SmartOracle: An Agentic Approach to Mitigate Noise in Differential Oracles

Srinath Srinivasan

Advisor: Dr. Timothy Menzies
Chair & Dept. Representative: Dr. Sharath Raghvendra
Area Representative: Dr. Wesley Klewerton Guez Assuncao

April 9th 2026

Journal Submission

Submitted to ACM Transactions on Software Engineering and Methodology (TOSEM)

Ranked No.6 Google Scholar - Software Systems

Manuscript under review

SmartOracle in a nutshell

For teams drowning in differential fuzzing noise, the path forward is not more agents, since the **domain tools** do the heavy lifting.

To solve this at production scale:

1. **Decompose, don't monolith**: agentic orchestration of lightweight models beats rigid prompt chaining on massive Large Reasoning Models (LRMs) (+16pp recall, RQ4)
2. **Tools over agents**: tool invocations are **3×** **more frequent** than sub-agent calls (traces, RQ4)
3. **Architecture beats scale**: Gemini Flash + agentic orchestration beats prompt-chained Gemini Pro at **10×** **lower cost** (RQ4)

On a 100:1 signal-to-noise real-world fuzzing campign, this cuts false positives to 18% at 10× lower cost than a monolithic LLM workflow.

Industry Interest from Google

Incoming SWE Intern, Google Cloud (GCE)

Tasked with building automated triage for high-volume cloud event alarms.
The same core problem addressed by SmartOracle.

Automated systems across industry generate alarm volumes that humans cannot manage without effective automation. Google's hiring for exactly this.

The noise problem is not unique to fuzzing.

Roadmap

Differential Fuzzing: How It Works

Fuzzing [Miller90]

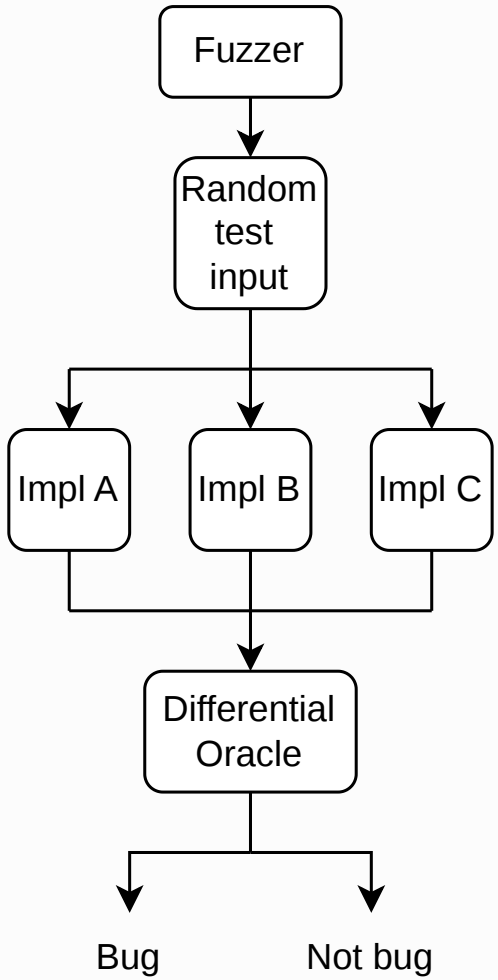
- Automatically generate massive volumes of random inputs
- Run them against a target program, look for crashes or misbehavior

Differential Testing [McKeeman98]

- Feed the *same* input to **multiple implementations** of the same spec
- Disagreement between implementations signals a potential functional bug

Differential Oracle

- The *judge*: decides whether a discrepancy is a **true bug**
- This is the hardest part [Dinella22 ICSE], and the bottleneck SmartOracle targets



Differential Fuzzing = Fuzzing + Differential Testing + **an Oracle to judge the output**

Motivation: Why Oracles Are the Bottleneck

Generating inputs is a solved problem. Judging them is not. [Barr14 TSE]

- **Overwhelming Volume:** 50 CPU cores for 2 days generates **10,000+ differential findings**.
- **High Noise:** Most findings are not bugs; they are allowed behaviors, undefined behaviors, or spec evolution [Lima21].
- **Expensive Triage:** Manual review takes **30+ minutes per finding** for domain experts [Wachter25 NDSS].
- **Brittle Automation:** Traditional rule-based filters break every time a specification changes or a new edge-case is discovered [Bushart23].

At modern fuzzing scales, the bottleneck is no longer finding bugs. It is filtering noise.

The Ultimate Stress Test: JavaScript Engines

JavaScript engines boast one of the most hostile differential testing environments.

