INTERNATIONAL JOURNAL OF INFORMATION AND COMMUNICATION TECHNOLOGY TRENDS

Vol. 6 No. 01 (2026)

e.ISSN: 2409-5222

https://ijictt.ictt.info

ProvLevel-Reason: An Oracle-Scored Benchmark for LLM Reasoning over Software Build-Provenance Attestations

Haoyu Zhang

Northeastern University Boston,

zhang.haoyu6@northeastern.edu

Zhuoxi Wang

Northeastern University Boston, USA

zhuoxi2000@gmail.com

Abstract : *A build-provenance attestation is the document a consumer reads to decide whether an artifact was produced by a trustworthy build: it records who built it, from what sources, on what platform, and whether the record is signed. Adjudicating it means applying a laddered rubric, the SLSA build levels, to a structured in-toto/DSSE document. As language models increasingly become the readers of this metadata, the question is whether they apply that rubric or merely pattern-match on its surface. We introduce ProvLevel-Reason, a controlled benchmark that gives a model one real-format attestation and asks it to (A) decide the SLSA build level achieved (L0–L3) and (B) given a target level, enumerate exactly which requirements are unmet and check three provenance-to-artifact consistency properties. Answers are scored against a deterministic oracle that encodes the SLSA v1.1 Build Track plus in-toto/DSSE structural checks, with zero human labels; a generator plants controlled failure modes and a round-trip self-test agrees with the oracle on 2000/2000 samples. Across five API models on Task A (350 items) and four on Task B (300 items, stratified by number of unmet requirements) we find three effects. First, multi-failure attribution is the hard axis: when two requirements are unmet, direct-prompted models report only one of the two 65–79% of the time. Second, chain-of-thought cuts both ways, and which way depends on the model. It rescues multi-failure recall for most models, yet on gpt-4o-mini it collapses precision on already-satisfied targets, inventing gaps on 75% of them (exact-match 0.98→0.25). Third, a structural completeness check is the consistent weak point, and how stubbornly it persists depends on the provider. When judging level, both OpenAI models and gemini-2.5-flash fail to treat a missing buildType as disqualifying even under CoT (accuracy ≤0.04), while Anthropic models recover it; asked instead to enumerate gaps, OpenAI recovers but gemini-2.5-flash still does not (incomplete-recall 0.05), a blindness that survives both scale and task framing. With parse_fail=0 and a total inference cost of $14, ProvLevel-Reason turns "is this build trustworthy, and if not, why" into an oracle-scored test.*

Keywords: *software supply chain, build provenance, SLSA, in-toto, large language models, benchmark, attestation*

1. Introduction

The SolarWinds compromise injected malicious code during the build, not in a source commit, so source review and dependency scanning saw nothing wrong [1]. The defense the industry converged on is *build provenance*: a signed, machine-readable record of how an artifact was produced (the builder, the source, the parameters, the materials) against which a consumer can decide whether to trust it. The SLSA framework [2] grades that record on a ladder of build levels, in-toto [3] and its attestation format [4] give it a structure, DSSE [5] wraps it in a signature envelope, and Sigstore [6] and verifiers such as slsa-verifier [7] check it in CI. The push for machine-readable supply-chain metadata that followed EO 14028 [8] made such attestations increasingly common.[1] Reading that record is now often delegated to a language model: a release agent that inspects an attestation and decides whether the build clears policy, or a triage assistant asked why it does not. The judgment is rule-governed: decode the DSSE envelope, confirm the in-toto Statement carries a SLSA provenance predicate with the required fields, check the signature, check the builder against a trust root. It is also the kind of structured-document reasoning on which models are known to be uneven. Whether a model applies the ladder or pattern-matches on a hardened-looking builder name is an open, measurable question. We isolate that judgment. Because the SLSA build levels are public rubric and an in-toto/DSSE document has a fixed structure, we can synthesize attestations whose true level is fixed by construction and score a model exactly, with no human annotation. The benchmark has two tasks. Task A asks for the achieved build level. Task B is harder and the focus of our findings: given a target level, the model must enumerate the *set* of unmet requirements, not a single label, and verify three provenance-to-artifact consistency properties. Set enumeration, not single classification, is where the interesting failures live.

