V1.0.0

Specification-Induced Deadlock in Zero-Tolerance Agent Protocols: A State-Machine Analysis of the Capability–Invocation Gap

Rowan Brad Quni-Gudzinas

QNFO-ULA

ABSTRACT

Agent protocols written for large language model (LLM) agents increasingly mandate live tool execution and forbid simulated or placeholder actions outright. Such zero-tolerance semantics can create a specification-level deadlock: the agent recognizes that the required capability exists in its action space, but the protocol provides no representation for the case in which that capability is present yet its invocation is blocked in the current context, so the agent classifies every candidate action as a forbidden simulation and halts. This paper formalizes that failure mode. We model a protocol as a state machine over (requirement, capability, invocation) triples, define zero-tolerance compliance semantics over it, and prove that a halt state is reachable and absorbing whenever the protocol admits the configuration "capability present, invocation blocked." We quantify the exposure: a binary-state protocol governing k permission-gated tools is specification-complete on only a $(2/3)^k$ fraction of the true three-state space, giving $32/243 \approx 0.132$ at $k = 5$ and a $57.7\times$ state-space gap at $k = 10$. We further show, by a reduction in the style of Rice's theorem, that deciding whether an arbitrary protocol halts compliantly is undecidable, so well-definedness must be built into the protocol's syntactic class. Under stated projection assumptions (a 0.1 per-invocation probability that capability exists but invocation is blocked, and 5 gated invocations per task), the per-task deadlock probability is 0.40951. The result reframes a class of agent halts as specification defects repairable by adding an explicit "available but unused" protocol state, rather than as capability failures requiring better models.

PUBLISHER CLASSIFICATION High-quality scholarly research (QNFO publisher classification for open-access, versioned research with a public revision history and reader correction channel; this is not a claim of independent peer review.)

PEER-REVIEW STATUS Not independently peer-reviewed

CITE AS Quni-Gudzinas, R. B. (n.d.). Specification-Induced Deadlock in Zero-Tolerance Agent Protocols: A State-Machine Analysis of the Capability–Invocation Gap. QNFO Preprint v1.0.0.

1. Introduction

Teams deploying LLM agents in production write protocol instructions of two common kinds: "use the live tool when the task requires it" and "never simulate, fabricate, or placeholder tool output." The first instruction demands capability use; the second, taken alone, is sound engineering, because fabricated tool output is a well-known reliability and audit hazard. The failure this paper analyzes arises when a protocol combines both demands without a third representational state. An agent faces a task, recognizes that the required capability exists in its toolset, but the current operational context — a rate-limited session, a tenant-level policy switch, an exhausted budget tier — blocks invocation. The agent then attempts an action, evaluates it against the zero-tolerance rule, classifies it as a forbidden simulation, and halts. The capability exists; the invocation is blocked; the protocol has no state for "tools available but not used in this instance." This is a catch-22 manufactured by the specification, not by the model.

The cost of this failure mode falls on three groups. Operators of agent platforms lose completed tasks: an agent that halts compliantly delivers nothing, and the session must be restarted or escalated to a human. Protocol authors face a diagnostic puzzle, because the halt looks like a capability failure — the agent "could not act" — and the natural reme-

diation, adding capabilities or improving the model, does not touch it. Downstream consumers of agent work products inherit silent gaps: a task that halts mid-workflow leaves partial state that other systems may consume as if complete. What would change if the analysis in this paper holds: protocol authors could add a single explicit middle state ("capability available, invocation not required in this context") and eliminate the halt class by construction; operators could predict which tool-availability configurations deadlock before deployment; and diagnostic effort could shift from model improvement to specification repair.

The core of the paper is a formal treatment. We define a protocol state machine over triples (r, c, i) — requirement, capability, invocation — with zero-tolerance compliance semantics, and we prove two properties: the halt state is reachable from admissible initial configurations, and once reached it is absorbing. We then quantify how much of the true configuration space a binary protocol leaves unspecified, sketch an undecidability result for protocol-compliance halting, and project per-task deadlock rates under stated assumptions. The formalization borrows its state-transition-system substrate from established agent-protocol literature [1], [2], its deadlock vocabulary from the verification literature [6], [8], and its motivation from recent work on three-state tool semantics in LLM agent protocols [9].

The contributions are:

- A state machine over (requirement, capability, invocation) triples with zero-tolerance semantics, and a proof that the specification-induced halt state D is reachable and absorbing (Sections 3 and 4).
- An exact combinatorial result: a binary-state protocol over k gated tools covers $(2/3)^k$ of the three-state configuration space, computed as $32/243 \approx 0.132$ at $k = 5$ and a $59049/1024 \approx 57.7$ state-space ratio at $k = 10$ (Section 4).
- A reduction sketch showing that the protocol-compliance-halting problem is undecidable, so no universal static check can certify deadlock freedom for arbitrary protocols (Section 4).
- A projection of per-task deadlock probability, $1 - (1 - q)^n = 0.40951$ under the stated assumptions $q = 0.1$ and $n = 5$, with sensitivity to both parameters (Sections 4 and 5).
- A repair pathway: a three-state protocol extension that removes the halt class by construction, and the smallest experiment that would confirm or refute the reachability prediction (Section 7).

