MANCHESTER 1824
The University of Manchester

ASE 2024 SACRAMENTO

ACM SIGSOFT Distinguished Paper Award

LLM-Generated Invariants for Bounded Model Checking Without Loop Unrolling




Muhammad Adil Anwar Pirzada

Ahmed Bhayat

Lucas Carvalho Cordeiro

Giles Reger

SCorCH

Is this program Correct?

```
int main()
{
  int x = 0;
  int y = 50;
  while (x < 100) {
    x = x + 1;
    if(x > 50) {
      y = y + 1;
    }
  }
  __VERIFIER_assert(y == 100);
}
```

The University of Manchester

With Bounded Model Checking

```
int main()
{
  int x = 0;
  int y = 50;
  x = x + 1;
  if(x > 50) {
    y = y + 1;
  }
  ... // 99 other loop unrollings
  __VERIFIER_assert(y == 100);
}
```

BMC can discover bugs but needs help proving correctness.

Expensive.

BMC struggles with loops that can’t be statically bound or if they have a large bound.

Loop Invariants

Inductive loop invariant:

Is a logical assertion that holds at a loop head whenever the program passes that location.

Base Case:

The invariant holds before the first iteration of the loop.

Step Case:

The invariant holds for every iteration of the loop.

The University of Manchester

Research Motivation

Synthesise Loop Invariants

Traditional techniques.
(Limitations)

Neural techniques.

LLMs.

