SlotGuard: Stop Oversharing Private Local Context in LLM Agent Transcripts

Haocheng Xia [1] **Yongjoo Park** [1]

Abstract

LLM agents can leak privacy (e.g., paths, emails) and credentials (e.g., API keys) as agent observations (e.g., tool outputs, shell logs, and file reads) are appended to provider-bound transcripts. Existing placeholder redaction is brittle: it can miss embedded or cross-turn references, over-redact benign lookalikes, and destroy the structure useful for reasoning. We present SlotGuard, a local transcript boundary that can hide sensitive data while retaining agents' performance. SlotGuard rewrites structural bindings as typed, suffix-aware slots, replaces secrets with format-preserving synthetic values, links cross-turn references with a lightweight session graph, and restores raw values only inside the trusted runtime. On controlled repository-oriented agent transcripts, SlotGuard removes all 20,814 annotated structurally sensitive characters across 9,229 paths and reduces credential leakage to 0.0% across 852 planted values. It remains close to raw-transcript task success across four upstream models, while generic redaction drops to 2.5%. Transcript rewriting takes a median of 14.424 μs per agent turn. The code is publicly accessible at https://github.com/illinoisdata/SlotGuard.

1. Introduction

LLM agents access sensitive information. To perform various tasks (e.g., development, enterprise automation, data analysis), agents employ various tools to interact with file systems, shells, processes, configuration files, and external services on the user's behalf (Chen et al., 2021; Yao et al., 2023; Schick et al., 2023; Xu et al., 2025). Those ordinary tool observations may contain sensitive data such as absolute paths (e.g., `/Users/alice/acme/legal/layoff_bob.pdf`), internal hostnames, branch names, entity names, shell commands, environment variables, and credentials. Since agents append tool outputs to future turns, these records can become part of the model-facing transcript even when the user did not explicitly type them.

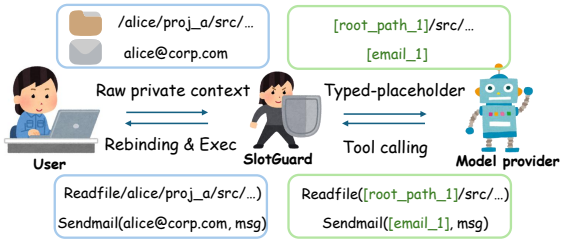

Figure 1. SlotGuard sits between the local agent runtime and the upstream provider. Local bindings are replaced with typed placeholders before transcript submission and rebound to raw values inside the trusted local runtime before execution.

Unfortunately, once sensitive information enters provider-bound transcripts, they may leave the local trust boundary and be exposed to model providers or, in some cases, other users. Consumer AI providers may use user content to improve or train models unless users opt out (OpenAI, 2026; DeepSeek, 2026). Even when model training is disabled, provider-bound transcripts may still be retained, logged, or cached by serving systems, and operational failures can expose cross-user metadata or content: for example, a ChatGPT glitch allowed some users to see titles from other users' chat histories (OpenAI, 2023). Provider-side inference systems also cache prompts and prefixes to improve serving efficiency (Kwon et al., 2023; Zheng et al., 2024; Ye et al., 2024; Jin et al., 2026). We need a local boundary that can protect sensitive data while allowing accurate reasoning.

Motivating example A local audit of agent session logs can reveal a credential as part of ordinary tool output. A shell debugging command prints the full command line of a running experiment with `ps`. The command includes an environment-variable fallback of the form `AZURE_OPENAI_KEY="${AZURE_OPENAI_KEY:-<raw-key>}"`. Once appended to the agent transcript, the credential-shaped value becomes provider-bound context and persists across later turns. This failure mode is easy to miss with assignment-only or vendor-

[1]Siebel School of Computing and Data Science, University of Illinois Urbana-Champaign. Correspondence to: Yongjoo Park <yongjoo@illinois.edu>.

Published at the Second Workshop on Agents in the Wild: Safety, Security, and Beyond (AIWILD) at ICML 2026.

specific scanners: the value appears inside shell parameter expansion, embedded in a process listing, rather than as a standalone `api_key=<value>` assignment. It also illustrates why the boundary must interpose on tool outputs and runtime observations, not only on user-authored prompts.

Our goal is to *hide local secrets and private bindings from model providers while keeping agent transcripts useful*. Our system, called **SlotGuard**, transparently sits between agent runtime and upstream models to perform the following operations. Before transcript submission, SlotGuard turns sensitive local values into stable, typed references that a model can still reason about. Given responses from the model (before passing them to the agent), SlotGuard resolves those references back to the original values inside the trusted runtime. Ideally, the model should still be able to understand that a value is a Python file, a repository path, a database URL, an API token, or a project-specific handle, without their raw original values.

