

*Large Language Model based **Code**
Completion is an Effective Genetic
Improvement Mutation*

Jingyuan Wang and Carol Hanna and Justyna Petke
GI Workshop @ ICSE 2025



Genetic Improvement

Large Language Models

**Our
Paper**

Genetic Improvement

An automated process that uses search-based techniques to enhance both functional and non-functional aspects of existing software

Large Language Models

Neural networks trained on vast text data for tasks like text generation and question answering.

**Our
Paper**

Genetic Improvement

More diverse patches
compared to LLMs [1]

Large Language Models

Patches more likely to
compile and pass
tests[1]

**Our
Paper**

[1] A. E. I. Brownlee, J. Callan, K. Even-Mendoza, A. Geiger, C. Hanna, J. Petke, F. Sarro, and D. Sobania, "Large language model based mutations in genetic improvement," ASE, 2024.

We experiment with **two new concepts**

The masking operator



Combining Masking with
traditional GI



The masking operator

```
def bubble_sort(arr):  
    for i in range(len(arr)-1):  
        for j in range(len(arr)-1-i):  
            if arr[j] < arr[j+1]:  
                arr[j], arr[j+1] = arr[j+1], arr[j]  
    return arr
```

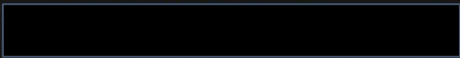
Buggy Bubble sort

The masking operator

```
def bubble_sort(arr):  
    for i in range(len(arr)-1):  
        for j in range(len(arr)-1-i):  
            if arr[j] < arr[j+1]:  
                arr[j], arr[j+1] = arr[j+1], arr[j]  
    return arr
```

Buggy Bubble Sort

The masking operator

```
def bubble_sort(arr):  
    for i in range(len(arr)-1):  
        for j in range(len(arr)-1-i):  
              
            arr[j], arr[j+1] = arr[j+1], arr[j]  
    return arr
```

Masking

The masking operator

```
def bubble_sort(arr):  
    for i in range(len(arr)-1):  
        for j in range(len(arr)-1-i):  
            if arr[j] > arr[j+1]:  
                arr[j], arr[j+1] = arr[j+1], arr[j]  
    return arr
```

Patched Bubble Sort

Research Questions

RQ1: Does the masking mutation create a denser **search space** with more unique, compiling, and test-passing patches compared to the replacement mutation?

RQ2: Can the masking mutation identify **patches that yield greater runtime improvements** compared to the replacement mutation when using the same LLMs on identical datasets?

RQ3: What is the **runtime efficiency of the masking mutation** compared to the replacement mutation?

RQ4: How does the **combination of masking and traditional GI** mutations compare to using either method alone?

Experimental Setup

- Gin Framework
- **5** Open-source projects: Jcodec, JUnit4, Gson, Commons-net, and Karate
- **4** LLMs: Gemma2:2B, Gemma2:9B, Llama3.1:8B Mistral:7B
- **2** Search strategies: local and random search
- Mini-experiments to identify impactful statement types to be used in the masking statement selection
- 30%, 50%, 70% probabilities of selecting the masking mutation over traditional GI mutations

