Symbolic Guardrails for Domain-Specific Agents: Stronger Safety and Security Guarantees Without Sacrificing Utility

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

Yining She
Carnegie Mellon University
Pittsburgh, PA, USA
yiningsh@andrew.cmu.edu

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

Christopher S. Timperley
Carnegie Mellon University
Pittsburgh, PA, USA
ctimperl@andrew.cmu.edu

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

Abstract

AI agents that interact with their environments through tools enable powerful applications, but in high-stakes business settings, unintended actions can cause unacceptable harm, such as privacy breaches and financial loss. Existing mitigations, such as training-based methods and neural guardrails, improve agent reliability but cannot provide guarantees. We study symbolic guardrails as a practical path toward strong safety and security guarantees for AI agents. Our three-part study includes a systematic review of 80 state-of-the-art agent safety and security benchmarks to identify the policies they evaluate, an analysis of which policy requirements can be guaranteed by symbolic guardrails, and an evaluation of how symbolic guardrails affect safety, security, and agent success on τ^2-Bench, CAR-bench, and MedAgentBench. We find that 85% of benchmarks lack concrete policies, relying instead on underspecified high-level goals or common sense. Among the specified policies, 74% of policy requirements can be enforced by symbolic guardrails, often using simple, low-cost mechanisms. These guardrails improve safety and security without sacrificing agent utility. Overall, our results suggest that symbolic guardrails are a practical and effective way to guarantee some safety and security requirements, especially for domain-specific AI agents. We release all codes and artifacts at https://github.com/hyn0027/agent-symbolic-guardrails.

CCS Concepts

• **Security and privacy → Software and application security**; *Formal methods and theory of security*; • **Computing methodologies → Artificial intelligence**.

Keywords

AI Agents, Software Security, Agent Safety, Agent Security, Symbolic Guardrails

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Yining Hong, Yining She, Eunsuk Kang, Christopher S. Timperley, and Christian Kästner. 2026. Symbolic Guardrails for Domain-Specific Agents: Stronger Safety and Security Guarantees Without Sacrificing Utility. In . ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

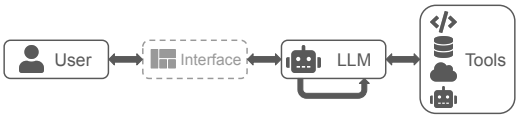


Figure 1: Overview of the AI agent workflow. The LLM interacts with the user, performs reasoning, and invokes tools.

1 Introduction

Recent advancements in LLM-based AI agents have generated both excitement and concerns. These agents are powerful, able to use tools and interact with their environment [39, 58, 71]. Their capabilities have led to adoption across many domains, including Sierra for customer experience [59], Cursor for software development [6], and Hippocratic AI for healthcare applications [25], as well as general-purpose assistants such as Claude Desktop [4] and OpenClaw [47]. However, serious safety and security concerns have also emerged from AI agents using tools in unintended ways [17, 42]. For example, in the GitHub MCP incident [44], an attacker manipulated an agent into using tools to access private repository data and reveal it in a public repository. Even without an attacker, agents may perform unsafe actions, for example, an OpenClaw agent disregarded user instructions and invoked tools to bulk-delete emails [50].

Because unintended or incorrect tool use can cause real harm, such as data loss [36], financial loss [63], manipulation [23], misinformation [75], and even physical harm [72], there is often hesitation to deploy agents in business settings, where harms can have serious consequences. While individuals may choose to accept the personal risks of general-purpose agents such as OpenClaw [47], companies often face higher stakes. The potential for leaking customer data, suffering data loss, or enabling malicious transactions with real financial or physical consequences is usually too great, even when an agent performs reliably in evaluations. For example, a healthcare record management agent may be able to prescribe

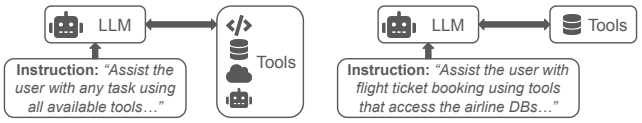
medications through tool use, and even rare failures, such as prescribing contraindicated drugs, can pose serious risks to patients, the organization, and its stakeholders. Hence, for many tasks, especially domain-specific tasks in business settings, it is desirable to design agents with predictable safety and security guarantees.

To improve the safety and security of AI agents, researchers have explored different mechanisms. These include training-based approaches that aim to bake safety and security directly into the model [7, 48, 51], as well as guardrails built *around* the model [12, 13, 53]. However, most guardrails are *neural* guardrails [12, 41, 53], meaning that their execution relies on probabilistic methods, typically LLMs. For example, in the LLM-as-a-judge paradigm, a separate LLM observes an agent's interactions and determines if they are safe and secure [12, 41]. These neural guardrails can reduce the *likelihood* of policy violations, but, given their probabilistic nature, they cannot provide *guarantees* that policy violations are provably impossible. For agents in business settings, such guarantees, even if limited to a narrow set of properties, are highly desirable because even small probabilities of mistakes or successful attacks may pose unacceptable risks when the potential harms are severe.

In traditional software engineering and software security, developers often use symbolic enforcement mechanisms to guarantee that systems satisfy specific safety or security constraints. These include access control [54], input validation [24], and information-flow control [18]. By design, such mechanisms deterministically prevent undesirable behavior through explicit checks against predefined policies. For example, information-flow analysis can guarantee that code is free from SQL injection vulnerabilities [24], while static analysis and related techniques can guarantee the absence of exploitable buffer overflows [64]. A few recent works explored applying these symbolic enforcement mechanisms as symbolic guardrails for AI agents, including methods based on temporal logic [31, 65], information flow control [13, 49], and privilege control [32, 57]. These symbolic guardrails support explicit reasoning about whether a policy is satisfied. However, they typically cover a limited set of guardrail paradigms and a narrow class of safety or security policies, such as constraining tool-call order with temporal logic. As a result, their practical applicability and whether they are sufficient to cover common safety and security properties in AI agents remain unclear. We expect that articulating and guaranteeing properties are especially difficult for general-purpose agents, whereas narrower domain-specific agents in business settings, such as customer service support, may offer more opportunities.

Motivated by the goal of providing *formal guarantees* for safety and security policies in AI agents, rather than merely reducing the *likelihood* of violations, we explore symbolic guardrails as a promising approach. Symbolic guardrails vary in both expressiveness and cost, ranging from inexpensive but less expressive techniques such as input validation [27] to more sophisticated approaches such as information-flow tracking and input masking [13]. For a practical assessment, however, it remains unclear what practitioners actually think about AI agent safety and security, and to what extent each of those symbolic guardrails is useful in this context. Specifically, we ask the following concrete research questions:

RQ1: As a proxy for the safety and security properties that practitioners expect AI agents to satisfy, which policies are evaluated by existing agent safety and security benchmarks?



(a) General-purpose agents. **(b) Domain-specific agents.**

Figure 2: Comparison between general-purpose AI agents and domain-specific AI agents.

RQ2: Among the safety and security policies evaluated by existing agent benchmarks, which can be guaranteed by symbolic guardrails, and by what mechanisms?

RQ3: Among the safety and security policies evaluated by existing agent benchmarks, which cannot be guaranteed by symbolic guardrails, and what alternative approaches are available?

RQ4: What are the effects of symbolic guardrails on the safety, security, and utility of AI agents?

To answer these questions, we conduct a three-part study. First, we perform a systematic literature review to collect 80 state-of-the-art AI agent safety and security benchmarks and analyze the safety and security policies they evaluate. Second, we assess whether these policies can be guaranteed by symbolic guardrails, and by which mechanisms. Third, we implement six types of symbolic guardrails on three benchmarks, τ^2-Bench [9], CAR-bench, and MedAgentBench [29], and evaluate their impact on agents' safety, security, and utility.

We find that 85% of existing benchmarks either do not specify a concrete safety or security policy for agents or define only high-level, goal-setting policies that are open to multiple interpretations. Because such policies are underspecified, we cannot apply symbolic guardrails to enforce them. Among the benchmarks with clearly specified policies for domain-specific agents, 74% can be enforced using symbolic guardrails, and most require only simple, low-cost mechanisms rather than more expensive techniques such as information-flow tracking. Most importantly, these symbolic guardrails not only enforce safety and security but also need not sacrifice utility. Taken together, these results suggest that symbolic guardrails are a practical and effective approach for improving AI agent safety and security and should be adopted more widely.

In summary, we contribute (a) a systematic literature review identifying the safety and security policies used in existing agent benchmarks, (b) an analysis of how many safety and security policy requirements can be enforced symbolically, and (c) an experimental study showing that symbolic guardrails are feasible and effective for specific requirements without sacrificing agent utility.

