Daily Scholar Papers Report — 2026-04-28

2026-04-28

Daily Scholar Papers Report — 2026-04-28

Download PDF

Window covered: 2026-04-27 → 2026-04-28 (Google Scholar alerts + user-curated self-emails, last 24 h). Backfilled on 2026-04-29 — the original scheduled run was skipped.

Executive Summary

The Scholar-alert side of this window was empty: the next batch from Google Scholar didn't arrive until late on 04-28 CST (and was caught by the 04-29 report). The user-curated self-email queue, however, contributed exactly one paper — flagged via the new STEP 2b pipeline that picks up self-addressed emails whose subject contains "paper" or "read".

That paper is **FIKA** (KTH + Université de Montréal), a pipeline that augments static dependency reachability with **dynamic executability proofs** for Java/Maven projects. Static analysis tools can show that a third-party library call site is reachable from a project's public methods, but cannot prove there is an actual input that reaches it at runtime. FIKA closes this gap: for each statically-reachable call site not covered by existing tests, it prompts an LLM to generate a self-contained "reachability scenario" — a runnable unit-test harness method that initialises the project, walks the call path, and triggers the target. The scenario is compiled, run, and validated by JaCoCo coverage tooling — success is defined by the coverage tool, never by the LLM's self-report.

Across 8 Java projects covering 3,219 third-party call sites, developer-written tests cover 1,754 (54%); FIKA adds 609 new executability proofs (42% of the not-covered sites), bringing the total dynamic guarantee to 73% — a 19-pp lift. On six of eight projects, FIKA crosses the 75%-guaranteed-coverage line. Applied to vulnerability triage, FIKA confirms strong reachability + executability for 31 of 59 CVEs that Semgrep had marked "undetermined" — converting ambiguous static results into prioritise-this-fix evidence. Cost: $0.0074 per successful scenario (DeepSeek V3.2), $4.52 total for the eight-project evaluation.

The architectural pattern echoes the rest of this fortnight's reading: LLM produces, deterministic oracle confirms.

Outstanding: 0 · **Keep:** 1 · **Borderline High-Priority:** 0

The full analysis follows.

Highlighted Papers

#	Title	Authors	Venue	Link
5.1	FIKA: Expanding Dependency Reachability with Executability Guarantees	Yogya Gamage, Meriem Ben Chaaben, Martin Monperrus, Benoit Baudry	arXiv 2604.20015 [cs.SE] (preprint, ICSE/FSE/ISSTA-style empirical)	arXiv

---

Keep Papers (Deep-Read)

5.1 · DEP-REACHABILITY · [USER-PICK] LLM-generated reachability scenarios + JaCoCo oracle lift dynamic dependency-coverage from 54% to 73% on 8 Java projects, $0.0074 per proof□□□

5.1 FIKA: Expanding Dependency Reachability with Executability Guarantees

arXiv:2604.20015

Paper

- **Title:** FIKA: Expanding Dependency Reachability with Executability Guarantees
- **Authors:** Yogya Gamage, Meriem Ben Chaaben (Université de Montréal), Martin Monperrus (KTH Royal Institute of Technology), Benoit Baudry (Université de Montréal)
- **Venue / Source:** arXiv:2604.20015 [cs.SE] — preprint, submitted 2026-04-21, ~12 pp + appendix. Empirical-track formatting (4 RQs, ablation, cost section, threats to validity, public artefact); plausible target ICSE / FSE / ISSTA / TSE.
- **Year:** 2026
- **Link:** https://arxiv.org/abs/2604.20015
- **License:** arXiv non-exclusive distribution (figures not embedded; pipeline recreated in Mermaid below).

Objective Summary

- **Problem.** Static dependency-reachability tools determine that a third-party library call site is *statically* reachable from a project's public methods, but cannot prove it is *executable* — that there exists an actual input exercising that call path at runtime. This drives notification fatigue across vulnerability scanners (Dependabot, Semgrep, etc.): thousands of "potentially reachable" CVEs that are never actually reached. Developers ignore most of them.
- **Approach.** FIKA adds a runtime-grounded layer. For every statically-reachable call site that is *not* covered by the project's existing test suite, FIKA prompts an LLM (DeepSeek V3.2) to generate a **reachability scenario** — a self-contained unit-test harness method that initialises the project state, invokes a public entry point, and provably triggers the target third-party call site. The LLM is given (i) the static call path produced by CHA + BFS shortest-path, (ii) the source code of every method along that path, (iii) the public entry point's constructors / factory methods / class-level state-setters, and (iv) feedback from previous failed attempts. Each

generated scenario is compiled and run; success is defined by **JaCoCo coverage** confirming the target line was executed — never by the LLM's self-report.