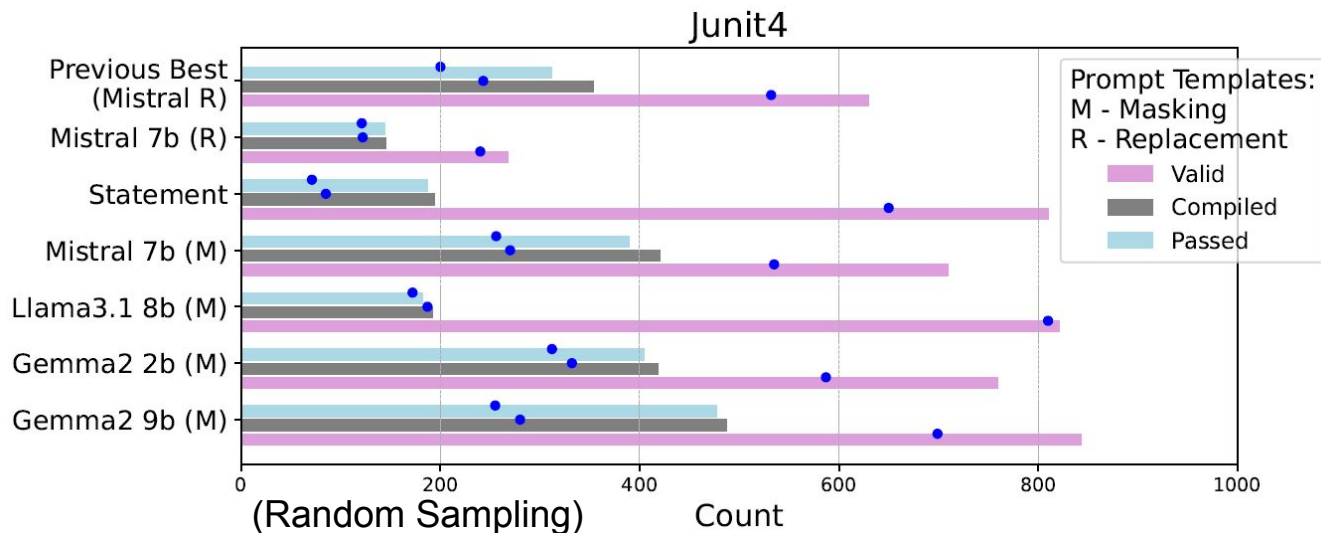
Masking Prompt

Please replace <<PLACEHOLDER>> sign in the method below with meaningfull implementation.

<<Masked Code>>

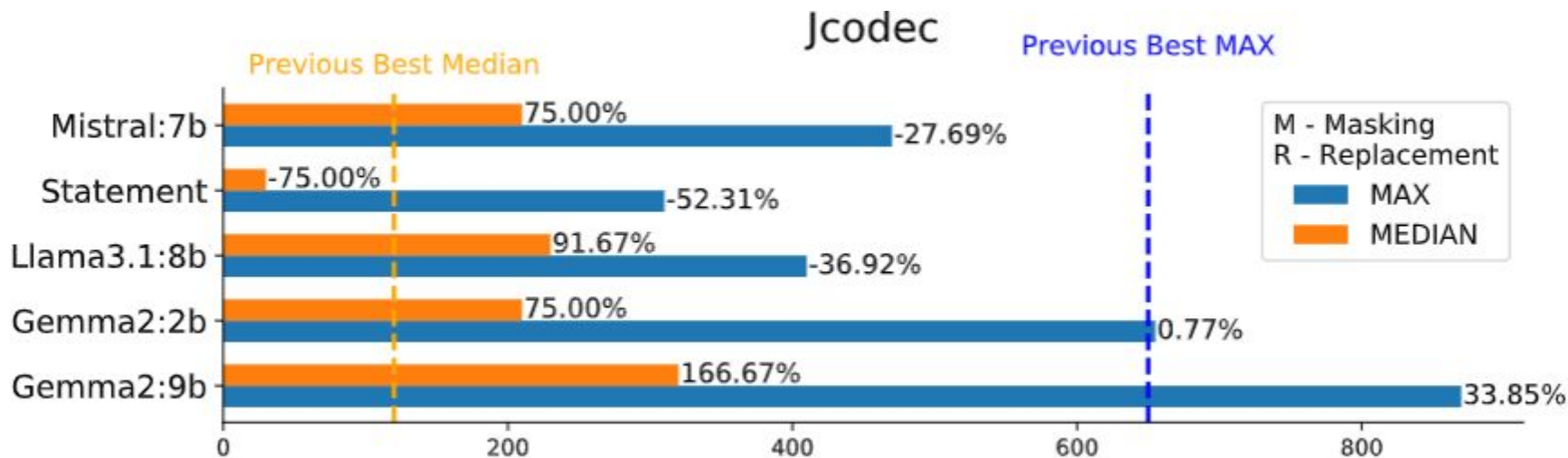
"This code belongs to project <<Project Name>>. Wrap all code in curly braces, if it is not already. Do not include any method or class declarations. Label all code as java.

RQ1: Does the masking mutation create a denser **search space** with more unique, compiling, and test-passing patches compared to the replacement mutation?



Answer to RQ1: The GI process using the LLM-based masking mutation operator provides a **denser search space** with compiling and test-passing patches compared to the LLM-based replacement mutation operator, although it produces **slightly fewer unique valid** patches.

RQ2: Can the masking mutation identify **patches** that yield **greater runtime improvements** compared to the replacement mutation when using the same LLMs on identical datasets?



