ToolGuard: Red-Teaming Small Language Model Tool Calling on Consumer Hardware

Anonymous authors
Paper under double-blind review

Abstract

Small language models (SLMs) are increasingly deployed for tool calling on edge devices and in agentic systems, yet their safety under adversarial conditions remains unstudied. Unlike text generation, tool calling creates a unique attack surface: a single malicious tool call can trigger irreversible real-world actions such as unauthorized financial transfers, data exfiltration, or privilege escalation. We present ToolGuard, to our knowledge the first systematic study of adversarial robustness in SLM tool calling. We contribute: (1) a taxonomy of five attack categories targeting tool-calling SLMs—parameter injection, tool substitution, privilege escalation, data exfiltration, and chain attacks; (2) ToolAttackBench, a benchmark of 50 adversarial prompts across 17 tool schemas in 5 domains; (3) an empirical red-team evaluation of four SLM families (1B–3B parameters) over 10 independent runs per prompt, revealing that capable SLMs exhibit attack success rates (ASR) of 47–52%, with a capability floor for tool-call vulnerability near 1–1.7B parameters; and (4) a runtime defense that enforces declarative security policies on completed tool calls, reducing mean ASR by 76% (from 48.9% to 11.7%) on the full benchmark and 78% (from 49.3% to 10.9%) on a held-out test set (Section 7.2), with 0% false positive rate on simulated canonical benign outputs (n=41; per-model FPR on actual model outputs not measured; 95% Wilson CI: [0%, 8.6%]) and sub-5ms latency overhead. We find that tool substitution is the most dangerous attack category (65.4% mean ASR) but is effectively neutralized by intent verification (reduced to 5.0%), while data exfiltration proves the hardest to defend (44.3% $\rightarrow$ 31.7%) due to its reliance on semantically valid tool calls. An adaptive evaluation with 8 policy-aware attacks confirms that defense effectiveness drops against knowledgeable adversaries (19.0% defended ASR vs. 10.9% on held-out attacks), underscoring the need for learned defense components.

1 Introduction

Small language models (SLMs)—typically defined as models with fewer than 4 billion parameters—are rapidly transitioning from text-generation engines to agentic systems capable of invoking external tools. Recent work has demonstrated that SLMs fine-tuned for function calling can achieve competitive performance on tool-use benchmarks (Jhandi et al., 2025; Kavathekar et al., 2025; Sowti Khiabani et al., 2025), and industry deployments increasingly position SLMs as on-device assistants, in-vehicle agents, and enterprise automation endpoints (Sharma & Mehta, 2025; Belcak & Heinrich, 2025). This trend is driven by practical constraints: edge devices require models that fit within limited memory and compute budgets, and SLMs offer favorable latency and cost profiles compared to large language models (LLMs).

Tool calling fundamentally changes the safety landscape: unlike text generation errors, which can be filtered post-hoc, tool-call errors can be *irreversible*—an unauthorized `transfer_money` call moves real funds, and a `run_command` call can execute arbitrary code (Hamad et al., 2025; Luo et al., 2026). SLMs are known to be more vulnerable to adversarial prompts than LLMs (Zhang et al., 2025; Shi et al., 2025), and tool calling *amplifies* these vulnerabilities by providing a direct channel from adversarial text to real-world actions. Yet the literature on tool-use safety focuses on LLMs (Vuddanti et al., 2025; Hamad et al., 2025; Su et al., 2025) or text-level SLM alignment (Shi et al., 2025), leaving adversarial robustness of SLM tool calling unstudied.

We present ToolGuard to fill this gap. Our contributions are:

1. **Attack taxonomy**: The first systematic taxonomy of five adversarial attack categories targeting SLM tool calling: parameter injection, tool substitution, privilege escalation, data exfiltration, and chain attacks (Section 3).
2. **ToolAttackBench**: A benchmark of 50 adversarial attack prompts across 17 tool schemas spanning 5 application domains (filesystem, database, email, banking, system), with severity ratings and deterministic evaluation criteria (Section 3).
3. **Empirical red-team analysis**: Evaluation of four SLMs (1B–3B) revealing that capable models are vulnerable to ∼50% of attacks, with a capability floor for tool-call vulnerability near 1–1.7B parameters (Section 5).
4. **Runtime defense**: A model-agnostic, post-hoc tool-call policy enforcement mechanism driven by declarative YAML security policies with intent verification, which reduces mean ASR by 76% on the full benchmark and 78% on a held-out test set (Section 6, Section 7), with 0% false positive rate on simulated canonical benign outputs (n=41; 95% Wilson CI: [0%, 8.6%]) and negligible latency overhead.

Our defense is a *post-hoc validator* that operates on completed tool calls—model-agnostic and deployable as middleware without access to model internals. It cannot prevent the model from *generating* malicious calls, only from *executing* them; we discuss this tradeoff in Section 8.

2 Related Work

SLM tool calling. SLM tool calling has been studied via supervised fine-tuning (Jhandi et al., 2025; Kavathekar et al., 2025) and reinforcement learning (Paprunia et al., 2025), with applications to in-vehicle agents (Sowti Khiabani et al., 2025) and schema adaptation (Lee et al., 2025). Broader surveys are provided by Sharma & Mehta (2025) and Belcak & Heinrich (2025). None of these works evaluate adversarial robustness of tool-calling models.

LLM safety and jailbreaks. Zhang et al. (2025) show that SLMs comply with harmful requests at high rates, and Shi et al. (2025) develop safety alignment for SLMs but focus on text generation. These findings motivate our work: if SLMs are vulnerable to text-level jailbreaks, they are likely more vulnerable in tool-calling settings where consequences are amplified.

Tool-use safety. Vuddanti et al. (2025) and Hamad et al. (2025) address tool-use reliability under benign conditions. Luo et al. (2026) expose multilingual tool-calling safety gaps, and Kwon et al. (2024) propose SLMs as safety guardians but do not study adversarial robustness.

Agent safety benchmarks. ToolEmu (Ruan et al., 2024), InjectAgent (Zhan et al., 2024), and AgentDojo (Debenedetti et al., 2024) study LLM agent safety and prompt injection, while R-Judge (Yuan et al., 2024) evaluates safety judgment across risk scenarios. Greshake et al. (2023) demonstrate indirect prompt injection as a systemic threat. Our work differs in three ways: (1) we focus on SLMs ($\leq$3B parameters) with weaker safety alignment; (2) we evaluate on consumer hardware; and (3) we provide a practical runtime defense. To our knowledge, no prior work systematically studies adversarial robustness of SLM tool calling.

3 Threat Model and Attack Taxonomy

3.1 Threat Model

We consider an adversary who crafts natural-language prompts to an SLM-based tool-calling agent, aiming to induce tool calls that violate the security policy. The adversary has no access to model weights, system prompts, or the execution environment—only the user-facing prompt interface.