Intelligent privacy protection mechanisms must distinguish different types of sensitive information. LLM agent transcripts can leak sensitive data along two surfaces. ***Structural bindings*** identify a user, organization, or execution environment without necessarily granting access: paths, workspace roots, hostnames, branch names, emails, and sensitive entities A path may be sensitive because it reveals its owner, customer, project, or semantic content. ***Secret values*** are credential-shaped strings that may grant permissions if disclosed, such as API keys, tokens, passwords, DB credentials, and signing material. This distinction is important because these surfaces fail differently: credential scanners can miss semantically sensitive filenames or entity names, while path sanitizers can miss tokens embedded in URLs, shell commands, process listings, or configuration snippets.

Naïve redaction is limited: it hides values by destroying the structure that agents need to act. Models use path syntax, file extensions, relative suffixes, URI grammar, and credential-shaped strings as execution cues. Replacing a path with `PATH` or a secret with `API_KEY` may remove the sensitive value, but it also erases the type, location, and syntactic context needed to plan valid tool calls. The result is a brittle trade-off: strong redaction protects privacy but disrupts workflow compatibility, whereas weak redaction preserves utility but leaves provider-bound transcripts exposed.

To address the issue, SlotGuard establishes a compatibility-preserving local boundary. Specifically, SlotGuard (1) abstracts structural bindings as typed local slots and (2) substitutes secret values with format-preserving synthetic tokens. The model, therefore, sees stable, executable-looking handles rather than raw private values. Across turns, SlotGuard maintains a session-scoped slot table and a lightweight semantic entity graph that conservatively links protected values to embedded occurrences, derived forms, and later keyword-based references. When the model requests tool execution, SlotGuard resolves registered handles only inside the trusted runtime and rejects attempts to spoof, traverse, dereference, or otherwise misuse them. We note that SlotGuard is designed as an enforceable boundary rather than a best-effort detector. It can interpose at the agent harness, where prompts, tool calls, and tool outputs enter the transcript; at a filesystem-facing path layer, where raw paths are virtualized at the source; or at a model-gateway layer such as LiteLLM, which provides a provider-bound last line of defense. Our primary design point is the harness, because it enables both transcript rewriting before provider submission and guarded rebinding before local execution.

We make the following contributions.

1. We identify structural bindings and secret values as two orthogonal privacy surfaces in provider-bound LLM agent transcripts.
2. We introduce SlotGuard, a compatibility-preserving local boundary that combines typed structural abstraction, format-preserving synthetic secret substitution, cross-turn tracking, and guarded local rebinding. While many of these techniques were studied individually (e.g., typed placeholders, format-preserving substitution, information-flow tracking), the key contribution is their systematic integration into a single transcript boundary, guided by the observation that preserving execution-relevant structure—path shape, file extensions, credential syntax—is critical for downstream agent performance.
3. We evaluate SlotGuard as an enforceable boundary, measuring privacy leakage, workflow compatibility, task success, model generalization, and runtime overhead across synthetic privacy corpora, controlled workflow probes, and a TheAgentCompany-derived replay set.

The rest of this paper is organized as follows. We formalize our threat model in §2. Then, we present the SlotGuard design in §3. Finally, we evaluate SlotGuard in §4.

2. Threat Model

We consider an honest-but-curious upstream provider: inference is correct, but provider-bound prompts, tool outputs, and transcripts may be retained. The local runtime, tools, files, and SlotGuard boundary are trusted. SlotGuard protects the local-to-provider boundary, primarily inside the agent harness before prompts, tool calls, shell outputs, process listings, and file contents are appended to the transcript. It protects two value classes: *structural bindings*, such as paths, workspace roots, branches, hostnames, emails, and sensitive entity names that identify a user, organization, project, or environment; and *secret values*, such as API keys, tokens, passwords, database credentials, and signing material. The goal is compatibility-preserving privacy: re-

move raw private values while preserving workflow signals needed for valid tool use, including path shape, file type, stable handles, parseable commands, and format-valid credential surrogates. We measure literal leakage, categorical leakage (i.e., credential type revealed through replacement-token labels[1]), reconstructive leakage, workflow compatibility, and task success.

3. SlotGuard Design

Boundary and slot identification SlotGuard is a local boundary layer between the agent runtime and the upstream provider. Its primary deployment point is the agent harness, before prompts, tool calls, tool outputs, shell stdout/stderr, process listings, and file contents are appended to provider-bound transcripts. This placement covers failure modes such as credentials printed by `ps` or shell debugging commands. SlotGuard can also be deployed at a filesystem-facing path layer, where raw paths are virtualized at the source, or at a model-gateway layer such as LiteLLM, which provides a provider-bound last line of defense across model backends.

