Deployment-Integrity Blindness in LLM-Agent Evaluation

Anonymous ACL submission

Abstract

NLP evaluation treats the model as the object under test. For deployed LLM agents, that object is instead a runtime stack: model and tokenizer, system prompts, tool schemas, retrieval sources, dependencies, and post-processing. We argue that *behavioral evidence* and *deployment-identity evidence* are orthogonal evaluation classes. Any finite output suite admits a family of behaviorally indistinguishable deployments, so output-only evaluation cannot identify which system produced a result, even in principle. We name the resulting gap *deployment-integrity blindness*, prove a short indistinguishability proposition showing it cannot be closed by stronger behavioral tests, and illustrate it empirically: across 7,560 scored examples, standard output evaluation accepts 5 of 6 controlled tamper conditions at an 0.80 suite threshold, while a minimal deployment manifest separates every tamper from clean. Two public-benchmark anchors, a GSM8K slice and the BFCL-simple function-calling subset, show the same pattern. We package the protocol as DIB-Bench[1] and lay out a falsifiable research agenda for evaluating deployed agents. As language models become deployed infrastructure, the evaluated object is the system, not the model; evaluation reports should specify and test both.

1 Introduction

NLP evaluation has historically been organized around model behavior: given inputs, collect outputs, then measure accuracy, F1, exact match, task success, refusal behavior, human preference, or model-judged quality. This remains necessary, but the unit of evaluation is no longer the model. Modern LLM applications are runtime stacks composed of model artifacts, tokenizers, system prompts, tool specifications, retrieval corpora, policy wrappers, orchestration logic, runtime dependencies, and post-processing components, all loaded and updated after training and varying across deployment instances. When a user, auditor, or downstream developer asks whether an agentic system can be trusted, the question is about the stack, not the weights. Yet a typical agent paper specifies a model id and a prompt template; the deployed system that produced its numbers is rarely described precisely enough to be reproduced or audited at the runtime level, and public agent leaderboards report a single rank per "model" even when the ranked entity is a composite system whose components could be swapped without changing the leaderboard entry. This is not a hygiene failure of individual papers; it is a missing category in the field's evaluation vocabulary.

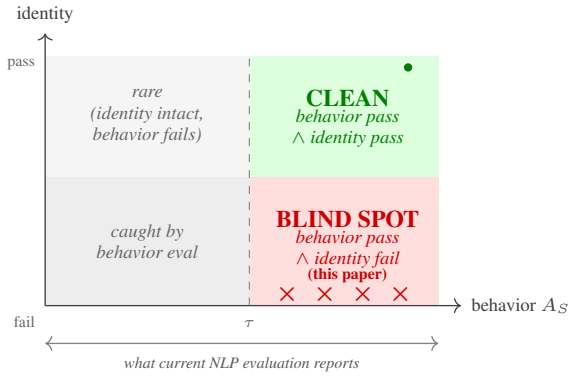

Figure 1: Behavioral evidence (A_S) and deployment-identity evidence are orthogonal evaluation classes. Current NLP evaluation reports a value only along the horizontal axis, projecting every deployment onto a single coordinate (bottom bracket). The bottom-right quadrant, where behavior passes while identity fails, is the deployment-integrity blindness this paper names. Schematic markers: a green dot for an intended clean deployment, red ×'s for behavior-preserving tampers populating the blind region. Empirical condition means are reported in Table 1.

We argue that behavioral evidence and deployment-identity evidence are formally

[1] Code and protocol: https://anonymous.4open.science/r/DIB-720E

orthogonal evaluation classes (Figure 1). Any finite output suite admits a family of distinct deployments whose outputs are indistinguishable under that suite, so output-only evaluation cannot in principle identify deployment; identity is a separate evidence channel and behavior is a separate evidence channel, and an NLP evaluation report that only specifies one of them is under-specified for any deployed agent. The relevant evaluation question for deployed LLM agents is therefore not only "what does the model output?" but "what system produced this output, and is it the system that was approved?" We name the missing class *deployment-identity evidence*, and the failure mode that follows from omitting it we call deployment-integrity blindness.

We make three contributions. First, we prove a basic indistinguishability result (Proposition 1, §2) showing that the gap between behavior and identity is structural and cannot be closed by stronger behavioral tests, including LLM-judges, behavioral canaries, and model-attribution methods. Second, we provide a controlled empirical illustration that behavior and identity can diverge maximally on the same examples: across 7,560 scored examples from 9 runs spanning 3 model-pair settings and 3 random seeds, standard output evaluation falsely accepts 5 of 6 tampered conditions at an 0.80 suite threshold while a minimal deployment manifest separates every controlled tamper from clean (§4). We replicate the qualitative pattern on two public-benchmark anchors: a 50-problem GSM8K slice and a 50-case slice of the BFCL-simple function-calling benchmark, on which all six tampers turn out to be perfectly behavior-preserving. Third, we lay out four falsifiable research directions for operationalizing this distinction (§8), each with a published-paper-scale success criterion. Our load-bearing claim is structural rather than algorithmic: behavioral evidence is provably insufficient for deployment identity in LLM agents, and NLP evaluation reports should therefore specify and evaluate both classes. Agent and prompt-injection benchmarks have already pushed evaluation beyond static single-turn tasks (Ruan et al., 2024; Yao et al., 2024; Lu et al., 2025; Debenedetti et al., 2024; Andriushchenko et al., 2025); the next step is to recognize identity evidence as a coordinate they do not span.

