Table 1. **Summarization of identifier splitting approaches**

Category	**Approach**	**Summary**	**Dataset**	**Performance**	**Tool Availability**	**Reference**
Heuristic-based	*Greedy*	Search longest prefix/suffix in dictionary iteratively; Do not discuss mix-case and digits; Prone to over-split	186 programs C/C++/Java/Fortran	P: 81%	N/A	[59]
	Samurai	Split mix-case and digit by heuristics; Split same-case based on local/global word frequencies	9,000 Java programs	Acc: 97%	Yes	[55]
	Spiral	Split mix-case and digit by heuristics; Split same-case based on global word frequencies	46,000 python projects	N/A	Yes	[81]
	Frequency-based	Compute all possible splits, and select splits based on word popularity and co-occurrence computed by search engine	60 Java projects	Acc: 30%	N/A	[166]
	TIDIER	Search word using *DTW* algorithm; Support transformed words splitting; Do not discuss digits	340 C programs	P: 54%	N/A	[119] [118] [71]
	GenTest	Generate all possible splits and score each split based on a set of metrics	186 programs C/C++/Java/Fortran	P: 82%	N/A	[103]
	INTT	Split mix-case and digit by heuristic rules, and split same-case by greedier algorithm	60 Java projects	Acc: 97%	Yes	[33] [29] [31]
Learning-based	FANN-based	Train FANN neural network model	186 programs C/C++/Java/Fortran	P: 95%	N/A	[59]
	n-gram based	Train n-gram language model; Specialize in abbreviations and acronyms splitting	5 Java projects	Acc: 94%	N/A	[145]
Graph-based	*TRIS*	Search words using heuristics and Dijkstra algorithm; Specialize in abbreviations & acronyms splitting	186 programs C/C++/Java/Fortran	P: 98% R: 94% F: 96%	N/A	[72] [70]
	LINSEN	Search words using BYP and Dijkstra algorithm; Specialize in abbreviations & acronyms splitting	24 projects C/C++/Java	P: 95%	N/A	[51]
	LIDS	Rank splits using word frequency and Dijkstra algorithm; Specialize in abbreviations & acronyms splitting	2 C projects	F: 95%	Yes	[39]

Note: For space limitation, only the best performance are presented here. The "P", "R", "F","Acc" stand for Precision, Recall, F-measure, and Accuracy, respectively.

Table 2. **Summarization of abbreviation expansion approaches**

Category	**Approach**	**Summary**	**Dataset**	**Performance**	**Tool Availability**	**Reference**
Heuristic-based	Dictionary-based	Search word/phrase in comment, source code, stop-list, and dictionary by exploiting heuristics	158 projects C/C++/Java	P: 58%	N/A	[105]
	AMAP	Search word/phrase in method, program, and software by exploiting heuristics; Select expansion based on word/phrase frequency	5 Java projects	Acc: 59%	Yes	[77]
	Similarity-based	Expand abbreviations in parameters based on the lexical similarity between arguments and parameters	9 Java projects	P: 95% R: 65%	N/A	[87]
	TIDIER	Search expansion in contextual dictionaries by exploiting DTW string matching algorithm	340 C programs	P: 48%	N/A	[119] [118] citeSMR2013
Learning-based	*Normalize*	Search word/phrase in source code, domain documentation, and Google data set by exploiting maximum coherence model	2 C projects	Acc: 66%	N/A	[102]
Graph-based	*LIDS*	Represent abbreviations in a graph, and search expansion in domain dictionary and software artifacts based on word frequency and dictionary weights	2 C projects	F: 91%	Yes	[39]
	TRIS	Represent abbreviations in a tree, and search expansion in source code based on word frequency and transformation costs	186 projects C/C++/Java	P: 98% R: 94% F: 96%	N/A	[72]
	LINSEN	Represent abbreviations in a graph, and search expansion in comment, source code, and dictionary by exploiting BYP string matching algorithm	2 projects C/C++/Java	F: 62%	N/A	[51]

Note: For space limitation, only the best performance are presented here. The “P", “R", “F",“Acc" stand for Precision, Recall, F-measure, and Accuracy, respectively.

Table 3. **Summarization of part of speech approaches**

Approach	**Summary**	**Dataset**	**Accuracy**	**Tool Availability**	**Reference**
Heuristic-based	Tag POS against naming conventions using heuristics	N/A	N/A	N/A	[110]
TreeTagger-based	Tag POS by providing guidance for TreeTagger	24 Java projects	Acc: 96%	N/A	[58]
Minipar-based	Tag POS by providing guidance for Minipar	1 C++ project	P: 86%	N/A	[6]
Stanford POS tagger-based	Tag POS by providing guidance for Stanford POS tagger	171 C++/Java projects	Acc: 88%	N/A	[22]
POSSE	Tag POS based on naming conventions and word frequency in software using heuristics	310 C++/Java programs	Acc: 91%	N/A	[74]
srcNLP	Tag POS based on stereotype data using finite state machine	5 C projects	Acc: 91%	N/A	[13] [12] [134]
S-POS	Train a maximum entropy Markov model	525 Stack Overflow posts	Acc: 93%	N/A	[181]

Note: For space limitation, only the best performance are presented here. The "P", "R", "F","Acc" stand for Precision, Recall, F-measure, and Accuracy, respectively.

