PyVeritas: On Verifying Python via LLM-Based Transpilation and Bounded Model Checking for C

Pedro Orvalho[1]*, and Marta Kwiatkowska[1]

[1] Department of Computer Science, University of Oxford, Oxford, UK

Post-AI Formal Methods Workshop @ AAAI 2026

Singapore, 26 January 2026

*PO is now affiliated with IIIA-CSIC, Spain.

Motivation

- Python has become the **dominant language** for general-purpose programming.

Motivation

- Python has become the **dominant language** for general-purpose programming.
- Yet it **lacks robust tools for formal verification**.

Motivation

- Python has become the **dominant language** for general-purpose programming.
- Yet it **lacks robust tools for formal verification**.
- **Languages such as C benefit from mature model checkers**, for example CBMC [Clarke et al., 2004], which enable exhaustive symbolic reasoning and fault localisation.

Motivation

- Python has become the **dominant language** for general-purpose programming.
- Yet it **lacks robust tools for formal verification**.
- **Languages such as C benefit from mature model checkers**, for example CBMC [Clarke et al., 2004], which enable exhaustive symbolic reasoning and fault localisation.
- The complexity of Python, coupled with the verbosity of existing transpilers (e.g., Cython), have historically limited the applicability of formal verification to Python.

Motivation

```python
def distributeCandies(n: int, limit: int) -> int:
    limit = min(limit, n)
    ans = 0
    ans = 0 + 1
    for i in range(limit + 1):
        if n - i > limit * 2:
            continue
        ans += min(limit, n-i)-max(0, n-i-limit)+1
    return ans
assert distributeCandies(n = 5, limit = 2) == 3
```

Motivation

```
1  def distributeCandies(n: int, limit: int) -> int:
2      limit = min(limit, n)
3      ans = 0
4      ans = 0 + 1
5      for i in range(limit + 1):
6          if n - i > limit * 2:
7              continue
8          ans += min(limit, n-i)-max(0, n-i-limit)+1
9      return ans
10 assert distributeCandies(n = 5, limit = 2) == 3
```

- Line 4 is an accidental duplicate that introduces a subtle off-by-one bug.

Motivation

```python
1  def distributeCandies(n: int, limit: int) -> int:
2      limit = min(limit, n)
3      ans = 0
4      ans = 0 + 1
5      for i in range(limit + 1):
6          if n - i > limit * 2:
7              continue
8          ans += min(limit, n-i)-max(0, n-i-limit)+1
9      return ans
10 assert distributeCandies(n = 5, limit = 2) == 3
```

- Line 4 is an accidental duplicate that introduces a subtle off-by-one bug.
- Existing model checkers (e.g., ESBMC-Python [Farias et al., 2024]) **support only small subsets** of Python.

Motivation

```python
1  def distributeCandies(n: int, limit: int) -> int:
2      limit = min(limit, n)
3      ans = 0
4      ans = 0 + 1
5      for i in range(limit + 1):
6          if n - i > limit * 2:
7              continue
8          ans += min(limit, n-i)-max(0, n-i-limit)+1
9      return ans
10 assert distributeCandies(n = 5, limit = 2) == 3
```

- Line 4 is an accidental duplicate that introduces a subtle off-by-one bug.
- Existing model checkers (e.g., ESBMC-Python [Farias et al., 2024]) **support only small subsets** of Python.
- For instance, ESBMC-Python **cannot verify this Python program**.

PyVeritas

A novel tool for **verification and fault localisation for Python programs** by leveraging Large Language Model (LLM)-based transpilation to C.

PyVeritas

A novel tool for **verification and fault localisation for Python programs** by leveraging Large Language Model (LLM)-based transpilation to C.

1. **Transpiles Python, using an LLM**, to a semantics-preserving C version.

PyVeritas

A novel tool for **verification and fault localisation for Python programs** by leveraging Large Language Model (LLM)-based transpilation to C.