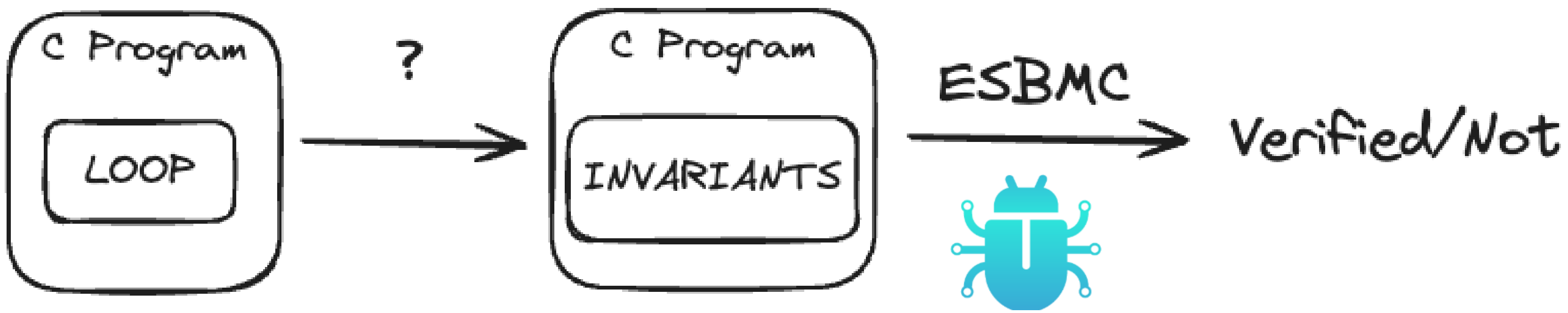
The University of Manchester

Replace Loop with Loop Invariants

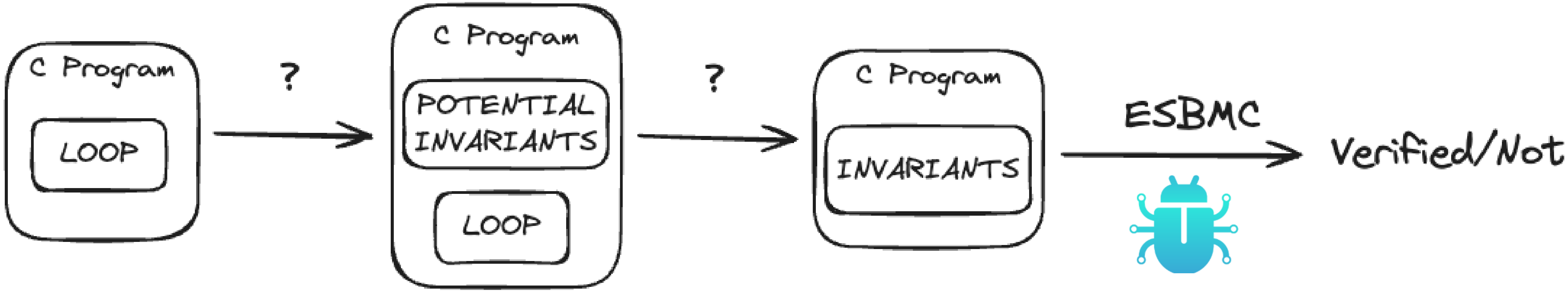
```
while (x < 100) {
   x = x + 1;
   if(x > 50) {
      y = y + 1;
   }
 }
```

```
int main()
{
  int x = 0;
  int y = 50;
  // The loop keeps x between 0 and 100
  __VERIFIER_assume(0 <= x && x <= 100);
  // If x is 50 or less then y is 50
  __VERIFIER_assume(x <= 50 ==> y == 50);
  // If x is greater than 50 then y = x
  __VERIFIER_assume(x >  50 ==> y == x);
  // At the end of the loop x is not <100
  __VERIFIER_assume(x >= 100);
  __VERIFIER_assert(y == 100);
}
```

