ASE 2026 · MUNICH

NL2VBench: A Benchmark for End-to-End Natural-Language to Verifiable Rust Generation

XIDIAN UNIVERSITY
Xi'an, China

Lianhe Hu[1], Cheng Wen[1], Dugang Liu[2], Zhiwu Xu[2], Zhuohua Li[3], Jie Su[1], Bin Yu[1], Shengchao Qin[1]

[1] Xidian University [2] Shenzhen University [3] The Chinese University of Hong Kong

PAPER DETAILS

Introduction

1. Introduction

Large language models can generate conventional code, but formally verifiable programs demand more: executable logic must be accompanied by specifications, loop invariants, and proof-oriented annotations accepted by a verifier.

Existing resources either start from a formal specification, a partial program, or a proof obligation. They therefore do not measure the full natural-language-to-verified-program setting.

NL2VBench fills this gap with a reusable benchmark for generating complete Verus programs from natural-language task descriptions and task-defining interfaces.

Why strict review? A program may pass verification under a weak or task-misaligned specification. We therefore report Strict Verified Correct (SVC), not verifier pass rate alone.

Benchmark scope. Each task requires a complete Verus artifact: executable Rust code, a functional specification, and —where needed—loop invariants or proof annotations.

Task Format: from Natural Language to Verified Rust

2. Benchmark Task Format

Input Task Description

Natural-language Task. Implement a function that, given a sorted array of 32-bit integers in non-decreasing order and a target integer, returns the smallest index at which the target could be inserted while preserving the array's order. Assume the array may be empty. The returned index must be between 0 and the length of the array, inclusive. Additionally, every element strictly before the returned index must be less than the target, and every element at or after the returned index must be greater than or equal to the target.

Interface Information.

```
fn binary_search(a: &Vec<i32>, key: i32) -> (n: usize)
```

Task input: intent and interface

Input and target output.
Each task exposes a human-revised natural-language description and a task-defining Rust/Verus function interface; its verifier-checked reference solution remains hidden. The model must synthesize a complete Verus artifact comprising executable Rust, a functional specification, and, for loop-based tasks, inductive invariants or proof annotations. Verifier acceptance is necessary, while Strict Verified Correct additionally requires manual confirmation that both the implementation and specification are consistent with the intended task.

```
use vstd::prelude::*;

verus! {
fn binary_search(a: &Vec<i32>, key: i32) -> (n: usize)
    requires                                          Contract
        forall|i: int, j: int| 0 <= i < j < a.len() ==> a[i] <= a[j],
    ensures
        0 <= n && n <= a.len(),
        forall|i: int| 0 <= i && i < n ==> a[i] < key,
        forall|i: int| n <= i && i < a.len() ==> key <= a[i],
{
    let mut lo: usize = 0;
    let mut hi: usize = a.len();

    while lo < hi
        invariant                                     Proof
            0 <= lo && lo <= hi && hi <= a.len(),
            forall|i: int| 0 <= i && i < lo ==> a[i] < key,
            forall|i: int| hi <= i && i < a.len() ==> key <= a[i],
            forall|i: int, j: int| 0 <= i < j < a.len() ==> a[i] <= a[j],
        decreases hi - lo
    {
        let mid = lo + (hi - lo) / 2;

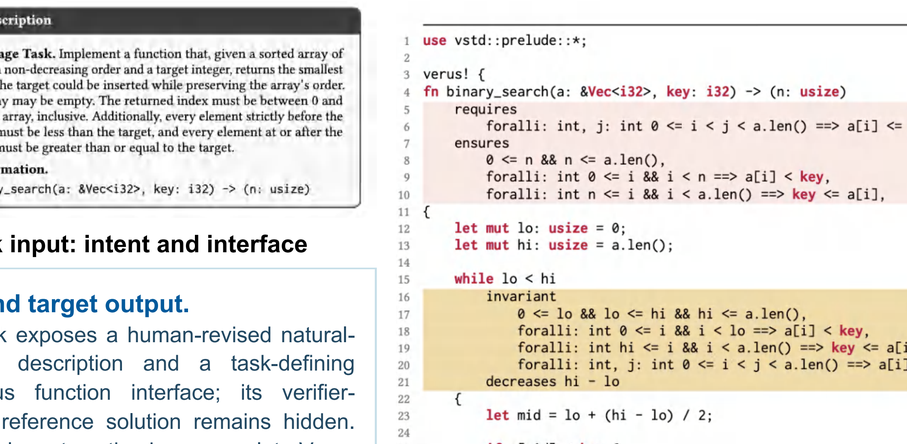
        if a[mid] < key {
            lo = mid + 1;
        } else {
            hi = mid;
        }
    }
    lo
}

fn main() {
    ...
    assert( binary_search(...) == ... )
}}
```

Reference artifact: verified Verus solution

Construction Pipeline

3. Dataset Construction Pipeline



Five public sources are filtered for validity and non-triviality, deduplicated, canonicalized, converted to natural-language tasks, and finalized by human curation. The final dataset contains 762 instances.