1. **Transpiles Python, using an LLM**, to a semantics-preserving C version.
2. Runs C model checkers (e.g., CBMC) to **check assertions and produce counterexamples through bounded model checking**.

PyVeritas

A novel tool for **verification and fault localisation for Python programs** by leveraging Large Language Model (LLM)-based transpilation to C.

1. **Transpiles Python, using an LLM**, to a semantics-preserving C version.
2. Runs C model checkers (e.g., CBMC) to **check assertions and produce counterexamples through bounded model checking**.
3. Runs **MaxSAT-based fault localisation** tools (e.g., CFaults [Orvalho et al., 2024]) to find the bugs in the C version.

PyVeritas

A novel tool for **verification and fault localisation for Python programs** by leveraging Large Language Model (LLM)-based transpilation to C.

1. **Transpiles Python, using an LLM**, to a semantics-preserving C version.
2. Runs C model checkers (e.g., CBMC) to **check assertions and produce counterexamples through bounded model checking**.
3. Runs **MaxSAT-based fault localisation** tools (e.g., CFaults [Orvalho et al., 2024]) to find the bugs in the C version.
4. Maps the buggy statements back to the Python code, providing **useful feedback**.

PyVeritas

A novel tool for **verification and fault localisation for Python programs** by leveraging Large Language Model (LLM)-based transpilation to C.

1. **Transpiles Python, using an LLM**, to a semantics-preserving C version.
2. Runs C model checkers (e.g., CBMC) to **check assertions and produce counterexamples through bounded model checking**.
3. Runs **MaxSAT-based fault localisation** tools (e.g., CFaults [Orvalho et al., 2024]) to find the bugs in the C version.
4. Maps the buggy statements back to the Python code, providing **useful feedback**.

Note: This is an **interim approach** until a robust, native Python symbolic model checker is available. Because PyVeritas **relies on LLMs** for transpilation and for mapping localised statements back to Python, **no completeness guarantees** are provided.

Running Example

- PyVeritas asks an LLM to transpiled this buggy Python program

```
def distributeCandies(n: int, limit: int) -> int:
    limit = min(limit, n)
    ans = 0
    ans = 0 + 1
    for i in range(limit + 1):
        if n - i > limit * 2:
            continue
        ans += min(limit, n-i)-max(0, n-i-limit)+1
    return ans
assert distributeCandies(n = 5, limit = 2) == 3
```

Running Example

- PyVeritas runs CBMC to check that the program is failing the assertion.

```c
int distributeCandies(int n, int limit) {
    limit = (limit < n) ? limit : n;
    int ans = 0;
    ans = 0 + 1;
    for (int i = 0; i <= limit; i++) {
        if (n - i > limit * 2) {
            continue;
        }
        ans += ((limit < n-i) ? limit : (n-i)) -
        ((n-i-limit > 0) ?(n-i-limit) : 0) + 1;
    }
    return ans;
}
int main() {
    assert(distributeCandies(5, 2) == 3);
    return 0;
}
```

Running Example

- PyVeritas uses CFaults to localise the buggy assignment in the C code.

```c
int distributeCandies(int n, int limit) {
    limit = (limit < n) ? limit : n;
    int ans = 0;
    ans = 0 + 1;
    for (int i = 0; i <= limit; i++) {
        if (n - i > limit * 2) {
            continue;
        }
        ans += ((limit < n-i) ? limit : (n-i)) -
        ((n-i-limit > 0) ?(n-i-limit) : 0) + 1;
    }
    return ans;
}
int main() {
    assert(distributeCandies(5, 2) == 3);
    return 0;
}
```

Running Example

- PyVeritas maps the localised buggy statement back to the original Python source code by leveraging the same LLM used during transpilation.

````
```python
    ans = 0 + 1
```
````

Running Example

- PyVeritas maps the localised buggy statement back to the original Python source code by leveraging the same LLM used during transpilation.

````
```python
    ans = 0 + 1
```
````

Thus, PyVeritas provides precise localisation for simple bugs in Python.

