Beyond Benchmarks: A Case Study of LLM-Generated Verus Specification Failures on Asterinas Vostd

Dugang Liu[1], Ding Wang[1], Chao Jiang[1], Cheng Wen[1], Zhiwu Xu[1], Zhong Ming[1],
Yuwei Liu[2], Xinyi Wan[2], Lin Huang[2]

[1]College of Computer Science and Software Engineering, Shenzhen University, China
[2]Ant Research, Ant Group, China





深圳大学
SHENZHEN UNIVERSITY



蚂蚁集团
ANT GROUP

Background

Formal Verification

- Formal verification proves that program behavior satisfies a formal specification, giving strong correctness guarantees for safety-critical systems such as OS kernels.



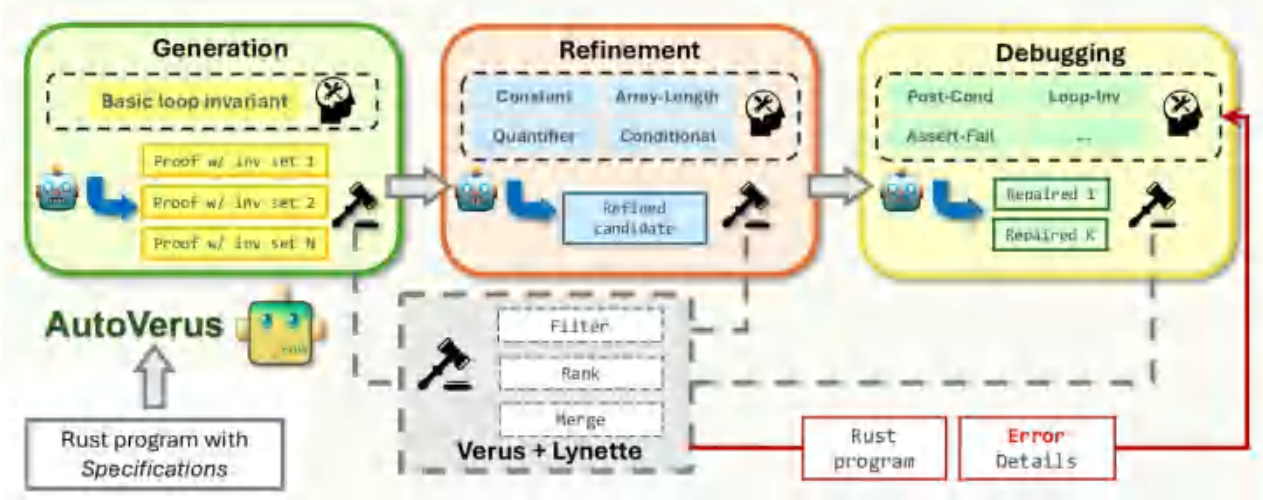
OS Kernel



Aerospace systems

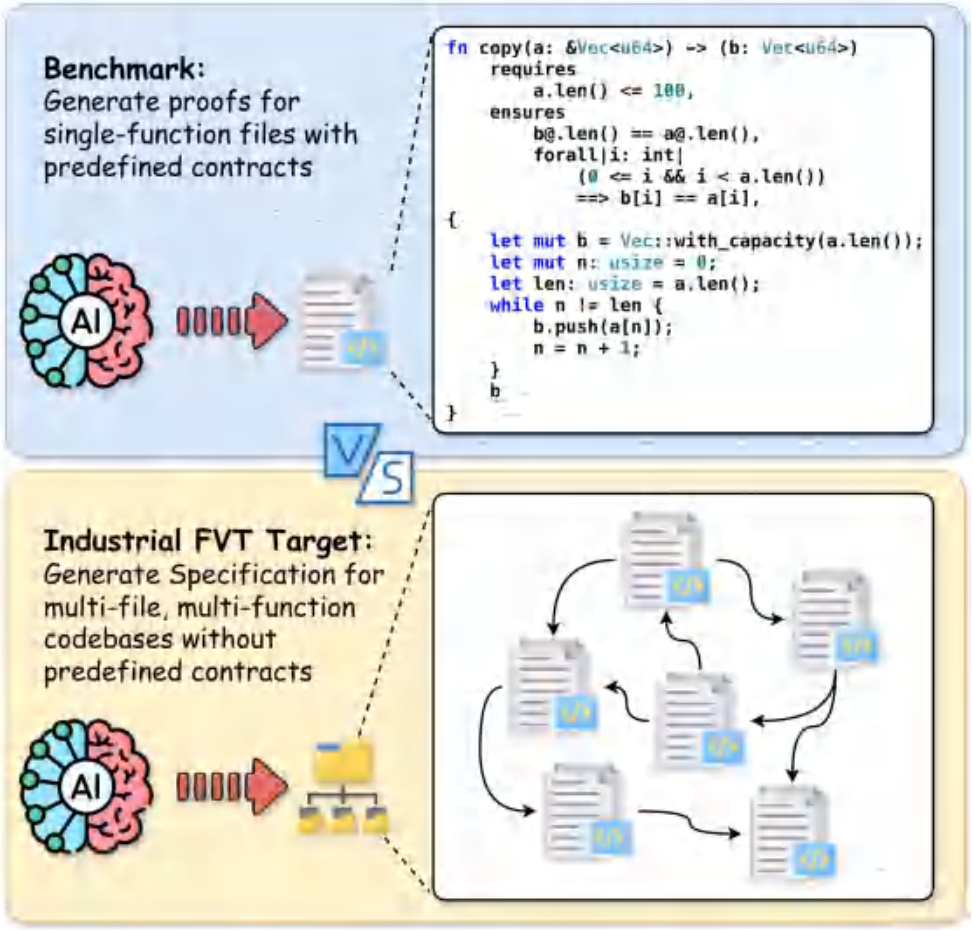
LLM Spec Generation

- Rust ensures memory safety, but not functional correctness; **Verus** adds deductive verification for Rust.
- LLMs show promise in generating specifications and proofs, but evidence mainly comes from **curated benchmarks**.