We contribute:

- **ProvLevel-Reason**, a benchmark of two tasks (SLSA level judgment; gap attribution with a consistency check), scored against a deterministic oracle that encodes the SLSA v1.1 Build Track plus in-toto/DSSE structural checks (Section 3);
- a generator that plants controlled failure modes (missing core fields, unsigned, untrusted/near-miss builder, hermetic decoys, tampered digests, broken envelopes) with the level fixed by construction, self-tested against the oracle on $2000/2000$ round-trip samples, all with **zero human labels**, and with the exploitable structural fact that the achievable gap set has cardinality at most two (Section 3);
- an evaluation of up to five API models surfacing three effects: direct-mode under-reporting of multi-failure sets, a capability-dependent double edge of chain-of-thought, and a provider-specific completeness blindspot that survives both scale and task framing (Sections 4-5).

All runs use temperature$= 0$, a strict line-format parser, fixed seeds, and cost $14 in total.

2.Related Work

2.1 Build provenance, attestation, and verification

SLSA [2] defines the build-level ladder we encode; in-toto [3] and the in-toto attestation framework [4] provide the Statement/predicate structure, DSSE [5] the signing envelope, and Sigstore [6] the keyless-signing infrastructure behind much real provenance. Verifiers such as slsa-verifier [7] mechanically check an attestation against a trusted-builder policy; our oracle mirrors that verifier behavior (a signature from an untrusted builder tops out below L2) rather than re-implementing cryptography. These are tools and specifications, not benchmarks: none exposes a controlled, oracle-scored item bank for the *reasoning* a consumer performs over an attestation.

2.2 Software supply-chain attacks and metadata

Build-time injection is a distinct and severe class of supply-chain attack [1], [9], and provenance is its targeted defense, supplying the transparency and artifact-integrity properties

[1] We treat EO 14028 only as historical origin and make no claim about current policy.

a secure chain requires [10]. A Software Bill of Materials enumerates the components present [11] but answers a different question: what is in the artifact, not how it was built and whether that build is trustworthy. Provenance reasoning sits at the build-integrity layer above the bill of materials.

2.3 LLMs for security and structured-document reasoning

Models reason about source-level vulnerabilities unreliably [12], [13] and struggle with the structure and semantics of security-relevant artifacts [14]; prompt and document formatting alone can shift their accuracy [15]. That work operates on source code or natural-language advisories. Our object is a signed, structured build-provenance document and a laddered rubric over it: a different layer of the supply chain, and a different failure mode.

2.4 Chain-of-thought and its limits

Chain-of-thought prompting [16] helps mainly on math and symbolic tasks and can fail to help, or hurt, elsewhere [17]. We add a concrete instance: on a set-enumeration task CoT raises recall but, for a small model, collapses precision on satisfied targets.

2.5 Programmatic benchmarks with deterministic oracles

Our method follows benchmarks that synthesize inputs with a deterministic, contamination-free oracle and a tunable difficulty axis, in function-calling [18] and code execution [19]. We point that design at build-provenance adjudication, where the quantity of interest is which rubric requirements a document meets.

3.The ProvLevel-Reason Benchmark

3.1 Inputs and oracle

An item is one build-provenance attestation: a DSSE envelope {payloadType, payload, signatures} whose base64 payload decodes to an in-toto Statement carrying a SLSA Provenance v1 predicate (buildDefinition, runDetails.builder.id, subject digest). The payload is shown decoded so the task tests SLSA reasoning, not base64 decoding. The consumer's trust policy, an explicit list of hosted (L2-capable) and hardened (L3-capable) builder identities, is supplied verbatim in the prompt, so membership is reasoned over, not memorized.