SlotGuard identifies protected spans using a two-tier detector. The synchronous tier is a heuristic engine for high-precision cases: known workspace roots, path parsers, emails, hostnames, branch names, environment-variable assignments, shell parameter expansions, credential flags, authorization headers, URI credentials, and provider-specific token patterns. The asynchronous tier optionally uses a small local model for semantic cases that are difficult to encode as rules, such as sensitive filenames, entity names, customer names, or project names. The local model does not rewrite transcripts directly. It proposes a line number, verbatim span, and slot type; SlotGuard admits the proposal only if the span appears in the local input and the slot type is allowed for the current task.

Compatibility-preserving rewriting SlotGuard follows a compatibility-preserving privacy principle: remove raw private values from provider-bound transcripts while preserving the workflow-level signals agents rely on in unprotected execution. Both structural bindings and secrets are stored in a session-scoped slot table $T : V \to H$, where V is the set of protected raw values and H is the upstream-visible handle space.

For structural bindings, deterministic typed slots are used:

$$h(v) = \mathrm{HMAC}_{k_s}(\tau(v) \parallel v),$$

where k_s is a per-session secret and $\tau(v)$ is the slot type. This makes handles stable within a session but unlinkable across sessions. Structural slots include scope roots such as `[REPO_ROOT_1]`, identity handles such as `[EMAIL_1]`, and sensitive-content handles such as `[SENSITIVE_DOC_1]`. For paths, SlotGuard performs suffix-aware abstraction: it replaces the longest trusted root, suppresses sensitive middle components, and preserves a bounded task-relevant suffix: `/Users/alice/acme/legal/layoff_bob.pdf` ⇒ `[REPO_ROOT_1]/legal/[SENSITIVE_DOC_1].pdf`. The result remains path-shaped and preserves directory role and file type, while removing user, organization, and sensitive document identifiers.

For secret values, SlotGuard uses format-preserving synthetic substitution (FPS) rather than generic labels. Generic replacements such as `API_KEY` can make the model infer that a credential is missing or produce commands that no longer match the surrounding grammar. FPS instead samples a fresh synthetic value from the secret's format class: $h(v) \sim \mathrm{Uniform}(F_{\tau(v)})$, where $F_{\tau(v)}$ is the set of valid strings for the credential format. The synthetic token is stored in T and reused for later occurrences. Thus a database URI remains parseable, a command-line credential remains credential-shaped, and an environment fallback remains syntactically configured, while the synthetic value contains no characters derived from the original secret. Conditioned on the format class, $I(\mathrm{FPS}(v); v \mid \tau(v)) = 0$.

SlotGuard maintains a session-scoped semantic entity graph (SEG) to extend protection beyond exact string matches. Nodes are protected values or their surrogates; edges encode coreference, embedding, derivation, and indirect reference. The SEG extends protection beyond exact string matches by recording conservative links between protected values and observed variants, including embedded occurrences, simple derived forms, and later keyword-based references. It is not intended to provide full semantic understanding; instead, it provides a lightweight session memory for common agent-transcript leakage patterns. In its current form, the SEG handles keyword- and pattern-based links reliably but does not attempt harder indirect references such as paraphrased coreferences or implicit reasoning chains. We view the current SEG as a conservative first step; extending it with richer coreference resolution or lightweight local models is a natural direction for future work.

Guarded rebinding The upstream model only sees typed slots and FPS tokens. Raw values are restored only inside the trusted runtime and only for validated local execution. For structural bindings, SlotGuard checks that a handle exists in the session table, has the expected type, and rebinds within the allowed scope; it rejects unknown handles, spoofed segments, out-of-scope paths, and shell metacharacter injection. For secrets, an FPS token is resolved to the raw credential only when a local tool call requires it, and the raw value is injected into the process, environment variable, request field, or connection string without being written back to the provider-bound transcript. Fabricated

[1] A placeholder `__API_KEY_abc__` discloses that the redacted value is an API key even when the value itself is withheld

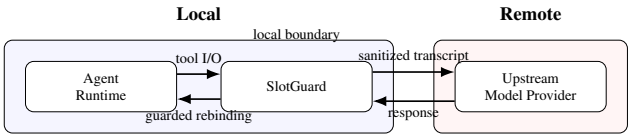

Figure 2. SlotGuard runs locally between the agent runtime and the upstream model provider. Sanitized transcripts go upstream; raw values are rebound only inside the trusted runtime.

credential-shaped strings that are not registered in the slot table are treated as untrusted literals.