LLM-Based Specification Synthesis for Rust

From Benchmarks to Industrial FVT

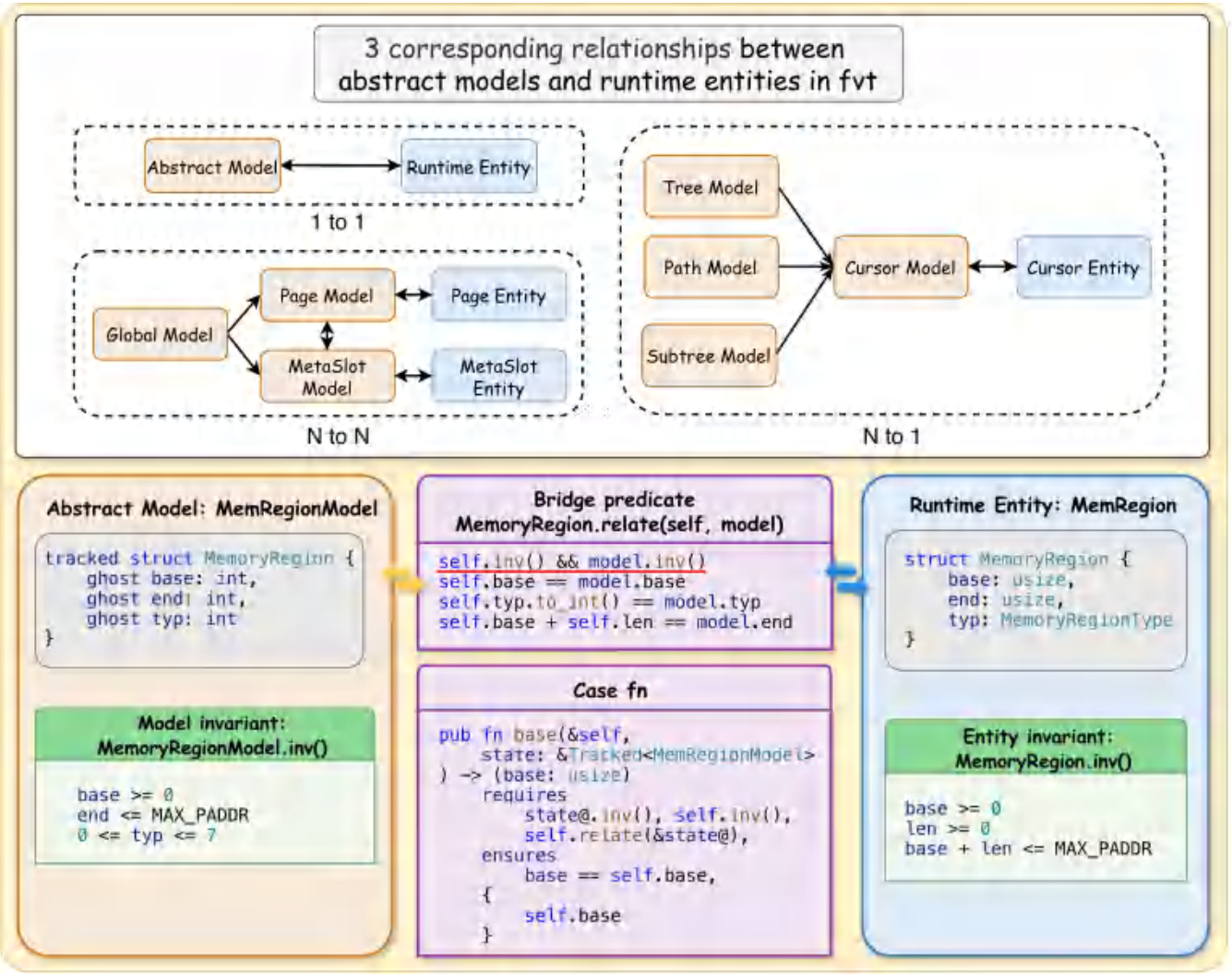


Self-contained Benchmark Tasks vs. In-Repository Industrial Targets.

- **Benchmarks:** isolated functions or stand-alone files.
- **Industrial FVT:** specs must compile inside the original codebase, respect cross-file dependencies and callee contracts, and preserve project-specific proof abstractions.

Does benchmark success transfer to industrial verification?

Entity – Model Proof Style in Vostd



Entity – Model proof style followed by FVT.

- Runtime entities are verified through **abstract ghost models**.
- **Bridge predicates** such as relate(self, model) link the two: 1-to-1, N-to-N, or N-to-1.

A spec can compile and be Verus-legal, yet encode the wrong Entity – Model relation.

Research Questions

Not only **whether** specs verify, but **where they break**, **whether diagnostics point to the real cause**, and **whether repair makes progress**.

RQ1 Benchmark-to-Deployment Gap

How well do LLM-generated specs verify on industrial targets compared with benchmarks?

RQ2 Stage-Based Failures

Where do failures occur across verification stages, models, and targets?

RQ3 Diagnostics & Repair

Do verifier diagnostics and verifier-guided repair lead to the real fix?