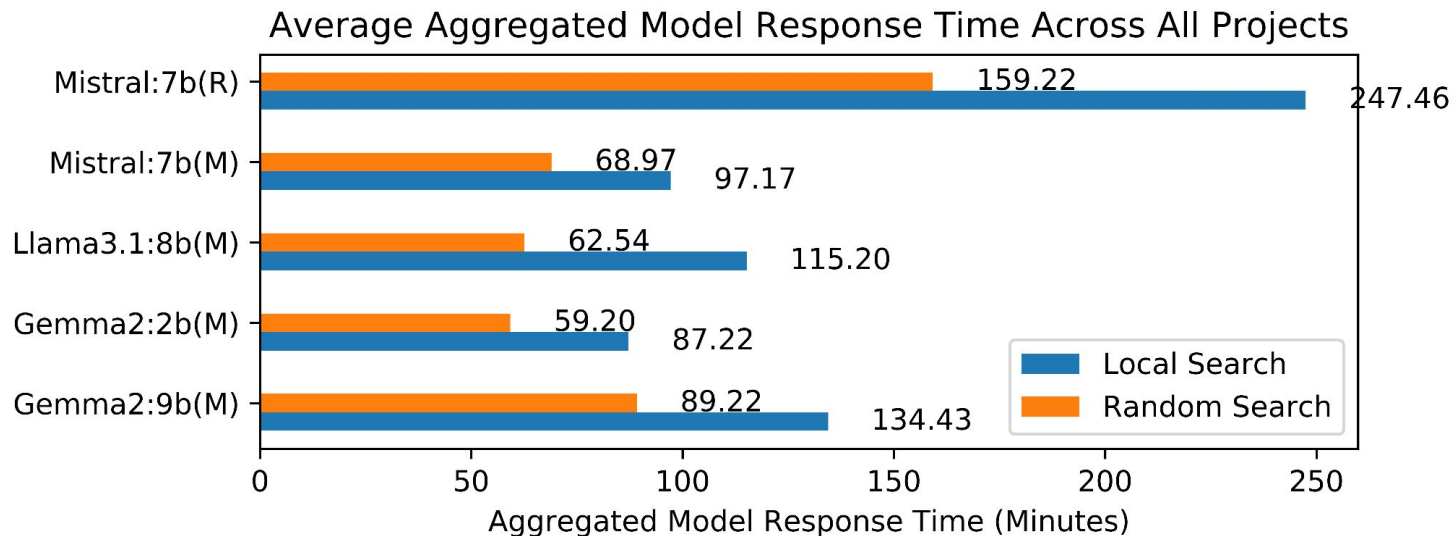
Answer to RQ2: The LLM-based masking mutation outperformed the LLM-based replacement mutation in 4 out of 5 projects.

RQ2: Can the masking mutation identify **patches that yield greater runtime improvements** compared to the replacement mutation when using the same LLMs on identical datasets?

	JCodec	Gson	Commons-net	Karate	Junit
Previous Best Mistral:7B (R)	28	47	24	83	74
Statement	18	17	34	65	57
Mistral:7B(R)	74	3	25	29	25
Mistral:7B(M)	59	59	27	124	98
Llama3.1:8B(M)	45	35	19	98	37
Gemma2:2B(M)	98	27	52	132	78
Gemma2:9B(M)	132	66	115	152	145

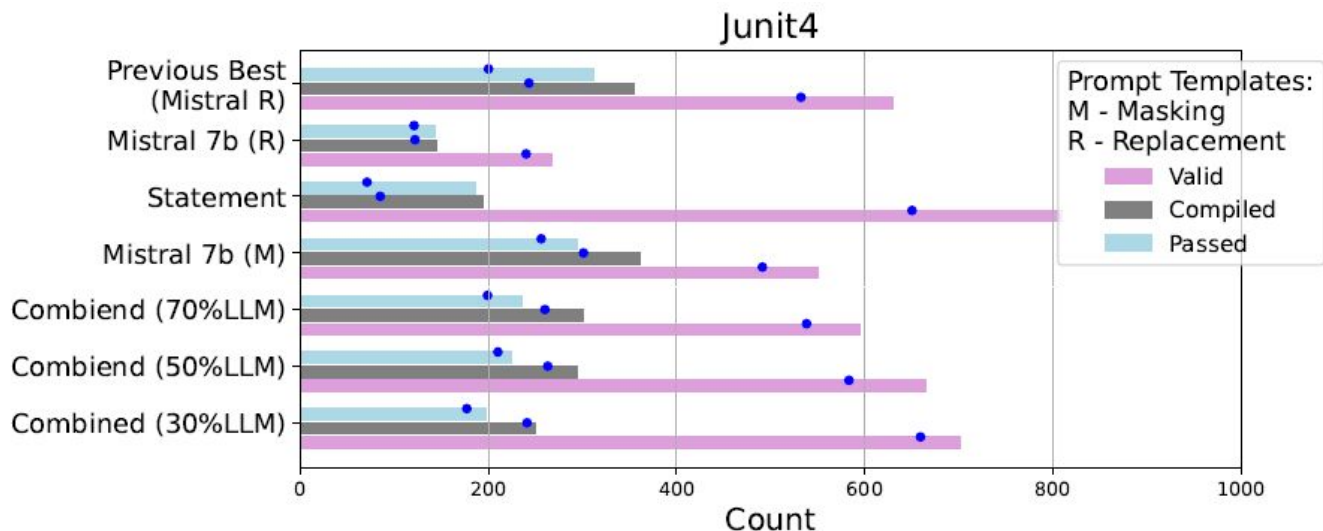
Answer to RQ2: The masking mutation generated more performance-improving patches and consistently outperformed traditional GI mutations across all experiments.

