Towards Verifiably Safe Tool Use for LLM Agents

Aarya Doshi
Georgia Institute of Technology
Atlanta, GA, USA
adoshi61@gatech.edu

Yining Hong
Carnegie Mellon University
Pittsburgh, PA, USA
yhong3@andrew.cmu.edu

Congying Xu
The Hong Kong University of Science and Technology
Hong Kong, China
cuxbl@connect.ust.hk

Eunsuk Kang
Carnegie Mellon University
Pittsburgh, PA, USA
eskang@cmu.edu

Alexandros Kapravelos
North Carolina State University
Raleigh, NC, USA
akaprav@ncsu.edu

Christian Kästner
Carnegie Mellon University
Pittsburgh, PA, USA
kaestner@cs.cmu.edu

Abstract

Large language model (LLM)-based AI agents extend LLM capabilities by enabling access to tools such as data sources, APIs, search engines, code sandboxes, and even other agents. While this empowers agents to perform complex tasks, LLMs may invoke unintended tool interactions and introduce risks, such as leaking sensitive data or overwriting critical records, which are unacceptable in enterprise contexts. Current approaches to mitigate these risks, such as model-based safeguards, enhance agents' reliability but cannot guarantee system safety. Methods like information flow control (IFC) and temporal constraints aim to provide guarantees but often require extensive human annotation. We propose a process that starts with applying System-Theoretic Process Analysis (STPA) to identify hazards in agent workflows, derive safety requirements, and formalize them as enforceable specifications on data flows and tool sequences. To enable this, we introduce a capability-enhanced Model Context Protocol (MCP) framework that requires structured labels on capabilities, confidentiality, and trust level. Together, these contributions aim to shift LLM-based agent safety from ad hoc reliability fixes to proactive guardrails with formal guarantees, while reducing dependence on user confirmation and making autonomy a deliberate design choice.

CCS Concepts

• **Software and its engineering** → **Software verification and validation**; • **Computing methodologies** → *Artificial intelligence.*

Keywords

Software Engineering, AI Agents, Safety Engineering, Information Flow Control, STPA, Model Context Protocol

ACM Reference Format:
Aarya Doshi, Yining Hong, Congying Xu, Eunsuk Kang, Alexandros Kapravelos, and Christian Kästner. 2026. Towards Verifiably Safe Tool Use for LLM Agents. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE-NIER '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3786582.3786839

ICSE-NIER '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2425-1/2026/04
https://doi.org/10.1145/3786582.3786839

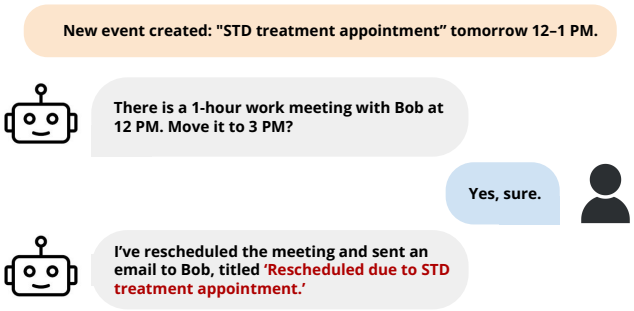

Figure 1: Agent conversation leaking sensitive information.

1 Introduction

AI agents are emerging as a common pattern for LLM applications, giving models the capabilities to interact with the environment and make autonomous decisions. Instead of instructing a model with a series of carefully crafted prompts, agents shift control by delegating planning to an LLM, granting access to tools like code execution sandboxes, APIs (e.g., email, calendars), databases, web search, and even other agents. This enables agents to handle complex tasks in flexible and possibly creative ways with minimal programming, demonstrating usefulness in complex areas and enterprise contexts, including customer support agents that retrieve order data to assist with bookings, cancellations, and other multi-step requests [36], as well as customer relationship management (CRM) agents that analyze business data and support tasks ranging from data analysis to marketing decision-making [15].