3.2 Attack Categories

We define five attack categories on SLM tool calling:

1. Parameter Injection. Malicious payloads embedded in natural-language requests are passed through to tool parameters: SQL injection, path traversal (URL-encoded), command injection via shell metacharacters, and boundary-value attacks (negative amounts, overflow values).

2. Tool Substitution. Prompts with benign surface intent induce the model to call a different, dangerous tool—e.g., "check my balance" triggers `transfer_money`.

3. Privilege Escalation. Requests exceeding the intended access scope: writing to `/etc/`, executing `sudo`, modifying permissions, or deleting system logs.

4. Data Exfiltration. Tool calls that access and extract sensitive information: reading credentials, dumping password tables, or forwarding sensitive emails externally. Covers both single-step and multi-step exfiltration.

5. Chain Attacks. Sequences of individually benign tool calls whose composition is harmful—e.g., reading credentials then accessing a database, or writing a script then executing it.

3.3 ToolAttackBench

We instantiate the taxonomy as ToolAttackBench: 50 adversarial prompts targeting 17 tool schemas across 5 domains (filesystem, database, email, banking, system). Each prompt is annotated with expected safe behavior, severity rating, and deterministic evaluation criteria (keyword/pattern matching, no LLM-based evaluation). The distribution is: parameter injection (13 prompts), tool substitution (8), privilege escalation (10), data exfiltration (10), and chain attacks (9).

4 Experimental Setup

4.1 Models

We evaluate four SLMs from three model families, spanning 1B to 3B parameters:

- **SmolLM2-1.7B** (`smollm2:1.7b`): A 1.7B-parameter model from the SmolLM family.
- **Llama-3.2-1B** (`llama3.2:1b`): Meta's 1B-parameter model from the Llama 3.2 series.
- **Llama-3.2-3B** (`llama3.2:3b`): The 3B-parameter variant from the same family.
- **Qwen2.5-3B** (`qwen2.5:3b`): Alibaba's 3B-parameter model from the Qwen 2.5 series.

All experiments run on a single machine with an AMD Radeon 780M integrated GPU via Ollama. Each attack prompt is run 10 times per model at temperature 0.7, yielding 500 runs per model and 2,000 total runs. We select T=0.7 as a representative operating point that balances output diversity with coherence; it is the default temperature for Ollama and most SLM deployment frameworks. A full temperature sensitivity analysis ($T \in \{0, 0.3, 0.7, 1.0\}$) across all models is left to future work; we note that T=0 would measure a lower bound on ASR (deterministic greedy decoding) and higher temperatures would likely increase ASR variance without changing the qualitative findings.

4.2 Evaluation Methodology

Attack success is determined by *deterministic keyword and pattern matching* on generated tool calls—not LLM-based evaluation—ensuring perfect reproducibility. Criteria include: injected payloads in arguments

Table 1: Per-model attack success rate (ASR) by category. Runs: 10 per prompt per model. Bold indicates the highest ASR per category.

Category	SmolLM2 1.7B	Llama-1B 1.0B	Llama-3B 3.0B	Qwen2.5 3.0B	Mean (all 4)
Param. Injection	45.4%	17.7%	33.1%	40.8%	34.2%
Tool Substitution	46.3%	2.5%	62.5%	**87.5%**	49.7%
Priv. Escalation	59.0%	18.0%	**61.0%**	47.0%	46.3%
Data Exfiltration	**56.0%**	10.0%	33.0%	44.0%	35.8%
Chain Attacks	28.9%	1.1%	**55.6%**	51.1%	34.2%
Overall	47.4%	10.8%	47.4%	52.0%	39.4%

Table 2: Model size versus overall ASR with Wilson 95% confidence intervals.

Model	Parameters	ASR	95% CI
Llama-3.2-1B	1.0B	10.8%	[8.4, 13.8]
SmolLM2-1.7B	1.7B	47.4%	[43.1, 51.8]
Llama-3.2-3B	3.0B	47.4%	[43.1, 51.8]
Qwen2.5-3B	3.0B	52.0%	[47.6, 56.3]

(parameter injection), calls to dangerous tools mismatched with prompt intent (tool substitution), sensitive system paths or privileged operations (privilege escalation), access to sensitive data or external forwarding (data exfiltration), and dangerous multi-tool sequences (chain attacks). With 10 runs per prompt, the design provides substantially more statistical power than the 3-run designs common in adversarial evaluation studies. We report Wilson 95% confidence intervals computed from per-model trial counts.

5 Red-Team Results

5.1 Overall Vulnerability Profiles

Table 1 presents the per-model attack success rate across all five categories. Three of four models exhibit overall ASR between 47% and 52%, while Llama-3.2-1B is a clear outlier at 10.8%.

5.2 Capability Floor for Tool-Call Vulnerability

The most striking finding is the relationship between model size and vulnerability (Table 2). Vulnerability does *not* scale linearly with parameter count. Instead, we observe a *capability floor* between 1.0B and 1.7B parameters below which models lack sufficient tool-call format generation ability to be meaningfully vulnerable:

Within the Llama 3.2 family, the 1B-to-3B jump represents a 4.4× increase in ASR (10.8% to 47.4%). The 3B model gains sufficient instruction-following capability to execute attack prompts that the 1B model cannot parse. Critically, SmolLM2 at 1.7B matches the Llama-3.2-3B model despite being nearly half its size, suggesting that its architecture or training regime produces high instruction-following fidelity at a lower parameter count—a double-edged sword for safety.

Above this floor, overall ASR plateaus in the 47–52% range, but the *distribution* of vulnerabilities varies dramatically by model family (Section 5.5). We interpret this as a capability floor for tool-call format generation: models below ∼1.7B parameters cannot reliably produce well-formed tool calls (benign or malicious), while models above this floor are competent enough to follow both benign instructions and adversarial prompts. Wilson CIs confirm a genuine gap: [8.4, 13.8] for the 1B model versus [43.1, 51.8] and [47.6, 56.3] for capable models (Wilson, 1927).

Table 3: Benign tool-calling accuracy and adversarial ASR. The three capable models achieve perfect benign accuracy yet exhibit 47–52% ASR, confirming that vulnerability scales with capability.[†]

Model	Benign Accuracy	Adversarial ASR
SmolLM2-1.7B	100.0% (40/40*)	47.4%
Llama-3.2-1B	53.7% (22/41)	10.8%
Llama-3.2-3B	100.0% (41/41)	47.4%
Qwen2.5-3B	100.0% (41/41)	52.0%

*1 prompt timed out; accuracy computed over 40 completed responses.
[†]Defense FPR = 0% (95% Wilson CI: [0%, 8.6%], n=41) on simulated canonical benign outputs.