The reachability-and-absorption result for strict-verifiability state machines may also bear on other settings where a verifier must distinguish "action not taken" from "action faked," including multi-agent communication systems in which agents must signal capability without invoking it [7], and formal verification settings where an over-strict guard set masks legal idle states [6].

2. Background and Related Work

Agent communication protocols as state transition systems. The formal substrate for this paper is the state transition system (STS) model that underlies much agent-communication-protocol work. [1] defines a State Transition System as a tuple (S, s_0, L, T, F), where S is a finite set of states, $s_0 \in S$ is the initial state, L is a finite set of labels, $T \subseteq S \times L \times S$ is a set of transitions, and $F \subseteq S$ is a set of final states; the same definition, with transitions labeled by communication-act classes from the CommOnt ontology, appears in [2]. We adopt this tuple as the skeleton of our protocol machine in Section 3, adding a typed state space over (requirement, capability, invocation) triples. [1] and [2] also build protocol semantics on speech-act theory, in which every communication act is the sender's expression of an attitude toward a proposition, so a protocol's meaning lives in the acts it licenses, not merely in its transition graph. Zero-tolerance semantics, in our treatment, is exactly a restriction on which acts a state licenses — and the deadlock arises when the restriction licenses none. [2] additionally argues for customizing standard communication protocols and managing them with standard Semantic Web tools, which is the same repair philosophy we apply: change the specification's state space, not the agent.

Protocol comparison and semantic relationships. [1] presents a mechanism for discovering semantic relationships among agent communication protocols by comparing the sets of fluents associated with protocol branches rather than comparing protocol structures, and reports that this favors a more flexible comparison of protocols. This matters for our repair proposal: a three-state extension of a binary protocol changes the fluent set of each branch, and a fluent-based comparison is the natural tool for checking that the extension preserves the original protocol's intended behavior while adding the missing middle state.

Deadlock in parameterized verification. The verification literature supplies both the vocabulary and a caution. [6] studies system-wide deadlock (s-deadlock) in symmetric cache-coherence protocols — the error state in which the entire protocol is blocked and no agent can make any transition — and notes that the key verification challenge is that an s-deadlock violation is a state in which all guards are false, i.e., $\bigwedge_{\tau} \neg(\tau.g)$, which simple over-abstractions easily mask. Our halt state D has exactly this all-guards-false character: under zero-tolerance semantics, every action guard evaluates to false in D, which is why naive testing of "does the agent act correctly?" masks the state rather than revealing it. [6] also demonstrates the practical verification route: deriving invariants from message-flow specifications that collectively imply deadlock freedom, verified on the German and Flash protocols with a proof finishing in $217\,\mathrm{s}$ over $7\,\mathrm{M}$ states explored. Section 7 proposes the analogous artifact for agent protocols: flow-level invariants over the (requirement, capability, invocation) machine whose conjunction implies halt-freedom.

Logical characterizations of deadlock-freedom. [8] establishes a correspondence between sequent-calculus derivations in the system PiL and computation trees of the recursion-free π-calculus, yielding a logical characterization of deadlock-freedom and a completeness result for choreographic programming with respect to mobile processes with cyclic dependencies. The relevance is structural: a choreography is a global specification of who does what when, and [8]'s result shows that deadlock-freedom can be read off the specification logically rather than discovered by execution. Our Section 4 proof is a miniature of this philosophy — the halt state's reachability is derived from the protocol's transition rules themselves — and our repair (adding an explicit state) is a choreographic fix: the global script gains a move for "capability present, not exercised."

Multi-agent communication. [7] surveys multi-agent deep reinforcement learning with communication, organizing the literature along nine dimensions and concluding that communication can be extended to non-cooperative settings, heterogeneous players, and larger agent populations. Two of its framing points bear directly on our model. First, [7] identifies partial observability as an essential assumption in multi-agent reinforcement learning — an agent's policy is computed from local observations, not the global state. Our deadlock is a partial-observability pathology inside a single agent's protocol: the agent observes "capability exists" but not "invocation is licensed in this context," and the protocol gives it no action that is compliant under that observation. Second, [7] treats communication as the mechanism that mitigates partial observability; in our setting the protocol's missing middle state is precisely a missing communicable state, and adding it restores the agent's ability to signal its situation.