Table 4. **Summarization of convention-based renaming opportunity identification approaches**

Category	**Approach**	**Summary**	**Dataset**	**Performance**	**Tool Availability**	**Reference**
Heuristic-based	Dictionary-based	Standardize the lexicon & syntax of identifiers	1 C project	N/A	N /A	[37] [36]
	LBSDetectors	Identify lexicon bad smells using Minipar and PaWs	2 C++ projects	P: 100%	N/A	[5]
	POS-based	Constrain syntax rules on field names	171 programs C++ and Java	N/A	N/A	[22]
	Nominal	Constrain syntax rules on class name	N/A	N/A	N/A	[33] [30]
	POS-based	Constrain syntax rules on different identifier types	N/A	N/A	N/A	[96]
	Smart Formatter	Constrain POS of prefix and first word in different identifier types	N/A	N/A	Yes	[45]
Learning-based	Bayes-based	Train learning algorithms to mine naming conventions	4 Java projects	N/A	N/A	[149]
	NATURALIZE	Train n-gram based on sequences of tokens in source code; Recommend variable names	10 Java projects	Acc: 94%	Yes	[10]
	LEAR	Train n-gram based on sequences of tokens in source code (excluding noisy data); Recommend parameters and local variables for methods	5 Java projects	Acc: 81%	N/A	[111]
	n-gram based	Train n-gram based on sequences of tokens from method names	1000 Java projects	P: 80% R: 75%	N/A	[167]

Note: For space limitation, only the best performance are presented here. The “P", “R", “F",“Acc" stand for Precision, Recall, F-measure, and Accuracy, respectively.

Table 5. **Summarization of inconsistency-based renaming opportunity identification approaches**

Category	Approach	Summary	Dataset	Performance	Tool Availability	Reference
Inconsistency between identifiers	*IDD*	Identify semantic inconsistency based on bijective mapping between concepts and identifiers	N/A	N/A	N/A	[49] [146] [50]
	CodeAmigo	Identify semantic and syntactic inconsistency based on WordNet and lexical similarity	7 Java projects	P: 85% R:84%	Yes	[96]
	Vaci	Identify semantic inconsistency based on transitive closures and method clustering	N/A	N/A	N/A	[89]
	Syntax-based	Identify syntactic inconsistency based on syntax rules	2 programs C/C++	P: 76%	N/A	[104]
	CA-RENAMING	Identify semantic inconsistency based on variable assignment and type information	7 Java projects	P: 66%	N/A	[170]
	Graph-based	Suggest consistent names by extracting concept ontology from source code	6 projects (C++/Java)	P: 80%	N/A	[7]
	JSNice	Predict syntactic names using CRFs graphical model & MAP techniques	10,567 Javascript projects	Acc: 63%	N/A	[148]
	CP-Miner	Identify inconsistency among clones by mapping identifiers based on frequency	4 C projects	N/A	N/A	[109]
	Clone-based	Identify inconsistency among clones by traversing identifiers in clone AST Filter out false positive using heuristics	2 C/Java projects	N/A	N/A	[86]
	ABCD	Identify inconsistency among clones by exploiting neural network technique	12 Java projects	P: 84% R: 75% F: 81%	N/A	[176]
	MPANalyzer	Identify inconsistency across revisions based on modification patterns	2 C projects	P: 89%	Yes	[100] [75]
	SPA	Identify inconsistency across revisions by analyzing commits and dataflow	4 Java projects	P: 73% R: 90%	N/A	[147]
	COCONUT	Inconsistency between artifacts and source code	4 Java projects	P: 50%	N/A	[47]
	RenameExpander	Identify inconsistency between the renamed entity and its related entities	4 Java projects	P: 82%	N/A	[114]

Table 5. **Summarization of inconsistency-based renaming opportunity identification approaches (Continued)**