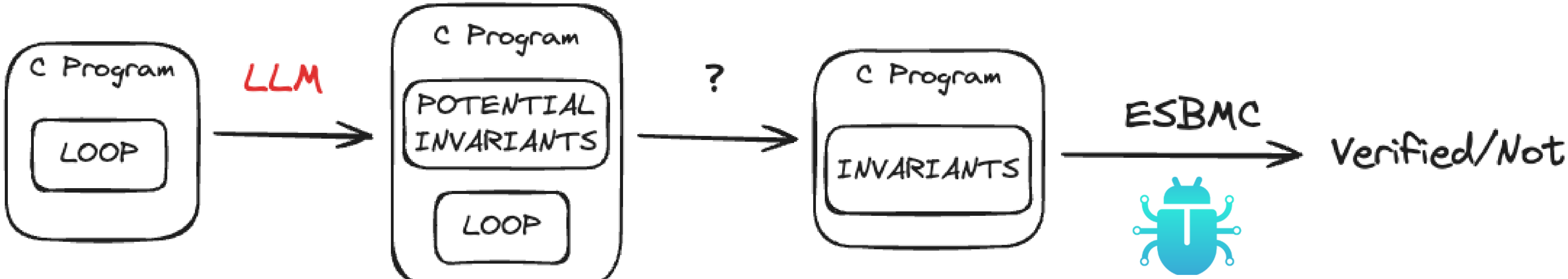
Replace Loop with Loop Invariants



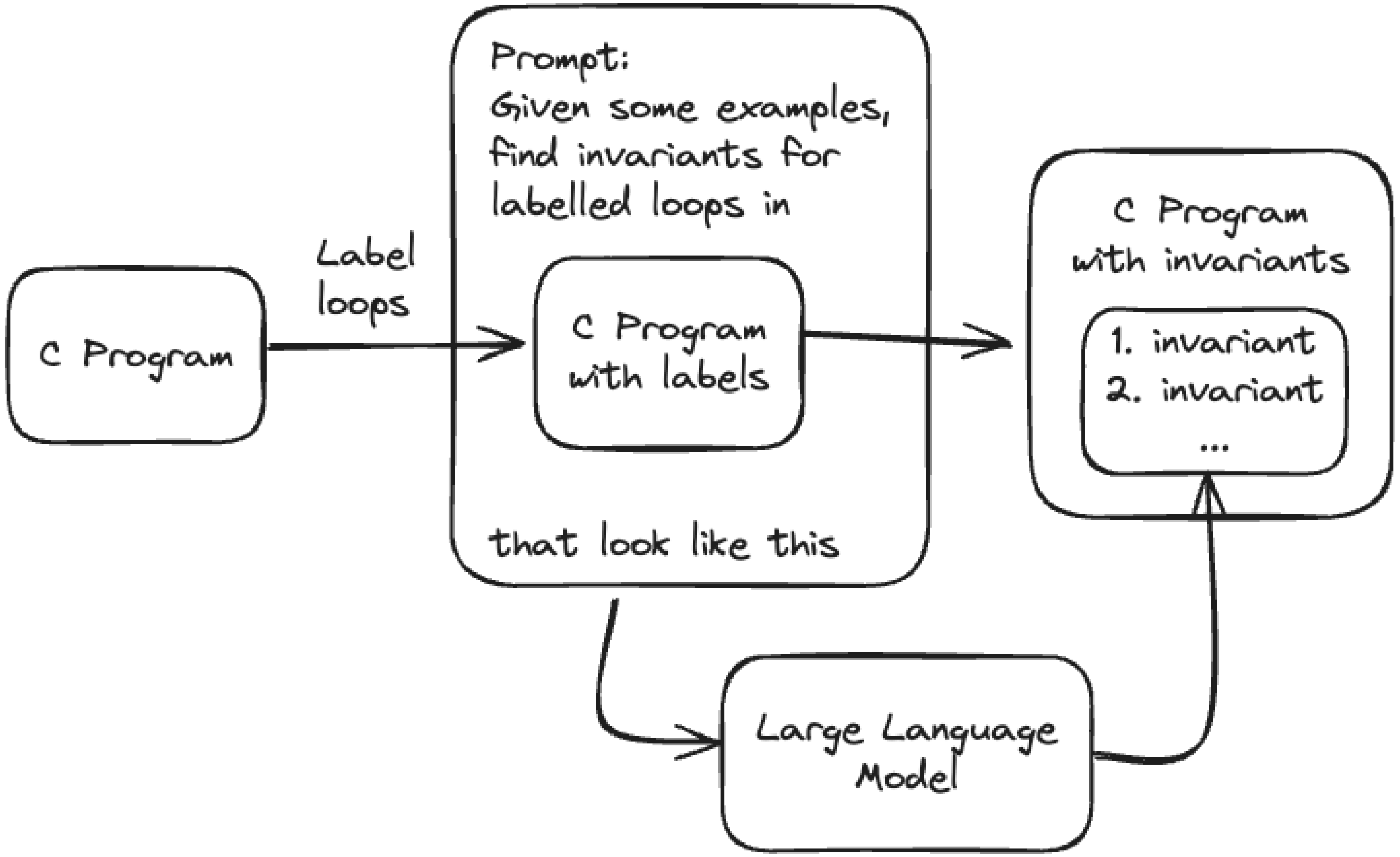
Replace Loop with Loop Invariants



Generating Potential Invariants



Generating Potential Invariants



Prompt Engineering for Invariant Generation

```
<Examples with explanation>

Based on these examples provided above can you
generate  a C invariant for the following code,

<Labelled Input C Program>

Print an invariant for this loop that holds in the
form '__invariant(...);'. They should help prove the
assertions. You can utilise '&&' or '||' if required.
No explanation.  Your answer should be in the form
'__invariant(. . . );'
```

Chain of Thought (CoT) Prompt Engineering

Small number of different prompts that vary by their examples e.g. :

- Single Invariant
- Multiple Invariants

Full Prompts, Constrained Prompts and Combined Prompts

Prompt Engineering for Invariant Generation

```
int main()
{
  int x = 0;
  int y = 50;
  while (x < 100) {
    x = x + 1;
    if(x > 50) {
      y = y + 1;
    }
  }
  __VERIFIER_assert(y == 100);
}
```

```
Provide two loop invariants for the loop in the
program that are strong enough to prove the
assertion at the end of the program.

Answer:
Invariant 1: 0 <= x && x <= 100

This ensures that x is always within the expected
range throughout the loop's execution.

Invariant 2: x <= 50 && y == 50 || x > 50 && y == x

This captures the relationship between x and y:
If x is less than or equal to 50, y remains 50.
If x is greater than 50, y increments in tandem with
x, effectively making y equal to x.
```

