Automated Vulnerability Injection in Smart Contracts Using Large Language Models

Nuno Laranjeiro
cnl@dei.uc.pt

University of Coimbra · Portugal

IFIP WG 10.4 · 89th Meeting · 6 May 2026 · Kaunas, Lithuania

MOTIVATION

Smart contract datasets are limited

- **Smart contracts are immutable.** Any vulnerability in the source code can be exploited, often with significant financial impact.
- Evaluating detection tools requires datasets with known vulnerabilities. Most datasets are small, manually curated, and narrow in coverage.
- The gap is severe for AI-based detectors, which need labeled data at scale.

94

vulnerability types
in OpenSCV

7

types in SolidiFI,
popular injection dataset

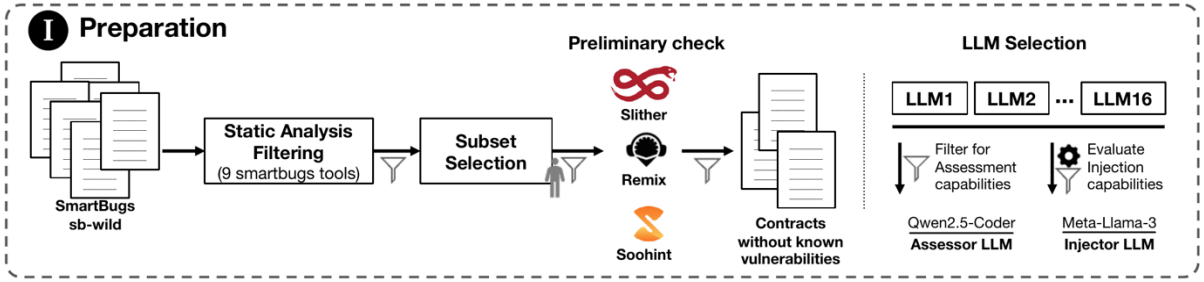
WHAT WE ASK

Three research questions

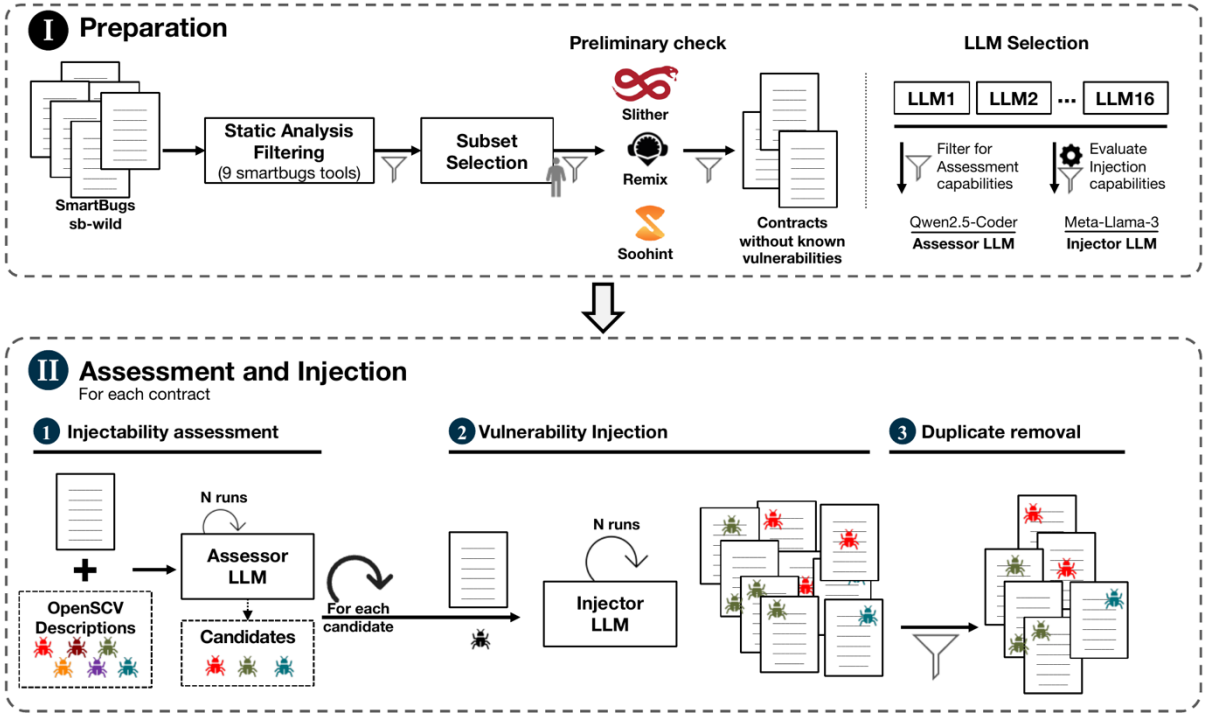
RQ1 Assessment	Can LLMs reliably tell which vulnerability types fit a given contract?
RQ2 Injection	What is the proportion of correct injections, and which types are most suitable?
RQ3 Demonstration	Are the resulting contracts useful for analyzing detection capabilities of static analyzers?

