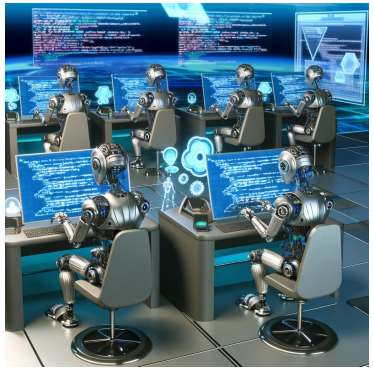
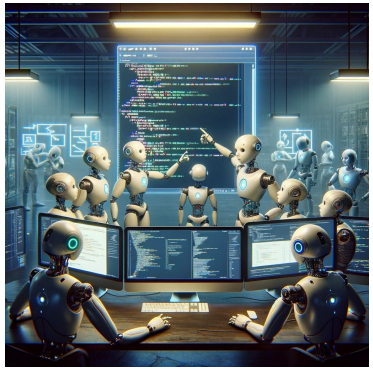
Is Self repair a silver bullet for Code Generation



Theo X. Olausson[1, †], **Jeevana Priya Inala**[2], **Chenglong Wang**[2], **Jianfeng Gao**[2], **Armando Solar-Lezama**[1]
[1]MIT CSAIL [2]Microsoft Research



Presentation by: Shrey Pandit

Aim of the paper

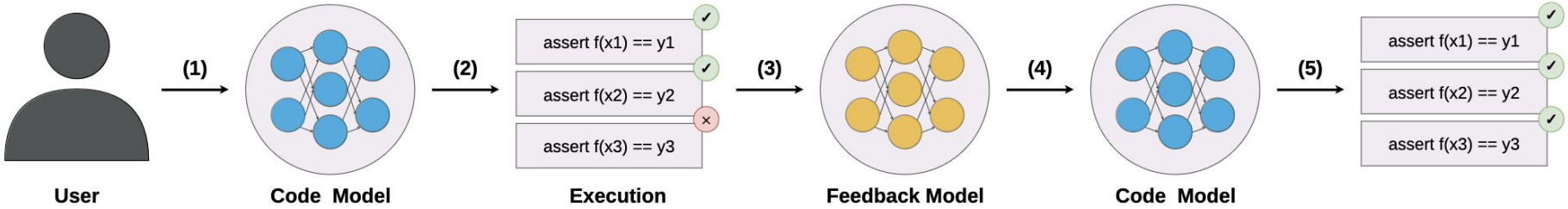
- Self repair requires repetitive invocation of model increasing the computing cost.

- Incorporating this cost, the question is- i.i.d sampling vs repairing at equivalent compute budget.

- In a process of repair (where you sample and then repair) where to spend your budget (in sampling more or creating more repairs)

- How does the quality of feedback (the model generating the feedback) affect the repairs.

Process of self repair



Code Model M_p can be same or different than M_F, this will be useful later for ablations.

Dataset: APPS, HumanEval (both give access to all test cases, public and private)

Code generation & Code execution

Generation:

Given specification (question + input output constraints and any other useful information) generate n_p samples of i.i.d code $\{p_i\}_{i=1}^{n_p} \overset{i.i.d.}{\sim} M_P(\psi)$

Given is a string s representing the day of the week today. s is one of SUN, MON, TUE, WED, THU, FRI, or SAT. After how many days is the next Sunday (tomorrow or later)?

```
# UNIT TESTS
# (EXECUTABLE)
assert f('MON') == 6
assert f('WED') == 4
assert f('SUN') == 7
```

```
def f(s):
  return (7 - ['SUN',  ... , 'FRI', 'SAT'].index(s)) % 7
```

Execution:

If a sample p passes all test cases (public+private) we stop.

Otherwise collect the run/compile time error, $\{e_i\}$ which we receive from the executor, OR an example of wrong input output pair.

Given input 'SUN', the program returned 0, but the expected output was 7.