LLMs Can Get it Wrong!

```
__invariant(c > 1 && c < 2 ==> i == 0);
```

```
__invariant(...);
```

```
__invariant(
```

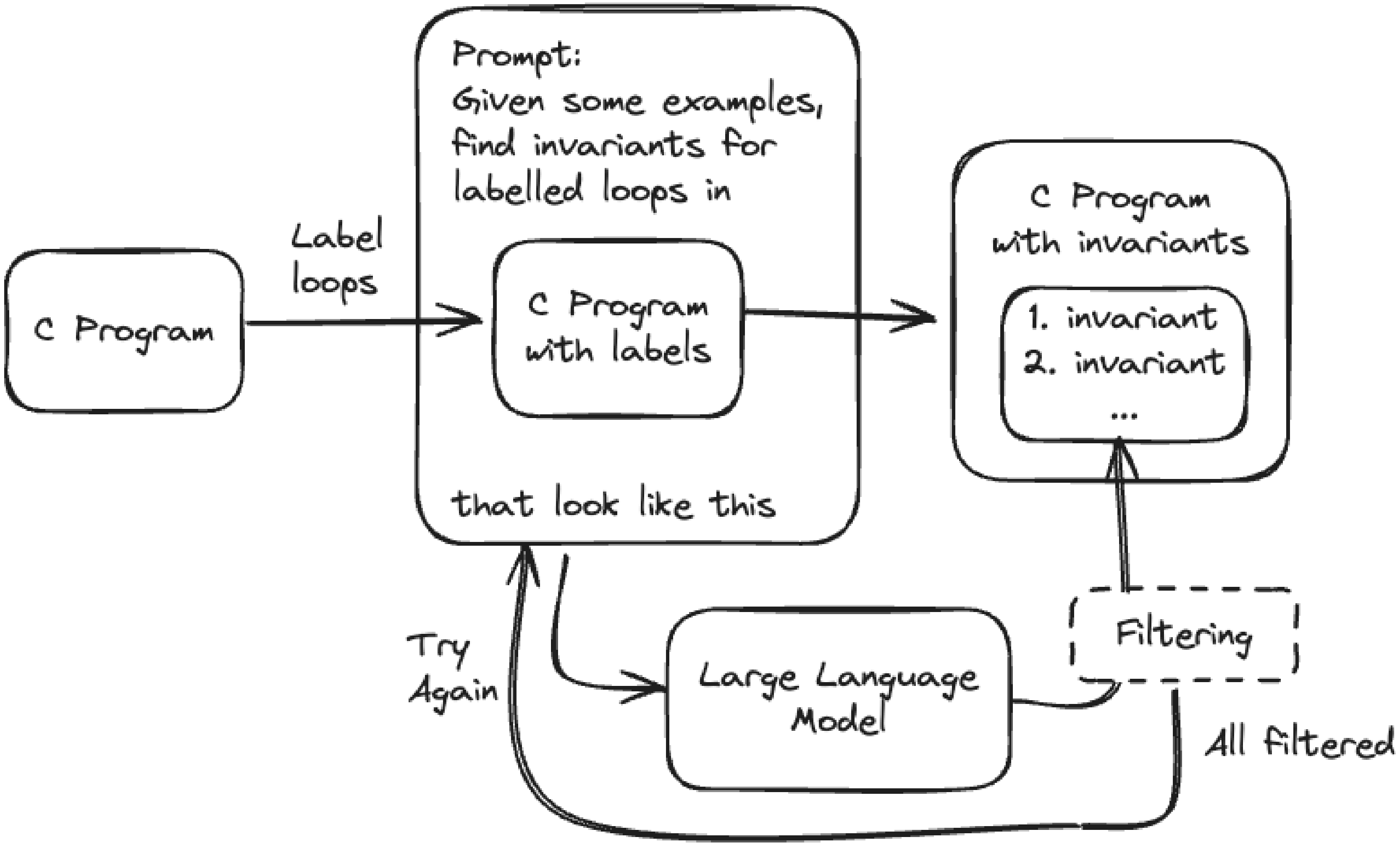
```
This is the invariant
__invariant(...); for
the c program.
```

```
int x;
int y;
__invariant(x > max);
```

