# CCTEST

## Testing and Repairing Code Completion Systems

THE UNIVERSITY OF TENNESSEE KNOXVILLE

# Code Generation

- LLM-based code completion systems are commonly used as an advanced form of autocomplete.
  - A user begins typing code, and the LLM “infers” the rest of the code.
- These LLMs are trained on thousands of codebases
  - GitHub Copilot, for example, is trained from public repositories on GitHub.

Next (Alt+]) Previous (Alt+[) Accept (Tab) Open GitHub Copilot (Ctrl+Enter)

```
class TrieNode:
    def startsWith(self, prefix):
        if not prefix:
            return True
        if prefix[0] not in self.children:
            return False
        return self.children[prefix[0]].startsWith(prefix[1:])
```

# Problem

- Code that is functionally identical to a human programmer but with slight mutations to its content when used as input to an LLM can produce dramatically different results or even cause generation to fail.
  - As of the time of this paper, there is no automated way to test and improve these code completion systems.

```python
def find_top_k(data_list, K):
    length = len(data_list)
    begin = 0
    end = length - 1
    index = divide(data_list,begin,end)

    if index == K:
        return data_list[index]
    elif index < K:
        return find_top_k(data_list,K)
    else:
        return find_top_k(data_list,K)
```

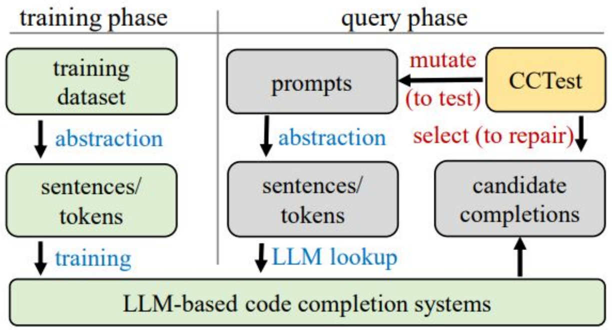
```python
def find_top_k(data_list, TOP_K):
    length = len(data_list)
    begin = 0
    end = length - 1
    index= divide(data_list,begin,end)

    if index == -1:
        return -1
    else:
        return index
```