Protocol-level enforcement for AI agents. Recent work on agent-native protocols shows both the enforcement machinery and the evaluation dimensions against which our proposal would be judged. [4] presents Mandato, which enforces digitally signed mandates on AI agent actions at the protocol level, with cryptographically chained audit trails, motivated by the observation that authorization logic otherwise lives unsigned in application code and the resulting logs lack evidentiary value. Mandato is the natural enforcement layer for a zero-tolerance rule: a signed mandate can carry the "invocation licensed in this context" bit that our missing middle state needs, making the license machine-checkable rather than inferred by the model. [5] presents ANX, an agent-native protocol with a decoupled architecture, and contributes a four-dimensional evaluation framework (Protocol & Tooling, Discovery & Retrieval, Security, Collaboration) with explicit sub-dimensions, plus a comparative taxonomy of browser-centric, tool-centric, security-specialized, and multi-agent-collaboration families. [5] also reports concrete token costs that quantify the price of protocol verbosity, the currency in which any state-space extension of ours must be paid: in its comparison, average task-

inclusive token consumption is $8.7\,\mathrm{k}$ for GUI-based agents, $6.85\,\mathrm{k}$ for MCP-based skills, and $3.35\,\mathrm{k}$ for ANX, against a baseline of $13.2\,\mathrm{k}$ tokens consumed by system prompts and core skills before task-specific execution. Our three-state extension adds one state to the protocol's semantics; Section 7 uses [5]'s framework to argue that the token overhead of one additional semantic label is small relative to these budgets, and [5]'s identification of deterministic long-horizon SOP execution as an open challenge locates exactly where our deadlock bites — long deterministic SOPs are where a mid-flow halt is most costly.

Agents as live actors. [3] reviews the use of virtual agents in the practice of live coding, a setting in which an agent must act in real time under an audience, and notes that the combination of AI and live coding had been little explored. Live settings make the halt failure maximally visible: an agent that deadlocks mid-performance produces silence, and the audience observes the specification failure directly. We use [3] as evidence that agents operating under real-time, no-placeholder expectations are an established practice area, which is the population our failure mode affects.

Prior formalization of the capability–availability mismatch. The closest prior work is [9], which formalizes the gap between a tool's capability and its per-context availability: tool state is modeled as a pair of capability and per-context permission, real deployments exhibit a third state beyond "available" and "absent," and the work derives that a binary-state protocol governing k permission-gated tools is specification-complete on a $(2/3)^k$ fraction of the true three-state space, with computed values $f(5) = 32/243 \approx 0.132$ and a $k = 10$ ratio of $3^{10}/2^{10} = 59049/1024 \approx 57.7$, of which $59049 - 1024 = 58025$ configurations contain at least one disabled tool. [9] also proves that the protocol compliance-halting problem — whether an agent loop under an arbitrary protocol halts with a compliant action in a given context — is undecidable by a Rice's-theorem reduction, concluding that well-definedness must be built into the protocol's syntactic class rather than verified after the fact. Our paper differs in the failure configuration: [9] analyzes capability present, permission denied; we analyze capability present, invocation blocked by context with no protocol state to represent it, and we prove the resulting halt is reachable and absorbing under zero-tolerance semantics. The two are complementary cells of the same three-state table, and our combinatorial arithmetic in Section 4 reproduces [9]'s coverage figures because both follow from the same state counting.

Verification discipline and epistemic failure. [10] argues that the transition from static LLMs to autonomous agents introduces epistemic failure modes driven by an "Orchestration Penalty" — a factorial increase in verification complexity across multi-step reasoning loops — and that safety is an engineering discipline of managing latency, entropy, and feedback. A specification-induced deadlock is a limiting case of that penalty: the verification loop cannot terminate because the specification admits no terminating compliant action, so the loop's cost is paid without any epistemic return. This framing also motivates the discipline we adopt throughout: every quantitative claim below is either an exact combinatorial consequence of the stated semantics or a projection whose assumptions are listed explicitly.

3. Methods

3.1 Modeling approach

This paper is a formal analysis: we define a state machine, prove two properties of it, and compute exact combinatorial quantities and labeled projections. No experiment, measurement, or simulation on hardware was performed; the empirical study that would test the reachability prediction is specified as a proposal in Section 7.

3.2 The protocol state machine

We adopt the STS tuple of [1], [2]: a machine is (S, s_0, L, T, F) with states S, initial state s_0, labels L, transitions $T \subseteq S \times L \times S$, and final states F. Our states are triples over Boolean components:

$$s = (r, c, i) \in \{0,1\} \times \{0,1\} \times \{0,1\},$$

where:

- r : requirement — the task at hand requires the capability ($r = 1$) or does not ($r = 0$);
- c : capability — the capability exists in the agent's action space ($c = 1$) or not ($c = 0$);
- i : invocation — the capability is legitimately invoked in the current operational context ($i = 1$) or not ($i = 0$).

The invocation component i aggregates everything that blocks a live call in a specific context: rate limits, tenant policy, exhausted budget, routing tier. It is the per-context permission-like bit of [9], evaluated at the moment of action.

The label set is $L = \{\mathsf{invoke}, \mathsf{refrain}, \mathsf{halt}\}$. The transition function is fixed by the protocol's semantics, defined next.