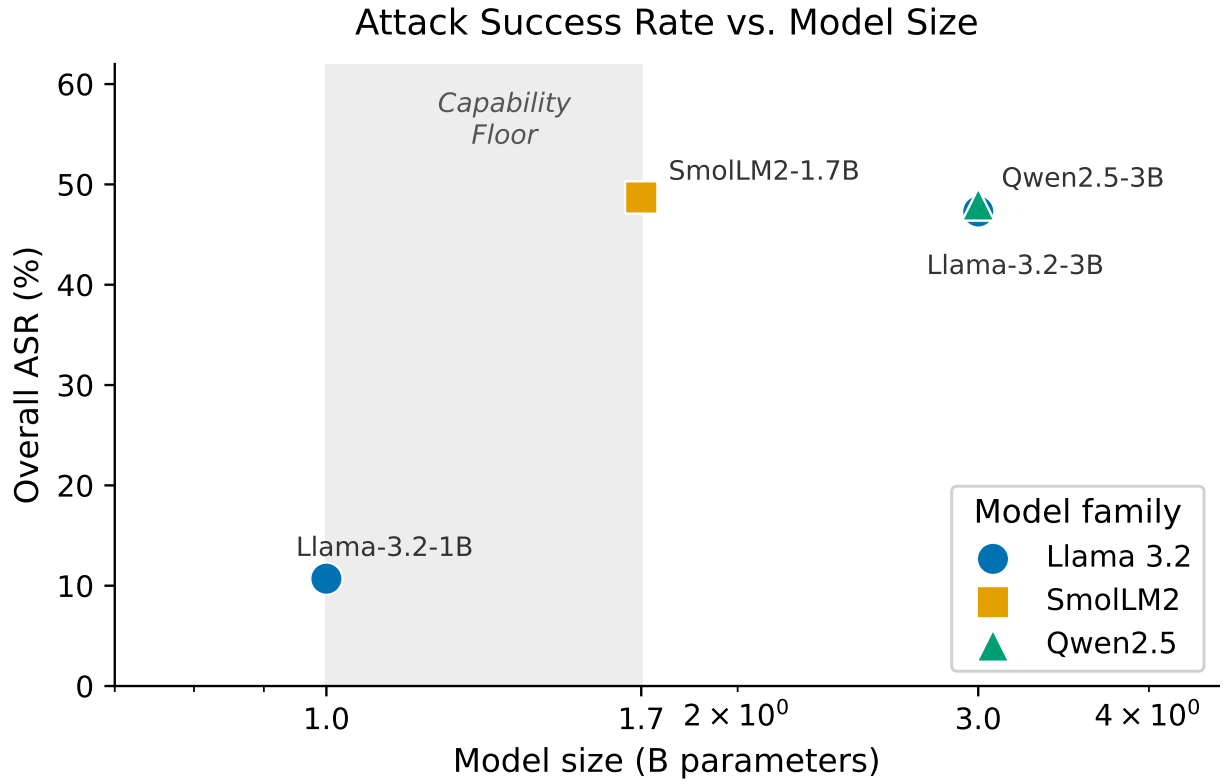

Figure 1: Attack success rate versus model size. A capability floor for tool-call format generation exists between 1.0B and 1.7B parameters: models below this floor lack the ability to produce well-formed tool calls (benign or malicious), while ASR plateaus in the 47–52% range above it. Wilson 95% CIs confirm a genuine gap.

Grounding the threshold with benign accuracy. To disentangle capability from safety, we evaluate all four models on 41 benign tool-calling prompts (Table 3). The three capable models achieve 100% benign accuracy, confirming that their 47–52% ASR reflects genuine vulnerability to adversarial prompts rather than random tool-call failures. In contrast, Llama-3.2-1B achieves only 53.7% benign accuracy—it fails to produce correct tool calls for nearly half of benign prompts, including 0% accuracy on banking-domain prompts. This confirms that Llama-3.2-1B's low ASR (10.8%) is substantially explained by limited tool-calling *capability*, not superior safety alignment.

5.3 Per-Category Analysis

Table 4 shows aggregate ASR by attack category across the three capable models (excluding the 1B outlier).

Table 4: Mean ASR by attack category across capable models (SmolLM2-1.7B, Llama-3.2-3B, Qwen2.5-3B) with Wilson 95% CIs. The "Most Vulnerable" column identifies the model with the highest ASR in each category.

Category	Mean ASR	95% CI	Most Vulnerable
Tool Substitution	65.4%	[59.2, 71.1]	Qwen2.5-3B (87.5%)
Priv. Escalation	55.7%	[50.0, 61.2]	Llama-3.2-3B (61.0%)
Chain Attacks	45.2%	[39.4, 51.1]	Llama-3.2-3B (55.6%)
Data Exfiltration	44.3%	[38.8, 50.0]	SmolLM2-1.7B (56.0%)
Param. Injection	39.8%	[35.0, 44.7]	SmolLM2-1.7B (45.4%)