# Solution: CCTest

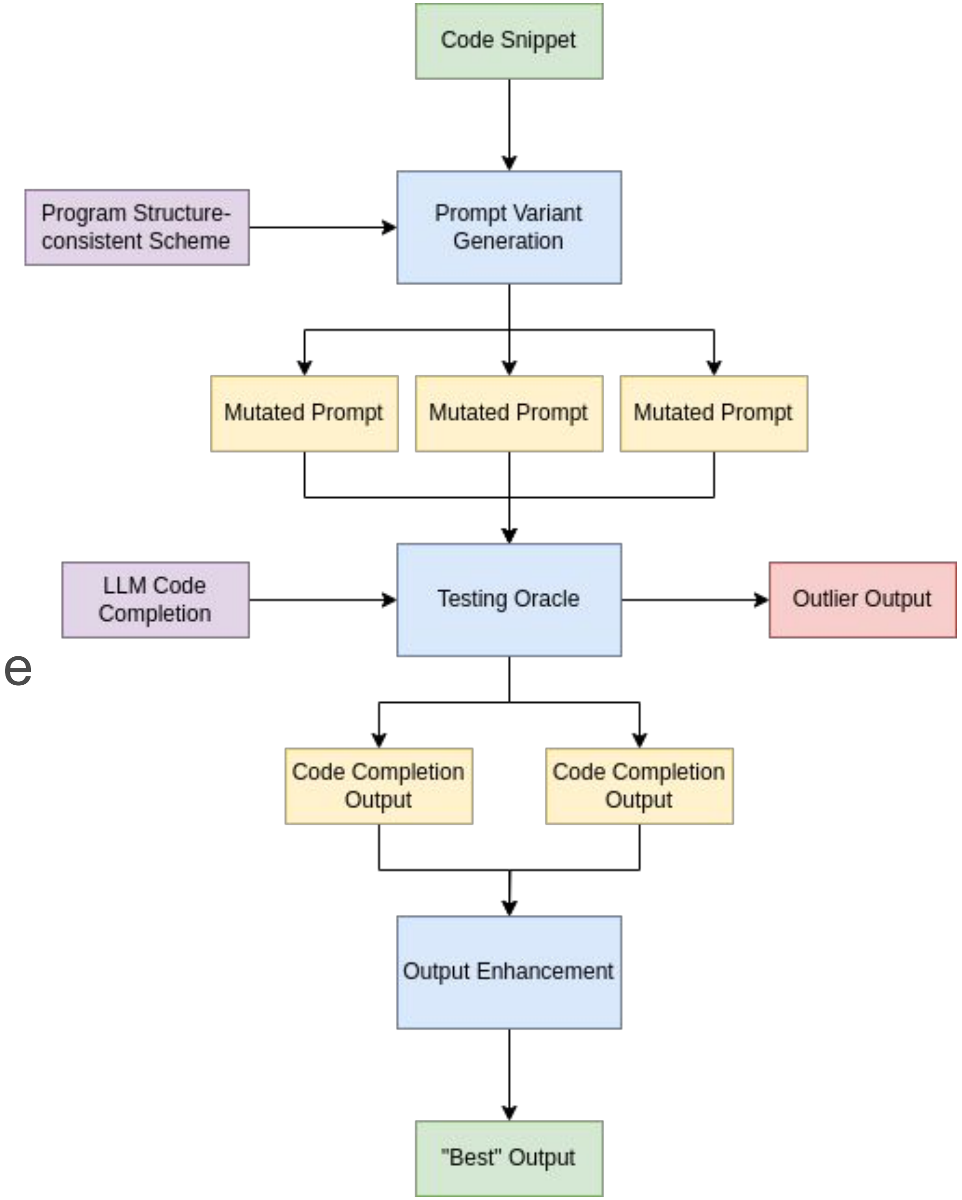
- Automated testing framework for LLM code completion systems.
- 3 main goals/steps:
  - Create mutated yet structurally similar input code prompts
  - Identify inconsistent outputs as potentially erroneous outliers
  - Enhance code generation outputs
- Tested on various code completion systems
  - GitHub Copilot, CodeParrot, GPT-Neo, GPT-J, and CodeGen
  - Completion system can be a “black box”
- Methods are applicable to any language
  - Only Python support currently

# Approach/Methodology

# Main Steps




- CCTest has 3 main steps
  - Generate mutants from given prompts (code snippets)
    - These variants should be identical functionally
    - These variants should be program structure-consistent
  - Identify outliers among the generated autocomplete code
    - Use metrics to find the average output appearance
    - Find outliers that deviate the furthest from that average
      - Label these as defective outputs / “bugs”
  - Exclude the outliers and recompute the average appearance output
    - Select the output closest to the average with outliers removed



# Mutant Prompt Generation

- Prompt variant generation schemes:
  - REP_R, REP_C
    - Rename function parameters
  - REL_R, REL_C
    - Rename local variables
  - IRR
    - Replace arithmetic operators with semantically equivalent forms
  - RTF
    - Replace boolean expressions with semantically equivalent forms
  - GRA_R, GRA_C
    - Insert garbage code that does not alter program semantics
  - INI
    - Insert print statement into prompt

```
// seed prompt
def add(a,b):
    res = a + b
```

```
def add(a,Param1):          // REP_R
    res = a + Param1
def add(a, Add_Param_b): // REP_C
    res = a + Add_Param_b
```

```
// seed prompt
def compare(a,b):
    res = a > b
```

```
def compare(a,b):      // REL_R
    LocalVar1 = a > b
def compare(a,b):      // REL_C
    compare_res = a > b
```

```
def addassign(a,b): // seed
    a += b
    return a
```

```
def addassign(a,b): // IRR
    a = a + b
    return a
```

```
def add (a,b,ignore): // seed
    if ignore:
        return a
```

```
def add (a,b,ignore): // RTF
    if ignore == (b==b):
        return a
```

```
// seed prompt
def add(a, b):
    res = a + b
```

```
def add(a, b):         // GRA_R
    if (False): TempVar = a
    res = a+b
def add(a, b):         // GRA_C
    if (b!=b): Add_TempVar = a
    res = a + b
```

```
// seed prompt
def add(a, b):
    res = a + b
```

```
def add(a, b): // INI
    print(b)
    res = a + b
```