Gold labels come from a deterministic oracle (Algorithm 1) that encodes the SLSA v1.1 Build Track. **L0**: no provenance, or provenance missing a core field (wrong _type/predicateType, or missing subject digest, buildType, or builder.id). **L1**: complete provenance exists, possibly unsigned. **L2**: L1 plus a well-formed DSSE signature *and* a builder in the hosted set. **L3**: L2 with the builder in the hardened set. hermetic/reproducible markers are stronger orthogonal properties [20] that do not raise the level; they appear only as decoys. For Task B the oracle also returns the *set* of unmet requirements blocking a target level, drawn from the closed label set {NO_PROVENANCE, INCOMPLETE_PROVENANCE, UNSIGNED, UNTRUSTED_BUILDER, NOT_HARDENED}, plus three consistency booleans: subject-digest match, DSSE envelope well-formedness, and builder trust.

Algorithm 1. SLSA build-level oracle

```
env <- ParseDSSE(doc); f <- Facts(env.statement)
if not CompleteProvenance(env, f):
return L0
if not (env.signed and env.sigWellformed):
5.    return L1    # complete but unauthenticated
if f.builder in hardened:
return L3
if f.builder in hosted:
return L2
10. return L1       # signed, but builder not a trusted host
```

3.2 Generation and self-consistency

Attestations are sampled from fixed seeds. The generator reasons in terms of which fields to emit and signatures to attach; the oracle re-derives the level by parsing the emitted document, so they share the SLSA *semantics* but not the *implementation*, making the round-trip a genuine check. Before any model call the oracle is self-tested: on 2000 randomized configurations the

generator's constructed truth and the oracle's re-derivation agree on $2000/2000$ for Task A and on the Task B gap sets and consistency booleans. Controlled switches plant each failure mode: provenance$\in$\{none, incomplete, full\} with which core field to drop, signed, builder tier (self-hosted / hosted / hardened / near-miss), a hermetic decoy, subject-digest tampering, and envelope corruption (bad base64, empty payload, signature without value).

A structural fact we exploit

Under this oracle the achievable gap set has cardinality at most two, and every two-element set pairs a completeness failure (NO_/INCOMPLETE_PROVENANCE) with exactly one higher-gate failure, because the L2/L3 gates (UNSIGNED, UNTRUSTED_BUILDER, NOT_HARDENED) are mutually exclusive. "Multi-failure" therefore means exactly two unmet requirements, giving a clean single-vs-multi stratification. The bank asserts each cohort's oracle gold equals its intended label set at build time, so a generator/oracle drift cannot silently mislabel a cohort.

3.3 Tasks and scoring

Task A (SLSA level, 350 **items,** 14 **cohorts)**: output one token in {L0,L1,L2,L3}; we score exact accuracy, per cohort. **Task B (gap attribution + consistency,** 300 **items,** 21 **cohorts)**: given a target level, output the set of unmet requirement labels and three yes/no consistency answers in a strict four-line format. The bank is stratified by number of unmet requirements: *MET* (60, gap set empty), *single* (128, one unmet), *multi* (112, exactly two). We score gap-set exact-match, per-label precision/recall/F_1, and consistency accuracy.

4.Experimental Setup

We evaluate gpt-4o-mini, gpt-4o, claude-haiku-4-5, claude-sonnet-4-6, and gemini-2.5-flash through their APIs at temperature$=0$, in two prompting modes: *direct* (answer only) and *CoT* (reason step by step, then the strict output). Task A runs all five models; Task B runs four (claude-sonnet-4-6 is omitted from Task B as a cost decision, since it is at ceiling and adds little signal). A strict parser maps each response to the closed schema and counts a parse_fail otherwise. A subset of long Anthropic CoT outputs initially truncated against the token cap; re-querying only those items at a higher cap recovers all of them, a formatting artifact rather than a capability gap, leaving parse_fail$=0$ on every reported run. Runs are budget-capped and cached; total inference cost was $14.