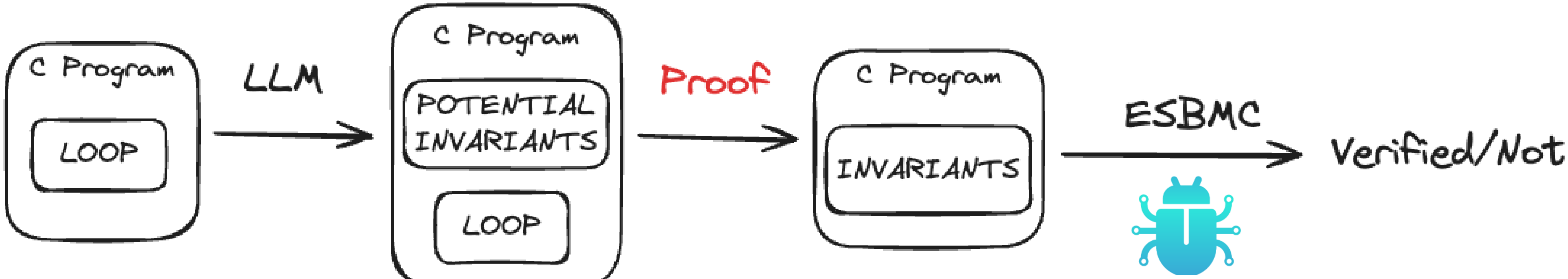
How to get better answers?