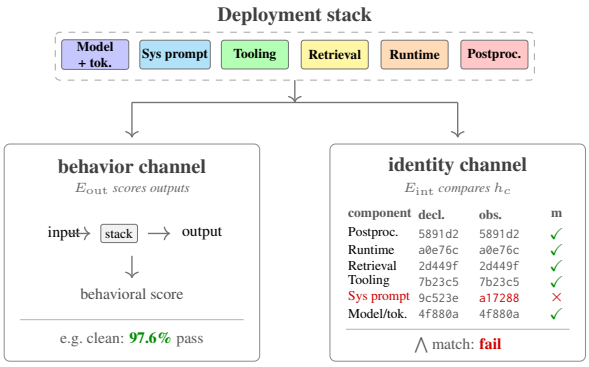

Figure 2: Two evaluation channels run in parallel off the same deployment stack (top). *Behavior channel* (left panel): E_{out} scores the generated output text. *Identity channel* (right panel): E_{int} compares declared and observed per-component hashes; a single mismatch (here, a tampered system prompt with `a17288`... $\neq$ `9c523e`...) is enough to fail the conjunction. The two channels are computed independently from the same stack, and report independent verdicts. Hashes shown are illustrative SHA-256 prefixes.

2 Behavior and Identity as Orthogonal Evaluation Classes

2.1 Evaluation Objects

Figure 2 shows what a deployment is: a runtime stack of components, each of which can change after training, after evaluation, and across deployment instances. We represent a deployment by a declared manifest M_{expected} and an observed runtime manifest M_{observed} covering model and tokenizer references, system prompt, tool specification and router, retrieval policy and corpus entries, runtime dependencies, and postprocessor identity. For an output evaluator E_{out} and an identity evaluator E_{int}, output evaluation passes if $E_{\text{out}}(\text{system}(x)) = \text{pass}$, and identity evaluation passes if $E_{\text{int}}(M_{\text{expected}}, M_{\text{observed}}) = \text{pass}$, defined as $h(M_{\text{expected}}[c]) = h(M_{\text{observed}}[c])$ for every component c, where h is SHA-256 over canonical JSON. The hash is intentionally lightweight: it isolates the NLP evaluation question rather than claiming production-grade attestation; signed provenance, in-toto-style metadata (Torres-Arias et al., 2019), SLSA attestations (SLSA Framework, 2026), or hardware-rooted attestation can substitute for the same channel role. We use *deployment-integrity blindness* for the failure mode and *deployment-identity evidence* for the channel that detects it.[2]

[2]The released implementation retains `integrity_*` metric names for code-schema compatibility.

2.2 The Structural Argument

Behavioral evidence is fundamentally a function of an output suite. For any finite output evaluation suite S with input set X_S and acceptance predicate A_S, define two deployments D_1, D_2 as S-equivalent when $A_S(D_1(x)) = A_S(D_2(x))$ for every $x \in X_S$. The basic indistinguishability fact is then:

Proposition 1 (Indistinguishability of behavior and identity)**.** *For any finite output suite $S = (X_S, A_S)$ and any deployment D_1 that passes S, there exists a deployment D_2 that differs from D_1 in at least one manifest component (i.e., $M_{\text{expected}}(D_2) \neq M_{\text{expected}}(D_1)$) but is S-equivalent to D_1. Consequently, the identity of the deployed system is not a function of $A_S(D(X_S))$.*

Proof sketch. X_S is finite, so $D_1(X_S)$ is a finite set of outputs. For any manifest component c, construct D_2 by replacing c with a variant that produces the same outputs on X_S: a fine-tune of the model satisfying $D_2(x) = D_1(x)$ for $x \in X_S$, an identity-on-$D_1(X_S)$ postprocessor, a prompt edit whose effect is conditional on a trigger absent from X_S, or a runtime swap that does not change the output function on X_S. Each such variant exists by construction. Then $A_S(D_2(x)) = A_S(D_1(x))$ for every $x \in X_S$, so D_1 and D_2 are S-equivalent. Since D_2 differs from D_1 in component c, identity is not determined by behavioral acceptance. □

Two readings of Proposition 1 are worth distinguishing. The *trivial* reading allows D_2 to modify D_1 in ways arbitrary outside X_S, so the class is infinite for free, but the modification could be a behavioral no-op of no practical interest. The *load-bearing* reading is that practical tamper classes (substituting the model, editing the system prompt, swapping the postprocessor) typically affect every input or are input-independent, so the modification is not localized outside X_S. The relevant question is whether a tamper that *does* affect every input can still be behaviorally indistinguishable from clean under a particular S, and the constructions in the proof are exactly such tampers: they are not no-ops, they are behavior-preserving precisely on X_S. The empirical version of this appears on GSM8K (§4), where four of our six tampers turn out to be strictly behavior-preserving because the math task does not elicit the outputs the tamper modifies. The structural claim is therefore not that no suite ever distinguishes any tamper; it is that no suite distinguishes *tampers designed to be invisible to it*, and an evaluator looking only at A_S cannot tell which case it is in.