APPROACH

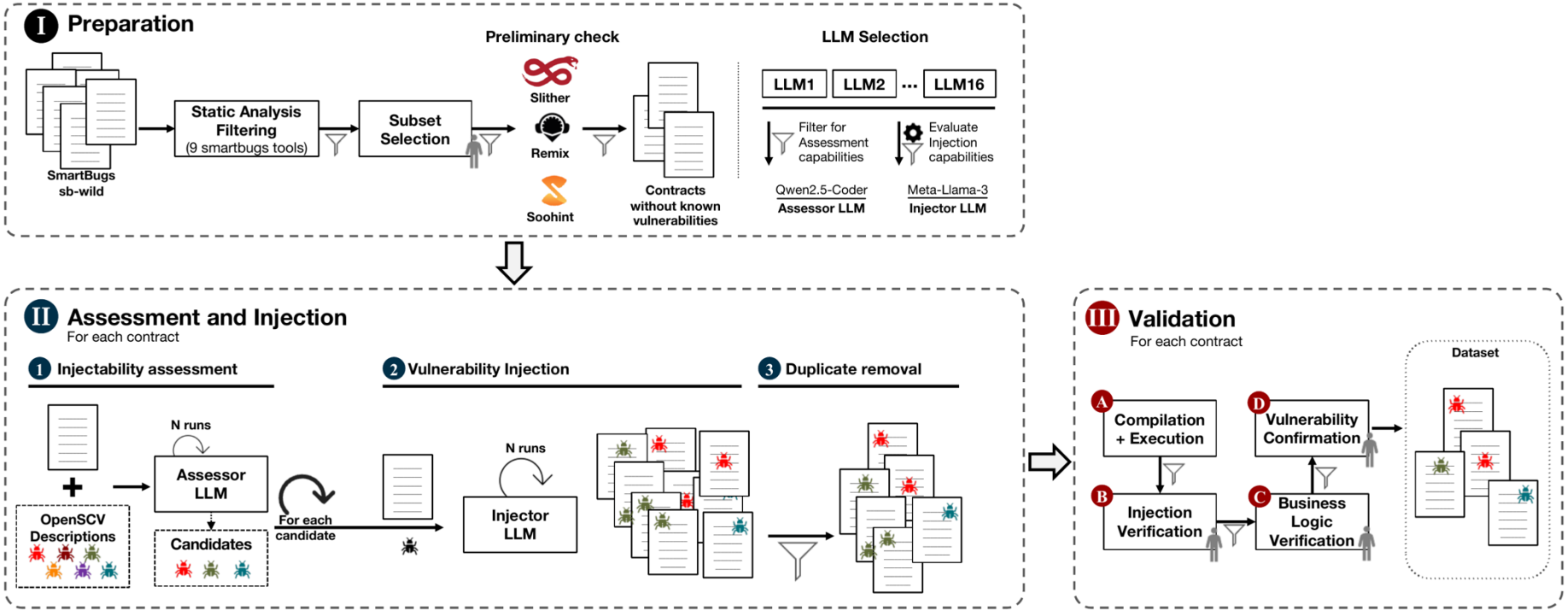
A four-phase experimental framework



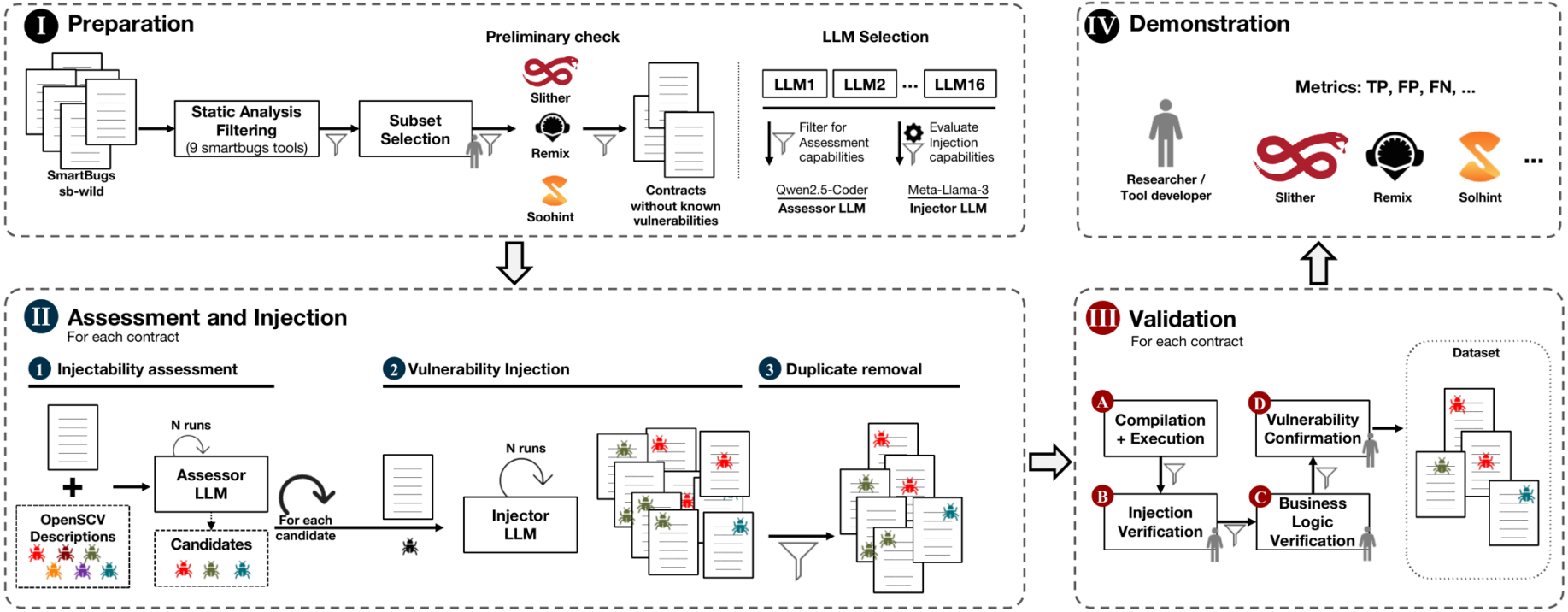
A four-phase experimental framework



A four-phase experimental framework



A four-phase experimental framework



PHASE II

Assessment and injection

ASSESSOR · Qwen2.5-Coder 14B

Picks, for each contract, which of 49 OpenSCV types can be introduced through local edits.