1. Constrain the prompt (no explanation as it can be confusing)
2. Filter the answers (simple regex filtering for now)

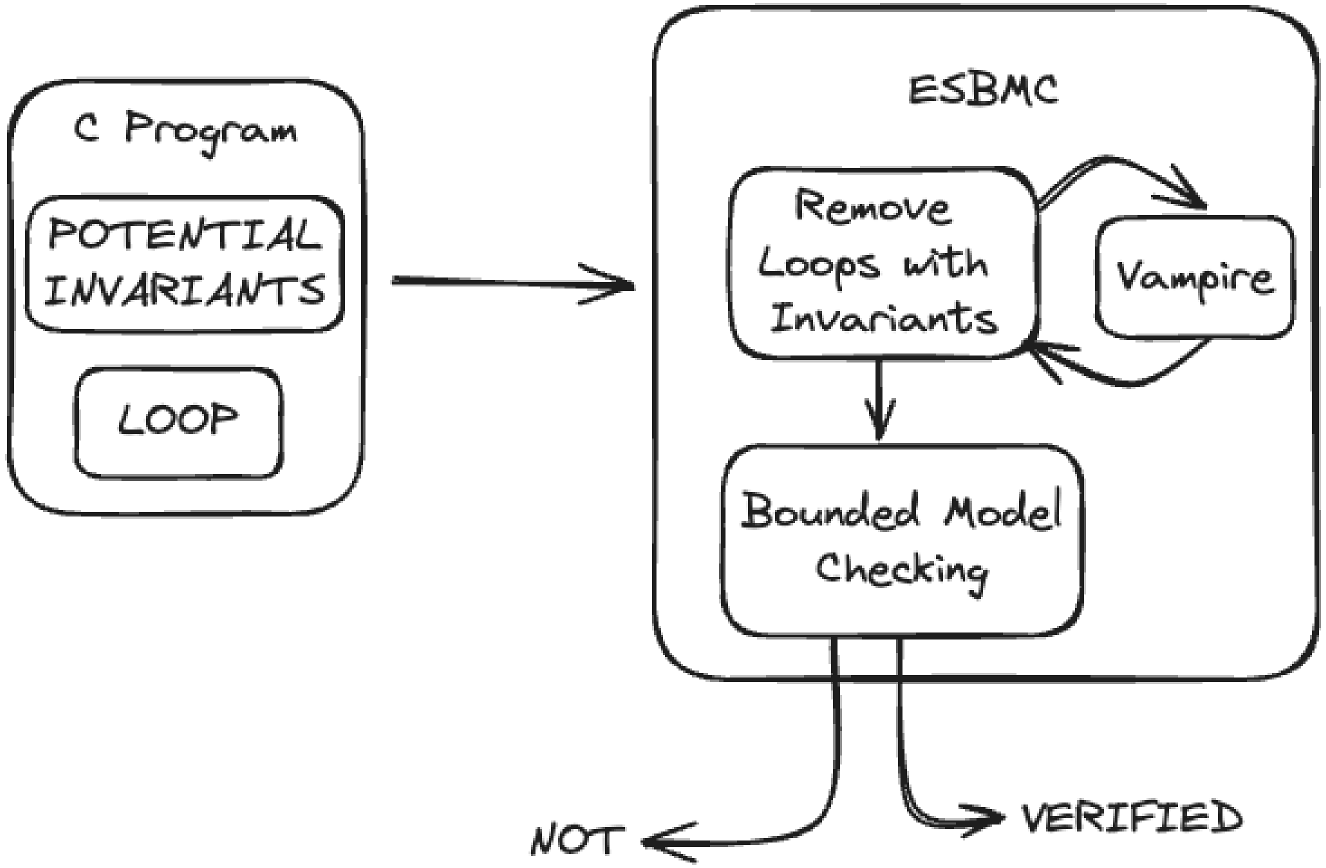
Generating Potential Invariants



Proving Potential Invariants



Proving Potential Invariants



Sound but <u>not</u> Complete

The approach is Sound

Valid means that for any input, if the program terminates (we check partial correctness) then the assertions hold. No invalid assertions can be proven; there are no false positives.

The approach is <u>not</u> Complete:

The LLM may never generate the invariants needed to prove a valid assertion i.e. that sufficiently capture the semantics of the loop.

Incompleteness Example

```
int main() {
  int i = __VERIFIER_nondet_int();
  int j = __VERIFIER_nondet_int();
  int k = __VERIFIER_nondet_int();
  int n = __VERIFIER_nondet_int();
  __ESBMC_assume(k >= 0);
  __ESBMC_assume(n >= 0);
  i = 0;
  j = 0;
  while (i <= n) {
      i  = (i + 1);
      j  = (j + i);
  }
  __VERIFIER_assert( ((i + (j + k)) > (2 * n)) );
}
```