Stronger output tests do not change the structure. Replacing A_S with a stricter rule-based check, an LLM-as-judge, a behavioral canary, a forensic output classifier, or a behavioral-fingerprinting attribution method (Pasquini et al., 2025; Pei et al., 2025) expands S to some larger finite S'. But Proposition 1 applies to S' verbatim: the class of tampers tuned to be invisible under S' remains non-empty. Output-only evaluation can refine behavioral coverage; it cannot identify deployment. The only way to identify which element of the S-equivalence class is deployed is a second evidence channel that depends on something other than $A_S(D(X_S))$: component hashes, signed provenance, attestation, or runtime traces. This paper uses a minimal manifest-hash channel for that role; the point is the channel, not the choice.

2.3 Blindness Metrics

The orthogonality argument motivates measuring the gap directly. We define the example-level blindness indicator as $\mathbb{1}[E_{\text{out}}(\text{system}(x)) = \text{pass} \wedge E_{\text{int}}(M_{\text{expected}}, M_{\text{observed}}) = \text{fail}]$ and report its mean as the example-level blindness rate. We also report a suite-level false-acceptance indicator at threshold τ, $\text{blind@}\tau = \mathbb{1}[\text{output_pass_rate} \geq \tau \wedge \text{integrity_pass_rate} = 0]$, which captures the operational question: would an output-only evaluation suite accept this deployment? The suite-level metric is undefined when clean does not meet τ (e.g., GSM8K, §4); example-level blindness remains well-defined in that case.

3 Protocol

We package the protocol as DIB-Bench, a reproducible benchmark and evidence interface that borrows surface structure from agent benchmarks (tasks, conditions, output evaluators, aggregate reports) and from supply-chain frameworks (manifests, per-component hashes, traces), but whose distinctive object is the bridge between them: output-passing identity failures. Appendix A gives the framework table and a benchmark card; the body of this paper uses the protocol only to anchor the orthogonality claim.

Conditions. We evaluate one *clean* condition and six single-component changes: *model sub-*

stitution (observed model/tokenizer differs from declared), *prompt tampering* (system prompt receives an unauthorized debug override), *tool tampering* (tool spec/router makes a restricted tool appear safe), *retrieval tampering* (retrieval policy or retrieved docs contain override text), *dependency tampering* (runtime descriptor changes without necessarily affecting outputs), and *postprocess tampering* (generated text is modified after inference). Each tampered condition changes exactly one manifest component, which both isolates the causal comparison and lets the identity channel be tested for component-level localization. The conditions are not an exhaustive threat model; they are controlled representatives of deployment changes not reducible to model weights alone.

Tasks and evaluators. The main study uses controlled generated tasks across five families (factual context QA, policy allow/deny, tool selection, secret-leak refusal, and canary probes) so that all conditions receive matched inputs. Each run contains 120 tasks per condition across the 7 conditions; the aggregate covers 9 runs (3 random seeds 42, 43, 44 crossed with 3 model-pair settings) for 7,560 scored examples, and every run passes validation. We compare four output-only evaluators: *standard* (task-specific correctness), *strict* (format and leakage-sensitive), *canary* (behavioral policy-override probes), and *forensic* (standard pass plus absence of obvious output-side signals). All four inspect generated text only. The identity evaluator compares declared and observed component hashes and additionally records retrieval, tool, and postprocessor traces. For ecological validity we also report public-benchmark anchors (§4) converted into the same DIB task schema used by the external-task adapter.

Models. We use three model-pair settings: Qwen2.5-14B-Instruct → Qwen2.5-7B-Instruct, Qwen2.5-7B-Instruct → Qwen2.5-1.5B-Instruct (Yang et al., 2024), and Llama-3.1-8B-Instruct → Qwen2.5-7B-Instruct (Team, 2024). The Qwen pairs probe scale-gap substitutions within a model family; the Llama-to-Qwen pair adds a cross-family axis. All inference uses vLLM (Kwon et al., 2023) with package versions recorded in the runtime manifest.

4 Results

We run the same seven conditions on three benchmarks: the controlled synthetic suite (n=7,560 across 3 model-pair settings ×3 seeds), a 50-problem slice of GSM8K (Cobbe et al., 2021), and a 50-case slice of the Berkeley Function-Calling Leaderboard *simple* subset (Patil et al., 2025). Both anchors are converted into DIB-Bench task records: GSM8K problems are wrapped to ask for a final `ANSWER: <number>`, BFCL cases to ask for `TOOL: <function_name>`; the existing fact-style and tool-style scorers apply without modification. Anchors use the Qwen2.5-7B-Instruct → Qwen2.5-1.5B-Instruct pair with one seed. Figure 3 and Table 1 report the result.

Reading across the three benchmarks. The synthetic suite places five of six tampers above the τ=0.80 suite threshold while identity is at 0%, yielding 88.6% mean example-level blindness across tampered conditions; the lone visible-control is *postprocess tampering* at 61.9% standard pass. GSM8K shows every tamper clustering near the 38% clean baseline: the four *behavior-preserving* tampers (prompt, tool, dependency, postprocess) sit within noise of clean because the math task does not elicit the outputs they modify; only model substitution shows degradation (28% standard, 10% strict). The suite-level threshold is undefined here since clean does not meet τ, but example-level blindness remains well-defined. BFCL is the empirical extreme: every tampered condition reaches 100% standard pass and 0% identity, so mean blindness is 100% and the suite threshold triggers for all six. Even postprocess tampering, which was the visible-control on the synthetic suite, is behavior-preserving on BFCL because the DENY-to-ALLOW regex does not fire on `TOOL:` outputs. The orthogonality is therefore not a property of the synthetic generator; it shows up in *stronger* form precisely where the benchmark task is most disjoint from the components the tampers modify.

