Ensemble Approaches for Automatic Program Repair

KEVIN ZHANG

Background

Automatic Program Repair autonomously offers fixes to bugs/errors

- Through generation of patches
- Test-suite based

Difficult problem, even with assumptions-

- Bugs approximately correct,
- Fault localization, contextual information

```
    public static int bitcount(int n) {
        int count = 0;
        while (n != 0) {
-           n = (n ^ (n - 1));
+           n = (n & (n - 1));
            count++;
        }
        return count;
    }
```

QuixBugs: BITCOUNT.java

Background

Data

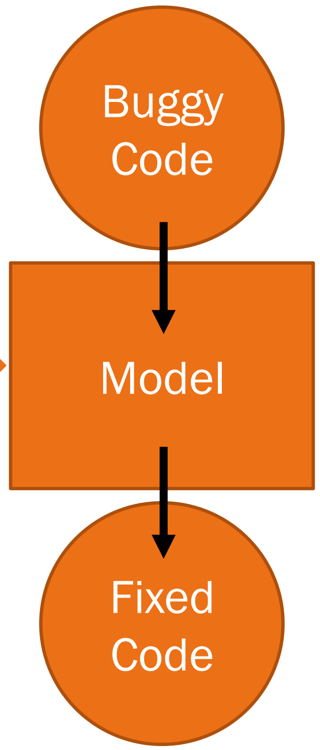
- Collection/Scraping
- Fault Localization
- Test Cases

Training

- Model Design
- Training
- Ensemble

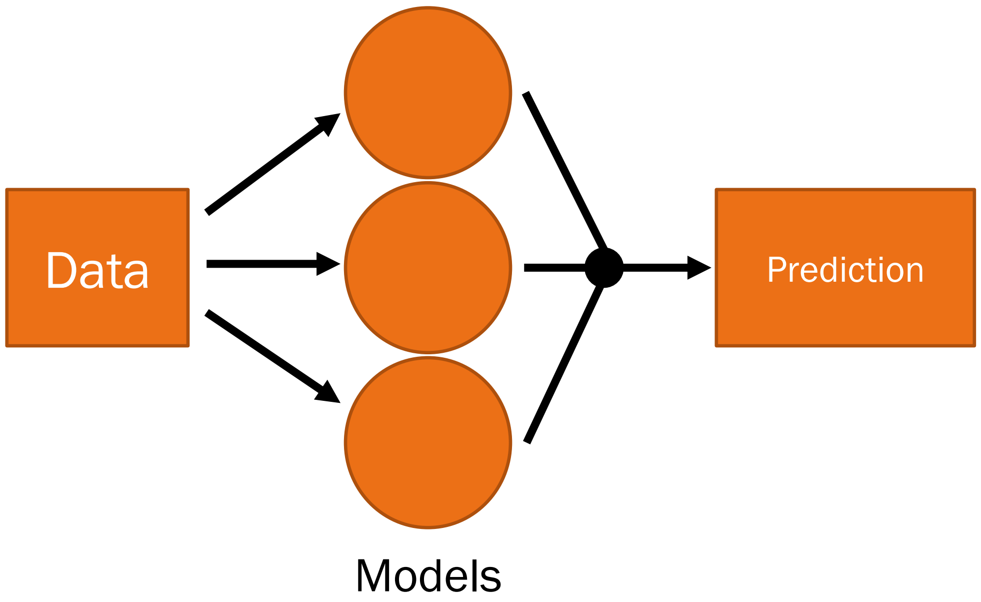
Inference

- <u>Patch Generation</u>
- Validation
- Time Concerns



Background

- Development of learning-based methods
  - Use neural networks and deep learning
  - Incremental improvements led to state-of-the art methodology
- Training: Learn/mimic fixing code
  - Ensemble improves by “wisdom of the crowd”