We want these invariants

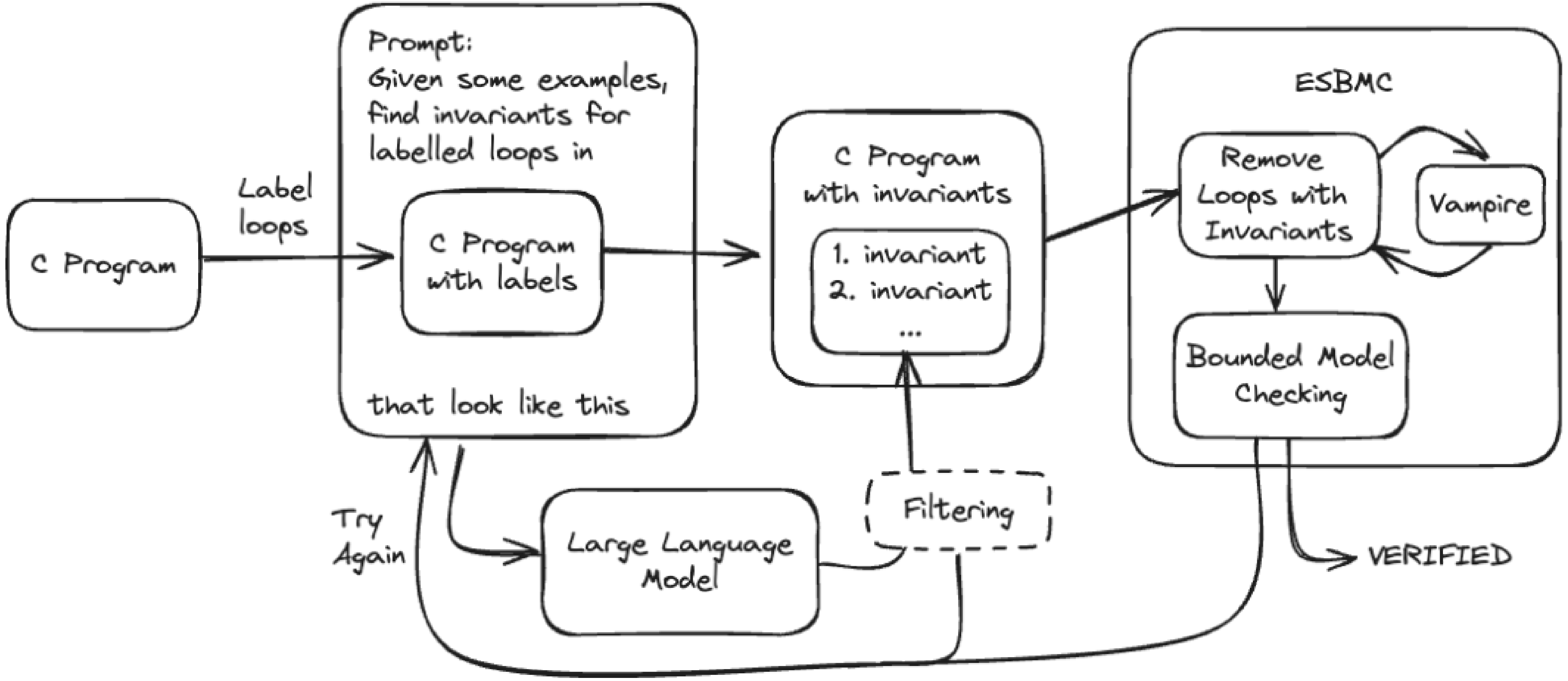
```
__invariant(i >= 0);
__invariant(j >= i);
```

But if we just have the first one we can get a counterexample

```
i = 2, n = 1, j = k
```

Our invariant overapproximated the reachable states

The ESBMC ibmc Tool



MANCHESTER 1824
The University of Manchester

Comparing Prompts and Answer Filtering

No Regex ← no-filter
Regex ← filter

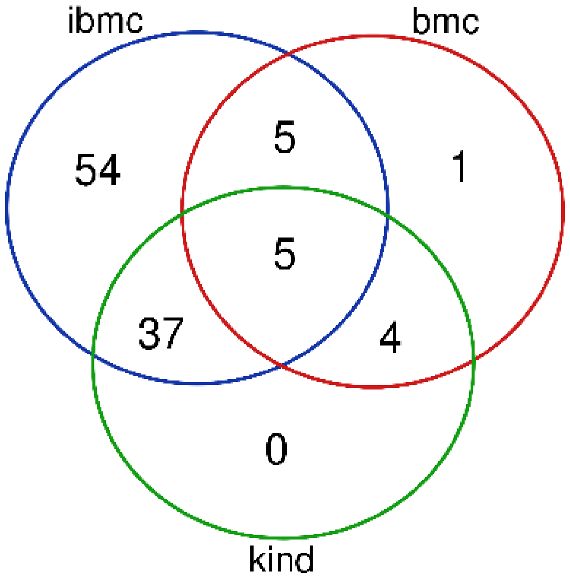
	Solved	Time		Iterations		
		mean	max	mean	max	err
Full Prompt						
no-filter	74 (18)	59.7	316.0	10.8	31	7.8
filter	62 (9)	71.0	273.1	11.8	30	0.2
Constrained Prompt						
no-filter	64 (8)	59.7	231.2	9.3	30	6.3
filter	65 (11)	49.1	352.4	7.8	29	0.1
Combined Prompt						
no-filter	79 (15)	109.1	643.5	17.6	59	13.2
filter	77 (13)	98.0	567.2	14.8	58	0.2