Stronger output evaluators are not a substitute. Figure 3 traces this point directly on the synthetic suite: across Standard, Strict, Forensic, and Canary scoring, the behavior-preserving tampers never separate from clean, and only the identity channel does. Strict scoring (Table 1) narrows but does not close the gap: on the synthetic suite, 3 of 6 tampers still pass at τ=0.80, down from 5, and at τ=0.90

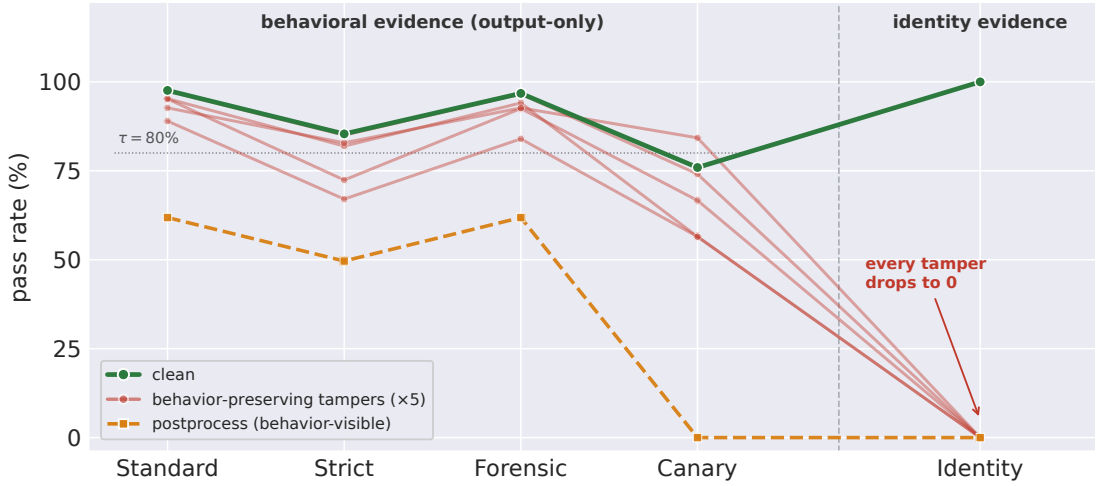

Figure 3: Behavioral evidence cannot separate tampered from clean deployments; identity evidence can. Each line traces one synthetic-suite condition across four output-only evaluators (Standard, Strict, Forensic, Canary) and then the identity channel. Through all four behavioral evaluators the five behavior-preserving tampers (red) stay bunched with the clean reference (green), most above the τ=0.80 acceptance threshold (dotted); only at the identity channel do they fall to 0% while clean stays at 100%. Postprocess tampering (orange, dashed) is the single behavior-visible control: it sits lower under the behavioral evaluators and is the only tamper the canary probe catches. No behavioral evaluator, however strict, closes the gap, which is the empirical face of Proposition 1. The figure shows the divergence across evaluator types on one benchmark; Table 1 gives per-condition numbers and extends the comparison to the two public-benchmark anchors.

standard scoring is still blind for 4 of 6 tampered conditions. On BFCL, strict scoring drops only on model substitution (to 74%, because the 1.5B model violates the exact output format more often), and even there all 50 examples still pass standard. This is the empirical face of §2: refining the output suite shrinks the S-equivalence class but cannot collapse it.

Robustness across model pairs. The synthetic-suite pattern is not a single-pair artifact. The primary Qwen2.5-14B → Qwen2.5-7B setting reaches 100% standard-output blindness for model substitution, prompt tampering, and dependency tampering. The smaller Qwen2.5-7B → Qwen2.5-1.5B pair is somewhat more behavior-visible for model substitution but blindness remains high. The cross-family Llama-3.1-8B → Qwen2.5-7B pair confirms the effect outside a Qwen-only scale gap. Appendix C reports the full per-pair table and heatmap; Appendix D reports component-localized identity evidence (each tamper mismatches exactly the component it changes, so the identity channel offers per-component evidence rather than a binary alarm).

5 Discussion

The most obvious objection is that an identity pass rate of 0% under every tamper is true by construction: we change one declared manifest component per condition and then check whether the manifest hash matches, so of course it does not. That is the demonstration, not a bug. Section 2 argues that behavior and identity are orthogonal evaluation classes; the experiment is the controlled illustration that they can disagree maximally on the same examples. The empirical interest is therefore not "does hashing work" but "under matched inputs, how high can output acceptance stay while identity acceptance is pinned at zero," and the answer is 88.6% on the synthetic suite and even higher on the GSM8K anchor for behavior-preserving tampers. A natural follow-up is that we should have used an LLM-as-judge, but the orthogonality argument says any behavioral evaluator is a finite output test whose acceptance is constant within its S-equivalence class; a frontier LLM-judge can shrink the class but cannot collapse it to a singleton, and behavioral fingerprinting and model-attribution methods (Pasquini et al., 2025; Pei et al., 2025) face the same limit. The empirical strict and forensic results in Table 1 indicate that current best-effort behavioral tests do not even saturate the class, let

