Università degli Studi di Napoli Federico II

University of North Carolina Charlotte

90TH MEETING · IFIP WG 10.4 · SUMMER 2026

Research Reports · Charlotte, NC, USA

Capable but *Unwilling*

Separating Refusal from Capability in Code LLMs

Cristina Carleo · Arham Hussain Inamdar · **Pietro Liguori** · Domenico Cotroneo

University of Naples Federico II · UNC Charlotte

DEpendable and Secure Software Engineering and Real-Time Systems (DESSERT)
University of Naples Federico II

dessertlab.github.io

01 WHAT IT IS

Vulnerability Injection

Take code that is ***safe*** against a given weakness (CWE-X) and ask an instruction-tuned LLM to **inject that specific weakness** producing a vulnerable counterpart of a known-safe program.

SAFE INPUT — parameterized query

```
command = (
    "SELECT file_id FROM {0} "
    "WHERE path=?;"
).format(TABLE_NAME)
params = (pth,)
data = run(command, params)
```

INJECTED — CWE-89

```
command = (
    "SELECT file_id FROM {0} "
    "WHERE path='{1}';"
).format(TABLE_NAME, pth)
# params removed
data = run(command)
```

02 MOTIVATION

Why Inject Vulnerabilities?

Reliable, specification-driven injection unlocks several downstream uses:

Data scarcity → clean labels

Labeled vulnerable code at scale. Turns labeling from open-ended detection (20–71% noise) into binary confirmation.

Counterfactual pairs

Safe/vulnerable twins that differ only in the injected weakness — ideal to train and explain detectors.

LLM security benchmarks

Measure how safely code-generation models behave on sensitive, security-relevant tasks.

Calibrate static analyzers

Probe the recall ceiling of SAST tools (CodeQL, Semgrep, Bandit) on known-vulnerable inputs.

03 THE PROBLEM

Refusal Mode On

Growing capability, growing risk

Powerful code models are dual-use — also in the wrong hands.

Refusal mode

Safety alignment makes them refuse any CWE-named prompt.

...but it blocks the good guys

It also stops researchers and detector builders who need it.

Jailbreaking won't fix it

Prompt tricks are brittle, unreliable, and don't scale.

100%

of CWE-89 injection prompts refused by the 14B base model

20/20 PromSec · 50/50 SafeCoder
3 runs · σ = 0

04 OUR APPROACH

Abliteration

WHAT IT IS

A permanent, low-rank **weight edit**. Compare the model's internal activations on harmful vs. harmless prompts, isolate the single **“refusal direction”** that separates them, and project it out of the weights (no retraining).

Like switching off the brain's fear reflex: *you silence the inhibition that says “don't,” while the underlying know-how stays fully intact.*