LLM-based Transpilation of Correct Code

Language Model	LiveCodeBench	Refactory
Qwen2.5-Coder (32B)	83.7%	92.0%
DeepSeek-Coder-V2 (16B)	65.1%	64.8%
GraniteCode (8B)	55.9%	52.0%
Llama3.2 (3B)	43.0%	28.0%

Table 1: Verification success rates for each LLM on both benchmarks. Percentages indicate the proportion of C programs that were successfully verified by CBMC and judged semantically equivalent to the original Python code.

MaxSAT-Based Fault Localisation

	Bug: **Wrong Binary Operator (WBO)**			
LLMs	**% Correct Bug Found**	**% Other Bugs**	**% Transpiled Fixed Code**	**% Compilation Err**
Qwen2.5-Coder (32B)	16.9%	20.2%	62.6%	0.3%
DeepSeek-Coder-V2 (16B)	16.1%	36.8%	42.9%	4.2%
GraniteCode (8B)	41.6%	34.6%	15.2%	8.6%
Llama3.2 (3B)	22.7%	47.9%	17.7%	11.6%
	Bug: **Assignment Duplication with Constant (ADC)**			
	% Correct Bug Found	**% Other Bugs**	**% Transpiled Fixed Code**	**% Compilation Err**
Qwen2.5-Coder (32B)	7.6%	11.0%	81.4%	0.0%
DeepSeek-Coder-V2 (16B)	6.7%	21.4%	69.5%	2.4%
GraniteCode (8B)	39.5%	11.9%	36.7%	11.9%
Llama3.2 (3B)	18.6%	39.0%	34.3%	8.1%

Table 2: Results of MaxSAT-Based Fault Localisation with PyVeritas on LiveCodeBench using WBO and ADC bugs.

Takeaway Message

- We propose **PyVeritas**, a novel framework for **verification and fault localisation for Python programs** by leveraging LLM-based transpilation to C.

Takeaway Message

- We propose **PyVeritas**, a novel framework for **verification and fault localisation for Python programs** by leveraging LLM-based transpilation to C.
- PyVeritas combines **LLM-based code transpilation, bounded model checking** with CBMC, and **MaxSAT-based fault localisation** using CFaults.

Takeaway Message

- We propose **PyVeritas**, a novel framework for **verification and fault localisation for Python programs** by leveraging LLM-based transpilation to C.
- PyVeritas combines **LLM-based code transpilation, bounded model checking** with CBMC, and **MaxSAT-based fault localisation** using CFaults.
- PyVeritas accurately **verifies small yet non-trivial Python programs where native verification tools fall short**.

Takeaway Message

- We propose **PyVeritas**, a novel framework for **verification and fault localisation for Python programs** by leveraging LLM-based transpilation to C.
- PyVeritas combines **LLM-based code transpilation, bounded model checking** with CBMC, and **MaxSAT-based fault localisation** using CFaults.
- PyVeritas accurately **verifies small yet non-trivial Python programs where native verification tools fall short**.
- PyVeritas can map localised faults in transpiled C code back to the original Python code, providing **valuable feedback**.

Thank you!

https://pmorvalho.github.io

References

Edmund M. Clarke and Daniel Kroening and Flavio Lerda (2004)
A Tool for Checking ANSI-C Programs
TACAS 2004.

Bruno Farias and Rafael Menezes and Eddie B. de Lima Filho and Youcheng Sun and Lucas C. Cordeiro (2024)
ESBMC-Python: A Bounded Model Checker for Python Programs.
ISSTA 2024.

P. Orvalho and M. Janota and V. Manquinho (2024)
CFaults: Model-Based Diagnosis for Fault Localization in C with Multiple Test Cases.
Formal Methods (FM) 2024.