- **Workflow split.**
  - **Static phase:** CHA call graph → BFS one shortest path per `(m_e, m_d^p, m_t^{tpl})` tuple → context extraction (method bodies + entry-point boilerplate).
  - **Dynamic phase 1:** run the existing test suite under JaCoCo to collect call sites already covered by developer-written tests.
  - **Dynamic phase 2:** for each not-covered call site, LLM generates a reachability scenario; compile, run, validate via JaCoCo; on failure, feed compiler / runtime / coverage errors back to the LLM for up to 5 iterations.
- **Headline numbers (verbatim §IV, Table II):**
  - **8 Java/Maven projects** (flink, graphhopper, jooby, mybatis-3, pdfbox, tablesaw, tika, poi-tl) covering **1,363 unique third-party methods** invoked across **3,219 distinct call sites**.
  - **RQ1:** developer-written tests cover **1,754 / 3,219 = 54%** of third-party call sites.
  - **RQ2:** FIKA generates a successful reachability scenario for **609 / 1,465 = 42%** of the not-covered sites. Total dynamic guarantee climbs to **2,363 / 3,219 = 73%**, a **+19 pp** lift. On 6 of 8 projects, FIKA crosses the 75% guaranteed-coverage line.
  - **RQ3 (ablation):** removing static-analysis context (BL1 = path only) collapses successes from 609 → 221 (-64%); removing entry-point boilerplate (BL2 = path + method bodies) drops to 387 (-37%); the five-iteration feedback loop adds 167 (503 → 609).
  - **RQ4 (vulnerability triage vs. Semgrep):** of 59 Semgrep-undetermined CVEs across 13 vulnerable modules, FIKA confirms strong reachability + executability for **31** (Table V).
  - **Cost: $0.0074 / successful scenario** (DeepSeek V3.2), **$4.52 total** for the eight-project evaluation. Generation latency, however, is hours-to-a-day per project — limits CI integration.
- **Datasets:** 8 mature Java open-source projects curated from prior work by Soto-Valero et al. [26]; 13 vulnerable modules drawn from the same dataset for the Semgrep comparison.
- **Backbone:** DeepSeek V3.2 via the LangChain / LangGraph orchestration framework. CHA call graph via Sootup; AST manipulation via Spoon.

Formal Definitions Quoted (§II.A)

The paper specifies its terminology explicitly. Reproduced verbatim under fair-use:

> "**Reachability of a third-party library method.** We define library reachability as the possibility of a given entry point method ((m_e)) in a project ((p)) to invoke a target method ((m_t)) in a third-party library ((tpl))."

> "**Reachability scenario.** We call reachability scenario a code snippet that invokes a (m_e) and initializes the project state in order to trigger a call path reaching the target (m_d^p) and invoke a (m^{tpl}). To execute a reachability scenario in isolation, we implement, and run it within a unit testing framework, and use test coverage tools to confirm executability. This resembles a unit test case, except

> that its purpose is to collect evidence of third-party library call site executability, and hence a reachability scenario does not include any assertion."

The four-tuple `(m_e, m_d^p, m_t^{tpl}, path)` is the unit of work; FIKA selects exactly one shortest static path per tuple via BFS to bound the search space.

Pipeline Diagram (Mermaid recreation; original figures not embedded)

```
flowchart LR
    A[Project source + pom.xml] --> B[Static analysis<br/>CHA call graph<br/>BFS shortest paths]
    B --> C[Set of (me, md^p, mt^tpl, path)<br/>tuples]
    C --> D[Dynamic phase 1<br/>run existing test suite<br/>JaCoCo coverage]
    D --> E{Call site<br/>covered?}
    E -->|yes| F[Mark executable<br/>via dev-written tests]
    E -->|no| G[Extract LLM context:<br/>path source + entry-point<br/>ctors/factories/setters]
    G --> H[DeepSeek V3.2 generates<br/>reachability scenario]
    H --> I[Compile + run as<br/>JUnit harness]
    I --> J{JaCoCo confirms<br/>target line hit?}
    J -->|yes| K[Mark executable<br/>via FIKA scenario]
    J -->|no| L[Feedback loop<br/>compiler/runtime errors<br/>back to LLM, ≤5 iters]
    L --> H
```