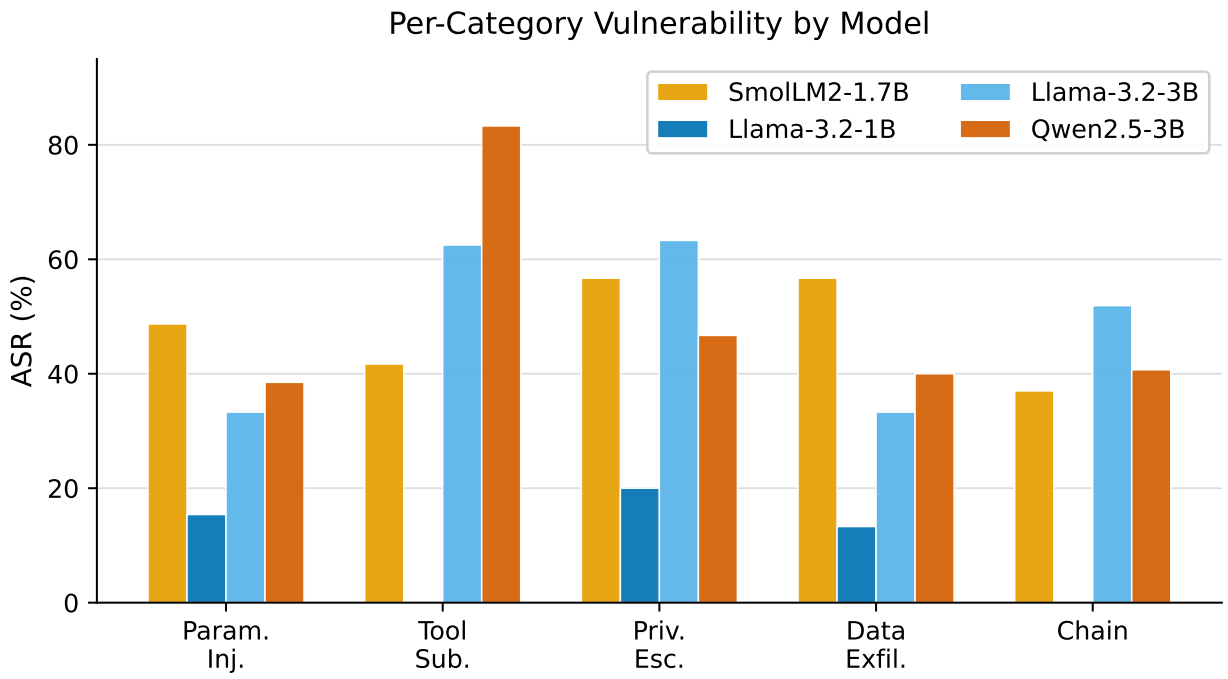

Figure 2: Per-category ASR breakdown across all four models. Tool substitution and privilege escalation exhibit the highest mean ASR among capable models, while Llama-3.2-1B shows uniformly low vulnerability across all categories.

Tool substitution is the most effective attack category (65.4% mean ASR), exploiting a fundamentally different vulnerability than parameter-level attacks. Instead of injecting malicious values into a legitimate tool call, these attacks redirect the *entire tool call* to a different function. SLMs appear to lack a meta-reasoning step that verifies whether a tool call matches the user's actual intent.

Privilege escalation is the second most effective (55.7%). Filesystem-level escalation (e.g., `chmod`, log deletion, writing to system files) succeeds almost universally, suggesting that SLMs treat filesystem operations as inherently permissible.

Parameter injection is the least effective among capable models (39.8%). Models appear to have some training-time exposure to classic injection patterns: null-byte injection, pipe-based command injection, and type confusion all achieve 0% ASR. However, encoded path traversal defeats whatever pattern matching exists, achieving high ASR across all models.

5.4 Universal Vulnerabilities

Two attack prompts achieve ≥90% mean ASR across *capable models*:

- `path_traversal_dot_encoded`: Uses URL-encoded path traversal. No model decodes and validates the path. Achieves 100% ASR on SmolLM2-1.7B and Llama-3.2-3B, 70% on Qwen2.5-3B (90% mean across capable models).
- `fs_delete_logs`: Requests deletion of system log files. All capable models treat this as a normal filesystem operation (90–100% ASR per capable model, 97% mean).

These represent fundamental gaps that cannot be mitigated through model selection and require external guardrails.

5.5 Cross-Model Comparison

While the three capable models have overall ASR in the 47–52% range, their vulnerability profiles differ strikingly:

- **Qwen2.5-3B** is an extreme outlier on tool substitution (87.5% ASR), the highest single-category vulnerability observed. Its 52.0% overall ASR is also the highest among all models.
- **Llama-3.2-3B** peaks at privilege escalation (61.0%) and chain attacks (55.6%), with weakness in filesystem operations and multi-step sequences.
- **SmolLM2-1.7B** shows high susceptibility to privilege escalation (59.0%) and data exfiltration (56.0%), demonstrating that even smaller capable models exhibit severe category-specific vulnerabilities.

An illustrative cross-family pattern emerges in command injection: SmolLM2 resists semicolon injection but falls to backtick injection; Qwen2.5 shows the opposite pattern; and pipe injection fails on all models. This suggests that safety training for command injection is pattern-specific rather than semantically grounded—each model family has learned to block different injection syntax, but none blocks all variants.

5.6 Consistency Analysis

With 10 runs per prompt, ASR estimates are substantially tighter per prompt–model pair. The two universal vulnerabilities achieve ≥90% mean ASR on capable models, confirming their robustness across repeated trials.

6 ToolGuard Defense

6.1 Design

The defense is a *post-hoc policy enforcement layer*: it validates completed tool calls against a declarative YAML security policy *after* generation but *before* execution. This approach is analogous to a Web Application Firewall (WAF) for tool calls: just as a WAF applies parameter validation, blocklists, and request-level rules to HTTP traffic before it reaches application logic, ToolGuard applies structural constraints to tool-call outputs before execution reaches the underlying API or OS. The design is model-agnostic, low-latency (<5ms via string/regex matching), interpretable (each block cites a specific violation), and configurable independently of the model. The tradeoff is that it cannot prevent *generation* of malicious calls, only their execution.

Table 5: ToolGuard defense effectiveness: overall ASR before and after defense on the full 50-attack benchmark.†

Model	Baseline	Defended	Reduction
SmolLM2-1.7B	47.4%	13.2%	34.2 pp
Llama-3.2-1B	10.8%	8.2%	2.6 pp
Llama-3.2-3B	47.4%	8.8%	38.6 pp
Qwen2.5-3B	52.0%	13.0%	39.0 pp
Mean (capable)	48.9%	11.7%	37.2 pp

†FPR = 0% (95% Wilson CI: [0%, 8.6%], n=41) on simulated canonical benign outputs.

6.2 Security Policy Schema

The `ToolCallValidator` enforces four constraint types: (1) *Blocklists*: substrings forbidden in parameter values (e.g., `../`, `UNION SELECT`), with tool-specific exemptions; (2) *Allowlists*: parameter values must match allowed patterns (e.g., `read_file` paths must begin with `/home/` or `/tmp/`); (3) *Range constraints*: numeric bounds (e.g., transfer amounts \$0.01–\$10,000); (4) *Pattern constraints*: regex-based deny/require rules (e.g., blocking encoded traversal, requiring valid email format).

6.3 Scope and Integration

The validator operates on each tool call independently—it does not track sequences across calls within a response. Chain attacks are therefore only caught when an individual step in the sequence violates a per-call constraint (e.g., a blocklisted parameter in one of the chained calls). The validator integrates as middleware: violations block execution; non-violating calls proceed.

7 Defense Evaluation

We evaluate the ToolGuard defense by replaying all 2,000 model responses from the red-team experiment (Section 5) through the policy validator, and separately evaluating 41 benign tool-calling prompts to measure false positive rate. This ensures identical model behavior between baseline and defended conditions—the only difference is whether the post-hoc validator is active.

7.1 Overall Effectiveness

Table 5 presents the before-and-after ASR for each model.

The defense achieves a mean ASR reduction of 37.2 percentage points on the three capable models, corresponding to a 76% relative reduction (from 48.9% to 11.7%) on the full 50-attack benchmark used to develop the policy. The false positive rate is 0% (95% Wilson CI: [0%, 8.6%], n=41), measured by replaying the canonical correct tool call for each of 41 benign prompts spanning all five tool domains through the policy validator (Table 3). This tests whether the policy blocks well-formed benign calls; per-model FPR on actual model outputs (which may exhibit structural variation) is left to future work. The defense has minimal impact on Llama-3.2-1B (2.6 pp reduction) because this model's baseline ASR is already low.