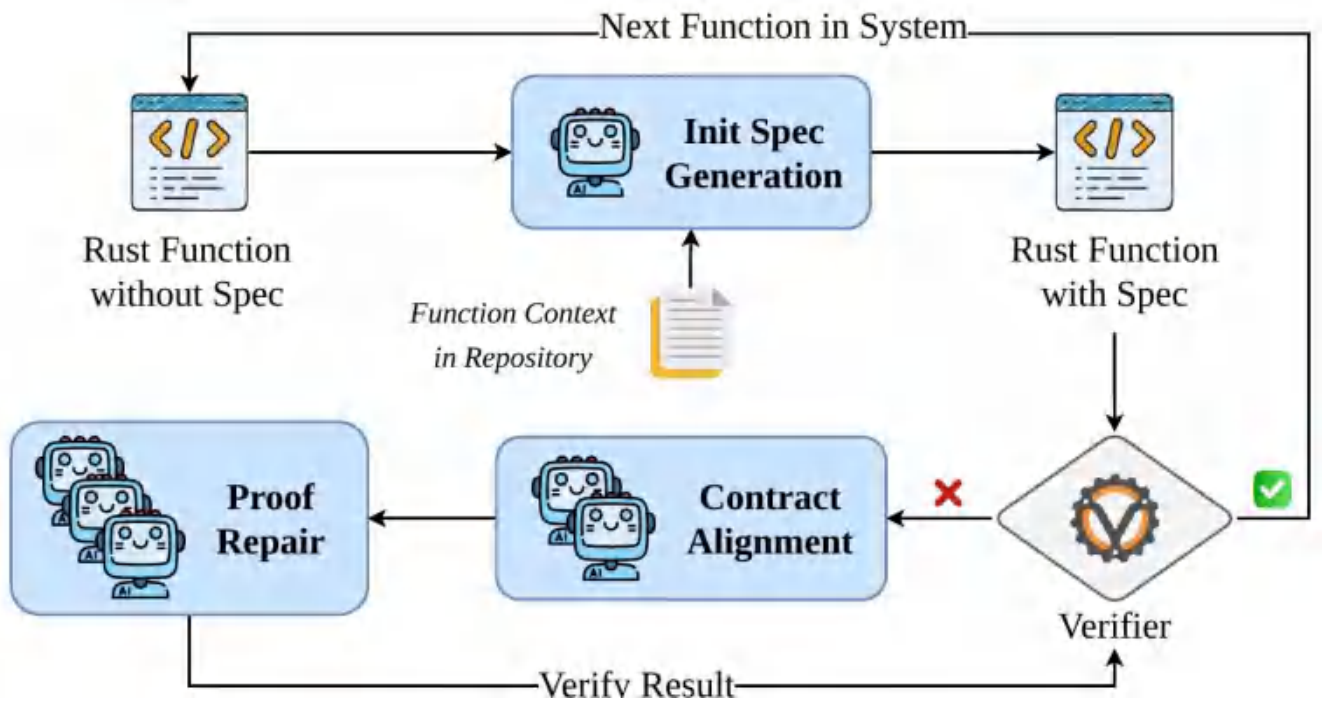
Study Setup

6
FVT targets in Asterinas Vostd

156
function-level tasks

7,149
lines of Rust code

3
LLMs



Repository-grounded generation-and-repair workflow.

- **Vostd:** an industrial OS-kernel verification project in Rust + Verus.
- Specs are generated and checked **inside the original repo**, in call-graph order.
- On failure: contract alignment → proof repair, $\leqslant$ 10 rounds.
- **LLMs:** DeepSeek-V4-Flash, GPT-4o, Qwen-Coder.