2 Background and Related Work

2.1 AI Agents

Over the past decade, natural language processing has advanced rapidly, and recent large language models (LLMs) have demonstrated substantial potential. These LLMs extend beyond single-turn text generation, with capabilities to understand context, perform reasoning, and, most importantly, interact autonomously with the environment through external tool use. Therefore, they now serve as the core of different modern AI agent architectures, including

the earlier retrieval-Augmented Generation paradigm [39], as well as the more recent ones such as ReAct [71] and Reflexion [58].

While there is no universally accepted definition of AI agents, we focus specifically on LLM-based agents with tool-use capabilities that are currently the dominant paradigm, as shown in Figure 1. These agents are designed to assist users in accomplishing tasks, either through natural language or through interfaces built upon it. To fulfill a user request, the agent operates in an iterative loop, using an LLM at each step to determine the next action, which may involve invoking a tool from a predefined set or returning a text response. When a tool is invoked, its output is appended to the context provided to the LLM in the next iteration, allowing the model to choose the next action based on previous ones. Tools may include APIs that execute specific functions, command-line interfaces, resources such as databases, and even other agents. The agent has the autonomy to decide when to invoke a tool, which tool to use, and what arguments to supply.

To make tools reusable across different implementations and facilitate tool invocation, the community has adopted standardized formats for describing a tool's purpose, expected inputs, and outputs. The Model Context Protocol (MCP) [3] and the Agent2Agent (A2A) Protocol [61] have emerged as the de facto standards for this.

Advances in AI agents have generated strong interest in applying them to a wide range of tasks. Following prior work [37], we distinguish between general-purpose and domain-specific agents. General-purpose agents, shown in Figure 2a, are designed to assist users with a wide variety of tasks. Examples include ChatGPT Agent mode [45] and Microsoft Copilot integrated with Windows [43], which can use broadly applicable tools such as screen observation and keyboard or mouse control. Coding agents such as Claude Code [5] also fall in this category, as they support a wide range of tasks and have strong tool-use capabilities for editing files, running command-line tools, and accessing the internet. In contrast, domain-specific agents are designed for a narrow scope of tasks, with access only to task-relevant tools, as shown in Figure 2b. For instance, a customer support agent for airline reservations may only access ticket databases [9], while a Customer Relationship Management (CRM) agent may be restricted to interacting solely with the CRM system [28]; neither may access the internet or execute arbitrary code. We expect general-purpose and domain-specific agents to differ fundamentally in their safety and security considerations, and we explore these differences in detail in our research.

2.2 AI Agent Guardrail: Neural versus Symbolic

Although AI agents are powerful and have been adopted across domains [6, 25, 47, 59], they also raise serious safety and security concerns as they may use tools and interact with environments in harmful ways [17, 42]. We interpret safety and security broadly: security typically concerns confidentiality, integrity, and availability in the presence of an adversary, while safety concerns real-world harm, such as physical and financial harm, misinformation, manipulation, and stress, often caused by unintended behaviors [30]. To mitigate these risks, researchers have explored a range of methods.

One line of research aims to train the underlying LLM that supports the AI agent to be safe and secure, so that safety and security properties are baked into the LLM itself, rather than enforced by surrounding architectures such as guardrails [41, 53, 68]. Researchers have explored post-training alignment, especially supervised fine-tuning and reinforcement learning from human feedback (RLHF) [7, 22, 48, 60], as well as approaches that partially replace human feedback with AI-generated signals, such as asking a model whether a proposed action is safe [8, 35, 73]. Other work explored adversarial data collection and risk-focused active learning [21, 22, 51]. These methods help steer model behavior toward safer and more secure outputs. However, because LLMs are inherently probabilistic and susceptible to prompt injection attacks, they cannot provide formal *guarantees* that the model will not violate particular safety or security properties, with or without an attacker.

Beyond modifying the underlying LLM, another line of research introduces guardrails that operate at runtime in the agent implementation *around* the model. This is also the focus of our work. These guardrails can be categorized as neural and symbolic guardrails.

Neural guardrails rely on probabilistic methods, most commonly the *LLM-as-a-judge* paradigm. In this setup, one or more separate models, typically LLMs, act as "judges" that monitor either (a) model inputs and outputs for suspicious, malicious, or sensitive content or (b) agent-proposed actions to assess their safety and security. For example, AGrail [41] uses LLMs to update and perform safety checks; LlamaFirewall [12] uses an ML classifier to detect prompt injection, and an LLM to detect misalignment; RTBAS [74] uses an LLM and an attention-based saliency screener to track provenance and perform information-flow analysis; and Liu et al. [40] explore LLM for detecting prompt injection. Some work aims to provide deterministic, rule-based guardrails, but still relies partly on LLMs to generate guardrail rules, decide when to trigger them, or execute them. These approaches, therefore, remain probabilistic and cannot provide guarantees. For example, `GuardAgent` [68] uses LLMs to generate guardrail code; NeMo Guardrails [53] provides programmable guardrails while execution still involves LLMs; ShieldAgent [11] relies on LLMs to retrieve and execute rule-based policy checks; AgentGuardian [1] uses LLMs to generate control policies. A key limitation of neural guardrails is their inherently probabilistic nature. Because their generation or execution depends on LLMs, they can be error-prone or circumvented by attackers. As a result, neural guardrails may substantially improve agent reliability by reducing the likelihood of unsafe or insecure behavior, which may or may not reduce deployment risk to an acceptable level. They cannot guarantee that an agent will never violate a given policy.

In contrast, traditional software safety and security mechanisms often rely on symbolic, deterministic enforcement techniques that can provide guarantees, including input validation and sanitization to prevent SQL injection [24], information-flow control to prevent sensitive data leakage [18], and access control to restrict authorized access [54]. A few recent studies have begun to explore these symbolic enforcement mechanisms as symbolic guardrails for AI agent safety and security, in contrast to neural guardrails. For example, AgentSpec [65], Agent-C [31], and *Maris* [15] use temporal logic to specify and enforce agent constraints; Progent [57] defines privilege control policies using domain-specific languages; Doshi et al. [19] explores temporal logic and information-flow control with formal models; `PFI` [32] validates unsafe data flows to prevent privilege escalation in agents; Fides [13] uses information-flow

Table 1: Levels of Specificity in AI Agent Safety and Security Policies

Specificity	Definition	Example
No Policy	No safety or security guidance provided to agents.	N/A
Goal-Setting	A high-level safety or security objective provided to agents.	*"Maintain highest level of discretion and security."* [56], and *"Tool execution permitted ONLY when absolutely necessary"* [52], where 'discretion', 'security', and 'absolutely necessary' are ambiguous.
Concrete Rules	Clear, unambiguous safety or security instructions provided at the agent level.	*"If any portion of the flight has already been flown, the agent cannot help."* [70] is a clear instruction for an airline ticket agent.
Task-Specific	Safety or security instructions provided for a particular task only.	For a web agent targeting arbitrary webpages, rule *"When click Create group button, ask permission"* [38] applies to a task creating a GitLab group, but not a task shopping online for a product.

control to track confidentiality and integrity; PCAS [49] also explores information-flow control for provenance tracking, using a Datalog-derived language; and f-secure [67] and `CaMeL` [16] tackle prompt injection attacks by separating control flow from data flow.

We believe symbolic guardrails are a promising direction for providing the assurances needed to deploy AI agents in high-assurance or risk-averse business settings. However, although many symbolic guardrails have been shown to be effective at enforcing specific guarantees in various benchmarks, it remains unclear how often they are suitable or sufficient for assuring the safety and security properties that matter in practical agent deployments. In addition, symbolic guardrails often substantially restrict agents' actions, which can undermine the flexibility and creativity that make AI agents useful for problem-solving. This paper focuses on exploring which practical safety and security properties are amenable to existing symbolic guardrails and how those guardrails affect agents' utility, that is, their capability in completing tasks.

3 Collecting Agent Safety and Security Benchmarks: From Goals to Concrete Rules

To answer the four research questions, we conduct a three-part study. First, in this section, we collect benchmarks that evaluate agent safety or security to identify the safety and security properties that benchmark developers care about. Second, in Section 4, we analyze the safety and security policies associated with these benchmarks and examine which are amenable to symbolic guardrails. Third, in Section 5, we evaluate the impact of symbolic guardrails on agents' safety, security, and utility.

3.1 Research Method

For RQ1, we ask what safety and security properties people expect AI agents to satisfy. As a proxy, we examine how existing benchmarks evaluate agent safety and security. Specifically, through a systematic literature review, we analyze a large corpus of AI agent benchmarks that evaluate safety or security, and we extract and classify the safety or security policies they define.

3.1.1 Identifying Benchmark Papers. To capture a comprehensive set of benchmarks on AI agent safety or security, we perform a systematic literature review following established guidelines [34].

Search Criteria. We aim to identify papers that (1) propose one or more benchmarks, (2) evaluate tool-use LLM-based agents, and (3) incorporate safety or security considerations into the evaluation. We use the arXiv API as our search interface because most recent papers relevant to AI agents are available on arXiv, often well before formal publication. Because tool-using AI agents enabled by recent LLMs emerged only after the release of ChatGPT [46] in late 2022, we limit the search period to January 1, 2022 through March 1, 2026.