5.Results

Table 1. Leaderboard, direct -> CoT. Task A: SLSA-level exact accuracy (350 items, 5 models). Task B: gap-set exact-match (300 items, 4 models). parse_fail=0 on every run. sonnet not run on Task B.

Model	Task A direct	Task A CoT	Task B direct	Task B CoT
gpt-4o-mini	0.543	0.823	0.503	0.623
gpt-4o	0.820	0.920	0.530	0.903
claude-haiku-4-5	0.603	0.980	0.490	0.857
claude-sonnet-4-6	0.826	1.000	---	---
gemini-2.5-flash	0.686	0.763	0.537	0.487

The leaderboard hides the structure. On Task A, Table 1 shows the expected pattern: capability lifts direct accuracy and CoT lifts it further, to a solved ceiling for claude-sonnet-4-6 (1.000). The one exception is gemini-2.5-flash, the only model still below 0.8 under CoT. Task B gap-set exact-match is uniformly lower (0.49–0.54 direct), and again CoT helps every model except gemini-2.5-flash, whose score *drops* ($0.537 \rightarrow 0.487$). The aggregate numbers do not explain why. The stratification does.

Table 2. Task B exact-match by number of unmet requirements, direct -> CoT. u-rep is the multi-failure under-report rate (gold has two unmet, model misses >=1). Direct mode collapses on multi; CoT rescues it for all but gemini-2.5-flash.

Model	MET	single	multi	u-rep
direct				
gpt-4o-mini	0.983	0.492	0.259	0.652
gpt-4o	0.783	0.805	0.080	0.705
claude-haiku-4-5	0.800	0.594	0.205	0.786

gemini-2.5-flash	0.967	0.680	0.143	0.714
CoT				
gpt-4o-mini	0.250	0.641	0.804	0.098
gpt-4o	0.883	0.938	0.875	0.062
claude-haiku-4-5	0.983	0.875	0.768	0.223
gemini-2.5-flash	0.867	0.680	0.062	0.732

Finding 1: multi-failure attribution is the hard axis. Table 2 localizes Task B difficulty in the number of unmet requirements. In direct mode every model collapses on multi-failure items: exact-match runs 0.08–0.26, and the under-report rate is 0.65–0.79, meaning that when two requirements are unmet the model names only one of them roughly three times in four. Single-failure items are far easier (0.49–0.81) and satisfied (MET) targets easier still. The failure is specifically one of *set completeness*: models identify *a* reason the target is missed but stop there, rather than enumerating all of them. In practice this is an audit that reports "unsigned" and misses that the provenance is also incomplete.

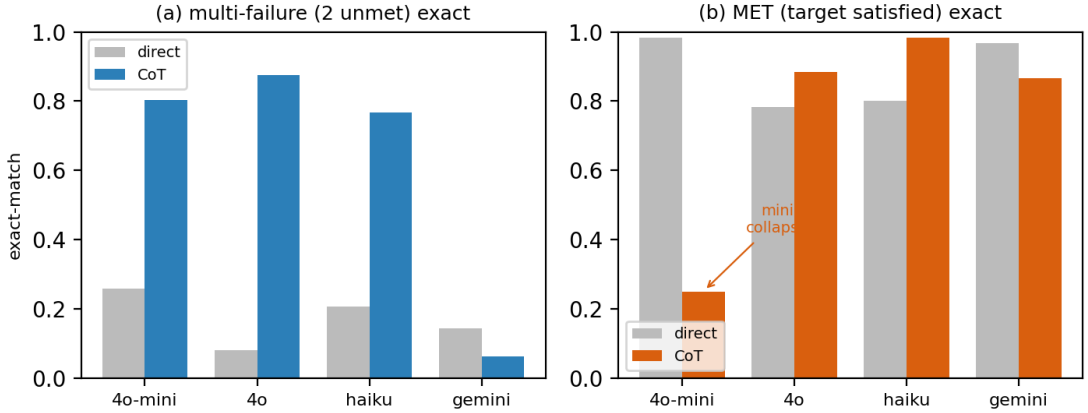


Figure 1. Chain-of-thought is double-edged. (a) On multi-failure items CoT rescues gpt-4o-mini, gpt-4o, and claude-haiku-4-5 but not gemini-2.5-flash. (b) On satisfied (MET) targets CoT collapses gpt-4o-mini while fixing claude-haiku-4-5.