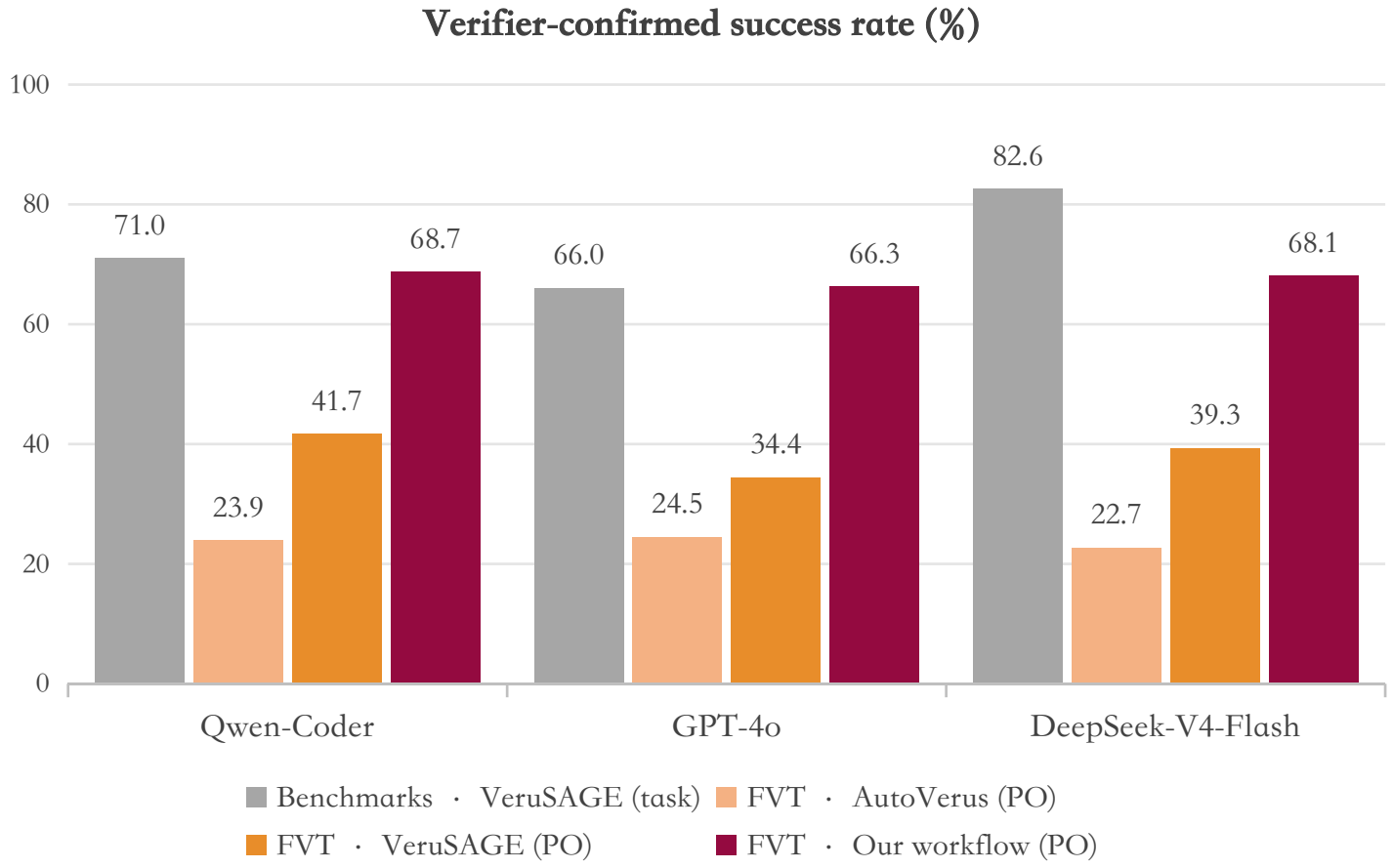
Four-Stage Failure Taxonomy

1,587 manually reviewed failure episodes from **270** repair traces

I	**Repository integration**	Does not compile in the repo: syntax errors, wrong imports, hallucinated APIs
II	**Verus legality**	Valid Rust, illegal Verus: exec/spec/proof modes, tracked/view, old() misuse
III	**Semantic modeling**	Legal Verus, wrong meaning: wrong Entity – Model relation, arguments, updates
IV	**Proof construction**	Right property, unusable proof: missing invariants, lemmas, triggers

Two labels per episode: visible diagnostic stage (what Verus reports) vs. **validated source stage** (the real root cause). A mismatch = **diagnostic misalignment**.