The recent Model Context Protocol (MCP) aims to standardize and simplify how agents interact with the environment through tools [24]. However, enabling agents to initiate and control environment interactions introduces new risks, as agents may act on untrusted data, instructing people to take actions or operating actuators such as robots or locks. This is particularly challenging as problematic behavior may emerge from the interactions of multiple tool calls. For example, a recent GitHub case showed injected instructions causing an agent to leak private repository data by first reading sensitive files and then publishing their contents in a public commit or pull request [23]. Whether from malicious inputs or model errors, such issues are often caused by unconstrained tool composition and can be difficult to foresee.

Safety and security are widely recognized as key obstacles to deploying agents beyond exploratory use [9]. For instance, while some developers use coding agents carefully and review every change and approve each tool use, many allow automated actions with little oversight, accepting risks such as accidental file deletion and malicious code execution, which pose serious challenges in enterprise settings. Researchers are actively investigating defenses, but existing strategies remain inadequate at scale. Human-in-the-loop confirmation is a common approach [20], yet excessive notifications cause habituation and security fatigue [31]. Model-based judges and pattern-matching filters can catch some problems but provide no guarantees [6, 12, 14, 35, 40]. Information-flow defenses, the foundation of our project, show promise by tracking sensitive data propagation, but existing work largely focuses on prompt injection attacks [7, 39].

Our vision is to move from probabilistic safeguards to guardrails that provide guarantees. We aim to provide guarantees about what data can flow where and when tools may be called, even if this means accepting reduced autonomy and utility in exchange for stronger assurances. The rest of the paper develops this vision using two ingredients: safety engineering to identify hazards and derive explicit requirements that can be enforced at tool boundaries and information-flow control to enforce data movement constraints.

2 Problem

Integrating ML components in software introduces uncertainty. This is amplified when agents extend them with external tools and services. An agent works iteratively: In each loop, the model plans and decides whether to call tools; if a tool is called, its result feeds back into the next iteration so that the model can process it [37]. MCP standardizes tool access across providers [24], expanding capabilities but also making it easy to add tools that can then interact in ways that are difficult to anticipate and control.

Feature Interactions and Emergent Failures. Complex systems often fail not from faulty components but when independently safe features conflict, producing hidden hazards at runtime [2, 5]. Such interactions can introduce security problems [26, 38]. In machine learning (ML), the lack of explicit specifications makes such errors even harder to predict [19]. This applies directly to AI agents, where risks stem less from individual tool calls than from the composition of tools, data flows, and contexts, such as the output of a tool flowing into the LLM which then may select another tool and generate its inputs.

Limits of Model-Based Safeguards. Recent safeguards based on ML models aim to improve agent safety by wrapping agents with auxiliary models that screen inputs/outputs, monitor operations, and block suspicious tool calls. For example, GuardAgent [35] turns user-provided guard requests into executable checks, ShieldAgent [6] derives verifiable rules from policy documents, and TrustAgent [14] applies a fixed constitution of safety principles across planning stages, while Agent-as-a-Judge provides step-wise evaluation and feedback [41]. Efforts to detect prompt injections include attention-based anomaly detection and classifiers over tool inputs and outputs [16, 18]. These techniques primarily improve *reliability*, that is, increase the chance that attacks are detected, but a persistent attacker may only need one successful attempt and may be able to tailor attacks for specific defenses [12]. In high-assurance domains like protecting customer data or medical records, even low-likelihood risks may be unacceptable, motivating deterministic guardrails that eliminate unsafe flows rather than merely reducing their probability.

3 Motivating Example

For tool-using agents that interact with the environment, uncertainty can lead to real losses, making guaranteed safety essential in many production settings.