4. Evaluation

We evaluate SlotGuard as a local transcript boundary: it should remove provider-visible bindings while preserving the workflow cues needed for local agent operations.[2] We study four questions: structural leakage (RQ1), credential leakage (RQ2), workflow compatibility (RQ3), and hot-path practicality (RQ4).

Setup We use three workloads: two synthetic privacy corpora (a 30-session structural corpus with 9,229 paths and a 200-session credential corpus with 852 planted values), synthetic workflow probes (a 12-task local probe and a 200-task multi-model probe), and a derived replay set, TAC-lite, from The Agent Company with 33 task families, 40 grounded handles, and 160 replay tasks (Xu et al., 2025). We compare *Raw*, *Generic redaction*, a restore-aware *VibeGuard-style typed-placeholder baseline* derived from `default.vgrules`[3], and *Full SlotGuard*. Component ablations and expanded setup details are deferred to §C.

4.1. RQ1–RQ2: Does SlotGuard remove private values?

Table 1 summarizes the core privacy result. Across the structural corpus, raw transcripts expose 20,814 annotated sensitive characters, while SlotGuard removes all provider-visible structural leakage. The rewrite still preserves execution cues: 93.6% of rewritten paths keep the original extension and 100.0% keep at least one safe suffix component. Across the credential corpus, typed placeholders and FPS-only (format-preserving synthetic substitution in §3) still leak 23.5% of planted credentials because split and embedded exposures remain reconstructible, whereas the full version with SEG (semantic entity graph in §3) leaks 0/200 sessions and 0/852 planted values. Detailed breakdowns, including format-validity results, appear in §C.

4.2. RQ3: Does SlotGuard retain performance?

The local probe shows the basic pattern: generic redaction collapses to 16.7% success, VibeGuard-style (inkdust2021, 2026b) placeholders recover some signal at 66.7%, and full SlotGuard reaches 83.3% with no invalid handles. We keep the local probe and ablations in §C.2. The larger 200-task probe across four upstream models shows the same ordering (Table 2). Generic redaction collapses to 2.5% task success for every model. VibeGuard-style typed placeholders are better than opaque replacement, but still leave a large gap, ranging from 20.5% to 38.5%. In contrast, full SlotGuard stays close to each model's raw baseline: it matches the raw score for Kimi and Llama, remains within 2.0 points for `gpt-5.4`, and within 10.5 points for DeepSeek. The main effect is therefore not merely that "typed placeholders help"; it is that path-shaped and format-preserving rewrites preserve far more execution-relevant structure than proxy-style placeholders.

We also evaluate the replay set, TAC-lite. TAC-lite extracts evaluator-grounded repo-relative handles and instantiates local replay tasks without TAC's Dockerized service stack. The same ordering reappears: generic redaction falls to 0.0% across all four models, so we omit it. VibeGuard-style placeholders reach only 26.9–36.9%, and full SlotGuard matches the raw score for `gpt-5.4`, Kimi, and Llama while remaining within 7.5 points on DeepSeek. Table 3 gives the extraction procedure and full TAC-lite table.

While our evaluation employs diverse data such as (1) synthetic privacy corpora, (2) controlled workflow probes, and (3) a replay-style benchmark derived from TheAgentCompany, they may not fully capture the diversity of real-world enterprise agent traces. Validation on production-like environments with naturally occurring sensitive data remains an important direction for future work.

Table 1. Key privacy results. Structural rows compare against raw transcripts; credential rows compare against the strongest baseline.

Metric	Baseline	SlotGuard
Structural privacy / path compatibility		
Sensitive chars visible upstream	20,814	0
Structural leakage	100.0%	0.0%
Paths retaining file extension	–	93.6%
Paths retaining safe suffix	–	100.0%
Credential leakage / secret exposure		
Credential session leak	100.0%	0.0%
Credential leak rate	23.5%	0.0%

Table 2. Model generalization on 200 shared workflow tasks.

Model	Raw	Generic	VG-style	SlotGuard
DeepSeek-V3.2	100.0%	2.5%	38.5%	89.5%
gpt-5.4	100.0%	2.5%	20.5%	98.0%
Kimi-K2.6	100.0%	2.5%	21.5%	100.0%
Llama-3.3-70B	93.5%	2.5%	31.5%	93.5%

[2] Artifacts: https://anonymous.4open.science/r/SlotGuard-0D2D

[3] https://raw.githubusercontent.com/inkdust2021/vgrules/refs/heads/main/default.vgrules

Table 3. TAC-lite replay benchmark.