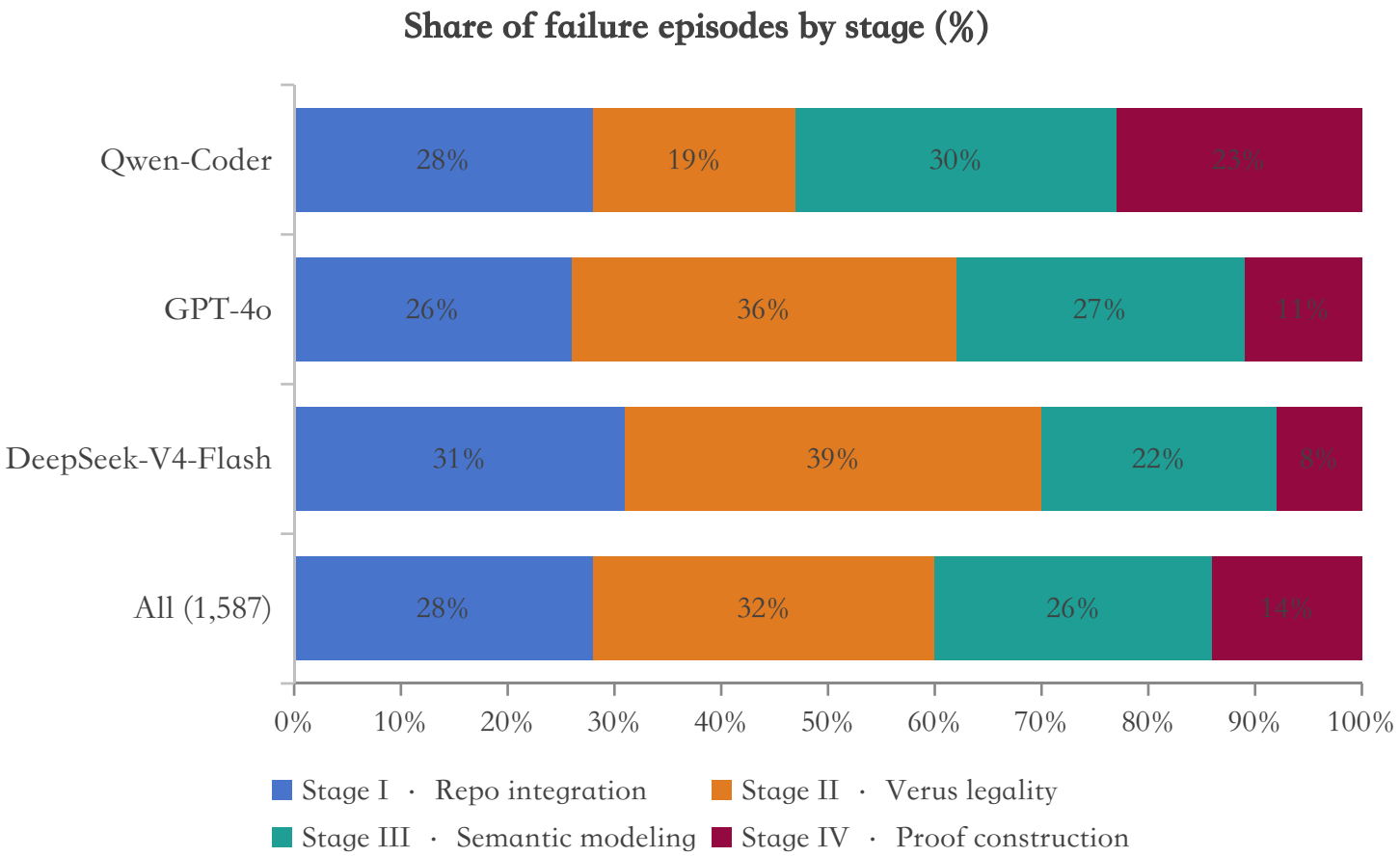
RQ1: Benchmark-to-Deployment Gap



- **Sharp drop:** 66 – 83% on benchmarks → 23 – 42% on FVT for existing methods.
- **Context helps:** repository-grounded context recovers 66 – 69%.
- **Targets dominate:** 100% vs. 44% across targets for the same model.

Answer to RQ1: benchmark pass rates are a weak proxy. Industrial FVT exposes codebase-integration constraints and target-level variation that benchmarks hide.

RQ2: Where Do Specs Fail?