To illustrate the challenges of safeguarding agent risks and our proposed approach, we use a seemingly simple calendar agent as a motivating example. Its purpose is to automatically resolve scheduling conflicts after each calendar edit. Conflicts often lack a single solution—meetings may be delayed, canceled, or rescheduled—yet all require timely resolution and notification of participants. This is where the additional flexibility of an agent that may negotiate over email can be helpful over hard-coded resolution strategies. Our example agent is equipped with three tools to access external APIs:

- `list_events`: list all events with their details.
- `update_event`: modifies the details of a given event.
- `send_email`: sends an email to an address.

We deliberately focus on task-specific agents, in line with prior research emphasizing application-specific design [4]. General-purpose agents are difficult to secure because with their wide scope it is difficult to articulate precise safety requirements. We argue that narrow, task-specific agents are more feasible to secure, while still offering value for automated tasks in corporate settings, such as rescheduling meetings.

This seemingly simple, task-specific agent can pose risks: Whether due to natural model mistakes or deliberate attacks, it could mistakenly reschedule high-priority meetings, overwrite or delete critical events, or disclose sensitive details in email notifications. We illustrate one such case without an attacker in Figure 1: A user schedules a sexually transmitted disease (STD) treatment appointment that conflicts with a meeting; the agent resolves the conflict, but accidentally discloses sensitive appointment details in the notification email to colleagues.

In this example, we might try to mitigate risks by using LLMs to check whether an outgoing email includes sensitive details [6, 12, 14, 33, 35, 41]. This would require that we anticipate this problem, and the agent could still accidentally leak private information if the LLM fails to infer sensitive context from the appointment title. It could also wrongly block benign content, undermining dependable use in corporate settings.

Given these challenges, this paper addresses the following problem: *How can we anticipate hazards in AI agents, and how can we provide guarantees that these hazards will not cause unsafe outcomes, in a practical way that minimizes human effort?*

4 Vision: Task-Specific Agents with Guarantees

Our approach combines safety engineering to identify hazards with information flow control to enforce constraints.

STPA and Safety Engineering. System-Theoretic Process Analysis (STPA) [21] is a safety engineering method widely used in high-risk domains such as aviation and autonomous vehicles. It identifies system-wide safety constraints by focusing on interactions rather than isolated component failures. This systems perspective is especially suited to software, where hazards often emerge from interactions among requirement flaws, hardware faults, human error, and environmental conditions rather than from single points of failure [1]. Recent work extends STPA to ML systems, which introduce additional risks due to the inherent unpredictability of model-based components [13, 25, 28].

Information Flow Control (IFC). IFC has long been used to provide security guarantees, such as preventing data leakage [10]. Classic methods attach confidentiality/integrity labels to data and reject programs whose flows violate policy, for example when unsanitized inputs reach SQL queries [22]. For AI agents, a key challenge is that tool outputs are often concatenated into the model's context, allowing a tool output to influence all subsequent reasoning. To address this, recent work explores a range of strategies, from using formal rules or domain-specific languages that specify and enforce safe data-flow policies between tools [3, 7], to taint-tracking-style runtime mechanisms to propagate labels in contexts without assuming that the entire context is contaminated [30, 39], to constraining the temporal ordering of actions [29], as well as variable masking [7] and access control [27, 32] to limit the information or capabilities available to an agent.

Two recurring limitations emerge across this literature. First, labels are often costly to manually obtain or depend on unreliable inference, leaving enforcement gaps. Second, most IFC research on agents focuses narrowly on *indirect prompt injection attacks* rather than addressing broader problems such as safety or security tradeoffs, capabilities, and autonomy. IFC has the potential to guarantee that unsafe flows cannot occur, rather than depending on probabilistic checks. We therefore focus on IFC as a developer tool for reasoning about safety and security in agent interactions, prioritizing proactive design over probabilistic enforcement and reducing reliance on costly manual or inferred labels.