Model	Raw	VG-style	SlotGuard
DeepSeek-V3.2	95.0%	34.4%	87.5%
gpt-5.4	92.5%	36.9%	92.5%
Kimi-K2.6	95.0%	35.6%	95.0%
Llama-3.3-70B	90.0%	26.9%	90.0%

4.3. RQ4: Is SlotGuard practical on the hot path?

The core rewrite is fast enough to incur only small overhead in practice. Rewriting a path takes 1.690 μs median, mapping it back takes 0.398 μs, and sanitizing a full agent turn takes 14.424 μs; even a 6 KB turn stays below 100 μs median. The optional local semantic detector is much slower (742 ms warm median), so SlotGuard does not rely on it for the fast path. Instead, the detector only suggests candidate sensitive spans, and SlotGuard verifies each suggestion before applying a rewrite. On a held-out sensitive-filename benchmark, this optional detector improves F_1 from 0.571 to 0.909. SlotGuard is also robust: all 9,229 rewritten paths round-trip correctly, all 64 invalid rebinding attempts are rejected. Additional details are in §C.

5. Conclusion

SlotGuard is a local transcript boundary for LLM agents. It removes structural bindings with typed, suffix-aware slots and protects credentials with format-preserving synthetic. Across our workloads, SlotGuard eliminates annotated structural leakage and credential leakage while remaining close to raw-transcript task success on both synthetic probes and TheAgentCompany-derived replay tasks. SlotGuard is not full anonymization, but a step toward local capability boundaries that mediate both transcript privacy and tool access.

Several directions remain open. First, our evaluation uses synthetic and replay-style workloads with planted ground truth; validating SlotGuard on production agent traces with naturally occurring sensitive data would strengthen the practical evaluation. A small real-world deployment case study is a natural next step. Second, we do not measure adversarial reconstruction by a capable attacker who actively attempts to recover protected values from residual context; such an evaluation would complement the current literal- and reconstructive-leakage proxies. Third, the semantic entity graph is an early implementation that tracks entities through keyword- and pattern-based links; standalone verification on harder indirect-reference benchmarks and extending it with richer coreference resolution would make the component more robust. Finally, comparison against production secret scanners, enterprise DLP systems, or LLM-specific guardrail products would better situate SlotGuard's detection coverage relative to deployed baselines.

Impact Statement

This work introduces a privacy boundary for tool-using LLM agents. Its intended benefit is to reduce unnecessary exposure of local identifiers and credentials to upstream model providers while preserving agent utility. SlotGuard is not a full anonymization system: it cannot prevent semantic inference from non-literal context, protect a compromised local runtime, or stop prompt-injection-driven exfiltration. The current evaluation does not measure adversarial reconstruction by a capable attacker who actively attempts to recover protected values from residual transcript context; such an evaluation would further stress-test the boundary's practical guarantees. It should therefore be deployed as one layer in a broader privacy posture. We do not identify a specific dual-use risk; the mechanism reduces provider-bound information flow and runs under user control.

Acknowledgements

This work was supported in part by the National Science Foundation under grants #2312561 and #2440498. This work used Delta and DeltaAI at the National Center for Supercomputing Applications (NCSA) through allocation CIS240661 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by U.S. National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296. This work was also supported by the IBM-Illinois Discovery Accelerator Institute.

References

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

Clause, J., Li, W., and Orso, A. Dytan: a generic dynamic taint analysis framework. In *Proceedings of the 2007 international symposium on Software testing and analysis*, pp. 196–206, 2007.

DeepSeek. Deepseek privacy policy. https://cdn.deepseek.com/policies/en-US/deepseek-privacy-policy.html, 2026.

inkdust2021. opencode-vibeguard: Vibeguard for opencode. https://github.com/inkdust2021/opencode-vibeguard, 2026a. GitHub repository, accessed 2026-05-07.

inkdust2021. VibeGuard: Uses just 1% memory while protecting 99% of your personal privacy. https://github.com/inkdust2021/VibeGuard, 2026b. GitHub repository, accessed 2026-05-07.

Jin, C., Zhang, Z., Jiang, X., Liu, F., Liu, S., Liu, X., and Jin, X. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *ACM Trans. Comput. Syst.*, 44 (1):2:1–2:27, 2026. doi: 10.1145/3768628. URL https://doi.org/10.1145/3768628.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In Flinn, J., Seltzer, M. I., Druschel, P., Kaufmann, A., and Mace, J. (eds.), *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pp. 611–626. ACM, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Microsoft. Microsoft Presidio: Data protection and de-identification sdk. https://microsoft.github.io/presidio/, 2026. Accessed 2026-05-07.

Nissenbaum, H. Privacy as contextual integrity. *Wash. L. Rev.*, 79:119, 2004.

OpenAI. March 20 chatgpt outage: Here's what happened. https://openai.com/index/march-20-chatgpt-outage/, 2023.