Condition	Synthetic suite			GSM8K			BFCL-simple		
	Std	Strict	Id.	Std	Strict	Id.	Std	Strict	Id.
clean	97.6	85.4	100.0	38.0	36.0	100.0	100.0	100.0	100.0
model sub.	95.2	72.4	0.0	28.0	10.0	0.0	100.0	74.0	0.0
prompt	95.3	81.9	0.0	40.0	36.0	0.0	100.0	100.0	0.0
tool	92.7	82.9	0.0	38.0	38.0	0.0	100.0	100.0	0.0
retrieval	89.0	67.0	0.0	44.0	36.0	0.0	100.0	100.0	0.0
dependency	97.4	85.0	0.0	38.0	36.0	0.0	100.0	100.0	0.0
postproc.	61.9	49.6	0.0	38.0	36.0	0.0	100.0	100.0	0.0

Table 1: Condition-level results across the three benchmarks. *Std* is standard output pass rate, *Strict* adds format and leakage checks, *Id.* is identity (manifest) pass rate. All values are percentages. For every tampered row, example-level blindness equals *Std* because *Id.* is 0 by construction; for every clean row, blindness is 0 because identity passes. Synthetic-suite cells are means over 9 runs (3 seeds $\times$ 3 model pairs); anchor cells are single seeds at n=50.

alone collapse it.

The result establishes that output-only acceptance and deployment-identity acceptance can diverge sharply on the same examples, in a structurally explained rather than contingent way: a finite behavioral suite remains useful and remains insufficient. It does not establish that the manifest is a universal defense. The manifest checks declared, measured components; it does not prove that a declared component is semantically safe, that a trusted dependency has no vulnerability, or that a runtime is hardware-isolated, and it does not survive an adaptive adversary who can forge the manifest itself or tamper with components the manifest does not cover. Production systems should replace or augment this software manifest with signed provenance, build attestations, and runtime attestation where appropriate; the load-bearing claim is that the channel is needed, not that this particular instantiation is sufficient.

The reframing has cognates outside NLP that should make adoption cheaper than it appears. In software engineering and MLOps, release engineering and configuration-drift tracking are standard practice, and observability frameworks could supply the runtime-trace half of the identity channel almost for free. In HCI and trust research, the user's model of "the agent I am talking to" is treated as stable, yet the deployed stack may rotate underneath it without surface signal; identity evidence lets a user query identity rather than infer it. In regulation, the EU AI Act's logging and provenance requirements (European Parliament and Council of the European Union, 2024) and the NIST AI Risk Management Framework (National Institute of Standards and Technology, 2023) already presuppose that deployed systems are identifiable artifacts with traceable lineage. In supply-chain security, in-toto and SLSA (Torres-Arias et al., 2019; SLSA Framework, 2026) supply mature provenance machinery; our contribution is the *translation problem* of bringing that evidence into an NLP evaluation report that reviewers, leaderboards, and downstream consumers can act on.

Documentation work such as data statements, datasheets, model cards, and AI FactSheets (Bender and Friedman, 2018; Gebru et al., 2021; Mitchell et al., 2019; Arnold et al., 2019) has long argued that ML artifacts should be reported with structured contextual evidence; recent work on compound AI systems frames deployments as composite, dynamic objects rather than static models (Zaharia et al., 2024). Agentic deployments need the same move at evaluation time: the evaluated object is the model plus the prompt, tools, retrieval sources, dependencies, policies, and postprocessors that mediate its behavior, and reporting only output scores leaves that object under-specified for any consumer who is not the paper's author. Section 7 states the minimal descriptor we propose adding to close that gap.

6 Related Work

Agent and tool evaluation. Work on tool-using language models and agents has expanded evaluation beyond static prompting (Schick et al., 2023). ToolEmu evaluates long-tail risks for tool-using agents with an LM-emulated sandbox (Ruan et al., 2024). τ-bench evaluates tool-agent-user interaction in realistic domains (Yao et al., 2024), and ToolSandbox adds stateful conversational tool-use evaluation (Lu et al., 2025). AgentHarm mea-

sures harmfulness in multi-step agent settings (Andriushchenko et al., 2025). These benchmarks evaluate agent behavior. Our work asks a complementary question: can behavioral evaluation detect that the deployed runtime stack differs from the declared one?

Prompt injection and retrieval poisoning. Indirect prompt injection work shows that external content can hijack LLM-integrated applications (Abdelnabi et al., 2023). BIPIA, InjecAgent, and AgentDojo provide benchmarks for indirect prompt injection and agent robustness (Yi et al., 2025; Zhan et al., 2024; Debenedetti et al., 2024). PoisonedRAG studies knowledge corruption attacks against retrieval-augmented generation (Zou et al., 2025). Our retrieval and prompt tamper conditions are inspired by this attack surface, but the focus differs. We measure whether output-only evaluation notices that deployment components changed, not only whether a particular attack succeeds behaviorally.

Model fingerprinting and behavioral attribution. A separate line of work asks the inverse of our question: given observed outputs, can we recover the model identity that produced them? LLM fingerprinting techniques use stylistic or distributional signals to identify which base model is serving a prompt (Pasquini et al., 2025), and behavioral fingerprinting has been proposed as a runtime defense and attribution signal for LLM deployments (Pei et al., 2025). These methods sharpen the S-equivalence class but, by the structural argument in §2, do not collapse it: behavior-preserving substitutions that match outputs on the fingerprint suite remain undetectable from outputs. Our work is complementary; it argues that a second, non-output evidence channel is needed alongside any such method.