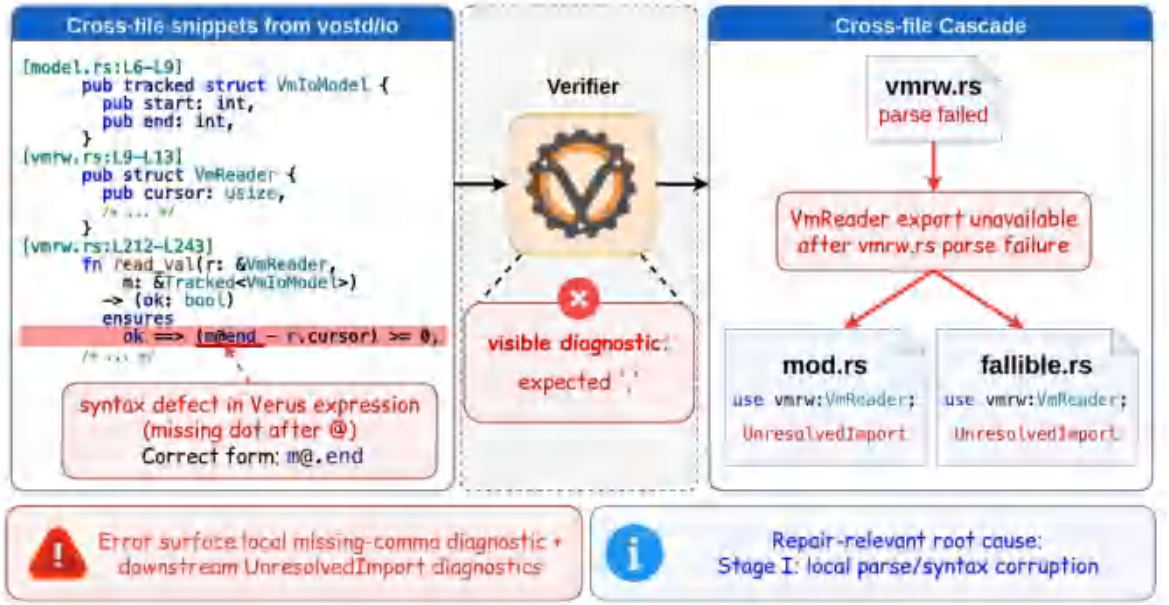
32%

Stage II (Verus legality) is the largest category

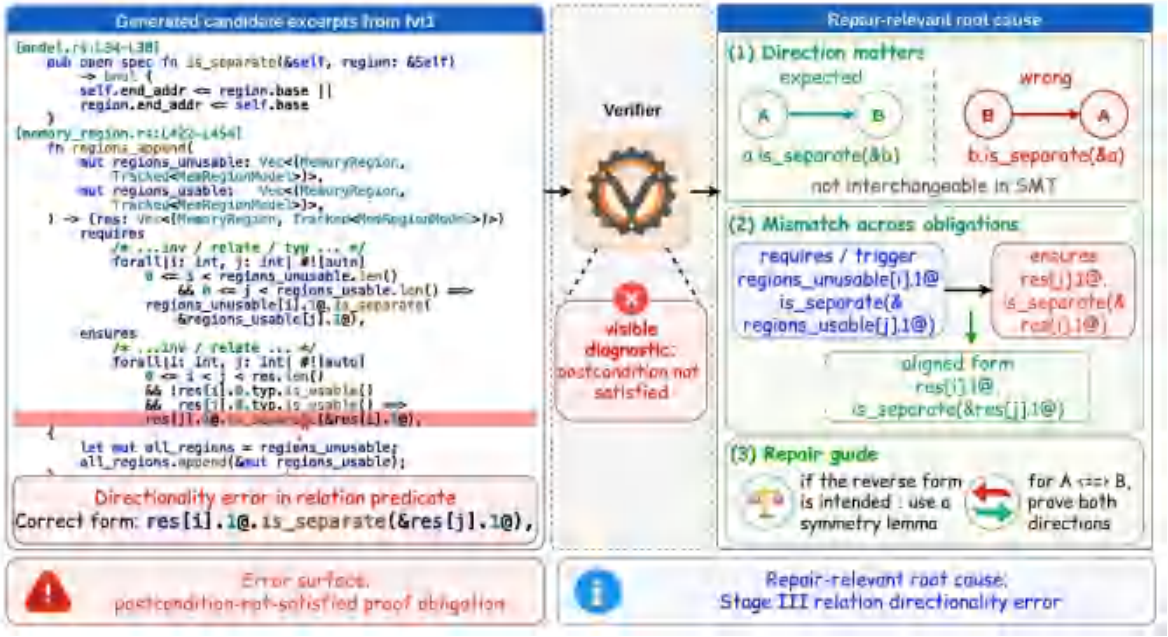
- **Qwen-Coder** fails late: Stage III – IV = **54%**
- **GPT-4o / DeepSeek** fail early: Stage I – II = **62% / 69%**

Answer to RQ2: failures are systematic and stage-dependent, and each LLM has its own failure profile.

RQ2: Representative Failures



Stage I: cross-file parse-error cascade



Stage III: relation directionality error

One missing dot (**`m@end`** vs. **`m@.end`**) breaks parsing of vmrw.rs and triggers UnresolvedImport in two downstream files.

`c.is_separate(a)` instead of **`a.is_separate(c)`**: legal Verus, but it surfaces as a Stage IV postcondition failure.

RQ3: Do Diagnostics Point to the Root Cause?