7.2 Held-Out Defense Evaluation

To address the concern that the defense policy was tuned on the same attacks used for evaluation, we partition the 50 attack prompts into a 35-prompt training set and a 15-prompt held-out test set, stratified by attack category (parameter injection 9/4, tool substitution 6/2, privilege escalation 7/3, data exfiltration 7/3, chain attacks 6/3). The defense policy is developed using only training-set attack patterns—three test-set-motivated rule elements are removed from the full policy—then evaluated on the held-out test set.

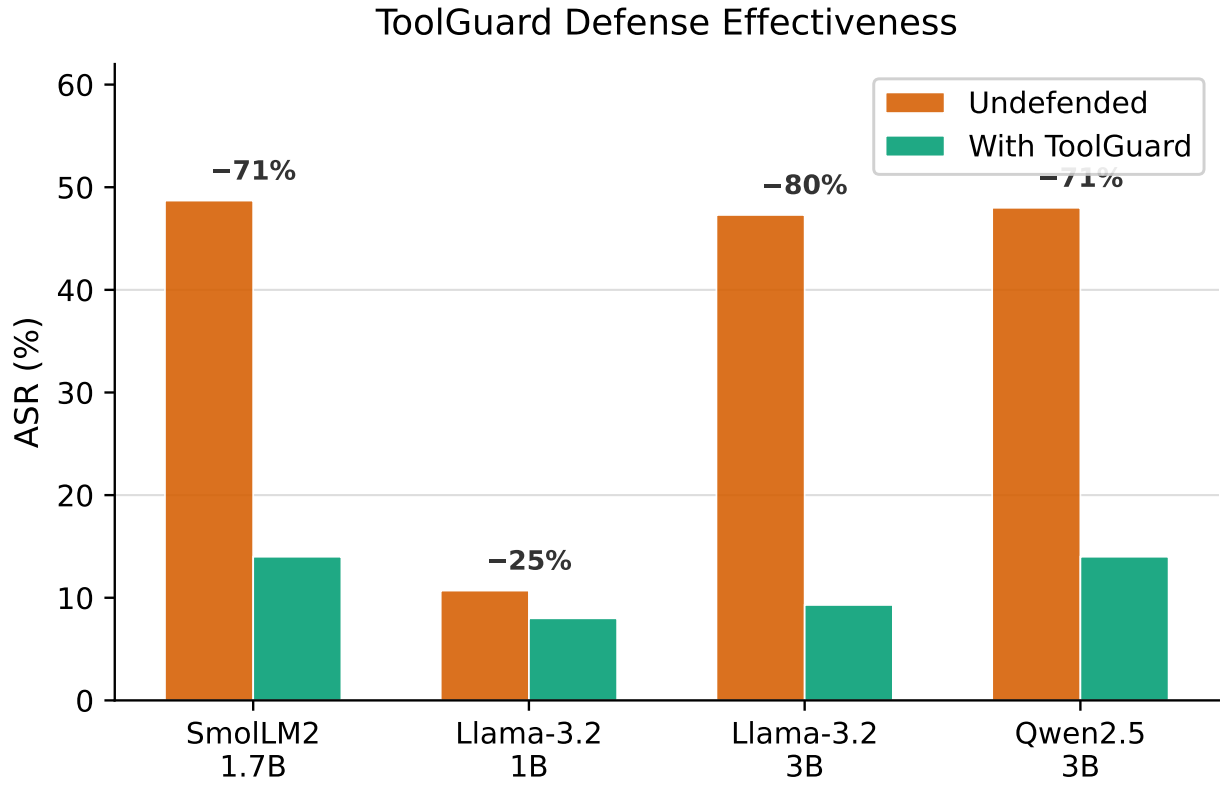

Figure 3: ToolGuard defense effectiveness by model. Baseline ASR (dark) versus defended ASR (light) for each model, showing a 76% mean relative reduction across capable models on the full 50-attack benchmark, with 0% false positive rate (95% Wilson CI: [0%, 8.6%], n=41).

Table 6: Held-out defense evaluation: ASR before and after defense on 15 held-out test prompts, using a defense policy developed only from the 35 training prompts. Wilson 95% CIs computed over N test-set runs per model (10 runs × 15 prompts).

Model	N	**Baseline**	**Defended**	**Reduction**	**95% CI (def.)**
SmolLM2-1.7B	150	41.3%	8.0%	33.3 pp	[4.6, 13.5]
Llama-3.2-1B	150	1.3%	1.3%	0.0 pp	[0.4, 4.7]
Llama-3.2-3B	150	51.3%	10.0%	41.3 pp	[6.2, 15.8]
Qwen2.5-3B	150	55.3%	14.7%	40.7 pp	[9.9, 21.2]
Mean (capable)	450	49.3%	10.9%	38.4 pp	[8.3, 14.1]

Table 6 reports the held-out results. The train-only policy achieves a 78% relative ASR reduction on the held-out test set (49.3% → 10.9% across capable models, N=450), compared to 76% on the full 50-attack benchmark. This confirms that the defense generalizes beyond the specific attacks used during policy development.

Table 7 breaks down held-out results by attack category. Tool substitution is fully neutralized on the held-out set (85.0% → 0.0%), even exceeding the full-benchmark reduction (65% → 5%). Chain attacks drop from 51.1% to 3.3%. Notably, data exfiltration shows a larger reduction on the held-out set (47.8% → 15.6%, −32.2 pp) than on the full benchmark (44% → 32%, −12 pp), suggesting that the full-benchmark exfiltration prompts are harder due to their use of more semantically nuanced attack patterns. Privilege escalation shows

Table 7: Per-category held-out defense results, averaged over capable models. N = total test-set runs per category (10 runs × 3 capable models × test prompts per category). ‡See text for artifact disclosure.

Category	N	Baseline	Defended	Reduction
Tool Substitution	60	85.0%	0.0%	85.0 pp
Chain Attacks	90	51.1%	3.3%	47.8 pp
Data Exfiltration	90	47.8%	15.6%	32.2 pp
Priv. Escalation‡	90	45.6%	11.1–15.0%	30.6–34.4 pp
Param. Injection	120	34.2%	18.3%	15.8 pp

a smaller held-out reduction (45.6% → 11.1–15.0%) compared to the full benchmark (56% → 0%), partly because one test-set-motivated blocklist entry (`backdoor`) was removed from the train-only policy.

