leonhard Applis, Annibale Panichella, and Arie van Deursen. "Assessing Robustness of ML-Based Program Analysis Tools using Metamorphic Program Transformations". In: *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ISSN: 2643-1572. Nov. 2021, pp. 1377–1381. DOI: 10.1109/ASE51524.2021.9678706. URL: https://ieeexplore.ieee.org/abstract/document/9678706 (visited on 09/07/2024).

[12] Leonhard Applis, Annibale Panichella, and Ruben Marang. "Searching for Quality: Genetic Algorithms and Metamorphic Testing for Software Engineering ML". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '23. New York, NY, USA: Association for Computing Machinery, July 2023, pp. 1490–1498. ISBN: 9798400701191. DOI: 10.1145/3583131.3590379. URL: https://dl.acm.org/doi/10.1145/3583131.3590379 (visited on 09/10/2024).

[13] Hongliang Ge et al. "RobustNPR: Evaluating the robustness of neural program repair models". en. In: *Journal of Software: Evolution and Process* 36.4 (2024). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.2586, e2586. ISSN: 2047-7481. DOI: 10.1002/smr.2586. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2586 (visited on 09/18/2024).

[14] Moshi Wei et al. "CoCoFuzzing: Testing Neural Code Models With Coverage-Guided Fuzzing". In: *IEEE Transactions on Reliability* 72.3 (Sept. 2023). Conference Name: IEEE Transactions on Reliability, pp. 1276–1289. ISSN: 1558-1721. DOI: 10.1109/TR.2022.3208239. URL: https://ieeexplore.ieee.org/abstract/document/9916170 (visited on 09/18/2024).

[15] Xi Ding et al. "Adversarial Attack and Robustness Improvement on Code Summarization". In: *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*. EASE '24. New York, NY, USA: Association for Computing Machinery, June 2024, pp. 17–27. ISBN: 9798400717017. DOI: 10.1145/3661167.3661173. URL: https://dl.acm.org/doi/10.1145/3661167.3661173 (visited on 09/18/2024).

[16] Jordan Henkel et al. "Semantic Robustness of Models of Source Code". In: *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. ISSN: 1534-5351. Mar. 2022, pp. 526–537. DOI: 10.1109/SANER53432.2022.00070. URL: https://ieeexplore.ieee.org/document/9825895 (visited on 09/12/2024).

[17] Shashank Srikant et al. *Generating Adversarial Computer Programs using Optimized Obfuscations.* arXiv:2103.11882 [cs]. Mar. 2021. doi: 10.48550/arXiv.2103.11882. url: http://arxiv.org/abs/2103.11882 (visited on 09/11/2024).

[18] Jinghan Jia et al. "ClawSAT: Towards Both Robust and Accurate Code Models". In: *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER).* ISSN: 2640-7574. Mar. 2023, pp. 212–223. doi: 10.1109/SANER56733.2023.00029. url: https://ieeexplore.ieee.org/document/10123554 (visited on 09/11/2024).

[19] Yulong Yang et al. "Exploiting the Adversarial Example Vulnerability of Transfer Learning of Source Code". In: *IEEE Transactions on Information Forensics and Security* 19 (2024). Conference Name: IEEE Transactions on Information Forensics and Security, pp. 5880–5894. issn: 1556-6021. doi: 10.1109/TIFS.2024.3402153. url: https://ieeexplore.ieee.org/document/10531252 (visited on 09/18/2024).

[20] Noam Yefet, Uri Alon, and Eran Yahav. "Adversarial examples for models of code". In: *Proc. ACM Program. Lang.* 4.OOPSLA (Nov. 2020), 162:1–162:30. doi: 10.1145/3428230. url: https://dl.acm.org/doi/10.1145/3428230 (visited on 09/10/2024).