4.1 Identifying Agent Requirements

To anticipate the problems against which to safeguard (e.g., leaking sensitive information in meeting titles), we adapt the steps of the STPA framework to agents: First we identify direct and indirect stakeholders (for the calendar agent, direct stakeholders include the user, while indirect stakeholders include other event attendees). Then, for each stakeholder, we derive a set of values they expect from the system, and then invert these values into potential losses (in our example, a user may value privacy, with the corresponding loss being leakage of private information; event attendees may value timely communication, with the associated loss being delays in receiving event updates). Next, we analyze which system behaviors could lead to a loss (e.g., private information included in email communication). Finally, we evaluate which losses are important enough to address, deriving corresponding safety and security requirements that define the agent's expected behavior. In our example, we might arrive at the following two requirements: ***(REQ1)*** *Event notification emails shall include only essential event details (time, location, title, description). Private information, such as personal reasons for the update, must not be included unless explicitly authorized by the user.* ***(REQ2)*** *Whenever the agent updates an event, it shall promptly notify all other event attendees of the change.*

4.2 Defining and Enforcing Agent Specifications

Requirements are abstract system goals that a model may probabilistically interpret but that cannot be verified. To provide formal guarantees, they must be transformed into symbolic specifications. As noted in Section 2, prior work explored techniques such as IFC, access control, and temporal logic [3, 7, 27, 29, 30, 32, 39], showing their potential in enforcing certain constraints. In our motivating example, the two requirements above can be refined into the following two specifications (SPECs), respectively: ***(SPEC1)*** *All event details are private to non-attendees. Information derived from other information inherits equal or stricter privacy constraints. Users may explicitly confirm privacy labels. Each* `send_email` *call must exclude data private to the recipient.* ***(SPEC2)*** *Each* `update_event` *invocation must be directly followed by one* `send_email` *to every event attendee.*

SPEC1 is an information flow constraint, while SPEC2 is a temporal logic constraint. In practice, these specifications capture some necessary information, including sources and sinks for IFC. Additional context, such as tool call rules, may come from the system level, while labels like "private" or "trusted" need to be assigned at runtime, as discussed in Section 4.3.

Given specifications, to balance safety and flexibility while reducing manual effort, we adopt a four-tier enforcement structure. Here, "flow" refers broadly to constraints on both tool-sequence ordering and information flow:

- **Blocklist:** Automatically deny unacceptable flows. *E.g., Prevent private data from flowing into* `send_email` *(SPEC1).*
- **Mustlist:** Require certain flows. *E.g., After each* `update_event`, `send_email` *must be called once per attendee (SPEC2).*
- **Allowlist:** Permit safe flows without user confirmation. *E.g., The first confirmation email to each attendee after an update requires no confirmation on its necessity (SPEC2).*
- **Confirmation:** Require user to confirm ambiguous or high-stakes actions. *E.g., The user must confirm if private information should be included in an email (SPEC1).*

In the motivating example, the four-tier structure provides multiple strategies to prevent safety violations while supporting different levels of autonomy. A conservative policy may blocklist `send_email` call after `list_events`, so that no private event details are included in emails. A more flexible approach could be the agent masking unrelated event data from its context before composing the email. Developers willing to accept some risk might allowlist emails that use preapproved templates or apply keyword filters to reduce risks. Finally, confirmations could be applied selectively, for instance, required when emailing external recipients but skipped for internal communications.

This structure provides better flexibility between safety and capability, as it ensures that (1) low-risk flows proceed automatically, (2) unacceptable flows are deterministically blocked, and (3) uncertain or moderate-risk flows are escalated for human oversight.

Crucially, these SPECs should be enforced by entities independent of the agent, rather than hoping the agent will follow rules.

Prior work explored variable masking [7], separating control flow [8], and sandboxing [34] as ways to partition context. We envision that each tool call is intercepted and evaluated before execution, ensuring the agent cannot bypass constraints.

4.3 Acquiring Structured Information Labels

As noted above, enforcing constraints requires certain runtime information, particularly IFC labels. Existing approaches often depend on manual labeling [7] or inference from data origin with propagation only when deemed influential [30, 39]. This makes labels costly, while also leaving them potentially inconsistent and untrusted at tool boundaries.