3.3 Zero-tolerance compliance semantics

A zero-tolerance protocol mandates: (Z1) if the task requires the capability, the agent must act on it live; (Z2) the agent must never produce simulated, fabricated, or placeholder tool output. The compliance predicate $\mathrm{Compliant}(s, \ell)$ over state s and label ℓ is:

$$\mathrm{Compliant}((r, c, i), \mathsf{invoke}) \iff c = 1 \wedge i = 1,$$

$$\mathrm{Compliant}((r, c, i), \mathsf{refrain}) \iff r = 0 \vee c = 0,$$

$$\mathrm{Compliant}((r, c, i), \mathsf{halt}) \iff \text{false}.$$

The critical clause is the second. Under zero-tolerance semantics as written, refraining is compliant only when the capability is absent or not required. There is no state in which the capability is present, the task requires it, and refraining is compliant — because the protocol's authors, reasoning from (Z1), treat "required and available but not used" as a violation of the mandate to act, and reasoning from (Z2), treat any attempted action that does not produce live output as a forbidden simulation. The protocol thus lacks the middle state "tools available but not used in this instance."

The agent's decision procedure, as protocols of this kind instruct it, is: attempt the action; if the attempt would not yield live output, classify it as a forbidden simulation and halt. Formally, from any state with $r = 1$, $c = 1$, $i = 0$:

- invoke is non-compliant because $i = 0$ (the call cannot complete live; the agent's own evaluation classes it as simulation, violating Z2);
- $\mathsf{refrain}$ is non-compliant because $r = 1$ and $c = 1$ (violating Z1);
- halt is non-compliant by definition.

This is the deadlock configuration. We name it:

$$D \equiv (r, c, i) = (1, 1, 0).$$

3.4 Proof obligations

We prove: (P1) reachability — D is reachable from an admissible initial configuration under the protocol's transition rules; (P2) absorption — once in D, no compliant transition leaves it, so the machine halts. We then compute the fraction of the configuration space in which the deadlock condition can arise, sketch the undecidability of protocol-compliance halting, and project per-task deadlock rates under stated assumptions. All arithmetic is shown in Section 4.

3.5 Assumptions

We assume: (A1) the three-state semantics of Section 3.2 is the correct state model, i.e., platforms do not advertise permissions for nonexistent tools, so the $(c = 0, i = 1)$ cell is excluded — the same idealization [9] states, and [9] notes that if it fails, the coverage fractions worsen in the direction that strengthens the specification-gap claim; (A2) the

agent's self-evaluation of "would this call complete live?" is accurate, so the agent correctly identifies $i = 0$ rather than attempting and failing; (A3) for the projections, independence of the per-invocation blocking event across invocations within a task.

We adopt the convention that projection parameters are stated assumptions of this paper rather than rates attributed to any cited work; a sensitivity case using an alternative assumed input is reported in Section 4.6.

4. Analysis

4.1 Reachability of the halt state (P1)

Claim. $D = (1, 1, 0)$ is reachable from an admissible initial state.

Derivation. Take the admissible initial state $s_0 = (1, 1, 1)$: the task requires the capability, the capability exists, and invocation is licensed. This is the ordinary operating state of any agent under a zero-tolerance protocol, so it is admissible by construction. The context can change i from 1 to 0 without changing r or c: a rate limit is reached, a tenant policy flips, a budget tier exhausts. These are context events, not agent actions; they modify the environment component i directly. The transition

$$(1, 1, 1) \xrightarrow{\text{context}} (1, 1, 0)$$

is therefore in the machine's transition relation whenever the protocol operates in any environment where context can gate invocation — which is precisely the deployment class the protocol's own zero-tolerance rule targets (environments where fabrication is a risk are environments where gating exists). Hence D is reachable in one context transition from the protocol's normal operating state. □

The reachability argument shows the deadlock is not an exotic corner: it sits one context event away from the protocol's designed operating point.

4.2 Absorption of the halt state (P2)

Claim. In D, no compliant action exists, and the machine halts.

Derivation. Evaluate the compliance predicate of Section 3.3 at $s = (1, 1, 0)$:

- $\text{Compliant}(D, \mathsf{invoke}) \iff c = 1 \wedge i = 1 \iff 1 \wedge 0 = \text{false}$.
- $\text{Compliant}(D, \mathsf{refrain}) \iff r = 0 \vee c = 0 \iff 0 \vee 0 = \text{false}$.
- $\text{Compliant}(D, \mathsf{halt}) = \text{false}$ by definition.

The compliant-label set at D is empty:

$$\{\ell \in L : \text{Compliant}(D, \ell)\} = \varnothing.$$

An agent that must act compliantly has no compliant action; the only terminating behavior is the forced halt. Moreover, the halt is absorbing in the compliance sense: no transition out of D changes r, c, or i under the agent's own action set, because every agent action is non-compliant and the protocol forbids non-compliant action. The state is a sink. □

This is the same all-guards-false structure that [6] identifies as the hard case in s-deadlock verification: the violation state is one where every action guard is false, $\bigwedge_\tau \neg(\tau.g)$, and [6] notes that over-abstractions mask exactly such states. A protocol test suite that only exercises states where some guard is true will never visit D.

4.3 State-space coverage of a binary protocol

Following the three-state semantics (with assumption A1 excluding the $(c = 0, i = 1)$ cell), each gated tool occupies one of three states: (capability absent), (capability present, invocation licensed), (capability present, invocation blocked). A binary protocol distinguishes only two: present or absent.