[21] Huangzhao Zhang et al. "Generating Adversarial Examples for Holding Robustness of Source Code Processing Models". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.01 (Apr. 2020). Number: 01, pp. 1169–1176. issn: 2374-3468. doi: 10.1609/aaai.v34i01.5469. url: https://ojs.aaai.org/index.php/AAAI/article/view/5469 (visited on 09/12/2024).

[22] Shasha Zhou et al. "Evolutionary Multi-objective Optimization for Contextual Adversarial Example Generation". In: *Proc. ACM Softw. Eng.* 1.FSE (July 2024), 101:2285–101:2308. doi: 10.1145/3660808. url: https://dl.acm.org/doi/10.1145/3660808 (visited on 09/10/2024).

[23] Zhou Yang et al. "Natural attack for pre-trained models of code". In: *Proceedings of the 44th International Conference on Software Engineering.* ICSE '22. New York, NY, USA: Association for Computing Machinery, July 2022, pp. 1482–1493. isbn: 978-1-4503-9221-1. doi: 10.1145/3510003.3510146. url: https://doi.org/10.1145/3510003.3510146 (visited on 09/10/2024).

[24] Huangzhao Zhang et al. "CodeBERT-Attack: Adversarial attack against source code deep learning models via pre-trained model". en. In: *Journal of Software: Evolution and Process* 36.3 (2024). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/ e2571. issn: 2047-7481. doi: 10.1002/smr.2571. url: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2571 (visited on 09/18/2024).

[25] Yu Zhou et al. "Adversarial Robustness of Deep Code Comment Generation". In: *ACM Trans. Softw. Eng. Methodol.* 31.4 (July 2022), 60:1–60:30. issn: 1049-331X. doi: 10.1145/3501256. url: https://doi.org/10.1145/3501256 (visited on 09/10/2024).

[26] Zongjie Li et al. *CCTEST: Testing and Repairing Code Completion Systems.* arXiv:2208.08289 [cs]. May 2023. doi: 10.48550/arXiv.2208.08289. url: http://arxiv.org/abs/2208.08289 (visited on 09/16/2024).

[27] Md Rafiqul Islam Rabin et al. "On the generalizability of Neural Program Models with respect to semantic-preserving program transformations". In: *Information and Software Technology* 135 (July 2021), p. 106552. issn: 0950-5849. doi: 10.1016/j.infsof.2021.106552. url: https://www.sciencedirect.com/science/article/pii/S0950584921000379 (visited on 09/12/2024).

[28] Shuzheng Gao et al. "Two Sides of the Same Coin: Exploiting the Impact of Identifiers in Neural Code Comprehension". In: *Proceedings of the 45th International Conference on Software Engineering.* ICSE '23. Melbourne, Victoria, Australia: IEEE Press, July 2023, pp. 1933–1945. isbn: 978-1-66545-701-9. doi: 10.1109/ICSE48619.2023.00164. url: https://doi.org/10.1109/ICSE48619.2023.00164 (visited on 09/10/2024).

[29] Shangqing Liu et al. "ContraBERT: Enhancing Code Pre-Trained Models via Contrastive Learning". In: *Proceedings of the 45th International Conference on Software Engineering.* ICSE '23. Melbourne, Victoria, Australia: IEEE Press, July 2023, pp. 2476–2487. isbn: 978-1-66545-701-9. doi: 10.1109/ICSE48619.2023.00207. url: https://doi.org/10.1109/ICSE48619.2023.00207 (visited on 09/10/2024).

[30] Huangzhao Zhang et al. "Towards Robustness of Deep Program Processing Models—Detection, Estimation, and Enhancement". In: *ACM Trans. Softw. Eng. Methodol.* 31.3 (Apr. 2022), 50:1–50:40. issn: 1049-331X. doi: 10.1145/3511887. url: https://dl.acm.org/doi/10.1145/3511887 (visited on 09/12/2024).