# Identifying Outlier Outputs

- For a given prompt and its mutant, iterate through every code completion output to find similarity between every other output
  - Similarity metric is based on Levenshtein edit distance
  - Obtains square matrix of similarity scores between every output
- If current output is less similar than another output's median similarity T times, then it is an outlier
  - T is a threshold hyperparameter

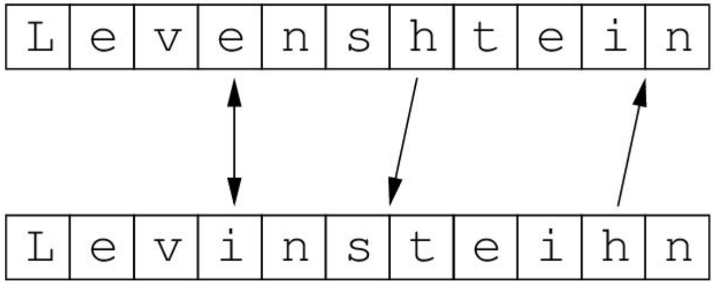
**Algorithm 1** Outlier selection algorithm.

**Input:** $\mathcal{O}$: Code completion output set of size $k$
**Input:** $T$: threshold
**Output:** $\mathcal{L}$: Outliers

```
ScoreMatrix = []
for i in 1 to k do
    for j in i to k do ScoreMatrix[i][j] = Sim(o_i, o_j)
Normalize(ScoreMatrix)
for i in 1 to k do
    count = 0
    for j in 1 to k do
        if ScoreMatrix[i][j] < Median(ScoreMatrix) then
            count = count + 1
            if count ≥ T then
                L.append(o_i)
                break
return L
```

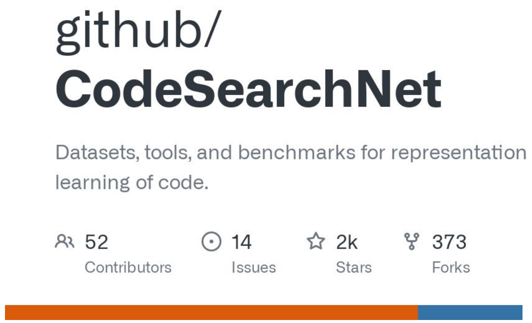
# Code Completion System Enhancement

- Find average similarity score of all code completion outputs
  - Excludes previously found outliers
  - Similarity metric based on Levenshtein edit distance
  - Essentially average value in previously found similarity square matrix (excluding outliers)
- Output with similarity score closest to this average is chosen as the “enhanced” code completion output



# Other Experimental Setup Info

- Dataset used for prompts are from LeetCode and CodesearchNet
  - Total of 2,910 programs + 19,898 mutant prompts = 22,808 total prompts
- Parse Python code into concrete syntax tree and determine which mutations can be performed
- Evaluated code completion systems:
  - GitHub Copilot
  - CodeParrot
  - GPT-Neo
  - GPT-J
  - CodeGen




LeetCode

# Findings/Results

# Mutant Prompt Generation

- pycode-similar used to determine distance between prompts and mutants and, therefore, effectiveness of CCTEST prompt mutation
  - Different distance than Levenshtein edit distance used for outputs
  - pycode-similar based on AST structures
- 98.46% of mutated prompts had distance < 0.1
  - Authors conclude that CCTEST successfully generates structurally-consistent prompt mutants

TABLE IV
DISTRIBUTION OF STRUCTURAL CONSISTENCY SCORES.

| **Distance** | $[0, 0.05]$ | $[0.05, 0.1]$ | $[0.1, 0.15]$ | $[0.15, 0.2]$ | $[0.2, 0.9]$ | $[0.9, 1.0]$ |
|---|---|---|---|---|---|---|
| **Freq. (%)** | 90.16 | 8.30 | 0.92 | 0.30 | 0.32 | 0 |
| **Cumulative Freq. (%)** | 90.16 | 98.46 | 99.38 | 99.68 | 100.0 | 100.0 |

# Identifying Outlier Outputs

- Significant number of outliers=defects found for every value of T
  - Copilot tended to produce the fewest defects
- Increasing threshold T decreases number of detected outliers
- Authors manually validated whether or not detected outliers were correctly classified as "defects"
  - Obtained TPs, FNs, precision, recall, F1 score
  - Determined T=9 appears to be best threshold hyperparameter, F1=0.865
- All mutations appear to contribute towards defects pretty equally