Compound AI systems and runtime composition. Zaharia et al. (2024) frame the recent shift from single-model evaluation to compound-system evaluation, arguing that the unit of analysis for modern AI applications is the orchestrated stack rather than the model. Our paper sharpens one consequence of that shift for evaluation reporting: when the unit is a stack, an evaluation report that names only the model is structurally under-specified, and the gap is not closed by adding stricter behavioral tests.

Component	Minimal identity descriptor
Model + tokenizer	weights or revision id; tokenizer and chat-template hash
System prompt	hash of the fully resolved prompt
Tooling	hash of tool specifications and router configuration
Retrieval	retrieval-policy hash and corpus snapshot id
Runtime	key dependency versions and container digest
Postprocessor	hash of the post-inference transform

Table 2: A minimal deployment-identity descriptor for an evaluation report. Each component is reported together with how the descriptor was obtained (self-declared, build provenance, or attestation) and a match verdict against the intended configuration.

Documentation, provenance, and attestation. Model cards, datasheets, data statements, and AI FactSheets emphasize structured transparency for models, datasets, and AI services (Mitchell et al., 2019; Gebru et al., 2021; Bender and Friedman, 2018; Arnold et al., 2019). Software supply-chain systems such as in-toto and SLSA provide stronger artifact-integrity and provenance mechanisms (Torres-Arias et al., 2019; SLSA Framework, 2026). Regulatory frameworks including the EU AI Act (European Parliament and Council of the European Union, 2024) and the NIST AI RMF (National Institute of Standards and Technology, 2023) already presuppose identifiable, lineage-traceable deployments. We connect these lines to NLP evaluation: if the evaluated object is a deployed LLM agent, component identity should become part of the evaluation report.

7 What an Evaluation Report Should Add

The orthogonality argument has an immediate and adoptable consequence: an evaluation report for a deployed LLM agent should carry, alongside its behavioral numbers, a deployment-identity descriptor that lets a reader verify which system produced those numbers. This is a reporting change rather than a new algorithm, and it is cheap enough to include in the next agent-evaluation paper. We propose that a report state, for each component of the evaluated stack, a stable identifier of the component as deployed, how that identifier was obtained, and whether it matches the configuration the authors intended to evaluate. Table 2 instantiates this for the manifest used here.

The descriptor is graded rather than binary. Its cheapest form is a self-reported manifest of content hashes, which is what DIB-Bench records; stronger forms substitute signed build provenance (Torres-Arias et al., 2019; SLSA Framework, 2026) or hardware-rooted runtime attestation where the threat model warrants it. What matters for the report is that the channel is present and auditable, not which point on this scale an author chooses. The descriptor is deliberately weaker than production attestation and deliberately stronger than current practice: it does not certify that a component is safe or correct, only that the deployed component is the one the report describes, which is precisely the claim behavioral numbers cannot make. A reader who has both channels can finally ask the question output scores assume away: are these numbers from the system I think I am evaluating?

8 A Research Agenda

If behavioral and identity evidence are orthogonal evaluation classes (§2), the immediate question is what work the field should do to make that recognition operational. We propose four primary directions, each scoped as a tractable next paper rather than a multi-year program, and each with a falsification criterion that would settle whether the orthogonality claim is empirically as well as conceptually load-bearing.

1. **Stealth-tamper benchmarks.** Construct tampers explicitly optimized to evade *both* channels (e.g., a model swap fine-tuned to match the original's outputs on a held-out probe set, or prompt edits style-matched to the declared prompt against a manifest-aware verifier), then report the Pareto frontier of behavioral detectability against identity-channel cost-to-forge. *Falsifies the claim if:* stealth tampers cheaply defeat both channels for realistic attacker budgets.

2. **Longitudinal deployment-drift studies.** Pick $N \geq 20$ popular open-source agent frameworks or hosted agent APIs and measure how often the observed runtime stack diverges from the declared configuration over (e.g.) three months. *Falsifies the claim if:* zero or negligible drift is observed in the wild, weakening the practical motivation.

3. **Formal undetectability characterization.** Proposition 1 is the basic indistinguishability fact; a stronger version would characterize the class of deployment changes that are PAC-undetectable from any output suite of size $\leq N$, with concrete sample-complexity bounds for the detectable class. *Falsifies the claim if:* the PAC-detectable subclass turns out to dominate practical tampers, so the indistinguishability reduces to a corner case.

4. **Leaderboard rank decomposition.** Existing agent leaderboards report a single rank per "system." Re-score one under controlled component swaps and report *two* ranks (model rank and stack rank) with a proposed leaderboard schema. *Falsifies the claim if:* model rank and stack rank coincide in practice.

Three further directions are secondary but worth doing: a practitioner mental-model survey of LLM-agent developers on whether they can enumerate runtime components without documentation; a reviewer-checklist pilot at one *ACL venue asking "did the paper specify the evaluated system, not just the model?"; and a cross-paper rescoring study under controlled component swaps for $K \geq 5$ published agent evaluations (the three-benchmark study in §4 is the smallest instance of this direction). Stating the agenda as falsifiable directions, rather than recommendations, lets the field settle the orthogonality claim by experiment: if every direction returns a negative answer, the claim is wrong and the recognition this paper urges is not needed.