[31] Claudio Ferretti and Martina Saletta. "Deceiving neural source code classifiers: finding adversarial examples with grammatical evolution". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion.* GECCO '21. New York, NY, USA: Association for Computing Machinery, July 2021, pp. 1889–1897. isbn: 978-1-4503-8351-6. doi: 10.1145/3449726.3463222. url: https://dl.acm.org/doi/10.1145/3449726.3463222 (visited on 09/18/2024).

[32] CheolWon Na, YunSeok Choi, and Jee-Hyong Lee. "DIP: Dead code Insertion based Black-box Attack for Programming Language Model". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).* Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 7777–7791. doi: 10.18653/v1/2023.acl-long.430. url: https://aclanthology.org/2023.acl-long.430 (visited on 09/13/2024).

[33] Nick Craswell. “Mean Reciprocal Rank”. In: *Encyclopedia of Database Systems*. Ed. by LING LIU and M. TAMER ÖZSU. Boston, MA: Springer US, 2009, pp. 1703–1703. ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_488. URL: https://doi.org/10.1007/978-0-387-39940-9_488.

[34] Kishore Papineni et al. “Bleu: a method for automatic evaluation of machine translation”. In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002, pp. 311–318.

[35] Chin-Yew Lin. “ROUGE: A Package for Automatic Evaluation of Summaries”. In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81. URL: https://aclanthology.org/W04-1013 (visited on 10/30/2024).

[36] Satanjeev Banerjee and Alon Lavie. “METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments”. In: *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. Ed. by Jade Goldstein et al. Ann Arbor, Michigan: Association for Computational Linguistics, June 2005, pp. 65–72. URL: https://aclanthology.org/W05-0909 (visited on 10/30/2024).

[37] Gonzalo Navarro. “A guided tour to approximate string matching”. In: *ACM Comput. Surv.* 33.1 (Mar. 2001), pp. 31–88. ISSN: 0360-0300. DOI: 10.1145/375360.375365. URL: https://dl.acm.org/doi/10.1145/375360.375365 (visited on 10/30/2024).

[38] Shuo Ren et al. *CodeBLEU: a Method for Automatic Evaluation of Code Synthesis*. arXiv:2009.10297. Sept. 2020. DOI: 10.48550/arXiv.2009.10297. URL: http://arxiv.org/abs/2009.10297 (visited on 10/28/2024).

[39] Irwin Pollack and Robert Hsieh. “Sampling variability of the area under the ROC-curve and of d’e.” In: *Psychological Bulletin* 71.3 (1969). Place: US Publisher: American Psychological Association, pp. 161–173. ISSN: 1939-1455. DOI: 10.1037/h0026862.

[40] Paul Henderson and Vittorio Ferrari. “End-to-End Training of Object Class Detectors for Mean Average Precision”. en. In: *Computer Vision – ACCV 2016*. Ed. by Shang-Hong Lai et al. Cham: Springer International Publishing, 2017, pp. 198–213. ISBN: 978-3-319-54193-8. DOI: 10.1007/978-3-319-54193-8_13.

[41] Helen R. Sofaer, Jennifer A. Hoeting, and Catherine S. Jarnevich. “The area under the precision-recall curve as a performance metric for rare binary events”. en. In: *Methods in Ecology and Evolution* 10.4 (2019). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/2041-210X.13140, pp. 565–577. ISSN: 2041-210X. DOI: 10.1111/2041-210X.13140. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/2041-210X.13140 (visited on 10/28/2024).

[42] Mathieu Bouchard, Anne-Laure Jousselme, and Pierre-Emmanuel Doré. “A proof for the positive definiteness of the Jaccard index matrix”. In: *International Journal of Approximate Reasoning* 54.5 (July 2013), pp. 615–626. ISSN: 0888-613X. DOI: 10.1016/j.ijar.2013.01.006. URL: https://www.sciencedirect.com/science/article/pii/S0888613X1300008X (visited on 10/30/2024).