Finding 2: chain-of-thought is double-edged, and capability sets the direction. CoT raises multi-failure recall sharply for three of four models (multi exact 0.08–0.26 → 0.77–0.88; Fig. 1a), confirming that the direct-mode failure is one of effort, not knowledge. But it does not come free. On satisfied targets, where the correct answer is the empty set, gpt-4o-mini under CoT *invents* unmet requirements: its hallucinated-gap rate on MET items rises from 0.017 to 0.750 and its MET exact-match falls from 0.983 to 0.250 (Fig. 1b). The same prompt moves claude-haiku-4-5 the other way (MET 0.800 → 0.983). CoT trades precision for recall on this set-enumeration task, and for the smallest model the trade is a net loss: its overall gap-exact rises only 0.503 → 0.623 because the multi-failure gain is paid back on the satisfied targets. Here "think step by step" helps the strong model and hurts the weak one on the very same items.

Table 3. Task B per-label recall under CoT: which unmet requirement gets dropped. gemini-2.5-flash almost never reports INCOMPLETE_PROVENANCE; claude-haiku-4-5 under-reports NO_PROVENANCE.

Model	**N_PROV**	**INCOMP**	**UNSGN**	**UNTRST**	**N_HARD**
4o-mini	1.00	0.94	0.99	0.90	0.90
4o	0.98	0.91	1.00	0.96	1.00
haiku	0.43	0.95	1.00	1.00	0.92
gemini	0.95	0.05	1.00	0.71	1.00

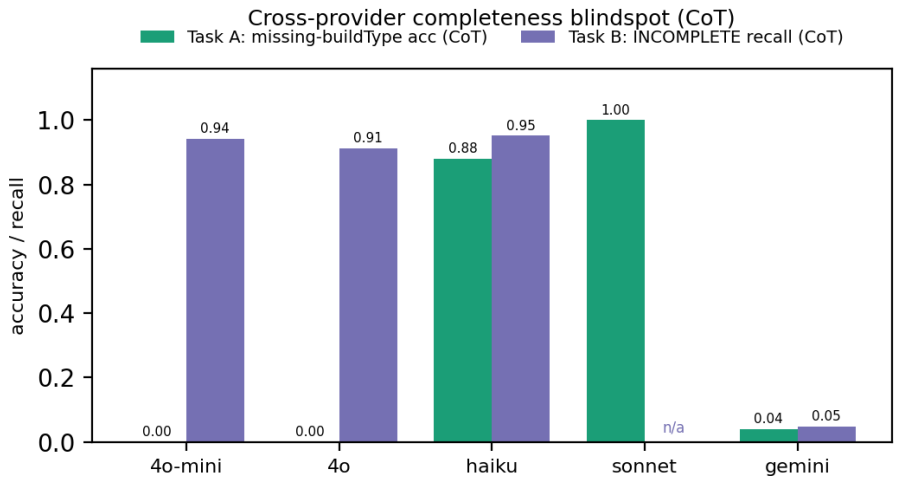


Figure 2. The completeness check is the consistent weak point, but provider-specific. Task A missing-buildType accuracy is near zero for OpenAI and gemini-2.5-flash even under CoT; only Anthropic recovers.