Methodological Reusable Ideas

1. **JaCoCo-as-oracle.** The success predicate ("did the generated scenario actually hit the target line?") is delegated to a deterministic coverage tool, not to LLM self-report. **Reusable any time the LLM's task can be reduced to "produce code; coverage/diff tool verifies."**
2. **Static analysis as scaffolding, not as verifier.** CHA + BFS gives the LLM one short, complete path's worth of context (method bodies + entry-point boilerplate) rather than expecting it to find its way through the codebase. The 64% success drop in BL1 (path-only) is the strongest evidence that *context shape* dominates LLM scale.
3. **Iterative feedback loop with structured error signals.** Compiler errors, runtime exceptions, and missing-coverage signals are fed back to the LLM for up to 5 iterations, lifting success from 503 → 609. A clean ablation of the "give the LLM another chance with the error message" pattern that is becoming standard in agentic SE tools.
4. **Scenarios-as-tests-without-assertions** is a clarifying framing: developer-written tests verify *correctness*, FIKA scenarios verify *reachability*. The two artifact types share infrastructure (JUnit, JaCoCo) but have different success predicates.

Limitations Honestly Reported

- Java/Maven only; Gradle / Kotlin / other ecosystems flagged as open transferability questions.
- LLM choice (DeepSeek V3.2) not ablated against other backbones — model-dependence unknown.

- Generation latency dominates: “several hours, sometimes more than a day” per project. The authors propose running FIKA pre-release rather than per-commit.
- 8-project evaluation is small relative to the SE empirical-evaluation norm.

Closing Quote (one verbatim line allowed)

> “By making the third-party library interactions explicit, FIKA converts ambiguous results into verifiable evidence that developers can act upon.”

Cross-Paper Synthesis (with the surrounding fortnight)

FIKA fits squarely into the dominant architectural pattern emerging across this fortnight’s papers — **LLM produces, deterministic oracle confirms.** Four papers in two weeks now land on this recipe in four different domains.

Pattern	FIKA (5.1, 04-28)	TraceScope (4.1, 04-29)	AeroReq2LTL (4.2, 04-29)	LLMVD.js (3.1, 04-27)
Production artifact	reachability scenario (Java unit test)	evidence bundle + checklist adjudication	NL→TNL→LTL formula	exploit driver (Node.js)
External oracle	JaCoCo line-coverage check	MITRE checklist + Evidence Citation Protocol	deterministic NL→LTL rules + expert ground truth	class-specific execution oracle
Ban on self-grading	LLM never reports success — JaCoCo does	adjudicator must cite resource ID	deterministic translation stage	side-effect verification
Cost lever	$0.0074/ scenario, ≤5 iters bound	$0.04/URL median, 60s exec cap	(cost not reported)	$0.05/valid exploit

A few specific cross-cutting ideas worth pulling out from FIKA:

- **The static-analysis-as-scaffolding insight** is the same shape as TraceScope’s evidence-bundle scaffolding and AeroReq2LTL’s SpaceRDL templates. In all three cases, a non-LLM component constrains the search space *before* the LLM runs, and the LLM operates over a small, well-typed input. The pattern is more general than “give the LLM tools” — it’s “give the LLM a discrete, finite menu of structured choices.”
- **Coverage-as-oracle is broadly transferable.** Any LLM task that produces code (test generation, refactoring, bug repair, exploit synthesis) can use the same scaffold: existing coverage tooling already reports execution traces, so verifying “did the LLM’s code achieve the structural goal” is essentially free. FIKA exploits this for reachability; the same idea applies to mutation testing, fault injection, and most LLM-driven code synthesis tasks.

- **Cost reporting at the per-artefact level is unusually clean here.** $0.0074 / valid scenario lets a project owner reason directly about a budget — “I’ll spend $50 to reach 75% guaranteed coverage.” Other papers should adopt this framing.

Writing & Rationale Insights

- **The “Answer to RQx” boxed callout** after each RQ section lets readers skim the answer first and decide whether to read the methodology. Strong template — costs nothing in space, dramatically improves skim-readability. Worth copying.
- **Four-symbol cross-tool matrix notation in Table V** (, , ,) is more compact than the usual yes/no/partial/N-A grid. Useful for any paper comparing N tools across M techniques.
- **Frank cost / latency reporting.** “Several hours, sometimes more than a day” of generation time is a real limitation; many papers bury this in a final paragraph. FIKA puts it in §V.Discussion (Time budget) and again in Threats to Validity, which makes it discoverable for readers who actually want to deploy the tool.
- **Pre-release-not-per-commit framing** is a useful operational frame for any tool whose latency rules out CI integration. Acknowledge the limitation and reposition the use-case.
- **Citing your own series.** The team has multiple recent papers in the same area (lockfiles [11], breaking-update benchmark [5], Java-bytecode debloating [18]); FIKA leverages this lineage for related-work coverage and dataset reuse. Useful template for building a paper-series strategy.