[43] Zhangyin Feng et al. *CodeBERT: A Pre-Trained Model for Programming and Natural Languages*. arXiv:2002.08155. Sept. 2020. DOI: 10.48550/arXiv.2002.08155. URL: http://arxiv.org/abs/2002.08155 (visited on 10/30/2024).

[44] Yiyang Li, Hongqiu Wu, and Hai Zhao. *Semantic-Preserving Adversarial Code Comprehension*. arXiv:2209.05130 [cs]. Sept. 2022. DOI: 10.48550/arXiv.2209.05130. URL: http://arxiv.org/abs/2209.05130 (visited on 09/16/2024).

[45] Thanh-Dat Nguyen et al. *Adversarial Attacks on Code Models with Discriminative Graph Patterns*. arXiv:2308.11161 [cs]. Aug. 2023. DOI: 10.48550/arXiv.2308.11161. URL: http://arxiv.org/abs/2308.11161 (visited on 09/16/2024).

[46] Guang Yang et al. *Assessing and Improving Syntactic Adversarial Robustness of Pre-trained Models for Code Translation*. arXiv:2310.18587 [cs]. Oct. 2023. DOI: 10.48550/arXiv.2310.18587. URL: http://arxiv.org/abs/2310.18587 (visited on 09/16/2024).

[47] Yong Yu et al. “A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures”. In: *Neural Computation* 31.7 (July 2019), pp. 1235–1270. ISSN: 0899-7667. DOI: 10.1162/neco_a_01199. URL: https://doi.org/10.1162/neco_a_01199 (visited on 10/30/2024).

[48] Zhen Li et al. *Towards Making Deep Learning-based Vulnerability Detectors Robust*. arXiv:2108.00669 [cs]. Aug. 2021. DOI: 10.48550/arXiv.2108.00669. URL: http://arxiv.org/abs/2108.00669 (visited on 09/16/2024).

[49] Daya Guo et al. *GraphCodeBERT: Pre-training Code Representations with Data Flow*. arXiv:2009.08366. Sept. 2021. DOI: 10.48550/arXiv.2009.08366. URL: http://arxiv.org/abs/2009.08366 (visited on 10/30/2024).

[50] Kexin Pei et al. *Exploiting Code Symmetries for Learning Program Semantics*. arXiv:2308.03312 [cs]. Sept. 2024. DOI: 10.48550/arXiv.2308.03312. URL: http://arxiv.org/abs/2308.03312 (visited on 09/17/2024).

[51] Wasi Uddin Ahmad et al. *A Transformer-based Approach for Source Code Summarization*. arXiv:2005.00653. May 2020. DOI: 10.48550/arXiv.2005.00653. URL: http://arxiv.org/abs/2005.00653 (visited on 10/30/2024).

[52] Chi Zhang et al. *Transfer Attacks and Defenses for Large Language Models on Coding Tasks*. arXiv:2311.13445 [cs]. Nov. 2023. DOI: 10.48550/arXiv.2311.13445. URL: http://arxiv.org/abs/2311.13445 (visited on 09/16/2024).

[53] Yaoxian Li et al. *A Closer Look into Transformer-Based Code Intelligence Through Code Transformation: Challenges and Opportunities*. arXiv:2207.04285 [cs]. July 2022. DOI: 10.48550/arXiv.2207.04285. URL: http://arxiv.org/abs/2207.04285 (visited on 09/16/2024).

[54] Uri Alon et al. “code2vec: learning distributed representations of code”. In: *Implementation, data and a trained model for the code2vec paper* 3.POPL (Jan. 2019), 40:1–40:29. DOI: 10.1145/3290353. URL: https://dl.acm.org/doi/10.1145/3290353 (visited on 10/30/2024).

[55] Md Rafiqul Islam Rabin and Mohammad Amin Alipour. *Evaluation of Generalizability of Neural Program Analyzers under Semantic-Preserving Transformations.* arXiv:2004.07313 [cs]. Mar. 2021. DOI: 10.48550/arXiv.2004.07313. URL: http://arxiv.org/abs/2004.07313 (visited on 09/13/2024).