Category	**Approach**	**Summary**	**Dataset**	**Performance**	**Tool Availability**	**Reference**
	Similarity-based	Identify inconsistency between actual argument and formal parameter	14 Java programs	P: 83%	N/A	[115]
Inconsistency between identifiers and entities	*Lancelot*	Identify inconsistency between method name and method implementation	100 Java projects	P: 70%	Yes	[78] [79] [80] [91]
	Micro-pattern based	Identify inconsistency between the suffix of class name and implementation of the class	9 Java projects	Acc: 75%	N/A	[161]
	Association rule-based	Identify inconsistency between method name and method implementation	6 Java projects	P: 60%	N/A	[93] [92]
	LAPD	Identify inconsistency among method names, documentations, and behaviors	4 Java projects	P: 72%	N/A	[19] [18]
	Learning-based	Train a SVM classifier to suggest method names	3 Java projects	P: 70%	N/A	[182]
	Learning-based	Train a log-bilinear neural network to suggest method and class names	20 Java projects	F: 50%	N/A	[11]
	Learning-based	Train a CNN model to suggest method names	430 Java projects	F: 68%	N/A	[116]

Note: For space limitation, only the best performance are presented here. The "P", "R", "F","Acc" stand for Precision, Recall, F-measure, and Accuracy, respectively.

Table 6. **Summarization of renamings execution approaches**

Category	**Approach**	**Summary**	**Applied Language**	**Tool Availability**	**Reference**
Precondition-based	AST-based	Based on semantic verification of AST transformation	C	N/A	[43]
	Go Doctor	Check transformation validity by parsing ASTs of dependent files	Go	N/A	[27]
	Differential-based	Guerantee name binding by differential precondition checking	Independent	N/A	[141] [142]
Reference-based	*JunGL*	Based on dataflow analysis & graph searching	Independent	N/A	[175]
	BeneFactor	Based on tracking workflow patterns	N/A	N/A	[66]
	CReN	Track references within clones based on AST analysis	Java	N/A	[83]
	LexId	Track references within clones based on AST analysis Support renaming part of identifiers	Java	N/A	[84]
	Annotation-based	Bind reference to declaration by assigning globally unique name	Java	N/A	[73]
	Annotation-based	Bind reference to declaration by creating symbolic name and inverted name lookup	Java	N/A	[153]
	Annotation-based	Bind references to declaration by creating qualified name	Independent	N/A	[46]
	JSRefactor	Rename dynamic type languages based on static point-to and type inference analysis	JavaScript	Yes	[60] [61]
	GRE-Refactoring	Rename Groovy elements in Java program by defining a search engine and generating corresponding edits	Java & Groovy	N/A	[95]
	Pattern-based	Rename multi-language applications based on references recorded in XML and *Observer design pattern*	Java SSH	N/A	[41]
	Annotation-based	Rename multi-language applications by adding annotations to restore object relational mapping	Hibernate	N/A	[156] [155]
	Logic-based	Rename multi-language applications based on binding logic of each framework	Multi-language	N/A	[124] [123]
	BabelRef	Rename multi-language web applications based on symbolic execution and *D-Model* tree	HTML & PHP	N/A	[137]

Table 7. **Summarization of renamings detection approaches**

Approach	**Summary**	**Dataset**	**Performance**	**Tool Availability**	**Reference**
Similarity-based	Detect renamed functions by computing similarities across revisions	2 C projects	Acc: 91%	N/A	[97]
AST-based	Detect renamed global variables, types, and functions based on partial AST matching	5 C projects	N/A	N/A	[132]
CHANGEDISTILLER	Detect renamed methods and fields by exploiting differencing algorithm on source code	3 Java projects	P: 66%	N/A	[63]
AST-based	Detect renamed classes, methods, and fields by exploiting differecing algorithm on revision histories	7 Java projects	P: 99%	N/A	[94]
renaming detector	Detect renamed variables by exploiting differencing algorithm on tokens	77 Java files	P: 100% R: 85%	N/A	[120]
REPENT	Detect and classify renamings based on differencing and data-flow analysis	5 Java projects	P: 88% R: 92%	N/A	[57] [20] [56]
Commit-based	Detect and classify renamings based on commit message	3,795 Java systems	N/A	N/A	[144]
Differential-based	Generic refactoring detection based on differencing analysis	N/A	N/A	N/A	[178] [179]
RefactorMiner	Generic refactoring detection based on differential analysis	N/A	N/A	N/A	[160]
Original analysis-based	Generic refactoring detection based on original analysis	N/A	N/A	N/A	[68]
Heuristic-based	Generic refactoring detection based on heuristics	N/A	N/A	N/A	[174]
Signature-based	Generic refactoring detection based on signature analysis	N/A	N/A	N/A	[177]
RefactoringCrawler	Generic refactoring detection based on semantic analysis	3 Java projects	Acc: 85%	N/A	[53]
RefacLib	Generic refactoring detection based on semantic analysis	5 Java projects	F: 96%	N/A	[169]
RFM-based	Generic refactoring detection based on RFM technique	N/A	N/A	N/A	[14]

Note: For space limitation, only the best performance are presented here. The "P", "R", "F","Acc" stand for Precision, Recall, F-measure, and Accuracy, respectively.