OpenAI. Chatgpt privacy settings. https://openai.com/consumer-privacy/, 2026.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., and Scialom, T. Toolformer: Language models can teach themselves to use tools. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *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*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Xu, F. F., Song, Y., Li, B., Tang, Y., Jain, K., Bao, M., Wang, Z. Z., Zhou, X., Guo, Z., Cao, M., Yang, M., Lu, H. Y., Martin, A., Su, Z., Maben, L., Mehta, R., Chi, W., Jang, L., Xie, Y., Zhou, S., and Neubig, G. Theagentcompany: Benchmarking llm agents on consequential real world tasks, 2025. URL https://arxiv.org/abs/2412.14161.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Ye, L., Tao, Z., Huang, Y., and Li, Y. Chunkattention: Efficient self-attention with prefix-aware KV cache and two-phase partition. In Ku, L., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 11608–11620. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.623. URL https://doi.org/10.18653/v1/2024.acl-long.623.

Zheng, L., Yin, L., Xie, Z., Sun, C., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., Barrett, C. W., and Sheng, Y. Sglang: Efficient execution of structured language model programs. In Globersons, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J. M., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

A. Related Work

Plugin-level redaction Open-VibeGuard (inkdust2021, 2026a) applies harness-level redaction rules like `regexSSN(?:^|\D)(\d{3}-\d{2}-\d{4})(?:$|\D)` and replaces matched values with stable category-hash placeholders such as `__VG_SSN_xxxxxxxxxxxx__`. This is a practical, lightweight defense for matched spans, but rule-based placeholder redaction is local and structurally brittle. It can over-redact benign lookalikes that share the same surface pattern, while missing sensitive values embedded in shell expansions, URIs, process listings, and multi-line configuration snippets, derived forms, or later cross-turn references. In agent workflows, placeholders may also be mistaken for missing values or copied into commands, URIs, and configuration snippets in forms that no longer match the expected grammar. SlotGuard addresses these gaps with a session-scoped slot table and SEG, suffix-aware structural slots, format-preserving synthetic secrets, and guarded local rebinding.

Network-level proxies Network-boundary redaction systems interpose below the application, for example as an HTTPS proxy with regex or NER-based rewriting (inkdust2021, 2026b). This placement can cover multiple clients without agent-specific integration, but it only sees serialized provider requests. It does not naturally expose agent structure such as tool calls, tool outputs, shell observations, local scopes, or execution intent. As a result, network proxies are well suited as last-line defenses, but cannot easily perform scope-aware path validation or rebind protected handles back into trusted local tool execution. They may also require trusted-CA installation, changing the user's network trust assumptions.

PII detection and anonymization General-purpose PII systems such as Microsoft Presidio (Microsoft, 2026) detect entities such as names, emails, phone numbers, and identifiers, and replace them with labels or anonymized values. These systems target text anonymization, whereas agent transcripts require compatibility-preserving rewriting. A replacement must hide the raw value while preserving the cues models rely on for tool use: path shape, file extensions, relative suffixes, URI structure, header grammar, and credential-shaped syntax. SlotGuard therefore treats transcript rewriting as part of an execution boundary, not only as entity anonymization.

Information flow control SlotGuard is conceptually related to dynamic taint analysis and information-flow control, which track sensitive values as they propagate through computation (Clause et al., 2007). Our protected sink is the provider-bound transcript of an LLM agent. The semantic entity graph plays a taint-like role by linking raw values, rewritten handles, embedded occurrences, derived forms, and later indirect references. The policy goal also follows contextual integrity (Nissenbaum, 2004): local values may be appropriate inside the trusted runtime, but inappropriate once copied into a remote model-provider transcript.

B. Motivating Transcript Leakage Example

During a local audit of agent session logs, we found that credentials can enter provider-bound transcripts through ordinary tool output rather than through user-authored prompts. One example came from a shell debugging command that printed a process listing. The resulting transcript contained a command line with an environment-variable fallback:

Example of session snippet

```
AZURE_OPENAI_KEY="${AZURE_OPENAI_KEY:-<AZURE_OPENAI_KEY_VALUE>}" uv run python experiment.py --data_path ... --resume
```

This pattern is easy to miss with vendor-specific or assignment-only secret scanners. The secret is not presented as a standalone API key, nor as a simple assignment of the form `api_key=<value>`. Instead, it appears inside shell parameter expansion, embedded in a process listing, and then becomes part of the agent transcript when the tool output is appended to the conversation.