Inescapable Scale:

- Powers **98.7% of all websites** [W3Techs25]
- Four distinct production engines (V8, JSC, SpiderMonkey, GraalJS) with diverging legacy behavior.
- Generates a signal-to-noise ratio exceeding **100:1** in production fuzzing.

Unforgiving Maturity:

- Engines are **decades-mature**. Surface bugs were fixed long ago.
- A valid new bug requires hitting hyper-specific edge cases.
- Publishing real JS engine bugs is rare enough to be noteworthy.

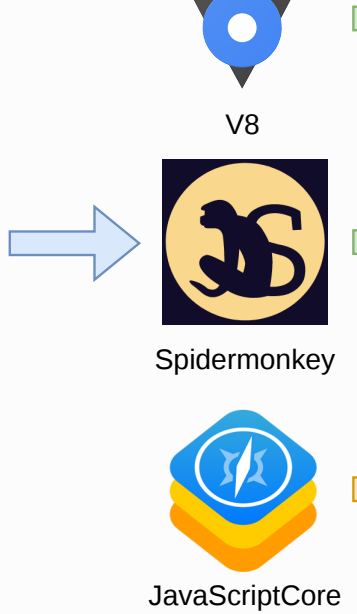
While finding new bugs in these mature engines is exceedingly rare, their daily execution in the trillions means even a handful of discoveries carries global impact.

Why Rule-Based Oracles Collapse

In JavaScript engines, **divergence is often within spec**. Rule-based filters can't keep up with the number of valid variations. (e.g., JSC's `write()` hits the file system, V8's prints to console).

```
function test() { return 42; }
var bound = test.bind(null);

print("Regular function:");
print(test.toString());
print("\nBound function:");
print(bound.toString());
```




```
function test() { return 42; }
function () { [native code] }
```

```
function test() { return 42; }
function() {
    [native code]
}
```

```
function test() { return 42; }
function test() {
    [native code]
}
```

All three outputs are correct. None are bugs.

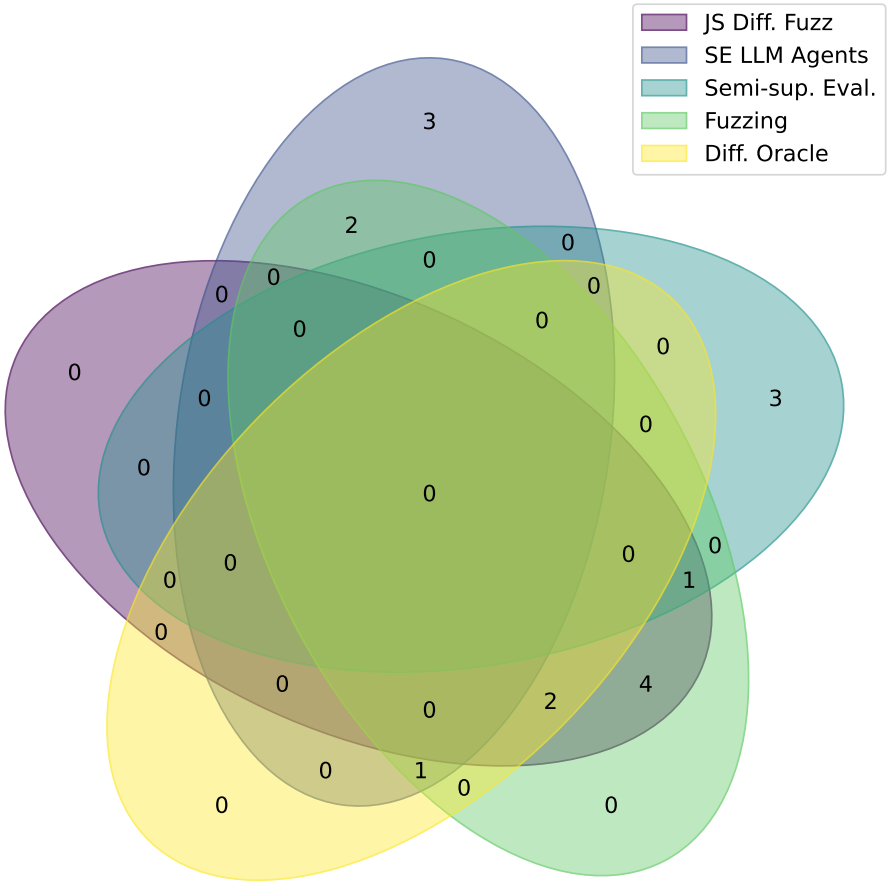
Rule-based oracles fail because they cannot adapt to moving targets.