Feedback generation & Code repair

Feedback generation:

For each wrong program p_i generate n_f feedback strings

$$\{f_{ij}\}_{j=1}^{n_f} \overset{i.i.d.}{\sim} M_F(\psi; p_i; e_i)$$

Feedback model can be same or different from the generation model

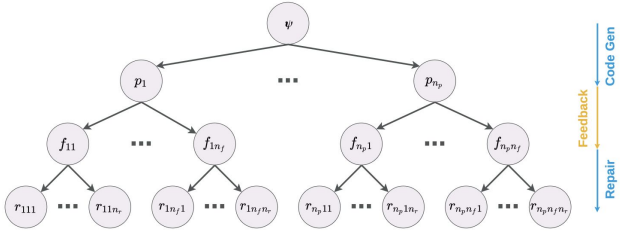
> The code does not account for the case where the input is 'SUN' and the output should be 7. This can be fixed by removing the modulo operation.

Code repair:

For each program p_i and feedback $f_{i,j}$ -> generate n_r candidate repaired programs

$$\{r_{ijk}\}_{k=1}^{n_r} \overset{i.i.d.}{\sim} M_P(\psi; p_i; e_i; f_{ij})$$

Repair Tree: This entire structure can be easily visualised as a tree with specification as the root with programs, and repairs as children



Pass @ k for self-repair

Without self repair - the probability that at least 1 of the k i.i.d programs sampled from the model satisfies the specification

With self repair: samples are drawn during the initial sample stage and also during the repair stage, so will need to modify pass@k.

$|\text{programs}(T)| = n_p + n_p * n_f * n_r$

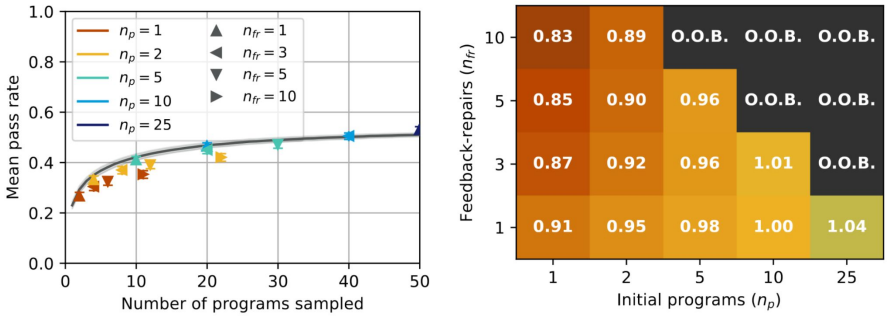
For a fair comparison we should compare k with |programs(T)| as a baseline

Implementation detail -

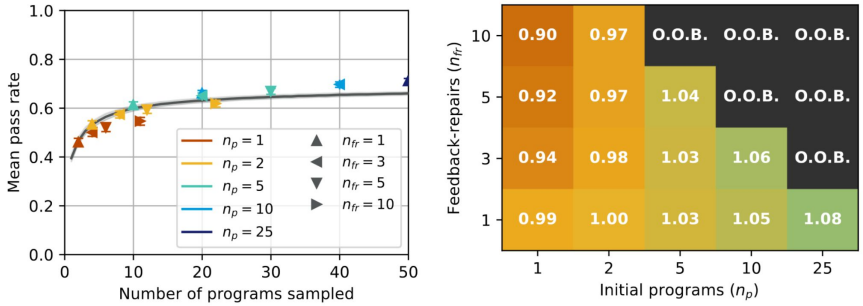
Independently generating soo many repair tree for each setting becomes computationally infeasible.

So authors generate a large tree of $N_p = 50$ and , $N_f = 25$ and then subsample N_t different sub-repair trees from this frozen dataset and average over runs.

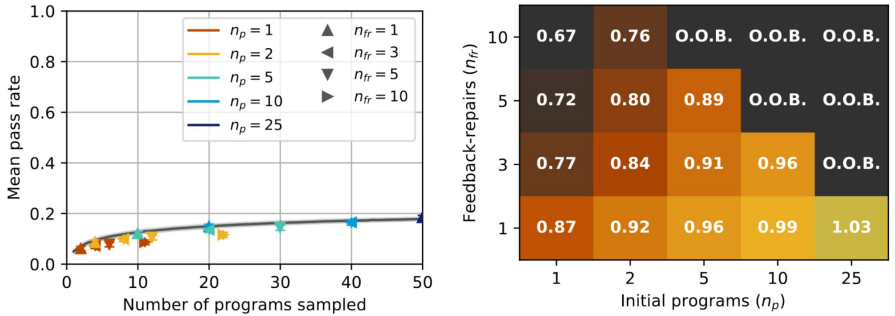
Insight 1 - Improvement with diverse initial samples



(a) GPT-3.5.