[56] Md Rafiqul Islam Rabin, Ke Wang, and Mohammad Amin Alipour. *Testing Neural Program Analyzers.* arXiv:1908.10711 [cs, stat]. Sept. 2019. DOI: 10.48550/arXiv.1908.10711. URL: http://arxiv.org/abs/1908.10711 (visited on 09/12/2024).

[57] Uri Alon et al. “code2seq: Generating Sequences from Structured Representations of Code”. In: *International Conference on Learning Representations.* 2019. URL: https://openreview.net/forum?id=H1gKYo09tX.

[58] Jacob M. Springer, Bryn Marie Reinstadler, and Una-May O’Reilly. *STRATA: Simple, Gradient-Free Attacks for Models of Code.* arXiv:2009.13562 [cs, stat]. Aug. 2021. DOI: 10.48550/arXiv.2009.13562. URL: http://arxiv.org/abs/2009.13562 (visited on 09/18/2024).

[59] Yue Wang et al. *CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation.* arXiv:2109.00859. Sept. 2021. DOI: 10.48550/arXiv.2109.00859. URL: http://arxiv.org/abs/2109.00859 (visited on 10/30/2024).

[60] Yujia Li et al. *Gated Graph Sequence Neural Networks.* arXiv:1511.05493. Sept. 2017. DOI: 10.48550/arXiv.1511.05493. URL: http://arxiv.org/abs/1511.05493 (visited on 10/30/2024).

[61] Yaqin Zhou et al. “Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks”. In: *Advances in Neural Information Processing Systems.* Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/hash/49265d2447bc3bbfe9e76306ce40a31f-Abstract.html (visited on 10/30/2024).

[62] Vincent J. Hellendoorn et al. “Deep learning type inference”. In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* ESEC/FSE 2018. New York, NY, USA: Association for Computing Machinery, Oct. 2018, pp. 152–162. ISBN: 978-1-4503-5573-5. DOI: 10.1145/3236024.3236051. URL: https://dl.acm.org/doi/10.1145/3236024.3236051 (visited on 10/30/2024).

[63] Josh Achiam et al. “Gpt-4 technical report”. In: *arXiv preprint arXiv:2303.08774* (2023).

[64] Lili Mou et al. “Convolutional Neural Networks over Tree Structures for Programming Language Processing”. en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 30.1 (Feb. 2016). Number: 1. ISSN: 2374-3468. DOI: 10.1609/aaai.v30i1.10139. URL: https://ojs.aaai.org/index.php/AAAI/article/view/10139 (visited on 10/30/2024).

[65] Wasi Uddin Ahmad et al. *Unified Pre-training for Program Understanding and Generation.* arXiv:2103.06333. Apr. 2021. DOI: 10.48550/arXiv.2103.06333. URL: http://arxiv.org/abs/2103.06333 (visited on 10/30/2024).

[66] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks.* arXiv:1609.02907. Feb. 2017. DOI: 10.48550/arXiv.1609.02907. URL: http://arxiv.org/abs/1609.02907 (visited on 10/30/2024).

[67] Heng Li et al. *Black-box Adversarial Example Attack towards FCG Based Android Malware Detection under Incomplete Feature Information.* arXiv:2303.08509 [cs]. Mar. 2023. DOI: 10.48550/arXiv.2303.08509. URL: http://arxiv.org/abs/2303.08509 (visited on 09/18/2024).

[68] Mostafa Dehghani et al. *Universal Transformers.* arXiv:1807.03819. Mar. 2019. DOI: 10.48550/arXiv.1807.03819. URL: http://arxiv.org/abs/1807.03819 (visited on 10/30/2024).