Per-tool coverage. The fraction of the true per-tool state space a binary protocol represents is:

$$f(1) = \frac{2}{3}.$$

Arithmetic step. Two representable states out of three total states: $2/3 = 0.666\ldots$

Per-task coverage over k gated tools. Assuming independence across tools (the same independence assumption [9] states for its derivation), a binary protocol is specification-complete on:

$$f(k) = \left(\frac{2}{3}\right)^k$$

of the true configuration space.

Arithmetic at $k = 5$.

$$f(5) = \left(\frac{2}{3}\right)^5 = \frac{2^5}{3^5} = \frac{32}{243}.$$

Computing the decimal: $32 \div 243 = 0.13168\ldots$, so

$$f(5) = \frac{32}{243} \approx 0.132.$$

Interpretation: at five gated tools, a binary protocol fully specifies about 13.2% of the true configuration space; the remaining $1 - 32/243 = 211/243 \approx 0.868$ of configurations contain at least one tool in the unspecified third state.

Arithmetic at $k = 10$.

$$3^{10} = 59049, \qquad 2^{10} = 1024.$$

The state-space ratio is:

$$\frac{3^{10}}{2^{10}} = \frac{59049}{1024} = 57.66\ldots \approx 57.7.$$

The number of configurations containing at least one blocked-invocation tool is:

$$59049 - 1024 = 58025.$$

$$\text{Unspecified fraction at } k = 10: \quad 1 - \frac{1024}{59049} = \frac{58025}{59049} \approx 0.98266.$$

Interpretation: at ten gated tools, a binary protocol leaves about 98.3% of the true configuration space without a representable state, and 58025 of the 59049 true configurations contain at least one tool whose capability exists while its invocation is blocked — exactly the configurations in which the deadlock condition of Section 3.3 can arise.

4.4 Undecidability of protocol-compliance halting

Claim. The problem PCH — given an arbitrary protocol P (as a state machine in the sense of Section 3.2) and an agent loop running under it, decide whether the loop halts with a compliant action in a given context — is undecidable.

Reduction sketch. The argument is a Rice's-theorem reduction in the style used by [9] for its protocol compliance-halting problem. Any candidate protocol P can encode an arbitrary Turing-machine computation M in its transition relation: the machine's tape contents and head position are encoded in the protocol's state components, and each machine step is one protocol transition. Construct P_M so that the loop reaches a halting-with-compliant-action configuration if and only if M halts. If a decider for PCH existed, it would decide the halting problem for M, contradicting undecidability of the halting problem. Hence PCH is undecidable. □

Two consequences follow. First, no universal static checker can certify deadlock freedom for arbitrary protocols; a guarantee must come from restricting the protocol's syntactic class so that the guarantee holds by construction, which is the design stance [9] draws from the same reduction ("well-definedness must be built into the protocol's syntactic class rather than verified after the fact"). Second, the repair of Section 7 is deliberately of this kind: adding an explicit middle state changes the protocol's state space so that the empty-compliant-set configuration is excluded syntactically, not verified empirically.

4.5 Per-task deadlock probability (projection)

The following is a projection, not a measurement. We assume (A3) independence of blocking across invocations within a task, and we state the two projection inputs as assumptions of this paper:

- $q = 0.1$: the per-invocation probability that the capability exists but invocation is blocked in the current context;
- $n = 5$: the number of permission-gated invocations per task.

Under these assumptions, the probability that a task contains at least one blocked invocation — the event that triggers the deadlock configuration of Section 3.3 — is:

$$p_{\text{task}} = 1 - (1 - q)^n.$$

Arithmetic. First compute $(1 - q)^n = 0.9^5$:

$$0.9^2 = 0.81, \qquad 0.9^4 = 0.81 \times 0.81 = 0.6561, \qquad 0.9^5 = 0.6561 \times 0.9 = 0.59049.$$

Then:

$$p_{\text{task}} = 1 - 0.59049 = 0.40951.$$

So under the stated assumptions, roughly 41% of tasks encounter at least one blocked invocation. This is an upper bound on the deadlock rate itself: a deadlock occurs only if the agent's protocol lacks the middle state (the zero-tolerance condition of Section 3.3) and the blocked invocation is one the task actually requires ($r = 1$, $c = 1$). We treat $p_{\text{task}} = 0.40951$ as the exposure rate and note that the realized deadlock rate is p_{task} multiplied by the fraction of blocked invocations that are required-and-capable, a factor no source in this analysis quantifies.

4.6 Sensitivity of the projection

Sensitivity to q at fixed $n = 5$. The exposure rate is $1 - (1 - q)^5$:

- $q = 0.01$: $1 - 0.99^5$. $0.99^2 = 0.9801$; $0.99^4 = 0.9801^2 = 0.96059601$; $0.99^5 = 0.96059601 \times 0.99 = 0.9509900499$; exposure $= 0.0490099501 \approx 0.049$.
- $q = 0.05$: $1 - 0.95^5$. $0.95^2 = 0.9025$; $0.95^4 = 0.81450625$; $0.95^5 = 0.7737809375$; exposure $= 0.2262190625 \approx 0.226$.
- $q = 0.2$: $1 - 0.8^5$. $0.8^2 = 0.64$; $0.8^4 = 0.4096$; $0.8^5 = 0.32768$; exposure $= 0.67232 \approx 0.672$.