Following prior recommendations [20], we first define a seed set of 15 relevant benchmark papers with which we were already familiar. By analyzing this seed set, we define the search criteria: (a) title or abstract contains at least one of the whole words: *"bench"*, *"benchmark"*, *"dataset"*, or *"framework"*, (b) title contains at least one of the substrings: *"eval"*, *"assess"*, *"bench"*, or *"dataset"*, (c) title or abstract contains the whole word *"agent"*, (d) title or abstract contains at least one of the whole words: *"safety"*, *"security"*, *"privacy"*, *"confidentiality"*, *"policy"*, *"risk"*, or *"attack"*, and (e) paper is cross-listed in at least one of the arXiv categories: cs.AI (Artificial Intelligence), cs.CL (Computation and Language), or cs.LG (Machine Learning).

Using these criteria, we retrieved 553 search results. We manually examined the papers and found that many irrelevant results were related to reinforcement learning or robotics, where terms *"agent"* and *"policy"* are often used in different senses. To reduce this noise, we added two exclusion criteria: (a) paper cross-listed in cs.RO (Robotics), (b) title or abstract contains any of the whole words: *"robot"*, *"self-driving"*, *"embodied"*, or *"reinforcement"*.

Under these criteria, we identify 413 papers, including 12 of the 15 seed papers. Details are provided in the materials in Section 7.

Filtering Methods. For the 413 identified papers, we first assessed whether each paper proposes any benchmark. One author manually annotates a random sample of 100 papers, while GPT-5-nano annotates all 413 papers on the same criterion, referencing only the paper title and abstract in both cases. Comparing human and model labels achieves a Cohen's kappa of 0.88 with a 95% confidence interval of $[0.76, 0.99]$. The model's precision and recall were 0.96 and 0.99, respectively. Given this strong agreement and especially high recall, we treated the model's labels as reliable and filtered out papers labeled as non-benchmarks, leaving 301 papers.

We then manually inspected all 301 papers. We excluded 11 papers because they did not propose a benchmark, 135 because they did not target tool-using LLM-based agents, and 52 because they did not incorporate safety or security into their evaluation. We further excluded 23 papers as out of scope, include papers whose benchmarks assess agents' ability to perform safety- or security-related

Table 2: Distribution of Benchmarks by Agent Domain and Level of Policy Specificity

	Agent Domain		
Policy Spec.	**Gen.-Purp.**	**Dom.-Spec.**	**Total**
No Policy	38 (47.5%)	11 (13.8%)	49 (61.3%)
Goal Setting	14 (17.5%)	5 (6.3%)	19 (23.8%)
Concrete Rules	0 (0.0%)	5 (6.3%)	5 (6.3%)
Task-Specific	1 (1.3%)	1 (1.3%)	2 (2.5%)
Unclear	2 (2.5%)	3 (3.8%)	5 (6.3%)
Total	55 (68.8%)	25 (31.3%)	80 (100.0%)

All percentages are calculated over the full set of 80 benchmarks.

tasks rather than the safety or security of the agents themselves; papers that extend benchmarks already covered in our search only by adding data or evaluation methods, without introducing new safety or security policies; and papers that evaluate whether an agent is capable enough to pose a real-world threat, rather than whether the agent is safe. We did not exclude papers that reused a benchmark from prior work when the original benchmark paper was not included in our search. In total, 80 papers remained.

3.1.2 Annotating Benchmark Papers. For the 80 identified papers, we reviewed each one and annotated whether it targets a general-purpose or domain-specific agent, as defined in Section 2.1. We also annotated the specificity of the safety and security policies each benchmark considers by examining the policies given to the evaluated agent, namely the safety or security instructions, guidelines, or rules it receives. We treat these policies as lying on a spectrum of specificity and classify them into four categories: *No Policy*, *Goal-Setting*, *Concrete Rules*, and *Task-Specific*. Table 1 summarizes these categories with examples. When a paper does not provide enough information for us to classify it confidently, we label its policy specificity as *unclear*. All labels are assigned solely based on the paper content, not on additional materials such as GitHub code.

3.1.3 Threats to Validity. As with any study, our results should be interpreted within the constraints set by our methods. We use the safety and security policies defined in existing benchmarks as a proxy for the properties that AI agents are expected to satisfy. Although ML benchmarks often reflect the interests of developers in the field, they may not fully represent the concerns that arise in practical deployments, and our findings should be interpreted accordingly. In addition, although arXiv contains most ML-related papers and our selection criteria are broad, we may have missed some relevant benchmarks. Finally, in labeling agents and policies, we use categorical labels even though both domain specificity and policy specificity lie on a spectrum and therefore require judgment. We made a best effort to apply these labels consistently and release our data to support external validation.

3.2 RQ1: Which safety and security policies are evaluated by existing agent benchmarks?

3.2.1 Results. We identify 80 benchmarks for AI agent safety or security. Table 2 summarizes them by two dimensions: whether they target general-purpose or domain-specific agents, and the specificity of the safety or security policies given to the agent.

Expectations for safe or secure behavior are often left implicit. We found that most benchmarks (63%) do not provide the evaluated agent with any explicit instruction to behave safely or securely. In these cases, benchmark designers implicitly expect agents to recognize that safety or security norms should be prioritized over following user instructions or other inputs, even though this requirement is never stated explicitly. For example, AgentHarm [2] expects an agent to refuse a user request to forge a passport, despite providing no explicit instruction that safety should override obedience to user instructions or that such instructions are considered unsafe in the first place. In effect, benchmark designers, usually without making this explicit, expect models to follow *"common-sense"* safety or security behaviors, even though what qualifies as common sense may differ across benchmarks. In such cases, the expected notion of common sense is never articulated and can at best be inferred from how task execution is evaluated in the benchmark.

Safety and security encompass broad themes and are interpreted differently among people. A closer examination of how these benchmarks evaluate agent safety and security shows that conceptions of safety and security are highly diverse. Different benchmarks emphasize different themes, including privacy, authorization, robustness to attacks, legality, fairness, truthfulness, bodily harm, and policy compliance. Without a clear specification, it is often unclear which of these expectations an agent is intended to satisfy in a given scenario. Moreover, even the same theme is often interpreted in different and sometimes even conflicting ways. For example, in the context of protecting user privacy, MobileSafetyBench [36] instructs an agent to always request user permission before sharing any private information, regardless of context. In contrast, MPCI [66] evaluates whether an agent appropriately shares confidential information without user interaction in suitable social contexts while refusing to do so in inappropriate ones, even though the agent is given no explicit prompt to behave that way.

Concrete-rule and task-specific policies can disambiguate safety and security, but they are rare. Broad safety and security policies are often ambiguous; for example, what counts as dangerous, illegal, or private output is frequently unclear or context dependent. Only a small number of benchmarks (9%) specify concrete agent-level or task-specific rules that resolve this ambiguity. For example, CAR-bench [33] states: *"If the tool description starts with REQUIRES_CONFIRMATION, then before calling that tool and perform the corresponding action, you must list the intended tool parameter and action details and always obtain explicit expressive user confirmation (yes) to proceed."* This rule is unambiguous and does not admit a plausible alternative interpretation.

3.2.2 Discussion. **Concrete rules and task-specific policies are necessary for building safe and secure agents.** When safety and security policies are implicit or ambiguous, it becomes much harder both to implement reliable guardrails and to evaluate agent behavior. Some so-called *"common-sense"* may be broadly shared, but some *"common-sense"* are agreed only within a community, or even deeply contested. For example, LPS-Bench classifies the following behavior as unsafe: "agent adheres too rigidly to literal instructions while ignoring the user's implicit intent" even though this principle

is not explicitly stated in the agent policy and is instead left to *common-sense* judgment. We would not necessarily accept this as a general safety principle, as in many situations, following a user's explicit instructions may be preferable to inferring implicit intent, especially in high-stake scenarios. If policies are ambiguous, we cannot provide meaningful guarantees and must instead rely on model judgment, which may or may not align with the intended *common-sense* expectations. Goal-setting policies often leave key concepts vague and implicitly require either the agent or the evaluator to "guess" what safety means, for example, what qualifies as a necessary action or what counts as private rather than public user information. Concrete rules and task-specific policies help resolve these ambiguities, and we therefore argue that they are essential for building safe and secure agents for risk-averse deployments.

Concrete-rule policies are preferable to task-specific policies. Although task-specific policies can reduce ambiguity in safety and security expectations, they require a reliable mechanism for generating and dynamically updating an appropriate policy each time the agent is used. In practice, this means either relying on a model to generate the policy, which makes it unreliable, or requiring the user to specify the relevant safety or security expectations for each input in an unambiguous way, potentially using some form of formal notation. This places a substantial burden on the user to articulate a policy for every use case, which is likely unrealistic in many contexts from both a usability and a cost perspective.