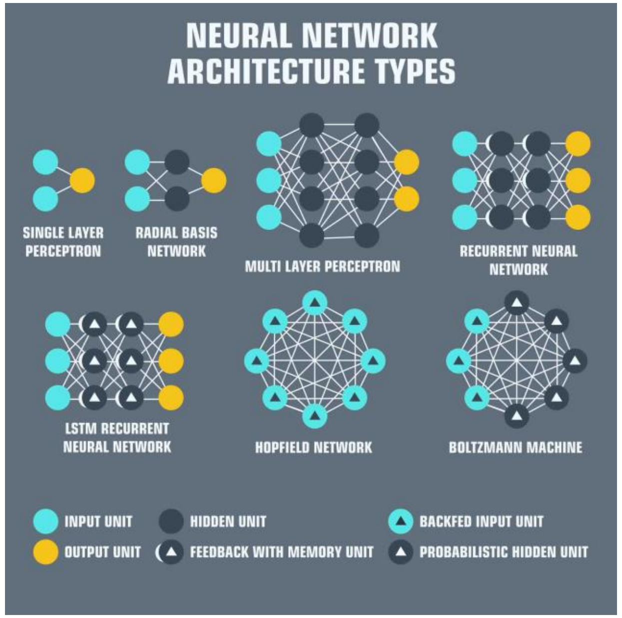


Motivation:

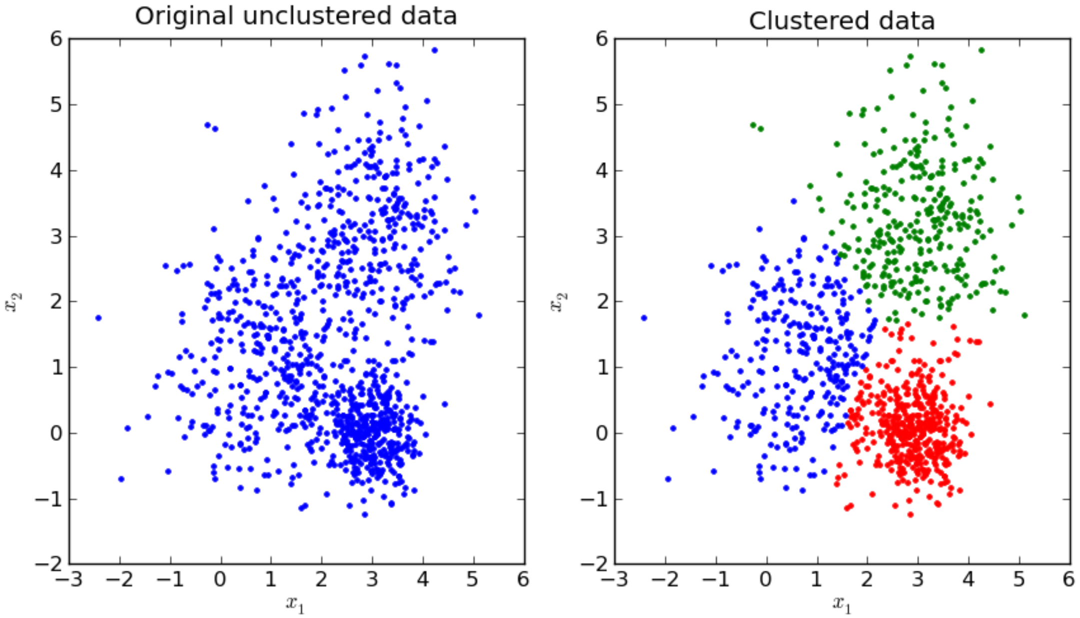
How can we characterize different clustering methods for training ensemble models?

Ensemble in Neural Networks:

1. Net Type   2. Data Clustering   3/4. Hyperparameters and Initial Weights



https://www.mygreatlearning.com/blog/cnn-model-architectures-and-applications/



https://towardsdatascience.com/k-means-data-clustering-bce3335d2203

Methods: Clustering

Fair comparison: Each method partitioned set into 4 clusters

- Addition of final "insertion" model to each method

Random Partitioning:

- Training and Evaluation evenly split into four "clusters"

Type Categorization:

- Training and Evaluation split via human-decided categories
- Statement, Return, If/While, Mixed

Machine Clustering:

- Use algorithms on machine understanding

Methods: Clustering

```
n = (n ^ (n - 1));  →  n = (n & (n - 1));
```

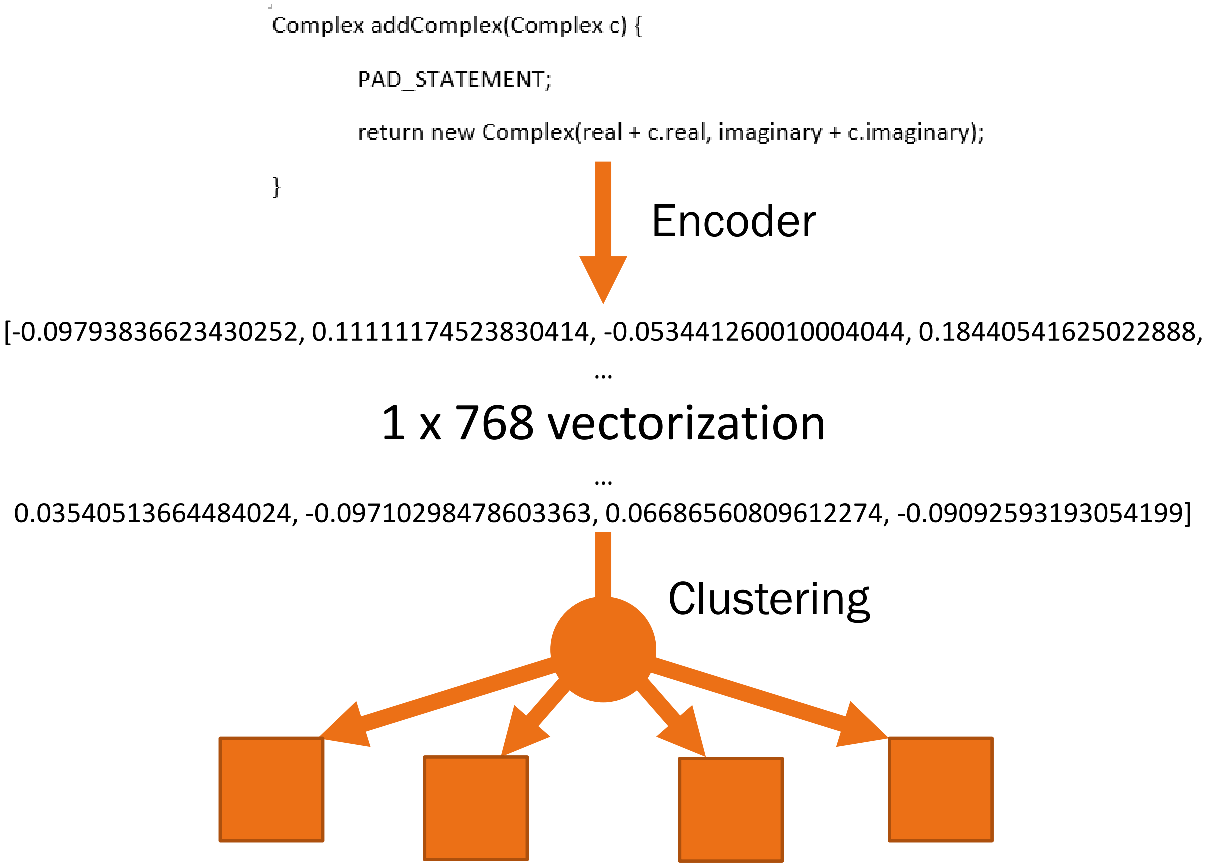
a) Statement Buggy Line Type

```
while (true) {  →  while (!queue.isEmpty()) {
```

b) If/While Buggy Line Type

```
return binsearch(arr, x, mid, end);
        → return binsearch(arr, x, mid+1, end);
```

c) Return Buggy Line Type



Methods: Training Data

Training data collected from GitHub

- Scrape source code from public projects
- Commits selected via messages: “fix”, “patch”
- JavaParser transforms into AST

450,000 data points in Training

50,000 data points in Evaluation

```
Complex addComplex(Complex c) {
        PAD_STATEMENT;
        return new Complex(real + c.real, imaginary + c.imaginary);
}
```