Related Work & Research Gap

Google Scholar Search (year $\geq$ 2020):

```
("differential fuzzing" AND "JavaScript") OR
("software engineering" AND "LLM" AND "agent" AND
"fuzz") OR ("label propagation" AND "software
engineering" AND "semi supervised")
```

- **Differential fuzzing** has weak oracles [Markus24 ASE]: JEST [Park21 ICSE], JIT-Picking [Bernhard22 CCS], DUMPLING [Wachter25 NDSS], AccuOracle [Li25] are rule-based
- **LLM agents** in SE have focused on code generation [Jimenez24 ICLR, Yang24 NeurIPS]. Recent work explored test oracle generation [Hayet24 TSE].
- **Semi-supervised learning** in SE has been successful for defect prediction [Tu21 ASE], and classifying tech debt [Tu22 EMSE].



Roadmap

Decompose, Don't Monolith: Reverse-Engineering Human Triage

The Limits of Prompt Chaining:

- Fuzzing triage requires distinct cognitive phases: isolate, check specification, verify, challenge, deduplicate.
- Single-prompt models lose coherence and hallucinate when juggling multiple constraints simultaneously [Xu25].

The Agentic Solution:

- We reverse-engineered human debugging behaviors from prior literature [Layman13].
- Enforcing strict "role-playing" across smaller, focused sub-agents drastically reduces hallucination [Gosmar25].

SmartOracle emulates human expert workflow using sub-agents

SmartOracle Architecture: Agents & Tools

Orchestrator:

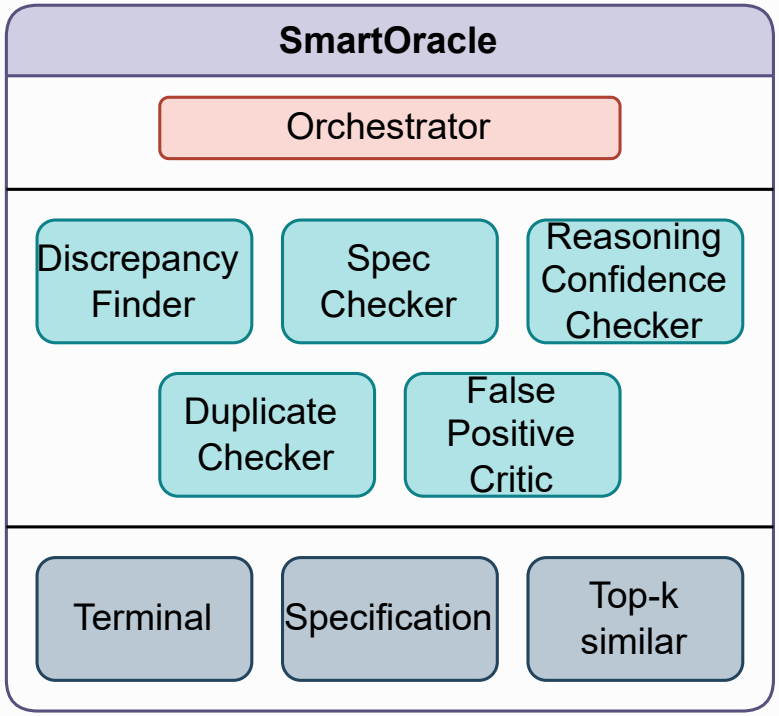
- Coordinates the workflow

5 Role-Playing Sub-Agents:

- **Discrepancy Finder**: Isolates code
- **Specification Checker**: Validates behavior
- **False Positive Critic**: Challenges findings
- **Duplicate Checker**: Avoids re-reporting
- **Confidence Checker**: Quantifies evidence