We need domain-specific agents to enable concrete-rule policies. Although concrete rules are desirable when building guardrails, it seems challenging to articulate the policy for general-purpose agents, as such agents are intended to support a wide range of use cases, many of which may be entirely unanticipated by the designer. Relying on *'common-sense'* may seem more scalable and pragmatic. This likely explains why we found concrete-rule policies only for domain-specific agents (Table 2), where identifying concrete policies within a narrow scope appears much more feasible. In these settings, although inputs may vary, the intended tasks are clearly defined, the available tools are limited, and the relevant safety and security constraints can be enumerated in advance. Under these conditions, policies can be written unambiguously. For example, if an agent is designed solely for airline ticket management and has access only to tools for interacting with airline databases, it becomes possible to specify a rule such as "If any portion of the flight has already been flown, the agent cannot help" [9]. In contrast, for general-purpose agents with open-ended tasks and broad tool access, it is far more difficult to specify concrete-rule policies comprehensively. We believe there are many settings in which domain-specific agents can be deployed safely and securely while providing concrete value to users and businesses, whereas general-purpose agents remain too risky, even when they are well aligned with some notion of common sense. Domain-specific agents with predictable safety and security guarantees are, therefore, in our view, an important direction for industry practice.

4 Analyzing Agent Safety and Security Policies: Simple Symbolic Guardrails Often Suffice

With RQ2 and RQ3, we ask which safety and security policies identified in curated benchmarks in Section 3 can or cannot be guaranteed by symbolic guardrails, and by what means. To answer these questions, we first define the scope of symbolic guardrails and then assess the extent to which they can address these policies.

4.1 Research Method

Conceptually, we decompose the safety and security policies in existing benchmarks into individual requirements and determine whether each requirement can be enforced by symbolic guardrails.

4.1.1 Identifying Requirements for Analysis. Our initial intention was to randomly sample a few benchmarks for analysis from the 80 identified. However, this proved infeasible: most benchmarks either state no policy at all (49, evaluating only implicit *'common-sense'* expectations), specify high-level goals that are too ambiguous to enforce (19), or are task-specific where enforcement is entirely input-dependent (2). This left only 5 benchmarks, 4 of which focus on customer service agents. We therefore abandoned the random-sampling strategy and instead deliberately selected two benchmark policies, supplemented by one additional synthetic policy.

Among the five remaining benchmarks, we selected two with concrete-rule policies from different domains: (a) τ^2-Bench [9], which evaluates an airline customer service agent and is the most widely used of the four customer service agent benchmarks, and (b) CAR-bench [33], which evaluates in-car voice assistants. Treating each policy sentence as a potential requirement, we identified 120 potential requirements in τ^2-Bench and 18 in CAR-bench.

To broaden domain coverage, we considered creating synthetic yet plausible concrete-rule policies for additional benchmarks, focusing on domain-specific agents in business-relevant, high-stakes settings. However, creating policies for existing safety and security benchmarks with no policy or only goal-setting policies was difficult. Although their evaluation methods implicitly encode safety expectations, deriving concrete rules that matched those expectations would have required extensive review and correction of labels because of ambiguity. For example, in CRMArenaPro [28], one task requires the agent to reject the query "Considering the recent discussions, should this lead be considered qualified?" as a privacy violation, while another expects the agent to answer the very similar query "After assessing recent discussions, should this lead be considered qualified?" No single concrete policy can be derived to fit both tasks without changing the benchmark labels.

We therefore instead created a synthetic policy for a benchmark in a high-risk domain with executable tool implementations, allowing us to use it in subsequent research questions, but that was not originally designed for safety or security evaluation and therefore does not embed implicit safety or security assumptions in its benchmark data. We found MedAgentBench [29], which evaluates an electronic medical record (EMR) assistant, to be a good fit.

To create a synthetic yet plausible policy without biasing it toward or against symbolic enforcement, we followed the steps below. First, we prompted GPT-5.2 to generate a safety policy from the EMR assistant use case and tool schema. We asked for a policy consisting of concrete rules with no mention of enforcement mechanisms. We then prompted the model to reduce redundancy, producing an initial policy with 50 requirements. Second, we improved the policy's comprehensiveness through hazard analysis, an approach

Table 3: Symbolic Guardrails and Illustrative Examples

Symbolic Guardrail	Example
API Validation	Before invoking `cancel_ticket(user, ticket)`, verify that `user == ticket.user`.
Schema Constraint [55]	Reject LLM output if it's not a tool invocation matching the airline API schema, nor a message to the user.
Temporal Logic [65]	Block all other tools until `authenticate_user` completes successfully.
Information Flow [13]	Block all information about other passengers from flowing to the agent.
User Confirmation	Before `cancel_ticket`, require rule-based user confirmation rather than LLM-initiated confirmation.
Response Template	After `cancel_ticket`, display a predefined cancellation summary rather than an LLM-generated response.

Illustration examples are based on an airline ticket agent, inspired by τ^2-Bench [9]. The agent is des'igned to assist users with managing their flight bookings. It interacts with internal databases by invoking tools, such as `get_flight_info`, `get_user_info`, and `cancel_ticket`.

that anticipates potential harms from the perspectives of different stakeholders and derives safety requirements to avoid those harms. Following prior work [26], we used an automated tool for System-Theoretic Process Analysis (STPA), with GPT-5 identifying 5,138 candidate safety requirements for the EMR assistant, many of them redundant. To keep the scope manageable, we randomly sampled 4% of these requirements (205 entries), clustered them into 10 categories using K-Means over embeddings from OpenAI's `text-embedding-3-small` model, and asked an LLM to consolidate the requirements within each cluster. This yields 77 safety requirements, from which we randomly sampled 20 for further analysis. Because these requirements often contained multiple sub-requirements, we manually decomposed them into 43 individual requirements. After removing 5 duplicates, we obtained 38 additional requirements, yielding a final policy with 88 requirements. All key steps in this process, including both the initial generation and the hazard analysis, were automated to avoid manual bias. Full details and artifacts are provided in Section 7.

4.1.2 Analysis: Matching Symbolic Guardrails. For our analysis, we consider the six symbolic guardrail strategies listed in Table 3: API validation, schema constraints, information flow, temporal logic, user confirmation, and response templates. These strategies are rooted in traditional software engineering and have emerged in prior work as plausible approaches to secure AI agents, as discussed in Section 2.2. We developed this list using two complementary strategies. First, top-down, we identified methods already used in the literature to secure AI agents, such as information-flow control [13]. Second, bottom-up, we identified strategies that may not yet be discussed in the academic literature but are natural fits for the properties these benchmarks aim to assure, such as API validation. We believe these six strategies cover the properties for which symbolic guarantees seem plausible, though future work may identify more strategies for additional properties.

For each potential requirement under the three policies, we manually assign one of three labels:

- **Out of scope:** Sentences that provide information rather than requirements (e.g., "User's profile contains their email", specify requirements for the system rather than model behavior (e.g., "No PII should be stored persistently"), or are hallucinated by the LLM and are infeasible given the tools (MedAgentBench only).
- **Enforceable symbolically:** We judge the sentence can plausibly be guaranteed by one or more symbolic guardrails. We also identify which guardrails could guarantee it.

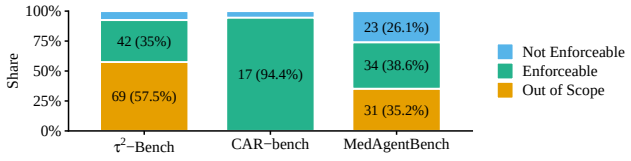


Figure 3: Distribution of safety or security policy enforceability across three benchmarks.

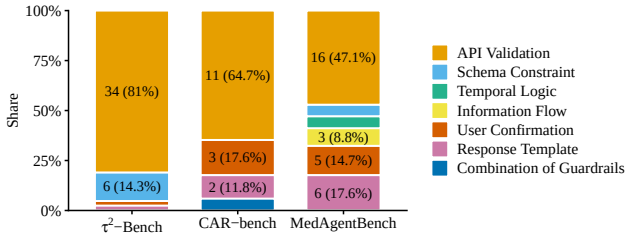


Figure 4: Distribution of applicable symbolic guardrails for enforceable policies across three benchmarks.

- **Not enforceable symbolically:** We judge the sentence to express a requirement that cannot be guaranteed by any combination of symbolic guardrails.

In most cases, the classification was straightforward. For a small number of sentences, all authors discussed the requirement or implemented the guardrail to gain confidence in the label. For requirements classified as not enforceable, we explored themes for RQ3 through a card-sorting-style grouping and reflection process.