Parser


```
{'id': 1, 'mappings': {'imaginary': 'VAR_1', 'real': 'VAR_2', 'PAD_STATEMENT': 'VAR_3', 'Complex': 'TYPE_1', 'complexAdd': 'METHOD_1', 'NaN': 'VAR_<UNK>', '
isNaN': 'VAR_<UNK>'}, 'nodes': ['MethodDeclaration', 'ReferenceType', 'Complex', 'complexAdd', 'FormalParameter', 'ReferenceType', 'Complex', 'c', 'Statemen
tExpression', 'MemberReference', 'PAD_STATEMENT', 'ReturnStatement', 'ClassCreator', 'ReferenceType', 'Complex', 'BinaryOperation', 'MemberReference', 'real
', '+', 'MemberReference', 'c', 'real', 'BinaryOperation', 'MemberReference', 'imaginary', '+', 'MemberReference', 'c', 'imaginary'], 'edges': [[0, 1, 'retu
rn_type'], [1, 2, 'name'], [0, 3, 'name'], [0, 4, 'parameters'], [4, 5, 'type'], [5, 6, 'name'], [4, 7, 'name'], [0, 8, 'body'], [8, 9, 'expression'], [9, 1
0, 'member'], [0, 11, 'body'], [11, 12, 'expression'], [12, 13, 'type'], [13, 14, 'name'], [12, 15, 'arguments'], [15, 16, 'operandl'], [16, 17, 'member'],
[15, 18, 'operator'], [15, 19, 'operandr'], [19, 20, 'qualifier'], [19, 21, 'member'], [12, 22, 'arguments'], [22, 23, 'operandl'], [23, 24, 'member'], [22,
 25, 'operator'], [22, 26, 'operandr'], [26, 27, 'qualifier'], [26, 28, 'member']], 'rem_roots': [8], 'add_roots': [{'nodes': ['IfStatement', 'BinaryOperati
on', 'MemberReference', 'isNaN', '||', 'MemberReference', 'c', 'isNaN', 'ReturnStatement', 'MemberReference', 'NaN'], 'edges': [[0, 1, 'condition'], [1, 2,
'operandl'], [2, 3, 'member'], [1, 4, 'operator'], [1, 5, 'operandr'], [5, 6, 'qualifier'], [5, 7, 'member'], [0, 8, 'then_statement'], [8, 9, 'expression']
, [9, 10, 'member']]}]}
```

Methods: Training

Training order randomized: Cycle through all data = epoch

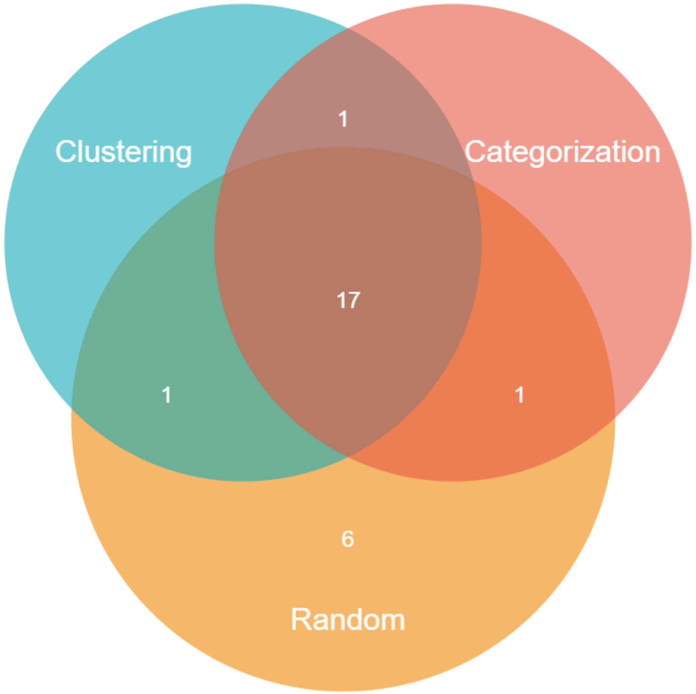
Training length approximately based on number of data points in cluster