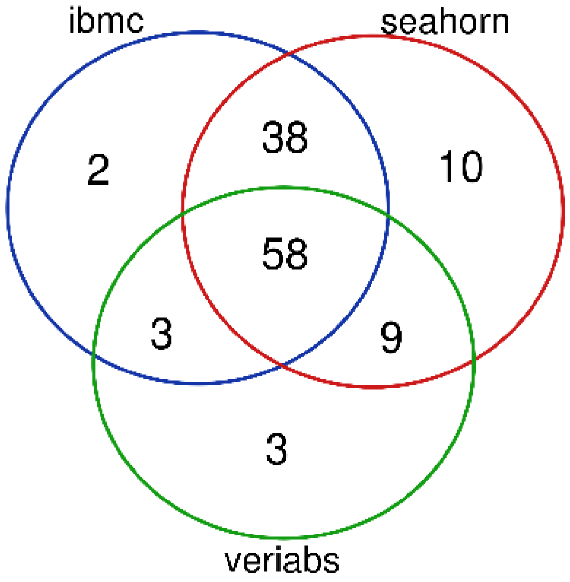
Total Code2inv Benchmarks = 133

Across all options (pipelines) = 101

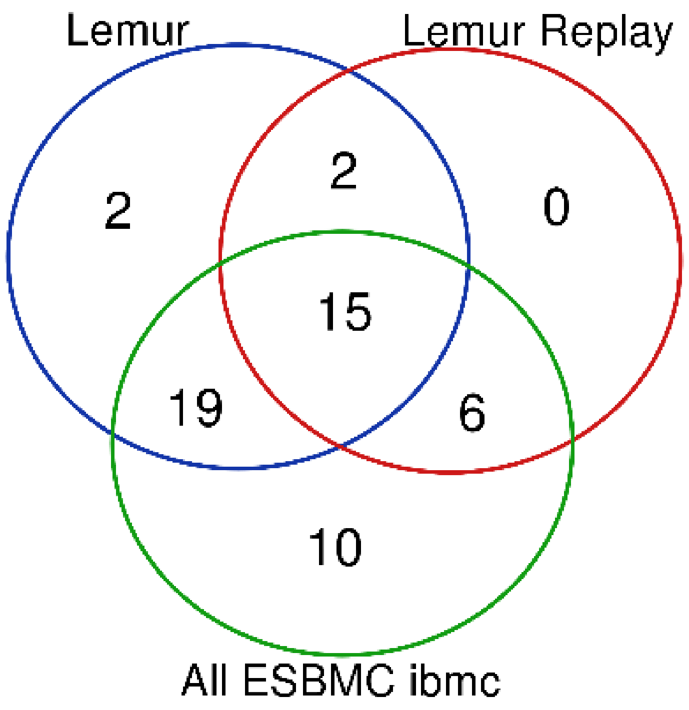
Comparing to Other Verifiers

Tool	Solved	Unique
ESBMC bmc	16	0
ESBMC k-induction	46	0
SeaHorn	115	10
VeriAbs	73	2
ESBMC ibmc	101	2





Comparison to LEMUR



LEMUR is similar as it also uses an LLM to suggest invariants.

We could not run LEMUR so replayed their invariants using our extended ESBMC

Using their invariants we verified 6 more programs

Using our invariants we verified 10 more programs

Potential lessons from both sides

Future Work

Theorem Prover can prove Quantified Invariants – can LLMs generate them?

Some programs need invariants about memory – can we prove them?

Current Prompt Engineering only handles small programs – can it scale?

We tried one LLM and one Prompt Engineering Approach – what else works?

Thank you