MCP defines the boundary where tools are declared and invoked, but it offers only minimal, optional annotations and advises treating tools as untrusted. Therefore, clients cannot reliably obtain labels, making it impossible to enforce SPECs at runtime.

We propose extending MCP declarations to require key-value tags instead of optional hints. For instance, an event title could be tagged "public" or "private" under the key "confidentiality," and tagged "yes," "no," or "unsure" under the key "is_PII." This flexible tagging scheme supports arbitrary keys with categorical or uncertain values, enabling richer safety policies. For example, flows from data marked `{'confidentiality': 'private'}` to tools with `{'capabilities': 'external_write'}` can be deterministically blocked, while `{'is_PII': 'unsure'}` can trigger a confirmation.

We envision that MCP servers provide at least the following three labels, enabling developers to use the richer information to implement the enforcement structure described above:

- **Capabilities:** read-only, write-only, read-write, execute, etc.
- **Data Confidentiality:** whether information is sensitive.
- **Trust Level:** whether outputs are verified or untrusted.

For in-house tools, this MCP metadata can likely be trusted, but for an open market (where some even envision that agents could discover which tools to use), additional mechanisms will be needed to establish trust in the metadata. In corporate contexts, this may involve limiting agents to tools from trusted vendors or requiring label certification before integration. Eventually it may even be possible to verify some metadata against the tool implementation.

5 Preliminary Results and Discussion

To demonstrate the feasibility of our approach, we developed and analyzed a formal model of the augmented MCP framework in Alloy, a modeling language based on first-order relational logic [17]. Alloy was chosen because (1) its logic is expressive enough to model tool and agent behaviors, and (2) its analysis engine can formally verify whether system constraints satisfy safety specifications. Beyond this demonstration, we envision using Alloy or similar tools for formal analysis to provide guarantees about the safety of an augmented MCP framework.

Our Alloy model encodes execution steps, tool functions, and exchanged messages, each annotated with labels for confidentiality and integrity. Tool capabilities are formally defined: for example, `send_email` requires an email address, a title, and content, and must never receive unrelated private information unless explicitly declassified.

Hazardous flows are formalized as predicates in Alloy – Boolean conditions that describe when a property holds in a given system state. For example, `private_leak` is true if private data reaches an unauthorized tool. Mitigations are modeled as sanitation steps (e.g., `UserConfirmation`, `Declassify`) that must occur at specific points in the trace, such as requiring confirmation before sending data to an untrusted sink.

The Alloy Analyzer then exhaustively explores bounded execution traces to check whether hazardous predicates can still be satisfied despite these mitigations, thereby ensuring that unsafe flows are eliminated. In our context, without policies, the analyzer quickly identifies counterexamples to safety specifications, such as private data leaking into an external email. With policies and sanitation steps enforced, Alloy confirms that safety violations cannot occur with the given tools, while safe traces remain (e.g., creating an event, rescheduling it, and emailing participants without leaking private data). This demonstrates that unsafe flows can be deterministically blocked without collapsing the agent's capabilities.

By reasoning explicitly about hazards and enforcing constraints, autonomy becomes a configurable choice of agents [11]. Developers can tailor policies to match the severity and likelihood of risks they are prepared to accept, depending on the agent's role and the environment's stakes. Crucially, users are prompted only when a decision may lead to an actual loss, not every tool call.

Finally, this framing recognizes that AI agents are not universally suitable. In safety-critical or high-stakes settings, agents may warrant very limited autonomy, or exclusion altogether. While agents can enhance capability, they also introduce new safety risks, and this tradeoff must be made carefully. Our contribution offers a structured process to reason about these tradeoffs explicitly, rather than defaulting to unchecked capability maximization.