(b) GPT-4.

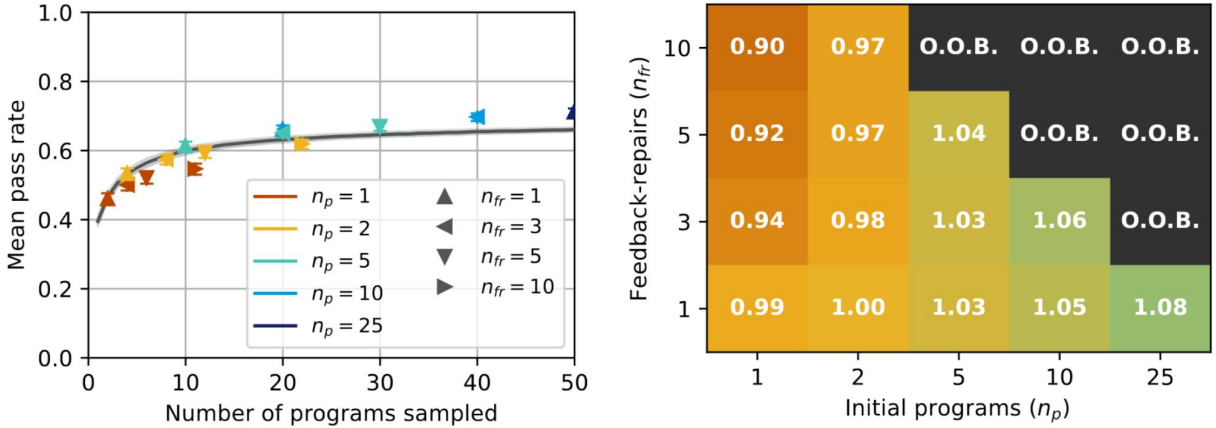


(b) CodeLlama-13b-instruct on APPS.

Value in histogram =
(mean pass rate with self repair) /
(mean pass rate with no repair (when same budget))

So if a value is
= 1 : self repair = baseline (sampling i.i.d)
≥ 1 : self repair better than baseline

Insight 1 - Improvement with diverse initial samples



(b) GPT-4.

- Fixing num of repairs (n_{fr}) if we increase the of feedbacks (n_p) = moving right along x axis on heatmap : Consistent performance gain
- Fixing the number of programs (n_p) and increasing the number of repairs (n_{fr}) = moving up along the y axis : Not worth the additional compute cost (even worse sometimes)
- So for a fixed budget, most imp factor is if self repair would work or not is the diversity of the base samples that are generated up front

Extra Results (from supplementary)

Table 2: Repair success rates in various settings. The repair success rate is computed as `number_of_passing_repairs` / `total_number_of_repairs_sampled`.

Dataset	Difficulty	Model	Repair Success Rate
APPS	introductory	Code Llama	2.8%
		Code Llama+GPT-3.5	5.4%
		GPT-3.5	13.7%
		GPT-3.5+GPT-4	29.1%
		GPT-4	28.8%
	interview	Code Llama	1.0%
		Code Llama+GPT-3.5	1.9%
		GPT-3.5	4.2%
		GPT-3.5+GPT-4	11.2%
		GPT-4	8.7%
	competition	Code Llama	0.1%
		Code Llama+GPT-3.5	0.4%
		GPT-3.5	1.5%
		GPT-3.5+GPT-4	3.3%
		GPT-4	8.6%
	overall	Code Llama	1.1%
		Code Llama+GPT-3.5	2.2%
		GPT-3.5	4.7%
		GPT-3.5+GPT-4	11.5%
		GPT-4	10.8%