- **Prompt:** role + rules + 49-type list + JSON
- **Output:** score over 10 runs, threshold 50%

INJECTOR · Meta-Llama-3 8B

Generates one vulnerable variant per pair, modifying one function with a // VULN HERE marker.

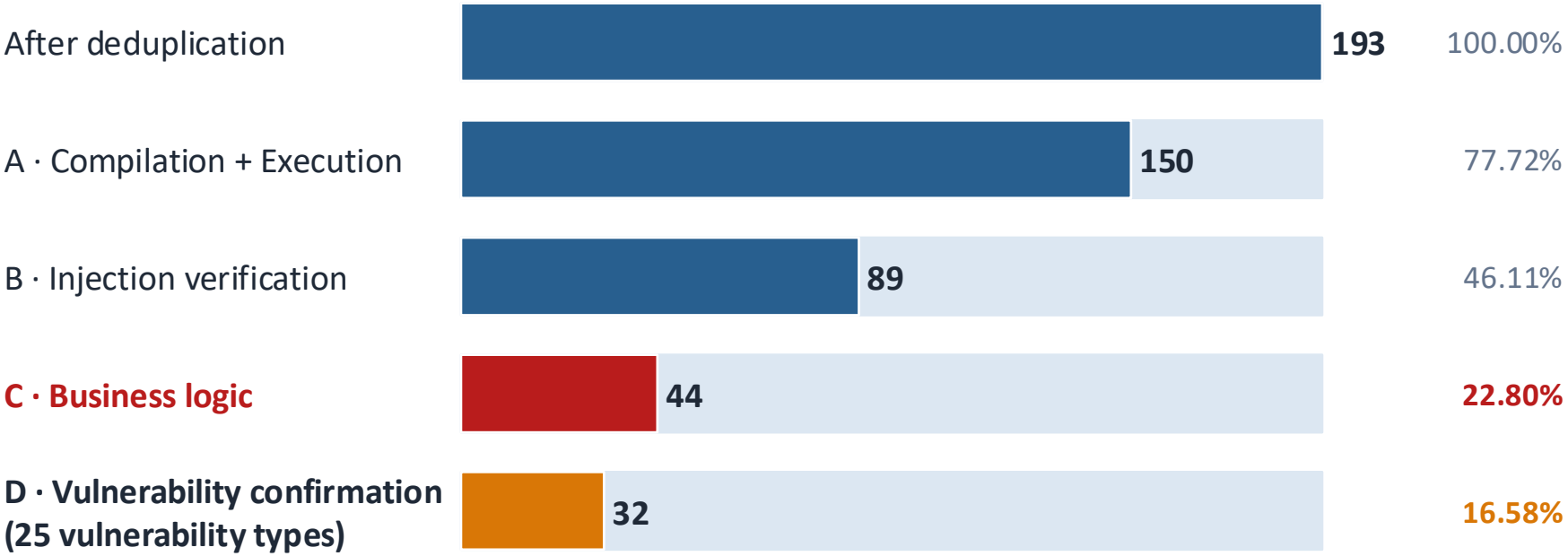
- **Prompt:** task + constraints + secure/vulnerable example
- **Constraints:** no new functions or defenses,…

49	25	997	193
OpenSCV vulnerability types	pass injectability threshold	raw LLM generations	distinct after deduplication

PHASE III

Multi-step validation pipeline

997 candidate variants



Survival rate: 16.58%. Step C (business logic) cuts the survivors roughly in half.