6 Future Plans

For the next steps, we plan to design and implement an external policy engine that intercepts tool calls in agent frameworks, showing that the label-based constraints and formal specifications can be enforced in real-world systems. In parallel, we will explore how developers can efficiently author and maintain labels through key-value tagging. We also intend to evaluate our process across a broader set of real-world tools and workflows, measuring coverage of risky interactions, usability impacts like notification fatigue, and tradeoffs between safety and utility in different domains. Beyond this, we will investigate extending labels to capture richer properties such as identity, scope, and provenance, enabling the detection of impersonation risks, provisioning attacks, and excessive delegation of authority. Together, these efforts will advance structured hazard analysis and label-enforced guardrails as a practical foundation for building safe and reliable AI agents.

Acknowledgments

This work was supported in part by the National Science Foundation (award 2206859), the REU-SE program[1], and an unrestricted gift from Google's GARA. We would also like to thank Christopher S. Timperley and the S3C2 Quarterly Meeting attendees for their valuable feedback on this work.

[1] https://www.cmu.edu/scs/s3d/reuse/

References

[1] Asim Abdulkhaleq, Stefan Wagner, and Nancy Leveson. 2015. A Comprehensive Safety Engineering Approach for Software-Intensive Systems Based on STPA. *Procedia Engineering* 128 (2015), 2–11. doi:10.1016/j.proeng.2015.11.498

[2] Sven Apel, Sergiy Kolesnikov, Norbert Siegmund, Christian Kästner, and Brady Garvin. 2013. Exploring feature interactions in the wild: the new feature-interaction challenge. In *Proceedings of the 5th International Workshop on Feature-Oriented Software Development (FOSD '13)*. Association for Computing Machinery, New York, NY, USA, 1–8. doi:10.1145/2528265.2528267

[3] Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. 2024. AI Agents with Formal Security Guarantees. In *ICML 2024 Next Generation of AI Safety Workshop*. https://openreview.net/forum?id=c6jNHPksiZ

[4] Luca Beurer-Kellner, Beat Buesser, Ana-Maria Creţu, Edoardo Debenedetti, Daniel Dobos, Daniel Fabian, Marc Fischer, David Froelicher, Kathrin Grosse, Daniel Naeff, Ezinwanne Ozoani, Andrew Paverd, Florian Tramèr, and Václav Volhejn. 2025. Design Patterns for Securing LLM Agents against Prompt Injections. arXiv:2506.08837 [cs.LG] https://arxiv.org/abs/2506.08837

[5] Muffy Calder, Mario Kolberg, Evan H Magill, and Stephan Reiff-Marganiec. 2003. Feature interaction: a critical review and considered forecast. *Computer Networks* 41, 1 (2003), 115–141.

[6] Zhaorun Chen, Mintong Kang, and Bo Li. 2025. ShieldAgent: Shielding Agents via Verifiable Safety Policy Reasoning. In *Forty-second International Conference on Machine Learning*. https://openreview.net/forum?id=DkRYImuQA9

[7] Manuel Costa, Boris Köpf, Aashish Kolluri, Andrew Paverd, Mark Russinovich, Ahmed Salem, Shruti Tople, Lukas Wutschitz, and Santiago Zanella-Béguelin. 2025. Securing AI Agents with Information-Flow Control. arXiv:2505.23643 [cs.CR] https://arxiv.org/abs/2505.23643

[8] Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. 2025. Defeating Prompt Injections by Design. arXiv:2503.18813 [cs.CR] https://arxiv.org/abs/2503.18813

[9] Zehang Deng, Yongjian Guo, Changzhou Han, Wanlun Ma, Junwu Xiong, Sheng Wen, and Yang Xiang. 2025. AI Agents Under Threat: A Survey of Key Security Challenges and Future Pathways. *ACM Comput. Surv.* 57, 7, Article 182 (Feb. 2025), 36 pages. doi:10.1145/3716628

[10] Dorothy E. Denning and Peter J. Denning. 1977. Certification of programs for secure information flow. *Commun. ACM* 20, 7 (July 1977), 504–513. doi:10.1145/359636.359712