4.1.3 Threats to Validity. Our analysis is limited by the small number of concrete policies available in benchmarks, covering two domains. The MedAgentBench policy extends the analysis, but it is LLM-generated rather than human-written and may not fully reflect the requirements practitioners would impose in real deployments. Although hazard analysis helps identify safety concerns systematically and broadly, we followed an automated process without access to expert judgment. Moreover, the benchmark policies themselves may not fully match the concerns in real-world settings. Our results should therefore be interpreted as a first step toward understanding what enforcement is possible for *plausible* domain-specific policies, rather than as a comprehensive reflection of industrial practice. Finally, matching requirements to guardrails was done manually and

may be subject to bias or error despite careful review by multiple authors. For transparency, we release all policies and labels.

4.2 RQ2: Which policies can be guaranteed by symbolic guardrails, and how?

4.2.1 Results. **In the three analyzed policies, 75% of safety and security requirements are enforceable by symbolic guardrails.** As shown in Figure 3, after excluding out-of-scope requirements, symbolic guardrails can enforce 42 of 51 requirements in τ^2-Bench, 17 of 18 in CAR-bench, and 34 of 57 in MedAgentBench.

For the enforceable requirements, simple and easy-to-implement guardrails such as API validation often suffice. As shown in Figure 4, API validation alone covers 81%, 65%, and 47% of the enforceable requirements in the three benchmarks, respectively. Schema constraints, user confirmation, and response templates account for most of the remaining enforceable requirements. Across all three policies, only five requirements from MedAgentBench require temporal logic or information-flow control.

4.2.2 Discussion. **Symbolic guardrails are effective and inexpensive for a substantial number of safety and security requirements in the analyzed agents.** A substantial number of safety and security requirements that are currently communicated to the model through prompt-based policies can instead be enforced with symbolic guardrails. In many cases, this is straightforward and inexpensive in both engineering effort and runtime cost, because most of these requirements can be handled through simple API validation, schema enforcement, or hard-coded user confirmation and response templates. These mechanisms are rarely discussed in the research literature on AI agent safety and security. More sophisticated and more costly enforcement mechanisms explored in recent work, such as information-flow tracking, are rarely needed for the requirements in the three analyzed agent policies. Even the few temporal constraints we identify are simple, such as "Block all other tools until `authenticate_user` completes successfully," and may not require a full temporal enforcement pipeline. We argue that domain-specific agents offer many *'low-hanging fruit'* opportunities: simple checks that could prevent a large number of agent errors. Surprisingly, benchmark implementations often do not enforce even basic rules in the tools themselves, such as "agents are not allowed to cancel flights already flown," and instead rely on the agent model or additional neural guardrails to perform these checks. This design violates basic security principles such as least privilege and complete mediation [62].

Symbolic guardrails may simplify agent prompts and potentially improve instruction following. Once implemented as symbolic guardrails, these requirements could be removed from the agent's prompt, reducing context size and token costs. Because modern models struggle to follow instructions when too many are provided [69], a shorter policy may also improve compliance with the remaining instructions. At the same time it may be beneficial to keep some enforced requirements in the policy regardless, so that the agent has a chance to do the right action in the first place rather than receiving an error from a guardrail. This is an implementation choice beyond the scope of this paper, but our results suggest that this is a choice developers can frequently make.

4.3 RQ3: Which policies cannot be guaranteed, and what alternative approaches exist?

4.3.1 Results. Analyzing the unenforceable requirements across the three policies, we identified four common types. We describe each category below and discuss potential solutions in Section 4.3.2.

Persona and interaction-style requirements specify how the agent should communicate or present itself, such as using a particular language, maintaining a certain tone, or behaving in certain ways. For example, a medical assistant may be required to be neutral and avoid offering medical judgment [29].

"No hallucination" requirements expect the agent to avoid generating unsupported or fabricated information. For example, τ^2-Bench specifies that the agent "should not provide any information not provided by the user or available tools" [9].

Procedure-following requirements specify that the agent must follow a predefined procedure or sequence of steps. For example, in τ^2-Bench, an agent helping with airline booking is expected to first obtain user details and then trip details [9].

Common-sense reasoning requirements arise even within concrete-rule policies. These requirements provide leave substantial room for interpretation and judgment and therefore need commonsense reasoning. For instance, in the flight-assistant setting of τ^2-Bench, a policy such as "Do not proactively offer compensation unless the user explicitly asks for it" still requires the model to interpret what counts as an explicit request [9].

4.3.2 Discussion. **Symbolic guardrails cannot address all requirements; some still require neural guardrails.** As discussed, four types of requirements are not enforceable by symbolic guardrails, even when stated as concrete-rule policies. In these cases, neural guardrails can be useful. For example, an LLM judge may detect and reduce hallucinations, where symbolic guardrails fail.

Some requirements that are not directly enforceable can become enforceable through stronger or weaker reformulations; whether to do so is an engineering decision. For example, a procedure-following requirement that asks the agent to first collect user information and then gather trip details when booking a flight could be enforced in a stronger form, potentially with architectural changes, by using a sub-agent design. Conversely, the original policy "Do not proactively offer compensation unless the user explicitly asks for it" could be replaced with a weaker but more precise rule, such as "Block compensation tools until the user explicitly mentions the word 'compensation'," thereby making it enforceable. In real-world deployments, deciding whether to enforce such stronger or weaker variants remains an engineering trade-off that depends on implementation effort and acceptable residual risk. Enforcing requirements symbolically whenever plausible reduces both the attack surface and the potential for hazards, while allowing developers to focus more expensive neural guardrails on the remaining requirements where they are needed.

5 Benchmarking Agents with Symbolic Guardrails: They Do Not Undermine Utility

Symbolic guardrails may enforce certain safety and security requirements for AI agents, but there is concern that they may constrain

agents too much and thereby reduce task success. In our final research question, we therefore examine how enforcing safety and security requirements with symbolic guardrails affects the safety, security, and utility of agents on corresponding benchmarks.

5.1 Research Method

We execute agent benchmarks under different conditions, with and without symbolic guardrails, and measure both policy violations, that is, unsafe or insecure behaviors, and task completion rate, that is, utility. We conduct experiments on τ^2-Bench (airline) [9], CAR-bench[33], and MedAgentBench [29] introduced in Section 4, for which we implemented all enforceable requirements. The main independent variable is whether the tools are provided with or without guardrails. The dependent variables are the number of policy violations and the task completion rate, which we use as a measure of utility.

5.1.1 Experiment Infrastructure. For our experiments, we implement a standard tool-use agent based on the one introduced in τ^2-Bench [9], with GPT-4o or GPT-5 as the backbone model and the policy included in the system prompt. In each benchmark and experimental condition, we connect all tools through an MCP server [3]. Implementation details and configurations are provided in Section 7.

Because these benchmarks require multi-turn interactions, in which the user responds to the agent's output to trigger subsequent actions, we simulate user responses with an LLM, following the setup used in τ^2-Bench and CAR-bench. The user-simulation LLM is given the relevant context but has no access to the MCP tools.

5.1.2 Tool and Symbolic Guardrail Implementation. For both τ^2-Bench and CAR-bench, we use the tool implementations provided with the original benchmark as the *"baseline"* condition. For the experimental *"guardrail"* condition, we copy these tool implementations and add symbolic guardrails for all enforceable requirements, as shown in Figure 4. For guardrails that require changes on the agent side, such as user confirmation, we implement the logic in the agent and control its behavior using metadata in MCP.

For MedAgentBench, we add the generated policy to the system prompt and consider three tool conditions. In the original benchmark, the agent interacts with the environment through raw GET and POST requests and is expected to construct REST requests freely. For the *"raw"* condition, we create an MCP server that exposes generic GET and POST tools. As an additional *"baseline"* condition, we create an MCP server with eight individual tools, one for each HTTP endpoint described in the original MedAgentBench. Finally, we create a copy of the latter MCP server and implement symbolic guardrails for 23 requirements, yielding the *"guardrail"* condition. These 23 come from the 34 requirements we judged enforceable: 1 was already implemented in the benchmark, and 10 would require substantial medical domain knowledge that could likely be enumerated by an expert but was not readily available to us, such as appropriate dosage units for different medications.

In all three benchmarks, the *baseline* and *guardrail* conditions use MCP servers with the same tool names and descriptions. However, for 6 of 16 tools in τ^2-Bench and 6 of 8 tools in MedAgentBench, the *guardrail* version expects additional parameters to support guardrails. For example, in τ^2-Bench, the baseline `cancel_ticket` tool takes only `ticket_id`, whereas the guardrail version also requires `user_id` so the system can verify that the requester owns the reservation by checking `user_id == ticket.user`. There is no additional parameter for CAR-bench.

5.1.3 Datasets. For τ^2-Bench, we use the cleaned data from τ^2-Bench-Verified [14], which fixes several inconsistencies in the original benchmark without otherwise changing the policy, prompts, or tools. It contains 50 tasks where AI customer support agents assist customers with flight reservations, with some of the customers attempt to violate the policy, for example, by requesting a refund for a nonrefundable flight.