RQ3: What is the **runtime efficiency of the masking mutation** compared to the replacement mutation?



Answer to RQ3: The GI process using the LLM-based masking mutation operator offers competitive performance while significantly reducing model response time compared to the replacement mutation.

RQ4: How does the **combination of masking and traditional GI** mutations compare to using either method alone?



Answer to RQ4: Alternating between traditional GI mutations and LLM-based masking mutations generates more valid patches than masking alone, but inconsistent runtime improvements suggest further investigation is needed to maximise this approach's effectiveness.

Common LLM Response Issues

Category 1: Incomplete Code Returned

e.g. only instructions or examples instead of executable code.

Category 2: Code Not in Expected Format

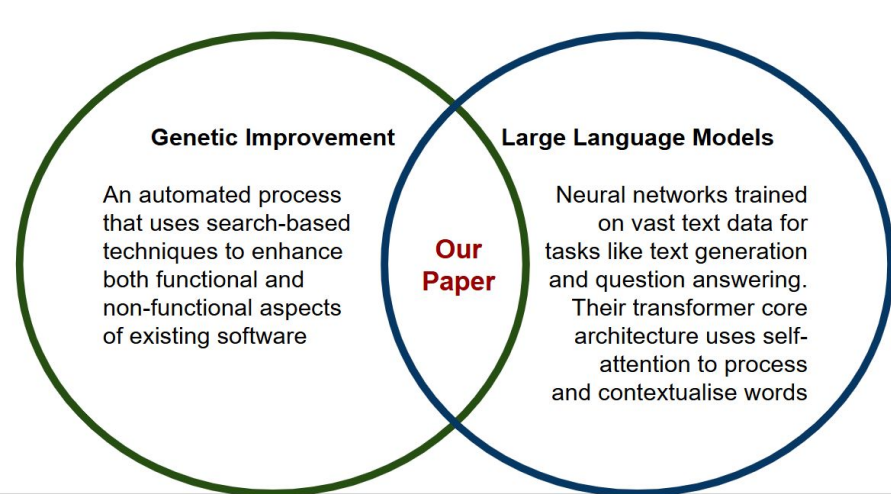
20% of cases deviated from the expected format and didn't allow for automated extraction

Category 3: Meaningful Code Without Improvement

e.g. returning input code unchanged, replacing placeholder with comments, and producing repetitive outputs across runs.

Threats to Validity

- Possible variability in results: black-box models + non-determinism
- Runtime improvement measurement
- Prompt Engineering



Research Questions

RQ1: Does the masking mutation create a denser **search space** with more unique, compiling, and test-passing patches compared to the replacement mutation?

RQ2: Can the masking mutation identify **patches that yield greater runtime improvements** compared to the replacement mutation when using the same LLMs on identical datasets?

RQ3: What is the **runtime efficiency of the masking mutation** compared to the replacement mutation?

RQ4: How does the **combination of masking and traditional GI** mutations compare to using either method alone?

Common LLM Response Issues

Category 1: Incomplete Code Returned

e.g. only instructions or examples instead of executable code.

Category 2: Code Not in Expected Format

20% of cases deviated from the expected format and didn't allow for automated extraction

Category 3: Meaningful Code Without Improvement

e.g. returning input code unchanged, replacing placeholder with comments, and producing repetitive outputs across runs.

	JCodec	Gson	Commons-net	Karate	Junit
Previous Best					
Mistral:7B (R)	28	47	24	83	74
Statement	18	17	34	65	57
Mistral:7B(R)	74	3	25	29	25
Mistral:7B(M)	59	59	27	124	98
Llama3.1:8B(M)	45	35	19	98	37
Gemma2:2B(M)	98	27	52	132	78
Gemma2:9B(M)	132	66	115	152	145