Sensitivity to n at fixed $q = 0.1$. At $n = 10$: $0.9^{10} = (0.9^5)^2 = 0.59049^2 = 0.3486784401$, so exposure $= 1 - 0.3486784401 = 0.6513215599 \approx 0.651$. Doubling the number of gated invocations raises the exposure from 0.40951 to about 0.651.

Alternative convention for q. This paper adopts the convention that q is a stated assumption rather than a rate attributed to any cited work (Section 3.5). As a sensitivity case, we take an alternative assumed input, $q = 0.27$ per gated invocation. Under that assumption at $n = 5$:

$$1 - 0.73^5: \quad 0.73^2 = 0.5329, \quad 0.73^4 = 0.5329^2 = 0.28398241, \quad 0.73^5 = 0.28398241 \times 0.73 = 0.2073071593,$$

so exposure $= 1 - 0.2073071593 = 0.7926928407 \approx 0.793$. The qualitative conclusion — exposure rises steeply with the number of gated invocations and is material already at five — holds under both conventions; the quantitative spread between 0.410 and 0.793 is the uncertainty the assumption choice carries.

5. Results

The paper derives four results. R1–R2 are exact consequences of the stated semantics; R3 is a projection under labeled assumptions.

R1 (Reachability and absorption, Sections 4.1–4.2). Under zero-tolerance compliance semantics, the configuration $D = (1, 1, 0)$ — capability present, task requires it, invocation blocked — is reachable in one context transition from the protocol's normal operating state $(1, 1, 1)$, and at D the compliant-label set is empty, $\{\ell \in L : \text{Compliant}(D, \ell)\} = \varnothing$, so the machine halts and the halt is absorbing. The deadlock is a specification property, not a capability property.

R2 (State-space coverage, Section 4.3). A binary-state protocol governing k permission-gated tools is specification-complete on a $(2/3)^k$ fraction of the true three-state configuration space. Computed values: $f(5) = 32/243 \approx 0.132$, with $211/243 \approx 0.868$ of configurations containing at least one tool in the unspecified third state; at $k = 10$ the state-space ratio is $3^{10}/2^{10} = 59049/1024 \approx 57.7$, with 58025 of 59049 configurations containing at least one blocked-invocation tool. These are exact combinatorial consequences of the three-state semantics, not measurements.

R3 (Undecidability, Section 4.4). The protocol-compliance-halting problem is undecidable by a Rice's-theorem reduction, so no universal static check certifies deadlock freedom for arbitrary protocols; deadlock freedom must be built into the protocol's syntactic class.

R4 (Exposure projection, Sections 4.5–4.6). Under the stated assumptions $q = 0.1$ per invocation and $n = 5$ gated invocations per task, the per-task exposure to at least one blocked invocation is $1 - 0.9^5 = 0.40951$. Sensitivity: exposure is 0.049 at $q = 0.01$, 0.226 at $q = 0.05$, 0.672 at $q = 0.2$, and 0.651 at $n = 10$, $q = 0.1$; under the alternative assumed input $q = 0.27$, exposure is 0.793. R4 bounds exposure, not realized deadlock rate; the realized rate additionally requires the blocked invocation to be required and capable.

6. Discussion

What the results do and do not claim. R1 and R2 are exact given the modeling choices; R3 is a formal result whose scope is bounded by its reduction sketch; R4 is a projection resting on assumed rates. We discuss each in turn, then state limitations and falsification conditions.

Scope of R1. The reachability proof assumes the context can flip i from 1 to 0 while r and c stay fixed. This holds in any deployment with rate limits, tenant policy, or budget tiers — which is the deployment class zero-tolerance rules target, since environments where fabrication is a risk are environments where gating exists. If a platform instead removes a tool from the toolset whenever it blocks invocation (c flips to 0 with i), the agent faces the $(r{=}1, c{=}0)$ state,

where refrain is compliant and no deadlock occurs. The deadlock therefore depends on the platform's gating design preserving the capability entry while blocking the call — the same third-state observation [9] makes about real deployments.

Scope of R2. The coverage arithmetic assumes independence across tools and the three-state idealization (A1). If platforms advertise permissions for nonexistent tools, a fourth state appears and coverage worsens to $(2/4)^k$; at $k = 5$ that is $(1/2)^5 = 1/32 = 0.03125$, so the direction of the error strengthens the specification-gap claim while changing the exact numbers.

Scope of R3. The undecidability reduction is a sketch, not a fully formal construction: it asserts that protocol transitions can encode Turing-machine steps without writing the encoding in detail. The claim's force is the standard one — arbitrary protocol state machines are programs, and halting for programs is undecidable — but a complete proof would fix the encoding and verify both directions of the equivalence.

