MANCHESTER 1824
The University of Manchester

Supporting Software Formal Verification with Large Language Models

The Semantic Gap: From Natural Language to Formal Verification

Weiqi Wang, Marie Farrell, Lucas C. Cordeiro, Liping Zhao
University of Manchester, Manchester, UK

August 28, 2025

The Core Challenge: Bridging Requirements and Formal Verification

Current Status

Formal verification requires specialized expertise that most domain experts lack

Current Reality:

$$\text{Requirements Engineer} \xleftrightarrow{\text{Cooperation Needed}} \text{Verification Expert}$$

$$\text{Natural Language Requirements} \rightarrow \text{Formal Verification Properties}$$

Key Barriers:

- Requires expertise across **multiple specialized tools**
- Manual variable mapping between requirements and implementation
- Limited expressiveness for complex temporal dependencies

Our Goal: Enable direct automation of formal verification for domain experts

SpecVerify Solution

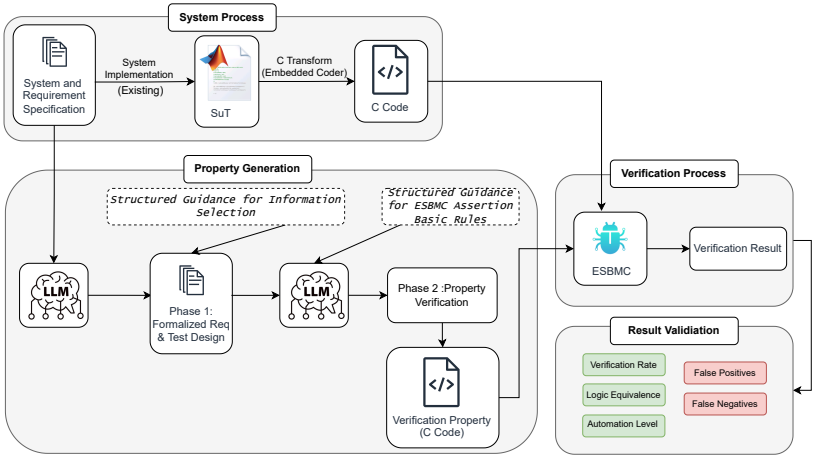

Figure: Two-Phase LLM-Assisted Verification Pipeline

Automated transformation from natural language requirements to C verification code

Why LLMs for Formal Verification? - Capability 1

Research Question: Can LLMs automatically bridge the semantic gap from natural language to formal properties?

Example 1 - Boundary Reasoning:

Input Requirement

"Output shall be bounded by Top and Bottom limits"

LLM Generated Multiple Verification Conditions:

- Boundary violation detection: `output > top_limit`
- Lower bound checking: `output < bottom_limit`
- Edge case handling: `top_limit == bottom_limit`
- Error state transitions

Insight: Single requirement $\rightarrow$ Multiple formal properties automatically

Why LLMs for Formal Verification? - Capability 2

Example 2 - Mathematical Reasoning:

Input Requirement

"Maintain target on port-side of vehicle"

LLM Semantic Understanding:

- "port-side" $\rightarrow$ left side in navigation
- Spatial relationship $\rightarrow$ geometric constraint
- Mathematical formulation: cross product test

Generated Formal Specification

Precondition: `valid_position(vehicle, target)`
Property: `cross_product_z > 0`
Implementation: `(target_x - vehicle_x) * vehicle_vy - (target_y - vehicle_y) * vehicle_vx > 0`

Key Insight: Domain knowledge $\rightarrow$ Mathematical constraints automatically

Challenge 1: Domain-Specific Semantic Interpretation

Case Study: TSM "Miscompare" Terminology

What Happened

Requirements: "Errors will appear as differences... called a miscompare"
LLM understood: Simple arithmetic difference between signals
Reality: Triple-redundancy voting disagreement (majority wins)

Root Issue: Domain-specific terms have specialized meanings

RE Implications:

- Automated tools need explicit domain context
- Requirements should define key terminology
- 3.4% of our cases had similar semantic gaps

Challenge 2: Revealing Hidden Assumptions in Requirements

Case Study: SWIM Airspeed Monitoring System

- **Requirement:** "Monitor airspeed... when vehicle air data impact pressure is less than warning trigger"
- **SpecVerify:** Generated properties from specification based this requirement
- **ESBMC counterexample:** Found violations with negative pressure values

Critical Discovery

CoCoSim team retroactively added physical constraint (pressure > 0) - **the only manual assumption across all cases**

Implications for RE Practice:

- Requirements omit "obvious" physical constraints
- Manual verification unconsciously adds assumptions
- Automated tools expose these hidden gaps systematically

Value: Reveals requirements incompleteness that humans overlook could be hidden

Logical Equivalence Analysis: Hoare Logic Comparison

Challenge: How to verify LLM-generated properties match manual verification?