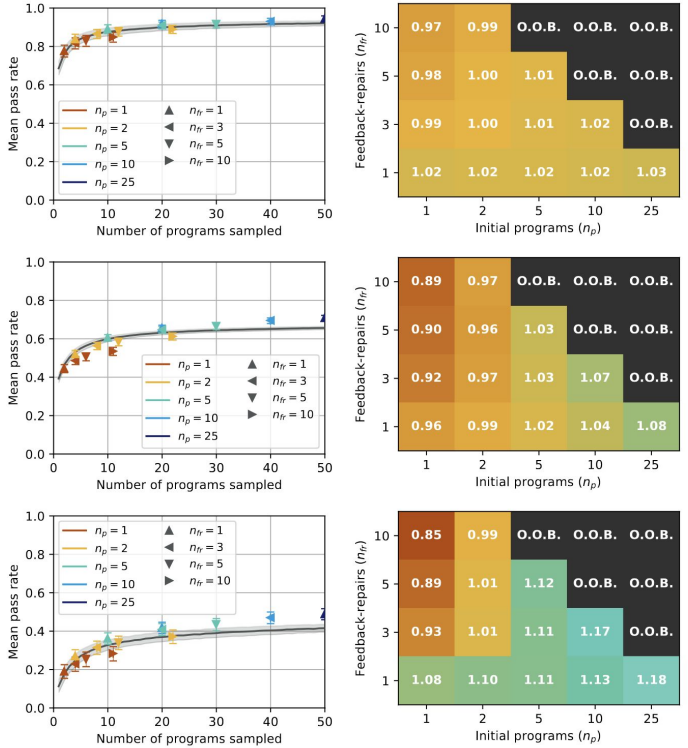
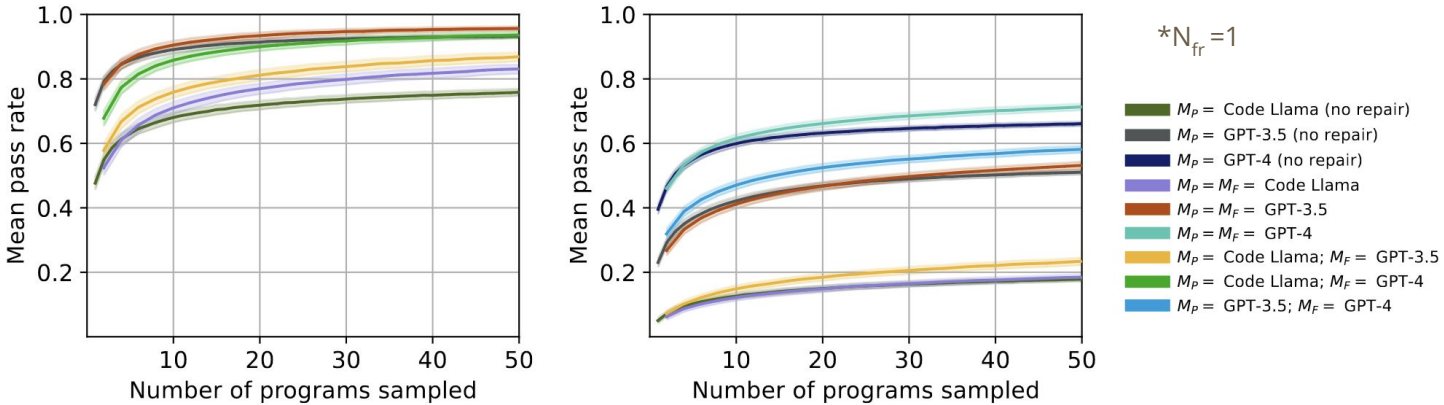


Figure 15: GPT-4 results from Figure 3 (Section 4.1) per APPS difficulty (row), from top to bottom: `introductory`, `interview`, and `competition`.

Feedback improvement leads to better performance



M_p and M_f can be independently set, so they try M_p is set to be a weaker model and M_f is made to be a stronger model.

Model performance improves with a stronger feedback model.

Summary: Feedback stage is crucial and improving it relives the bottleneck in self-repair.

Q: Is doesn't look fair to compare M_p = M_f Code Llama with M_p= Code Llama and M_f = GPT 3.5, because the computational cost would be vastly different?

Prompt example

Listing 1: Code generation prompt for call-based tasks.

```
=====system=====
You are an expert Python programmer. You will be given a question (problem specification) and will generate a
      correct Python program that matches the specification and passes all tests. You will NOT return
      anything except for the program. Put your fixed program within code delimiters, for example: ```python
# YOUR CODE HERE
```.
=====user=====
QUESTION
A palindrome is a word, phrase, number, or other sequence of characters which reads the same backward as
      forward. Examples of numerical palindromes are: `2332, 110011, 54322345`

For a given number ```num```, write a function which returns an array of all the numerical palindromes
      contained within each number. The array should be sorted in ascending order and any duplicates should
      be removed.

In this kata, single digit numbers and numbers which start or end with zeros (such as `010` and `00`) are
      NOT considered valid numerical palindromes.