Falsification. The central claim — that zero-tolerance protocols without a middle state deadlock exactly when capability exists but invocation is blocked — is falsified by a single counterexample configuration: an agent operating under such a protocol that, in state $(1, 1, 0)$, produces a compliant, useful, non-halting behavior (for example, a protocol-sanctioned "skip and continue" action). Equivalently, the R4 projection is falsified if a controlled study finds halt rates near zero in blocked-invocation configurations while halt rates in capability-absent configurations are also near zero — that pattern would show agents are not tracking the compliance predicate Section 3.3 defines. The R2 arithmetic cannot be falsified except by rejecting the three-state semantics itself, which is an empirical question about real platforms.

Limitations list.

- *Three-state semantics is an idealization* (A1). The coverage numbers change if a fourth state exists; the direction of the error favors the claim.
- *Accurate self-evaluation* (A2). The agent must correctly identify $i = 0$ rather than attempt and fail. If the agent misjudges and attempts the call, the failure mode becomes a live error, not a compliant halt — a different (and more visible) failure.
- *Independence* (A3). The projection assumes blocking events are independent across invocations within a task; correlated blocking (one tenant switch gating many tools) raises the joint probability above $1 - (1 - q)^n$, so 0.40951 is a lower bound under positive correlation.
- *Projection inputs are assumptions.* $q = 0.1$ and $n = 5$ are stated assumptions, not measured rates; the alternative assumed input $q = 0.27$ yields 0.793, so the exposure figure carries a factor-of-two uncertainty from the assumption alone.
- *Exposure is not deadlock rate.* The realized deadlock rate is exposure times the fraction of blocked invocations that are required and capable; no input here quantifies that fraction.
- *Reduction sketch.* R3's proof is not fully formalized (see above).

Open questions. What fraction of production agent protocols are zero-tolerance in the Section 3.3 sense? Do agent frameworks' retry and escalation layers already provide a de facto middle state that masks the deadlock in practice? How does the deadlock interact with human-in-the-loop confirmation flows of the kind [5] lists under its Security dimension?

7. Practical Implications and Engineering Pathways

(a) Who would use the result and what changes. The users are protocol authors for agent platforms and operators deploying agents under strict tool-use policies. If the result holds, a protocol author adds one explicit semantic state — "capability available, invocation not licensed in this context" — with a compliant refrain-class action attached, and the

halt class is removed by construction rather than by model improvement. An operator gains a deployment-time check: enumerate the gated tools, compute the exposure $1-(1-q)^n$ under measured q and n, and decide whether the protocol needs the middle state before rollout. A diagnostic team gains a discriminator: a halt in a $(1,1,0)$ configuration is a specification defect, not a capability gap, so the fix is a protocol patch, not a model upgrade.

(b) Readiness. This work is analytical and not peer reviewed: under the NASA Technology Readiness Level definitions it sits at TRL 1–2 (basic principles observed and reported; concept formulated). No claim of implementation, prototype, or manufacturing readiness is made, and analytical work of this kind does not exceed TRL 3.

(c) Smallest next test. The smallest experiment that would confirm or refute the reachability prediction: run an LLM agent under a written zero-tolerance protocol (mandate live use, forbid simulation, no middle state) across a factorial set of tool-availability configurations spanning (r,c,i), with i manipulated by an injected rate limit or policy switch, and measure halt rates per configuration. The prediction is that halts concentrate in the $(1,1,0)$ cell and are absent in $(1,0,\cdot)$ and $(0,\cdot,\cdot)$ cells. The refuting outcome is a halt rate in $(1,1,0)$ statistically indistinguishable from the other cells, or a high rate of compliant non-halt behavior in $(1,1,0)$. A second arm runs the same agent under the three-state extended protocol; the prediction is that halts in $(1,1,0)$ drop to the level of the other cells.

(d) Binding constraints. The first constraint is specification, not compute: the middle state must be expressible in the protocol's action vocabulary and the agent's system prompt, and the agent must reliably classify i at action time — assumption A2, which is an empirical property of current models, not a given. The second is enforcement plumbing: making the "invocation licensed in this context" bit machine-checkable requires a protocol-level authorization layer of the kind [4] describes, in which signed mandates carry per-context permissions with cryptographically chained audit trails, rather than leaving the license bit to be inferred by the model. The third is token budget: [5] reports average task-inclusive token consumption of $8.7\,\mathrm{k}$ for GUI-based agents, $6.85\,\mathrm{k}$ for MCP-based skills, and $3.35\,\mathrm{k}$ for ANX, against a $13.2\,\mathrm{k}$ baseline consumed by system prompts and core skills before task-specific execution; one additional semantic label is small against these budgets, but the full decision procedure for classifying i must fit the same budget. Data constraints bind the proposed study: per-cell halt rates require enough trials per configuration to distinguish rates near zero from rates near 0.4, which is a sample-size question left open here.