3 Domain Tools:

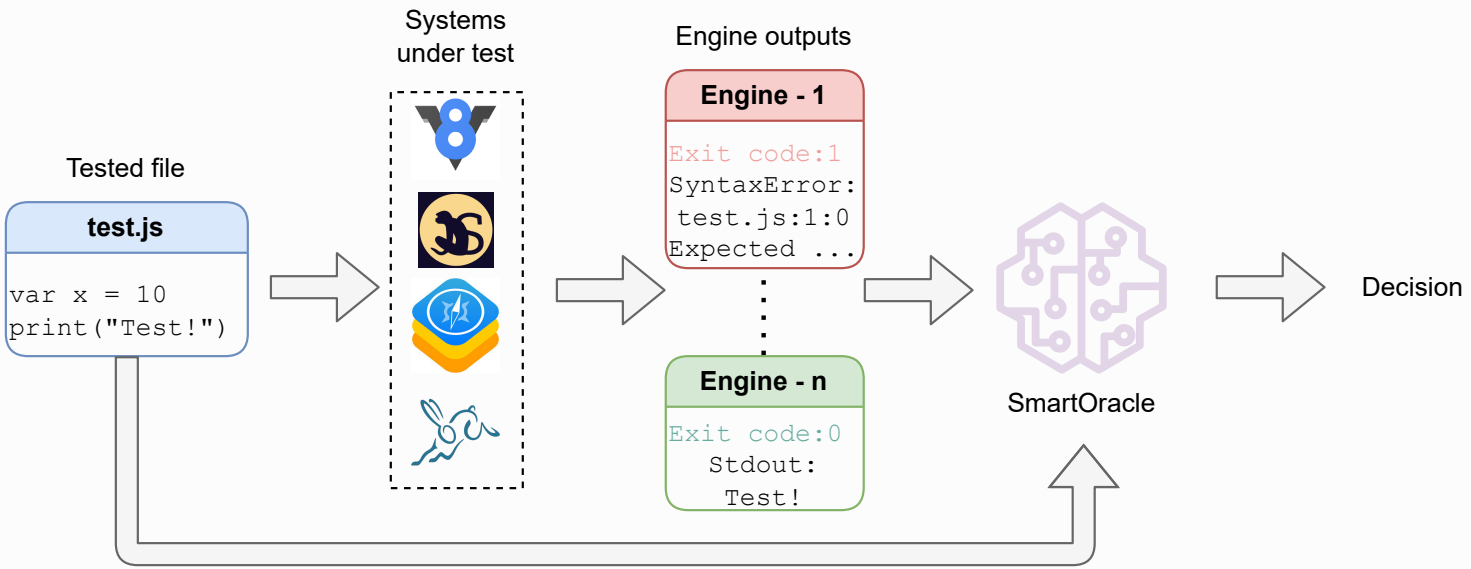
- **Terminal**: Native JS execution
- **Spec**: Cached ECMA-262
- **History**: Top-K past findings



Q: Why these specific agents?

A: Because it actually doesn't matter as much
While we designed these agents referencing debugging literature, our empirical traces reveal a plot twist: **the real power in agentic systems is the tool interaction**, not sub-agent communication.

The SmartOracle Workflow



- Agents dynamically invoke tools & sub-agents based on finding complexity.
- Terminal execution and spec-querying do the heavy lifting, acting as the ultimate source of ground truth.
- SmartOracle is fuzzer agnostic. Inputs are: tested code & engine outputs.

Roadmap

Research Questions

To evaluate SmartOracle and address the lack of ground truth in differential testing, we investigate:

RQ1 **Ground Truth:** Can semi-supervised labeling effectively overcome the ground-truth bottleneck in differential fuzzing?

RQ2 **Effectiveness:** Can SmartOracle accurately identify true bugs while maintaining a low false positive rate?

RQ3 **Bug Discovery:** Can this framework discover previously unknown bugs in mature, production JavaScript engines?

RQ4 **Efficiency:** Does an agentic architecture outperform monolithic reasoning models in triage time and cost?

The Evaluation Hurdle

We cannot evaluate an agentic system without a benchmark. Because historical differential bugs lack reproducible multi-engine environments, solving RQ1 (our semi-supervised pipeline) is the necessary foundation that makes answering the rest of these questions possible.

*Underlined topics will be defined in detail later

Roadmap

Why Historical Bugs Are Hard to Reproduce

Oracle evaluation requires a reproducible multi-engine environment, not just a bug report.

1. **The Structural Problem**
   - **Version Coupling:** Bugs require specific builds of *multiple* engines simultaneously.
   - **Rapid Decay:** A valid divergence in V8 `r9.4` may silently vanish in `r9.5`.
   - **Missing Archives:** Prior literature rarely releases version-pinned engine snapshots.
2. **The Evaluation Risk**
   - **False Negatives:** Evaluating an oracle without the exact original cross-engine environment is scientifically unsound.
   - **Regression Suites:** Standard single-engine suites only test *fixed* bugs.

Our Solution: A Live Campaign

We built our dataset from a **live 48-hour fuzzing campaign** across current engine versions, guaranteeing exact reproducibility.

Evaluation Datasets: Why These Papers?