Finding 3: a completeness blindspot that is provider-specific and survives task framing. The single most resistant error is the structural completeness check, and the two tasks together localize it precisely (Table 3, Fig. 2). On Task A, an attestation missing its buildType is L0, but both OpenAI models judge it valid (≥L2) even under CoT (missing-buildType accuracy 0.00), as does gemini-2.5-flash (0.04); only Anthropic recovers it (claude-haiku-4-5 0.88, claude-sonnet-4-6 1.00). On Task B, where the prompt explicitly asks for every unmet requirement, OpenAI recovers (INCOMPLETE_PROVENANCE recall 0.91–0.94): the explicit enumeration framing cues the check that level-judgment skipped. gemini-2.5-flash does not recover; its incomplete-recall is 0.05, which directly explains its multi-failure collapse (0.062) and its lack of CoT benefit, since every multi-failure set contains a completeness label. The blindness is thus intrinsic for one provider (gemini-2.5-flash, across both tasks and under CoT) and framing-dependent for another (OpenAI, blind when judging level, recovered when enumerating gaps). The consistency triad is not the difficulty: digest-match is near 1.00 for all models, envelope well-formedness 0.82–0.96, and builder-trust 0.92–0.99.

6.Discussion

Three readings follow for anyone wiring a model into provenance adjudication. First, aggregate accuracy is the wrong metric: it is dominated by satisfied and single-failure items and hides the multi-failure under-reporting that matters most, since a gap report that names one of two blockers sends an incomplete remediation. An assistant should be judged on exact-set recovery under multiple simultaneous failures. Second, chain-of-thought is not a safe default here: it helps strong models enumerate failures but degrades a small model's precision badly enough to lose accuracy outright, so the prompting strategy must be chosen per model rather than turned on globally. Third, the completeness check, whether the document even contains the fields that make it valid provenance, is the brittle primitive, and one provider fails it regardless of how the question is posed. That this same structural blindness appears in two independent tasks scored by two independent banks, for the same model, argues it is a property of the model rather than of any one prompt. The operations involved (field presence, signature well-formedness, set membership) are simple, and the strongest model performs them perfectly, so the failures reflect a default disposition to read provenance by surface plausibility, not missing capability.

7.Limitations

Our attestations are synthetic, because real provenance lacks ground-truth levels without exactly the human labeling we set out to avoid; we anchor realism by emitting real-format in-toto/DSSE documents over published builder identities and by mirroring slsa-verifier's trusted-builder semantics, and we self-test the oracle on 2000 round-trip samples. We do not cross-check the oracle against an external Go implementation of slsa-verifier/in-toto; that external differential test is future work and would further harden the labels. We model L3 as

membership in a hardened-builder registry, exactly as a verifier is configured in practice, rather than attempting to assess isolation from the document alone. Finally, we evaluate single-turn adjudication; agentic settings that fetch the trust policy or re-verify signatures with tools before answering are out of scope.

8.Conclusion

As build-provenance attestations become the artifact that gates a release and models become their readers, the reasoning behind "is this build trustworthy, and if not, why" needs controlled measurement. ProvLevel-Reason provides an oracle-scored, zero-annotation yardstick for it and surfaces three effects current models share: they under-report when several requirements fail at once, chain-of-thought helps or hurts depending on model capability, and a structural completeness check is the brittle primitive, blind for one provider regardless of how the question is asked. None of these is visible in a single aggregate accuracy number, which is the case for measuring the reasoning, not just the verdict.

Reproducibility

The reference oracle, the generator, the round-trip self-test, the cohort-asserted banks, the strict parsers, the fixed seeds, the per-model response caches, and the budget ledger are released with the benchmark. All runs use temperature$= 0$.

References

P. Ladisa, H. Plate, M. Martinez, and O. Barais, "SoK: Taxonomy of attacks on open-source software supply chains," in *2023 IEEE symposium on security and privacy (SP)*, 2023, pp. 1509–1526. doi: 10.1109/SP46215.2023.10179304.

Open Source Security Foundation, "SLSA: Supply-chain levels for software artifacts, specification v1.0." https://slsa.dev/spec/v1.0/, 2023.

S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, "In-toto: Providing farm-to-table guarantees for bits and bytes," in *Proc. 28th USENIX security symposium*, 2019, pp. 1393–1410.

in-toto maintainers, "In-toto attestation framework specification." https://github.com/in-toto/attestation, 2023.