9 Conclusion

The core claim of this paper is structural, not empirical. Behavioral evidence is a property of an output-suite equivalence class; deployment identity is the choice of element within that class; and no finite output suite can identify the element. NLP evaluation reports today specify only the class. This is a missing coordinate in the field's evaluation vocabulary, not a hygiene failure to be fixed by stricter behavioral tests. DIB-Bench is a minimal illustration that the two coordinates can be made to disagree maximally on the same examples; §8 lays out four primary directions (and three secondary ones) for settling, refining, or refuting the claim.

As LLMs become deployed infrastructure, the evaluated object stops being a model and becomes a system. For LLM agents, the concrete consequence is that evaluation reports should specify behavior *and* identity as separate evidence channels.

Ethics Statement

This work studies deployment tampering and prompt, retrieval, and tool manipulation in a controlled synthetic environment. The goal is defensive measurement: to show that evaluations of deployed LLM agents should report runtime identity evidence alongside behavioral results. The protocol includes synthetic secret tokens and policy-bypass prompts, but no real secrets, users, or external services. Because the work could inform stronger tamper testing, released artifacts should avoid presenting the prompts as operational attack recipes and should emphasize the defensive evaluation framing.

Limitations

The main empirical package uses controlled generated tasks for causal one-component tampering, anchored by two public-benchmark slices: a 50-problem GSM8K slice and a 50-case BFCL-simple slice (§4). These anchors illustrate the orthogonality on real benchmarks but do not characterize how it varies across benchmark domains or sample sizes; the cross-paper rescoring direction in §8 calls for a multi-benchmark replication at scale. The tampering conditions are single-component and do not cover adaptive adversaries who can forge the manifest itself, multi-component attacks, or stealthy changes to undeclared components; the stealth-tamper direction in §8 is the natural extension. The manifest is software-level evidence: it does not provide hardware-rooted guarantees, can only check components that are declared and measured, and does not establish the semantic correctness of those components. The output-only evaluators we compare against are rule-based plus a small forensic-signal classifier; we argue in §2 that the orthogonality result is robust to stronger evaluators such as LLM-judges and behavioral fingerprinting, because no finite output suite can collapse its S-equivalence class, but empirically running strong LLM-judges across these conditions would tighten the argument. Finally, Proposition 1 states the basic indistinguishability fact; it does not give PAC sample-complexity bounds or characterize which subclass of tampers is detectable from how many output samples, so it is not yet a quantitative impossibility result, which the formal-undetectability direction in §8 would supply.

References

Sahar Abdelnabi, Kai Greshake, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security, AISec 2023, Copenhagen, Denmark, 30 November 2023*, pages 79–90. ACM.

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, J. Zico Kolter, Matt Fredrikson, Yarin Gal, and Xander Davies. 2025. Agentharm: A benchmark for measuring harmfulness of LLM agents. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Matthew Arnold, Rachel K. E. Bellamy, Michael Hind, Stephanie Houde, Sameep Mehta, Aleksandra Mojsilovic, Ravi Nair, Karthikeyan Natesan Ramamurthy, Alexandra Olteanu, David Piorkowski, Darrell Reimer, John T. Richards, Jason Tsay, and Kush R. Varshney. 2019. Factsheets: Increasing trust in AI services through supplier's declarations of conformity. *IBM J. Res. Dev.*, 63(4/5):6:1–6:13.

Emily M. Bender and Batya Friedman. 2018. Data statements for natural language processing: Toward mitigating system bias and enabling better science. *Trans. Assoc. Comput. Linguistics*, 6:587–604.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. 2024. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

European Parliament and Council of the European Union. 2024. Regulation (EU) 2024/1689 of the european parliament and of the council laying down harmonised rules on artificial intelligence (artificial intelligence act). Official Journal of the European Union, L series, 12 July 2024.

Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna M. Wallach, Hal Daumé III, and Kate Crawford. 2021. Datasheets for datasets. *Commun. ACM*, 64(12):86–92.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving

with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pages 611–626. ACM.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. 2025. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, Findings of ACL, pages 1160–1183. Association for Computational Linguistics.

Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2019. Model cards for model reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency, FAT* 2019, Atlanta, GA, USA, January 29-31, 2019*, pages 220–229. ACM.

National Institute of Standards and Technology. 2023. Artificial intelligence risk management framework (AI RMF 1.0). Technical Report NIST AI 100-1, U.S. Department of Commerce.

Dario Pasquini, Evgenios M. Kornaropoulos, and Giuseppe Ateniese. 2025. Llmmap: Fingerprinting for large language models. In *34th USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025*, pages 299–318. USENIX Association.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The berkeley function calling leaderboard (BFCL): from tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net.

Zehua Pei, Hui-Ling Zhen, Yingchun Zhang, Zhiyuan Yang, Xing Li, Xianzhi Yu, Mingxuan Yuan, and Bei Yu. 2025. Behavioral fingerprinting of large language models. *CoRR*, abs/2509.04504.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. 2024. Identifying the risks of LM agents with an lm-emulated sandbox. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

SLSA Framework. 2026. Supply-chain levels for software artifacts. `https://slsa.dev/spec/v1.2/about`.

Llama Team. 2024. The llama 3 herd of models. *CoRR*, abs/2407.21783.