Random Partitioning	Type Categorization				Machine Clustering*			
All Clusters	Return	If/While	Statement	Mixed	0	1	2	3
113,000	48,000	47,000	252,000	106,000	136,000	92,000	67,00	46,000

Random Partitioning	Type Categorization			Machine Clustering*		
All Clusters	Return & If/While	Statement	Mixed	0/1	2	3
10 Epochs	20 Epochs	15 Epochs	10 Epochs	10 Epochs	15 Epochs	20 Epochs

Results: Overview

Random Partitioning	Type Categorization	Machine Clustering	State-of-the Art
25	19	19*	26



Model trained on randomly partitioned data performed best

Generate very similar patches

- Model type and data the same

Differences in Generation-

- Differing Parameters
- Patch Ranking

Results: Generation

QuixBug: LEVENSHTEIN

Original

- return 1 + levenshtein(source.substring(1), target.substring(1))

Random

- return 0 + levenshtein(source.substring(1), target.substring(1))

Cluster

- return levenshtein(source.substring(1), target.substring(1))

Category

- return 1 * levenshtein(source.substring(1), target.substring(1))

Results: Generation

QuixBug: FIND_IN_SORTED

Original

- Return binsearch(arr, x, mid, end)

Random

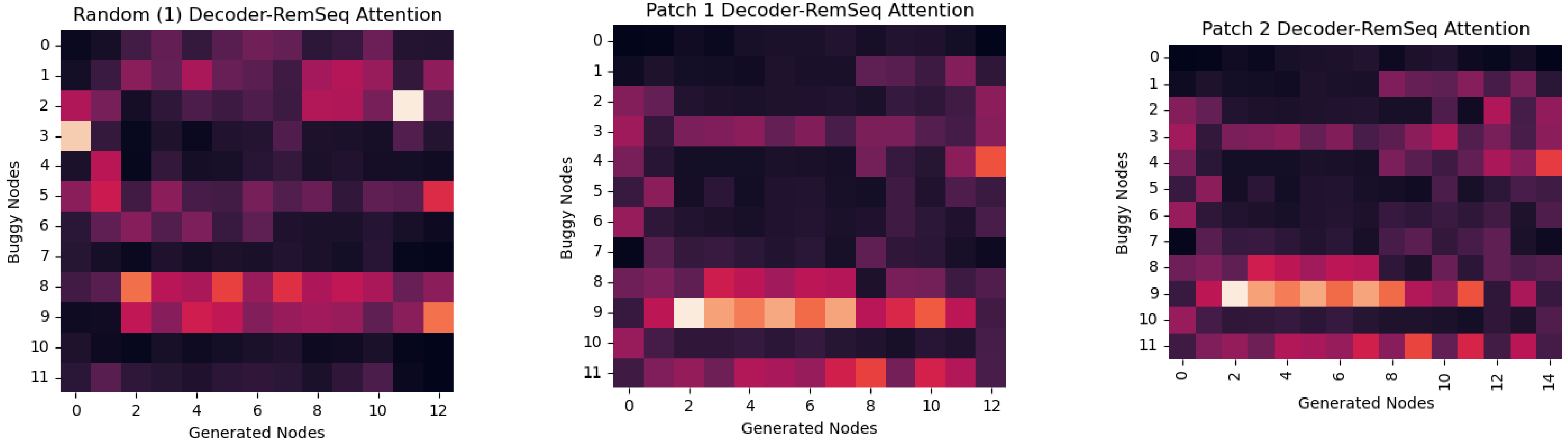
- return binsearch(arr, x, mid + 1, end)

Category/Cluster

- (insert) mid++

Results: Attention

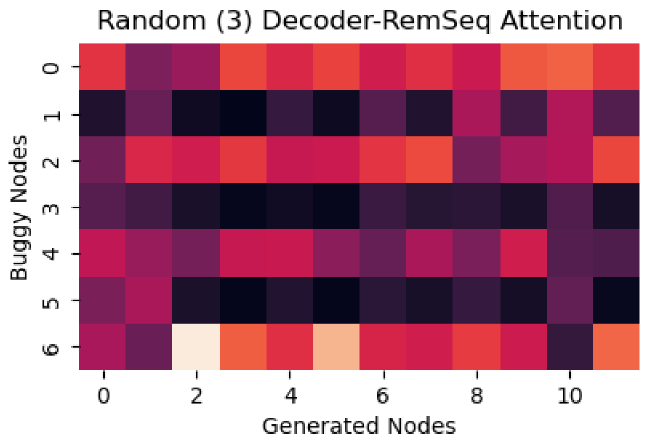
Differences in Attention:

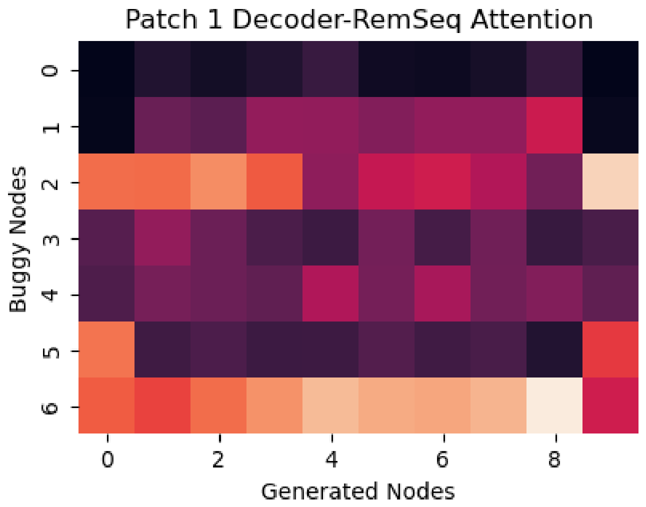


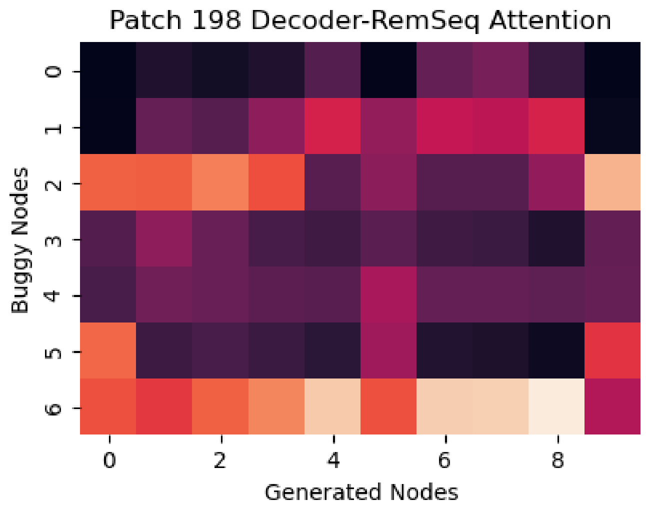
QuixBugs bug NEXT_PALINDROME

Results: Attention

Differences in Attention:







QuixBugs bug KTH

Results: Categorization Efficiency

"Ablation Study": Compare index of perfect match

- Ranking with all models/only specific category

QuixBugs ID:	1	2	3	6	9	11	12	14
Category	StateEx	If	Mixed	If	Return	StateEx	Return	If
Class Rank	0	669	10	27	N/A	5	N/A	6
All Rank	0	1187	54	10	484	25	70	13

QuixBugs ID:	17	20	23	24	27	29	33
Category	Return	StateEx	StateEx	If	If	StateEx	Mixed
Class Rank	N/A	0	3	1	0	192	N/A
All Rank	2834	0	15	3	0	881	109

Conclusions

Ensemble trained on randomly partitioned data performed best:

- Machine Clustering ~ Human Categorization

General vs Specific domain knowledge

Class imbalances

Model convergence and metrics

Extensions:

- Apply to wider variety of model types
- Additional ensemble methods

Acknowledgements

Nan Jiang was instrumental in providing guidance and knowledge through the research process, and for this I am extremely grateful. I would additionally like to thank my faculty advisor Professor Lin Tan for her supervision through the process, the Purdue University SURF Program for support, and the University of Waterloo for allowing the use of its servers to train the models examined in the experiment