If `num` contains no valid palindromes, return `"No palindromes found"`.
Otherwise, return `"Not valid"` if the input is not an integer or is less than `0`.

Examples

```
palindrome(1221)      =>  [22, 1221]
palindrome(34322122)  =>  [22, 212, 343, 22122]
palindrome(1001331)   =>  [33, 1001, 1331]
palindrome(1294)      =>  "No palindromes found"
palindrome("1221")    =>  "Not valid"
```

Other Kata in this Series:

Numerical Palindrome #1
Numerical Palindrome #1.5
Numerical Palindrome #2
Numerical Palindrome #3
Numerical Palindrome #3.5
Numerical Palindrome #4
Numerical Palindrome #5
Use Call-Based Format
=====assistant=====
```python
def is_pal(s):
    return s == s[::-1]
```

Listing 3: Feedback-only prompt.

```
=====system=====
You are a helpful programming assistant and an expert Python programmer. You are helping a user debug a
      program. The user has written some code, but it has some errors and is not passing the tests. You will
      help the user by giving a concise (at most 2-3 sentences) textual explanation of what is wrong with the
      code. You will *not* generate any code, because the user wants to fix the code themselves.
=====user=====
### QUESTION
A palindrome is a word, phrase, number, or other sequence of characters which reads the same backward as
      forward. Examples of numerical palindromes are: `2332, 110011, 54322345`

For a given number ```num```, write a function which returns an array of all the numerical palindromes
      contained within each number. The array should be sorted in ascending order and any duplicates should
      be removed.

In this kata, single digit numbers and numbers which start or end with zeros (such as `010` and `00`) are
      **NOT** considered valid numerical palindromes.

If `num` contains no valid palindromes, return `"No palindromes found"`.
Otherwise, return `"Not valid"` if the input is not an integer or is less than `0`.


## Examples

```
palindrome(1221)      =>  [22, 1221]
palindrome(34322122)  =>  [22, 212, 343, 22122]
palindrome(1001331)   =>  [33, 1001, 1331]
palindrome(1294)      =>  "No palindromes found"
palindrome("1221")    =>  "Not valid"
```

Other Kata in this Series:

Numerical Palindrome #1
Numerical Palindrome #1.5
Numerical Palindrome #2
Numerical Palindrome #3
Numerical Palindrome #3.5
Numerical Palindrome #4
Numerical Palindrome #5
Use Call-Based Format
INCORRECT PYTHON CODE
```python
def is_pal(s):
    return s == s[::-1]

def palindrome(num):
    if not isinstance(num, int) or num < 0:
        return 'Not valid'
    s = str(num)
    pals = set()
    for i, ch in enumerate(s):
        for j in range(i + 2, len(s) + 1):
            test = s[i:j]
            if is_pal(test):
                pals.add(test)
    return sorted(int(x) for x in pals) or 'No palindromes found'
```
The code above is wrong and contains a bug. Given input "1001331" the output was "['0', '33', '1001',
      '1331']" but the expected output was "['33', '1001', '1331']".
=====assistant=====
The following is a concise explanation of the issue: The problem description states that numbers which start
      or end with zeros (such as `010` and `00`) are NOT considered valid numerical palindromes. However, the
      code above does not take this into account and therefore returns `00` as a valid palindrome. This can
      be fixed by checking if the first or last character is `0` before adding the string to the set of
      palindromes.
```

Human Feedback on GPT4 repair

- Asking humans for feedback for the buggy code results in increase in success rate by 1.58x

- With increasing difficulty of questions this gap increased

- GPT4 feedbacks are more likely to be inaccurate (32/80) as compared to human feedback (7/80)

Table 1: Success rate of repair with GPT-4's explanations vs. with those of our human participants.

Difficulty	Introductory	Interview	Competition	Overall
GPT-4 Feedback	42.64%	19.33%	3.67%	33.30%
Human Feedback	62.21%	45.67%	14.67%	52.60%

Isn't this obvious? 🤔

Takeaways

1) When cost of carrying out repair is taken into account, the performance gains from self-repair are modest, relying mostly on diversity of initial programs
2) When feedback of a weaker model is replaced by a stronger model, performance increased significantly.
3) Replacing GPT-4 self generated feedback with a human programmer feedback improved the performance

So in the process of self repair, the models are held back by their inability to produce useful feedback of why the code is wrong.