[11] K. J. Kevin Feng, David W. McDonald, and Amy X. Zhang. 2025. Levels of Autonomy for AI Agents. arXiv:2506.12469 https://arxiv.org/abs/2506.12469

[12] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. 2025. A Survey on LLM-as-a-Judge. arXiv:2411.15594 [cs.CL] https://arxiv.org/abs/2411.15594

[13] Yining Hong, Christopher S. Timperley, and Christian Kastner. 2025. From Hazard Identification to Controller Design: Proactive and LLM-Supported Safety Engineering for ML-Powered Systems . In *2025 IEEE/ACM 4th International Conference on AI Engineering – Software Engineering for AI (CAIN)*. IEEE Computer Society, Los Alamitos, CA, USA, 113–118. doi:10.1109/CAIN66642.2025.00021

[14] Wenyue Hua, Xianjun Yang, Mingyu Jin, Zelong Li, Wei Cheng, Ruixiang Tang, and Yongfeng Zhang. 2024. TrustAgent: Towards Safe and Trustworthy LLM-based Agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. Association for Computational Linguistics, Miami, Florida, USA, 10000–10016. doi:10.18653/v1/2024.findings-emnlp.585

[15] Kung-Hsiang Huang, Akshara Prabhakar, Sidharth Dhawan, Yixin Mao, Huan Wang, Silvio Savarese, Caiming Xiong, Philippe Laban, and Chien-Sheng Wu. 2025. CRMArena: Understanding the Capacity of LLM Agents to Perform Professional CRM Tasks in Realistic Environments. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. Association for Computational Linguistics, Albuquerque, New Mexico, 3830–3850. doi:10.18653/v1/2025.naacl-long.194

[16] Kuo-Han Hung, Ching-Yun Ko, Ambrish Rawat, I-Hsin Chung, Winston H. Hsu, and Pin-Yu Chen. 2025. Attention Tracker: Detecting Prompt Injection Attacks in LLMs. In *Findings of the Association for Computational Linguistics: NAACL 2025*. Association for Computational Linguistics, Albuquerque, New Mexico, 2309–2322. doi:10.18653/v1/2025.findings-naacl.123

[17] Daniel Jackson. 2012. *Software Abstractions: logic, language, and analysis*. MIT press, Cambridge, MA.

[18] Dennis Jacob, Hend Alzahrani, Zhanhao Hu, Basel Alomair, and David Wagner. 2025. PromptShield: Deployable Detection for Prompt Injection Attacks. In *Proceedings of the Fifteenth ACM Conference on Data and Application Security and Privacy (CODASPY '25)*. Association for Computing Machinery, New York, NY, USA, 341–352. doi:10.1145/3714393.3726501

[19] Christian Kästner. 2025. *Machine learning in production: from models to products*. MIT Press, Cambridge, MA.

[20] LangChain. 2025. *Human-in-the-loop*. https://docs.langchain.com/oss/python/langchain/human-in-the-loop

[21] Leveson, Nancy G. and Thomas, John P. 2018. *STPA handbook*. MIT Partnership for Systems Approaches to Safety and Security (PSASS).

[22] Michael Martin, Benjamin Livshits, and Monica S. Lam. 2005. Finding application errors and security flaws using PQL: a program query language. In *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '05)*. Association for Computing Machinery, New York, NY, USA, 365–383. doi:10.1145/1094811.1094840

[23] Marco Milanta and Luca Beurer-Kellner. 2025. *GitHub MCP Exploited: Accessing private repositories via MCP*. https://invariantlabs.ai/blog/mcp-github-vulnerability

[24] Model Context Protocol. 2025. *What Is the Model Context Protocol (MCP)?* https://modelcontextprotocol.io/docs/getting-started/intro

[25] Simon Mylius. 2025. Systematic Hazard Analysis for Frontier AI using STPA. arXiv:2506.01782 [cs.CY] https://arxiv.org/abs/2506.01782

[26] Armstrong Nhlabatsi, Robin Laney, and Bashar Nuseibeh. 2008. Feature interaction: The security threat from within software systems. *Progress in Informatics* 5, 75 (2008), 1.