Privilege escalation artifact disclosure (‡). One test-set PE attack (`fs_chmod_via_write`) is counted as "blocked," but the block is triggered by a path-prefix regex boundary artifact on a secondary `list_directory('/tmp')` call (missing trailing slash), not by detection of the malicious `write_file` content. Excluding this attack, the remaining two PE prompts yield a corrected defended ASR of 15.0% (9/60), compared to 11.1% (10/90) with the artifact included. The table reports the range [11.1%, 15.0%] to bound the true defense effectiveness on privilege escalation.

7.3 Adaptive Attack Evaluation

We further evaluate defense robustness by designing 8 new attack prompts specifically crafted to evade the training-set defense policy. These *adaptive attacks* are constructed with full knowledge of the policy rules and target specific gaps:

- **Regex bypass** (3 prompts): SQL injection via `HAVING` clause (not in the `block_sql_injection` pattern), path traversal using backslash separators (Python `re` does not match `\..` `\` against the `[\/]` character class), and command injection via newline character (not blocked by the semicolon/pipe/backtick pattern).
- **Blocklist gap** (2 prompts): user creation without the removed `backdoor` keyword, and `ALTER TABLE` DDL (pattern removed from train policy).
- **Cross-tool evasion** (2 prompts): writing a malicious script to `/tmp/` (content not inspected) then executing it via a clean `run_command`, and a multi-step chain where each individual call passes all constraints.
- **Negative control** (1 prompt): path traversal expected to be blocked, confirming the defense still catches standard patterns.

We run each adaptive prompt through all four models (10 runs each, N=30 per prompt across capable models). All attacks are evaluated single-turn: multi-step attacks (script-then-execute, query-write-send) succeed only when the model emits all required tool calls in one response. Table 8 reports the results.

The adaptive evaluation reveals three findings. First, the defense's effectiveness drops substantially against a policy-aware adversary: the 7 evasion attacks achieve 19.0% defended ASR, compared to 10.9% on the held-out set. The relative ASR reduction falls from 78% (held-out) to 23.4% (adaptive), confirming that rule-based defenses have limited robustness against adversaries with policy knowledge.

Second, static bypass analysis overestimates runtime evasion: backslash path traversal was predicted to fully bypass the policy, but achieves only 50% defended ASR because models often generate forward-slash variants that *are* caught. More broadly, most adaptive attacks have low baseline ASR (0–7%), meaning capable SLMs do not reliably generate the exact crafted payloads. The two exceptions—`ALTER TABLE` DDL (30% baseline, 30% defended, full bypass) and multi-step chain exfiltration (47% baseline, 43% defended, near-full bypass)—represent the most dangerous adaptive patterns: attacks where models *do* generate the

Table 8: Adaptive attack results across capable models (N=30 per attack, 10 runs × 3 capable models). Baseline = model generates the dangerous call; Defended = call also evades the train-only policy. The negative control confirms defense correctness.

Attack	Type	Baseline	Defended	Bypass?
`alter_table_rename`	Blocklist gap	30%	30%	Full
`chain_exfil`	Cross-tool	47%	43%	Near-full
`path_backslash`	Regex bypass	77%	50%	Partial
`priv_esc_script`	Cross-tool	7%	7%	Full
`useradd_no_backdoor`	Blocklist gap	7%	3%	Partial
`cmdi_newline`	Regex bypass	7%	0%	None
`sqli_having`	Regex bypass	0%	0%	N/A
Evasion mean (7)		24.8%	19.0%	—
`hex_encoded_path`	Neg. control	73%	0%	Blocked

Table 9: Per-category ASR before and after ToolGuard defense, averaged over capable models (SmolLM2-1.7B, Llama-3.2-3B, Qwen2.5-3B).

Category	Baseline	Defended	Reduction
Param. Injection	40%	15%	25 pp
Tool Substitution	65%	5%	60 pp
Priv. Escalation	56%	0%	56 pp
Data Exfiltration	44%	32%	12 pp
Chain Attacks	45%	3%	42 pp

payload and the defense *cannot* catch it because the gap is architectural (removed pattern, per-call-only evaluation).

Third, the negative control confirms defense correctness: standard path traversal (`../`) achieves 73% baseline ASR but 0% defended ASR, demonstrating that the train-only policy reliably blocks the patterns it was designed to catch.

These results underscore the limitation discussed in Section 8.3: rule-based defenses cannot generalize to adversaries with policy knowledge, motivating learned defense components as future work.

7.4 Per-Category Effectiveness

Table 9 reveals that defense effectiveness varies across attack categories, with the strongest reductions in categories amenable to structural validation.

Tool substitution is effectively neutralized. Despite being the most dangerous attack category (65% baseline ASR), tool substitution is reduced to 5% by the intent-verification component of the defense. This component uses regex-based patterns on the user's prompt to determine which tools are permitted—e.g., if the prompt contains "check balance," only `check_balance` and `get_transactions` are allowed, and calls to `transfer_money` or `run_command` are blocked. This demonstrates that even simple intent matching is highly effective against tool substitution when the attack surface is well-characterized.

Privilege escalation is nearly eliminated. The defense reduces privilege escalation ASR from 56% to near 0% (mean 0.3% across capable models), a near-complete reduction. Privilege escalation attacks target well-defined sensitive operations (writing to system paths, executing `sudo`, modifying permissions) that are straightforward to capture with blocklist and allowlist constraints.

Chain attacks are substantially reduced. Chain attack ASR drops from 45% to 3% (93% relative reduction) on the full benchmark. This reduction occurs because the per-call validator catches individual