CoCoSim (Lustre):

- Precondition: `limits & not standby & not apfail & supported`
- Function: State machine logic
- Postcondition: `pullup`

Claude Generated:

- Precondition: `rtU.limits == true, rtU.standby == false...`
- Function: `fsm_12B_global_step()`
- Postcondition: `rtY.pullup == true`

Manual Equivalence Analysis:

- Compare precondition-function-postcondition structure
- Map abstract Lustre variables to C implementation variables
- Verify same logical conditions and outcomes

No automated tool exists - manual analysis required here

Result: 79.31% logical equivalence across 58 requirements

Benchmark Performance

Task Category	CoCoSim	SLDV	Claude +ESBMC	ChatGPT +ESBMC
Signal Processing				
TSM	3/4/4	**4/4/4**	3/4/4	2/4/4
TUI	3/3/5	3/3/5	**4/5/5**	0/0/5
Finite State Control				
FSM†	**13/13/13**	**13/13/13**	**13/13/13**	**13/13/13**
REG	**5/10/10**	0/10/10	**5/10/10**	0/0/10
Navigation				
NLG	0/7/7	0/7/7	**1/7/7**	0/0/7
NN	0/4/4	0/4/4	0/4/4	0/4/4
EB	0/3/5	0/3/5	0/5/5	0/5/5
System Integration				
SWIM	**2/2/2**	1/2/2	1/2/2	0/1/2
EUL	**1/8/8**	0/8/8	0/8/8	0/8/8
Performance Metrics				
Verified/Formed/Total	**27/54/58**	21/54/58	**27/58/58**	15/35/58
Verification Rate (%)	**46.5**	36.2	**46.5**	25.9
False Positives⋆	2	**0**	**0**	8
False Negatives⋆	6	**0**	2	2
Assertion Errors	**0**	**0**	**0**	23
sin/cos approx. error	6	**0**	**0**	**0**

Performance Summary:

- **46.5%** verification rate (matches CoCoSim)
- **28% better** than SLDV
- **Zero** false positives vs CoCoSim's 2
- **Fewer** false negatives (2 vs 6)

Key Advantages:

- Full automation (no manual mapping)
- Beyond LTL expressiveness
- Found 2 new floating-point errors

Remaining challenge: Neural networks & complex matrices

Technical Discovery: Floating-Point Precision Error

SpecVerify Found Critical Error That CoCoSim Missed

Case: TSM Median Selection Algorithm

- Three inputs: $a = 1.813 \times 10^{24}$, $b = 2.328 \times 10^{-10}$, $c = 1.999$
- Expected median: $c = 1.999$ (middle value by comparison)
- Implementation bug: uses mean-based selection instead of direct comparison
- IEEE 754 precision loss: $\mu = \frac{a+b+c}{3} \approx \frac{a}{3}$, selects b as closest
- Both verification approaches tested comparison-based properties but missed implementation precision

Verification Results:

- **CoCoSim & SLDV:** Missed - used rational arithmetic
- **SpecVerify + ESBMC:** Found - IEEE 754 floating-point semantics
- **Confirmation:** Generated test case reproduces the bug

Key insight: Implementation-level verification reveals errors hidden by mathematical abstractions

Manual Verification Step and Abstraction Introduces Errors

Case 1: REG-003 Simulink Wiring Error

```
if (input > 50.0)
  then 0      // should be +1 (counter increment)
  else 0;     // always constant 0
```

Visual connection mistake: counter connected to wrong output

Case 2: Trigonometric Lookup Table Inconsistencies

- Manual cosine/sine lookup tables used inconsistent π constants
- Example: $\cos(617663/131072) = 1/2$ vs $-1/2$ for same input (expected $\cos(3/2\pi) = 0$)
- All 6 trigonometric errors caused by this approximation inconsistency

Two Lessons:

- Manual steps introduce human errors (wiring mistakes)
- Direct automated translation minimizes abstraction risks

Research Achievements

Technical Contributions:

- End-to-end automation: requirements $\rightarrow$ C verification code
- Found floating-point precision errors missed by model-level tools
- Reduced manual modeling errors (wiring, approximations)

Key RE Insights:

- Domain terminology needs explicit clarification
- Formal verification reveals hidden assumptions
- 79.31% logical equivalence shows LLM potential

Impact: LLMs can bridge requirements-verification gap for domain experts

Future Directions & Conclusion

My Research Roadmap:

- Interactive refinement: Counter-example guided specification improvement
- Scalable verification: Program slicing for complex systems and introduction to verify with loop invariants
- Requirements quality feedback: Automated completeness checking

Thoughts for RE Community:

- Formal verification is now accessible - LLMs lower the expertise barrier
- High-quality requirements become critical enabler for automation
- Requirements maintenance gains new importance in verification workflows

Vision: Requirements engineering drives practical formal verification adoption

Thank You - Questions & Discussion

Contact: Weiqi.Wang-2@postgrad.manchester.ac.uk