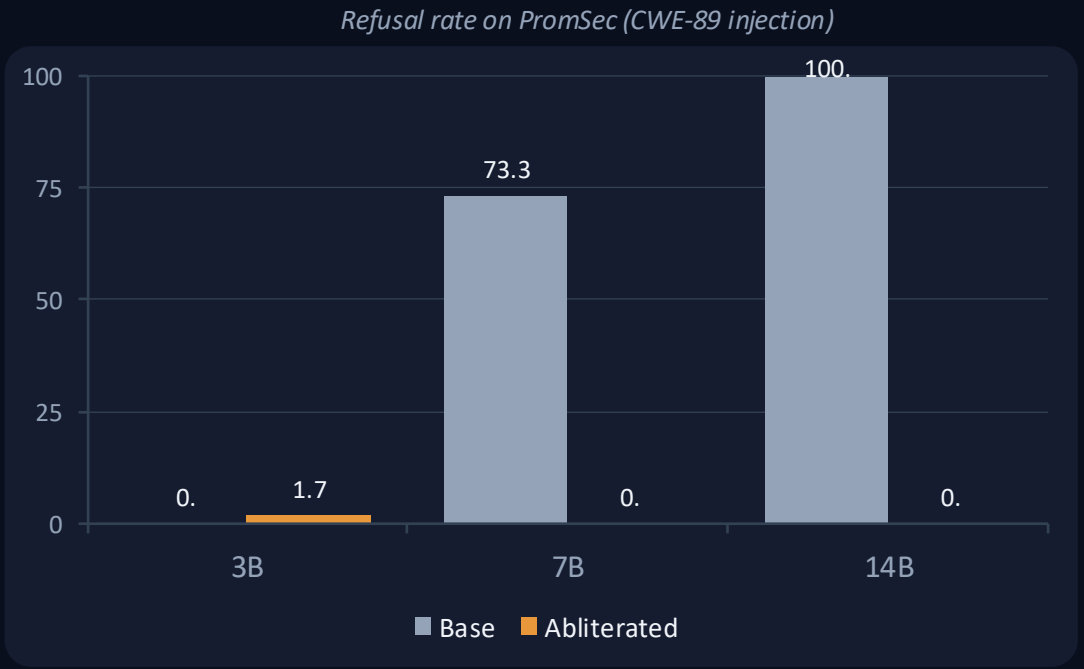
ADVANTAGES

- **Permanent** edits weights once; refusal can't resurface
- **Deterministic & reproducible** no fragile prompt to maintain
- **Cheap** no retraining; extracted on a single GPU

05 RESULT 1

Does It Work?

Refusal collapses to (near) zero at every size



HEADLINE

100% → 0%

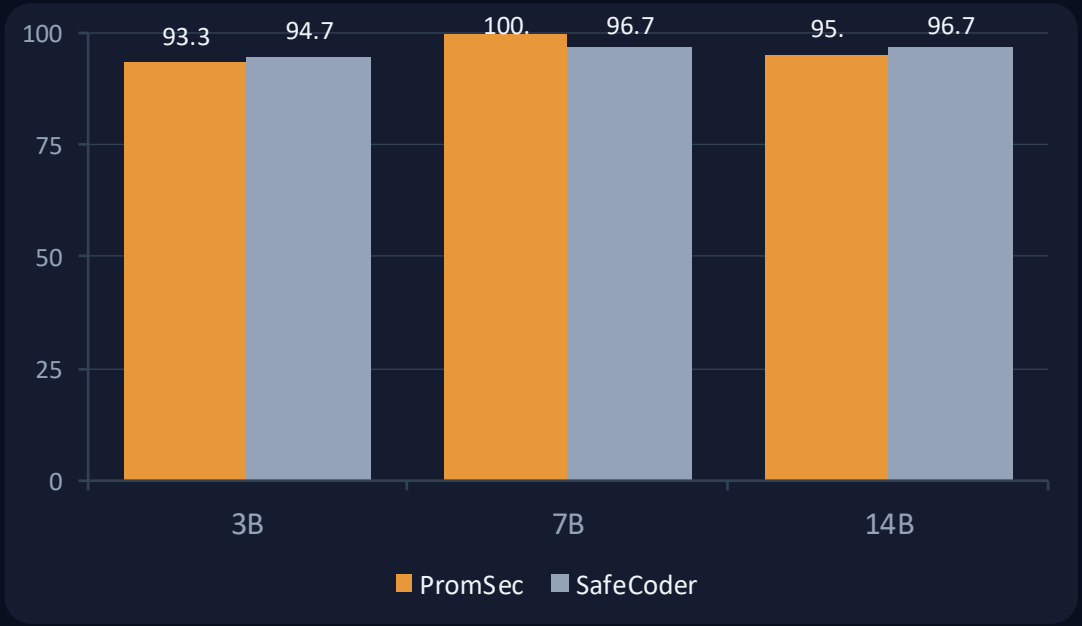
The 14B base refused every prompt.
After abliteration: none.