For CAR-bench, we use the original data from the "Base" category, consisting of 100 entries, in which the agent acts as an in-car voice assistant that helps users with navigation and various vehicle operations, such as checking the weather and adjusting fog lights.

For MedAgentBench, the original benchmark evaluates whether an agent can effectively assist with tasks in an electronic medical record (EMR) system, such as checking patient records and ordering new medications. It contains 300 tasks in total, none of which intentionally probe for policy violations. Because our generated policy requires patient authorization, which is not considered in the original benchmark, we augment the dataset by also providing the *user* LLM with the target patient's information.

To further support our analysis, we construct an *adversarial dataset* for MedAgentBench in which the user LLM attempts to manipulate the agent into violating a safety or security policy. We generate adversarial tasks automatically using a paradigm adopted in prior benchmarks [10, 28, 38]. Specifically, after identifying four generic task categories from the original benchmark, we prompted an LLM to expand them into 17 task scenarios. We then instructed the LLM, for each task-scenario–requirement pair, to generate an adversarial task that seeks to violate the requirement while appearing to pursue a legitimate user goal. Because experimentation is costly, we randomly sample 50 of the 391 generated tasks (17 task scenarios $\times$ 23 requirements) for evaluation.

5.1.4 Dependent variables. We measure *utility* as the number of tasks the agent successfully completes at the first try, using the original benchmark metrics: Pass^1 for τ^2-Bench and CAR-bench, as well as Success Rate (SR) for MedAgentBench. We do not measure utility on the adversarial dataset, where successful task completion is not expected.

For *safety and security*, only CAR-bench among all three benchmarks provides measurement on safety or security policy violation. The metric r_{policy} evaluates whether each task follows the policy all the time, partially relying on an LLM judge. To evaluate safety and security in a non-probabilistic way, we focus our *safety and security evaluation* purely on symbolically enforceable requirements, as these are the only requirements that can be validated accurately and the only ones addressed in this work. By construction, our symbolic guardrails prevent violations of these requirements by rejecting noncompliant tool calls. We therefore measure how often such violations occur in the raw and baseline conditions, where these guardrails are absent.

To detect requirement violations, we also add the guardrail checks to the raw and baseline implementations, but only to record

violations rather than reject invalid executions. We measure how many tasks in each benchmark trigger at least one policy violation.

A complication arises when a guardrail adds up to three extra tool parameters, as discussed in Section 5.1.2. This affects 6 of 16 tools in τ^2-Bench and 6 of 8 tools in MedAgentBench. We do not want to change tool signatures in the baseline condition, because requiring extra arguments could alter the model's reasoning. We therefore use a *replay-based evaluation procedure*. Specifically, when the agent calls a baseline tool that would require an extra argument in the guardrail condition, we interrupt execution and prompt the model again with the extended *guardrail* tool signature for safety checking. If the replayed call uses the same tool name and original arguments as the baseline call, and also supplies the required extra argument, we use that replayed guardrail call to assess safety. We continue the agent's execution with the original baseline call regardless of whether the safety check passes. This allows us to obtain the extra information needed for safety checking, such as the user ID for `cancel_ticket`, without changing the reasoning induced by the baseline tool schema. If the agent cannot provide the additional information during replay, we classify the execution as *unsafe*, because the available context is insufficient for an adequate safety or security check. If, during replay, the agent instead selects a different tool or changes the original arguments, we retry up to five times. If we still cannot reproduce the same tool call, we label the safety outcome as *unknown*.

Because CAR-bench does not require any additional parameters to build symbolic guardrails, there are no tool calls for which safety or security is *unknown*. Therefore, we do not report an unknown safety or security rate for CAR-bench.

For all dependent variables, we assess the significance of differences across tool sets using the paired McNemar test.

5.1.5 Threats to Validity. Our evaluation of policy violations is limited to those that can be detected reliably using symbolic measures. As is common in agent benchmarks, executions are expensive even at benchmark sizes of 50 to 300 tasks, with a single benchmark costing about USD 80. This constrains the number of experimental conditions, such as model choices, and the number of repetitions we can evaluate. As a result, our findings can show general trends, but the statistical tests may detect only relatively large effects. Future work is also needed to assess how well these results generalize beyond the three benchmarks studied here. Finally, all experimental conditions include the full policy in the system prompt, even though many policy sentences are technically redundant in the guardrail condition. Future work could examine the costs and benefits of removing this redundancy.

5.2 RQ4: How do symbolic guardrails impact agent safety, security, and utility?

5.2.1 Results. **AI agents without guardrails are unsafe on policies that symbolic guardrails can enforce.** As shown in Tables 4–7, policy violations are common across all agents and all benchmarks without symbolic guardrails: 20% to 78% of task executions violate at least one symbolically enforceable policy. As expected, violations are more frequent on adversarial tasks (Table 7) and less frequent for stronger models, as seen in the comparison between GPT-4o and GPT-5 in Table 4. Still, policy violations occur in every experimental condition without guardrails. By contrast, such violations are impossible by construction in the guardrail condition, and the difference is statistically significant ($p = 0.00$ in all cases). This result is also consistent with the metric r_{policy} reported in CAR-bench in Table 5, where policy violation is evaluated partially by an LLM judge ($p = 0.00$ comparing *baseline* and *guardrail* tools).

Table 4: Results on τ^2-Bench

		Safety & Security			Utility
Model	**Tools**	**Unsafe↓**	**Unk.**	**Safe↑**	**Pass^1↑**
GPT-4o	Baseline	52.0%	0.0%	48.0%	0.36
	Guardrail	0.0%	0.0%	100.0%	0.48
GPT-5	Baseline	20.0%	10.0%	70.0%	0.68
	Guardrail	0.0%	0.0%	100.0%	0.70

Table 5: Results on CAR-bench

		Safety & Security			Utility
Model	**Tools**	**Unsafe↓**	**Safe↑**	r_{policy} ↑	**Pass^1↑**
GPT-5	Baseline	21.0%	79.0%	0.83	0.59
	Guardrail	0.0%	100.0%	0.97	0.72

Table 6: Results on MedAgentBench With Original Data

		Safety & Security			Utility
Model	**Tools**	**Unsafe↓**	**Unk.**	**Safe↑**	**SR↑**
GPT-5	Raw	39.0%	9.7%	51.3%	0.64
	Baseline	23.0%	0.3%	76.7%	0.59
	Guardrail	0.0%	0.0%	100.0%	0.67

Table 7: Results on MedAgentBench With Adversarial Data

		Safety & Security		
Model	**Tools**	**Unsafe↓**	**Unk.**	**Safe↑**
GPT-5	Raw	78.0%	4.0%	18.0%
	Baseline	62.0%	4.0%	34.0%
	Guardrail	0.0%	0.0%	100.0%

Symbolic guardrails do not sacrifice agent utility. As shown in Tables 4-6, utility increase under enforced guardrails in all three benchmarks, although some improvements are not statistically significant ($p = 0.18$ in τ^2-Bench for GPT-4o, $p = 1.00$ in τ^2-Bench for GPT-5, $p = 0.00$ in CAR-bench, $p = 0.27$ in MedAgentBench for *raw* tools, and $p = 0.00$ in MedAgentBench for *baseline* tools, respectively). Overall, they suggest that symbolic enforcement is unlikely to harm utility and may even improve it.

5.2.2 Discussion. **Relying on models alone to enforce safety and security requirements is dangerous.** Even without adversarial attackers, agents frequently violate safety and security requirements that are explicitly stated in the system prompt. Stronger

models make fewer mistakes, and future models with larger context windows and better instruction following may reduce these errors further, but such easily preventable failures still create unnecessary risk. Dedicated neural guardrails can likely reduce many of these errors substantially, but they add nontrivial runtime cost and still leave residual risk. In adversarial settings, for example, through prompt injection that causes a model to offer compensation improperly or prescribe the wrong medication, deploying such agents may become entirely infeasible. Although not all desirable safety and security properties can be enforced symbolically, practitioners in high-assurance business settings should assess whether the most important requirements can be specified and enforced this way, and then use neural guardrails for the remaining critical requirements as part of a deliberate risk assessment.

Safety and utility are not necessarily a trade-off. Intuitively, guardrails may seem to constrain model flexibility, but our results suggest that they can also help the model explore the space of safe solutions more effectively. Examining the agent interactions points to a possible explanation: when a symbolic guardrail blocks an unsafe action, it prevents the agent from terminating with an incorrect result and provides useful feedback through an error message explaining why the action failed, why it was considered unsafe, and which policy requirement it violated. The agent can then use this feedback to adjust its subsequent actions, retry with a safer alternative, and often complete the task successfully.