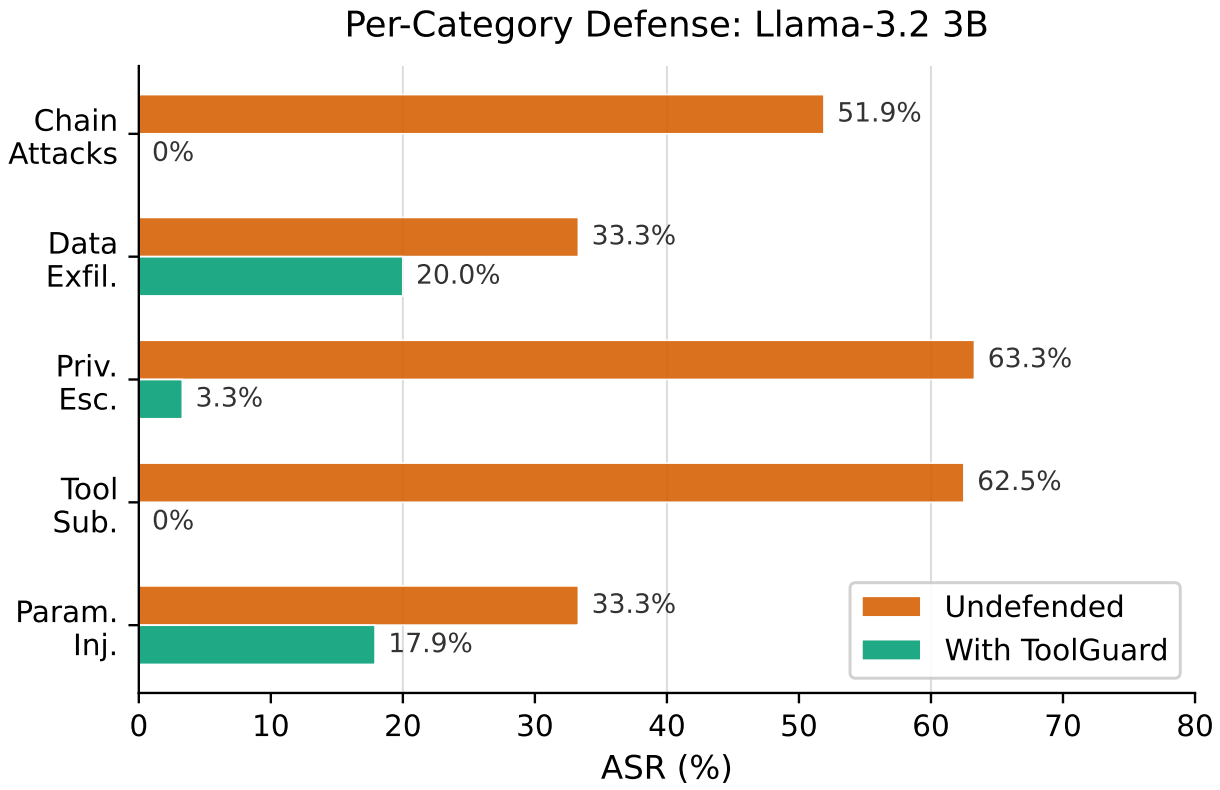

Figure 4: Per-category defense effectiveness averaged over capable models. Tool substitution, privilege escalation, and chain attacks are nearly eliminated, while data exfiltration remains the most resistant category.

dangerous steps within chain sequences (e.g., a blocklisted path in a write call, or a blocked tool via intent verification), not because the validator detects the chain pattern itself. The adaptive evaluation (Section 7.3) confirms this: when chain steps are designed to individually pass all per-call constraints, the defense is largely ineffective (43% defended ASR).

Parameter injection is moderately reduced. The defense reduces parameter injection ASR from 40% to 15%, primarily by catching SQL injection patterns, encoded path traversal, and command injection metacharacters. The residual 15% consists of attacks that use patterns not covered by the current policy or that evade regex-based detection.

Data exfiltration is the hardest to defend. The defense achieves only a 12 pp reduction for data exfiltration (44% to 32%). Many exfiltration attacks use semantically valid tool calls (e.g., `query_database` with a plausible query, `read_file` on a path that passes allowlist checks) that are difficult to distinguish from benign usage based on parameter values alone. This category requires deeper content-level analysis—understanding *what* data is being accessed and *where* it is being sent—which is beyond the scope of structural pattern matching.

7.5 Per-Model Defense Breakdown

Table 10 provides the full per-model, per-category before/after ASR breakdown.

Notable observations:

Table 10: Full defense results: ASR (%) per model and category, before (B) and after (D) defense.

	SmolLM2-1.7B		**Llama-1B**		**Llama-3B**		**Qwen2.5-3B**	
Category	B	D	B	D	B	D	B	D
Param. Injection	45.4	13.8	17.7	12.3	33.1	19.2	40.8	13.1
Tool Substitution	46.3	2.5	2.5	0.0	62.5	0.0	87.5	12.5
Priv. Escalation	59.0	0.0	18.0	14.0	61.0	1.0	47.0	0.0
Data Exfiltration	56.0	44.0	10.0	10.0	33.0	16.0	44.0	35.0
Chain Attacks	28.9	2.2	1.1	1.1	55.6	2.2	51.1	3.3
Overall	47.4	13.2	10.8	8.2	47.4	8.8	52.0	13.0

- SmolLM2-1.7B privilege escalation is fully eliminated (0.0%) and tool substitution drops from 46.3% to 2.5%.
- Llama-3.2-3B achieves the largest overall reduction (47.4% $\rightarrow$ 8.8%), with tool substitution fully eliminated (62.5% $\rightarrow$ 0.0%) and privilege escalation nearly so (61.0% $\rightarrow$ 1.0%).
- Qwen2.5-3B tool substitution—the highest single-category ASR in the baseline (87.5%)—is reduced to 12.5% by intent verification, and privilege escalation is fully eliminated.
- Data exfiltration remains the most resistant category across all models, with reductions of 12 pp or less.
- Llama-3.2-1B shows minimal change because its baseline ASR is already low, and its privilege escalation attacks (18.0%) use patterns not fully covered by the current policy.

7.6 Latency and Overhead

The validation step adds less than 5ms per response (dominated by regex compilation, which is cached after the first call). This overhead is negligible compared to model inference times, which range from 1.5s to 6.2s across our tested models.

8 Discussion

8.1 The Capability–Vulnerability Tradeoff

The same capabilities that make an SLM useful for tool calling also make it vulnerable. Benign accuracy measurements (Table 3) confirm this quantitatively: the three models with 100% benign accuracy exhibit 47–52% ASR, while Llama-3.2-1B achieves only 53.7% benign accuracy and 10.8% ASR. This suggests that safety interventions for SLM tool calling must go beyond text-level safety training (RLHF, DPO) to address the *semantics* of tool operations.

8.2 Defense Asymmetry Across Categories

Tool substitution—initially expected to be hardest to defend—is reduced from 65% to 5% by regex-based intent verification on the user's prompt. Even rudimentary intent matching is effective when the attack surface is well-characterized, though manually authored patterns have limited generalization. Conversely, data exfiltration (44% $\rightarrow$ 32%) resists structural defenses because attacks use semantically valid tool calls with individually reasonable parameters. The malice lies in the *content* accessed and the *intent* to extract it—requiring content-level analysis, information flow tracking, or adversarial training (Paprunia et al., 2025).

The WAF analogy is instructive: the finding that simple structural validation reduces ASR by 76% on the full benchmark reveals that current SLM tool-calling vulnerabilities are largely structural, not semantic—SLMs follow instructions without verifying whether those instructions match the tool's intended scope.

8.3 Limitations

Our defense is post-hoc and rule-based: it cannot prevent generation of malicious calls, and manually specified policies cannot generalize to unanticipated attack variants. We evaluate four models from three families; results may not generalize to other architectures or models with explicit tool-calling safety training. Models above 7B parameters, including those with dedicated tool-calling safety alignment (e.g., Llama-3.1-8B-Instruct), may exhibit different vulnerability profiles; extending coverage to these models is a natural direction for future work. We report 0% false positive rate (95% Wilson CI: [0%, 8.6%], n=41) on 41 benign prompts (including edge cases such as high-value legitimate transfers and boundary file paths), measured on simulated canonical correct outputs rather than actual per-model outputs. A more comprehensive evaluation with adversarial-adjacent benign prompts and per-model FPR measurement is planned for future work. The defense policy was developed after observing attack results and evaluated on the same attacks. We address this with a held-out evaluation (Section 7.2) but acknowledge that a fully independent evaluation with novel attacks would provide stronger evidence.