Dataset	Engines	Bugs	Cites	Why chosen
JEST [Park21 ICSE]	V8, GraalJS, QJS, XS	44	∼37	ICSE 2021
Lima et al. [Lima21]	JSC, V8, ChakraCore, SM, Hermes	16	∼20	Software Quality Journal
Manual (ours)	V8, SM, JSC, GraalJS	**238***	–	4-engine coverage; includes diverse manifestations

Together, these span the four production engines powering >93% of web traffic.

* *Includes diverse manifestations of bugs, providing a stricter test of correctness.*

Most Recent Prior Work: AccuOracle [Li25]

A 2025 rule-based oracle for JIT differential fuzzing. Direct comparison is restricted because their bug set is unreleased and focuses on JIT bugs & not on specification divergences.

Generating Ground Truth in a Label-Starved Domain

The Challenge: JavaScript is a label-starved domain. Bugs are hard to find, making evaluation impossible without manufacturing surrogate labels.

The Insight: We don't label everything. We label clusters.

The Semi-Supervised Pipeline:

1. **Generate:** 48-hour KITTEN [Xie25 ISSTA] campaign across 4 engines $\rightarrow$ **~10,000 findings**
2. **Group:** Segment findings by their **cross-engine exit code tuples**
3. **Cluster:** Apply **K-means** to group similar behavioral states within each tuple
4. **Propagate:** Human domain experts triage only the **cluster medoids**, propagating the label to the rest of the cluster

A massive labeled dataset at a fraction of the manual cost.

Roadmap

RQ1: Can Semi-Supervised Labeling Replace Manual Triage?

1. The Binary Win (Bug vs. No-Bug)

- Grouping by exit codes alone achieves **98.7% accuracy**.
- *Takeaway:* Exit codes are a near-perfect first filter to separate signal from noise.

2. The Root Cause Problem (Why K-means is necessary)

- Exit codes are blunt: 1 exit code $\neq$ 1 root cause.
- Baseline propagation (exit codes alone) only achieves **77.3% accuracy** for identifying the actual failure mode.
- **K-means text clustering improves this to 89.1% (+11.8pp).***

Where K-means saves the pipeline:

Exit Code Scenario	Baseline	K-means	Gain
High Diversity (*e.g., 1 pattern masking 9 distinct root causes*)	28.6%	**92.9%**	**+64.3pp**
Already Homogeneous (*e.g., 1 pattern = 1 root cause*)	98.9%	99.3%	+0.4pp

K-means untangles messy clusters without over-segmenting clean ones.

** On a manually labeled dataset of 710 findings with 14 exit code groups and 31 root causes*

RQ2: Can SmartOracle Find True Bugs?

0.84 Recall | **18% False Positive Rate**

Dataset	Engines in dataset	Bugs	Recall
JEST [Park21 ICSE]	V8, GraalJS, QJS, XS	44	**0.84**
Lima et al. [Lima21]	JSC, V8, ChakraCore, SM, Hermes	16	0.75
Manual (ours)	V8, SM, JSC, GraalJS	238	0.73

The FPR Hurdle: Evaluating Against Known Noise

Prior work evaluates oracles solely on *positive bugs* because non-bugs are never publicized. To rigorously measure noise suppression, we evaluated SmartOracle against a purpose-built negative set of **136 manually labeled, non-reportable findings**.

Result: SmartOracle correctly suppressed 112 of these findings, yielding an **18% FPR**.

RQ3: Real Bugs in Production Engines

SmartOracle found 8 previously unknown bugs across 3 engines:

Engine	Bug Summary	Status
GraalJS #931	Parsing: Incorrect SyntaxError due to ASI	**Fixed**
V8 #438787152	Object.prototype.__proto__ assignment	Confirmed
V8 #446261067	d8: new print() doesn't throw TypeError	Confirmed
V8 #446661133	d8: write() lacks input validation	Confirmed
JSC #297423	String.prototype.search TypeError	Pending triage
JSC #298112	__proto__ identifier TypeError	Pending triage
JSC #299296	jsc write() file system side effects	Pending triage
JSC #299441	__proto__ global object TypeError	Pending triage

Is 8 bugs enough? *For decades-mature JavaScript engines, yes.*

- These are **new, previously unknown divergences** in shipping engines.
- Confirmation rate is the meaningful signal here, not raw volume.