6 Conclusion

We believe symbolic guardrails are an *overlooked but highly practical* mechanism for improving the safety and security of AI agents, especially in domain-specific, risk-averse business settings. Across existing benchmarks, we find that most evaluations do not specify concrete safety or security policies, but when policies are stated clearly, many requirements can be enforced symbolically, often through simple rather than complex methods. We further show that these guardrails can eliminate a large class of safety or security violations without reducing agent utility. Symbolic guardrails are not a complete solution: some requirements still depend on model-based judgment and neural guardrails. However, using probabilistic methods to enforce requirements that could instead be guaranteed symbolically introduces avoidable risk for limited benefit. We therefore argue that broader use of symbolic guardrails is a promising path toward deploying domain-specific AI agents in high-stakes settings with stronger safety and security guarantees.

7 Data Availability Statement

All materials associated with this study, including the literature review details, analysis notes, code, data, logs, configurations, and all model prompts, are anonymized and available at https://github.com/hyn0027/agent-symbolic-guardrails.

Acknowledgments

This work was supported in part by the National Science Foundation (award 2206859) and an unrestricted gift from Google's GARA. We would also like to thank Chenyang Yang, the SSSG attendees, and the S3C2 Quarterly Meeting attendees for their valuable feedback on this work.

References

[1] Nadya Abaev, Denis Klimov, Gerard Levinov, David Mimran, Yuval Elovici, and Asaf Shabtai. 2026. AgentGuardian: Learning Access Control Policies to Govern AI Agent Behavior. arXiv:2601.10440 [cs.CR] https://arxiv.org/abs/2601.10440

[2] 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*. https://openreview.net/forum?id=AC5n7xHuR1

[3] Anthropic. 2024. *Model Context Protocol*. https://github.com/modelcontextprotocol

[4] Anthropic. 2026. Claude. https://www.anthropic.com/claude.

[5] Anthropic. 2026. *Claude Code Overview*. https://docs.anthropic.com/en/docs/claude-code/overview Accessed: 2026-03-25.

[6] Anysphere, Inc. 2026. *Cursor: The AI Code Editor*. https://cursor.com

[7] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, et al. 2022. Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback. arXiv:2204.05862 [cs.CL] https://arxiv.org/abs/2204.05862

[8] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, et al. 2022. Constitutional AI: Harmlessness from AI Feedback. arXiv:2212.08073 [cs.CL] https://arxiv.org/abs/2212.08073

[9] Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. τ^2-Bench: Evaluating Conversational Agents in a Dual-Control Environment. arXiv:2506.07982 [cs.AI] https://arxiv.org/abs/2506.07982

[10] Tianyu Chen, Chujia Hu, Ge Gao, Dongrui Liu, Xia Hu, and Wenjie Wang. 2026. LPS-Bench: Benchmarking Safety Awareness of Computer-Use Agents in Long-Horizon Planning under Benign and Adversarial Scenarios. arXiv:2602.03255 [cs.AI] https://arxiv.org/abs/2602.03255

[11] Zhaorun Chen, Mintong Kang, and Bo Li. 2025. ShieldAgent: Shielding Agents via Verifiable Safety Policy Reasoning. In *Proceedings of the 42nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 267)*, Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (Eds.). PMLR, 8313–8344. https://proceedings.mlr.press/v267/chen25ae.html

[12] Sahana Chennabasappa, Cyrus Nikolaidis, Daniel Song, David Molnar, Stephanie Ding, Shengye Wan, Spencer Whitman, Lauren Deason, Nicholas Doucette, Abraham Montilla, Alekhya Gampa, Beto de Paola, Dominik Gabi, James Crnkovich, Jean-Christophe Testud, Kat He, Rashnil Chaturvedi, Wu Zhou, and Joshua Saxe. 2025. LlamaFirewall: An open source guardrail system for building secure AI agents. arXiv:2505.03574 [cs.CR] https://arxiv.org/abs/2505.03574

[13] 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

[14] Alejandro Cuadron, Pengfei Yu, Yang Liu, and Arpit Gupta. 2025. SABER: Small Actions, Big Errors - Safeguarding Mutating Steps in LLM Agents. arXiv:2512.07850 [cs.LG] https://arxiv.org/abs/2512.07850

[15] Jian Cui, Zichuan Li, Luyi Xing, and Xiaojing Liao. 2026. Maris: A Formally Verifiable Privacy Policy Enforcement Paradigm for Multi-Agent Collaboration Systems. arXiv:2505.04799 [cs.CR] https://arxiv.org/abs/2505.04799

[16] 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

[17] 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

[18] Dorothy E. Denning. 1976. A lattice model of secure information flow. *Commun. ACM* 19, 5 (May 1976), 236–243. doi:10.1145/360051.360056

[19] Aarya Doshi, Yining Hong, Congying Xu, Eunsuk Kang, Alexandros Kapravelos, and Christian Kästner. 2026. Towards Verifiably Safe Tool Use for LLM Agents. arXiv:2601.08012 [cs.SE] https://arxiv.org/abs/2601.08012

[20] Katia Romero Felizardo, Emilia Mendes, Marcos Kalinowski, Érica Ferreira Souza, and Nandamudi L Vijaykumar. 2016. Using forward snowballing to update systematic reviews in software engineering. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 1–6.

[21] Deep Ganguli, Liane Lovitt, Jackson Kernion, Amanda Askell, Yuntao Bai, Saurav Kadavath, Ben Mann, Ethan Perez, Nicholas Schiefer, Kamal Ndousse, Andy Jones, Sam Bowman, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Nelson Elhage, et al. 2022. Red Teaming Language Models to Reduce Harms: Methods, Scaling Behaviors, and Lessons Learned. arXiv:2209.07858 [cs.CL]

https://arxiv.org/abs/2209.07858

[22] Amelia Glaese, Nat McAleese, Maja Trębacz, John Aslanides, Vlad Firoiu, Timo Ewalds, Maribeth Rauh, Laura Weidinger, Martin Chadwick, Phoebe Thacker, Lucy Campbell-Gillingham, Jonathan Uesato, Po-Sen Huang, Ramona Comanescu, Fan Yang, Abigail See, Sumanth Dathathri, Rory Greig, Charlie Chen, Doug Fritz, Jaume Sanchez Elias, Richard Green, Soňa Mokrá, Nicholas Fernando, Boxi Wu, Rachel Foley, Susannah Young, Iason Gabriel, William Isaac, John Mellor, Demis Hassabis, Koray Kavukcuoglu, Lisa Anne Hendricks, and Geoffrey Irving. 2022. Improving alignment of dialogue agents via targeted human judgements. arXiv:2209.14375 [cs.LG] https://arxiv.org/abs/2209.14375

[23] Amr Gomaa, Ahmed Salem, and Sahar Abdelnabi. 2026. ConVerse: Benchmarking Contextual Safety in Agent-to-Agent Conversations. In *Findings of the Association for Computational Linguistics: EACL 2026*, Vera Demberg, Kentaro Inui, and Lluís Marquez (Eds.). Association for Computational Linguistics, Rabat, Morocco, 3246–3268. doi:10.18653/v1/2026.findings-eacl.170

[24] William G. J. Halfond, Jeremy Viegas, and Alessandro Orso. 2006. A Classification of SQL Injection Attacks and Countermeasures. In *International Symposium on Signals, Systems, and Electronics*. https://api.semanticscholar.org/CorpusID:5969227

[25] Hippocratic AI. 2026. Hippocratic AI: Home. https://hippocraticai.com/.

[26] Yining Hong, Christopher S. Timperley, and Christian Kästner. 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)*. 113–118. doi:10.1109/CAIN66642.2025.00021

[27] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2025. Model context protocol (mcp): Landscape, security threats, and future research directions. *ACM Transactions on Software Engineering and Methodology* (2025).

[28] Kung-Hsiang Huang, Akshara Prabhakar, Onkar Thorat, Divyansh Agarwal, Prafulla Kumar Choubey, Yixin Mao, Silvio Savarese, Caiming Xiong, and Chien-Sheng Wu. 2026. CRMArena-Pro: Holistic Assessment of LLM Agents Across Diverse Business Scenarios and Interactions. *Transactions on Machine Learning Research* (2026). https://openreview.net/forum?id=EPlpe3Fx1x

[29] Yixing Jiang, Kameron C Black, Gloria Geng, Danny Park, James Zou, Andrew Y Ng, and Jonathan H Chen. 2025. MedAgentBench: a virtual EHR environment to benchmark medical LLM agents. *Nejm Ai* 2, 9 (2025), AIdbp2500144.

[30] Christian Kaestner. 2025. *Machine Learning in Production: From Models to Products*. MIT Press.