Secure Systems Lab and in-toto maintainers, "DSSE: Dead simple signing envelope specification." https://github.com/secure-systems-lab/dsse, 2021.

Z. Newman, J. S. Meyers, and S. Torres-Arias, "Sigstore: Software signing for everybody," in *Proc. 2022 ACM SIGSAC conf. On computer and communications security (CCS)*, 2022, pp. 2353–2367. doi: 10.1145/3548606.3560596.

SLSA Framework, "slsa-verifier: Verify provenance from SLSA-compliant builders." https://github.com/slsa-framework/slsa-verifier, 2024.

The White House, "Executive order 14028: Improving the nation's cybersecurity." Federal Register, 86 FR 26633, 2021.

M. Ohm, H. Plate, A. Sykosch, and M. Meier, "Backstabber's knife collection: A review of open source software supply chain attacks," in *Proc. 17th int. Conf. On detection of intrusions and malware, and vulnerability assessment (DIMVA)*, 2020, pp. 23–43. doi: 10.1007/978-3-030-52683-2_2.

C. Okafor, T. R. Schorlemmer, S. Torres-Arias, and J. C. Davis, "SoK: Analysis of software supply chain security by establishing secure design properties," in *Proc. 2022 ACM workshop on software supply chain offensive research and ecosystem defenses (SCORED)*, 2022, pp. 15–24. doi: 10.1145/3560835.3564556.

B. Xia, T. Bi, Z. Xing, Q. Lu, and L. Zhu, "An empirical study on software bill of materials: Where we stand and the road ahead," in *Proc. 45th int. Conf. On software engineering (ICSE)*, 2023, pp. 2630–2642. doi: 10.1109/ICSE48619.2023.00219.

S. Ullah, M. Han, S. Pujar, H. Pearce, A. Coskun, and G. Stringhini, "LLMs cannot reliably identify and reason about security vulnerabilities (yet?): A comprehensive evaluation, framework, and benchmarks," in *2024 IEEE symposium on security and privacy (SP)*, IEEE, 2024.

Y. Ding *et al.*, “Vulnerability detection with code language models: How far are we?” in *Proc. IEEE/ACM 47th int. Conf. On software engineering (ICSE)*, 2025. doi: 10.1109/ICSE55347.2025.00038.

Y. Li *et al.*, “SV-TrustEval-C: Evaluating structure and semantic reasoning in large language models for source code vulnerability analysis,” in *2025 IEEE symposium on security and privacy (SP)*, IEEE, 2025, pp. 2791–2809.

J. He, M. Rungta, D. Koleczek, A. Sekhon, F. X. Wang, and S. Hasan, “Does prompt formatting have any impact on LLM performance?” arXiv:2411.10541, 2024.

J. Wei *et al.*, “Chain-of-thought prompting elicits reasoning in large language models,” in *Advances in neural information processing systems (NeurIPS)*, 2022, pp. 24824–24837.

Z. Sprague *et al.*, “To CoT or not to CoT? Chain-of-thought helps mainly on math and symbolic reasoning,” in *Int. Conf. On learning representations (ICLR)*, 2025.

S. Maekawa, J. Hassell, P. Pezeshkpour, T. Mitchell, and E. Hruschka, “Towards reliable benchmarking: A contamination-free, controllable evaluation framework for multi-step LLM function calling,” in *Int. Conf. On learning representations (ICLR)*, 2026.

Gu, B. Rozière, H. J. Leather, A. Solar-Lezama, G. Synnaeve, and S. Wang, “CRUXEval: A benchmark for code reasoning, understanding and execution,” in *Proc. 41st int. Conf. On machine learning (ICML)*, in PMLR, vol. 235. 2024, pp. 16568–16621.

Lamb and S. Zacchiroli, “Reproducible builds: Increasing the integrity of software supply chains,” *IEEE Software*, vol. 39, no. 2, pp. 62–70, 2022, doi: 10.1109/MS.2021.3073045.