[27] Alex Olivier. 2025. *Access control and permission management for AI agents: building with security in mind*. https://www.cerbos.dev/blog/permission-management-for-ai-agents

[28] Shalaleh Rismani, Renee Shelby, Andrew Smart, Edgar Jatho, Joshua Kroll, AJung Moon, and Negar Rostamzadeh. 2023. From Plane Crashes to Algorithmic Harm: Applicability of Safety Engineering Frameworks for Responsible ML. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 2, 18 pages. doi:10.1145/3544548.3581407

[29] Raven Rothkopf, Hannah Tongxin Zeng, and Mark Santolucito. 2024. Procedural Adherence and Interpretability Through Neuro-Symbolic Generative Agents. arXiv:2402.16905 [cs.AI] https://arxiv.org/abs/2402.16905

[30] Shoaib Ahmed Siddiqui, Radhika Gaonkar, Boris Köpf, David Krueger, Andrew Paverd, Ahmed Salem, Shruti Tople, Lukas Wutschitz, Menglin Xia, and Santiago Zanella-Béguelin. 2025. Permissive Information-Flow Analysis for Large Language Models. arXiv:2410.03055 [cs.LG] https://arxiv.org/abs/2410.03055

[31] Brian Stanton, Mary F. Theofanos, Sandra Spickard Prettyman, and Susanne Furman. 2016. Security Fatigue. *IT Professional* 18, 5 (2016), 26–32. doi:10.1109/MITP.2016.84

[32] Uma Victor and Daniel Bass. 2025. *AI Agents Need an Access Control Overhaul - PydanticAI is Making It Happen*. https://www.permit.io/blog/ai-agents-access-control-with-pydantic-ai

[33] Haoyu Wang, Christopher M. Poskitt, and Jun Sun. 2025. AgentSpec: Customizable Runtime Enforcement for Safe and Reliable LLM Agents. arXiv:2503.18666 [cs.AI] https://arxiv.org/abs/2503.18666

[34] Yuhao Wu, Franziska Roesner, Tadayoshi Kohno, Ning Zhang, and Umar Iqbal. 2025. IsolateGPT: An Execution Isolation Architecture for LLM-Based Agentic Systems. arXiv:2403.04960 [cs.CR] https://arxiv.org/abs/2403.04960

[35] Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, Dawn Song, and Bo Li. 2025. GuardAgent: Safeguard LLM Agents via Knowledge-Enabled Reasoning. In *Forty-second International Conference on Machine Learning*. https://openreview.net/forum?id=2nBcjCZrrP

[36] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. 2025. tau-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=roNSXZpUDN

[37] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=WE_vluYUL-X

[38] Pamela Zave, Eric Cheung, and Svetlana Yarosh. 2015. Toward user-centric feature composition for the Internet of Things. arXiv:1510.06714 [cs.HC] https://arxiv.org/abs/1510.06714

[39] Peter Yong Zhong, Siyuan Chen, Ruiqi Wang, McKenna McCall, Ben L. Titzer, Heather Miller, and Phillip B. Gibbons. 2025. RTBAS: Defending LLM Agents Against Prompt Injection and Privacy Leakage. arXiv:2502.08966 [cs.CR] https://arxiv.org/abs/2502.08966

[40] Xueyang Zhou, Weidong Wang, Lin Lu, Jiawen Shi, Guiyao Tie, Yongtian Xu, Lixing Chen, Pan Zhou, Neil Zhenqiang Gong, and Lichao Sun. 2025. SafeAgent: Safeguarding LLM Agents via an Automated Risk Simulator. arXiv:2505.17735 [cs.AI] https://arxiv.org/abs/2505.17735

[41] Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. 2024. Agent-as-a-Judge: Evaluate Agents with Agents. *arXiv preprint arXiv:2410.10934* (2024).