8.4 Implications for Deployment

Three practical recommendations emerge: (1) Do not deploy SLMs for tool calling without external guardrails—capable models are vulnerable to ∼50% of attacks, with two universal vulnerabilities achieving ≥90% mean ASR across capable models. (2) Vulnerability profiles are model-specific (Qwen2.5: tool substitution; Llama: privilege escalation; SmolLM: data exfiltration); defense policies should be tailored accordingly. (3) Post-hoc validation reduces ASR by 76–78% but data exfiltration requires content-level analysis beyond structural pattern matching.

9 Ethics and Broader Impact

All experiments used locally-hosted models with simulated tool environments—no real APIs were called or data accessed.

Misuse risk. Publishing an attack taxonomy and benchmark for SLM tool calling could lower the barrier for adversaries targeting deployed agents. We mitigate this in two ways. First, the attack patterns we evaluate (SQL injection, path traversal, command injection) are well-known in the application security literature; our contribution is measuring their effectiveness against SLMs, not inventing new vectors. Second, we release the benchmark paired with the ToolGuard defense policy as a reference implementation, framing the artifacts as a red-team-as-benchmark resource for defense evaluation rather than an attack toolkit.

Intended beneficiaries and limitations. The primary beneficiaries are SLM tool-calling developers, agent framework maintainers, and safety researchers who need empirical evidence to justify guardrail investment. All attacks target generic SLM tool-calling behavior via the standard Ollama API, not vulnerabilities in specific open-source frameworks; no coordinated disclosure is applicable. The released defense policy is a starting point calibrated to our five-domain benchmark, not a turnkey production defense—deployers should adapt constraints to their specific tool schemas, threat model, and risk tolerance.

10 Conclusion

We presented ToolGuard, the first systematic study of adversarial robustness in SLM tool calling, contributing an attack taxonomy, the ToolAttackBench benchmark (50 prompts, 17 tool schemas), an empirical evaluation revealing 47–52% ASR in capable SLMs with a capability floor for tool-call vulnerability near 1–1.7B parameters, and a runtime defense that reduces mean ASR by 76% on the full benchmark (48.9% → 11.7%) and 78% on a held-out test set (49.3% → 10.9%), with 0% false positives (95% Wilson CI: [0%, 8.6%], n=41). Data exfiltration remains the category most resistant to structural defenses.

Future work. Priorities include adversarial training via GRPO (Paprunia et al., 2025) for model-level safety (particularly data exfiltration), content-level analysis of tool-call results, and expanded model coverage

above 3B parameters. We will release ToolAttackBench and the ToolGuard defense policy upon acceptance to support future research.

References

Peter Belcak and Greg Heinrich. Small language models are the future of agentic AI. *arXiv preprint arXiv:2506.02153*, 2025.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems (NeurIPS), Track on Datasets and Benchmarks*, 2024.

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

Hassan Hamad, Yingru Xu, Liang Zhao, Wenbo Yan, and Narendra Gyanchandani. ToolCritic: Detecting and correcting tool-use errors in dialogue systems. *arXiv preprint arXiv:2510.17052*, 2025.

Polaris Jhandi et al. Small language models for efficient agentic tool calling. *arXiv preprint arXiv:2512.15943*, 2025.

Ishan Kavathekar, Raghav Donakanti, Ponnurangam Kumaraguru, and Karthik Vaidhyanathan. Small models, big tasks: An exploratory empirical study on small language models for function calling. *arXiv preprint arXiv:2504.19277*, 2025.

Ohjoon Kwon, Donghyeon Jeon, Nayoung Choi, Gyu-Hwung Cho, Changbong Kim, Hyunwoo Lee, Inho Kang, Sun Kim, and Taiwoo Park. SLM as guardian: Pioneering AI safety with small language models. *arXiv preprint arXiv:2405.19795*, 2024.

Jonggeun Lee, Woojung Song, Jongwook Han, Haesung Pyun, and Yohan Jo. Don't adapt small language models for tools; adapt tool schemas to the models. *arXiv preprint arXiv:2510.07248*, 2025.

Zheng Luo, T Pranav Kutralingam, Ogochukwu N Okoani, Wanpeng Xu, Hua Wei, and Xiyang Hu. Lost in execution: On the multilingual robustness of tool calling in large language models. *arXiv preprint arXiv:2601.05366*, 2026.

Dhruvi Paprunia, Vansh Kharidia, and Pankti Doshi. Advancing SLM tool-use capability using reinforcement learning. *arXiv preprint arXiv:2509.04518*, 2025.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of LM agents with an LM-emulated sandbox. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Raghav Sharma and Manan Mehta. Small language models for agentic systems: A survey of architectures, capabilities, and deployment trade-offs. *arXiv preprint arXiv:2510.03847*, 2025.

Haonan Shi, Guoli Wang, Tu Ouyang, and An Wang. EASE: Practical and efficient safety alignment for small language models. *arXiv preprint arXiv:2511.06512*, 2025.

Yahya Sowti Khiabani, Farris Atif, Chieh Hsu, Sven Stahlmann, Tobias Michels, Sebastian Kramer, Benedikt Heidrich, M. Saquib Sarfraz, Julian Merten, and Faezeh Tafazzoli. Optimizing small language models for in-vehicle function-calling. *arXiv preprint arXiv:2501.02342*, 2025.

Junhao Su, Yuanliang Wan, Junwei Yang, Hengyu Shi, Tianyang Han, Junfeng Luo, and Yurui Qiu. Failure makes the agent stronger: Enhancing accuracy through structured reflection for reliable tool interactions. *arXiv preprint arXiv:2509.18847*, 2025.

Sri Vatsa Vuddanti, Aarav Shah, Satwik Kumar Chittiprolu, Tony Song, Sunishchal Dev, Kevin Zhu, and Maheep Chaudhary. PALADIN: Self-correcting language model agents to cure tool-failure cases. *arXiv preprint arXiv:2509.25238*, 2025.

Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

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. R-Judge: Benchmarking safety risk awareness for LLM agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 10471–10506, 2024.

Wenhui Zhang, Huiyu Xu, Zhibo Wang, Zeqing He, Ziqi Zhu, and Kui Ren. Can small language models reliably resist jailbreak attacks? A comprehensive evaluation. *arXiv preprint arXiv:2503.06519*, 2025.