LLM assistance is used only to draft task descriptions from curated artifacts. Two authors manually revise descriptions, remove ambiguity, and verify consistency between the natural language, interface, and reference solution.

Filtering removes malformed or syntactically incomplete programs, missing or severely incomplete specifications, trivial examples, and duplicate or near-duplicate tasks. Canonicalization preserves the task-defining interface and verifier-checked solution while normalizing the artifact format.

Human curation checks problem clarity and correctness, verifies X/Y consistency between task input and code-specification pairs, and finalizes benchmark instances. Only human-revised descriptions are included in the release.

Release quality. The benchmark retains only human-revised descriptions whose stated intent, function interface, and verifier-checked solution have been checked for consistency.

Positioning against Existing Benchmarks

4. Positioning among Existing Benchmarks

Table 1: Comparison of NL2VBench with existing benchmarks for formally verified code generation.

Dataset / Resource	Source Work	Input	Target	Verification
VerusBench	AutoVerus [17]	Verus code + spec. (proofs omitted)	Proofs	Verus
HumanEval-Verus / MBPP-verified	AlphaVerus [1]	Verus task/spec	Verified Verus code	Verus
DAFNY2VERUS-COLLECTION	AlphaVerus [1]	Dafny	Verus code	Verus
VeriCoding	VeriCoding [2]	Formal specification + partial program	Verified code	Dafny/Verus/Lean
NL2VBench	**This work**	**Natural language + interface**	**Verified Verus code**	**Verus**

NL2VBench is distinct because the input is **natural language + interface** and the target is a **complete verified Verus program**, rather than only a proof, repair, or translation from another formal language.

Dataset Characteristics

5. Dataset Characteristics

157 Simple tasks
598 Loops tasks
7 Complex tasks
558 quantified specifications
660 rich data structures
88 recursive tasks

The benchmark is verification-intensive in both specification and implementation. Most tasks involve arrays, sequences, or complex data structures; 598 tasks require loop invariants.

Simple tasks mainly require interface-level pre/postconditions. Loops tasks contain one or more loops and require invariants to establish correctness. Complex tasks involve richer structures such as recursion, nested loops, or sophisticated data manipulation.

The benchmark is intentionally diverse rather than balanced: it captures the common challenge of synthesizing executable logic together with the formal artifacts needed for successful verification.

Evaluation and Main Results

6. Evaluation Protocol and Main Results

We evaluate five representative LLMs under four prompting strategies: zero-shot, few-shot, chain-of-thought, and least-to-most. Each task receives five generated candidates; Pass@5 credits success if any candidate verifies.

Pass@1 uses a fixed single candidate from the same batch. This makes Pass@1 and Pass@5 comparable under the same model, prompt, and decoding configuration.

Table 3: Baseline results on NL2VBench using direct NL-to-Verus generation.

Model	Best Pass@1 / Pass@5	Best SVC@1 / SVC@5
deepseek-v3.2	5.25% / 13.52%	4.59% / 11.81%
gpt-5.2	29.92% / 50.00%	27.82% / 48.69%
claude-sonnet-4.5	6.96% / 17.59%	6.17% / 15.88%
gpt-4o-mini	2.76% / 4.20%	2.23% / 3.81%
o4-mini	3.81% / 8.79%	3.67% / 8.01%

Table 4: Best baseline results by structural category.

Category	#Tasks	Best Pass@5	Best SVC@5
Simple	157	70.70%	70.06%
Loops	598	44.65%	43.31%
Complex	7	42.86%	28.57%

Best direct NL-to-Verus results. Pass reports verifier acceptance; SVC additionally requires manual expert approval that the implementation and specification match the task intent.

GPT-5.2 performs best with 50.00% Pass@5 and 48.69% SVC@5, yet still solves only about half of the benchmark after five attempts. The difficult Loops subset reaches 44.65% Pass@5 and 43.31% SVC@5.

Verifier acceptance alone cannot fully capture semantic correctness: a program may verify under a weak or task-misaligned specification. In the direct NL-to-Verus setting, two authors with Verus experience independently reviewed 3,981 verifier-passed outputs, obtaining 93.4% raw agreement before disagreements were resolved.

The improvement from Pass@1 to Pass@5 shows that repeated attempts can help, but no evaluated model solves the benchmark reliably. These results demonstrate both the utility of the benchmark and the remaining difficulty of end-to-end verified code generation.

Research opportunity. The large gap between reliable verification and current model performance makes NL2VBench suitable for evaluating prompting, agentic repair, retrieval, and proof-synthesis methods.

Conclusion. NL2VBench provides a challenging and reusable benchmark for end-to-end natural-language to verified Rust generation. The best evaluated model reaches only 48.69% SVC@5, motivating stronger models, prompting strategies, and verification-aware agents.

DOI: 10.1145/3832783.3834619