[69] Rebecca Russell et al. “Automated Vulnerability Detection in Source Code Using Deep Representation Learning”. In: *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA).* Dec. 2018, pp. 757–762. DOI: 10.1109/ICMLA.2018.00120. URL: https://ieeexplore.ieee.org/abstract/document/8614145 (visited on 10/30/2024).

[70] Michele Tufano et al. “An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation”. In: *ACM Trans. Softw. Eng. Methodol.* 28.4 (Sept. 2019), 19:1–19:29. ISSN: 1049-331X. DOI: 10.1145/3340544. URL: https://dl.acm.org/doi/10.1145/3340544 (visited on 10/30/2024).

[71] Thibaud Lutellier et al. “CoCoNuT: combining context-aware neural translation models using ensemble for program repair”. In: *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis.* ISSTA 2020. New York, NY, USA: Association for Computing Machinery, July 2020, pp. 101–114. ISBN: 978-1-4503-8008-9. DOI: 10.1145/3395363.3397369. URL: https://dl.acm.org/doi/10.1145/3395363.3397369 (visited on 10/30/2024).

[72] Zimin Chen et al. “SequenceR: Sequence-to-Sequence Learning for End-to-End Program Repair”. In: *IEEE Transactions on Software Engineering* 47.9 (Sept. 2021). Conference Name: IEEE Transactions on Software Engineering, pp. 1943–1959. ISSN: 1939-3520. DOI: 10.1109/TSE.2019.2940179. URL: https://ieeexplore.ieee.org/abstract/document/8827954 (visited on 10/30/2024).

[73] Thanh Le-Cong et al. *Evaluating Program Repair with Semantic-Preserving Transformations: A Naturalness Assessment.* arXiv:2402.11892 [cs]. Feb. 2024. DOI: 10.48550/arXiv.2402.11892. URL: http://arxiv.org/abs/2402.11892 (visited on 09/17/2024).

[74] Qihao Zhu et al. *A Syntax-Guided Edit Decoder for Neural Program Repair.* arXiv:2106.08253. Mar. 2022. DOI: 10.48550/arXiv.2106.08253. URL: http://arxiv.org/abs/2106.08253 (visited on 10/30/2024).

[75] Srinivasan Iyer et al. "Summarizing Source Code using a Neural Attention Model: 54th Annual Meeting of the Association for Computational Linguistics 2016". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* 1 (Aug. 2016). Publisher: Association for Computational Linguistics, pp. 2073–2083. ISSN: 9781510827585. DOI: 10.18653/v1/P16-1195.

[76] Ashish Vaswani et al. *Attention Is All You Need.* arXiv:1706.03762. Aug. 2023. DOI: 10.48550/arXiv.1706.03762. URL: http://arxiv.org/abs/1706.03762 (visited on 10/30/2024).

[77] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach.* arXiv:1907.11692. July 2019. DOI: 10.48550/arXiv.1907.11692. URL: http://arxiv.org/abs/1907.11692 (visited on 10/30/2024).

[78] Mark Chen et al. *Evaluating Large Language Models Trained on Code.* arXiv:2107.03374. July 2021. DOI: 10.48550/arXiv.2107.03374. URL: http://arxiv.org/abs/2107.03374 (visited on 10/30/2024).

[79] Aram Bahrini et al. "ChatGPT: Applications, Opportunities, and Threats". In: *2023 Systems and Information Engineering Design Symposium (SIEDS).* Apr. 2023, pp. 274–279. DOI: 10.1109/SIEDS58326.2023.10137850. URL: https://ieeexplore.ieee.org/abstract/document/10137850 (visited on 10/30/2024).

[80] Aohan Zeng et al. *GLM-130B: An Open Bilingual Pre-trained Model.* arXiv:2210.02414. Oct. 2023. DOI: 10.48550/arXiv.2210.02414. URL: http://arxiv.org/abs/2210.02414 (visited on 10/30/2024).

[81] GitHub. *GitHub copilot · your AI pair programmer.* 2024. URL: https://github.com/features/copilot.