Production Impact: GraalJS #931

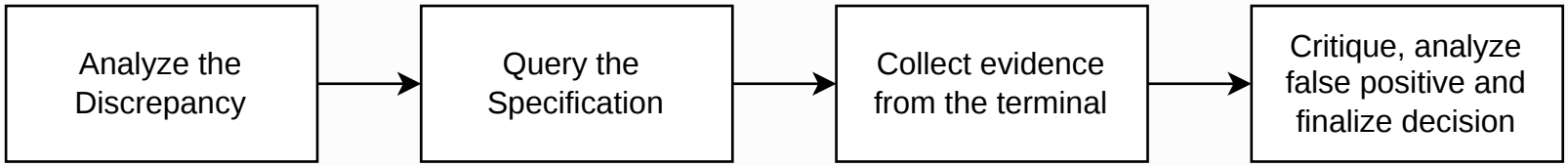
SmartOracle discovered a novel `SyntaxError` divergence that engine maintainers have since **confirmed and patched in production**. This validates that our agentic approach successfully isolates complex, valid bugs in decades-mature software.

AI-Generated Bug Reports & Developer Skepticism

Engine maintainers are increasingly wary of AI-assisted reports. Our JSC findings are **pending triage**, not rejected.

RQ4 Setup: The Baseline: Massive LRM + Prompt Chaining

To prove the value of agentic architecture, we evaluate it against a state-of-the-art monolithic baseline.



Baseline: Massive LRM + Prompt Chaining

- **Definition:** Relying on a single Large Reasoning Model to handle all phases
- **Model:** Gemini 2.5 Pro (Massive scale)
- **Mechanism:** Rigid sequential prompting
- **The Flaw:** Inflexible execution. It forces a pre-scripted path.

SmartOracle

- **Definition:** Decomposing the task into specialized roles.
- **Model:** Gemini 2.5 Flash (Lightweight / High-speed)
- **Mechanism:** Dynamic orchaestration and conditional execution.

RQ4 Results: Architecture Beats Scale

Head-to-head evaluation on the JEST dataset (44 bugs)

Method	Recall	Time	Cost
SmartOracle (Flash)	**0.84**	**20s**	**$0.003**
LRM + Prompt Chaining (Pro)	0.68	91s	$0.040

+16pp recall. 4× faster. 10× cheaper.

Triaging our entire 10,000-finding campaign costs $30.

A domain expert at 30 min/finding would take **5,000 person-hours** costing **more than $250K***
SmartOracle does it overnight for the price of a dinner.

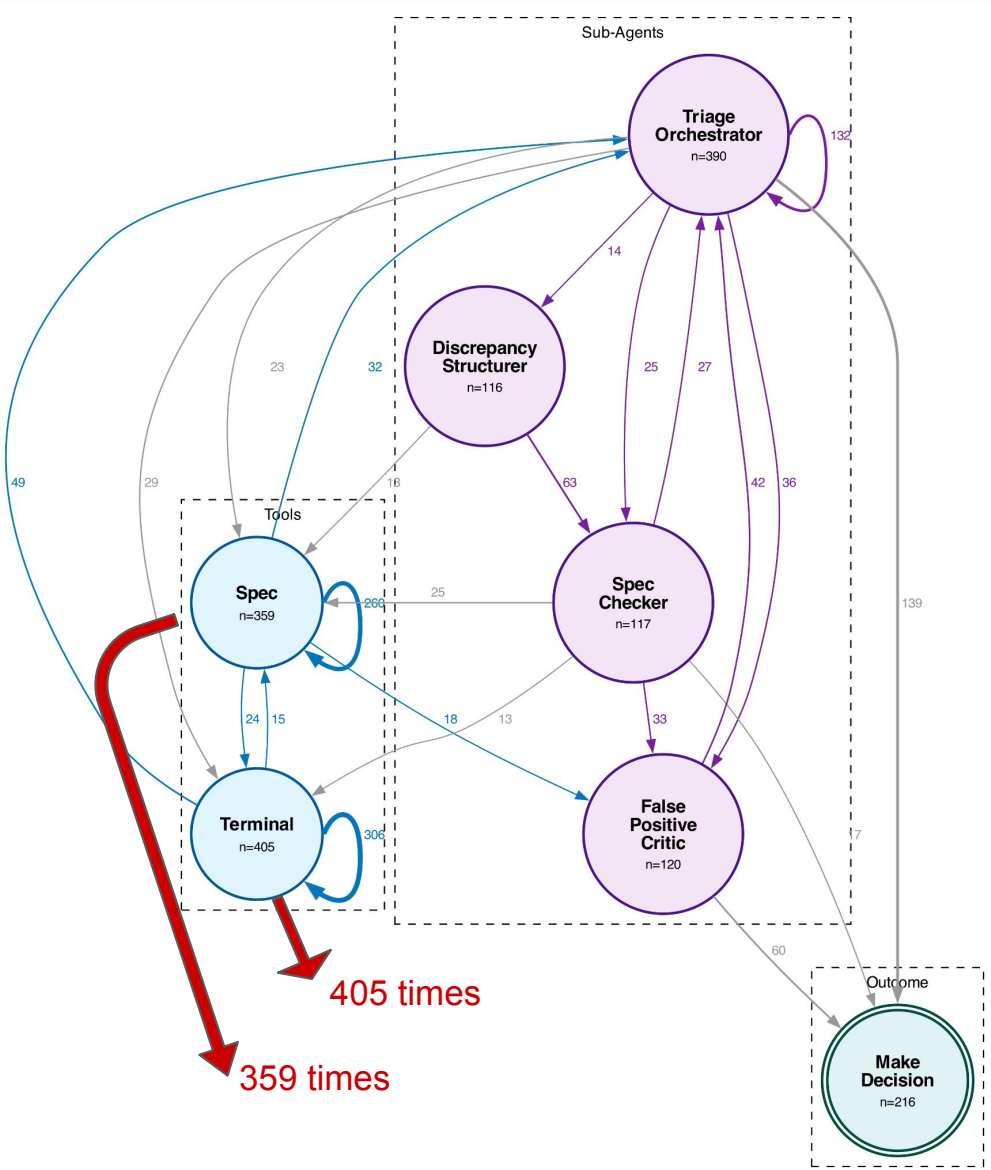
*Labor cost estimated at $55/hr.