TABLE VI
ASSESSING CCTEST'S FINDINGS WITH MANUAL INVESTIGATION.

| | T=1 | T=3 | T=5 | T=7 | T=9 |
|---|---|---|---|---|---|
| TP | 216 | 342 | 429 | 537 | 689 |
| FN | 26 | 38 | 70 | 90 | 104 |
| Precision | 0.270 | 0.427 | 0.536 | 0.671 | 0.861 |
| Recall | 0.892 | 0.900 | 0.859 | 0.856 | 0.868 |
| F1 score | 0.414 | 0.579 | 0.660 | 0.752 | 0.865 |

TABLE V
OVERVIEW OF OUTLIER DETECTION RESULTS. FOR EACH SYSTEM, WE USE 4909 LEETCODE PROMPTS AND 17899 CODESEARCHNET PROMPTS TO TEST.

| System | #No Results | #Outliers | | | | |
|---|---|---|---|---|---|---|
| | | T= 1 | T= 3 | T= 5 | T= 7 | T= 9 |
| Copilot | 4+37 | 3003 + 12101 | 1347 + 7928 | 803 + 5570 | 559 + 3899 | 293 + 2184 |
| CodeParrot | 1+0 | 4798 + 17605 | 4379 + 16118 | 3631 + 13368 | 2359 + 9023 | 904 + 3778 |
| CodeParrot-small | 2+0 | 4812 + 17611 | 4469 + 16069 | 3776 + 13454 | 2470 + 9344 | 1033 + 4009 |
| GPT-J | 0+0 | 4606 + 17280 | 3776 + 15073 | 2832 + 12005 | 1786 + 7875 | 586 + 3174 |
| GPT-NEO-13B | 1+0 | 4729 + 17427 | 4088 + 15495 | 3311 + 12536 | 2120 + 8462 | 794 + 3463 |
| GPT-NEO-125M | 0+2 | 4734 + 17509 | 4281 + 15556 | 3654 + 12907 | 2523 + 8966 | 1079 + 3700 |
| Codegen-2B-mono | 2+9 | 4661 + 17221 | 3794 + 15112 | 2731 + 12241 | 1638 + 8460 | 639 + 3761 |
| Codegen-6B-mono | 0+0 | 4578 + 17016 | 3575 + 14595 | 2493 + 11546 | 1452 + 7866 | 584 + 3559 |
| Total | 10+48 | 35921 + 133770 | 29709 + 115946 | 23231 + 93627 | 14907 + 63895 | 5912 + 27628 |

# Code Completion System Enhancement

- Found BLEU score and Levenshtein edit similarity between code completion outputs and ground truth from datasets
  - Calculated enhancement ratio as (r = (s'-s)/s), where s' is score after enhancement and s is score before enhancement (no use of mutant prompts)
  - Found average enhancement to be 40% by BLEU score and 62% to 73% by Levenshtein edit distance, depending on the evaluated dataset
- Found each mutation contributed pretty equally to enhancement
- Performed human study in which experts were tasked to score random code completions from 1-5
  - Some were enhanced, some unenhanced
  - Average unenhanced score was 2.32 while enhanced was 3.16
  - 97.8% of respondents believed that enhanced outputs looked better than or equivalent to unenhanced outputs

# Conclusion

- CCTEST is successful at:
  - Creating program structure-consistent code prompt mutations
  - Finding defects in LLM code completion systems
  - Enhancing LLM code completion outputs
- CCTEST would likely be cost-effective to deploy, especially on the server side (e.g. by GitHub Copilot)
- Authors hope CCTEST sets the tone for future work in testing and improvement of code completion systems

# Paper Reference


```
@inproceedings{li2023cctest,
    title={Cctest: Testing and repairing code completion systems},
    author={Li, Zongjie and Wang, Chaozheng and Liu, Zhibo and Wang, Haoxuan and Chen, Dong and Wang, Shuai and Gao, Cuiyun},
    booktitle={2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)},
    pages={1238--1250},
    year={2023},
    organization={IEEE}
}
```