SlotGuard treats shell commands, process listings, environment assignments, and parameter-expansion fallbacks as credential-bearing contexts. In this case, the raw fallback value would be replaced by a format-preserving synthetic surrogate before transcript submission:

Processed by SlotGuard

```
AZURE_OPENAI_KEY="${AZURE_OPENAI_KEY:-<FPS_AZURE_OPENAI_KEY>}" uv run python experiment.py --data_path ... --resume
```

The rewritten command remains syntactically valid and continues to signal that a credential is configured, but the provider-bound transcript contains no value-level information about the original credential. During validated local execution, the FPS

Table 4. Structural privacy and path-compatibility results.

Metric	Baseline	SlotGuard
Sensitive chars visible upstream	20,814	0
Structural char leakage	100.0%	0.0%
Paths retaining file extension	–	93.6%
Paths retaining ≥ 1 safe suffix component	–	100.0%
Fully opaque paths	–	0.0%

Table 5. Credential leakage on 200 sessions / 852 planted values.

Method	Session leak	Credential leak	Categorical leak
No protection	100.0%	100.0%	0.0%
Typed placeholder baseline	100.0%	23.5%	100.0%
FPS only	100.0%	23.5%	0.0%
SlotGuard (FPS+SEG)	0.0%	0.0%	0.0%

token is resolved through the session slot table inside the trusted runtime.

C. Additional Evaluation Details

Expanded setup The structural corpus contains 30 randomized repository-like sessions with 9,229 paths and transcript fragments containing user-home prefixes, workspace roots, repository roots, emails, internal hosts, and sensitive document names. The credential corpus contains 200 sessions with 852 planted credentials spanning direct, embedded, split, derived, and cross-turn contexts. The workflow suite contains a 12-task local probe against `qwen2.5-coder:1.5b` via an OpenAI-compatible Ollama endpoint and a 200-task shared probe across four Azure-hosted models. The VibeGuard-style baseline is derived from `vgrules.md` categories such as `EMAIL`, `API_KEY`, `BEARER_TOKEN`, `JWT`, `AWS_KEY`, and `PRIVATE_KEY`, and rewrites matches to restore-aware placeholders of the form `__VG_<CATEGORY>_<hash>__` (inkdust2021, 2026b;a).

C.1. Detailed Privacy Results

The split-exposure cases highlight the main limitation of non-graph methods: all 200 split cases leak under both typed placeholders and FPS-only, whereas FPS+SEG eliminates this leakage. We also evaluate whether generated surrogates preserve the grammar of their original contexts. Across API keys, OAuth-style tokens, private-key markers, and internal hostnames, parse success ranges from 83.3% to 100.0%; the remaining failures are concentrated in deliberately awkward PII and multiline edge cases, rather than in the credential classes used by the workflow probes.

We emphasize that the current SEG is still an early implementation. It tracks protected entities through keyword- and pattern-based links rather than full semantic understanding, so it should be viewed as a conservative first step toward semantic transcript memory rather than a complete solution to indirect leakage. A dedicated standalone evaluation of the SEG—measuring recall on diverse indirect-reference patterns such as paraphrased mentions, implicit coreference, and multi-hop derivation chains—would help quantify its coverage gaps and guide improvements. We consider this an important direction for strengthening the component.

C.2. Workflow Ablations

The ablations separate the two design contributions. Removing FPS preserves some path utility but weakens secret handling, while removing suffix preservation keeps the privacy guarantee but makes structural responses much less execution-ready. SlotGuard needs both components to preserve the workflow signal that raw transcripts provide.

C.3. TAC-lite Real-Task Replay

To reduce dependence on fully synthetic task wording, we derive a lightweight replay benchmark from The Agent Company. We do *not* run the original TAC Dockerized multi-service stack. Instead, we scan `task.md` and evaluator code for grounded local file handles (e.g., `open(...)` targets, evaluator working directories, and destination paths), retain repo-centric tasks that resolve to local workspace files, and instantiate each grounded handle under four randomized local workspace roots. This yields 33 source task families, 40 grounded handles, and 160 replay tasks. Because every item is structural, the main metrics are exact handle selection, execution-ready rate, tool-valid rate, and invalid-handle rate.

Table 6. Workflow compatibility on the local 12-task probe with ablations.

Condition	Task success	Exec.-ready	Invalid handles	Avg. latency
Raw	100.0%	100.0%	0.0%	1551 ms
Generic redaction	16.7%	0.0%	83.3%	637 ms
VibeGuard-style placeholder	66.7%	33.3%	0.0%	724 ms
No FPS	50.0%	66.7%	0.0%	718 ms
No suffix preservation	50.0%	0.0%	0.0%	805 ms
Full SlotGuard	83.3%	66.7%	0.0%	719 ms

Table 7. TAC-lite four-model comparison on 160 evaluator-grounded replay tasks derived from The Agent Company.