How did we achieve 10× lower cost? The traces tell the story.

Note: Large+Agentic omitted due to prohibitive deployment costs; [Belcak25] (NVIDIA Research) supports that architecture yields higher returns than parameter scale.

What the Traces Revealed: Tools Over Agents

216 logged reasoning traces. This is empirical, not a design assumption.



The thickness of transition lines tells the story: agents spend most of their time in **evidence-gathering cycles with tools**, not talking to each other.

Key insight: Rich tool interfaces matter more than adding agents.

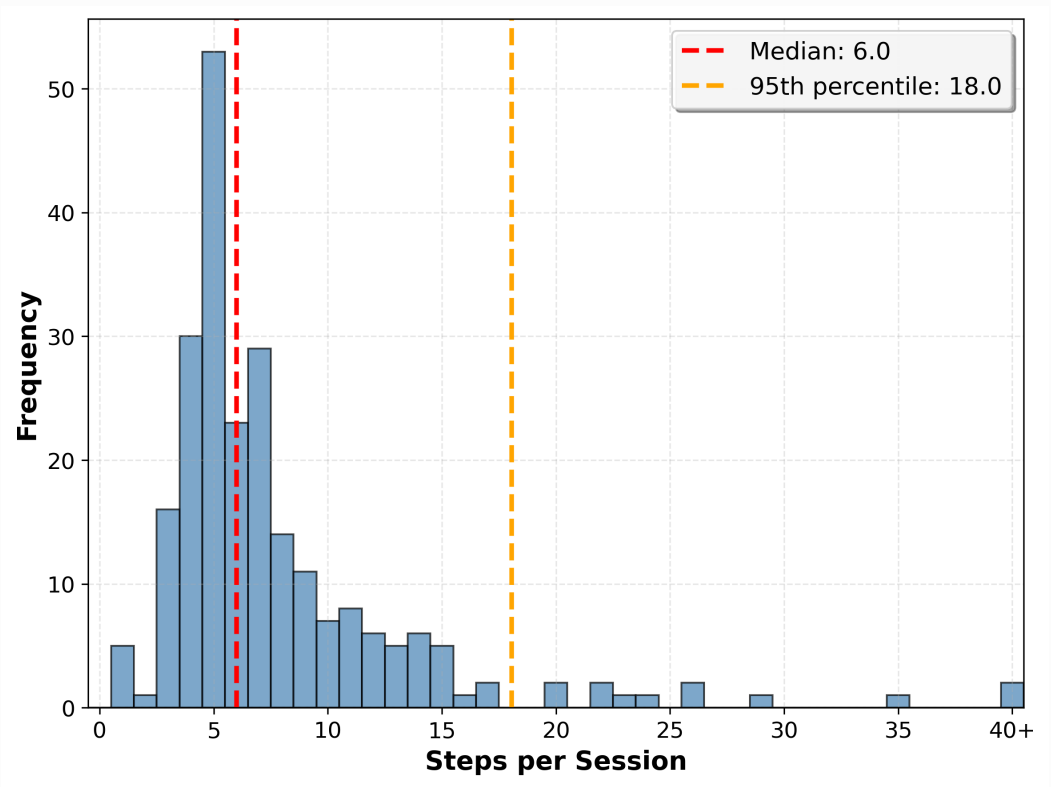
Google's Validation

[Tomašev26](Google Deepmind) reached a similar conclusion in Feb 2026 that current multi-agent delegation *"relies on simple heuristics"* and fails to generalize. **Our trace data preceded and empirically grounds this claim.**

If you had to cut one thing: cut agents, keep tools.