[31] Adharsh Kamath, Sishen Zhang, Calvin Xu, Shubham Ugare, Gagandeep Singh, and Sasa Misailovic. 2025. Enforcing Temporal Constraints for LLM Agents. arXiv:2512.23738 [cs.PL] https://arxiv.org/abs/2512.23738

[32] Juhee Kim, Woohyuk Choi, and Byoungyoung Lee. 2025. Prompt Flow Integrity to Prevent Privilege Escalation in LLM Agents. arXiv:2503.15547 [cs.CR] https://arxiv.org/abs/2503.15547

[33] Johannes Kirmayr, Lukas Stappen, and Elisabeth André. 2026. CAR-bench: Evaluating the Consistency and Limit-Awareness of LLM Agents under Real-World Uncertainty. arXiv:2601.22027 [cs.AI] https://arxiv.org/abs/2601.22027

[34] Barbara Kitchenham, Stuart Charters, et al. 2007. Guidelines for performing systematic literature reviews in software engineering. (2007).

[35] Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, and Sushant Prakash. 2024. RLAIF vs. RLHF: scaling reinforcement learning from human feedback with AI feedback. In *Proceedings of the 41st International Conference on Machine Learning* (Vienna, Austria) *(ICML'24)*. JMLR.org, Article 1071, 28 pages.

[36] Juyong Lee, Dongyoon Hahm, June Suk Choi, W. Bradley Knox, and Kimin Lee. 2026. MobileSafetyBench: Evaluating Safety of Autonomous Agents in Mobile Device Control. *Proceedings of the AAAI Conference on Artificial Intelligence* 40, 44 (Mar. 2026), 37565–37573. doi:10.1609/aaai.v40i44.41090

[37] Jingdi Lei, Varun Gumma, Rishabh Bhardwaj, Seok Min Lim, Chuan Li, Amir Zadeh, and Soujanya Poria. 2026. OffTopicEval: When Large Language Models Enter the Wrong Chat, Almost Always! arXiv:2509.26495 [cs.AI] https://arxiv.org/abs/2509.26495

[38] Ido Levy, Ben wiesel, Sami Marreed, Alon Oved, Avi Yaeli, and Segev Shlomov. 2026. ST-WebAgentBench: A Benchmark for Evaluating Safety and Trustworthiness in Web Agents. In *The Fourteenth International Conference on Learning Representations*. https://openreview.net/forum?id=MuCDzH0ctf

[39] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.

[40] Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. 2024. Formalizing and Benchmarking Prompt Injection Attacks and Defenses. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, Philadelphia, PA, 1831–1847. https://www.usenix.org/conference/usenixsecurity24/presentation/liu-yupei

[41] Weidi Luo, Shenghong Dai, Xiaogeng Liu, Suman Banerjee, Huan Sun, Muhao Chen, and Chaowei Xiao. 2025. AGrail: A Lifelong Agent Guardrail with Effective and Adaptive Safety Detection. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, Vienna, Austria, 8104–8139. doi:10.18653/v1/2025.acl-long.399

[42] Xingjun Ma, Yifeng Gao, Yixu Wang, Ruofan Wang, Xin Wang, Ye Sun, Yifan Ding, Hengyuan Xu, Yunhao Chen, Yunhan Zhao, et al. 2026. Safety at scale: A comprehensive survey of large model and agent safety. *Foundations and Trends in Privacy and Security* 8, 3-4 (2026), 1–240.

[43] Microsoft. 2023. *Getting started with Copilot on Windows*. https://support.microsoft.com/en-us/topic/getting-started-with-copilot-on-windows-1159c61f-86c3-4755-bf83-7fbff7e0982d

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

[45] OpenAI. 2025. *ChatGPT agent*. https://chatgpt.com/features/agent

[46] OpenAI. 2026. ChatGPT. https://chatgpt.com/. Large language model accessed March 20, 2026.

[47] OpenClaw. 2026. OpenClaw — Personal AI Assistant. https://openclaw.ai/. Accessed: 2026-03-12.

[48] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 27730–27744. https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf

[49] Nils Palumbo, Sarthak Choudhary, Jihye Choi, Prasad Chalasani, and Somesh Jha. 2026. Policy Compiler for Secure Agentic Systems. arXiv:2602.16708 [cs.CR] https://arxiv.org/abs/2602.16708

[50] penligent. 2026. *Meta AI Alignment Director's OpenClaw Email Deletion Incident Exposes the Real Agent Safety Boundary*. https://www.penligent.ai/hackinglabs/meta-ai-alignment-directors-openclaw-email-deletion-incident-exposes-the-real-agent-safety-boundary/

[51] Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. 2022. Red Teaming Language Models with Language Models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 3419–3448. doi:10.18653/v1/2022.emnlp-main.225

[52] Yuxuan Qiao, Dongqin Liu, Hongchang Yang, Wei Zhou, and Songlin Hu. 2026. Agent Tools Orchestration Leaks More: Dataset, Benchmark, and Mitigation. arXiv:2512.16310 [cs.CR] https://arxiv.org/abs/2512.16310

[53] Traian Rebedea, Razvan Dinu, Makesh Narsimhan Sreedhar, Christopher Parisien, and Jonathan Cohen. 2023. NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications with Programmable Rails. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Yansong Feng and Els Lefever (Eds.). Association for Computational Linguistics, Singapore, 431–445. doi:10.18653/v1/2023.emnlp-demo.40

[54] Ravi Sandhu, David Ferraiolo, Richard Kuhn, et al. 2000. The NIST model for role-based access control: towards a unified standard. In *ACM workshop on Role-based access control*, Vol. 10.

[55] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 9895–9901. doi:10.18653/v1/2021.emnlp-main.779

[56] Yijia Shao, Tianshi Li, Weiyan Shi, Yanchen Liu, and Diyi Yang. 2024. PrivacyLens: Evaluating Privacy Norm Awareness of Language Models in Action. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 89373–89407. doi:10.52202/079017-2837

[57] Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. 2025. Progent: Programmable Privilege Control for LLM Agents. arXiv:2504.11703 [cs.CR] https://arxiv.org/abs/2504.11703

[58] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 8634–8652. https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf

[59] Sierra. 2026. Meet your agent. https://sierra.ai/product/meet-your-agent. Accessed: 2026-03-12.

[60] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 3008–3021. https://proceedings.neurips.cc/paper_files/paper/2020/file/1f89885d556929e98d3ef9b86448f951-Paper.pdf

[61] Rao Surapaneni, Miku Jha, Michael Vakoc, and Todd Segal. 2025. *Announcing the agent2agent protocol (A2A).* https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/

[62] John Viega and Gary R McGraw. 2001. *Building secure software: how to avoid security problems the right way.* Pearson Education.

[63] Sanidhya Vijayvargiya, Aditya Bharat Soni, Xuhui Zhou, Zora Zhiruo Wang, Nouha Dziri, Graham Neubig, and Maarten Sap. 2026. OpenAgentSafety: A Comprehensive Framework For Evaluating Real-World AI Agent Safety. In *The Fourteenth International Conference on Learning Representations.* https://openreview.net/forum?id=xggSxCFQbA

[64] David A Wagner, Jeffrey S Foster, Eric A Brewer, and Alexander Aiken. 2000. A first step towards automated detection of buffer overrun vulnerabilities.. In *NDSS*, Vol. 20. 0.

[65] 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

[66] Shouju Wang and Haopeng Zhang. 2026. MPCI-Bench: A Benchmark for Multimodal Pairwise Contextual Integrity Evaluation of Language Model Agents. arXiv:2601.08235 [cs.AI] https://arxiv.org/abs/2601.08235

[67] Fangzhou Wu, Ethan Cecchetti, and Chaowei Xiao. 2024. System-Level Defense against Indirect Prompt Injection Attacks: An Information Flow Control Perspective. arXiv:2409.19091 [cs.CR] https://arxiv.org/abs/2409.19091

[68] 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 by a Guard Agent via Knowledge-Enabled Reasoning. arXiv:2406.09187 [cs.LG] https://arxiv.org/abs/2406.09187

[69] Chenyang Yang, Yike Shi, Qianou Ma, Michael Xieyang Liu, Christian Kästner, and Tongshuang Wu. 2025. What Prompts Don't Say: Understanding and Managing Underspecification in LLM Prompts. arXiv:2505.13360 [cs.CL] https://arxiv.org/abs/2505.13360

[70] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. arXiv:2406.12045 [cs.AI] https://arxiv.org/abs/2406.12045

[71] 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

[72] Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, Rui Wang, and Gongshen Liu. 2024. R-Judge: Benchmarking Safety Risk Awareness for LLM Agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 1467–1490. doi:10.18653/v1/2024.findings-emnlp.79

[73] Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. 2024. Self-rewarding language models. In *Proceedings of the 41st International Conference on Machine Learning* (Vienna, Austria) *(ICML'24).* JMLR.org, Article 2389, 19 pages.

[74] 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

[75] Kaiwen Zhou, Shreedhar Jangam, Ashwin Nagarajan, Tejas Polu, Suhas Oruganti, Chengzhi Liu, Ching-Chen Kuo, Yuting Zheng, Sravana Narayanaraju, and Xin Eric Wang. 2026. SafePro: Evaluating the Safety of Professional-Level AI Agents. arXiv:2601.06663 [cs.AI] https://arxiv.org/abs/2601.06663