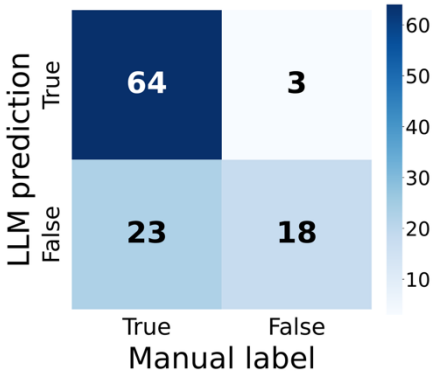
Conservative, accurate where syntax suffices

75.9%
Accuracy
108 manual labels

95.5%
Precision
3 false positives

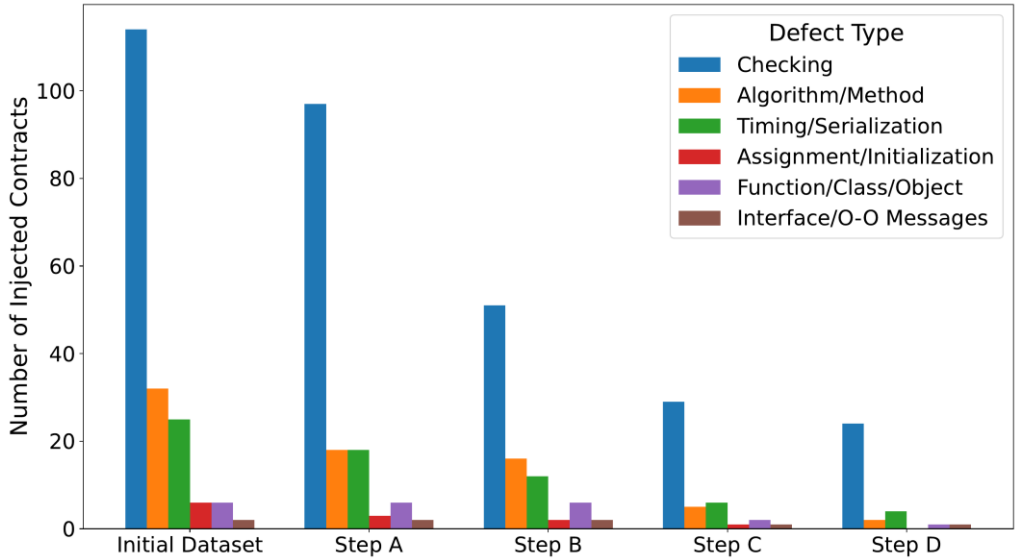
73.6%
Recall
23 false negatives

- **25 of 49** types pass the 50% injectability threshold.
- Few false-positives, but misses about 23 of injectable types
- Quite good on syntactic-pattern types; drops sharply on types needing semantic reasoning.



The dataset is biased toward Checking defects

- **11 of 25** surviving types belong to the **Checking** class.
- Structural defects rarely survive: minimal local edits cannot reach them.
- Simpler contracts also yield far more validated injections.



RQ3 · TOOL COMPARISON

The tools show complementary capabilities, but cannot handle all contracts

Tool	TP	FP	FN	Prec.	Recall
Slither	11	0	8	**1.000**	0.579
Solhint	12	2	5	0.857	**0.706**
Remix	11	3	5	0.786	0.688

32 validated contracts; 13 fail static analysis on legacy Solidity, leaving 19 analyzable.

WHAT WE SEE

- **Slither** — perfect precision, lowest recall
- **Solhint** — highest recall, some FPs
- **Remix** — middle ground, more FPs

Coverage is partially complementary. Only 9 of 25 types are detected by any tool.

40.6% of contracts could not be analyzed by any tool.

LESSONS LEARNED

Where LLM-based injection helps, and where it does not

- **Assessor approval is not enough.** Business logic preservation is the real bottleneck.
- **Generated diversity is limited.** 997 generations collapse to 193 distinct outputs.
- **Complexity amplifies failure.** Higher CC and more external calls reduce survival at every step.
- **Defect type shapes the dataset.** Checking-class vulnerabilities survive; structural ones rarely do.
- **Manual validation does not scale.** Steps B–D are entirely manual; new strategies will be required.
- **Recent SATs cannot handle older solidity versions.** 13 of 32 contracts cannot be analyzed by any tool.

Questions?

Nuno Laranjeiro
cnl@dei.uc.pt