** sub-agents (Confidence checker & Duplicate checker) omitted to highlight dynamic reasoning*

What the Traces Revealed: Decision Efficiency



Efficiency of Conditional Execution

Monolithic models pay maximum inference costs for every query. SmartOracle dynamically scales its compute, spending pennies on obvious cases and reserving deep reasoning for edge cases.

- **Median (6 steps) — The "Fast Path":** Simple cases are resolved quickly.
- **95th percentile (18 steps) — The "Slow Path":** Ambiguous cases trigger deeper deliberation.
- **The Long Tail — "Deep Reasoning":** For the most esoteric divergences, the agent dynamically scales its effort, spending its compute budget only where needed.

Roadmap

Before vs After SmartOracle

Before		After
Noisy differential outputs	→	Structured triage via specialized agents
Single brittle rule-based oracle	→	Dynamic reasoning with 5 sub-agents
Manual filtering (30+ min/finding)	→	**Automated classification in 20 seconds**
Rigid prompt chaining with massive LRMs	→	Dynamic lightweight orchestration at **10× lower cost**
Constant rule maintenance	→	Self-updating decisions grounded in specification

What Did We Learn?

The System (SmartOracle):

- Achieves **0.84 recall** and **18% FPR** via 5 tool-augmented sub-agents
- Operates **4× faster** and **10× cheaper** than monolithic reasoning models
- Discovered **8 previously unknown bugs** in production engines (1 fixed upstream)

Three Key Design Lessons:

1. **Decompose, don't monolith:** Tool-augmented lightweight models beat prompt chaining on massive LRMs (+16pp recall)
2. **Tools over agents:** Tool invocations are 3× more frequent than sub-agent calls
3. **Architecture beats scale:** Targeted orchestration outperforms massive parameter counts

The bottleneck in differential fuzzing is filtering noise. The solution isn't bigger models, it's specialized agents wielding the right domain tools.

Future Work

Agent & Tool Ablation:

- Which of the 5 sub-agents actually drive recall?
- Traces suggest tools dominate let's quantify the margin.

Generalization:

- Apply to other domains like WebAssembly engines, databases, compilers.

Human-in-the-Loop Augmentation:

- SmartOracle already produces a **confidence score**.
- Route low-confidence findings to a human reviewer
- Feed confirmed labels back into the system
- Target: human effort only where the agent is genuinely uncertain.

Specification Drift:

- Do decisions degrade as ECMA-262 evolves?
- Build a regression harness per spec release.

Thanks for Listening!
Any questions?

References

- **[Barr14 TSE]** Barr et al., *IEEE TSE* 2014
- **[Miller90]** Miller et al., *CACM* 1990
- **[McKeeman98]** McKeeman, *Digital Technical J.* 1998
- **[Lima21]** Lima et al., *SQJ* 2021
- **[Markus24 ASE]** Fleischmann et al., *ASE* 2024
- **[Bernhard22 CCS]** Bernhard et al., *CCS* 2022
- **[Wachter25 NDSS]** Wachter et al., *NDSS* 2025
- **[Park21 ICSE]** Park et al., *ICSE* 2021
- **[Dinella22 ICSE]** Dinella et al., *ICSE* 2022
- **[Li25]** Li et al., *C&S* 2025
- **[Belcak25]** Belcak et al., *arXiv* 2025
- **[Tomašev26]** Tomašev et al., *arXiv:2602.11865* 2026
- **[He25 TOSEM]** He et al., *TOSEM* 2025
- **[Xu25b]** Xu et al., *arXiv* 2025
- **[Tu21 ASE]** Tu & Menzies, *ASE* 2021
- **[Tu22 EMSE]** Tu & Menzies, *EMSE* 2022
- **[Jimenez24 ICLR]** Jimenez et al., *ICLR* 2024
- **[Yang24 NeurIPS]** Yang et al., *NeurIPS* 2024
- **[Hayet24 TSE]** Hayet et al., *IEEE TSE* 2024
- **[Xie25 ISSTA]** Xie et al., *ISSTA* 2025
- Full bibliography in paper