Santiago Torres-Arias, Hammad Afzali, Trishank Karthik Kuppusamy, Reza Curtmola, and Justin Cappos. 2019. in-toto: Providing farm-to-table guarantees for bits and bytes. In *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 1393–1410. USENIX Association.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024. Qwen2.5 technical report. *CoRR*, abs/2412.15115.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *CoRR*, abs/2406.12045.

Jingwei Yi, Yueqi Xie, Bin Zhu, Emre Kiciman, Guangzhong Sun, Xing Xie, and Fangzhao Wu. 2025. Benchmarking and defending against indirect prompt injection attacks on large language models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining, V.1, KDD 2025, Toronto, ON, Canada, August 3-7, 2025*, pages 1809–1820. ACM.

Matei Zaharia, Omar Khattab, Lingjiao Chen, Jared Quincy Davis, Heather Miller, Christopher Potts, James Zou, Michael Carbin, Jonathan Frankle, Naveen Rao, and Ali Ghodsi. 2024. The shift from models to compound AI systems. `https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/`. Berkeley AI Research Blog.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. 2024. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, Findings of ACL, pages 10471–10506. Association for Computational Linguistics.

Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. 2025. Poisonedrag: Knowledge corruption attacks to retrieval-augmented generation of large language models. In *34th USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025*, pages 3827–3844. USENIX Association.

Layer	Borrowed shape	DIB-Bench instantiation	Reported artifact
Task environment	Agent benchmarks define tasks, tools, and adversarial contexts.	DIB-Bench defines factual, policy, tool, secret, and canary task families.	Output text and task metadata.
Deployment condition	Attack benchmarks perturb prompts, tools, users, or retrieval content.	DIB-Bench perturbs one declared runtime component at a time.	Condition label and changed component.
Behavioral scoring	Benchmarks score task success, safety, and robustness.	DIB-Bench reports standard, strict, forensic, and canary output scores.	Output-only pass rates and CIs.
Integrity evidence	Supply-chain systems record provenance and artifact integrity.	DIB-Bench records expected-vs-observed component hashes and traces.	Manifest matches, mismatches, hashes, and traces.
Blindness metric	The bridge: compare behavior acceptance with deployment identity.	DIB-Bench measures output-passing integrity failures.	Example and suite-level blindness.

Table 3: DIB-Bench interface: a bridge between agent behavior benchmarks and deployment-identity evidence.

Threshold	Std. blind	Strict blind	Forensic blind	Canary blind	Mean ex. blind
0.70	5/6	4/6	5/6	2/6	88.6
0.80	5/6	3/6	5/6	0/6	88.6
0.90	4/6	0/6	4/6	0/6	88.6

Table 4: Threshold sensitivity. Counts are tampered conditions accepted by the output suite while identity fails.

A DIB-Bench Framework

Table 3 summarizes DIB-Bench as a bridge between agent behavior benchmarks and supply-chain provenance frameworks. The benchmark surface (task families, deployment conditions, output evaluators, aggregate reports) is borrowed from agent benchmarks; the identity surface (manifests, per-component hashes, mismatch lists, traces) is borrowed from supply-chain frameworks; the distinctive object is the bridge metric, output-passing identity failures.

B Threshold Sensitivity

Table 4 reports suite-level false acceptance under three output-suite thresholds. The core conclusion survives stricter thresholds: even at τ=0.90, standard output evaluation remains blind for 4 of 6 tampered conditions.

C Robustness Across Model Pairs

Figure 4 shows blindness rates per (tamper condition, model pair). The pattern persists across two within-family Qwen substitutions and one cross-family Llama-to-Qwen substitution. The Qwen-7B $\rightarrow$ Qwen-1.5B pair is the most behavior-visible (because a $4.5\times$ scale gap is the easiest substitution to detect from outputs) but example-level blindness remains high for every tampered condition; for the cross-family pair, blindness is high across the board.

D Component-Localized Evidence

Table 5 reports component-level mismatch evidence. Each controlled tamper is localized to the intended manifest component, so the identity channel offers per-component evidence rather than a binary alarm. This is also true by construction; it is reported as a sanity check on the implementation.

Figure 4: Blindness by (condition, model pair).

E Reproducibility Artifacts

Aggregate artifacts used in this draft are stored under `paper_runs/aggregate/` (synthetic suite), `paper_runs/gsm8k_anchor/`, and `paper_runs/bfcl_anchor/`. The released package includes `dib_bench_framework.json`, `dib_bench_card.md`, JSON schemas for manifest / identity evidence / scored example, the per-run and aggregate CSVs, and the paper figures included above. The synthetic aggregate contains 7,560 scored examples (9 runs, 3 model pairs, 3 seeds); each public anchor contains 350 scored examples (1 run $\times$ 7 conditions $\times$ 50 cases).

Condition	Model	Prompt	Tooling	Retrieval	Runtime	Postproc.	Any
clean	0.0	0.0	0.0	0.0	0.0	0.0	0.0
model_substitution	100.0	0.0	0.0	0.0	0.0	0.0	100.0
prompt_tamper	0.0	100.0	0.0	0.0	0.0	0.0	100.0
tool_tamper	0.0	0.0	100.0	0.0	0.0	0.0	100.0
retrieval_tamper	0.0	0.0	0.0	100.0	0.0	0.0	100.0
dependency_tamper	0.0	0.0	0.0	0.0	100.0	0.0	100.0
postprocess_tamper	0.0	0.0	0.0	0.0	0.0	100.0	100.0

Table 5: Component-localized identity evidence. Rates are percentages.