Qwen-Coder aligned 66.5%

V \ S	I	II	III	IV
I	**127**	9	33	0
II	7	**83**	**50**	0
III	0	0	**49**	49
IV	0	0	15	**65**

GPT-4o aligned 76.0%

V \ S	I	II	III	IV
I	**147**	17	31	0
II	5	**180**	27	0
III	0	12	**91**	**34**
IV	0	5	11	**32**

DeepSeek-V4-Flash aligned 74.4%

V \ S	I	II	III	IV
I	**130**	10	17	0
II	25	**178**	**48**	0
III	1	0	**43**	16
IV	0	8	5	**27**

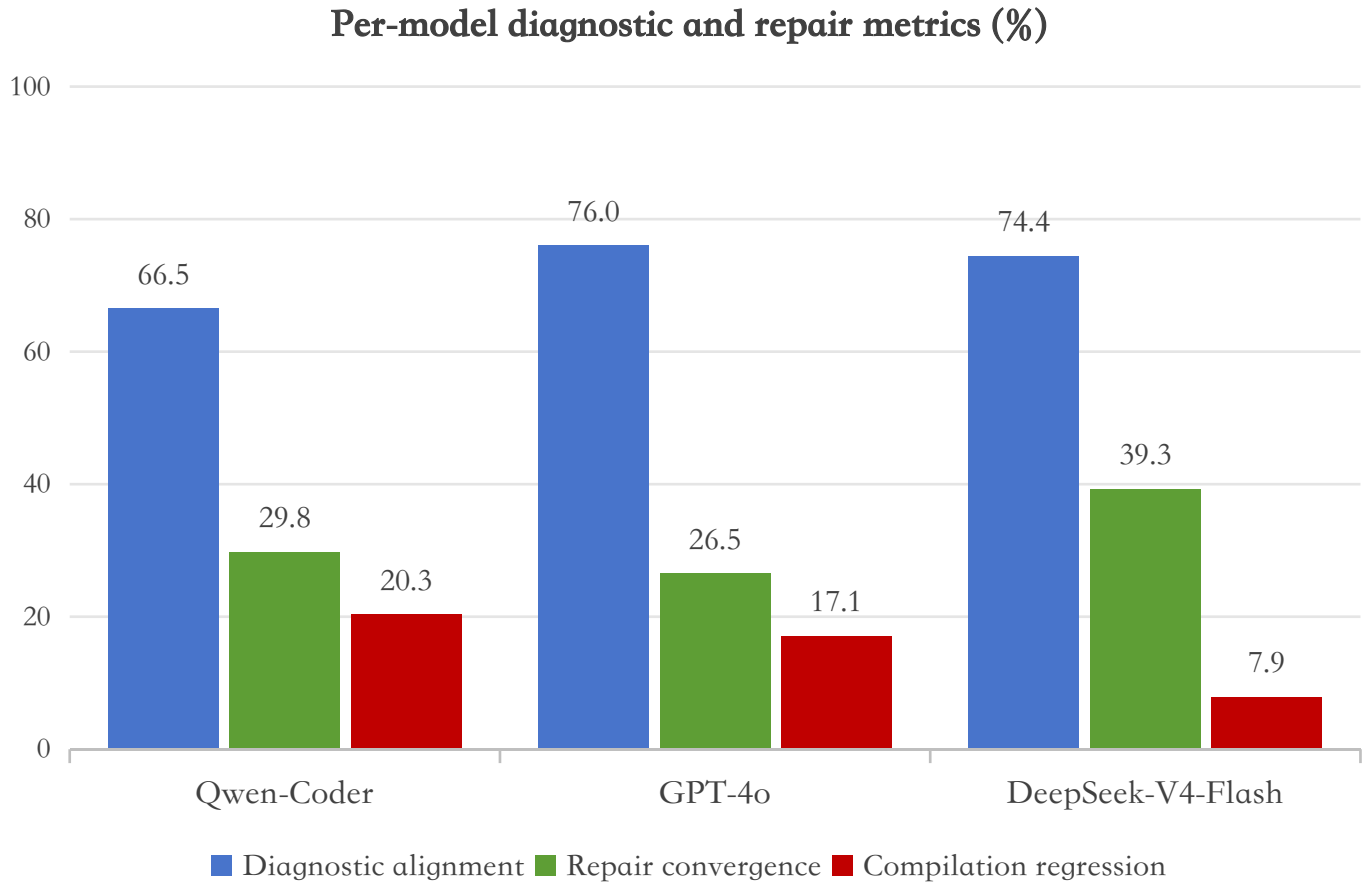
Rows: visible diagnostic stage (V). Columns: validated source stage (S). Gray: agreement; pink: largest off-diagonal drift.

27.4%

of 1,587 episodes are misaligned

- Diagnostics are useful **error surfaces**, but imperfect **root-cause labels**.
- Typical drift: the real defect sits **one stage earlier** than reported (II → III, III → IV).
- A repair loop that follows the latest diagnostic is routed to the **wrong failure layer**.

RQ3: Does Repair Make Progress?



31.5%

of 270 repair traces converge

15.1%

regress to compile failure

- **Alignment ≠ convergence:** GPT-4o aligns best (76.0%) but converges least (26.5%).
- **Target-dependent:** 100% vs. 0% convergence across targets for Qwen-Coder.

Answer to RQ3: verifier-guided repair is not reliably convergent; effective repair needs source-stage attribution, not just verifier messages.

Takeaways

数据智能决策实验室

深圳大学 SHENZHEN UNIVERSITY

1 Evaluation

Report where specs fail, not only pass rates.

Similar pass rates can hide very different stage-wise failures.

2 Tool Design

Attribute the failure stage before repair.

Stage classifiers, stage-conditioned repair, and progress monitors.

3 Industrial Practice

Keep engineers in the repair loop.

Use LLMs as spec drafters; escalate traces that regress or oscillate.

Verifier success rates are a weak proxy for industrial readiness. Artifact: doi.org/10.5281/zenodo.21375096

Thank You !

Beyond Benchmarks: A Case Study of LLM-Generated Verus Specification Failures on Asterinas Vostd

Presented by