[82] CodeParrot. *Codeparrot (CodeParrot).* 2024. URL: https://huggingface.co/codeparrot.

[83] Sid Black et al. "GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow". In: Version Number: 1.0. Zenodo, Mar. 2021. DOI: 10.5281/ZENODO.5297715. URL: https://zenodo.org/record/5297715 (visited on 10/30/2024).

[84] EleutherAI. *ELEUTHERAI/GPT-J-6B · hugging face.* 2024. URL: https://huggingface.co/EleutherAI/gpt-j-6b.

[85] Erik Nijkamp et al. *CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis.* arXiv:2203.13474. Feb. 2023. DOI: 10.48550/arXiv.2203.13474. URL: http://arxiv.org/abs/2203.13474 (visited on 10/30/2024).

[86] Mohammed Abuhamad et al. "Large-Scale and Language-Oblivious Code Authorship Identification". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.* CCS '18. New York, NY, USA: Association for Computing Machinery, Oct. 2018, pp. 101–114. ISBN: 978-1-4503-5693-0. DOI: 10.1145/3243734.3243738. URL: https://dl.acm.org/doi/10.1145/3243734.3243738 (visited on 10/30/2024).

[87] Aylin Caliskan-Islam et al. "De-anonymizing programmers via code stylometry". In: *Proceedings of the 24th USENIX Conference on Security Symposium.* SEC'15. USA: USENIX Association, Aug. 2015, pp. 255–270. ISBN: 978-1-931971-23-2. (Visited on 10/30/2024).

[88] Marc Brockschmidt. *GNN-FiLM: Graph Neural Networks with Feature-wise Linear Modulation.* arXiv:1906.12192. June 2020. DOI: 10.48550/arXiv.1906.12192. URL: http://arxiv.org/abs/1906.12192 (visited on 10/30/2024).

[89] Jian Zhang et al. "A Novel Neural Source Code Representation Based on Abstract Syntax Tree". In: *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE).* ISSN: 1558-1225. May 2019, pp. 783–794. DOI: 10.1109/ICSE.2019.00086. URL: https://ieeexplore.ieee.org/document/8812062 (visited on 10/30/2024).

[90] Xuan Huo and Ming Li. "Enhancing the unified features to locate buggy files by exploiting the sequential nature of source code". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence.* IJCAI'17. Melbourne, Australia: AAAI Press, Aug. 2017, pp. 1909–1915. ISBN: 978-0-9992411-0-3. (Visited on 10/30/2024).

[91] Hui-Hui Wei and Ming Li. "Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence.* IJCAI'17. Melbourne, Australia: AAAI Press, Aug. 2017, pp. 3034–3040. ISBN: 978-0-9992411-0-3. (Visited on 10/30/2024).

[92] Hao Yu et al. "Neural detection of semantic code clones via tree-based convolution". In: *Proceedings of the 27th International Conference on Program Comprehension.* ICPC '19. Montreal, Quebec, Canada: IEEE Press, May 2019, pp. 70–80. DOI: 10.1109/ICPC.2019.00021. URL: https://dl.acm.org/doi/10.1109/ICPC.2019.00021 (visited on 10/30/2024).

[93] Bolin Wei et al. *Code Generation as a Dual Task of Code Summarization.* arXiv:1910.05923. Oct. 2019. DOI: 10.48550/arXiv.1910.05923. URL: http://arxiv.org/abs/1910.05923 (visited on 10/30/2024).

[94] Jian Zhang et al. "Retrieval-based neural source code summarization". In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering.* ICSE '20. New York, NY, USA: Association for Computing Machinery, Oct. 2020, pp. 1385–1397. ISBN: 978-1-4503-7121-6. DOI: 10.1145/3377811.3380383. URL: https://dl.acm.org/doi/10.1145/3377811.3380383 (visited on 10/30/2024).