Model	Raw	Generic	VG-style	Full SlotGuard
DeepSeek-V3.2	95.0%	0.0%	34.4%	87.5%
gpt-5.4	92.5%	0.0%	36.9%	92.5%
Kimi-K2.6	95.0%	0.0%	35.6%	95.0%
Llama-3.3-70B	90.0%	0.0%	26.9%	90.0%

The TAC-lite results mirror the synthetic workflow probe. Generic redaction collapses to 0.0% task success for every model because the returned handles are typically unrebindable or structurally invalid. VibeGuard-style placeholders recover some semantics, but only reach 26.9–36.9% exact handle success. Full SlotGuard matches the raw score on `gpt-5.4`, Kimi, and Llama, and remains within 7.5 points on DeepSeek. Tool-valid rates for full SlotGuard remain in the 92.5–97.5% range, close to raw, whereas generic redaction induces invalid-handle rates between 70.0% and 100.0%.

C.4. Why Proxy-Style Placeholders Can Be Slower

The occasional latency penalty of VibeGuard-style placeholders is not caused by extra local rewrite work. SlotGuard's deterministic operations are microsecond-scale regardless of the transcript condition. The difference appears during model inference: placeholders such as `__VG_API_KEY_<hash>__` and `__VG_PATH_<hash>__` are high-entropy tokens with weaker structural cues than SlotGuard's path-shaped and format-preserving rewrites, so some models take longer to reason over them and more often require response retries. Our end-to-end latency accounting includes request backoff and JSON-parse retries, which amplifies this gap on the affected models.

Table 8 shows that this is a model interaction rather than a universal effect. On `Kimi-K2.6`, proxy-style placeholders are dramatically slower than full SlotGuard because the model struggles with the opaque handles and triggers many more retries. On `gpt-5.4`, the two conditions are nearly identical, and on DeepSeek the ordering is reversed. The important point is that the slowdown, when it appears, reflects model friction with the placeholder representation rather than extra local processing cost. The same pattern reappears on TAC-lite: on `Kimi-K2.6`, VG-style placeholders average 10.97 s per task versus 3.16 s for full SlotGuard, while on `gpt-5.4` the two conditions remain close (492 ms vs. 459 ms).

C.5. Hot-Path and Semantic-Layer Details

The core deterministic path remains negligible relative to provider inference: `abstract_path` has a median cost of 1.690 μs, `rebind_path` 0.398 μs, and `abstract_text/agent_turn` 14.424 μs. A larger 6 KB transcript-sized turn remains under 100 μs median (96.803 μs). On a 30-example held-out filename probe, rules alone achieve precision $= 1.00$, recall $= 0.40$, and $F_1 = 0.571$. Adding a small local CPU-runnable model raises recall to 1.00 and $F_1 = 0.909$. The semantic pass has a cold first-call latency of 919 ms, a warm median of 742 ms, and a p95 of 929 ms, so we keep it optional and advisory under a verify-then-replace gate.

C.6. Hard and Adversarial Cases

Across all 9,229 rewritten paths, guarded rebinding succeeds on every round-trip, with zero mismatches and zero unexpected errors. On adversarial inputs, SlotGuard rejects all 64 crafted invalid rebindings, including traversal attempts, spoofed placeholders, and shell-metacharacter injections. These checks complement the workflow results: the same slot table that preserves compatibility in the benign case also enforces fail-closed behavior under malformed or adversarial local inputs. The lone hard case-suite miss is a root-owned absolute path `/root/.ssh/id_rsa`, which exposes a remaining out-of-scope home-prefix edge case in the current prototype rather than a rebinding failure.

Table 8. Average end-to-end latency on the 200-task model-generalization probe.

Model	VG-style	Full SlotGuard
DeepSeek-V3.2	1283 ms	4701 ms
gpt-5.4	684 ms	701 ms
Kimi-K2.6	26174 ms	5774 ms
Llama-3.3-70B	570 ms	856 ms

Table 9. Hard and adversarial cases. "Rejected" means input refused with explicit error; "sanitized" means input neutralized. One current out-of-scope absolute-path edge case remains.

Category	N	Outcome
.. traversal in rebinding	6	6 rejected
Spoofed placeholder not in slot table	4	4 rejected
Shell metacharacters (; | & $ ‘)	5	5 rejected
Out-of-scope absolute paths	3	2 rejected
Mixed encoding / non-UTF-8 fragments	2	2 sanitized
Pathological size (path >10 KB)	2	2 sanitized
Bracket syntax in legitimate filenames	2	2 sanitized
Total	**24**	**23/24 contained**