(e) Order-of-magnitude bounds (projection). Under the assumptions of Section 4.5 ($q=0.1$, $n=5$), a platform running 10^4 agent tasks per day would see on the order of $0.40951\times10^4\approx4.1\times10^3$ tasks per day exposed to at least one blocked invocation, of which the deadlock subset is smaller by the required-and-capable fraction. This is a projection from the stated assumptions only, not a measurement of any platform.

(f) Risks and misuse paths. The three-state extension, if implemented carelessly, could become a loophole: an agent that classifies every inconvenient invocation as "blocked" and refrains compliantly, laundering non-performance as a licensed skip. The mitigation is the enforcement layer of (d): the license bit must be platform-asserted and audited, not agent-asserted. A second risk is over-broad application of the diagnosis — attributing genuine capability failures to specification defects — which the per-configuration discriminator of (c) is designed to prevent.

8. Conclusion

Zero-tolerance agent protocols that mandate live tool use while forbidding simulated actions, without a third representational state, manufacture their own deadlock: the configuration $D=(1,1,0)$ is reachable in one context transition from the protocol's normal operating state, and at D no compliant action exists, so the halt is absorbing. A binary protocol over k gated tools specifies only $(2/3)^k$ of the true configuration space — $32/243\approx0.132$ at $k=5$, a $57.7\times$ gap at $k=10$ — and the protocol-compliance-halting problem is undecidable, so deadlock freedom must be built in syntactically. Under stated assumptions, per-task exposure is 0.40951, with sensitivity from 0.049 to 0.793 across the assumption range. The practical upshot is a reclassification: a class of agent halts are specification defects, repairable by

adding an explicit "available but unused" state with a platform-asserted license bit, and diagnosable by a per-configuration test that distinguishes specification halts from capability failures. The reachability-and-absorption structure may also apply wherever a strict-verifiability protocol must distinguish "action not taken" from "action faked," including multi-agent settings in which agents signal capability without invoking it and verification settings where an over-strict guard set masks legal idle states.

Provenance and AI Involvement

- This paper was drafted, reconciled, reviewed, revised and checked by the QNFO automated research pipeline (qnfo-research-exec 0.45.3). Language models were tools in that process; they are not authors. The named author is accountable for the content.
- Models: drafting by openai/gpt-oss-120b, zai-org/glm-5.3-flash; reconciliation, completion, review, revision and verification by zai-org/glm-5.3-flash, openai/gpt-oss-120b, moonshotai/kimi-k2.6, with the QNFO model router as a fallback.
- Sources: every reference is drawn from a bibliography retrieved from arXiv and the QNFO corpus before drafting; every cited work's full text was retrieved (arXiv HTML, or the QNFO paper's body) and the writers were given excerpts of its own sections; statements about a cited work are restricted to its retrieved title, abstract and those excerpts.
- Checks passed before publication: required sections present and finished, cross-references resolve, reference list rendered from the retrieved bibliography, no assistant residue or placeholders, adversarial model review with its blocking findings resolved, and a citation existence check at publication.
- Independent recomputation: the execution sandbox was unavailable, so a second language model recomputed 13 quantitative claims step by step in text from the paper's stated inputs (no code was executed); 0 differed from the stated value at the stated precision.
- Not checked: whether each cited source supports the sentence that cites it beyond the retrieved full-text excerpts; derivation steps other than the recomputed claims. This paper has not been peer reviewed, and this version was published by the pipeline without a prior manual review by the author.

References

[1] A mechanism for discovering semantic relationships among agent communication protocols. arXiv:2401.16216v1. https://arxiv.org/abs/2401.16216v1 [2] Semantic Web Technology for Agent Communication Protocols. arXiv:2401.11841v1. https://arxiv.org/abs/2401.11841v1 [3] Virtual Agents in Live Coding: A Short Review. arXiv:2106.14835v1. https://arxiv.org/abs/2106.14835v1 [4] Mandato: Protocol-Level Enforcement of Digitally Signed Mandates on AI Agent Actions with Cryptographically Chained Audit Trails. arXiv:2608.14074v1. https://arxiv.org/abs/2608.14074v1 [5] ANX: Protocol-First Design for AI Agent Interaction with a Supporting 3EX Decoupled Architecture. arXiv:2604.04820v1. https://arxiv.org/abs/2604.04820v1 [6] Using Flow Specifications of Parameterized Cache Coherence Protocols for Verifying Deadlock Freedom. arXiv:1407.7468v1. https://arxiv.org/abs/1407.7468v1 [7] A Survey of Multi-Agent Deep Reinforcement Learning with Communication. arXiv:2203.08975v2. https://arxiv.org/abs/2203.08975v2 [8] Formulas as Processes, Deadlock-Freedom as Choreographies (Extended Version). arXiv:2501.08928v2. https://arxiv.org/abs/2501.08928v2 [9] https://papers.qnfo.org/papers/the-capability-availability-mismatch-three-state-tool-semantics-and-forced-halt-. QNFO: The Capability–Availability Mismatch: Three-State Tool Semantics and Forced-Halt Conditions in LLM Agent Protocols. [10] https://papers.qnfo.org/papers/agentic-collapse. QNFO: Agentic Collapse.