[95] Hamel Husain et al. *CodeSearchNet Challenge: Evaluating the State of Semantic Code Search.* arXiv:1909.09436. June 2020. DOI: 10.48550/arXiv.1909.09436. URL: http://arxiv.org/abs/1909.09436 (visited on 10/30/2024).

[96] Jeffrey Svajlenko and Chanchal K. Roy. “Evaluating clone detection tools with BigCloneBench”. In: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Sept. 2015, pp. 131–140. doi: 10.1109/ICSM.2015.7332459. url: https://ieeexplore.ieee.org/abstract/document/7332459 (visited on 10/30/2024).

[97] Veselin Raychev, Pavol Bielik, and Martin Vechev. “Probabilistic model for code with decision trees”. In: *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. OOPSLA 2016. New York, NY, USA: Association for Computing Machinery, Oct. 2016, pp. 731–747. isbn: 978-1-4503-4444-9. doi: 10.1145/2983990.2984041. url: https://dl.acm.org/doi/10.1145/2983990.2984041 (visited on 10/30/2024).

[98] Bander Alsulami et al. “Source code authorship attribution using long short-term memory based networks: 22nd European Symposium on Research in Computer Security, ESORICS 2017”. In: *Computer Security – ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Proceedings*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (2017). Ed. by Einar Snekkenes, Simon N. Foley, and Dieter Gollmann. Publisher: Springer Verlag, pp. 65–82. issn: 9783319664019. doi: 10.1007/978-3-319-66402-6_6. url: http://www.scopus.com/inward/record.url?scp=85029521566&partnerID=8YFLogxK (visited on 10/30/2024).

[99] CodeChef. *Codechef: Practical coding for everyone*. 2024. url: https://codechef.com/.

[100] Xing Hu et al. “Summarizing source code with transferred API knowledge”. In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. IJCAI’18. Stockholm, Sweden: AAAI Press, July 2018, pp. 2269–2275. isbn: 978-0-9992411-2-7. (Visited on 10/30/2024).

[101] Ruchir Puri et al. *CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks*. arXiv:2105.12655. Aug. 2021. doi: 10.48550/arXiv.2105.12655. url: http://arxiv.org/abs/2105.12655 (visited on 10/30/2024).

[102] Shuai Lu et al. *CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation*. arXiv:2102.04664. Mar. 2021. doi: 10.48550/arXiv.2102.04664. url: http://arxiv.org/abs/2102.04664 (visited on 10/30/2024).

[103] Google. *Google Code Jam*. 2017. url: https://code.google.com/.

[104] Pavol Bielik. *ETH-Sri/robust-code: Adversarial robustness for code*. 2020. url: https://github.com/eth-sri/robust-code.

[105] Louis Kim and Rebecca Russell. *Draper VDISC dataset - vulnerability detection in source code*. Apr. 2024. url: https://osf.io/d45bw/.

[106] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. *Learning to Represent Programs with Graphs*. arXiv:1711.00740. May 2018. doi: 10.48550/arXiv.1711.00740. url: http://arxiv.org/abs/1711.00740 (visited on 10/30/2024).

[107] Rjust. *RJUST/DEFECTS4J: A database of real faults and an experimental infrastructure to enable controlled experiments in Software Engineering Research*. 2024. url: https://github.com/rjust/defects4j.

[108] JiayangWu. *Jiayangwu/leetcode-python: LeetCode Solutions in python2. LeetCode in python*. 2023. url: https://github.com/JiayangWu/LeetCode-Python.

[109] Yao Wan et al. “Improving automatic source code summarization via deep reinforcement learning”. In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ASE ’18. New York, NY, USA: Association for Computing Machinery, Sept. 2018, pp. 397–407. isbn: 978-1-4503-5937-5. doi: 10.1145/3238147.3238206. url: https://dl.acm.org/doi/10.1145/3238147.3238206 (visited on 10/30/2024).