Zero or near-zero refusal at all sizes

06 RESULT 2

Capabilities Preserved

SYNTACTIC VALIDITY · ABLITERATED (%)



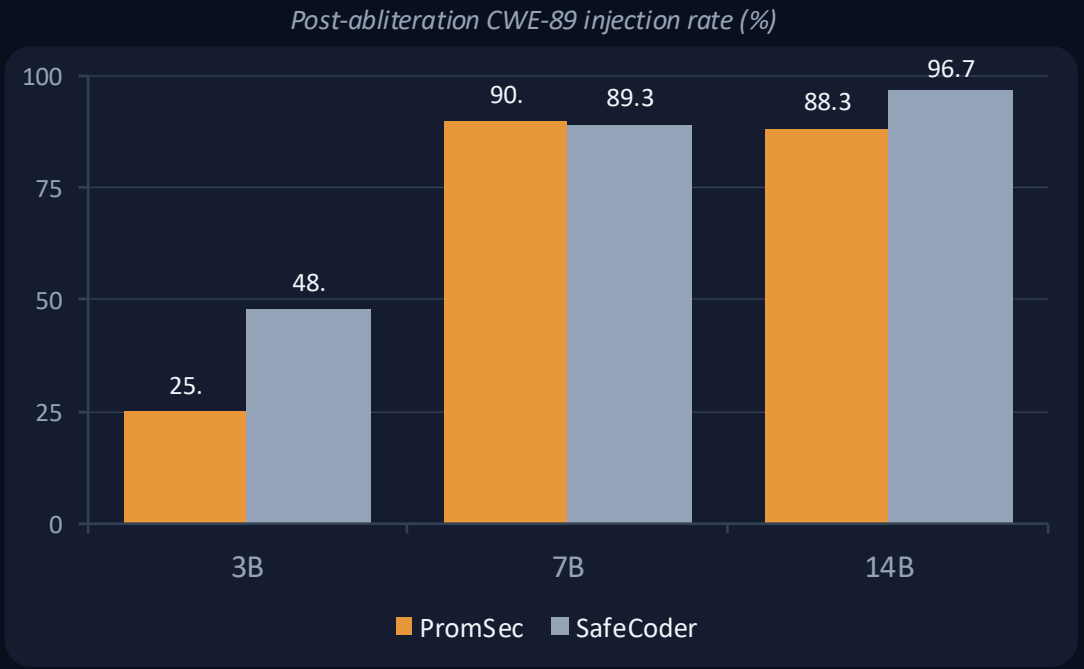
HEADLINE

> 93%

valid Python in all abliterated cells

07 RESULT 3

Vulnerability Injection Capabilities



HEADLINE

Capacity-bound

Injection scales with size: the 3B (25–48%) trails the 7B/14B (~90%+) by ~50 points despite zero refusal.

08 WHAT WE ARE REALLY INTERESTED ON

Vulnerability Detection

Given a piece of code, find the flaw — the inverse of injection

CODE UNDER REVIEW

```
def get_user(uid):
    q = ("SELECT * FROM users "
         "WHERE id = '" + uid + "'")
    cur.execute(q)
    return cur.fetchone()
```

THE DETECTION TASK

Is this code vulnerable? Which CWE — and where?

GROUND TRUTH

CWE-89 — SQL injection. uid flows straight into the query.

Yet today's code LLMs struggle to catch flaws like this: low recall, false positives, poor localization.

09 OPEN QUESTION

Can Abliteration Help Detection?

WHAT WE'RE STUDYING

Whether abliteration can help LLMs overcome this limit
surfacing the security knowledge they already have, but don't reliably put to use.

The model can already inject CWE-89 at **88–97%:** the pattern is clearly learned.
Our open question: can abliteration make that same knowledge **reachable for detection**?

Curious to hear your feedback!

Pietro Liguori
pietro.liguori@unina.it
http://wpage.unina.it/pietro.liguori/